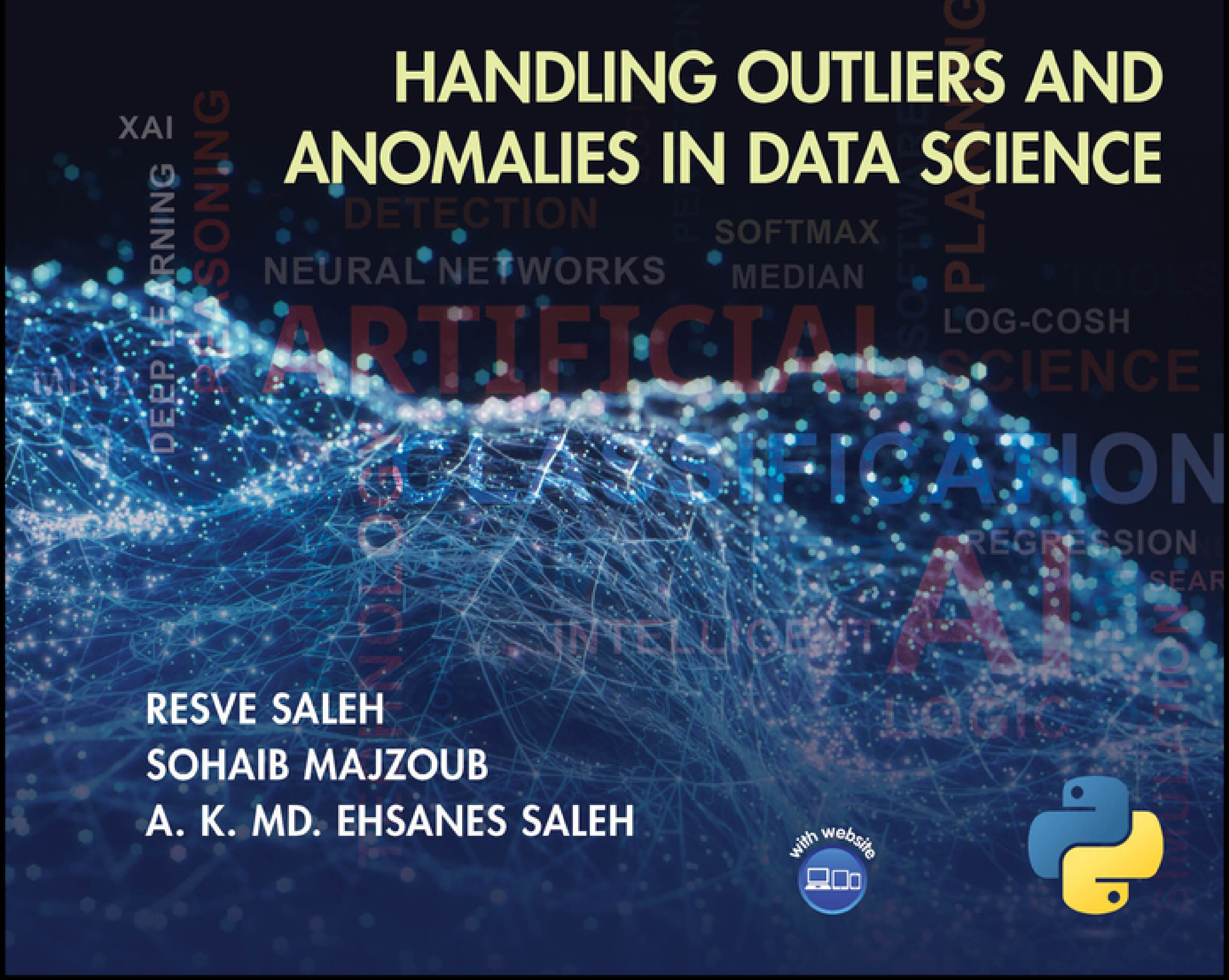
FUNDAMENTALS OF ROBUST MACHINE LEARNING

HANDLING OUTLIERS AND ANOMALIES IN DATA SCIENCE



RESVE SALEH
SOHAIB MAJZOUB
A. K. MD. EHSANES SALEH

with website

Fundamentals of Robust Machine Learning

Fundamentals of Robust Machine Learning

Handling Outliers and Anomalies in Data Science

Resve Saleh
University of British Columbia, Vancouver, Canada

Sohaib Majzoub
University of Sharjah, Sharjah, United Arab Emirates

A. K. Md. Ehsanes Saleh
Carleton University, Ottawa, Canada

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data Applied for:

Hardback ISBN: 9781394294374

Cover Design: Wiley
Cover Image: © Yuichiro Chino/Getty Images, © master_art/Shutterstock

Set in 9.5/12.5pt STIXTwoText by Straive, Chennai, India

We dedicate this book to

Shahidara Saleh
Lynn Hilchie Saleh
Bayan Majzoub

Contents

Preface

Outliers are part of almost every real-world dataset. They can occur naturally as part of the characteristics of the data being collected. They can also be due to statistical noise in the environment that might be unavoidable. More commonly, they are associated with measurement error or instrumentation error. Another source is human error, such as typographical errors or misinterpreting the measurements of a device. If there are extreme outliers, they are often referred to as **anomalies**. Sometimes, the true data is differentiated from outliers by calling them **inliers**. While outliers may represent a small portion of the dataset, their impact can be quite significant.

The machine learning and data science techniques in use today largely ignore outliers and their potentially harmful effects. For many, outliers are somewhat of a nuisance during model building and prediction. They are hard to detect in both regression and classification problems. Therefore, it is easier to ignore them and hope for the best. Alternatively, various ad hoc techniques are used to remove them from the dataset even at the risk of inadvertently removing valuable inlier data in the process. But we have reached a point in data science where these approaches are no longer viable. In fact, new methods have emerged recently with great potential to properly address outliers and they should be investigated thoroughly.

The cost of ignoring this under-reported and often overlooked aspect of data science can be significant. In particular, outliers and anomalies in datasets may lead to inaccurate models that result in making bad business decisions, producing questionable explanations of cause-and-effect, arriving at the wrong conclusions, or making incorrect medical diagnoses, just to name a few. A prediction is only as good as the model on which it is based, and if the model is faulty, so goes the prediction. Even one outlier can render a model unusable if it happens to be in the wrong location. Machine learning practitioners have not yet fully embraced a class of robust techniques that would provide more reliable models and more accurate predictions than is possible with present-day methods. Robust methods are better-suited to data science, especially when outliers are present. The overall goal of this book is to provide the rationale and techniques for robust machine learning and then build on that material toward robust data science.

This book is a comprehensive study of outliers in datasets and how to deal with them in machine learning. It evaluates the robustness of existing methods such as linear regression using least squares and Huber's method, and binary classification using the cross-entropy loss for logistic regression and neural networks, as well as other popular methods including *k*-nearest neighbors, support vector machines and random forest. It provides a number of new approaches using the log-cosh loss which is very important in robust machine learning. Furthermore, techniques that surgically remove outliers from datasets for both regression and classification problems are presented. The book is about the pursuit of methods and procedures that recognize the adverse effects that outliers can have on the models built by machine learning tools. It is intended to move the field toward robust data science where the proper tools and methodologies are used to handle outliers. It introduces a number of new ideas and approaches to the theory and practice of robust machine learning and encourages readers to pursue further investigation in this field.

This book offers an interdisciplinary perspective on robust machine learning. The prerequisites are some familiarity with probability and statistics, as well as the basics of machine learning and data science. All three areas are covered in equal measure. For those who are new to the field and are looking to understand key concepts, we do provide the necessary introductory and tutorial material in each subject area at the beginning of each chapter. Readers with an undergraduate-level knowledge of the subject matter will benefit greatly from this book.

You may have heard the phrase “regression to the mean.” In this book, we discuss “regression to the median.” The methods currently in use target the **mean** of the data to estimate the model parameters. However, the **median** is a better target because it is more stable in the presence of outliers. It is important to recognize that data science should be conducted using methods that are reliable and stable which is what the median-based approach can offer. There are good reasons why we frequently hear phrases like “the median house price” or “the median household income.” It holds the key to building outlier-tolerant models. Furthermore, robust methods offer stability and accuracy with or without outliers in the dataset.

We use the term “robust machine learning” as many of the techniques originate in the field of robust statistics. The term **“robust”** may seem somewhat unusual and confusing to some, but it is a well-established term in the field of statistics. It was coined in the 1950s and has been used ever since. Note that the term *robust machine learning* has been used in other contexts in the literature, but here we specifically refer to “outlier-tolerant” methods.

One may wonder why robust methods have not already been incorporated into machine learning tools. This is in part due to the long history of the non-robust estimation methods in statistics and their natural migration to the machine learning community over the past two decades. Attempts to use the L_1 loss function (which is robust) were not successful in the past, whereas the L_2 loss (which is not robust) was much easier to understand and implement. It is strongly tied to the Gaussian distribution which made it even more compelling, especially in terms of the maximum likelihood procedure. The same can be said of the cross-entropy loss used in binary classification. Most practitioners today are still employing least squares and cross-entropy methods, both of which are not robust in the presence of outliers. We will show that the log-cosh loss is robust, that it can be derived using maximum likelihood principles and inherits all the nice properties required of a loss function for use in machine learning. This removes all the past reasons for not using robust methods.

The approach taken in this book regarding outliers is to show how to robustify existing methods and apply them to data science problems. It revisits a number of the key machine tasks such as clustering, linear regression, logistic regression, and neural networks, and describes how they can all be robustified. It also covers the use of penalty estimators in the context of robust methods. In particular, the ridge, LASSO, and aLASSO methods are described. They are evaluated in terms of their ability to mitigate the effects of outliers. In addition, some very interesting approaches are described for variable ordering using aLASSO.

Outlier detection for regression and classification problems are addressed in detail. Previous approaches have not been able to perform this function without removing valuable data along with the outliers. The methods are essentially ad hoc in nature. In this book, practical solutions are provided using robust techniques. Of note is an iterative boxplot method for linear regression and a histogram-based method for classification problems. Anomaly detection is another form of outlier detection where the outliers are at extreme locations and represent unusual and unexpected occurrences in the dataset. Identifying such anomalies is very important in the detection of suspicious activity such as bank fraud, spam in emails, and network intrusion. The techniques to be described in this book include k-nearest neighbors (k-NN), DBSCAN, and Isolation Forest as they are popular techniques in this category. Also included is a new method based on robust statistics and k-medians clustering called MADmax, which is shown to provide better results than current methods.

We wanted to write a book suitable for senior-level (or fourth year) undergraduate and first/second-year graduate **students**, that is also useful as a stand-alone guide for **researchers** or **practitioners** in the field. As a result, there are equal parts of the theory and practice. Detailed derivations, theoretical support for the methods, as well as a substantial amount of "know-how" and experience are part of every chapter with code segments that can be executed by the reader to improve understanding. We found that when we view existing methods through the lens of outliers, it leads to a deeper understanding of how current methods work and why they may fail. In this sense, some new knowledge will be gained in every chapter.

The programming code provided in this book are based on **Python** which is the workhorse language for the machine learning community. Many libraries and utilities are available in Python. We introduce code segments for all of the techniques for regression and classification, as well as the code for outlier removal in the form of projects at the end of each chapter. The reader would be well-served to follow along the descriptions in the book while implementing the code wherever possible in Python. This is the best way to get the most out of this book.

The book is spread over 12 chapters. Chapter 1 begins with an introduction to machine learning and the importance of considering outliers in both regression and classification problems. Then, the k-means clustering algorithm is described and transformed into a k-medians algorithm as an example of the robustification of an existing method. Chapter 2 covers the L_1 and L_2 loss functions and describes a combination of the two called Huber's method. Chapter 3 provides a detailed analysis of the log-cosh loss function. Chapter 4 discusses outlier detection, metrics, and standardization. Chapters 5 and 6 address robust penalty estimation using ridge, LASSO and aLASSO. Chapter 7 introduces a flexible log-cosh loss function for quantile regression. Chapters 8–10 address the binary and multi-class classification problems and the robustness of various methods for these tasks. Chapter 11 addresses the important topic of anomaly detection. Finally, Chapter 12 concludes with applications of robust methods to some well-known data science problems.

A suitable undergraduate course in robust machine learning would consist of Chapters 1–6 and a graduate course would consist of Chapters 7–12. For practitioners, we recommend Chapters 1, 4, and 8–12. For researchers, we recommend Chapters 1–3, 5–7, and 12. It is our hope that readers will find something of value in each chapter, and that it will lead to many fruitful areas of future research and development.

Professor R. Saleh wishes to thank his parents, Dr. Ehsanes Saleh and Mrs. Shadidara Saleh, for their love and support. He also acknowledges the encouragement and support from Lynn Saleh, Jody Fast, Isme Alam, and Raihan Saleh. Special thanks to Dr. Mina Norouzirad for extensive reviews of all chapters and to Paolo Pignatelli for his contributions during the writing of this book.

Professor S. Majzoub is immensely grateful to his parents, Dr. Samir Majzoub and Mrs. Zeinab Khalidi, for their unwavering support and guidance. Heartfelt appreciation is expressed to his wife, Bayan Almogharbel, and children, Baraa, Asma, and Ibrahim, whose love and encouragement are cherished. Thanks are given to Haitham Tayyar, Mulhim Aldoori, Mottaqiallah Taouil, and Said Hamdioui for their invaluable assistance and encouragement throughout this journey.

Professor A.K. Md. E. Saleh is grateful to NSERC for supporting his research for more than four decades. He is grateful to his loving wife, Shahidara Saleh, for her support over 70 years of marriage. He also appreciates his grandchildren Jasmine Alam, Sarah Alam, Migel Saleh, and Malique Saleh for their loving care.

April 2025

Resve Saleh
Vancouver, Canada

Sohaib Majzoub
Sharjah, UAE

A.K. Md. Ehsanes Saleh
Ottawa, Canada

About the Companion Website

This book is accompanied by a companion website:

www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

The website includes:

- Student only: Python problems for students
- Instructor only: Answers

1 Introduction

The field of machine learning (ML) has been advancing very rapidly since 2010 when it gained momentum in both academia and industry. In the intervening years, many innovative ideas have been implemented and applied to some very challenging data science problems with great success. Underpinning the work in these areas are the mathematical foundations within the field of probability and statistics. The relationships between the three areas of data science, machine learning, and probability and statistics are conceptualized in Figure 1.1. The **interdependence of the three disciplines** is perhaps the most important point to note here. Data science relies heavily on probability and statistics and the machine learning tools that build the models for a particular application domain. Applications drive the investigation of the type of mathematics and machine learning capabilities that need to be developed. Likewise, machine learning relies on the foundations that lie in probability and statistics and also on the requirements of the applications that will eventually be used to develop the necessary tools. There is a symbiotic relationship between these three disciplines. This book places a strong emphasis on all three areas in equal measure to introduce the ideas and concepts to be described. As such, some familiarity with one of these three areas will be helpful in navigating through the material to be presented.

This book is about robust machine learning. It addresses an under-reported and often overlooked aspect of data science which is the effect of outliers on the model building process. A prediction is only as good as the model on which it is based, and if the model is faulty, so is the prediction. In particular, outliers and anomalies in datasets can lead to inaccurate models, resulting in bad business decisions, questionable explanations of cause-and-effect, incorrect conclusions, or inaccurate medical diagnoses, just to name a few. Even one outlier can render a model unusable if it happens to be in the wrong place. Robust methods are better-suited to data science, especially when outliers are present. ML practitioners have not yet fully embraced a class of robust techniques that would provide more reliable models and more accurate predictions than is possible with present-day methods. The overall goal of this book is to provide the rationale and techniques for robust ML and then build on that material toward robust data science.

The topics to be covered in this book span a wide variety of subjects and there is substantial ground to cover to gain *deep knowledge* in this field. This opening chapter covers introductory material and a number of basic concepts in machine learning. Initially, descriptions of robust machine learning and robust data science are provided as a backdrop for the material to be presented in the rest of the book. Then, we focus on robustifying one ML algorithm. There are several options to choose from in this initial chapter to demonstrate the value of using robust methods. Among the options, the one selected is simple

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.
© 2025 John Wiley & Sons, Inc. Published 2025 by John Wiley & Sons, Inc.
Companion Website: www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

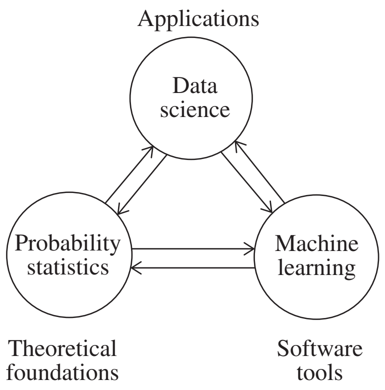


Figure 1.1 Synergistic relationships between data science, machine learning, and probability and statistics.

and yet provides a clear picture of the power of robust methods in general. Since clustering is one of the well-known applications of machine learning, it will be used as the first vehicle to understand the way that robust methods can be applied to handle outliers in datasets. It will also serve as a template for the rest of the book.

1.1 Defining Outliers

Outliers are typically a small subset of data that deviates from the majority of the data within the same dataset. Sometimes the true data is differentiated from outliers by calling them **inliers**. The sources of outliers can vary significantly. Outliers can occur naturally as part of the characteristics of the data being collected. Outliers can also be due to significant noise in the environment that might be unavoidable. More commonly, they are associated with measurement error, or instrumentation error. Another source is due to human error such as typographical errors or perhaps misinterpreting the units of measure of a device. If there are large variances in the features of a dataset and a small amount of data, this could lead to some of the data being declared as outliers. And other times, it is just inherently part of the data collection process. Unusual or extreme outliers are often referred to as **anomalies**.

There are basically two options of handling outliers. The first is to build machine learning tools that are tolerant to outliers. These are the robust machine learning tools. This involves understanding how the current ML tools work and then seeking an alternative approach that is more effective in the presence of outliers. Conceptually, the current approaches in ML seek the mean of the data to build a model. Robust methods seek the median instead which makes the models much more stable and reliable. This is one of the main thrusts of the book. The problems surrounding the use of the mean will be presented, followed by the advantages of using the median. More generally, the first approach is to convert a non-robust machine learning method into a robust one. Many of the existing techniques are not robust and those should be identified and updated accordingly. However, if a method is inherently robust, it should be preferred over its non-robust counterpart.

A second option is to detect and remove outliers and then apply the current ML tools or extract outliers from a dataset for further inspection. In current practice, there is a large investment in non-robust methodologies that are hard to change or replace, so it is easier to find ways to eliminate outliers and use the existing infrastructure. Since robust methods are not widely used in Python (or well-understood), it seems easier to remove outliers rather than build a new infrastructure based on robust methods.

However, as appealing as it may sound, it is not straightforward and requires an extra level of care to ensure that inliers are not inadvertently removed with outliers. There are other cases where the goal of the project is to find the outliers or anomalies in the dataset rather than to build a model. The detected outliers would be inspected to learn something about the anomalies or any unusual aspects of the data being collected. And finally, there are statistical tools and procedures that rely on the data being outlier-free in order to produce meaningful results (e.g. correlation coefficients and analysis of variance). Hence, the secondary thrust of the book is to describe efficient and effective methods to detect and remove both outliers and anomalies.

1.2 Overview of the Book

With the background and preliminaries covered, we will now move to the main thrust of this book which is the study of standard machine learning techniques and their robust counterparts. Table 1.1 provides a roadmap of the topics to be described and their associated chapters.

1.3 What Is Robust Machine Learning?

Machine learning involves the development of software that analyzes data and produces predictive models based on algorithms and mathematics developed by computer scientists, engineers, and statisticians. One can view it as the programming portion of the problem to build software tools and flows to process data in furtherance of some specific data science objective. Most of the existing techniques are essentially based on finding the **mean** of the data. Robust methods to be described in this book are based on finding the **median** of the data. Thus far, robust methods have not seen widespread use due to their somewhat mysterious nature. These techniques borrow heavily from *robust statistics* (see Maronna et al. 2019) which is a well-known and long-standing field in its own right. However, robust methods have only been of recent interest in machine learning. The objective of this chapter, and indeed the whole book, is to demystify robust methods so that more data scientists can gain access to such methods when the need arises.

Truth be told, most realistic datasets have outliers in one form or another. Some are harmless relative to the rest of the data while others may cause significant errors, especially during inference or prediction. The problem is that it is difficult to know *a priori* which case you are dealing with. You may not be able to tell if the results are reliable unless additional steps are taken to ensure reliability. If, for example, the outliers could be removed in some effective manner, then the traditional methods are quite effective. However, outlier detection and removal are prone to error so this may not be a reliable approach either. It is often considered to be tampering with the data. Unless you can successfully diagnose and detect outliers in large datasets, the models generated by standard ML methods on these modified datasets should be viewed as suspect. The more prudent approach is to employ machine learning tools that use robust procedures wherever possible to avoid the problems altogether.

Generally speaking, robust ML is the development of tools, procedures, and methodologies that are tolerant to outliers. In its most simplistic form, it implies the use of the median (or some neighboring quantile) rather than the mean thereby producing outlier-tolerant models. It also includes the detection and removal of outliers using robust ML. Of course, robust methods introduce a set of different issues that must be addressed before they can be used effectively. As we will show throughout this book,

Table 1.1 Topics, chapters, and descriptions.

Topic	Chapter	Description
Robust Clustering	1	k-means clustering algorithm Robust k-medians clustering algorithm
Robust Linear Regression	2 3 4	L_1, L_2, Huber loss functions log-cosh loss function Outlier detection, metrics (MSE, MAE), robust standardization
Robust Models via Regularization	5 6	Penalty functions (ridge, LASSO, aLASSO) Model generalization, model complexity
Quantile Regression	7	Replacing the quantile check function with a flexible log-cosh function
Robust Binary Classification	8	Logistic regression Cross-entropy vs. log-cosh loss functions, SVM, kNN, random forest
Neural Networks for Binary Classification and Classification Metrics	9	Activation functions: relu, sigmoid, training, backprop, gradient descent, cross-entropy vs. log-cosh loss functions recall, precision, F_1 score, AUROC
Multi-class Classification and Optimization	10	Categorical loss, softmax activation Modified National Institute of Standards and Technology (MNIST) dataset, Adam optimization
Anomaly Detection and Evaluation Metrics	11	kNN, Isolation Forest, density-based spatial clustering of applications with noise (DBSCAN), MADmax, precision, recall, AUROC
Application to Data Science and Artificial Intelligence (AI)	12	Boston housing, Titanic datasets climate change time series (ARIMA) explainable AI (XAI)

the advantages of robust ML greatly outweigh the disadvantages. Our goal is to provide side-by-side comparisons of traditional methods with robust methods so that the reader can make their own judgments about the value of robust ML.

1.3.1 Machine Learning Basics

Before launching into the key ideas of this book, it is useful to provide some background material on ML and robust methods. ML grew out of the computer science field in the 1980s but went dormant for a time in the 1990s, during which it was viewed as a dead-end field. It experienced a reawakening around 2000 and a significant resurgence around 2010. It continues to gain traction and momentum to this day with the advent of generative AI. Of course, in the 1980s, computers were relatively slow, datasets were not readily available, and algorithms for ML were in their infancy. Fast forward a few decades and we find the landscape has changed dramatically. Today, access to high-speed computing is at everyone's fingertips, data is ubiquitous, and many advanced ML algorithms have been developed, implemented, and tested. They are now widely available through open-access online mechanisms.

Some of the algorithms in ML are grounded in statistical theory, while others are based on nonparametric methods or a set of heuristics. We will study methods that fall into all these categories. There are three main varieties of ML tools: supervised learning, unsupervised learning, and reinforcement learning.[1] We focus only on the supervised and unsupervised learning methods in this book. To understand these two methods, consider a dataset given by a matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, where n is the number of observations (rows in the dataset) and d is the number of variables (columns in the dataset). The "ground truth" or true responses associated with each observation is optionally provided in a vector $\boldsymbol{y} \in \mathbb{R}^n$. To contrast the two categories, supervised ML requires both $\boldsymbol{X}$ and $\boldsymbol{y}$, while unsupervised ML requires only $\boldsymbol{X}$.

For example, linear regression, logistic regression, neural networks, k-nearest neighbors, tree-based methods, and so on fall into the supervised category. In this book, we focus on the robustness aspects of these methods. On the other hand, data clustering and anomaly detection are examples of unsupervised learning methods. We will discuss robust clustering in this chapter as a way of introducing robust concepts and their value in machine learning. Anomaly detection will be covered in Chapter 11.

For supervised learning, we mainly consider two types of problems: **regression** and **classification**. Figure 1.2 shows the two cases graphically. On the left side, we show a set of points that are fitted with a line, where the x-axis is the *year* and the y-axis is the number of *sales* in that *year*. Fitting a line to the data is called *linear regression*. The ML objective in this case is to use the data to find the **slope and intercept**

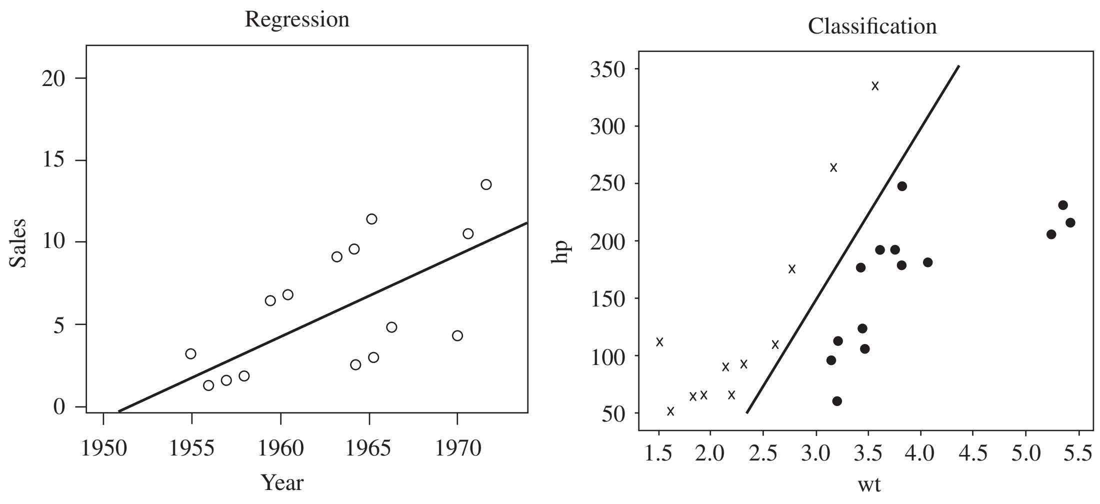


Figure 1.2 Linear regression vs. linear classification.

1 Although an important topic in ML, reinforcement learning is outside the scope of this book.

of the line that “best fits” the given points shown in the plot. The best fit is based on some objective function used during optimization. We are, in effect, training the model to learn about the dataset. Once the estimates of these parameters are obtained, predictions can be made for values that are not in the original dataset. For example, we can now test our model by asking the question: “what is the expected number of *sales* for *year* = 1968?” and the answer comes back as “*sales* = 8.2.” We did not have this value in the dataset before but now we are able to make predictions using the fitted line. Building a prediction model is the essence of machine learning.

The figure on the right shows two sets of points, represented using symbols “•” and “x,” respectively. Each symbol represents a distinct class. The ML objective here is to establish a line that best separates the “•” points from the “x” points. Finding the dividing line between the two classes is referred to as *linear classification*. The data is linearly separable in this case and the separating line does a good job of defining a *decision boundary* between the two sets of points. We say that the model was trained using the given data, which actually means that we optimized an objective function to produce the slope and intercept parameters. We can now ask another question: “if a new point is introduced at wt = 2.5 and hp = 90, what class does it belong to?” and the answer would come back as “it belongs to the ‘x’ class.” A large portion of the supervised machine learning tools involve this type of classification.

1.3.2 Effect of Outliers

Now consider the effect of outliers on the two cases above. This is shown in Figure 1.3. On the left side, there is one new data point which is an outlier. The effect on traditional linear regression is to shift the line and make it more horizontal than in Figure 1.2. Now if the question is asked, “what is the expected *sales* for *year* = 1968?,” the answer comes back as “*sales* = 6.5” which is not accurate. Apparently, the one outlier has produced misleading results. If used to make a business decision, this outlier may prove to be very costly.

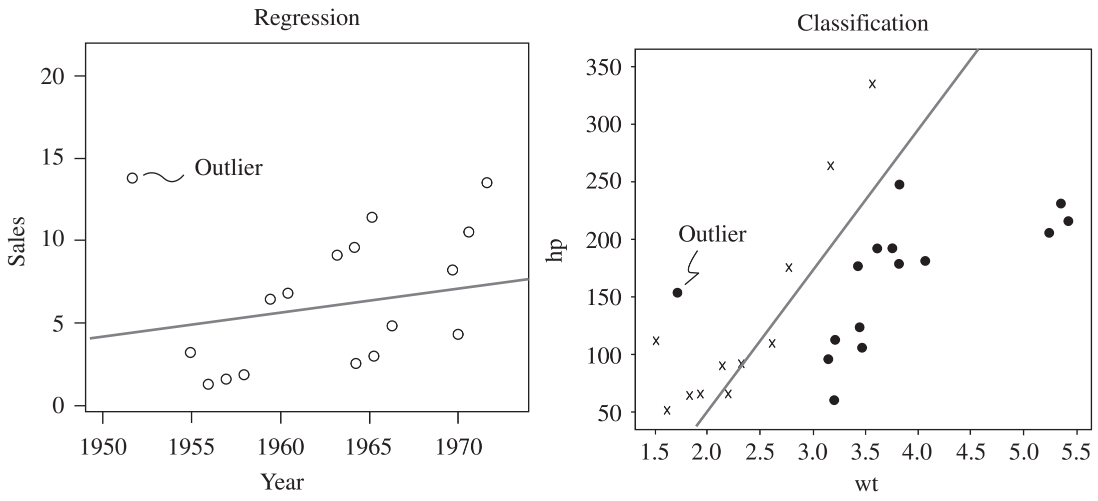


Figure 1.3 Impact of outliers in regression and classification.

In the second case on the right side of Figure 1.3, we place an outlier by adding a "•" on the left side of the boundary, where members of the "x" class reside. The effect of this outlier is to shift the line toward it, relative to the corresponding one in Figure 1.2. The outlier should not affect the boundary, but it does. When we again ask, "if a new point is introduced at wt = 2.5 and hp = 90, what class does it belong to?," the answer would come back as "it belongs to the '•' class." Of course, this is incorrect. It belongs to the "x" class as we saw earlier. The outlier has moved the decision boundary thereby changing the predicted response. As before, any business decision made using these results may be costly.

Ideally, we would like to obtain the results shown in Figure 1.2 given the data of Figure 1.3. Robust methods offer this possibility along with the methodologies to achieve this outcome. Robust ML involves methods that are suitable for datasets with or without outliers. They produce models as if the data were outlier-free. This is the key benefit of using robust methods. You do not need to wonder what the effect of outliers may be in the dataset or where they are located. You simply use this class of methods on all problems without any concerns about outliers. In some applications, you may be interested mainly in detecting and removing the outliers. This issue is also addressed by robust methods, as detailed in this book.

1.3.3 What Is Robust Data Science?

Data science involves a set of application-specific tasks that focus on creating a coherent dataset from a large number of observations, performing thorough exploratory data analysis (EDA), and carefully interpreting results obtained from machine learning tools and statistical analysis. It is a rather expansive field with many layers of depth and a wide variety of applications. We will briefly describe some of these layers and emphasize the key aspects of this discipline. If we step back and look at statistics and machine learning from a higher perspective, we can understand their roles in data science. Figure 1.4 shows the traditional view. We start with the ground truth in the form of a probability distribution (assumed to be the true distribution of a population). By sampling this distribution, we generate a dataset. Once we have a dataset, then traditional statistics/ML seeks to find a model that best approximates the ground truth (i.e. the original distribution). From this model, we can do whatever we want, such as making predictions. Unfortunately, this is an idealized view of the world of data science.

1.3.4 Noise in Datasets

In the real world, there is noise that will add outlier observations to our dataset. Consider a slightly modified version of Figure 1.4 as shown in Figure 1.5. Here, the data generation path is modified to include additive noise. It is the "signal" plus "noise" that combine to produce more realistic datasets.

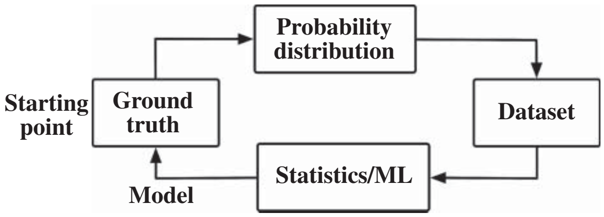


Figure 1.4 Traditional view of the model building process.

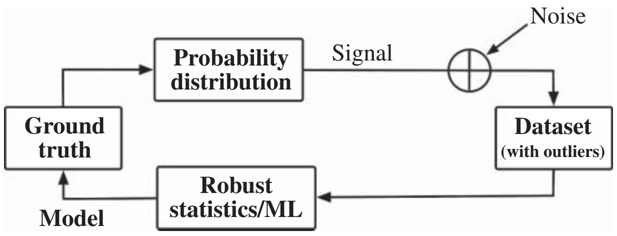


Figure 1.5 Realistic view of the model building process.

This is not the typical Gaussian noise that is usually taken into account. Rather, the noise is disruptive enough to produce bad models if special care is not taken. As a result, in order to construct a model that best approximates the ground truth, we must turn to robust statistics/ML that effectively removes noise and generates a better model. Without robust methods, we may not be able to produce high-quality models that approximate the ground truth when outliers are present.

In robust data science, the focus is on identifying outliers, utilizing data cleaning methods to remove unwanted outliers or applying procedures that are largely insensitive to such outliers. We believe that the field will naturally migrate in this direction, given the fundamentals and benefits of the methods to be presented.

1.3.5 Training and Testing Flows

Data science involves building models using datasets and a number of supervised and unsupervised ML tools. A typical flow for supervised learning is shown in Figure 1.6. We are given an initial dataset with a design matrix $\boldsymbol{X}$ and known responses $\boldsymbol{Y}$. Here, $\boldsymbol{Y}$ is usually a vector but can also be a matrix of responses depending on the application. The dataset is divided into a training set and a test set. We build models using training data and then assess the accuracy using test data. There are many different ways to determine the overall accuracy as described in later chapters. Not shown in the figure are the potential iterations that are needed to experiment with different *hyperparameters*, which are user-specified values that steer the optimization methods toward a desirable model.

The sequence of steps is as follows. Let us assume that the dataset contains outliers. First, the data is randomly split into a training set and a test set. We will have no idea where the outliers have landed; they could be all in the training set, all in the test set, or in both. In practice, we must assume they are in both and develop methodologies and metrics under that assumption. The training data, shown as

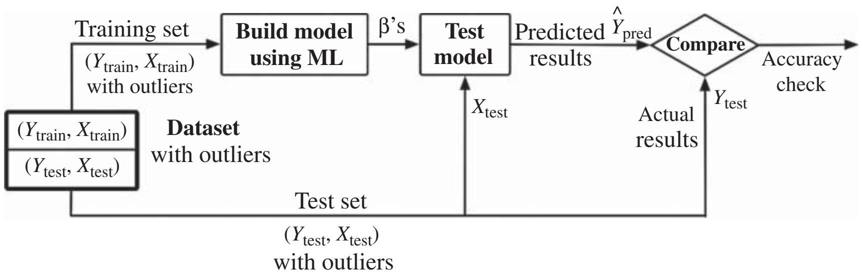


Figure 1.6 Training and test sets for supervised learning.

$(\boldsymbol{X}_{\text{train}}, \boldsymbol{Y}_{\text{train}})$, with potential outliers, is used as input to a selected ML tool, say logistic regression or a neural network, in order to estimate the model parameters, $\boldsymbol{\beta}$. Often, a cross-validation (CV) scheme is used whereby the training set is divided into a number of subsets, called *folds*, so that a more general model can be produced by combining the results of k different folds. Next, the prediction accuracy of the model must be evaluated on an independent test set $(\boldsymbol{X}_{\text{test}}, \boldsymbol{Y}_{\text{test}})$ to properly assess its accuracy. This is carried out by taking $\boldsymbol{X}_{\text{test}}$, possibly with outliers, and applying the trained model to make predictions, and then comparing the predicted results, $\hat{\boldsymbol{Y}}_{\text{pred}}$, against the true values, $\boldsymbol{Y}_{\text{test}}$.

Quality assessment is typically performed using an appropriate set of metrics or statistics.[2] One must be careful when using an error measure with outliers present. Since outliers may appear in the training set, the test set, or both, there may be a need for robust error measures, which will differ for regression and classification. Consider the case where the training set has no outliers but the test set is full of outliers. No amount of training will be able to produce good results on the test set using standard error measures. So we will have to select some suitable robust error measure in order to properly evaluate accuracy.

In unsupervised learning, the dataset is given by $\boldsymbol{X}_{\text{train}}$ but does not involve $\boldsymbol{Y}_{\text{train}}$ or a test set. However, the goal is still to minimize some objective function associated with the specific machine learning task. Whether we are doing supervised or unsupervised learning, the presence of outliers can affect the quality of the model so we have to consider robust ML. With this brief rationale, we can now proceed to the technical details of how robust ML works and where it should be applied.

1.4 Robustness of the Median

Our starting point for robust ML is a brief review of two useful statistics: the mean and the median. The mean is very well-known and forms the basis for the most popular ML methods, whereas the median is less well-known but important for our purposes here.

1.4.1 Mean vs. Median

Given a set of n values, $\boldsymbol{x} = \{x_1, \dots, x_n\}$, the mean is given by

$$\bar{x} = \text{mean}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i\,, \tag{1.4.1}$$

which is the sum of the values divided by n.

The median of the set depends on whether n is odd or even. First, we order the values from smallest to largest. After ordering, the new set is denoted by $\{x_{(1)}, \dots, x_{(n)}\}$, where $x_{(1)} \le \cdots \le x_{(n)}$. These are referred to as the order statistics of $\boldsymbol{x}$ with $x_{(1)} = \min\{x_i, i = 1, \dots, n\}$ and $x_{(n)} = \max\{x_i, i = 1, \dots, n\}$, and the $x_{(i)}$'s are the ordered observations. If n is odd, we take the middle value in the ordered set,

$$\text{median}(\boldsymbol{x}) = x_{(\frac{n+1}{2})}. \tag{1.4.2}$$

If n is even, we take the average of the two values in the middle of the ordered set. Assuming the above ordering, it can be expressed compactly as follows:

$$\text{median}(\boldsymbol{x}) = \frac{1}{2}\left[x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}\right]. \tag{1.4.3}$$

2 A statistic is any calculation or measurable quantity that can be obtained using data.

Here, the upper value $x_{(\frac{n}{2}+1)}$ and lower value $x_{(\frac{n}{2})}$ are averaged to obtain the median, but technically *any value between the upper and lower values* can serve as the median which implies an infinite number of possible values.

We can already see a problem in that the median seems a little complicated, requires several steps, along with a caveat when n is even, while the mean is easy to understand in one simple equation. This provides insight into why the median has not been embraced as readily as the mean. Furthermore, the mean is always **unique**, whereas the median may or may not be unique. In fact, it can either be unique or any one of an **infinite number of values** in a bounded region. However, the median is more important when it comes to finding the essence of a dataset with outliers, as we will see shortly. It is for good reason that we hear phrases like "the median house price" or "the median household income."

Table 1.2 compares and contrasts these two statistics for some trivial cases. The first set $\{3, 1, 4, 2, 5, 3\}$ produces 3 for both the mean and median. This is consistent with a visual examination of the values in the set. However, if we change one of the values from 3 to 100 to produce the second set, $\{100, 1, 4, 2, 5, 3\}$, the mean increases significantly to 19.2, while the median is 3.5, which is close to the previous value. In this case, the value of 100 would be considered an *outlier* because it is far away from the rest of the data. The mean is greatly affected by one outlier which makes it unstable. On the other hand, the median seems rather stable in the presence of one outlier.

Continuing the exercise, if we examine Set 3 listed as $\{100, 100, 1, 4, 2, 5, 3\}$ in the final column of Table 1.2, we now have two outliers and the mean shifts further beyond the range of the original elements of Set 1 from 3.0 to 30.7. On the other hand, the median maintains relative stability at 4.0. We refer to the median as having the *robustness* property for this reason.

Returning to Set 2 for a moment, if we let $x_{(n)}$ represent the outlier value and set $x_{(n)} = 100$ initially, what happens as $x_{(n)} \to \infty$? The mean tends to ∞ while the median remains stable at 3.5. This is fundamentally why robust methods are needed in ML. If outliers are present, the model cannot be trusted using traditional methods.

1.4.2 Effect on Standard Deviation

The standard deviation is also greatly affected by outliers. Let μ be the population mean, and let $\bar{x}$ of Eq. (1.4.1) be an estimate of μ. Further, let the population variance, σ^2, be defined by

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2,$$

Table 1.2 Comparison of mean and median, and standard deviation (SD), and the normalized median absolute deviation (MADN) on three small datasets.

Measure	**Set 1** $\{3, 1, 4, 2, 5, 3\}$	**Set 2** $\{100, 1, 4, 2, 5, 3\}$	**Set 3** $\{100, 100, 1, 4, 2, 5, 3\}$
Mean	3.0	19.2	30.7
Median	3.0	3.5	4.0
SD	1.41	36.9	47.3
MADN	1.48	2.22	3.0

and the unbiased estimator of σ be given by

$$\mathrm{SD}(\boldsymbol{x}) = \hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{1.4.4}$$

which is the standard deviation (SD) of the sample set. Then, as the outlier increases to infinity, the mean goes to infinity and, as a consequence, the standard deviation goes to infinity. Hence, one outlier can have a domino effect on various other statistics that rely on the mean.

To address this issue, each statistic based on the mean could be replaced by a robust counterpart. A robust statistic for standard deviation is the median absolute deviation (MAD) given by

$$\mathrm{MAD}(\boldsymbol{x}) = \underset{1\le i\le n}{\mathrm{median}}\left(|x_i - \mathrm{median}(\boldsymbol{x})|\right). \tag{1.4.5}$$

A consistent and robust estimator of the population standard deviation based on a **normalized MAD** can be computed as follows:[3]

$$\mathrm{MADN}(\boldsymbol{x}) = k \times \mathrm{MAD}(\boldsymbol{x}), \tag{1.4.6}$$

where $k = 1.4826$. This is used so that SD($\boldsymbol{x}$) and MADN($\boldsymbol{x}$) are approximately equal for outlier-free data. If we take Set 1 in Table 1.2, the original set with no outliers, then SD($\boldsymbol{x}$) = 1.41 and MADN($\boldsymbol{x}$) = 1.48. For the case of Set 2 with $x_{(n)} = 100$, SD($\boldsymbol{x}$) = 39.6 and MADN($\boldsymbol{x}$) = 2.22. Note that MADN($\boldsymbol{x}$) is stable compared to SD($\boldsymbol{x}$).

If we now change the outlier in Set 2 to $x_{(n)} = 1000$, then SD($\boldsymbol{x}$) = 407.0 and MADN($\boldsymbol{x}$) = 2.22; that is, MADN remains unchanged. Clearly, a single large outlier can skew basic statistics such as the mean and standard deviation, while the median and MADN($\boldsymbol{x}$) remain relatively stable even as $x_{(n)} \to \infty$. For Set 3, with two outliers, SD($\boldsymbol{x}$) = 47.3 and MADN($\boldsymbol{x}$) = 3.0. We seek this type of stability in most data science problems with outliers.

Exercise: Given the set $\boldsymbol{x} = \{1, 2, 3, 4, 5, 1000\}$, compute the mean, median, SD($\boldsymbol{x}$), and MADN($\boldsymbol{x}$).

Ans.: mean =1015/6 =169.17, median = 3.5, SD($\boldsymbol{x}$) = 407, MADN($\boldsymbol{x}$) = 2.2. (Note: to be more precise, the median $\in (3, 4)$. We use the average of 3.5 simply for convenience when we seek a unique value for it). □

1.5 L_1 and L_2 Norms

There are two vector norms that are pertinent to our discussion of traditional methods vs. robust methods: the so-called L_1-norm and L_2-norm. These norms are simply measures of the magnitude of a vector or the distance between two vectors. In particular, the two norms of a vector $\boldsymbol{v}$ with n elements are given as follows:

$$L_1 = \|\boldsymbol{v}\|_1 = \sum_{i=1}^{n}|v_i| = |v_1| + |v_2| + \dots + |v_n|,$$

3 Here, the word *consistent* means that MADN(x) will produce the true value if the sample size is large enough (see Chapter 2). We will use MADN(x) whenever k is used as a multiply factor of MAD(x). We will use MAD(x) when we do not apply k. Here, k is a factor that depends on the distribution. For a Gaussian normal distribution, $k = 1.4826$.

and

$$L_2 = \|\boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2} = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}.$$

The Manhattan distance between two vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$, is given by

$$L_1 = \|\boldsymbol{u} - \boldsymbol{v}\|_1 = \sum_{i=1}^{n} |u_i - v_i|,$$

and the Euclidean distance is given by

$$L_2 = \|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}.$$

Exercise: Let $\boldsymbol{x} = (-1, 2, 3)^\top$. What are the values of the L_1 and L_2 norms?

Ans.: $L_1 = 6, L_2 = \sqrt{14}$. (Note: the $\top$ superscript used in $(-1, 2, 3)^\top$ refers to the transpose of the vector as it is being represented horizontally for convenience rather than vertically.) □

Exercise: Let $\boldsymbol{u} = (-1, 2, 3)^\top$ and $\boldsymbol{v} = (-2, 3, 4)^\top$. What is the distance between these two vectors in terms of L_1 and L_2 norms?

Ans.: $\|\boldsymbol{u} - \boldsymbol{v}\|_1 = 3, \|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{3}$. □

Often, the phrase "$L2$" may be used even if we square the value of the norm as in

$$L_2^2 = \|\boldsymbol{u} - \boldsymbol{v}\|_2^2 = \sum_{i=1}^{n} (u_i - v_i)^2.$$

This looser definition is used in cases where we are comparing relative magnitudes, especially in the case of distance measures, loss functions, or penalty functions. Typically, we are referring to the sum of squares when we use the phrase "$L2$" and sum of absolute values when referring to "$L1$."

Exercise: Let $\boldsymbol{u} = (-1, 2, 3)^\top$ and $\boldsymbol{v} = (-2, 3, 4)^\top$. What are the values of $\|\boldsymbol{u} - \boldsymbol{v}\|_2^2$ as compared to the L_2-norm given by $\|\boldsymbol{u} - \boldsymbol{v}\|_2$?

Ans.: $\|\boldsymbol{u} - \boldsymbol{v}\|_2^2 = 3, \|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{3}$. One takes the square root of the sum of squares while the other does not take the square root. □

1.6 Review of Gaussian Distribution

One of the most important distributions in statistics is the well-known Gaussian or normal distribution. This subject has been covered in countless papers, books, and online resources. It will be referenced many times throughout this book, so it is presented here for completeness. It is assumed that the reader is well-versed in this area as a prerequisite, but a brief tutorial is provided for review purposes.

The Gaussian distribution has a probability density function (PDF) given by

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1.6.1}$$

where μ is the mean and σ^2 is the variance. A random variable, X, is said to follow a Gaussian distribution if any sample is distributed according to the equation above. A more compact notation is

$$X \sim \mathcal{N}(\mu, \sigma^2). \tag{1.6.2}$$

This is equivalent to Eq. (1.6.1) except that we are now explicitly stating that X is a random variable distributed as a Gaussian with mean μ and variance σ^2.

The cumulative distribution function (CDF) of the Gaussian is defined as

$$F_X(x) = \mathbb{P}(X \le x) = \int_{-\infty}^{x} f_X(t)dt. \tag{1.6.3}$$

The CDF gives the probability that a random variable X is less than a specific number x. It is obtained by taking the integral of the PDF from $-\infty$ to x. The total area under any PDF is 1.0 as an axiom of probability theory. Therefore, the results of the integral will always lie in the range $[0, 1]$ as probabilities cannot exceed this range.

We are often interested in computing probabilities from the CDF of the Gaussian distribution but since the integral in Eq. (1.6.3) does not have a closed form, we resort to lookup tables using z-scores which have a normal distribution. The z-score is computed as

$$z = \frac{x - \mu}{\sigma}, \tag{1.6.4}$$

where z is the standardized value of x and $Z \sim \mathcal{N}(0, 1)$. For notational convenience, we refer to the PDF of the Gaussian as $\mathcal{N}(\mu, \sigma^2)$ and the CDF of the Gaussian as $\Phi(x)$.

The multivariate Gaussian distribution for a random variable, $\boldsymbol{X} \in \mathbb{R}^d$, is denoted by

$$\boldsymbol{X} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{1.6.5}$$

where $\boldsymbol{\mu}$ is a $d \times 1$ vector and $\boldsymbol{\Sigma}$ is a $d \times d$ covariance matrix, with the corresponding PDF given by

$$f_X(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \mathrm{e}^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}, \tag{1.6.6}$$

where $\boldsymbol{\Sigma}^{-1}$ is the inverse of the covariance matrix and $|\boldsymbol{\Sigma}|$ is the determinant.

While the Gaussian distribution is invaluable in statistics, we will later discover that we need to move away from this elegant distribution in favor of other lesser-known distributions to achieve robustness.

1.7 Unsupervised Learning Case Study

We are now ready for our first encounter with a machine learning problem that falls in the category of unsupervised learning. We will take a deep dive into data clustering in the rest of this chapter to introduce and illustrate many of the basic concepts in machine learning with a particular focus on the rationale and proper use of robust techniques. Many of the methods in use today are not robust and therefore our goal is to understand why the existing tools may not be good choices and where robustness can help most.

1.7.1 Clustering Example

Clustering (see James et al. 2023) is a well-known and widely understood problem in ML so it serves as a good entry point into our study of robust methods. The procedure in clustering involves taking a set of unassigned multi-dimensional data points, $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, and assigning them to one of K groups or clusters based on their proximity to one another. The value of K is always an integer and serves as a hyperparameter specified by the user based on the application of interest with $K > 1$ (since $K = 1$ is a trivial case). We will walk through the basics of clustering starting with simple problems and then work our way up to the complete set of algorithms for both non-robust and robust methods. In the process, we will follow the loops of Figures 1.4 and 1.5.

Exercise: Let the matrix $\boldsymbol{X}$ be defined as follows:

$$\boldsymbol{X} = \begin{pmatrix} 1 & 2 \\ 1 & 4 \\ 2 & 3 \end{pmatrix}.$$

What is the value of n and d in this case? In this chapter, we will be referring to row i of this matrix using the notation $\boldsymbol{x}^{(i)}$. What is $\boldsymbol{x}^{(2)}$? What is $\boldsymbol{x}^{(2)} - \boldsymbol{x}^{(1)}$?

Ans.: Throughout this book, the notation $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ will be used to refer to a real matrix $\boldsymbol{X}$ with n rows and d columns. In this trivial example, $n = 3$ since there are 3 rows, and $d = 2$ since there are 2 columns. Then, $\boldsymbol{x}^{(2)} = (1, 4)^\top$. Also, $\boldsymbol{x}^{(2)} - \boldsymbol{x}^{(1)} = (0, 2)^\top$. □

1.7.2 Clustering Problem Specification

Given a set of n points in $\mathbb{R}^d$, and an integer hyperparameter, K, the goal of clustering is to assign each point to one of K clusters, $C_1, \dots, C_K$, based on their proximity to a set of defined cluster centers, $\boldsymbol{\mu}_1, \dots \boldsymbol{\mu}_K$, such that it minimizes a distance-based cost function. The cluster centers are computed using the set of points assigned to each cluster. The algorithm used for clustering starts with an initial assignment of points to clusters and then proceeds in an iterative fashion until convergence of cluster assignments is achieved.

To understand the clustering process, consider Figure 1.7. The figure on the left is the original dataset in $\mathbb{R}^2$ represented by variables x_1 and x_2 for each point. The number of clusters is chosen to be $K = 3$ in this case and the clustering procedure after several iterations produces the figure on the right. The representative centers are indicated with the symbol $\otimes$. The three circular/elliptical boundaries around the data are shown only as a visual aid to identify the different clusters. Often, different colors or symbols are used to represent different clusters.

The problem of clustering is harder than it looks; in fact, it belongs to the class of NP-hard problems, meaning that it cannot be solved in polynomial time as the problem size increases. Therefore, locally optimal solutions are usually sought using "greedy" algorithms, starting with an initial guess for the centers and then repeating the process of clustering the data and recomputing the centers until convergence. Results may vary significantly depending on the choice of initial starting points.

A generic unsupervised clustering algorithm is shown in Algorithm 1.1. It begins by randomly picking K centers. Then, the iterative loop is entered. In the first phase of the loop, K groups are defined by

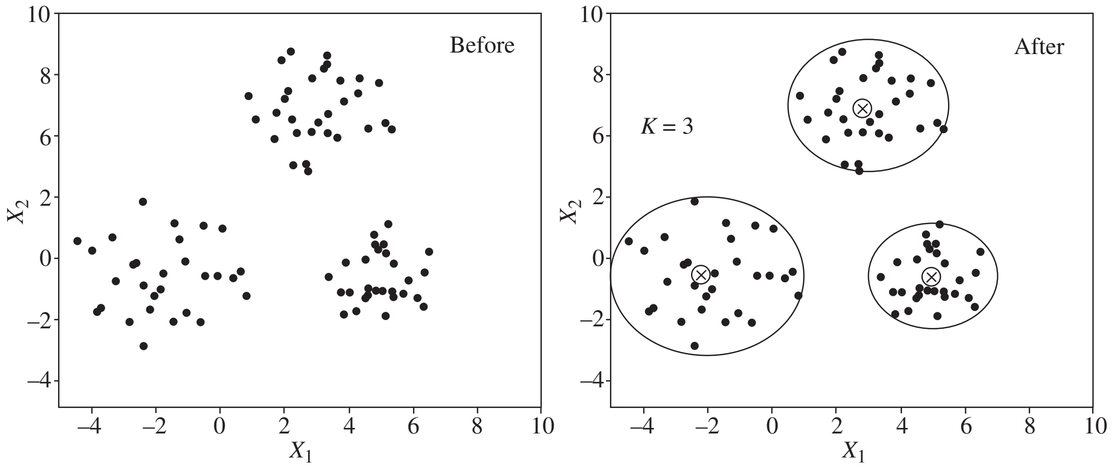


Figure 1.7 Dataset before and after clustering.

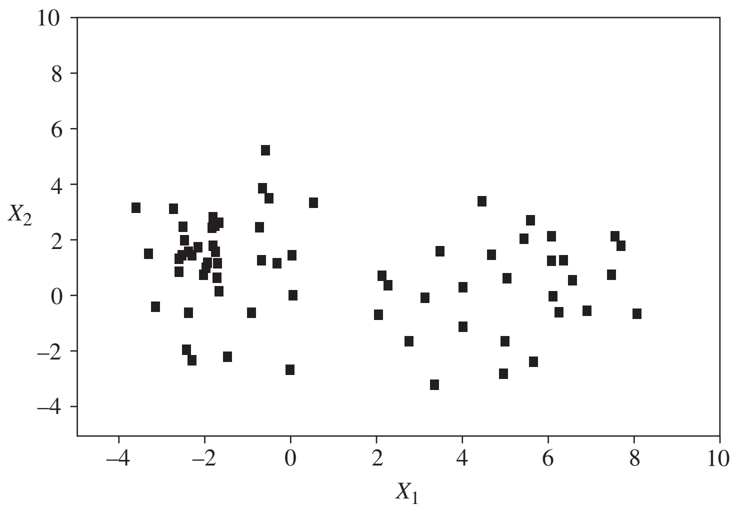
assigning data points to their closest center. In the second phase, the centers of each group are recomputed using the data points assigned to it. Then a suitable cost function is computed for the new arrangement. The iterations continue through the two phases until convergence is achieved. This occurs when the cost function is minimized and the K groups no longer change. The loop is referred to as a Lloyd-style iteration process.

The algorithm is sensitive to the initial guesses so it is worth discussing some of the options for initialization. The simplest approach is to pick K random points from the n total points in the dataset as suggested in the algorithm. Unfortunately, these random values may be close to one another and that is not desirable because it may lead to unbalanced clustering. Another way is to choose $K + m$ initial centers and later remove m centers to produce K clusters. This is sometimes used to find outliers in the data. A third initialization scheme involves picking K centers that are far away from each other. This allows a more balanced set of clusters to form. In any case, some clever method should be used to pick the K centers. Otherwise, the results may not be satisfactory for their intended purpose.

Algorithm 1.1 Generic clustering algorithm using Lloyd-style iteration.

```
1: Input: dataset $\boldsymbol{X}$, number of clusters = $K$;
2: Initialize centers using $K$ random points of $\boldsymbol{X}$, $\text{cost}_{\text{old}} = 0$, $\text{cost}_{\text{new}} = \text{maxcost}$;
3: repeat until ($|\text{cost}_{\text{new}} - \text{cost}_{\text{old}}| < \varepsilon$):
4:     Define $K$ new clusters, $C_1, \dots, C_K$, by assigning each point to the nearest center;
5:     Recompute $K$ cluster centers of $C_1, \dots, C_K$;
6:     $\text{cost}_{\text{old}} = cost_{\text{new}}$;
7:     Compute $\text{cost}_{\text{new}}$ using a distance-based cost function;
8: end repeat
9: Output: final clusters, $C_1, \dots, C_K$;
```

Exercise: Consider the scatter plot of data points below. What is the value of d and a suitable value of K in this case?



Ans.: Here, $d = 2$ because it is a two-dimensional problem, and it appears that $K = 2$ is a suitable choice since there are apparently two clusters. □

1.8 Creating Synthetic Data for Clustering

In the development of ML tools, we will often create representative datasets to validate the workings of the program. These are referred to as *synthetic datasets*. Synthetic data is also used to test corner cases, introduce artificial outliers, and augment existing real-world datasets to improve the quality of the models generated by ML tools. Before we solve the clustering problem, we need a way to generate synthetic datasets that are composed of any number of clusters in a multi-dimensional space. We start with a probability distribution of some sort as depicted in Figure 1.4. For clustering, an appropriate distribution is a linear combination of several Gaussian distributions called a **Gaussian mixture**. For simplicity, we start the case of two clusters and then generalize to K clusters. For testing purposes, we also need to generate clusters of points in 1-dimension, 2-dimensions, or any size up to d-dimensions.

1.8.1 One-Dimensional Datasets

A one-dimensional problem with two Gaussian distributions in a mixture takes the following form:

$$f_X(x) = \pi_1\, \mathcal{N}(\mu_1, \sigma_1^2) + \pi_2\, \mathcal{N}(\mu_2, \sigma_2^2), \tag{1.8.1}$$

where π_i are the probability weighting factors (**not** the numerical value π) associated with each distribution of the form $\mathcal{N}(\mu_i, \sigma_i^2)$. By definition, $\pi_1 + \pi_2 = 1$. If we had K Gaussian distributions in the mixture,

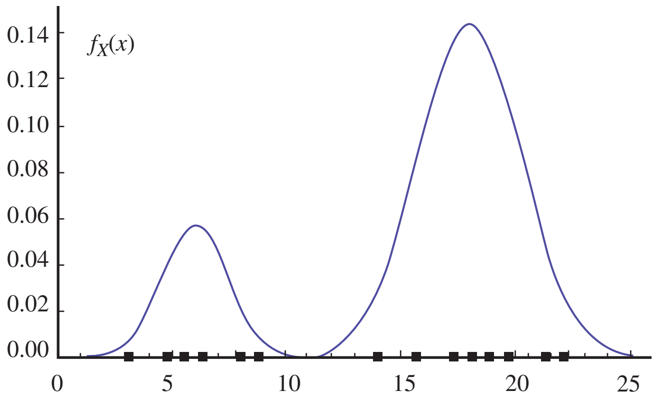


Figure 1.8 One-dimensional Gaussian mixture.

then we would require that

$$\sum_{i=1}^{K} \pi_i = 1$$

in order to satisfy a basic probability axiom associated with a valid PDF.

Example: The graph shown in Figure 1.8 represents a one-dimensional Gaussian mixture. It is based on the Gaussian mixture model of Eq. (1.8.1). The parameters of the mixture are $\pi_1 = 0.2$, $\pi_2 = 0.8$, $\mu_1 = 6$, $\sigma_1^2 = 2$, $\mu_2 = 18$, and $\sigma_2^2 = 5$. When we draw a sample of 14 observations from such a distribution, we obtain a set of random points as depicted along the x-axis. This comprises the dataset that we are given without any prior knowledge about the original distribution. The objective is to identify the two clusters using this data and reconstruct the original distribution on which the sample was based, as represented earlier in Figure 1.4. That is, we want to build a model of an unknown distribution from which the data arose given only the random sample of points. The key concept is that data is always tied to some unknown distribution. The best we can do with the given data is to build an approximate model of it using machine learning. □

Exercise: What is the area under the curve (AUC) of the Gaussian mixture of Figure 1.8 from $-\infty$ to ∞?

Ans.: The AUC = 1.0 by definition of a valid probability distribution. In fact, all valid PDFs have normalizing constants to ensure that they integrate to 1. □

1.8.2 Multidimensional Datasets

Consider a more realistic d-dimensional Gaussian distribution denoted by $\mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = (\mu_1, \dots, \mu_d)^\top$ and $\boldsymbol{\Sigma}$ is a $d \times d$ nonsingular covariance matrix. In its most general form, we have the following Gaussian mixture:

$$f_X(\boldsymbol{x}) = \sum_{i=1}^{K} \pi_i \, \mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i). \tag{1.8.2}$$

Thus, we may define mixtures of two d-dimensional Gaussian distributions as

$$f_X(\boldsymbol{x}) = \pi_1\,\mathcal{N}_d(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + \pi_2\,\mathcal{N}_d(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2). \tag{1.8.3}$$

In the rest of this chapter, we will assume that the covariance matrix takes the simplified form of $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}_d$, where $\boldsymbol{I}_d$ is a $d \times d$ identity matrix. This is suitable for circular or spherically shaped distributions of data. Then, for $K = 2$ and an arbitrary d,

$$f_X(\boldsymbol{x}) = \pi_1\,\mathcal{N}_d(\boldsymbol{\mu}_1, \sigma_1^2 \boldsymbol{I}_d) + \pi_2\,\mathcal{N}_d(\boldsymbol{\mu}_2, \sigma_2^2 \boldsymbol{I}_d). \tag{1.8.4}$$

In the case when $K = 2$ and $d = 2$, we have

$$f_X(\boldsymbol{x}) = \pi_1\,\mathcal{N}_2(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + \pi_2\,\mathcal{N}_2(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2). \tag{1.8.5}$$

where $\pi_1 + \pi_2 = 1$, $\boldsymbol{\mu}_1 = (\mu_{11}, \mu_{12})^\top$, $\boldsymbol{\mu}_2 = (\mu_{21}, \mu_{22})^\top$,

$$\boldsymbol{\Sigma}_1 = \sigma_1^2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_2 = \sigma_2^2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

For notational convenience, we can define a set $\Theta = \{\pi_1, \pi_2, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \sigma_1, \sigma_2\}$ that contains all the true parameters. Data can be generated from the probability distribution of this two-dimensional Gaussian mixture in the same manner as described for the one-dimensional case. Based on the data, we can estimate the parameters of the mixture to reconstruct the distribution. When we compute these parameters, we will use $\hat{\Theta} = \{\hat{\pi}_1, \hat{\pi}_2, \hat{\boldsymbol{\mu}}_1, \hat{\boldsymbol{\mu}}_2, \hat{\sigma}_1, \hat{\sigma}_2\}$, where the "hat" represents an estimate of each parameter. This would constitute the **model** of the distribution.

Example: Consider the data in Figure 1.9. It was produced by sampling an unknown Gaussian mixture. We want to express it in the form given in Eq. (1.8.5). How can we do this? Assume we can identify the two clusters and the points belonging to each one using some clustering algorithm. Then we find that the centers of the two clusters are (1.1,1.2) and (6.9,7.1), respectively, and assuming the covariance matrices of the two clusters are symmetric, we can compute the variances as 1.3 and 2.1, respectively. Also, we find that there are twice as many points in one cluster compared to the other. To write a Gaussian mixture model using Eq. (1.8.5), we set $\hat{\pi}_1 = 1/3$ and $\hat{\pi}_2 = 2/3$. The centers of the clusters are

$$\hat{\boldsymbol{\mu}}_1 = (1.1, 1.2)^\top, \quad \text{and} \quad \hat{\boldsymbol{\mu}}_2 = (6.9, 7.1)^\top.$$

Then the covariance matrices are

$$\hat{\boldsymbol{\Sigma}}_1 = 1.3 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \hat{\boldsymbol{\Sigma}}_2 = 2.1 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

It turns out that the true values are $\pi_1 = 0.3$, $\pi_2 = 0.7$, $\boldsymbol{\mu}_1 = (1, 1)^\top$, $\boldsymbol{\mu}_2 = (7, 7)^\top$, $\sigma_1^2 = 1$, and $\sigma_2^2 = 2$. The model is quite accurate in this case and it completes the flow shown in Figure 1.4. The only missing step is how to create the clusters in the first place, which is the subject of the next section. □

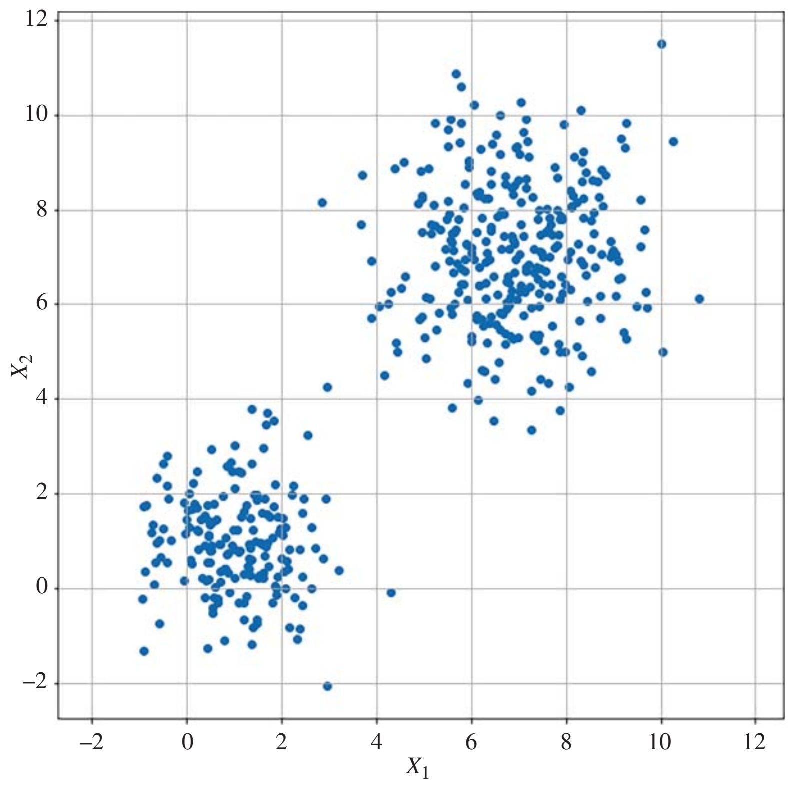


Figure 1.9 Two-dimensional Gaussian mixture.

1.9 Clustering Algorithms

We have already described a Gaussian mixture method of generating data and a generic Lloyd-style iteration loop to create clusters. We now examine two ways to implement Algorithm 1.1. Once we choose a distance measure, a cost function and a centering criteria, the generic algorithm becomes a specific algorithm.

1.9.1 *k*-Means Clustering

The first approach is called *k-means clustering*. In phase 1, points are assigned to their closest center based on an L_2 distance measure. Each distance is computed as follows:

$$\begin{aligned}\text{distance}(i,j) &= \left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\right\|_2^2 \\ &= (x_1^{(i)} - \mu_1^{(j)})^2 + (x_2^{(i)} - \mu_2^{(j)})^2 \dots (x_d^{(i)} - \mu_d^{(j)})^2,\end{aligned}$$

where $\boldsymbol{x}^{(i)}$ is the ith observation of $\boldsymbol{X}$ and $\boldsymbol{\mu}^{(j)}$ is the jth center out of K possible centers. Then each of the $\boldsymbol{x}^{(i)}$, $i = 1, \dots, n$, is assigned to the closest $\boldsymbol{\mu}^{(j)}$, $j = 1, \dots, K$ based on the distance formula. In the second stage, the *centroids* of the clusters are computed using the points assigned to each cluster for all K clusters. This is simply the column-wise mean of the data points assigned to each cluster,

$$\boldsymbol{\mu}^{(j)} = \text{mean}\,(\boldsymbol{C}_j) \;\; \forall j = 1, \dots, K.$$

In the above, $\boldsymbol{C}_j$ is the cluster matrix for cluster j.

The cost is computed as the total sum of all distances between data points inside each cluster to their respective cluster centers for all clusters:

$$\text{cost} = \sum_{j=1}^{K} \sum_{i \in C_j} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\|_2^2. \tag{1.9.1}$$

In the above, C_j is the set of row numbers of $\boldsymbol{X}$.

The old and new costs are compared relative to a small value, ε, to determine convergence as follows:

$$|\text{cost}_{\text{new}} - \text{cost}_{\text{old}}| < \varepsilon. \tag{1.9.2}$$

The two-phase loop continues until the convergence criterion is satisfied and none of the centers change in two consecutive loops.

Exercise: Let $\boldsymbol{x}^{(i)}$ be the ith observation of matrix $\boldsymbol{X} \in \mathbb{R}^{60\times 2}$ given by

$$\boldsymbol{X} = \begin{pmatrix} 1 & 2 \\ 1 & 4 \\ \vdots & \vdots \\ 2 & 3 \end{pmatrix}.$$

Only the 1st, 2nd, and 60th rows of the matrix are shown above. Let cluster C_1 consist of only those three points, specifically $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}$, and $\boldsymbol{x}^{(60)}$. By our notation, $C_1 = \{1, 2, 60\}$. What is the cluster centroid, which we denote by $\boldsymbol{\mu}^{(1)}$?

Ans.: The data points in C_1 are $\boldsymbol{x}^{(1)} = (1, 2)^\top$, $\boldsymbol{x}^{(2)} = (1, 4)^\top$, and $\boldsymbol{x}^{(60)} = (2, 3)^\top$. The centroid is the coordinate-wise average. Therefore, the centroid is $\boldsymbol{\mu}^{(1)} = ((1 + 1 + 2)/3, (2 + 4 + 3)/3)^\top = (4/3, 3)^\top$. □

Exercise: Let $\boldsymbol{X} \in \mathbb{R}^{60\times 2}$ be defined as above. What is the L_2 distance from $\boldsymbol{x}^{(2)}$ to a cluster center $\boldsymbol{\mu}^{(1)} = (4/3, 3)^\top$?

Ans.: The distance is given by

$$\text{distance}(2, 1) = \|\boldsymbol{x}^{(2)} - \boldsymbol{\mu}^{(1)}\|_2^2 = (x_1^{(2)} - \mu_1^{(1)})^2 + (x_2^{(2)} - \mu_2^{(1)})^2.$$

Therefore, $(1 - 4/3)^2 + (4 - 3)^2 = 10/9$. □

Exercise: Let cluster C_1 consist of three elements of $\boldsymbol{X} \in \mathbb{R}^{60\times 2}$ given by $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}$ and $\boldsymbol{x}^{(60)}$ for the matrix given above. What is the L_2 cost of this cluster assuming $\boldsymbol{\mu}^{(1)} = (4/3, 3)^\top$?

Ans.: The L_2 cost is given by

$$\text{cost}(C_1) = \|\boldsymbol{x}^{(1)} - \boldsymbol{\mu}^{(1)}\|_2^2 + \|\boldsymbol{x}^{(2)} - \boldsymbol{\mu}^{(1)}\|_2^2 + \|\boldsymbol{x}^{(60)} - \boldsymbol{\mu}^{(1)}\|_2^2.$$

Therefore, $[(1 - 4/3)^2 + (2 - 3)^2] + [(1 - 4/3)^2 + (4 - 3)^2] + [(2 - 4/3)^2 + (3 - 3)^2] = 24/9$. □

1.9.2 *k*-Medians Clustering

One problem with k-means clustering is that it is not robust to outliers. This issue will be highlighted in sections to follow, but there are many clustering problems that contain outliers and they will disrupt the ability to obtain acceptable results using k-means. A logical alternative is to use *k-medians clustering* which employs different distance, cost, and center calculations. The distance is typically given by the Manhattan distance (although Euclidean distance can also be used) as follows:

$$\begin{aligned}\text{distance}(i,j) &= \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\|_1 \\ &= |x_1^{(i)} - \mu_1^{(j)}| + |x_2^{(i)} - \mu_2^{(j)}| + \ldots + |x_d^{(i)} - \mu_d^{(j)}|. \end{aligned} \tag{1.9.3}$$

This L_1 distance measure is used to determine the assignment of observations to clusters. Although this is typically used in k-medians, the L_2 distance is also suitable for this purpose. The reason is that distance is a relative measure here so either of the two options may be used. What is crucial for robustness is the computation of the centers. After assignment, the centers are subsequently determined as the column-wise medians of the observations in each cluster:

$$\boldsymbol{\mu}^{(j)} = \text{median}(\boldsymbol{C}_j) \ \ \forall j = 1, \ldots, K,$$

where matrix $\boldsymbol{C}_j$ contains the subset of rows of $\boldsymbol{X}$ that belong in cluster j.

The cost is computed as follows:

$$\text{cost} = \sum_{j=1}^{K} \sum_{i \in C_j} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\|_1. \tag{1.9.4}$$

The use of the median for center computation is what robustifies the algorithm. The centers are no longer controlled by outliers and reasonable values of centers that truly represent each cluster will be found using this approach. To summarize, both the cost and distance measures of k-means can be used in the k-medians approach, so long as the centers are computed using the median.

Exercise: Let cluster C_1 consist of 3 elements of $\boldsymbol{X} \in \mathbb{R}^{60\times 2}$ given by $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}$, and $\boldsymbol{x}^{(60)}$ of the matrix

$$\boldsymbol{X} = \begin{pmatrix} 1 & 2 \\ 1 & 4 \\ \vdots & \vdots \\ 100 & 100 \end{pmatrix}.$$

Hence, $C_1 = \{1, 2, 60\}$ in this case. What is the cluster median?

Ans.: The data points in C_1 are $\boldsymbol{x}^{(1)} = (1,2)^\top$, $\boldsymbol{x}^{(2)} = (1,4)^\top$ and $\boldsymbol{x}^{(60)} = (100,100)^\top$. The center is the coordinate-wise median. Therefore, $\boldsymbol{\mu}^{(1)} = (1,4)^\top$. This is about the same as the centroid $(4/3, 3)^\top$ found in an earlier exercise even though this dataset has an outlier at $(100,100)^\top$ which is part of cluster C_1. □

Exercise: Let $\boldsymbol{X} \in \mathbb{R}^{60\times 2}$ be defined as above. What is the Manhattan distance from $\boldsymbol{x}^{(2)}$ to a cluster median $\boldsymbol{\mu}^{(1)} = (1,4)^\top$?

Ans.: The Manhattan distance is given by

$$\text{distance}(2,1) = \|\boldsymbol{x}^{(2)} - \boldsymbol{\mu}^{(1)}\|_1 = |x_1^{(2)} - \mu_1^{(1)}| + |x_2^{(2)} - \mu_2^{(1)}|.$$

Therefore, $|1-1| + |4-4| = 0$. □

Exercise: Let cluster C_1 consist of three elements of $\boldsymbol{X} \in \mathbb{R}^{60\times 2}$ given by $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}$, and $\boldsymbol{x}^{(60)}$ of the matrix given above. What is the L_1 cost of this cluster assuming $\boldsymbol{\mu}^{(1)} = (1, 4)^\top$?

Ans.: The L_1 cost is given by

$$\text{cost} = \|\boldsymbol{x}^{(1)} - \boldsymbol{\mu}^{(1)}\|_1 + \|\boldsymbol{x}^{(2)} - \boldsymbol{\mu}^{(1)}\|_1 + \|\boldsymbol{x}^{(50)} - \boldsymbol{\mu}^{(1)}\|_1.$$

Therefore, $[|1 - 1| + |2 - 4|] + [|1 - 1| + |4 - 4|] + [|100 - 1| + |100 - 4|] = 2 + 0 + [99 + 96] = 197.$ □

1.10 Importance of Robust Clustering

With the data generation and clustering algorithms in place, it is time to compare the two approaches on synthetic datasets. Initially, the dataset will not have any outliers to assess how each method performs on outlier-free datasets. Then, outliers will be inserted to examine the consequences of their presence on each method.

1.10.1 Clustering with No Outliers

Let us return to the simple flow shown in Figure 1.4 and follow a complete sequence around the loop. Consider a Gaussian mixture in $\mathbb{R}^2$ given by

$$f_\Theta(x) = 0.6\,\mathcal{N}_2((-2, 1)^\top, 3\boldsymbol{I}_2) + 0.4\,\mathcal{N}_2((5, 0)^\top, 4\boldsymbol{I}_2), \tag{1.10.1}$$

which is the ground truth in this example. We draw a sample from this distribution to create the dataset, as prescribed by the upper portion of the loop in Figure 1.4. The resulting dataset composed of a total of 60 points is shown in Figure 1.10 and it does not have any outliers. Although not immediately apparent, there are two clusters in the scatter plot since the Gaussian mixture has two terms.

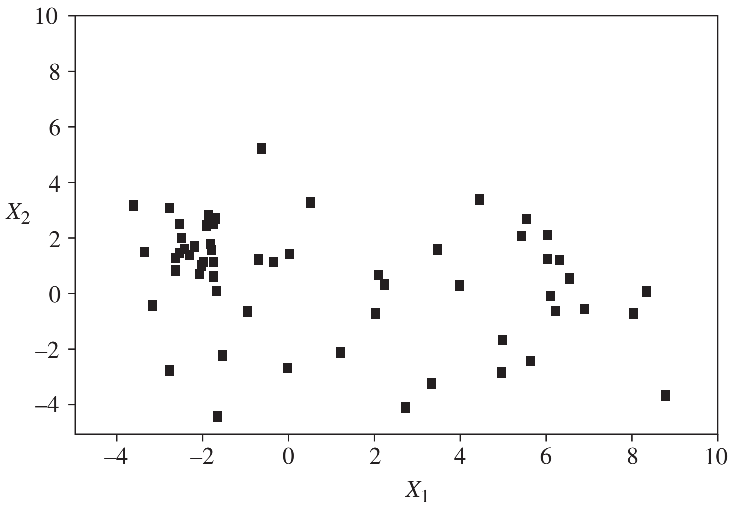


Figure 1.10 Dataset before clustering.

In practice, we are given the data but no access to the ground truth. Assume all we know is that the hyperparameter is fixed at $K = 2$. The question is, can we reconstruct the Gaussian mixture using only the given data to close the loop in Figure 1.4? The answer is yes. We can do this using a clustering algorithm followed by the estimation of the parameters of Θ.

The k-means clustering technique can be invoked using `KMeans` in the `sklearn` Python library. The k-medians approach is not available in `sklearn` so a simple version can be written in Python for comparison purposes.[4]

If we run the `KMeans` algorithm to convergence, the resulting cluster centers are

$$\hat{\boldsymbol{\mu}}^{(1)} = (-1.92, 1.17)^\top \quad \text{and} \quad \hat{\boldsymbol{\mu}}^{(2)} = (4.93, -0.17)^\top.$$

These values are close to the cluster centers associated with the original distribution of Eq. (1.10.1). If we run the k-medians algorithm, we obtain

$$\hat{\boldsymbol{\mu}}^{(1)} = (-1.89, 1.46)^\top \quad \text{and} \quad \hat{\boldsymbol{\mu}}^{(2)} = (5.48, 0.30)^\top.$$

These centers are also close to the original distribution values.

The results are shown graphically in Figure 1.11. The k-means and k-medians methods produce roughly the same two clusters and centers (indicated by $\otimes$ in the figure). Both sets of results are acceptable as they are close to the two centers of the original distribution. To close the loop of Figure 1.4, we need the variances of each cluster by computing the covariance matrix of the form $\sigma^2 \boldsymbol{I}_2$, and also the probability weights based on the ratio of points in each cluster relative to the total number of points. In doing so, using k-means, we obtain:

$$f_{\hat{\Theta}}(x) = 0.6\mathcal{N}_2((-1.9, 1.0)^\top, 2.4\boldsymbol{I}_2) + 0.4\mathcal{N}_2((4.9, -0.2)^\top, 3.8\boldsymbol{I}_2) \tag{1.10.2}$$

and with k-medians, we obtain:

$$f_{\hat{\Theta}}(x) = 0.62\mathcal{N}_2((-1.9, 1.5)^\top, 2.5\boldsymbol{I}_2) + 0.38\mathcal{N}_2((5.5, 0.3)^\top, 3.7\boldsymbol{I}_2) \tag{1.10.3}$$

Both models are reasonably accurate. In practice, we will not be able to compare our model to the ground truth. This is as far as we can go with the given data but we have no way to determine if the results are acceptable. It will be hard to prove our model is correct. But since this data is outlier-free and we know $K = 2$, we can have a high degree of confidence in the correctness of the model in this case.

1.10.2 Clustering with Outliers

So far, we have found k-means and k-medians to be about the same for the given outlier-free dataset. What if one of the points was inadvertently entered manually as $(-400, -400)$ instead of the correct value of $(-4.00, -4.00)$? This outlier along with the original data are shown in Figure 1.12. The outlier is due to human error and we want to investigate the effect on the two competing clustering methods. Note that the outlier would be easy to detect and remove in a two-dimensional setting, but in a higher-dimensional setting, where visualization is not possible, it would be more difficult.

The initial question is: how well do the two methods perform in reproducing the original distribution from which the data arose before noise is added. Referring back to Figure 1.5, the noise is the outlier

4 The k-medians code in Python is available in one of the projects at the end of this chapter.

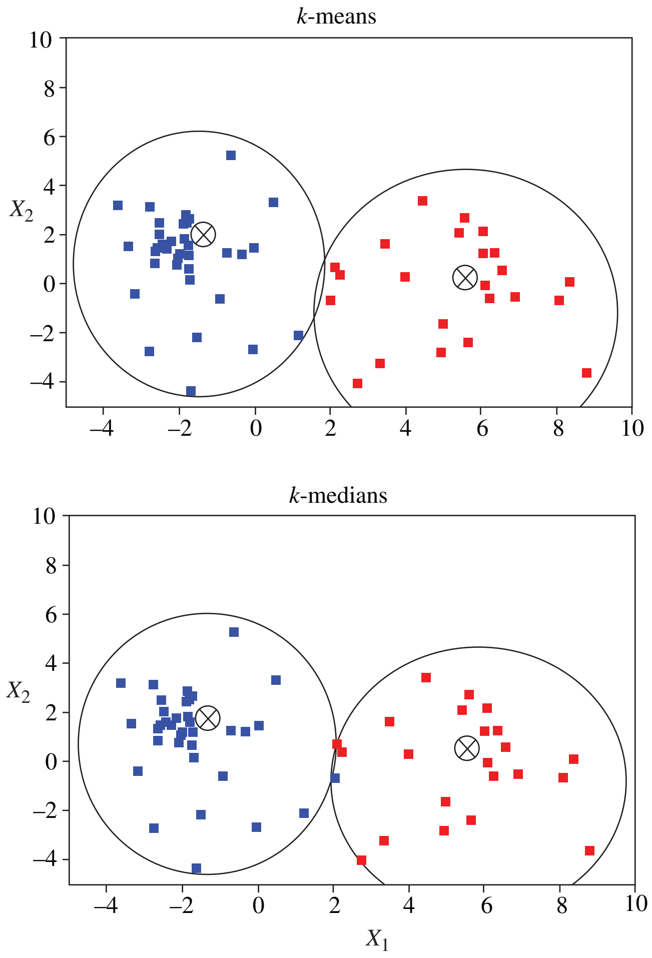


Figure 1.11 *k*-means and *k*-medians clustering.

introduced due to a typographical error. The results of k-means are shown graphically in the upper panel of Figure 1.13. One large cluster is created with all the data and another cluster is created containing only the outlier. The outlier cluster is not shown in Figure 1.13 at the scale of the plot but it is clear that one of the cluster centers is forced outside of the region of the data by the k-means algorithm because only one is visible in the diagram. The two centers are

$$\hat{\boldsymbol{\mu}}^{(1)} = (-400, -400)^\top \quad \text{and} \quad \hat{\boldsymbol{\mu}}^{(2)} = (1.10, 0.6)^\top.$$

The first cluster has a center that is well outside the cluster region located at $(-400, -400)$. The other center is located in the middle of the true data. It is evident that even a single outlier in the dataset can significantly impact k-means clustering. We saw this effect in an earlier section regarding the mean vs. the median and we see it here again in the clustering problem.

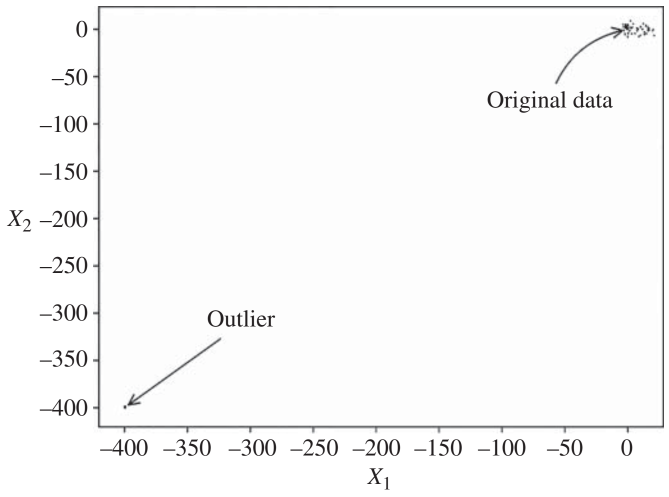


Figure 1.12 Data with 1 outlier at (−400, −400).

Running the k-medians algorithm produces the following centers:

$$\hat{\boldsymbol{\mu}}^{(1)} = (-1.89, 1.33)^\top \quad \hat{\boldsymbol{\mu}}^{(2)} = (5.48, 0.30)^\top.$$

Unlike the k-means case, the centers have not changed very much. This is highly desirable because it indicates that the outlier has not affected the outcome of clustering. This is shown graphically in lower panel of Figure 1.13 where the same two clusters are produced. The k-medians results are relatively stable compared to k-means. This is a key point to emphasize: using a robust method produces superior results in the presence of outliers and similar results with no outliers.

1.10.3 Detection and Removal of Outliers

One way to produce better results using k-means is to detect and remove outliers. This requires some techniques that are themselves robust in nature since the non-robust methods fail to address the issue. Consider the dataset with 1 outlier as described in the previous section. We must first run k-medians clustering since the results from k-means results are not meaningful. More generally, in order to find outliers, a robust method is always part of the methodology.

Once we have the results of robust clustering, we can apply simple rules to remove outliers. One approach is to use the "3-sigma edit" rule. This rule is used to detect and delete any points that are greater than $3\sigma_j$ away from the center of cluster C_j, where $j = 1, \dots, K$. This rule uses the criteria that, if point $\boldsymbol{x}^{(i)}$ is in matrix $\boldsymbol{C}_j$,

$$\text{delete}\, \boldsymbol{x}^{(i)} \quad \text{if } \|\boldsymbol{x}^{(i)} - \hat{\mu}^{(j)}\|_2 > 3\hat{\sigma}_j \quad \text{for } i = 1, \dots, n,$$

where $\hat{\mu}^{(j)}$ and $\hat{\sigma}_j$ are the mean and standard deviation of the points in cluster j. Unfortunately, the computation of the cluster mean and standard deviation will be skewed by outliers so they cannot be trusted.

A better approach is to use a robust form of the 3-sigma edit rule. Since the mean and variance cannot be computed reliably under outlier conditions, robust statistics can be used in their place. Data will only

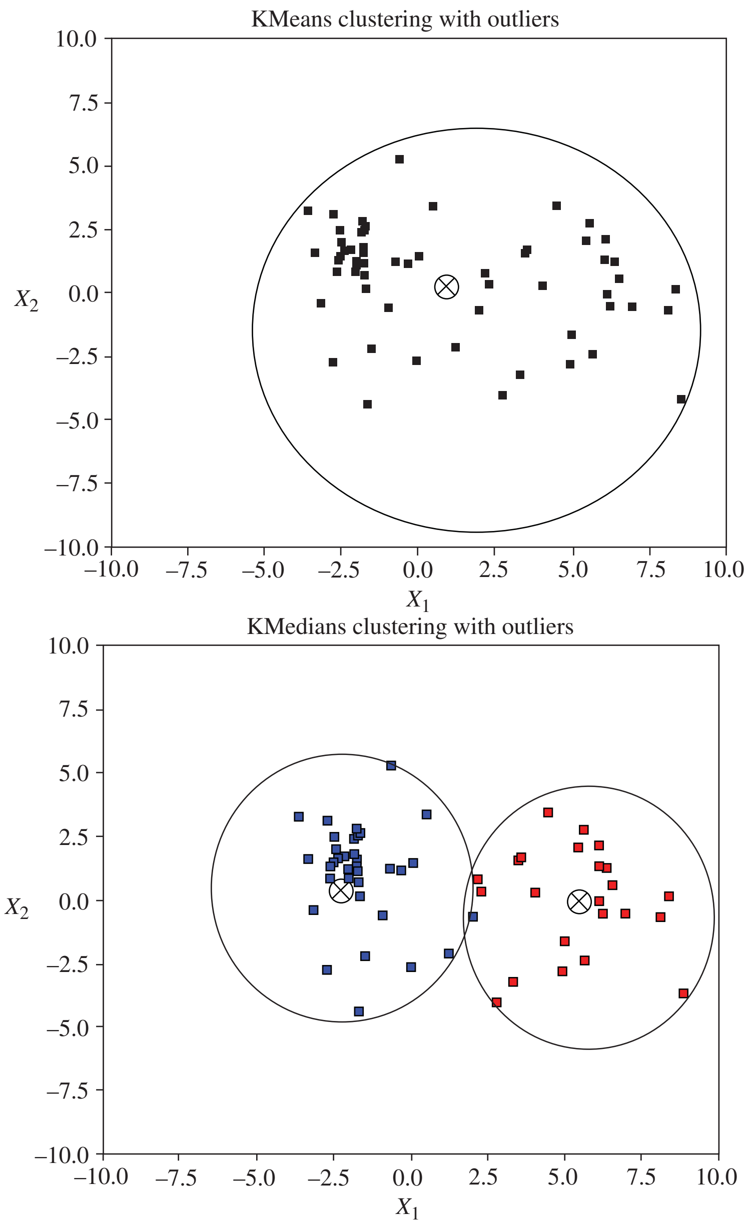


Figure 1.13 *k*-means and *k*-medians clustering with 1 outlier at (−400, −400) which is not shown as it is outside the limits of the plots.

be removed if any $\boldsymbol{x}^{(i)}$ belonging in cluster j satisfies the "3-MADN edit" rule. That is, for all points $\boldsymbol{x}^{(i)}$ in matrix $\boldsymbol{C}_j$,

$$\text{delete}\, \boldsymbol{x}^{(i)} \quad \text{if } \|\boldsymbol{x}^{(i)} - \text{median}\,(\boldsymbol{C}_j)\|_2 > 3\text{MADN}(\boldsymbol{C}_j) \quad \text{for } i = 1, \ldots, n.$$

This robust version will find and delete all the outliers, regardless of their number and location outside the cluster region.

To compare the two methods, first consider the standard 3-sigma edit rule for the previous dataset with only one outlier. If we use a Gaussian distribution for each cluster, then we find that $\hat{\sigma}_2 = 64.8$ which is too large to be of any value. The 3-sigma edit rule would force us to reject points that are $3 \times 64.8 = 194.4$ units in all directions from the center. It will not delete any points and therefore the 3-sigma statistic is not useful. Contrast this with the robust criteria. The estimate of $\text{MADN}(\boldsymbol{C}_2) = 11.2$ and so the 3-MADN edit rule would yield $3 \times 11.2 = 33.6$. This is a much more reasonable distance to use as a threshold for identifying and deleting outliers, especially since the median is used as the cluster center. This would delete the single outlier in the data. Once this is done, the mean and variance computations will be reasonably accurate. Furthermore, the `KMeans` tool can be used to build the model after the outlier is removed.

Exercise: Given the one-dimensional dataset $\{1, -2, 5, -4, -6, 3\}$ and $K = 2$, compute the centers using k-means. Next, recompute the centers using k-medians.

Ans.: k-means $\{-4, 3\}$. k-medians $\{-4, 3\}$. □

Exercise: Given the one-dimensional dataset $\{1, -2, 5, -4, -66, 3\}$ and $K = 2$, compute the centers using k-means. Next, recompute the centers using k-medians.

Ans.: k-means $\{-24, 3\}$. k-medians $\{-4, 3\}$. □

The 3-MADN edit rule is somewhat arbitrary and may be too strict for most datasets. In practice, we may need to adjust the rule to avoid declaring inliers as outliers. For example, a 10-MADN edit rule may be better to ensure that true outliers are found far from the centers of the clusters. Some tuning may be needed to find the proper multiplier. On the other hand, the 3-sigma rule would fail because the value of $\hat{\sigma}$ would be unreliable. The key point here is that outlier detection requires the use of robust procedures and robust statistics.

1.11 Summary

This chapter highlighted the use of the median as a key to robust methods. It provided a case study of how to robustify an existing algorithm in machine learning and how to detect and remove outliers. It is clear that the k-medians approach is robust to outliers and superior to k-means in this regard. Using k-medians avoids having to find and remove outliers or be concerned with centers that do not represent the data well enough for use in other applications. The k-means method can be safely used only if there are no outliers in the data. Unfortunately, this assumption is not a realistic for most datasets. Another key point is that the myth that "having lots of data in the dataset tends to reduce the effect of outliers" is dispelled here. Any of the clusters can be affected by just one outlier even if there are many points in the dataset. It is more about the location of the outliers and their effect on cluster centers rather than the number of outliers. Another key point is that robust methods are needed in the detection of outliers and anomalies. This will be a recurring theme throughout the book as we progress through different algorithms that require robustification.

In the rest of this book, we address machine learning through the lens of outliers and anomalies, as opposed to the traditional view which tends to ignore or mishandle them. The initial machine learning example of clustering given in this chapter serves as a template for how we will be covering other methods in this book. However, our focus will shift to the supervised learning tasks of linear regression, logistic regression, and neural networks. The same issues will arise in all these areas, so the techniques themselves will have to be modified to robustify the procedures.

Problems

To complete the projects in this book, you will need a Python programming environment for machine learning. Jupyter notebooks are recommended for running the provided code segments, but you can also use PyCharm or any other Integrated Development Environment (IDE) of your choice. You should set yourself up accordingly at this stage to get the most out of the book.

1.1 a) For the set $S = \{1, 2, 3, 4, 5, 100, 101, 102\}$, find the median and the mean. How many more values above 100 would have to be added to the set before the median is ≥ 100?

b) For the set $S = \{1, 2, 3, 4, 5, 100, 101, 102\}$, compute the standard deviation, SD($\boldsymbol{x}$), and the normalized median absolute deviation, MADN($\boldsymbol{x}$). Assume that we want to remove outliers. Use the 3-sigma and 3-MADN edit rules. Which approach is better at removing the outliers?

c) For the matrix given below

$$\boldsymbol{X} = \begin{pmatrix} 5 & 2 & 7 \\ 1 & 9 & 2 \\ 2 & 1 & 1 \end{pmatrix},$$

compute the column-wise median and mean.

1.2 a) For the vector quantity $\boldsymbol{u} = (-1, -2, 3, 4, 5)^{\top}$, compute the L_1- and L_2-norms. Which norm is larger? Will this always be true? If not, construct a vector to demonstrate your answer.

b) For the same vector, compute the squared L_2-norm. Again, compare the L_1-norm against the squared L_2-norm. Which is larger and will this always be the case? Provide a counterexample if not true.

c) Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be two vectors in $\mathbb{R}^d$. How likely is it that $||\boldsymbol{v}||_1 \leq ||\boldsymbol{v}||_2^2$? Can you find a vector $\boldsymbol{u}$ such that $||\boldsymbol{u}||_1 > ||\boldsymbol{u}||_2^2$? If so, state the vector. If not, explain why.

1.3 A data matrix $\boldsymbol{X} \in \mathbb{R}^{7\times 2}$ is clustered into two groups as follows:

$$\boldsymbol{C}_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 2 & 1 \\ -8 & -8 \end{pmatrix}, \quad \boldsymbol{C}_2 = \begin{pmatrix} 5 & 1 \\ 6 & 1 \\ 6 & 2 \end{pmatrix}$$

a) For k-means, compute the cluster means and total clustering cost.
b) For k-medians, compute the cluster medians and total clustering cost.
c) Plot all the points from both clusters on two graphs, one for k-means and one for k-medians. Indicate the means and medians with $\otimes$. Comment on the results in terms of the tolerance of the two methods with respect to the outlier data $(-8, -8)$.
d) Compute the covariance matrix, $\sigma^2 \boldsymbol{I}$ of each cluster. This requires only the estimation of σ^2 for each cluster. Use the mean values computed above. For σ_1^2 in cluster 1, compute matrix $\boldsymbol{C}_1 - \boldsymbol{\mu}_1$, then average the sum the squares of all terms. In Python, `sigma12 = np.mean((C1-mu1)**2)` would produce the estimate. The same procedure can be used for cluster 2.
e) Estimate π_1 and π_2 using a ratio of the number of points in a cluster to the total number of points in the dataset. Write an expression for the model of the Gaussian mixture. Is this a good model?
f) How would you use the median and MADN to build a better model? Write the expression for the better model.

1.4 (PROJECT) In this project, the objective is to create a dataset for clustering using Gaussian mixtures.
a) Import the needed libraries.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from sklearn.cluster import KMeans
```

b) Create a dataset for the Gaussian mixture given by

$$f_X(x) = 0.6\,\mathcal{N}_2((-2,1)^\top, 3\boldsymbol{I}_2) + 0.4\,\mathcal{N}_2((5,0)^\top, 4\boldsymbol{I}_2).$$

The data can be generated and plotted with the following code:

```
np.random.seed(8)
mu = np.array([[-2,1],[5,0]])
sigma = np.array([[[3,0],[0,3]],[[4,0],[0,4]]])
Y1 =  np.random.multivariate_normal(mu[0], \
    sigma[0], size=60)
Y2 =  np.random.multivariate_normal(mu[1], \
    sigma[1], size=40)
X = np.vstack((Y1,Y2))
plt.xlim([-10,10])
plt.ylim([-10,10])
plt.scatter(X[:,0],X[:,1], \
    marker="$.$",color="black")
plt.show()
```

1.5 (PROJECT) The goal of this project is to compare `KMeans` against `KMedians` using the same dataset as in the project above.

a) Run the `KMeans` program in the `sklearn.cluster` library for the dataset *without any outliers* using:

```
kmeans = KMeans(n_clusters=2, init="k-means++", \
        random_state=1).fit(X)
labels = kmeans.labels_
means = kmeans.cluster_centers_
print("Centers from KMeans")
print(means)
```

b) The Python code given below, called `KMedians`, generates robust cluster centers given $\boldsymbol{X}$ and the number of clusters, K. Enter and run it in your Python notebook.

```
# Assign points to clusters
def assign_clusters(X, M):
    distances = np.sum(np.abs(X[:,  \
        np.newaxis] - M), axis=2)
    clusters = np.argmin(distances, axis=1)
    return clusters

# Find the cluster centers
def calculate_cluster_medians(X, clusters, \
        num_clusters):
    cluster_medians = np.zeros((num_clusters, \
        X.shape[1]))
    for i in range(num_clusters):
        cluster_rows = X[clusters == i]
        if len(cluster_rows) > 0:
            cluster_medians[i] = \
                np.median(cluster_rows, axis=0)
    return cluster_medians

# Compute the cost of this arrangement
def calculate_total_cost(X, clusters, \
    cluster_medians):
    total_cost = 0
    for i in range(len(X)):
        cluster_median = \
        cluster_medians[clusters[i]]
        total_cost += np.sum(np.abs(X[i] \
            - cluster_median))
    return total_cost

# Plot 2D cluster arrangement
def plot_clusters(X, clusters, cluster_medians):
    plt.figure(figsize=(10, 6))
    num_clusters = cluster_medians.shape[0]
    # Plot each cluster in a different color
    for i in range(num_clusters):
        cluster_points = X[clusters == i]
        plt.scatter(cluster_points[:, 0], \
                    cluster_points[:, 1], \
                    label=f'cluster i}')
    # Plot the cluster medians
    plt.scatter(cluster_medians[:, 0], \
                cluster_medians[:, 1], \
                color='black', marker='x', \
                s=100, label='cluster medians')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('clusters and their medians')
    plt.legend()
    plt.show()

# Perform k-medians clustering
def KMedians(X,K):
    seed = 0
    np.random.seed(seed)
    n, d = X.shape
    M = X[np.random.choice(n, K, replace=False)]
    prev_cost = None
    new_cost = None
    #Use Lloyd-style iteration
    while (prev_cost is None or \
        np.abs(prev_cost - new_cost) > 1e-4):
        clusters = assign_clusters(X, M)
        num_clusters = M.shape[0]
        M = calculate_cluster_medians(X, \
        clusters, num_clusters)
```

```
            prev_cost = new_cost
            new_cost = calculate_total_cost(X, \
            clusters, M)
        return clusters,new_cost, M
```

c) Run KMedians using the dataset ***X*** generated previously and plot the results. How do the two methods compare in terms of computing the centers of the clusters?

```
K = 2
cluster, cost, medians = KMedians(X,K)
print("Centers from KMedians")
print(medians)
plot_clusters(X, cluster, medians)
```

d) Use the code below to plot the results of the `KMeans` and `KMedians` programs next to each other as subplots.

```
X1md = np.array(X[cluster == 1])
X2md = np.array(X[cluster == 0])
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].scatter(X[:, 0], X[:, 1], c=labels, \
            cmap="plasma", marker="$.$", \
            edgecolors="k", s=50)
axes[0].set_title("KMeans Clustering")
axes[0].set_xlabel("Feature 1")
axes[0].set_ylabel("Feature 2")
axes[0].scatter(means[0,0],means[0,1], \
    marker="x",color="red")
axes[0].scatter(means[1,0],means[1,1], \
    marker="x",color="red")
axes[0].set_xlim([-10,10])
axes[0].set_ylim([-10,10])
axes[1].scatter(X1md[:, 0], X1md[:, 1], c="yellow",
        marker="$.$", edgecolors="k", s=50)
axes[1].scatter(X2md[:, 0], X2md[:, 1], c="black",
        marker="$.$",edgecolors="k", s=50)
axes[1].set_title("KMedians Clustering")
axes[1].set_xlabel("Feature 1")
axes[1].set_ylabel("Feature 2")
plt.scatter(medians[0,0],medians[0,1], \
    marker="x",color="red")
plt.scatter(medians[1,0],medians[1,1], \
    marker="x",color="red")
axes[1].set_xlim([-10,10])
```

```
axes[1].set_ylim([-10,10])

plt.tight_layout()
plt.show()
```

What do you notice about the cluster centers in each case?

e) Next, insert outliers using the code below and then rerun both methods. For `KMeans`, use

```
X[1] = np.array([-400,-400])
X[2] = np.array([-400,0])
kmeans = KMeans(n_clusters=2,  init="k-means++", \
        random_state=1).fit(X)
labels = kmeans.labels_
means = kmeans.cluster_centers_
print("New centers from KMeans")
print(means)
```

For `KMedians`, use

```
cluster, cost, medians = KMedians(X,K)
print("New centers from KMedians")
print(medians)
```

Compare `KMedians` and `KMeans` on the dataset *with outliers*.

f) Use the same plotting code as in part (c) to plot the `KMeans` and `KMedians` results again showing the clusters in each case after outliers are inserted. How many cluster centers can you find in each case?

1.6 (PROJECT) In this project, the objective is to detect outliers in the dataset from the previous project. To do this, we need to use the `KMedians` results and the 10-MADN edit rule. Which outliers are found using the code below?

```
# Outlier detection method using MADN
X1md = np.array(X[cluster == 1])
X2md = np.array(X[cluster == 0])
def MADN(r):
    return(1.4826*np.median(np.abs(r-np.median(r))))
dist1 = (np.abs((X1md - medians[0,:]))).sum(axis=1)
mad1 = MADN(dist1)
dist2 = (np.abs((X2md - medians[1,:]))).sum(axis=1)
mad2 = MADN(dist2)
for i in range(X1md.shape[0]):
    if (np.sum(np.abs(X1md[i, :] - medians[0,:])) \
        > 10*mad1):
        print("Outlier found at",X1md[i,:]," \
                in Cluster 1")
    for i in range(X2md.shape[0]):
        if (np.sum(np.abs(X2md[i, :] - medians[1,:])) \
            > 10*mad2):
            print("Outlier found at",X2md[i,:]," \
                in Cluster 2")
```

References

James, G., Witten, D., Hastie, T. et al. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer.

Maronna, R.A., Douglas Martin, R., Yohai, V.J., and Salibian-Barrera, M. (2019). *Robust Statistics, Theory and Methods (with R)*. Wiley.

2

Robust Linear Regression

2.1 Introduction

Supervised learning is the most common form of machine learning (ML) and will be covered extensively in the rest of the book. As linear regression is the simplest form of supervised learning, it serves as a launching point to understand this category of ML in a familiar setting. The subject of linear regression has been thoroughly studied over many decades and is widely used in many applications in data science. In fact, it predates ML itself. Our goal here is to address linear regression from the perspective of **outliers** in the data, covering the essentials needed for our study of robust methods.

In this and the next chapter, we will be describing a number of different estimators for robust linear regression. In this context, we need to understand how to derive these estimators and provide mechanisms to compare them against one another. Several different criteria will be presented to assess which one performs best and under what conditions. The chapter begins with a brief tour of the techniques and issues associated with linear regression. Then we will make the case for robust regression using Huber's method and identify certain shortcomings of the approach. This will lead to an alternative approach to be presented in Chapter 3. Once the basics of robust linear regression are understood, it can be applied to more advanced ML methods such as quantile regression, logistic regression, and neural networks in subsequent chapters.

2.2 Supervised Learning

Supervised learning takes on a variety of different forms because there are a wide variety of suitable applications for machine learning. Typical applications include predicting the value of a home in a certain region of the country given the housing data associated with that region; or deciding who should get a loan based on the loan history of existing and previous customers; or recognizing handwritten digits from 0 to 9 based on a large dataset of labeled digits. The applications are endless and the techniques used in supervised machine learning are, in turn, quite diverse.

The basic idea in supervised learning is to use a set of data to build a model with predictive power. This is the value proposition of machine learning: take a training set, possibly with noise, define a suitable

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.
© 2025 John Wiley & Sons, Inc. Published 2025 by John Wiley & Sons, Inc.
Companion Website: www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

model, estimate the model parameters (i.e. train the model), and use this parameterized model to make predictions on test data with (hopefully) a high degree of accuracy. Since we do not know the true value of the parameters, the best we can do is estimate them using some well-grounded mathematical procedures and the given data. This leads to the development of suitable estimators, the functions that will allow us to carry out the estimation.

Estimates are the numerical values of the parameters produced by an estimator. Naturally, we want to find the best parameter values that fit the data closely so that the model is able to make good predictions. But how do we know if it is a good model if outliers are present? We need to check the accuracy using an unseen test set with a known ground-truth to validate the whole process. The overall goal in this book is to develop **robust models for both regression and classification** such that outliers in the dataset do not adversely affect the prediction results and we want to eventually apply these methods to neural networks (see Chapters 9 and 10). However, understanding the principles of neural networks begins with an in-depth study of linear regression and logistic regression. The study of advanced ML methods is built on top of these fundamentals, so it is important to fully understand these topics on our journey toward robust ML.

2.3 Linear Regression

Assume we are given a pair of vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$, and asked to fit a line to the data by computing its slope and intercept based on the data. For n observations, we usually write $\boldsymbol{x} \in \mathbb{R}^n$ to denote a real vector with n elements associated with the explanatory variable, and $\boldsymbol{y} \in \mathbb{R}^n$ to denote the vector associated with the response variable. The data is composed of a set of (x_i, y_i) pairs, with $i = 1, \dots, n$. A suitable model for the data in this case is a straight line. We could try to fit a more complex curve to the data but assume we have *a priori* knowledge that there is a linear relationship between the $\boldsymbol{x}$ and $\boldsymbol{y}$ values so we need to fit a line to the data. Therefore, the objective in linear regression (see Montgomery et al. 2012) is to estimate the slope and intercept of the line that best fits the data.

Figure 2.1 shows a scatter plot of the points with the two parameters of interest labeled on the line. The true line (which is unknown to us) is given by a well-known equation,

$$y = \beta_0 + \beta_1 x, \tag{2.3.1}$$

where β_0 is the intercept and β_1 is the slope. Unfortunately, it is not possible to find the exact values of these two parameters unless we have access to the entire population. The best we can do is estimate β_0 and β_1 from a given sample of the population. This is the purpose of linear regression.

An **estimator** tries to minimize a loss function to produce an optimal set of estimates of the parameters. This is usually carried out using a numerical optimization method such as gradient descent. A **loss function** is essentially a cost function that evaluates the total error associated with specific values for the slope and intercept. Of course, if we could somehow compute the exact parameter values, they would not be considered as estimates. But since we will only have access to a finite amount of data (i.e. a finite sample set), the parameter values we produce must be viewed as **estimates**. Once we estimate the two parameters, now denoted as $\hat{\beta}_0$ and $\hat{\beta}_1$ to distinguish them from the true values, we can use

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x \tag{2.3.2}$$

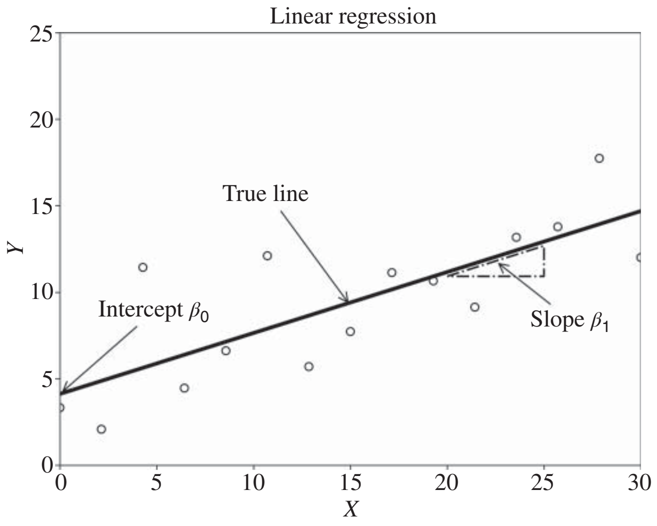


Figure 2.1 Slope and intercept parameters of a line.

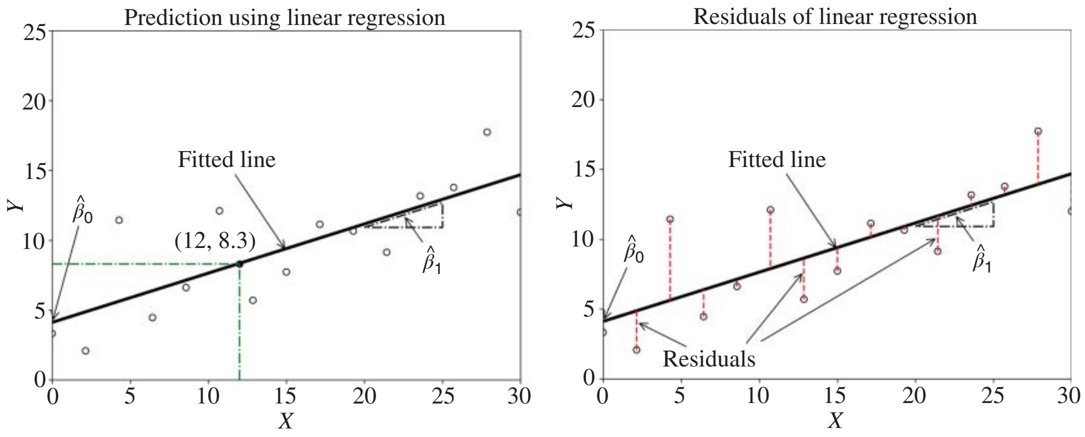


Figure 2.2 Prediction and residuals associated with linear regression.

to predict new values. These two estimated parameters are shown on the left panel of Figure 2.2, along with the fitted line itself. Prediction using the estimates is rather straightforward. For example, it is possible to predict the response y at $x = 12$, which is not represented in the original data. In this case, the regression line predicts that $\hat{y} = 8.3$ when $x = 12$. Note that the line shown in the figure is but one of the many possible lines that "fit" the given data, and we will explore many different ways to estimate the slope and intercept parameters. Simple linear regression involves only one explanatory variable (also called feature or input) and one response variable (or output). Multiple linear regression has many explanatory variables but usually one response variable.

An important point to mention here is that estimates inherently have some amount of uncertainty associated with them. More specifically, they have a variance which is a measure of the uncertainty around the estimates. Different estimators have different variances, and we seek ones that have small variances. Further, there are many ways to estimate parameters, so we should not view any one estimator as the gold standard but rather as one of many options for the application at hand. There are many trade-offs to consider, and variance is just one of the factors. Another factor is robustness to outliers. The well-known ordinary least squares (OLS) method seeks the mean of the data to obtain the estimates. However, it is not robust. Methods that seek the median of the data are robust since they are tolerant to outliers. The goal of this chapter is to illustrate why the median is so important in robust regression.

Notation: Estimates always use the "hat" notation from statistics. For example, if β_0 is the true intercept, then $\hat{\beta}_0$ is the estimate of the intercept. If β_1 is the true slope, then $\hat{\beta}_1$ is the estimate of the slope. Similarly $\hat{y}$ is the predicted response using the parameter estimates. For a multidimensional problem, we use a bold version, i.e. $\boldsymbol{\beta}$ or $\hat{\boldsymbol{\beta}}$ as the case may be. Sometimes θ will be used as a scalar parameter of interest in one-dimensional problems, such as the location problem, and $\hat{\theta}$ is the estimate.

2.4 Importance of Residuals

In this section, we discuss the errors made by a model relative to the original data used to build the model. These errors can be computed after using regression and are called **residuals**. They play an important role in defining loss functions, selecting a loss function based on their probability distribution, and in outlier detection. Therefore, we will spend some time to understand what they are and why they are important.

2.4.1 Defining Errors and Residuals

Observe that, on the right panel of Figure 2.2, the regression line does not necessarily go through all the points or any of the points in the data. This is typical of the results of linear regression. The regression line is actually a summary of the data so that prediction can be performed. There is a vertical distance between the original points and the line. The vertical differences are referred to as *residuals* and their distribution plays a key role in defining the loss function, as we shall see shortly. Figure 2.2 on the right panel highlights the set of residuals for this dataset as vertical bars between the regression line and the data points.

If we include errors in the equation given earlier for a line, it can be rewritten to accurately reflect the relationship between the data and the line. This gives rise to the simple linear model, which takes the form

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x} + \varepsilon, \tag{2.4.1}$$

where ε represents the errors between the **true line** and the data points. These errors arise due to a variety of sources such as effects unaccounted for in our model or random noise in the system. There is valuable information within the errors and their distribution that will lead us to the right type of estimator.

The above equation can also be written in matrix form as

$$\boldsymbol{y} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}, \tag{2.4.2}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$ and $\boldsymbol{X}$ is now a matrix with 1's inserted in the first column. The equation for the error term is

$$\boldsymbol{\varepsilon} = \boldsymbol{y} - \boldsymbol{X\beta}. \tag{2.4.3}$$

Exercise: A dataset has $\boldsymbol{x} = (1, 2, 3, 4, 5)^\top$ and $\boldsymbol{y} = (1.1, 1.9, 3.4, 4.1, 4.8)^\top$. Write the matrix equation in the form of Eq. (2.4.2).

Ans.: The equation takes the form:

$$\begin{bmatrix} 1.1 \\ 1.9 \\ 3.4 \\ 4.1 \\ 4.8 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \end{bmatrix}.$$

The 1's in the first column of the $\boldsymbol{X}$ matrix are needed to incorporate the intercept parameter, β_0. □

In the exercise above, the two parameters should be considered the true values of β_0 and β_1. Unfortunately, we do not know them so we have no way to obtain the true errors. However, we can compute the residuals using the estimates of the parameters and use them as a proxy for the true errors. The residuals are given by

$$\boldsymbol{r} = \boldsymbol{y} - (\hat{\beta}_0 + \hat{\beta}_1 \boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}. \tag{2.4.4}$$

Again, the true errors are based on the true parameter values while the residuals are based on the estimated parameter values.

2.4.2 Residuals in Loss Functions

Our goal in estimation is to minimize the *sum of terms that are a function of the residuals* to obtain $\hat{\boldsymbol{\beta}}$. This can be done in a variety of different ways but they will all require the use of the residuals. In addition, we need to assume that the residuals are distributed in a certain way so that we can apply suitable techniques to estimate the slope and the intercept. *Making assumptions about the error distribution will, in turn, define the loss function.* This is an important point. When we choose a loss function, we are implicitly requiring that the residuals obey a particular distribution associated with that loss function. After regression, the computed residuals may be used to check if our assumptions about their distribution are correct.

Two important loss functions that arise from well-known distributions are the L_1 and L_2 loss functions. We saw how the L_1- and L_2-norms were used in Chapter 1 and they will be the two competing loss functions for simple linear regression in this chapter. First, we define the ith residual as, $r_i = y_i - (\hat{\beta}_0 + x_i\hat{\beta}_1)$, for $i = 1, \dots, n$. We will frequently use the more general form $r_i = y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}$ for convenience. Then, the L_1 loss function is the sum of the absolute values of the residuals, given by

$$L_1 \text{ loss} = \sum_{i=1}^{n} |r_i|, \tag{2.4.5}$$

while the L_2 loss function is half the sum of the squares of the residuals, given by

$$L_2 \text{ loss} = \frac{1}{2}\sum_{i=1}^{n} r_i^2. \tag{2.4.6}$$

Minimizing the L_1 loss is commonly referred to as least absolute deviation (LAD) while minimizing the L_2 loss is called least squares estimation (LSE). These loss functions are minimized to obtain their respective estimates of β_0 and β_1 which may be quite different. We will derive both functions and explore their differences shortly.

Exercise: Assume that we know the true values of the regression parameters to be $\beta_0 = 3$ and $\beta_1 = 1$. Using LSE, we find that $\hat{\beta}_0 = 3.2$ and $\hat{\beta}_1 = 1.1$. Compute the residuals and errors (the two are different) assuming that $\boldsymbol{y} = (3.3, 4.1, 5.5)^\top$ and $\boldsymbol{x} = (0, 1, 2)^\top$.

Ans.: The residuals are given by $r_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$: $r_1 = 3.3 - (3.2 + 1.1 \times 0) = 0.1$, $r_2 = 4.1 - (3.2 + 1.1 \times 1) = -0.2$, and $r_3 = 5.5 - (3.2 + 1.1 \times 2) = 0.1$. On the other hand, the errors are given by $\varepsilon_i = y_i - (\beta_0 + \beta_1 x_i)$: $\varepsilon_1 = 3.3 - (3 + 1 \times 0) = 0.3$, $\varepsilon_2 = 4.1 - (3 + 1 \times 1) = 0.1$, and $\varepsilon_3 = 5.5 - (3 + 1 \times 2) = 0.5$. Note that ideally $\sum r_i = 0$ (a property of least squares), whereas $\sum \varepsilon_i = 0.9 \neq 0$. □

Note: It is important to recognize that the roots of machine learning lie in the field of probability and statistics. The fundamentals of probability distributions, estimators and estimation, convexity and convergence, and error measures are all based on theoretical underpinnings provided in this field. Essentially, they ensure that the ML techniques will work; that is, they will converge to the desired values with a certain amount of assurance and a certain level of accuracy. As such, it is necessary to cover the basics of probability and statistics as it relates to the subject matter at hand.

2.4.3 Distribution of Residuals

We are now ready to carry out a statistical analysis of the probability distribution of regression errors. Since we do not have access to the true errors, our focus will be on the residuals which we can compute after estimation. The starting point of the statistical analysis is to examine the properties of random variables because the residuals are themselves random variables. In this section, we use X as a generic random variable but you should view it as the random variable representing residuals. More formally, we define X as an independent and identically distributed (i.i.d.) random variable (r.v.). This type of assumption will be carried throughout the book.

A random variable may be distributed in a variety of different ways. The probability density function (PDF) specifies a continuous distribution denoted by $f_X(x)$. The cumulative distribution function (CDF) gives the probability that the random variable is less than a particular value and is denoted as $F_X(x)$.

These two functions are directly related to one another. If we know the CDF, we can obtain the PDF by differentiation:

$$f_X(x) = \frac{dF_X(x)}{dx}. \tag{2.4.7}$$

Likewise, if we know the PDF, we can integrate to obtain the CDF. That is,

$$F_X(x) = \mathbb{P}(X \leq x) = \int_{-\infty}^{x} f_X(t)dt. \tag{2.4.8}$$

Recall that the Gaussian distribution has a PDF given by:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{2.4.9}$$

where σ^2 is the variance. A random variable, X, is said to follow a Gaussian distribution if any sample of X is distributed according to the equation above. The compact notation used for random variable X following a Gaussian distribution is

$$X \sim \mathcal{N}(0, \sigma^2). \tag{2.4.10}$$

Here, the $\mathcal{N}$ symbol refers to the normal distribution and, inside the parentheses, 0 is the mean and σ^2 is the variance. In some sense, both Eq. (2.4.9) and Eq. (2.4.10) are saying the same thing; one is just a compact version of the other. The normal distribution is important for a variety of reasons but for our purposes, it leads to the L_2 loss function which is not tolerant to outliers.

Another important but less well-known distribution is the Laplace distribution, which is commonly referred to as the double-exponential distribution. The basic form is given by

$$f_X(x) = \frac{1}{2} e^{-|x|}. \tag{2.4.11}$$

The interesting part for our purposes is that this distribution leads us to the L_1 loss function which is tolerant to outliers in the dataset. For this reason, we will be encountering this distribution several times in this and other chapters.

The distribution of well-behaved errors is usually taken to be $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. That is, the mean is always zero but the variance may differ from dataset to dataset. The tacit assumption is that the residuals of linear regression follow a Gaussian distribution. Outliers cause the distribution to depart from this standard assumption which is why robust methods, involving the use of the L_1 loss, are considered. This is a subtle but important point. If the residuals are not normally distributed, then robust methods are likely to produce better results.

Two quantities of great interest when working with probability distributions are the expected value and the variance. The expected value of X is the first moment of the PDF, given by

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)dx. \tag{2.4.12}$$

We use the notation $\mathbb{E}[X]$ for convenience to represent this integral. It can be viewed as the expected value operator $\mathbb{E}[\cdot]$ acting on X. In simpler terms, this is the value you would expect if you ran an experiment of independently selecting X a large number of times (approaching infinity) and then taking the average.

It is commonly known as the mean of the distribution. We are also interested in the second moment as follows:

$$\mathbb{E}[X^2] = \int_{-\infty}^{\infty} x^2 f_X(x)dx. \tag{2.4.13}$$

The reason for our interest in the second moment is that the variance can be expressed in terms of this quantity. In particular, the variance of X can be expressed using the two expectations above as

$$\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2; \tag{2.4.14}$$

that is, the second moment minus the first moment squared is the variance. Intuitively, this gives us a measure of spread or variability of the random variable about the mean.

For those unfamiliar with the expectation represented as $\mathbb{E}[X]$, there is a simple way to think about it. Every time it is encountered, view $\mathbb{E}$ as an averaging operator on X. If you draw x_i (an observation of X) from a distribution n times, then the average, $\frac{1}{n}\sum_{i=1}^{n} x_i$, is an estimate of the expectation. Now if you draw x_i an infinite number of times and take the average, you obtain the true mean of the distribution which is expressed as $\mathbb{E}[X]$. More rigorously,

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} x_i = \mathbb{E}[X].$$

In the case of $\mathbb{E}[X^2]$, it can be viewed simply as $\frac{1}{n}\sum_{i=1}^{n} x_i^2$, where n goes to infinity. More rigorously,

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} x_i^2 = \mathbb{E}[X^2].$$

These results follow directly from the Law of Large Numbers and limit theory.

2.5 Estimation Background

Having described the goals of linear regression, the models, the errors, the residuals, and two competing loss functions, we will now step back and build up the knowledge of how to derive the functions that will lead to the estimates of the slope and intercept. We will briefly explore desirable characteristics of estimators and how to use them for comparison purposes. This section concludes with a rudimentary description of numerical optimization using gradient descent which is how the estimates are actually computed.

2.5.1 Linear Models

For problems with one explanatory variable, the equation was shown earlier to be

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x}_1 + \boldsymbol{\varepsilon} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}. \tag{2.5.1}$$

This is referred to as the **simple linear model**. For problems with more than one variable, the equation becomes

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x}_1 + \beta_2 \boldsymbol{x}_2 + \cdots + \beta_d \boldsymbol{x}_d + \boldsymbol{\varepsilon} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}. \tag{2.5.2}$$

This is referred to as the **multiple linear model**. If there are n observations and d variables, then $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$, $\boldsymbol{X} \in \mathbb{R}^{n\times(d+1)}$, and $\boldsymbol{\varepsilon} \in \mathbb{R}^n$. The only difference relative to the simple linear model is that we now have $d + 1$ unknowns. The solution to this setup proceeds in the same manner as for simple linear regression to obtain $\hat{\boldsymbol{\beta}}$.

2.5.2 Desirable Properties of Estimators

In this book, we will investigate several estimators for robust regression so we have to follow certain rules of engagement in order to ensure that they are well-suited for the machine learning task. Hence, we should understand certain properties of estimators that are considered desirable. Specifically, we prefer estimators that are consistent, asymptotically normal, and have a low bias and a low variance. While the material to be presented here is essential in the theoretical sense, it is not needed to follow most of the material in the rest of this book. However, in the interest of completeness, we provide the definitions below.

Let θ be a scalar parameter of interest and $\hat{\theta}_n$ be an estimate obtained using some estimator with sample size n. Then, such an estimator is **consistent** if

$$\hat{\theta}_n \xrightarrow{\mathbb{P}} \theta \text{ as } n \to \infty. \tag{2.5.3}$$

In simple terms, we want an estimator that is able to produce the true value given an infinite number of observations. That is, we want our estimator to converge "in probability" (denoted by $\xrightarrow{\mathbb{P}}$ above) to the true value as the number of observations in our dataset goes to infinity. This is an expression of the *Law of Large Numbers*. In practice, we have a finite amount of data but if the estimator is consistent, we can reasonably expect the estimate to be near the true value. Otherwise, we have little or no hope of finding its correct value.

We prefer estimators that exhibit **asymptotic normality**, that is

$$\sqrt{n}(\hat{\theta}_n - \theta) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2), \text{ as } n \to \infty.$$

This is a complicated looking expression, but we are expressing the desire to have an estimator that produces estimates that are distributed as a Gaussian. In simple terms, we want the distribution of the estimates obtained by using many different samples to be a Gaussian centered at the true value θ with variance σ^2 as $n \to \infty$. More formally, we would like to have convergence "in distribution" (denoted by $\xrightarrow{\mathcal{D}}$ above) of the quantity $\sqrt{n}(\hat{\theta}_n - \theta)$ to a Gaussian with a mean 0 and asymptotic variance σ^2 (i.e. the asymptotic variance of $\hat{\theta}$). This property is derived from the *Central Limit Theorem* and says that the estimates have a Gaussian distribution which makes it easier to compute the uncertainty around each estimated value.

Finally, the **bias** of an estimator is given as

$$\text{bias} = \mathbb{E}[\hat{\theta}] - \theta. \tag{2.5.4}$$

That is, how close is the expected value of our estimator to the true value. Ideally, the bias should be 0 in which case the estimator is said to be **unbiased**. Most of the estimators in this book will be unbiased except for the ones where we purposely introduce bias to reduce the variance. One notable example of a biased estimator is that of the sample variance. Most readers are familiar with the equation,

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2, \tag{2.5.5}$$

and tend to use it without much thought. However, this is a biased estimator. It can be shown that the unbiased estimator for σ^2 is actually

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2. \tag{2.5.6}$$

However, both estimators are consistent meaning that, in the limit as $n \to \infty$, they both converge to the true variance. In machine learning, the bias–variance trade-off is an important concept to understand in detail. This is why we need to introduce the bias of estimators at this stage.

2.5.3 Maximum-Likelihood Estimation

We are now ready to derive estimators from distributions using a well-known procedure called *maximum-likelihood estimation* (MLE). The reason we are interested in such a procedure is that estimators derived using MLE inherit all the nice properties described in the last section, assuming some mild regularity conditions. The basic idea is that if we have an i.i.d. random variable X with sample observations of $x_1, x_2, \ldots, x_n$ and we seek to estimate a parameter, θ, using an associated distribution, then MLE can be used to construct a suitable estimator with some nice properties.

Specifically, if we have a PDF, $f_\theta(x)$, and we wish to estimate θ, then we can construct a likelihood function as follows:

$$L(x_1, x_2, \ldots, x_n; \theta) = \prod_{i=1}^{n} f_\theta(x_i). \tag{2.5.7}$$

This likelihood function is the joint density of i.i.d. observations from $f_\theta(x)$. Since they are i.i.d., we can take the product of the distributions. We want to find the θ that maximizes $L(x_1, x_2, \ldots, x_n; \theta)$ and this will constitute an estimate of our unknown parameter, θ. In effect, we are trying to find the θ that would best represent the distribution from which our x_i values were drawn. This is a key point to understand: we are given the distribution $f_\theta(x)$ and asking what value of θ would be most likely to be associated with it, given set of x_i values. There is an implicit assumption or belief that the x_i's were drawn from a specific distribution but the assumption may not be true. This is worth keeping in mind when using MLE. It does not guarantee that the data came from the given distribution. Rather, the estimate $\hat{\theta}$ is the one that maximizes the likelihood function if the data were sampled from the given distribution.

The estimate itself is denoted by $\hat{\theta}$. Then to find this value, we use

$$\hat{\theta} = \underset{\theta \in \mathbb{R}}{\operatorname{argmax}}\, L(x_1, x_2, \ldots, x_n; \theta) = \underset{\theta \in \mathbb{R}}{\operatorname{argmax}} \prod_{i=1}^{n} f_\theta(x_i). \tag{2.5.8}$$

This notation simply states that we should find the θ that maximizes the likelihood function. In other words, MLE finds the estimate $\hat{\theta}$ that maximizes the probability that the n observation, $x_1, x_2, \ldots, x_n$, were sampled from the distribution $f_{\hat{\theta}}(x)$.

The way to find $\hat{\theta}$ is to take the derivative of the likelihood function with respect to θ and set it to 0. The procedure is facilitated by taking the logarithm of the likelihood function, the so-called log-likelihood form, as follows:

$$\ell(x_1, x_2, \ldots, x_n; \theta) = \log L(x_1, x_2, \ldots, x_n; \theta) = \sum_{i=1}^{n} \log(f_\theta(x_i)). \tag{2.5.9}$$

By doing so, the product in Eq. (2.5.8) has now been converted to a sum in Eq. (2.5.9). Next, we use a numerical optimizer to produce $\hat{\theta}$. However, most optimizers are designed to find the minimum (rather than the maximum) so we need one more adjustment to the equation: we negate the quantity. Thus, the negative log-likelihood expression is given by

$$-\ell(x_1, x_2, \dots, x_n; \theta) = -\sum_{i=1}^{n} \log(f_\theta(x_i)). \tag{2.5.10}$$

The estimator $\hat{\theta}$ is the solution to the equation

$$-\frac{\partial \ell(x_1, x_2, \dots, x_n; \theta)}{\partial \theta} = 0. \tag{2.5.11}$$

In some simple one-dimensional cases, a closed-form solution can be found. But this is not generally true. However, the equation can be solved in a straightforward manner using an optimization technique, as described in the next section, the steps for which can be expressed compactly as

$$\hat{\theta} = \underset{\theta \in \mathbb{R}}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} \log f_\theta(x_i) \right]. \tag{2.5.12}$$

Example: Suppose we believe that the distribution associated with a random variable X has the form $f_\theta(x) = \theta e^{-\theta x}$ and we would like to estimate θ using the maximum-likelihood procedure. An i.i.d sample of X produces the following five observations: 0.1, 0.2, 0.5, 1, and 3. The likelihood function is

$$L(x_1, x_2, \dots, x_5; \theta) = \prod_{i=1}^{5} \theta e^{-\theta x_i} = \theta^5 e^{-\theta \times (0.1+0.2+0.5+1+3)} = \theta^5 e^{-4.8\theta}.$$

If the likelihood function is plotted to find θ, it produces the left panel of Figure 2.3 which has a maximum at the quantity we seek.

The MLE procedure uses the negative log-likelihood which produces

$$-\ell(x_1, x_2, \dots, x_5; \theta) = -5 \log \theta + 4.8\theta,$$

as shown on the right in Figure 2.3 and then takes the derivative and sets it to zero. Doing so, we find that it has a minimum at $\hat{\theta} = 25/24$. The true value is $\theta = 1$. The estimate using the MLE procedure is close to the actual value with only five observations. A larger number of observations would move the estimate increasingly closer to the true value since MLE produces a consistent estimator. □

Exercise: Using MLE, derive the L_2 loss function assuming that residuals $(r_1, r_2, \dots, r_n)$ are i.i.d values drawn from random variable $R \sim f(r) = e^{-r^2/2}/\sqrt{2\pi}$, where $r_i = y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}$.

Ans.: The likelihood function is

$$L(r_1, r_2, \dots, r_n; \boldsymbol{\beta}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-r_i^2/2}. \tag{2.5.13}$$

The negative log-likelihood expression is

$$-\ell(r_1, r_2, \dots, r_n; \boldsymbol{\beta}) = \frac{n}{2} \log(2\pi) + \frac{1}{2} \sum_{i=1}^{n} r_i^2. \tag{2.5.14}$$

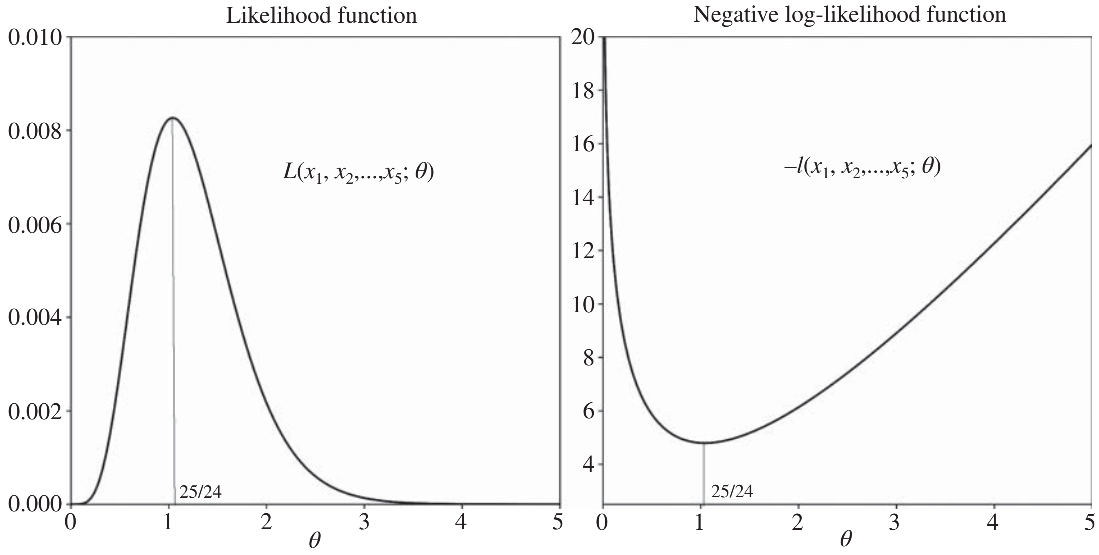


Figure 2.3 The likelihood and negative log-likelihood functions.

The first term is constant for a given n which is ignored during minimization and this gives rise to the L_2 loss function in terms of residuals:

$$L_2 \text{ loss} = \frac{1}{2}\sum_{i=1}^{n} r_i^2. \tag{2.5.15}$$

□

Exercise: Using MLE, derive the L_1 loss assuming that $r_1, r_2, \dots, r_n$ are i.i.d values drawn from random variable $R \sim f(r) = \frac{1}{2}e^{-|r|}$, where $r_i = y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}$.

Ans.: The likelihood function is given by:

$$L(r_1, r_2, \dots, r_n; \boldsymbol{\beta}) = \prod_{i=1}^{n} \frac{1}{2} e^{-|r_i|}. \tag{2.5.16}$$

The negative log-likelihood expression is

$$-\ell(r_1, r_2, \dots, r_n; \boldsymbol{\beta}) = n \log 2 + \sum_{i=1}^{n} |r_i|. \tag{2.5.17}$$

The first term is constant for a given n which is ignored during minimization and this gives rise to the L_1 loss function in terms of residuals:

$$L_1 \text{ loss} = \sum_{i=1}^{n} |r_i|. \tag{2.5.18}$$

□

The MLE procedure guarantees, under mild assumptions of continuity and differentiability, that the estimator is consistent and asymptotically normal. These are two important and desirable characteristics of an estimator. Furthermore, it produces estimators with low variance, which is indeed another desirable property. Therefore, we should pursue robust estimators that follow the MLE procedure, if possible. While we will not delve much further into the theory behind maximum-likelihood methods here, we will be using the procedure in a number of places so it is worthwhile to consult other sources for a more in-depth study.

2.5.4 Gradient Descent

The majority of the techniques to be described in the rest of this book involve optimizing a loss function to obtain estimates of the parameters of interest. Therefore, it is worth spending some time on the desirable characteristics of loss functions and, in particular, the *convexity property*.

Figure 2.4 shows two hypothetical loss functions, labeled as $f(x)$. The one in the left panel is a non-convex function. It has three minima: one global minimum (A) and two local minima (B and C). This type of function is difficult to optimize in general since the techniques typically used may end up at one of the two local minima instead of the global minimum. The panel on the right shows a convex function with only one minimum value and it is rather straightforward to optimize this type of function. Whenever possible, the loss function that we use in machine learning applications should be convex with a unique solution. However, it is not always possible to find such a function, especially in the case of neural networks. Nevertheless, we usually find a minimum (whether it be local or global) using a numerical optimization technique and generate estimates associated with this point.

One of the many ways to check for convexity is to take the second derivative of the loss function and see if it is non-negative. If so, then the function is convex and there is a unique solution available. If not, then there may be multiple solutions or an infinite number of solutions. In fact, having a strictly convex loss function is highly desirable and leads to unique solutions.

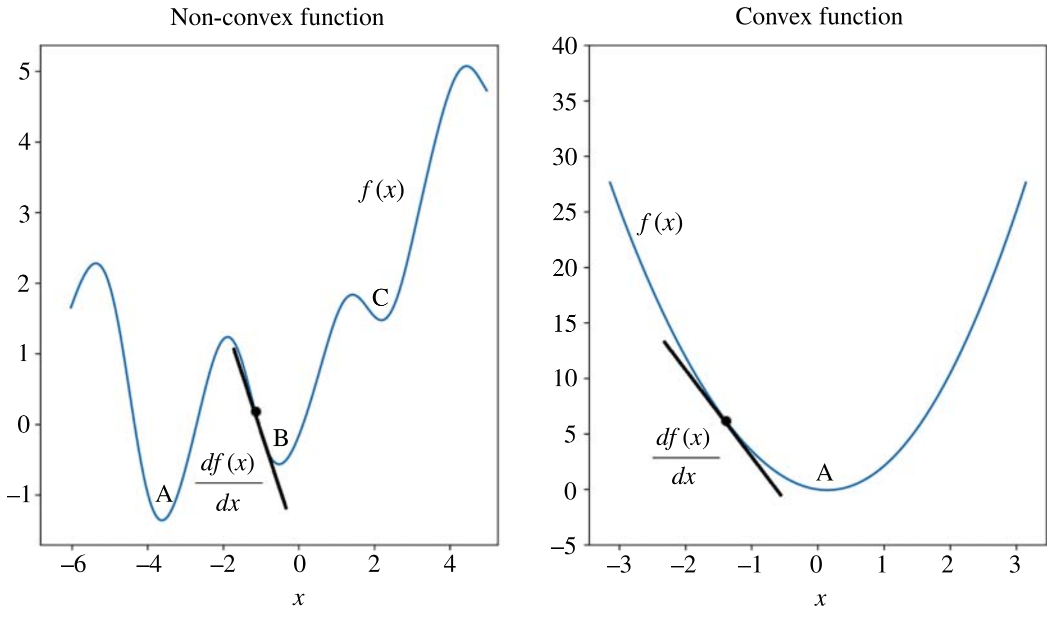


Figure 2.4 Comparing non-convex and convex functions.

The method generally used to minimize a convex function is called *gradient descent*. There are many variations of this technique but for now it suffices to understand the basic idea. In gradient descent, we start with a random initial point on the function and find the gradient at that point. This is shown in the vicinity of the minimum B for the non-convex case and in the vicinity of the minimum A for the convex case in Figure 2.4. If the slope is negative, as in this case, then the minimum lies to the right of the point. Conversely, if the slope is positive, the minimum lies to the left of the initial guess. Using $f(x)$ as the loss function and t as the iteration index, we can update the value of our next guess of the optimal value of x using

$$x^{(t+1)} = x^{(t)} - \left.\frac{df(x)}{dx}\right|_{x^{(t)}}. \tag{2.5.19}$$

The negative sign reflects the fact that we should move in the opposite direction of the slope. However, it may not make sense to take a large step in one direction or the other due to the fact that we could oscillate back-and-forth or go well beyond the optimal value. For this reason, we introduce a factor called (in the parlance of machine learning) the *learning rate*, ℓ, to tune the step size, as follows:

$$x^{(t+1)} = x^{(t)} - \ell \times \left.\frac{df(x)}{dx}\right|_{x^{(t)}}. \tag{2.5.20}$$

The value of ℓ can be fixed or adjusted during each iteration to enable faster convergence. Using this simple approach in the convex case would produce a minimum at A but in the non-convex case, it might produce a minimum at B, which is local minima. Clearly, the advantage of the convex optimization is that the global minimum is always obtained (considering some mild assumptions about the existence and continuity of the function and its first derivative).

Exercise: Consider the function $f(x) = (x-2)^2$. Use the update equation

$$x^{(t+1)} = x^{(t)} - \left.\frac{df(x)}{dx}\right|_{x^{(t)}}$$

and the initial condition $x^{(0)} = 3$ to find the minimum. Apply only two steps of the iteration.

Ans.: The derivative is $df(x)/dx = 2(x-2)$.

Iteration 1:

$$x^{(1)} = x^{(0)} - 2(x^{(0)} - 2) = 3 - 2(3-2) = 1$$

Iteration 2:

$$x^{(2)} = x^{(1)} - 2(x^{(1)} - 2) = 1 - 2(1-2) = 3$$

This will not convergence as it will oscillate from one side to the other. □

Exercise: Consider the function $f(x) = (x-2)^2$ again. This time, use the update equation

$$x^{(t+1)} = x^{(t)} - \ell \times \left.\frac{df(x)}{dx}\right|_{x^{(t)}}$$

and the initial condition $x^{(0)} = 3$ to find the minimum. Apply only two steps of the iteration using $\ell = 1/2$.

Ans.: The derivative is $df(x)/dx = 2(x-2)$.

Iteration 1:

$$x^{(1)} = x^{(0)} - \frac{1}{2} \times 2(x^{(0)} - 2) = 3 - 2(3-2)/2 = 2$$

Iteration 2:

$$x^{(2)} = x^{(1)} - \frac{1}{2} \times 2(x^{(1)} - 2) = 2 - 2(2-2)/2 = 2$$

Convergence occurred in two iterations. □

As we are considering various loss functions for use in machine learning, we should check if they are convex or non-convex. We can do this using the second derivative of a function. First, let $x \in I$ where I is some interval of interest. Then the function $f(x)$ is convex in interval I if and only if $f''(x) \geq 0$ for all $x \in I$, and strictly convex if and only if $f''(x) > 0$. So we can test for convexity using this criterion. For most of the loss functions used in practice, this will be true but the condition should be checked with any new loss functions. However, in machine learning, especially in deep neural networks, this may not be true and a local minimum is acceptable. In such cases, the optimization techniques will be much more complex than the simple gradient descent described above, as presented later in Chapters 9 and 10.

2.6 M-Estimation

In our search for a robust loss function for machine learning, we will need a method to compare different candidates that we encounter along the way. In robust statistics, there is a class of estimators called *M-estimators* (see Maronna 2019) that will be useful throughout this book for this purpose. M-estimators are used for robust regression, where the objective is to minimize a function of the residuals. The simplest way to view these estimators is that they are all defined by a generic loss function, as follows:

$$\text{M-loss} = \sum_{i=1}^{n} \rho(r_i) = \sum_{i=1}^{n} \rho(y_i - x_i^{\top} \boldsymbol{\beta}). \tag{2.6.1}$$

The idea is that the function $\rho(r_i)$ is selected to deliver robust estimates. The function itself depends solely on your ingenuity and not necessarily on rigorous derivations carried out using MLE. If we can specify the $\rho(r_i)$ function such that it is convex and gives desirable estimates, then we can optimize it to obtain a robust model. Here we do not need to specify the distribution or carry out any steps of the MLE procedure. We simply develop an estimator based on some $\rho(r_i)$ that satisfies a few requirements of continuity and convexity. The derivative of the $\rho(r_i)$ is given by

$$\psi(r_i) = \frac{\partial \rho(r_i)}{\partial r_i}. \tag{2.6.2}$$

The continuity requirement is needed to carry out differentiation so that gradients can be used to numerically solve the equation. Assuming that $\rho(r_i)$ is differentiable, the estimates are the set of values that satisfy the following condition:

$$\sum_{i=1}^{n} \psi(r_i) = 0. \tag{2.6.3}$$

The convexity condition is needed to find a unique minimum. We can assess the convexity of the loss function using the second derivative:

$$\psi'(r_i) = \frac{\partial^2 \rho(r_i)}{\partial^2 r_i}. \tag{2.6.4}$$

Then, the **convexity property** is satisfied if and only if

$$\psi'(r_i) \geq 0 \ \text{ for all } r_i \tag{2.6.5}$$

and it is *strictly convex* if $\psi'(r_i) > 0$ for all r_i.

The M-estimation approach to specifying loss functions can also be used for non-robust estimators. For L_2, we have that

$$\rho_{\mathrm{L2}}(r_i) = \frac{r_i^2}{2}.$$

Therefore,

$$\psi_{\mathrm{L2}}(r_i) = r_i$$

and

$$\psi'_{\mathrm{L2}}(r_i) = 1.$$

So L_2 has a strictly convex loss function. Now if we check L_1 we find that

$$\rho_{\mathrm{L1}}(r_i) = |r_i|.$$

Therefore,

$$\psi_{\mathrm{L1}}(r_i) = \mathrm{sgn}(r_i)$$

and

$$\psi'_{\mathrm{L1}}(r_i) = \begin{cases} \text{undefined} & \text{if } r_i = 0 \\ 0 & \text{if } r_i \neq 0 \end{cases}. \tag{2.6.6}$$

The function is convex but not strictly convex and is undefined at $r_i = 0$. So we cannot guarantee that a unique solution exists. In fact, there may be an infinite number of localized solutions using L_1.

Figure 2.5 plots the $\rho(r_i)$ and $\psi(r_i)$ functions for L_1 and L_2. Note that L_1 follows an absolute function, $|r_i|$, while L_2 follows a quadratic function, $r_i^2/2$. The problem with using L_2 is that the residuals are squared, which amplifies the importance of large residuals. During linear regression, outliers are considered to be, in some sense, more important than the majority of the data since outliers tend to have larger residuals. Thus, it is highly sensitive to outliers. This is the main problem with the L_2 estimator.

The L_1 estimator is not as sensitive to outliers because it sums the absolute value of the residuals in the loss function, but it has other problems. The $\rho(r_i)$ function has a distinctive kink at $r_i = 0$. When we examine its first derivative in the lower-left panel of Figure 2.5, we note that it is a discontinuous function whereas the first derivative of L_2 is continuous, as seen in the lower-right panel. The discontinuity of L_1 poses problems both mathematically and in numerical optimization when used for machine learning. Further, the second derivative of L_1 produces an impulse function at the origin while the second derivative of L_2 is flat and continuous at a value of 1. Clearly, L_1 and L_2 both have notable disadvantages so alternative methods are needed to handle outliers.

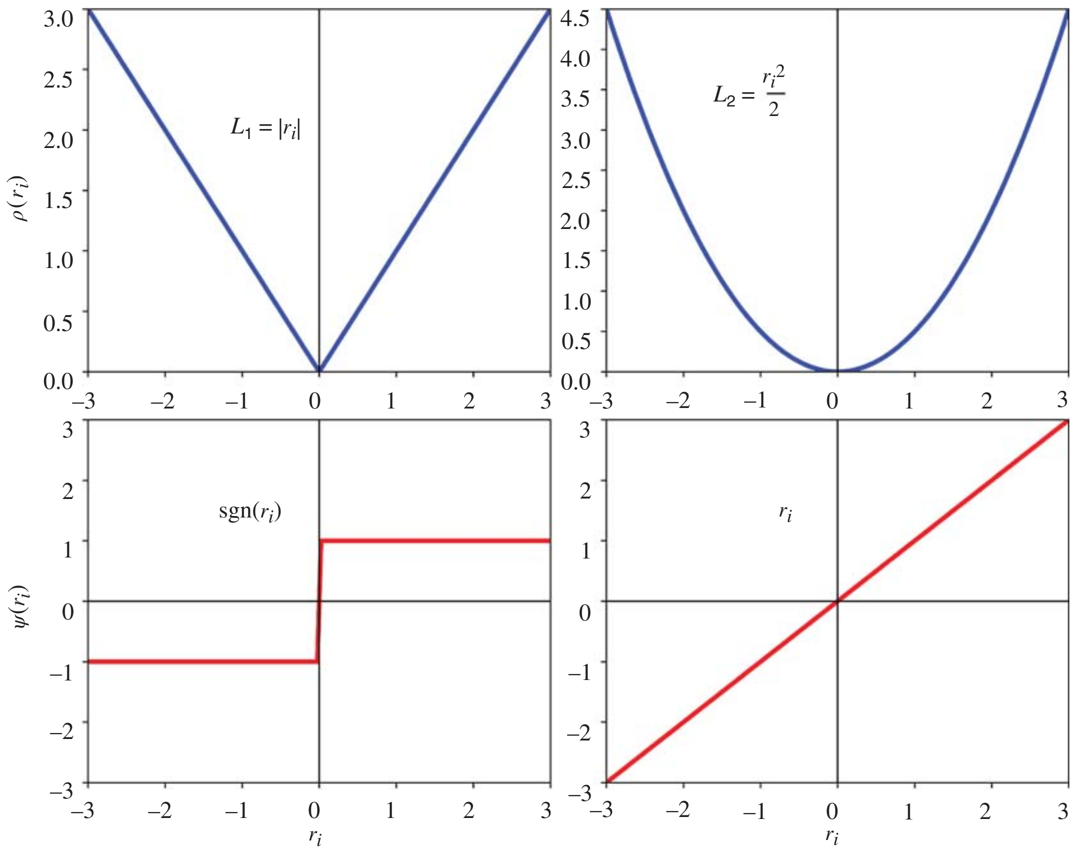


Figure 2.5 Comparing the loss term and its derivative for L_1 and L_2 as a function of the residual value, r_i.

The question that immediately arises is: How do we come up with a better M-estimator? The answer is to try out different options and compare them against one another. In particular, if you want to find the median of the data and you find that the L_1 loss function is not suitable, then any other $\rho(r_i)$ function that seeks the median would be a suitable replacement. In a later section, we will consider the Huber estimator as our first M-estimator in this chapter, followed by the log-cosh estimator in the next chapter.

Notation: The general form of an M-estimator is given by:

$$\hat{\boldsymbol{\beta}}^* = \underset{\beta}{\text{argmin}} \left[\sum_{i=1}^{n} \rho(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}) \right].$$

The use of **argmin** indicates the loss function should be minimized to find the associated $\boldsymbol{\beta}$ located at the minimum. The superscript * is usually replaced by acronyms such as L1, L2, LS, LAD, LC, and H to refer to the specific type of loss function used.

2.7 Least Squares Estimation (LSE)

We are now ready to examine the least squares method in a more formal setting using the proper notation. Consider the multiple linear model in matrix form as follows:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{2.7.1}$$

where $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$ is a response vector, $\boldsymbol{X}$ is an $n \times (d+1)$ design matrix of constants with an embeded column of 1's, and $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)^\top$ is the error term. Here we assume $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2\boldsymbol{I}_n)$, the normal distribution with mean $\boldsymbol{0}$ and variance–covariance matrix $\sigma^2\boldsymbol{I}_n$. Regression parameters $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_d)^\top$ are the $d+1$ coefficients of interest.

The solution to a multidimensional problem is no longer a line but rather a plane or hyperplane. An example of a two-variable regression model and its solution is illustrated in Figure 2.6. In this case, the slopes are β_1 and β_2, while the intercept on the y-axis is β_0. The slopes represent the angle of the plane in each dimension. Beyond two dimensions, the plane becomes a hyperplane. For obvious reasons, the hyperplanes cannot be visualized; however, the same concepts apply.

We can write the L_2 loss function in two different ways. The first uses matrix notation, as follows:

$$J(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}). \tag{2.7.2}$$

This is just the sum of the squares of the errors written in matrix notation. To obtain the least squares estimates, $\hat{\boldsymbol{\beta}}^{\mathrm{LS}}$, we minimize this loss function by taking the derivative of $J(\boldsymbol{\beta})$ w.r.t. $\boldsymbol{\beta}$ and setting it to 0, that is

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0. \tag{2.7.3}$$

Then, it follows that

$$-\boldsymbol{X}^\top\boldsymbol{y} + \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\beta} = 0, \tag{2.7.4}$$

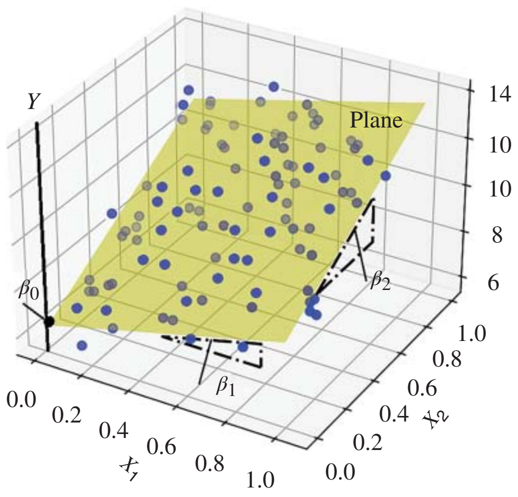


Figure 2.6 A plane with slopes β_1 and β_2 and intercept β_0 representing the solution for a multiple linear regression problem with two variables.

which can then be rearranged to produce the well-known closed-form solution:

$$\hat{\boldsymbol{\beta}}^{\text{LS}} = (\boldsymbol{X}^{\top}\boldsymbol{X})^{-1}\boldsymbol{X}^{\top}\boldsymbol{y}. \tag{2.7.5}$$

This is the direct matrix solution for LSE and it produces the best linear unbiased estimate (BLUE), provided $\boldsymbol{X}^{\top}\boldsymbol{X}$ is invertible. By best, we mean the one with the *lowest variance*. It is also referred to in the literature as the ordinary least squares (OLS) estimator. There are volumes of books and information available about OLS which we will not cover here.

Another way to obtain the least squares estimates is to apply an iterative method. We saw earlier that the L_2 loss function is given by

$$L_2\,\text{loss} = \frac{1}{2}\sum_{i=1}^{n} r_i^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^{\top}\boldsymbol{\beta})^2. \tag{2.7.6}$$

Then, the least squares solution is obtained as follows:

$$\hat{\boldsymbol{\beta}}^{\text{LS}} = \underset{\beta\in\mathbb{R}^{d+1}}{\text{argmin}} \left[\frac{1}{2}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^{\top}\boldsymbol{\beta})^2\right]. \tag{2.7.7}$$

Rather than try to solve this equation in closed form, it is more appropriate to use gradient descent to find the minimum. If we take the derivative of the loss function w.r.t. β_j and use it in the gradient descent iterative equation, we obtain:

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \ell\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^{\top}\boldsymbol{\beta})(-x_{ij}). \tag{2.7.8}$$

At each iteration, t, we simply update each $\beta_j^{(t)}, j = 0, \ldots, d$, with the gradient term to produce $\beta_j^{(t+1)}$ until convergence. The direct matrix approach and the iterative approach produce identical results, if the matrix is well conditioned. However, the iterative approach does not require a matrix inversion and therefore avoids issues posed by a large $\boldsymbol{X}$ matrix, or the possibility of an ill conditioned or singular matrix. To avoid multicollinearity problems, it is also well suited to the application of a penalty function, such as ridge or LASSO (as described in Chapter 5).

We can gain further insight into the solution if we state the loss function in terms of residuals in the form of an M-estimator as follows:

$$L_2\,\text{loss} = \frac{1}{2}\sum_{i=1}^{n}\rho_{L2}(r_i) = \frac{1}{2}\sum_{i=1}^{n} r_i^2 \tag{2.7.9}$$

which is just a restatement of an earlier MLE result. Then, taking the derivative w.r.t. r_i and setting it to 0, we obtain:

$$\sum_{i=1}^{n} r_i = 0. \tag{2.7.10}$$

The solution of the L_2 loss function is a set of estimates that forces the sum of the residuals to be identically 0. Hence, in effect, we are seeking the **mean** line through the data such that the average error is 0.

2.8 Least Absolute Deviation (LAD)

An alternative approach to LS to mitigate the effects of outliers is to use the LAD approach whereby an L_1 loss function is used as follows:

$$L_1 \text{ loss} = \sum_{i=1}^{n} |r_i| = \sum_{i=1}^{n} |y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}|. \tag{2.8.1}$$

The goal is to minimize the sum of the magnitudes of the residuals rather than the squares of the residuals. This can be stated as follows:

$$\hat{\boldsymbol{\beta}}^{\text{LAD}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \left[\sum_{i=1}^{n} |y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}| \right]. \tag{2.8.2}$$

The solution to this equation is not as straightforward as it looks so care must be taken to select the proper way to find the minimum. For LAD, the loss function is *piecewise linear* and continuous. Therefore, a linear programming method or equivalent is needed to obtain a solution. The issues are that there may be an infinite number of localized solutions and a numerical optimization method such as gradient descent may struggle with L_1 since the derivative is discontinuous. We will see an alternate approach shortly that is more effective than L_1.

We can gain further insight into the solution if we state the loss function in terms of residuals in the form of an M-estimator as follows:

$$L_1 \text{ loss} = \sum_{i=1}^{n} \rho_{L1}(r_i) = \sum_{i=1}^{n} |r_i|, \tag{2.8.3}$$

which is just a restatement of an earlier MLE result. Then, taking the derivative w.r.t. r_i and setting it to 0, we obtain:

$$\sum_{i=1}^{n} \text{sgn}(r_i) = 0. \tag{2.8.4}$$

The minimum of the L_1 loss function occurs where the number of positive residuals equals the number of negative residuals. Hence, *the number of points above the line should be equal to the number of points below the line*. This is the fundamental difference between L_1 and L_2. In one case, the points above and below the line are balanced and, in the other case, the sum of the residuals must be equal to 0 (within a numerical tolerance level). We can begin to see why the L_1 loss leads to robust estimates: it is effectively counting and equalizing the points above and below the line without any concern about how far they are away from the line. Hence, we are obtaining the **median** line through the data. Two problems with L_1 are that there may be an infinite number of solutions, and we cannot use gradient descent since the first derivative of the loss function is discontinuous and the second derivative is undefined at one location. These issues open the door to a plethora of other techniques to achieve robust results while avoiding these problems.

Exercise: The residuals from two different estimators, known to be LSE and LAD, are given as follows: $\boldsymbol{r}_1 = (1.3, 2.3, 4.5, -2.3, -1.3, -2.5, -2.5, 1.0)^\top$ and $\boldsymbol{r}_2 = (1.3, 2.3, 6.0, -2.3, -1.3, -2.5, -2.5, -1.0)^\top$. Which one produced $\boldsymbol{r}_1$ and which one produce $\boldsymbol{r}_2$?

Ans.: $\boldsymbol{r}_1$ has the same number of positive and negative residuals so it came from the LAD estimator whereas $\boldsymbol{r}_2$ values sum to 0.0 so it is from LSE. □

Exercise: The residuals associated with a linear regression problem are given as follows: $\boldsymbol{r} = (-1, 1, -1, 1, 1, 1, 1, -1, -1, 1, 10)^\top$. Compute the L_1 and L_2 losses.

Ans.: For L_1, we have the sum of the absolute values. Therefore the loss is 20. For L_2, it is one-half of the sum of the squares. Therefore, the loss is 55. In the case of L_2, the value of 10 is an outlier and inflates the loss. □

2.9 Comparison of LSE and LAD

In order to gain further insight into the two methods, we examine results for the simple linear model and the location model. The simple linear model is used to illustrate the differences between the two in estimating the slope and the intercept. The location problem is used to explore the nature of the minimization process.

2.9.1 Simple Linear Model

For the simple linear model (see Section 2.5.1), we will use the Belgium telephone dataset provided in Table 2.1. It contains the number of calls (in units of 10s of millions) made in Belgium in the years between 1950 and 1973. There are 24 data points, with 6 outliers due to a change in measurement technique (call length instead of number of calls) for a 6-year period from 1964 to 1969.

Figure 2.7 shows the results of linear regression using LSE and LAD. The scatter points represent the data in Table 2.1. Note that most of the points occur in the lower portion of the plot. The points in the upper portion are outliers due to differences in measurement techniques in those intervening years. The two lines are based on estimates from Eq. (2.7.7) for LSE and Eq. (2.8.2) for LAD.

In the case of LSE, we notice that the line traces a path through the middle of the data whereas the LAD regression line is located in such a way as to split the data in half. This is due to the fact that the LSE method seeks the line that averages the errors to 0 while the LAD method seeks the line that equalizes the number of data points above and below the line. We can observe that LAD is ignoring the outlier data and capturing the essence of the true data.

Table 2.1 Belgium telephone dataset.

x	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961
y	0.44	0.47	0.47	0.59	0.66	0.73	0.81	0.88	1.06	1.2	1.35	1.49
x	1962	1963	1964	1965	1966	1967	1968	1969	1970	1971	1972	1973
y	1.61	2.12	11.9	12.4	14.2	15.9	18.2	21.2	4.3	2.4	2.7	2.9

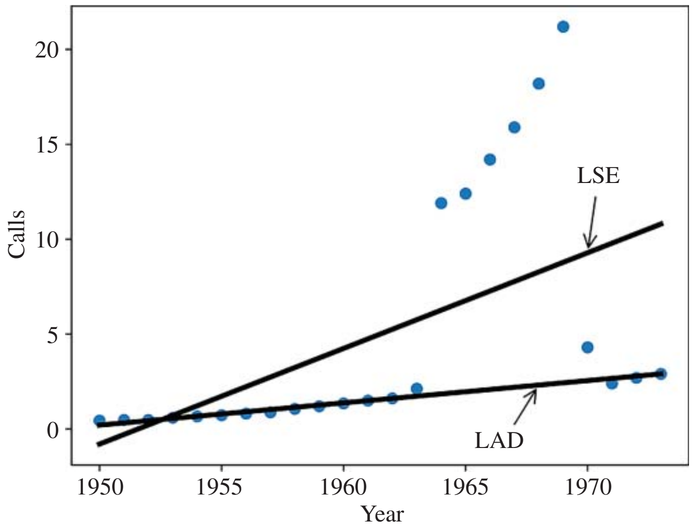


Figure 2.7 Comparing LSE and LAD using the Belgium telephone dataset. LSE is affected by outliers, whereas LAD is tolerant of outliers.

2.9.2 Location Problem

To emphasize the other key issue with the L_1 loss function compared to L_2, we will examine the location problem which is the simplest regression problem to tackle but offers additional insight into the inner workings of the two approaches. The location problem involves a one-dimensional dataset and is used to estimate the center of a sample of n observations, $x_1, \ldots, x_n$, for some random variable X. This could be the sample mean or median or any other similar type of estimate depending on the overall objective. The formulation of the **location model** is given by

$$x_i = \theta + \varepsilon_i, \quad i = 1, \ldots, n$$

where θ is the location parameter and $\varepsilon_1, \ldots, \varepsilon_n$ are i.i.d. random variables representing the errors.

To estimate the location parameter, we could use LSE by minimizing the loss function given by

$$L_2(\theta) = \frac{1}{2}\sum_{i=1}^{n}(x_i - \theta)^2. \tag{2.9.1}$$

The mean is obtained using the L_2 loss function. The gradient function is

$$L_2'(\theta) = -\sum_{i=1}^{n}(x_i - \theta). \tag{2.9.2}$$

Alternatively, another way to estimate the location parameter is to use the LAD estimator which minimizes the loss function

$$L_1(\theta) = \sum_{i=1}^{n}|x_i - \theta|. \tag{2.9.3}$$

The median is obtained using the L$_1$ loss function. The gradient function is

$$L_1'(\theta) = -\sum_{i=1}^{n} \mathrm{sgn}(x_i - \theta). \tag{2.9.4}$$

To illustrate one of the problems with L$_1$, consider the data $\boldsymbol{x} = (0, 3, 5, 7, 10)^\top$. We seek the mean and the median using the L_2 and L_1 loss functions, respectively. By inspection, both the mean and median are 5. The loss functions have been plotted in Figure 2.8 along with the gradient functions. Note that the L_2 loss is continuous and smooth, whereas the L_1 loss is continuous but **piecewise linear**. This is further confirmed by the gradient plots where the L_1 gradient is composed of a series of steps while the L_2 gradient is linear.

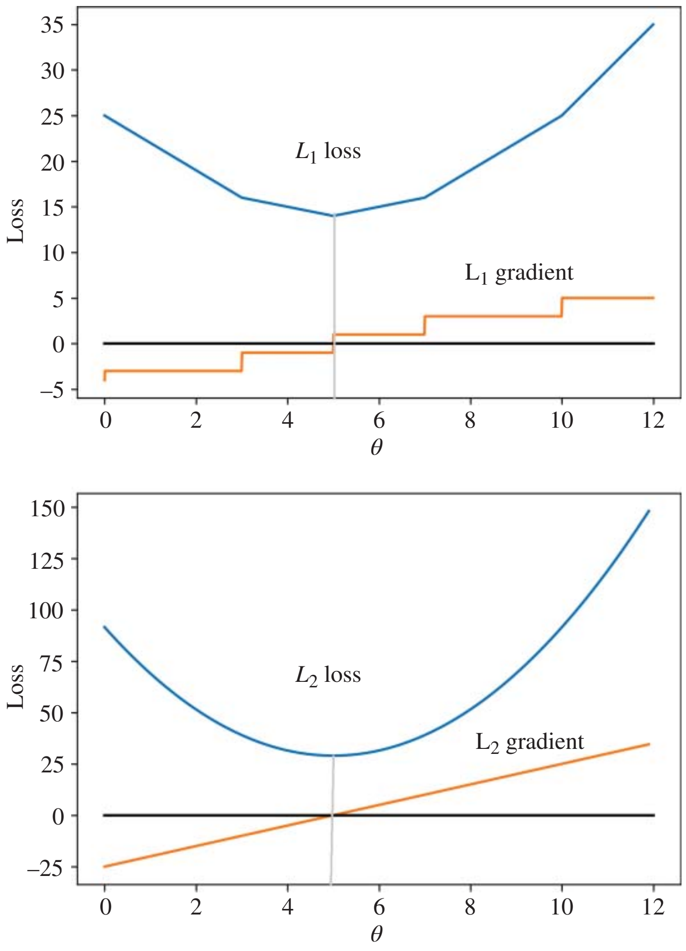


Figure 2.8 Comparing L_1 and L_2 for the location model.

The L_1 loss function requires the use of special optimization techniques in the form of linear programming, such as the simplex method (for small datasets) or an interior point method (for large problems). However, the L_2 loss can be minimized with gradient descent. This difference is crucial in machine learning since gradient descent techniques are routinely used to minimize convex loss functions, while linear programming is rarely used. Therefore, the L_1 loss function is not ideally suited for machine learning applications for this reason, along with a host of other reasons.

With the details of L_1 and L_2 established, the next step is to explore methods of obtaining a robust estimator that does not have the inherent limitations of the L_1 or L_2 loss functions but instead exploits the advantages of both. In particular, we will examine the Huber loss function.

2.10 Huber's Method

One way to create a more effective robust estimator is to try to combine the L_1 and L_2 loss functions. The advantage of the L_1 loss is that it is robust to outliers. The disadvantage is that it has continuity problems in the first and second derivatives, and it requires the use of elaborate methods such as the simplex algorithm or interior point methods to obtain accurate estimates. The L_2 loss is continuously differentiable and can be solved using gradient descent but it is highly sensitive to outliers.

2.10.1 Huber Loss Function

Referring back to Figure 2.5, Huber proposed a method that uses L_2 in the vicinity of the origin (where the kink in L_1 resides) and then switches to L_1 a certain distance away from the origin (see Huber and Ronchetti 2009). The method is quite straightforward. We first define a small interval around the origin given by $[-\delta, +\delta]$. Inside this region, we use the L_2 loss function since it does not have a kink. Outside this region, we use the L_1 loss but we take great care to match the function and its first derivative at the interface between the different regions. Therefore, the Huber loss term $\rho_H(r_i)$ and its derivative $\psi_H(r_i)$ are both continuous in all regions. The resulting Huber loss function is given by

$$\text{Huber loss} = \sum_{i=1}^{n} \rho_H(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}) = \sum_{i=1}^{n} \rho_H(r_i), \tag{2.10.1}$$

where the loss term is

$$\rho_H(r_i) = \begin{cases} r_i^2/2 & \text{if } |r_i| \le \delta \\ \delta(|r_i| - \frac{\delta}{2}) & \text{if } |r_i| > \delta \end{cases}. \tag{2.10.2}$$

The derivative of the Huber loss function is given by

$$\psi_H(r_i) = \rho'_H(r_i) = \begin{cases} r_i & \text{if } |r_i| \le \delta \\ +\delta & \text{if } r_i > +\delta \\ -\delta & \text{if } r_i < -\delta \end{cases}. \tag{2.10.3}$$

Further, the Huber loss can be used to define the Huber estimator as

$$\hat{\boldsymbol{\beta}}^{\mathrm{H}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho_H(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}). \tag{2.10.4}$$

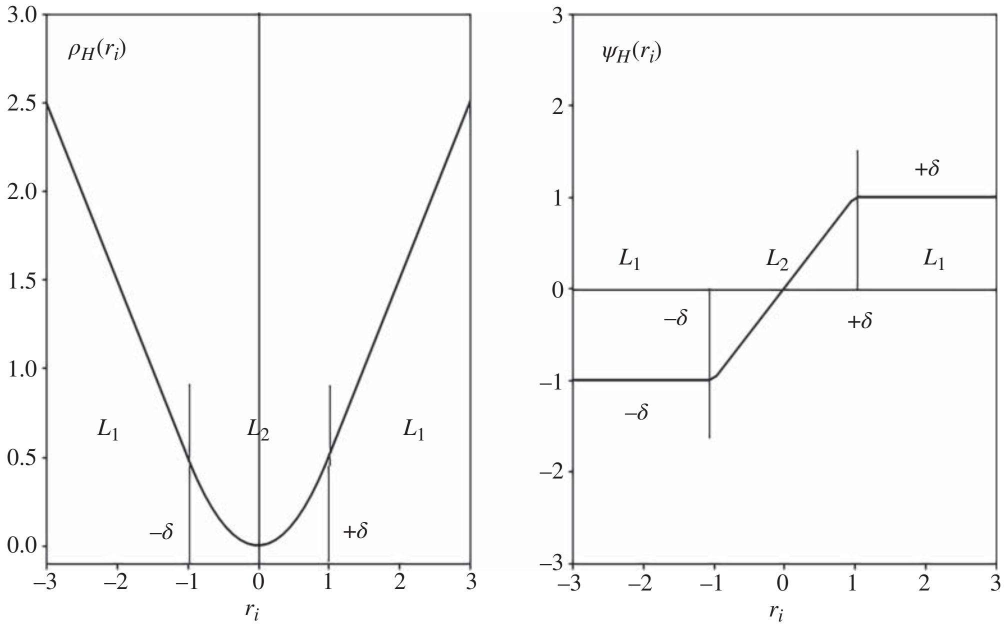


Figure 2.9 Plots of the loss term and its derivative for Huber's method as a function of the residual value, r_i, using $\delta = 1$.

Consider the two plots in Figure 2.9 with $\delta = 1$. There are a total of 3 regions defined by the loss function: $r_i < -1$ (left L_1 region), $r_i > +1$ (right L_1 region), and $|r_i| \leq 1$ (L_2 region). As shown in the figure, the region $|r_i| \leq 1$ uses the L_2 loss term while the two regions defined by $|r_i| > 1$ use the L_1 loss term. This defines a specific Huber loss term that has the following form:

$$\rho_H(r_i) = \begin{cases} r_i^2/2 & \text{if } |r_i| \leq 1 \\ |r_i| - \frac{1}{2} & \text{if } |r_i| > 1 \end{cases}. \tag{2.10.5}$$

The 1/2 term in the second case is used for continuity at the boundary of the different regions. Plugging in $r_i = 1$, both cases are equal to 1/2 in Eq. (2.10.5). More importantly, we now have a robust loss function without the L_1 kink that proved troublesome for the LAD estimator. The δ parameter in the figure can be used to adjust the boundary based on the needs of each dataset, depending on the number and location of outliers in it. By default, the common practice is to set $\delta = 1.345$ based on experimentation on a wide variety of datasets. This formulation is acceptable but we can do better by choosing the optimal boundary between regions, if needed.

Figure 2.10 shows the behavior of the Huber loss term as δ is varied. In essence, small values ($\delta \leq 1.345$) lead to characteristics that resemble the L_1 V-shape, while large values ($\delta > 1.345$) lead to characteristics that resemble the L_2 U-shape. We seek estimates that are close to the median (be it a point, line, plane, or hyperplane) and therefore employ small values of δ when outliers exist in the data. However, if the δ is too small, the issues of L_1 will reappear.

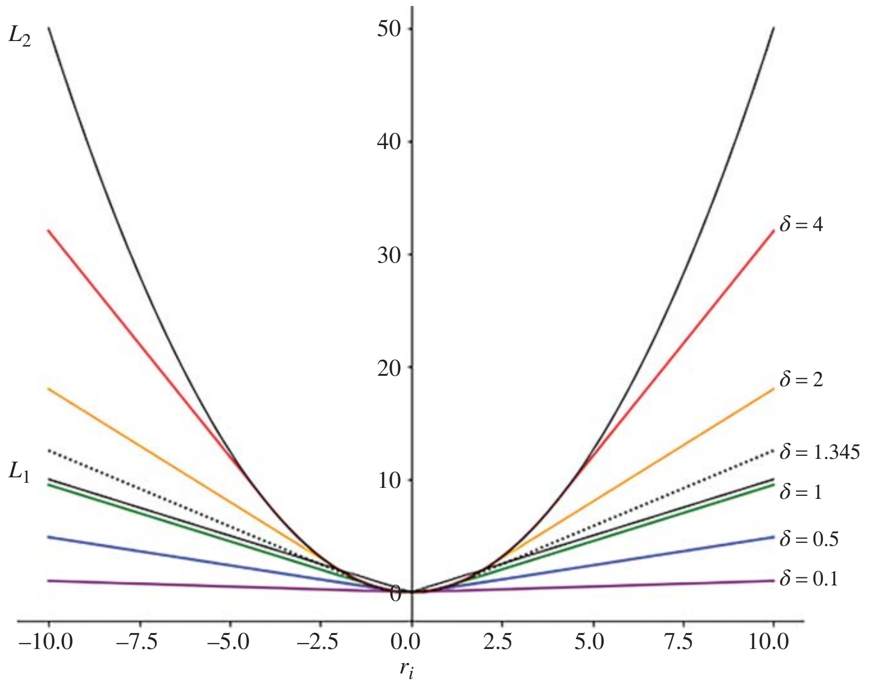


Figure 2.10 Comparing L_1, L_2, and Huber (for different values of δ).

Exercise: Write down the Huber loss term with $\delta = 0.1$ which may be used when the dataset has many outliers. Assume that r_i is the ith residual. State the intervals over which they apply (see Figure 2.9, left panel).

Ans.:

$$\rho_H(r_i) = \begin{cases} r_i^2/2 & \text{if } |r_i| \leq 0.1 \\ 0.1\left(|r_i| - \frac{0.1}{2}\right) & \text{if } |r_i| > 0.1 \end{cases}.$$

□

Exercise: Let $\delta = 0.1$. Write down the derivative of the Huber loss term (see Figure 2.9, right panel).

Ans.:

$$\psi_H(r_i) = \begin{cases} r_i & \text{if } |r_i| \leq 0.1 \\ +0.1 & \text{if } r_i > +0.1 \\ -0.1 & \text{if } r_i < -0.1 \end{cases}.$$

□

Another issue that exists in the Huber method is the convexity of the loss function. While the loss is always convex with respect to $\boldsymbol{\beta}$ (for a fixed δ), it is not necessarily convex with respect to both δ and $\boldsymbol{\beta}$. This presents a particular challenge since both δ and $\boldsymbol{\beta}$ must be determined iteratively and simultaneously. This is another reason why δ is set to a specific value, typically $\delta = 1.345$.

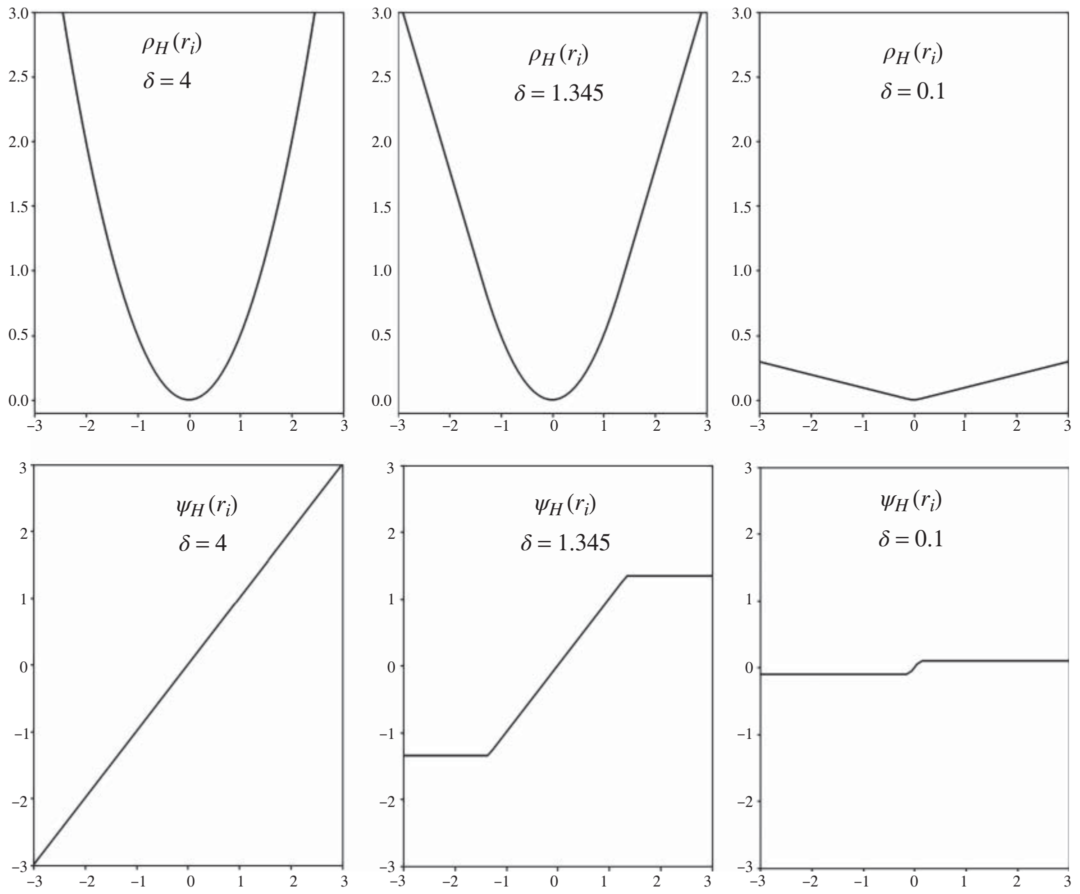


Figure 2.11 $\rho(r_i)$ and $\psi(r_i)$ as a function of the residual value, r_i, for $\delta = 4, 1.345$, and 0.1.

Exercise: Plot the Huber loss term and its derivative for $\delta = 4, 1.345$ and 0.1 using Eqs. (2.10.2) and (2.10.3). Comment on their differences. (Hint: see Figure 2.10.)

Ans.: See Figure 2.11 for solution. With $\delta = 4$, the plots are similar to L_2. For $\delta = 0.1$, the plots are similar to L_1 except they are scaled by 0.1. The plot for $\delta = 1.345$ strikes a balance between the two. □

Exercise: Write the expression for and plot the second derivative of the Huber function for $\delta = 0.1$.

Ans.: It will have two discontinuities, one at −0.1 and one at +0.1, as described by the following equation:

$$\psi'_H(r_i) = \begin{cases} 1 & \text{if } |r_i| \leq 0.1 \\ 0 & \text{otherwise} \end{cases}.$$

The resulting discontinuous Huber second derivative plot with $\delta = 0.1$ is shown in Figure 2.12. □

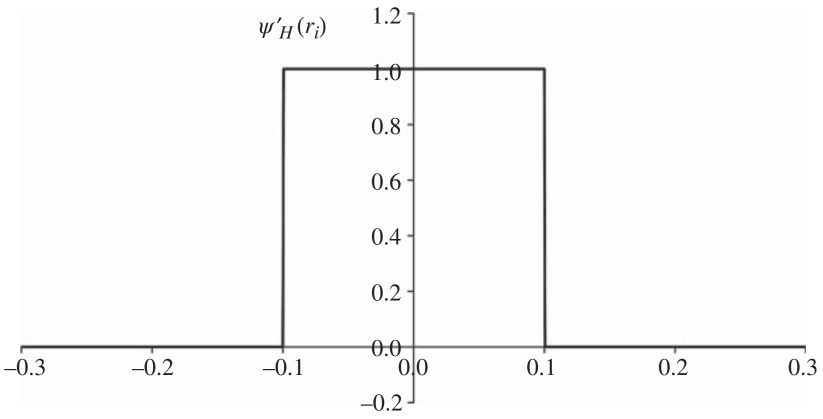


Figure 2.12 The second derivative of the Huber function $\psi'_H(r_i)$ with $\delta = 0.1$.

From the exercise above, it is clear that the Huber loss does not have a continuous second derivative. As a result, various improvements have been devised to make it continuously differentiable using smoothing techniques which will not be elaborated here since they introduce other issues as a side effect. Perhaps more troubling is that the optimization problem begins to look like L_1 as δ decreases which implies that gradient descent is not as effective, and the results become unreliable. In spite of these issues, Huber's method is a popular alternative to LSE when outliers are present.

Some insight into the inner workings of Huber's method can be gained by expressing the loss in an alternative form as follows:

$$\text{Huber loss} = \sum_{\substack{i=1\\|r_i|\le\delta}}^{n} r_i^2/2 + \sum_{\substack{i=1\\|r_i|>\delta}}^{n} \delta(|r_i| - \delta/2). \tag{2.10.6}$$

Setting $\delta = 1$ for convenience, if we take the derivative of the loss function in the various regions and set it to 0, we would obtain

$$\sum_{\substack{i=1\\|r_i|\le 1}}^{n} r_i + \sum_{\substack{i=1\\|r_i|>1}}^{n} \text{sgn}(r_i) = 0. \tag{2.10.7}$$

What Eq. (2.10.7) expresses is that the sum of the signs of the residuals in the two L_1 regions plus the sum of the residuals in the L_2 region must equal 0. One way to satisfy this condition is that we can have an equal number of residuals in the two L_1 regions, and the remainder of the residuals in the L_2 region. In the L_1 regions, the terms that are summed can only be −1 or +1. Therefore, a balance must be reached that sums to 0; otherwise, the L_2 region will have to make up the difference to make the total sum equal to 0. The method is robust since the magnitude of the residuals in the L_1 region can be any (large) value but they will only contribute −1 or +1 to the sum. Hence, Huber is robust to outliers provided the proper δ is selected.

Table 2.2 Estimates from LSE, LAD, and Huber.

	LSE	LAD	Huber
Intercept	−983.9	−305.0	−298.0
Slope	0.504	0.155	0.154
δ	—	—	0.269

2.10.2 Comparison with LSE and LAD

We compare the three methods of estimation, namely, LSE, LAD, and Huber's method, on the Belgium telephone dataset. The estimates are provided in Table 2.2. For Huber, the optimal choice of δ is data dependent as mentioned earlier. In this case, the optimal value is $\delta = 0.269$. The results of Huber's method and LAD are quite similar and both are quite different from LSE, as expected.

To gain a better understanding of the effect of δ, consider the plot of the linear regression results in Figure 2.13. In this case, 100 different values of δ were selected between 0 and 100. The resulting 100 lines produced by Huber shade the region bounded by LSE and LAD. This is expected because, at one extreme, Huber's method reduces to LSE and, at the other extreme, it becomes LAD. Again, δ is data dependent but is usually set to a default value of $\delta = 1.345$ in practice.

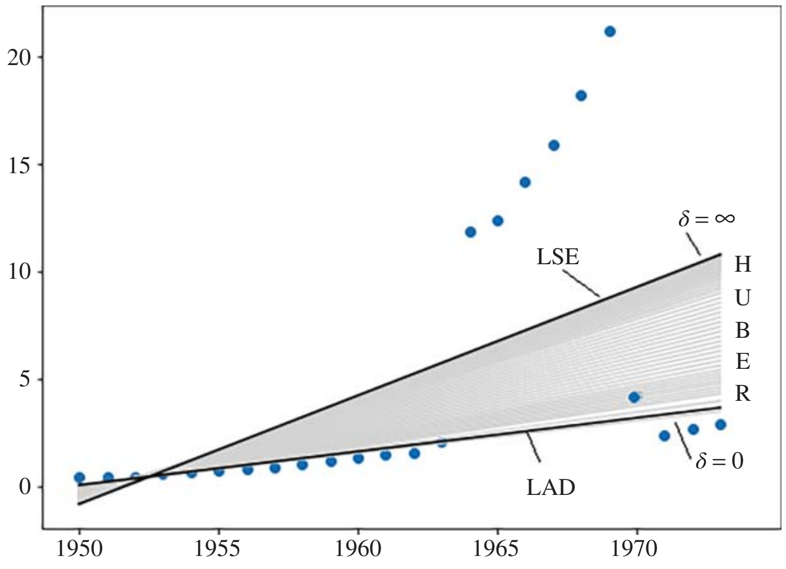


Figure 2.13 Comparing LSE, LAD, and Huber using the Belgium telephone dataset. LSE and LAD results form the boundaries for the Huber results.

2.11 Summary

This chapter introduced L_1, L_2, and Huber loss functions for linear regression. The L_1 loss function is robust to outliers but has mathematical and numerical issues that make it difficult to use in practice. The L_2 loss function, which produces the ordinary least squares estimates, is not tolerant to outliers. Huber's method combines the advantages of the L_1 and L_2 loss functions and is tolerant to outliers. However, it also has problems in that its second derivative is not continuous and we must be able to find a suitable value for δ along with the estimates themselves. Furthermore, small values of δ may not be suitable for optimization using gradient descent. In the next chapter, another approach will be presented which avoids the inherent problems with Huber's method and holds great promise in robust machine learning.

Problems

2.1 Consider the following small dataset.

x_i	0.10	0.20	0.30	0.40	0.50	0.60
y_i	3.2	4.0	4.2	4.7	6.5	5.5

Let the data be modeled as

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x} + \boldsymbol{\varepsilon}.$$

Perform hand calculations, or use Python, to do the following.

a) Find the least squares (LS) estimates using

$$\hat{\beta}_1^{\text{LS}} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}, \quad \text{and} \quad \hat{\beta}_0^{\text{LS}} = \overline{y} - \hat{\beta}^{\text{LS}}\overline{x},$$

where $\overline{x}$ and $\overline{y}$ are means of their respective variables.

b) Plot the points and the LS regression line. Identify the slope and intercept in the plot.

c) Indicate the residuals using vertical lines in the plot. What is the sum of the residuals?

2.2 What is the maximum-likelihood estimator for μ for the following distribution?

$$f_\mu(x) = \frac{1}{1 + (x - \mu)^2}$$

2.3 The L_2 loss function for the location problem is given by

$$L_2(\theta) = \frac{1}{2}\sum_{i=1}^{n}(x_i - \theta)^2.$$

The gradient function is

$$L_2'(\theta) = -\sum_{i=1}^{n}(x_i - \theta).$$

a) Plot the loss function and the gradient function for the dataset $\boldsymbol{x} = \{1, 2, 3, 4, 5, 6, 7, 100\}$. Choose values of θ between 0 and 30.
b) What is the value of θ at the minimum, referred to as $\hat{\theta}^{LS}$.

2.4 The L_1 loss function for the location problem is given by

$$L_1(\theta) = \sum_{i=1}^{n} |x_i - \theta|.$$

The gradient function is

$$L_1'(\theta) = -\sum_{i=1}^{n} \text{sgn}(x_i - \theta).$$

a) Plot the loss function and the gradient function for the dataset $\boldsymbol{x} = \{1, 2, 3, 4, 5, 6, 7, 100\}$. Choose values of θ between 0 and 10.
b) What is the value of θ at the minimum, referred to as $\hat{\theta}^{LAD}$.

2.5 The Huber loss function for the location problem is given by

$$H(\theta) = \sum_{i=1}^{n} \rho_H(x_i - \theta),$$

where

$$\rho_H(\theta) = \begin{cases} (x_i - \theta)^2/2 & \text{if } |(x_i - \theta)| \le \delta \\ \delta(|x_i - \theta| - \frac{\delta}{2}) & \text{if } |(x_i - \theta)| > \delta \end{cases}.$$

The gradient function is

$$H'(\theta) = \sum_{i=1}^{n} \psi_H(x_i - \theta).$$

where

$$\psi_H(\theta) = \begin{cases} (x_i - \theta) & \text{if } |(x_i - \theta)| \le \delta \\ -\delta & \text{if } (x_i - \theta) > +\delta \\ +\delta & \text{if } (x_i - \theta) < -\delta \end{cases}$$

a) Set $\delta = 1$ and plot the functions $\rho_H(\theta)$, $\psi_H(\theta)$ and $\psi_H'(\theta)$ as a function of $r_i = x_i - \theta$. Is the function convex?
b) Plot $H(\theta)$ as a function of θ for the dataset $\boldsymbol{x} = \{1, 2, 3, 4, 5, 6, 7, 100\}$. Choose values of θ between 0 and 10.
c) Plot $H'(\theta)$ as a function of θ for the dataset $\boldsymbol{x} = \{1, 2, 3, 4, 5, 6, 7, 100\}$. Choose values of θ between 0 and 10.
d) Compute the Huber estimate of the location parameter $\hat{\theta}^{H}$ for the dataset $\boldsymbol{x} = \{1, 2, 3, 4, 5, 6, 7, 100\}$.
e) Repeat part (d) with $\delta = 10$ and $\delta = 0.1$. Which is the most suitable value of δ for this dataset among $\delta = 1$, $\delta = 10$, and $\delta = 0.1$.

2.6 This problem concerns the use of gradient descent in machine learning. Assume that the objective is to minimize the L_2 loss function for a simple linear regression problem as follows:

$$\hat{\boldsymbol{\beta}}^{\mathrm{LS}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^2}{\operatorname{argmin}} \left[\frac{1}{2} \sum_{i=1}^{n} (y_i - \boldsymbol{x}_i^{\top} \boldsymbol{\beta})^2 \right].$$

What are the update equations for the least squares estimator? Note that it will take the form of:

$$\beta_j^{t+1} = \beta_j^t - \ell \times \frac{\partial f(\boldsymbol{\beta})}{\partial \beta_j}$$

where $j = 0$ or 1, and ℓ is a predefined learning rate and $f(\boldsymbol{\beta})$ is the loss function.

2.7 (PROJECT) In this project, you will solve for the slope and intercept of a dataset using robust regression with Huber's method.

a) Import the needed libraries.

```
import numpy as np
from sklearn.linear_model import HuberRegressor
import matplotlib.pyplot as plt
```

b) Create data similar to the telephone data set.

```
np.random.seed(0)
X = np.linspace(0, 23, 24).reshape(-1, 1)
y = np.array([ 0.44,  0.47,  0.47,  0.59,  0.66, \
              0.73,  0.81,  0.88,  1.06, 1.2 , \
              1.35,  1.49,  1.61,  2.12, 11.9 , \
              12.4 , 14.2 , 15.9 , 18.2 , 21.2 , \
              4.3 ,  2.4 ,  2.7 ,  2.9 ])
```

c) Fit the Huber regressor using the data.

```
huber = HuberRegressor(epsilon=4.0)
huber.fit(X, y)
slope = huber.coef_[0]
intercept = huber.intercept_
print("Slope=", f"{slope:.1f}","Intercept=",f"{intercept:.1f}")
y_pred = huber.predict(X)
```

d) Plot the results.

```
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, y_pred, color='red', \
    label='Huber Regressor')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Huber Regressor with Outliers')
plt.show()
```

e) Using $\delta = 4, 1.345$, and 0.1, plot the results (note: `epsilon` is the δ parameter). Which value of δ produces the robust solution? Explain why $\delta = 0.1$ might be a problem" (Hint: does it make Huber closer to L_1 or L_2?).

References

Montgomery, D.C., Peck, E.A., and Vining, G.G. (2012). *Introduction to Linear Regression Analysis*, 4e. Wiley.
Huber, P.J. and Ronchetti, E.M. (2009). *Robust Statistics*, 2e. John Wiley and Sons.

3

The Log-Cosh Loss Function

3.1 Introduction

In the previous chapter, we introduced the L_1 loss function (more commonly known as the least absolute deviation (LAD)) and the Huber loss function as our first encounter with robust estimators. In this chapter, we explore a more recent entry into the robust category, the log-cosh loss function, which is central to the robust methods to be presented in this book. This loss function belongs to the class of associated robust estimators because, like Huber, it also prefers solutions in the vicinity of the median rather than the mean. Another view is that it is more tolerant to outliers in the dataset because the family of associated estimators is related to the L_1 estimator. One standout feature is that it is a single, continuously differentiable function that is parametric in nature and therefore possesses all the desirable properties of estimators such as convexity, consistency and asymptotic normality. This is in contrast to the Huber loss which has multiple segments and a discontinuous second derivative.

In this chapter, we demonstrate that the log-cosh loss function is superior to other robust estimators for machine learning. The chapter provides a detailed and thorough treatment of the log-cosh loss function by deriving it from first principles, starting with its distribution. Then, we will study properties such as the bias, variance, and standard errors of estimation. We will also explore various extensions to the basic log-cosh loss using the location and scale parameters. Finally, the general M-estimator form of the log-cosh loss will be shown to be a very powerful tool for robust machine learning.

3.2 An Intuitive View of Log-Cosh

In order to gain an intuitive understanding of log-cosh, we need to go back to the L_1 loss function and identify its primary weakness. Minimization of this loss function results in the median as the estimate, dividing the data in half. However, it has a kink in the loss term which leads to a discontinuous first derivative and an undefined second derivative.[1] This implies that the optimization of the loss function may not produce a unique solution, and the solution may require methods other than gradient descent. In fact, one of the reasons that L_2 is in widespread use today is that the L_1 loss requires linear programming

1 More accurately, the second derivative is an impulse function at the origin and zero elsewhere.

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.

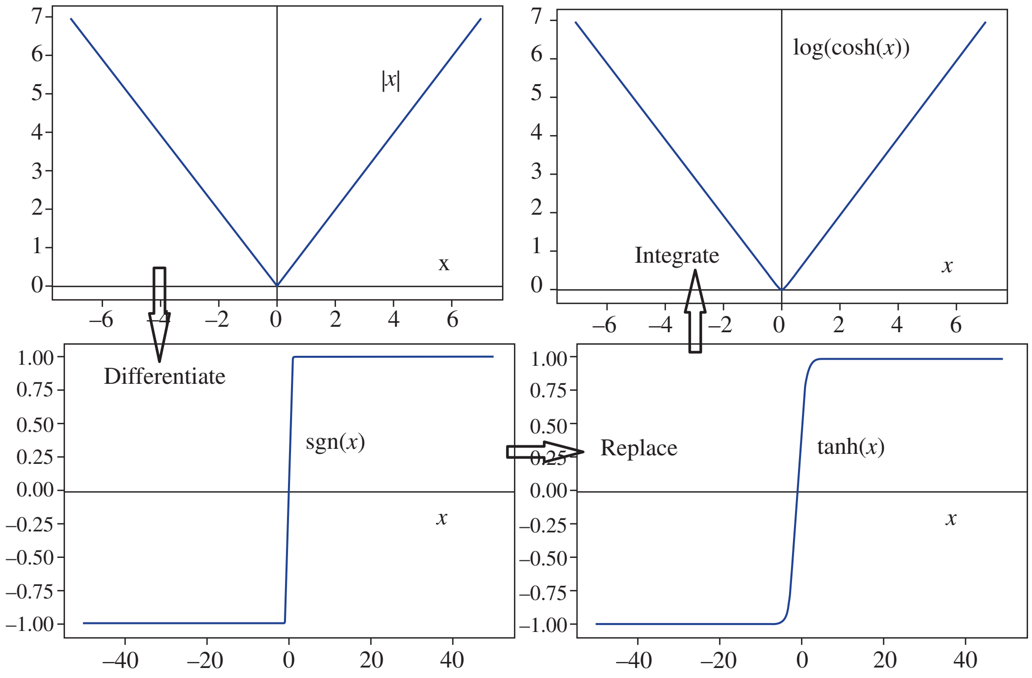


Figure 3.1 Developing a continuous L_1 function using log-cosh.

methods for optimization. If an alternative can be found that retains the robust property of L_1 and does not suffer from discontinuity issues while offering a unique solution to the minimization problem using gradient descent, then it will be better suited for machine learning.

The objective is now clear: we seek a loss term that has continuous first and second derivatives to replace the L_1 function. Consider how one might do this starting with the L_1 loss term, as shown in Figure 3.1 on the top left panel. The absolute value loss term, $|x|$, is clearly piecewise continuous with a V-shaped kink. When we take its derivative, we obtain $\text{sgn}(x)$ which is discontinuous at $x = 0$ as seen in the bottom left panel. This discontinuity will invite a certain amount of trouble in gradient-based methods. What if we were to replace the discontinuous derivative function with a continuous function, such as the $\tanh(x)$ function? Since $\tanh(x)$ is continuous in all regions and ranges from -1 to 1, it appears to be a very good option to replace $\text{sgn}(x)$. This is shown in the bottom right panel. The two functions, $\text{sgn}(x)$ and $\tanh(x)$, look about the same but we know that one is discontinuous and the other is continuous.

Having found a suitable replacement for the derivative, all that remains is to obtain the loss term by integrating the $\tanh(x)$ function, whereby we obtain $\log(\cosh(x))$. This is shown in the top right panel. It is a continuous function that is smooth around the origin. That is, it has the characteristic V-shape of L_1 but does not have a kink (although it may appear to have one in the figure). Therefore, rather than using $|x|$ as the loss term, we can replace it with $\log(\cosh(x))$ and hopefully all of the issues associated with L_1 can be resolved in this manner. This is the key motivation for the so-called log-cosh (LC) loss function.

3.3 Hyperbolic Functions

The log-cosh loss is built using hyperbolic functions for the distributions, loss functions and derivatives. As such, it is necessary to refresh our memories about the definitions and identities associated with hyperbolic trigonometric functions. We review the equations and some of their properties below.

Recall the three main hyperbolic functions are

$$\sinh(x) = \frac{e^x - e^{-x}}{2},$$
$$\cosh(x) = \frac{e^x + e^{-x}}{2},$$

and

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

We also have the reciprocal hyperbolic functions given by

$$\operatorname{csch}(x) = \frac{1}{\sinh(x)} = \frac{2}{e^x - e^{-x}},$$
$$\operatorname{sech}(x) = \frac{1}{\cosh(x)} = \frac{2}{e^x + e^{-x},}$$

and

$$\coth(x) = \frac{1}{\tanh(x)} = \frac{e^x + e^{-x}}{e^x - e^{-x}}.$$

Some useful properties are as follows:

$$\sinh(-x) = -\sinh(x),$$

$$\cosh(-x) = \cosh(x),$$

and

$$\tanh(-x) = -\tanh(x).$$

Other useful equations include the following:

$$\cosh^2(x) - \sinh^2(x) = 1,$$

and

$$\int_{-\infty}^{x} \frac{1}{\cosh(x)} = \arctan\left[\sinh(x)\right] + c.$$

3.4 M-Estimation

The log-cosh loss function in machine learning can be used to construct an M-estimator (refer to Chapter 2). The loss function associated with an M-estimator has the general form

$$\text{M-loss} = \sum_{i=1}^{n} \rho(x_i). \tag{3.4.1}$$

Any desired function may be used for the loss term, $\rho(x_i)$, as long as it satisfies some minimal set of conditions. Say we are interested in estimating some quantity $\boldsymbol{\beta}$ which is associated with the explanatory variables of an $n \times (d+1)$ design matrix $\boldsymbol{X}$ and the response vector $\boldsymbol{y}$. Then the M-estimator of $\boldsymbol{\beta}$ is defined as,

$$\hat{\boldsymbol{\beta}} = \underset{\beta \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(r_i), \tag{3.4.2}$$

where $r_i = y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}$ is the residual, for $i = 1, \dots, n$. The solution is typically carried out using numerical optimization employing some form of gradient descent in the context of machine learning.

For the log-cosh loss, we already determined the loss term to be

$$\rho_{\mathrm{LC}}(x) = \log(\cosh(x)). \tag{3.4.3}$$

Then, it follows that

$$\psi_{\mathrm{LC}}(x) = \rho'_{\mathrm{LC}}(x) = \tanh(x) \tag{3.4.4}$$

and

$$\psi'_{\mathrm{LC}}(x) = \rho''_{\mathrm{LC}}(x) = \operatorname{sech}^2(x). \tag{3.4.5}$$

Hence, the first and second derivatives can be easily derived and all functions are continuous. This is important for machine learning applications. We desire functions that are continuously differentiable to avoid numerical issues in convergence during gradient descent.

The three functions are illustrated in Figure 3.2. Since $\psi'_{\mathrm{LC}}(x)$ is always positive, this implies that the loss function is strictly convex and optimizing it provides a unique solution (unlike L_1). The corresponding log-cosh (LC) estimator, given by

$$\hat{\boldsymbol{\beta}}^{\mathrm{LC}} = \underset{\beta \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{n} \log(\cosh(r_i)), \tag{3.4.6}$$

can be solved in a straightforward manner using convex optimization due to the fact that it is globally convex. That is, the second derivative of $\rho_{\mathrm{LC}}(x)$ is always non-negative which is the condition for convexity of the loss function.

3.4.1 Asymptotic Behavior

The asymptotic behavior of $\log(\cosh(x))$ involves analysis as $x \to \infty$, $x \to -\infty$ and $x \to 0$. Recall that

$$\cosh(x) = \frac{e^x + e^{-x}}{2}.$$

Starting with the first case, as $x \to \infty$, $\cosh(x) \to e^x/2$. Then, in the limit,

$$\log(\cosh(x)) \approx x - \log(2). \tag{3.4.7}$$

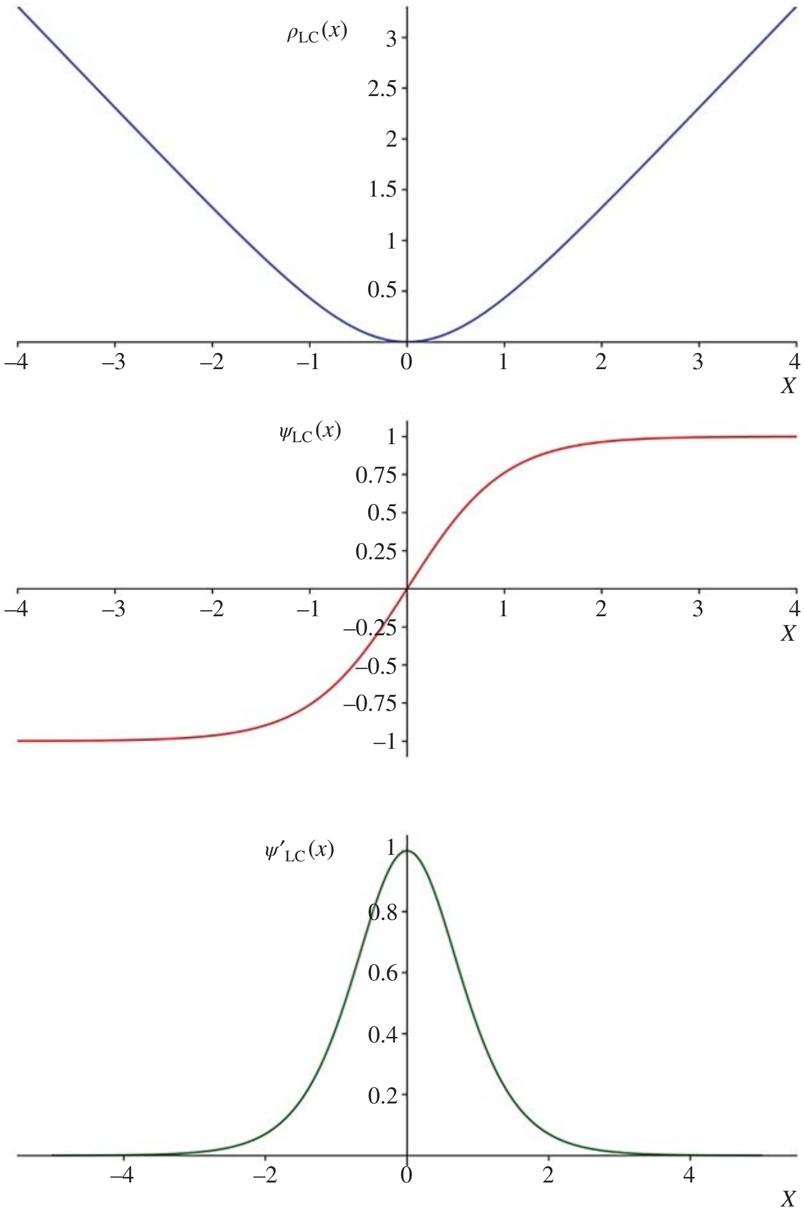


Figure 3.2 The M-estimator functions $\rho(x), \psi(x)$, and $\psi'(x)$ for log(cosh(x)).

In the second case, as $x \to -\infty$, $\cosh(x) \to e^{-x}/2$ and, in the limit,

$$\log(\cosh(x)) \approx -x - \log(2). \tag{3.4.8}$$

Therefore, far from $x = 0$, the function behaves as

$$\log(\cosh(x)) \approx |x| - \log(2) \tag{3.4.9}$$

which is like L_1 except for the constant term (which is of no consequence).

Finally, consider its behavior around the origin. As $x \to 0$, we can take a Taylor series expansion at $x = 0$ as follows:

$$\log(\cosh(x)) = \frac{x^2}{2} - \frac{x^4}{12} + \frac{x^6}{45} + O(x^8) \tag{3.4.10}$$

and therefore it is, to first order, like L_2. That is, for small x values,

$$\log(\cosh(x)) \approx \frac{x^2}{2}. \tag{3.4.11}$$

Since it behaves like L_2 close to the origin and L_1 far from the origin, one can view it as the best of both worlds. This single function can replace the three-segment function used in the Huber loss. For robust regression, the log-cosh loss should be considered as the function of choice.

3.4.2 Linear Regression Using Log-Cosh

Before proceeding further into a theoretical study, it is instructive to solve a simple linear problem using the log-cosh loss to demonstrate that it is indeed a robust estimator. The LC estimator is given by

$$\hat{\boldsymbol{\beta}}^{\mathrm{LC}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{n} \log(\cosh(r_i)), \tag{3.4.12}$$

where $r_i = y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}$ are the residuals. The LC loss function is

$$\text{LC loss} = \sum_{i=1}^{n} \log(\cosh(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta})). \tag{3.4.13}$$

If we take the derivative of the loss function w.r.t. β_j and use it in the gradient descent iterative equation, we obtain:

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \ell \sum_{i=1}^{n} \tanh(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta})(-x_{ij}), \tag{3.4.14}$$

where ℓ is the learning rate. At each iteration, t, we simply update each $\beta_j^{(t)}$, $j = 0, \dots, d$ with the gradient term to produce $\beta_j^{(t+1)}$ until convergence. The obvious point of comparison is with the least squares (LS) estimator which was detailed in the previous chapter. The LS estimator is given by:

$$\hat{\boldsymbol{\beta}}^{\mathrm{LS}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{n} \frac{r_i^2}{2}. \tag{3.4.15}$$

Figure 3.3 shows the results of linear regression using LS and LC on the Belgium telephone dataset of Chapter 2. Note that most of the points occur in the lower portion of the plot. The ones in the upper

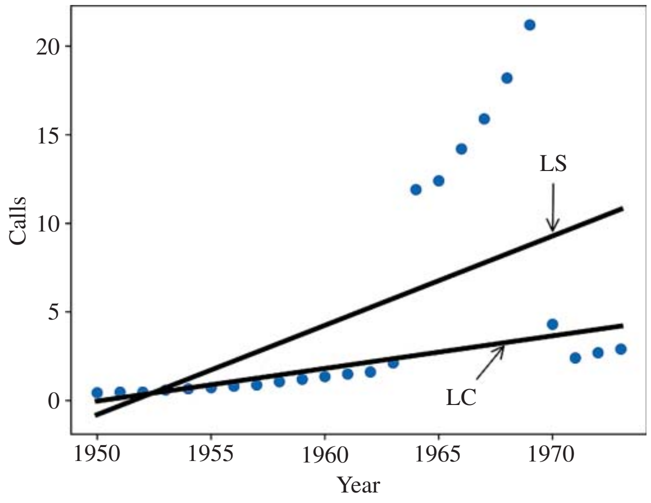


Figure 3.3 Comparing LS and LC using the Belgium telephone dataset of Chapter 2, Section 2.9.1. LS is affected by outliers while LC is tolerant of outliers.

portion are outliers due to differences in measurement techniques. In the case of LS, it is highly sensitive to these outliers and fails to produce an acceptable model. We notice that the line is greatly affected by the outliers, whereas the LC regression line fits the true data and effectively ignores the outliers. This is due to the fact that the LS method, which uses the L_2 loss function, seeks the mean of the data (meaning the line that averages out the errors to 0), while the LC method seeks the median (meaning the line that has the same number of points above and below it). We can observe that LC is ignoring the outliers and capturing the essence of the inlier data. This provides the needed evidence that log-cosh is a robust estimator. With that knowledge, we can now go ahead and understand the detailed mathematics and characteristics of this loss function.

3.5 Deriving the Distribution for Log-Cosh

It is important to understand the log-cosh loss function from both theoretical and foundational perspectives in order to appreciate its true value. Therefore, we need to study probability-based statistical models associated with this loss function and relate the results to the underlying distributions. In this section, we show how to derive this loss function and then provide a more mathematically rigorous view of its origin.

The log-cosh loss function can be derived from a probability density function (PDF) using the maximum-likelihood procedure. The challenge is to figure out which probability distribution to use as a starting point when only the loss function is known. The goal in this section is to find the associated PDF, $f_X(x)$. To do this, consider the relationship between $f_X(x)$ and one term of the loss function, $\rho(x)$. Since we know that the maximum-likelihood procedure (after constants are removed) produces

$$\rho(x) = -\log f_X(x),$$

then it follows that the **reverse MLE procedure** would require that

$$f_X(x) = \frac{1}{\kappa} e^{-\rho(x)}, \tag{3.5.1}$$

where κ is a normalization coefficient. Here, $f_X(x)$ is the PDF we seek. We should keep in mind that the distribution associated with a loss function must be centered at 0; in other words, we expect it to be symmetric about 0. Hence, the mean of the distribution is 0, or mathematically $\mathbb{E}[X] = 0$. The reverse MLE procedure can be applied to well-known loss functions, such as L_1, L_2, and Huber. We illustrate the steps for L_1 and L_2 in the following examples and leave the derivation of the Huber distribution as an exercise for the reader.

Example: To find the distribution associated with the L_1 loss function, we begin with the term

$$\rho_{L1}(x) = |x|.$$

Then we apply the reverse MLE rule above to obtain

$$f_X(x) = \frac{1}{\kappa} e^{-|x|}.$$

We can find κ as follows:

$$\kappa = \int_{-\infty}^{\infty} e^{-|x|} dx = 2.$$

Hence, the basic form of the distribution for the L_1 loss is

$$f_X(x) = \frac{1}{2} e^{-|x|}, \tag{3.5.2}$$

which is called the *Laplace distribution*. □

Example: To find the distribution associated with the L_2 loss function, we begin with the term

$$\rho_{\mathrm{L2}}(x) = x^2/2.$$

Then we apply the reverse MLE rule above to obtain

$$f(x) = \frac{1}{\kappa} e^{-x^2/2}.$$

We can find κ as follows:

$$\kappa = \int_{-\infty}^{\infty} e^{-x^2/2} dx = \sqrt{2\pi}.$$

Hence, the basic form of the distribution for the L_2 loss function is

$$f_X(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, \tag{3.5.3}$$

which is, of course, the Gaussian distribution. □

If we now apply the reverse MLE rule to the log-cosh loss, we start with

$$\rho_{\mathrm{LC}}(x) = \log(\cosh(x)).$$

Then we apply the reverse MLE rule above to obtain

$$f(x) = \frac{1}{\kappa} e^{-\rho_{\mathrm{LC}}(x)} = \frac{1}{\kappa} e^{-\log(\cosh(x))} = \frac{1}{\kappa}\frac{1}{\cosh(x)}.$$

We can find κ as follows:

$$\kappa = \int_{-\infty}^{\infty} \frac{1}{\cosh(x)} dx = \pi.$$

Hence, the basic form of the distribution associated with log-cosh is

$$f_X(x) = \frac{1}{\pi}\frac{1}{\cosh(x)}. \tag{3.5.4}$$

The resulting PDF is sometimes referred to as the *hyperbolic secant distribution* but we will use the term **Cosh distribution** to keep in mind its connection to the log-cosh loss function.

The cumulative distribution function (CDF) is obtained by integrating the PDF function up to some desired point, x, as follows:

$$F_X(x) = \int_{-\infty}^{x} f_X(x) dx. \tag{3.5.5}$$

For the Cosh distribution, the CDF is found using

$$F_X(x) = \int_{-\infty}^{x} \frac{1}{\pi}\frac{1}{\cosh(x)} dx. \tag{3.5.6}$$

Then, integrating and setting the constant of integration to 1/2, the CDF is given by:

$$F_X(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\sinh(x)\right). \tag{3.5.7}$$

The CDF is plotted in Figure 3.4(a) along with the PDF in Figure 3.4(b).

More generally, we have a location-scale family of distributions of the form

$$f_X(x;\theta,\sigma) = \frac{1}{\pi\sigma\,\cosh(\frac{x-\theta}{\sigma})}, \tag{3.5.8}$$

where θ is the location parameter and σ is the scale parameter (not to be confused with the standard deviation). We note here that

$$\mathbb{E}[X] = \theta \quad \text{and} \quad \mathbb{E}[X^2] = \frac{\pi^2\sigma^2}{4} + \theta^2$$

so the asymptotic variance of the distribution is

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \frac{\pi^2\sigma^2}{4}. \tag{3.5.9}$$

The location and scale parameters can be estimated from the data (i.e. residuals) as

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \hat{\sigma} = 2\sqrt{\widehat{\mathrm{Var}(X)}}/\pi, \tag{3.5.10}$$

respectively, where

$$\widehat{\mathrm{Var}(X)} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\theta})^2. \tag{3.5.11}$$

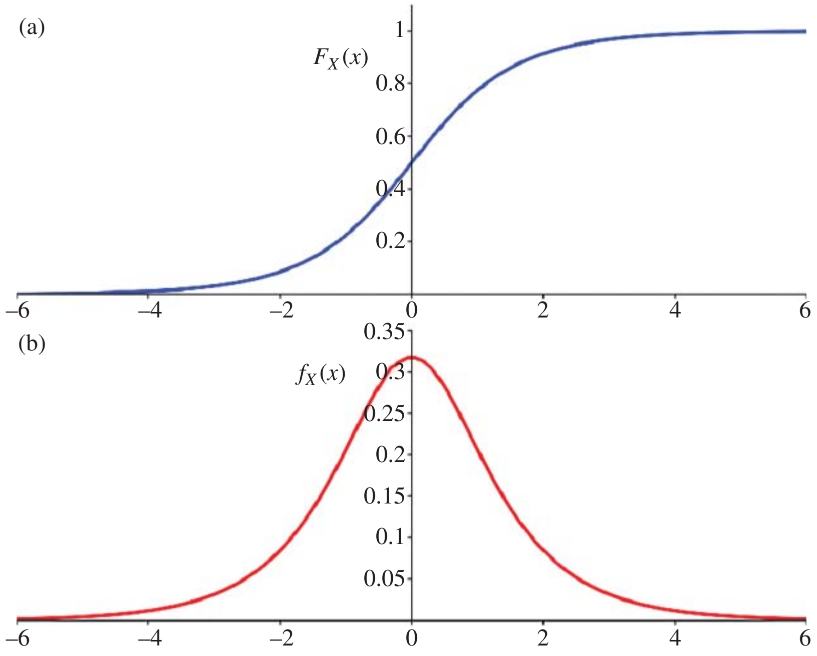


Figure 3.4 The (a) CDF and (b) PDF plots of the Cosh distribution.

For the CDF, we have that

$$F_X(x;\theta,\sigma) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\sinh\left(\frac{x-\theta}{\sigma}\right)\right). \tag{3.5.12}$$

Typically, the location is 0 for distributions used to derive loss functions. Hence, for $\theta = 0$, the CDF is given by

$$F_X(x;\sigma) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\sinh\left(\frac{x}{\sigma}\right)\right). \tag{3.5.13}$$

It is interesting to compare the four main distributions we have mentioned thus far: Gaussian, Laplace, Huber, and Cosh. This is provided in Figure 3.5. Both Cosh and Huber ($\delta = 1.345$) look similar to one another, whereas Laplace and Gaussian are quite different. In robust statistics, the Laplace distribution is considered to have heavy tails. It is outlier-friendly because heavy tails imply a higher probability for the occurrence of outliers. That is, outliers are easily accommodated by the PDF of the Laplace distribution without affecting the model greatly, specifically, the mean and the variance. This makes it more robust to outliers. In contrast, the Gaussian distribution has light tails so outliers can have a significant effect on the mean and variance. Therefore, the Gaussian distribution is less suitable for outliers. Notice that the Cosh distribution has heavier tails which is similar to the Laplace distribution. In the case of Huber, the moderate tails are not as heavy as Cosh due to the fact that $\delta = 1.345$. However, it can be made heavier by reducing the value of δ, or lighter by increasing its value.

The importance of the different distributions should now be clear. Each one has a different sensitivity to outliers. When we select a certain loss function, we are forcing the residuals of the regression to follow

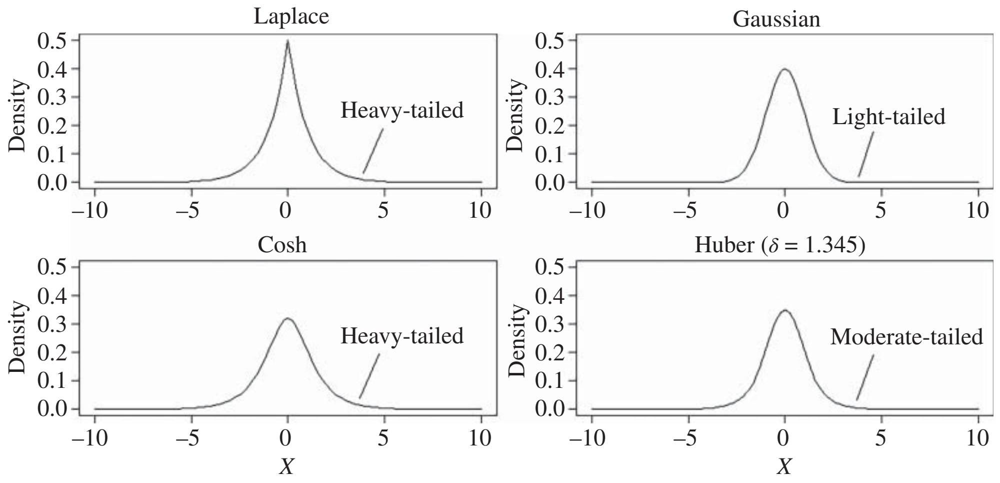


Figure 3.5 Distributions for Laplace, Gaussian, Cosh, and Huber ($\delta = 1.345$).

the associated probability distribution. Specifically, if we choose the L_2 loss function, the residuals will try to follow a Gaussian distribution (Eq. (3.5.3)). If we choose L_1, it will try to follow the Laplace distribution (Eq. (3.5.2)). A selection of log-cosh will follow the Cosh distribution (Eq. (3.5.4)), and a selection of Huber will follow some form of the Huber distribution, depending on δ. The implications of this are profound in the sense that if we have outliers, we should not choose the L_2 loss function (which follows a Gaussian distribution) as it is highly sensitive to outliers. This was shown earlier in Figure 3.3. Fundamentally, this is why we seek robust loss functions that are more suitable for datasets with outliers.

3.6 Standard Errors for Robust Estimators

Standard error (s.e.) is a measure of the uncertainty around an estimate (see Montgomery et al. 2012). Since we know that different samples of a population will produce different estimates, it is desirable to get a handle on the degree of this uncertainty. In fact, it is possible to show that the estimates from a log-cosh estimator are asymptotically normal, meaning they follow a Gaussian distribution.[2] This makes life a lot easier since we can follow the usual procedures to obtain the standard errors, which is an estimate of the standard deviation of each parameter.

Consider the multiple linear regression model in matrix form as follows:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{3.6.1}$$

where $\boldsymbol{y} = (y_1, \dots, y_n)^\top$ is a response vector, $\boldsymbol{X}$ is an $n \times (d+1)$ design matrix of constants with $X_1 = (1, \dots, 1)^\top$, and $\boldsymbol{\varepsilon} = (\varepsilon_1, \dots, \varepsilon_n)^\top$ is the error term. Regression parameters $\boldsymbol{\beta} = (\beta_0, \dots, \beta_d)^\top$ are the $d + 1$ coefficients of interest. Here we assume $\boldsymbol{\varepsilon}$ has some known distribution with mean $\boldsymbol{0} = (0, \dots, 0)^\top$.

2 To be clear, the residuals follow the Cosh distribution while the estimates follow a Gaussian distribution.

In a $(d+1)$-dimensional space, each estimate will have its own s.e. value. Least squares regression has a well-known formula for the **covariance matrix** given by

$$\text{Var}(\hat{\boldsymbol{\beta}}^{\text{LS}}) = \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}. \tag{3.6.2}$$

The parameter variances lie along the diagonal and their square root produces the standard errors as follows:

$$\text{s.e.}(\hat{\boldsymbol{\beta}}^{\text{LS}}) = \sqrt{\text{diag}(\hat{\sigma}^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1})}, \tag{3.6.3}$$

where the unbiased estimator of σ^2 is given by

$$\hat{\sigma}^2 = \frac{1}{n-(d+1)}\sum_{i=1}^{n}(r_i - \bar{r})^2. \tag{3.6.4}$$

Hence, the residuals are needed to obtain the standard errors.

There is an issue with this formulation when outliers are present. Specifically, the value of $\hat{\sigma}^2$ will be inflated due to the fact that outliers will contribute large residuals to the sum and each one will be squared to increase their values even further.

In the case of log-cosh, a different approach is needed but with the same general formulation. For large n, we treat the estimates as being normally distributed with

$$\text{Var}(\hat{\boldsymbol{\beta}}^{\text{LC}}) = v^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1} \tag{3.6.5}$$

and then define the term v^2 as follows (see Maronna et al. 2019)[3]:

$$v^2 = s^2\frac{\mathbb{E}[\psi_{\text{LC}}(r/s)^2]}{\mathbb{E}[\psi'_{\text{LC}}(r/s)]^2} \tag{3.6.6}$$

which has an unbiased estimate that can be obtained by replacing expectations with averages

$$\hat{v}^2 = \hat{s}^2\frac{\frac{1}{n}\sum\psi_{\text{LC}}(r_i/\hat{s})^2}{\left[\frac{1}{n}\sum\psi'_{\text{LC}}(r_i/\hat{s})\right]^2}\frac{n}{n-d-1}. \tag{3.6.7}$$

In the above, the residuals are $r_i = y_i - \boldsymbol{x}_i^\top\hat{\boldsymbol{\beta}}^{\text{LC}}$ and, to provide a robust estimate of v, we estimate s using

$$\hat{s} = \text{MADN}(\boldsymbol{r}), \tag{3.6.8}$$

where the function MADN($\boldsymbol{x}$) was defined in Chapter 1, Eq. (1.4.6). To compute the terms under the summations, recall that

$$\psi_{\text{LC}}(x) = \rho'_{\text{LC}}(x) = \tanh(x) \tag{3.6.9}$$

and

$$\psi'_{\text{LC}}(x) = \rho''_{\text{LC}}(x) = \text{sech}^2(x). \tag{3.6.10}$$

These functions must be applied as required in Eq. (3.6.7). The s.e.'s are computed as the square root of the diagonal elements of the covariance matrix, $\hat{v}^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$.

3 For a derivation, see Maronna et al. (2019, pp. 99–100).

3.6.1 Example: Swiss Fertility Dataset

We now employ the well-known Swiss Fertility dataset to compute estimates and standard errors for LS and LC for comparison purposes. The Swiss Fertility data[4] itself is provided in Table 3.1 and the variable definitions and their abbreviations are provided in Table 3.2.

The dataset has 5 variables that may influence the fertility rates (`F`) in a 1888 study with 47 observations. The abbreviations of the five variables are `I`, `Ex`, `Ed`, `A`, and `C` and the objective is to find the coefficient

Table 3.1 Swiss fertility dataset (`F` = Fertility, `A` = Agriculture, `Ex` = Examination, `Ed` = Education, `C` = Catholic, and `I` = Infant Mortality).

Region	F	A	Ex	Ed	C	I	Region	F	A	Ex	Ed	C	I
Courtelary	80.20	17.00	15	12	9.96	22.20	Delemont	83.10	45.10	6	9	84.84	22.20
Franches-Mnt	92.50	39.70	5	5	93.40	20.20	Moutier	85.80	36.50	12	7	33.77	20.30
Neuveville	76.90	43.50	17	15	5.16	20.60	Porrentruy	76.10	35.30	9	7	90.57	26.60
Broye	83.80	70.20	16	7	92.85	23.60	Glane	92.40	67.80	14	8	97.16	24.90
Gruyere	82.40	53.30	12	7	97.67	21.00	Sarine	82.90	45.20	16	13	91.38	24.40
Veveyse	87.10	64.50	14	6	98.61	24.50	Aigle	64.10	62.00	21	12	8.52	16.50
Aubonne	66.90	67.50	14	7	2.27	19.10	Avenches	68.90	60.70	19	12	4.43	22.70
Cossonay	61.70	69.30	22	5	2.82	18.70	Echallens	68.30	72.60	18	2	24.20	21.20
Grandson	71.70	34.00	17	8	3.30	20.00	Lausanne	55.70	19.40	26	28	12.11	20.20
La Vallee	54.30	15.20	31	20	2.15	10.80	Lavaux	65.10	73.00	19	9	2.84	20.00
Morges	65.50	59.80	22	10	5.23	18.00	Moudon	65.00	55.10	14	3	4.52	22.40
Nyone	56.60	50.90	22	12	15.14	16.70	Orbe	57.40	54.10	20	6	4.20	15.30
Oron	72.50	71.20	12	1	2.40	21.00	Payerne	74.20	58.10	14	8	5.23	23.80
Paysd'enhaut	72.00	63.50	6	3	2.56	18.00	Rolle	60.50	60.80	16	10	7.72	16.30
Vevey	58.30	26.80	25	19	18.46	20.90	Yverdon	65.40	49.50	15	8	6.10	22.50
Conthey	75.50	85.90	3	2	99.71	15.10	Entremont	69.30	84.90	7	6	99.68	19.80
Herens	77.30	89.70	5	2	100.00	18.30	Martigwy	70.50	78.20	12	6	98.96	19.40
Monthey	79.40	64.90	7	3	98.22	20.20	St Maurice	65.00	75.90	9	9	99.06	17.80
Sierre	92.20	84.60	3	3	99.46	16.30	Sion	79.30	63.10	13	13	96.83	18.10
Boudry	70.40	38.40	26	12	5.62	20.30	La Chauxdfnd	65.70	7.70	29	11	13.79	20.50
Le Locle	72.70	16.70	22	13	11.22	18.90	Neuchatel	64.40	17.60	35	32	16.92	23.00
Val de Ruz	77.60	37.60	15	7	4.97	20.00	ValdeTravers	67.60	18.70	25	7	8.65	19.50
V. De Geneve	35.00	1.20	37	53	42.34	18.00	Rive Droite	44.70	46.60	16	29	50.43	18.20
Rive Gauche	42.80	27.70	22	29	58.33	19.30							

4 This dataset is part of the R environment in the *datasets* package but is included here for the reader to create the dataset manually as a file named `swiss.csv` for use in this and later chapters.

Table 3.2 Swiss fertility dataset definitions.

Feature	Symbol	Meaning
Region		French-speaking provinces of Switzerland in 1888
Fertility	`F`	Ig, a common standardized fertility measure
Agriculture	`A`	% of males involved in agriculture as occupation
Examination	`Ex`	% draftees receiving highest mark on army examination
Education	`Ed`	% education beyond primary school for draftees
Catholic	`C`	% Catholic (as opposed to Protestant)
Infant Mortality	`I`	Live births who live less than 1 year

Table 3.3 Swiss fertility estimates and standard errors for least squares (LS) and log-cosh (LC).

Feature	$\hat{\beta}^{\text{LS}}$	s.e. ($\hat{\beta}^{\text{LS}}$)	$\hat{\beta}^{\text{LC}}$	s.e. ($\hat{\beta}^{\text{LC}}$)
(Intercept)	66.92	10.706	63.12	10.556
Agriculture	−0.17	0.070	−0.20	0.069
Examination	−0.26	0.253	−0.27	0.250
Education	−0.87	0.183	−0.89	0.180
Catholic	0.10	0.035	0.10	0.036
Infant Mortality	1.08	0.381	1.40	0.376

values or estimates for these five parameters and the intercept in order to predict the response variable `F`. Hence, $n = 47$ and $d = 5$.

We first note that all the data in Table 3.1 lies in the range 0–100 so scaling is not essential and the estimates can be reasonably interpreted relative to one another. The parameter values from the LS and LC methods are shown in Table 3.3. We find that the LS and LC are similar, both for estimates and standard errors. This is due to the fact that there are no outliers in the Swiss dataset, as will be shown in Chapter 4. This example demonstrates that robust estimates and standard errors are similar to least squares when there are no outliers.

3.6.2 Example: Boston Housing Dataset

The Boston Housing dataset is a much larger dataset compared to the Swiss dataset. The linear regression problem is to build a model that predicts the median home price (`MEDV`) given data associated with 13 explanatory variables. In this dataset, approximately 10% of the observations can be categorized as outliers (see Chapter 4). Therefore, LS will have difficulty in obtaining reasonable estimates and standard errors. In Table 3.4, the estimates and s.e. values are shown for LS and LC. If the second and fourth columns are compared, the standard errors for LS are much higher than that of LC. This is due to the fact

Table 3.4 Boston Housing data estimates and standard errors for least squares (LS) and log-cosh (LC).

Feature	$\hat{\beta}^{\text{LS}}$	s.e. $(\hat{\beta}^{\text{LS}})$	$\hat{\beta}^{\text{LC}}$	s.e. $(\hat{\beta}^{\text{LC}})$
(Intercept)	36.494	5.102	15.623	3.521
CRIM	−0.121	0.033	−0.142	0.023
ZN	0.046	0.014	0.037	0.009
INDUS	0.020	0.062	0.017	0.042
CHAS	2.692	0.862	1.452	0.594
NOX	−17.802	3.824	−9.321	2.636
RM	3.813	0.418	5.258	0.288
AGE	0.001	0.013	−0.028	0.009
DIS	−1.482	0.199	−1.026	0.138
RAD	0.305	0.066	0.182	0.046
TAX	−0.012	0.004	−0.010	0.003
PTRATIO	−0.954	0.131	−0.747	0.090
B	0.009	0.003	0.011	0.002
LSTAT	−0.526	0.051	−0.302	0.035

that the computation of $\hat{\sigma}$ in the LS case will have large residuals which get squared thereby inflating all the s.e. values. Clearly the LC results are superior and they represent the data more accurately.

3.7 Statistical Properties of Log-Cosh Loss

In this section, we derive the maximum-likelihood estimator (MLE) to understand the statistical properties of the log-cosh estimator. For readers who prefer to skip the detailed mathematical explanations, you may proceed to the next section without loss of continuity in the overall subject matter. While it may not be critical to machine learning to understand this material, it is useful to know some of these key properties when comparing one estimator against another and to carry out further mathematical analysis.

3.7.1 Maximum-Likelihood Estimation

Let us begin by deriving the log-cosh estimator for a location parameter θ starting with the probability distribution. If $x_1, x_2, \dots, x_n$ are i.i.d. random variables drawn from the distribution,

$$f(x;\theta) = \frac{1}{\pi\,\sigma\cosh(\frac{x_i-\theta}{\sigma})}, \tag{3.7.1}$$

then the likelihood function is given by:

$$L(x_1, x_2, \dots, x_n;\theta) = \prod_{i=1}^{n} \frac{1}{\pi\,\sigma\cosh(\frac{x_i-\theta}{\sigma})}. \tag{3.7.2}$$

The negative log-likelihood expression is given by

$$-\ell(x_1, x_2, \dots, x_n; \theta) = n \log \pi\sigma + \sum_{i=1}^{n} \log\left(\cosh\left(\frac{x_i - \theta}{\sigma}\right)\right). \tag{3.7.3}$$

Assume we are interested in estimating θ. Then, after removing the constant term, the estimator $\hat{\theta}$ can be written as

$$\hat{\theta}^{\mathrm{LC}} = \operatorname{argmin}_{\theta\in\mathbb{R}} \sum_{i=1}^{n} \log\left(\cosh\left(\frac{x_i - \theta}{\sigma}\right)\right). \tag{3.7.4}$$

Since the log-cosh estimator was derived using MLE and assumes some mild regularity conditions, we find that asymptotic normality holds. That means that the uncertainty around the estimates will follow a Gaussian distribution. More formally (using the notation given in Section 2.5.2), as $n \to \infty$, the Central Limit Theorem tells us that

$$\sqrt{n}\left(\hat{\theta}_n^{\mathrm{LC}} - \theta\right) \xrightarrow{\mathcal{D}} \mathcal{N}\left(0, \mathrm{Var}\left(\hat{\theta}_n^{\mathrm{LC}}\right)\right), \tag{3.7.5}$$

where $\mathrm{Var}(\hat{\theta}_n^{\mathrm{LC}})$ is the asymptotic variance which can be derived[5] to obtain $2\sigma^2$. Hence, we find the distribution of $\hat{\theta}^{\mathrm{LC}}$ is

$$\hat{\theta}_n^{\mathrm{LC}} \sim \mathcal{N}\left(\theta, \frac{2\sigma^2}{n}\right). \tag{3.7.6}$$

We can also state that the MLE estimator of θ is consistent such that

$$\hat{\theta}_n^{\mathrm{LC}} \xrightarrow{\mathbb{P}} \theta \tag{3.7.7}$$

and that the asymptotic bias is

$$\mathrm{bias} = \mathbb{E}[\hat{\theta}_n^{\mathrm{LC}}] - \theta = 0.$$

That is, the log-cosh estimator of θ is unbiased. In summary, the log-cosh estimator has all the nice properties of least squares but is robust to outliers.

3.8 A General Log-Cosh Loss Function

There is one more improvement needed to the log-cosh loss to make it a superior robust method and that is the subject of this section. We begin with the standard loss term using M-estimation notation,

$$\rho(x) = \log(\cosh(x)), \tag{3.8.1}$$

and its derivative,

$$\psi(x) = \tanh(x). \tag{3.8.2}$$

5 For those with a statistics background, the asymptotic variance can be derived in a few steps as the reciprocal of the Fisher information.

The slope of the tanh(x) function around the origin is quite shallow. However, it can be made tunable by introducing a hyperparameter, a, as follows:

$$\rho(x; a) = \tanh(ax). \tag{3.8.3}$$

The new hyperparameter controls the steepness of the slope as the function transitions from −1 to 1. By integrating, we obtain

$$\rho(x; a) = \frac{1}{a}\log(\cosh(ax)). \tag{3.8.4}$$

This is the general loss term for log-cosh with a hyperparameter a which gives it more flexibility. This flexibility, in effect, controls the degree of robustness of the general log-cosh loss. We called this hyperparameter the *robustness coefficient*.

The general form of the loss function can be written as

$$J^{\mathrm{LC}}(x; a) = \sum_{i=1}^{n}\frac{1}{a}\log(\cosh(ax)) = \sum_{i=1}^{n}\rho(x; a), \tag{3.8.5}$$

where a is the robustness coefficient that can be adjusted as needed. We have been using $a = 1$ thus far in this chapter but we will find that most of the useful values are associated with $a > 1$.

The loss term $\rho(x; a)$ is shown for different values of a in Figure 3.6 in the top panel along with the L_1 loss term with the V-shape given in bold. The derivative is given by

$$\psi(x; a) = \tanh(ax)). \tag{3.8.6}$$

The effect of increasing a on the derivative is to sharpen the slope of the transition from −1 to 1 as shown in Figure 3.6 in the lower panel. The derivative of the L_1 loss is given in bold for comparison purposes.

The general log-cosh estimator is defined as

$$\hat{\boldsymbol{\beta}}^{\mathrm{LC}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{d+1}}{\operatorname{argmin}}\sum_{i=1}^{n}\frac{1}{a}\log(\cosh(ar_i)). \tag{3.8.7}$$

The importance of the robustness coefficient cannot be overstated. It determines whether the loss function is weakly robust, that is, slightly better than L_2, or strongly robust, that is, approaching L_1.

The distribution from which this loss function arises can be derived by running the MLE procedure in reverse, as shown earlier in Section 3.5. We first take one term of the summation and note that it was the result of the negative log-likelihood. Therefore, we should negate it to obtain

$$-\rho(x) = -\frac{1}{a}\log(\cosh(ax)). \tag{3.8.8}$$

Next we can move the leading coefficient inside the log expression

$$-\rho(x) = \log(\cosh{(ax)}^{-\frac{1}{a}}). \tag{3.8.9}$$

Then we exponentiate the expression to obtain

$$e^{-\rho(x)} = \frac{1}{(\cosh(ax))^{\frac{1}{a}}}. \tag{3.8.10}$$

If we integrate this from $-\infty$ to ∞, it will produce the normalization coefficient,

$$\int_{-\infty}^{\infty}\frac{1}{((\cosh(ax))^{\frac{1}{a}}} = \kappa_a. \tag{3.8.11}$$

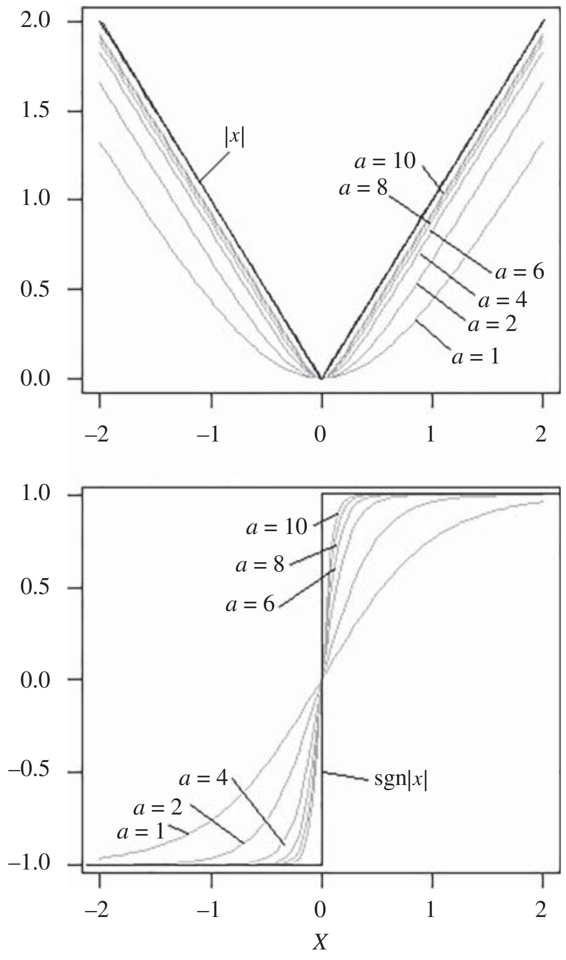


Figure 3.6 Plots of $\rho(x)$ and $\psi(x)$ for different values of a along with the L_1 loss term.

Hence, the PDF is

$$f_X(x) = \frac{1}{\kappa_a} \frac{1}{((\cosh(ax))^{\frac{1}{a}}}. \tag{3.8.12}$$

Here, κ_a is different for each selected value of a. We can choose different values of a to examine how the hyperparameter affects the distribution. In each case, an integration of Eq. (3.8.11) is needed for κ_a but to simplify the process, we can use a fitted equation given by $\kappa_a = (\pi - 2)/a + 2$. This is shown in Figure 3.7. Some of the computed values are indicated on the curve. For example, when $a = 1$, we obtain $\kappa_a = \pi$, and when $a = 100$, we obtain $\kappa_a = 2$, and so on.

From the results, we find that, as a is increased from 1 to 100, the log-cosh distribution approaches $e^{-|x|}/2$ which is the L_1 distribution. We can see the effect of this in Figure 3.8. The analysis confirms that the log-cosh family of distributions approaches the Laplace distribution when a is large. This is an important result because it clearly explains why it is robust.

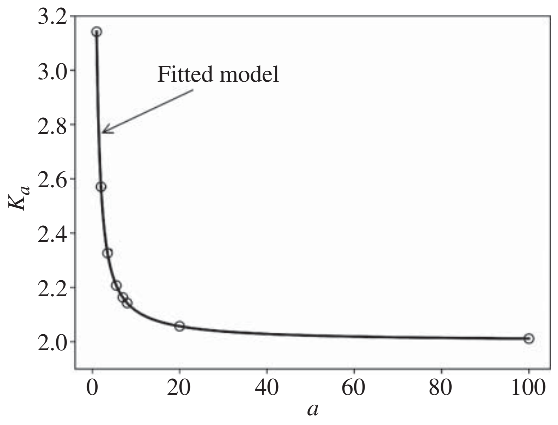


Figure 3.7 κ_a as a function of a and a fitted model of $\kappa = (\pi - 2)/a + 2$.

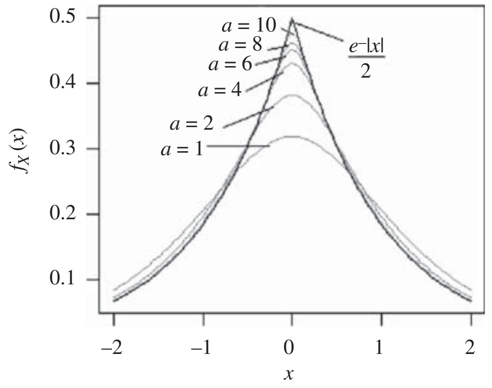


Figure 3.8 Comparison of double-exponential with log-cosh for different values of a.

To select appropriate values of a, we return to Figure 2.8 in Chapter 2 showing the losses and gradients for L_1 and L_2. A similar set of plots can be produced for log-cosh as a function of a. Using the same data to find the location parameter θ, namely $\{0, 3, 5, 7, 10\}$, the results are shown in Figure 3.9. Note that as a increases from 1 to 100, the continuous loss becomes piecewise linear while the continuous gradient becomes stepwise continuous. It is clear that the log-cosh loss can behave similar to L_2 at one end and L_1 at the other. However, values of a above 10 are not recommended since it may introduce convergence problems in gradient descent. Therefore, the log-cosh loss function should always be used in its most general form as

$$\text{LC loss} = \frac{1}{a}\sum_{i=1}^{n} \log(\cosh(a\, r_i)) \tag{3.8.13}$$

with typical values of a in machine learning tasks being set to 1, 1.5, 2, 5, or 10. For linear regression, a value of $a = 2$ is recommended for reasons provided in Chapter 4. Note that a is not data dependent like δ in Huber. It only depends on the type of machine learning task being undertaken. The standard errors of the log-cosh estimates from linear regression can be obtained using the same procedure outlined in Section 3.6.

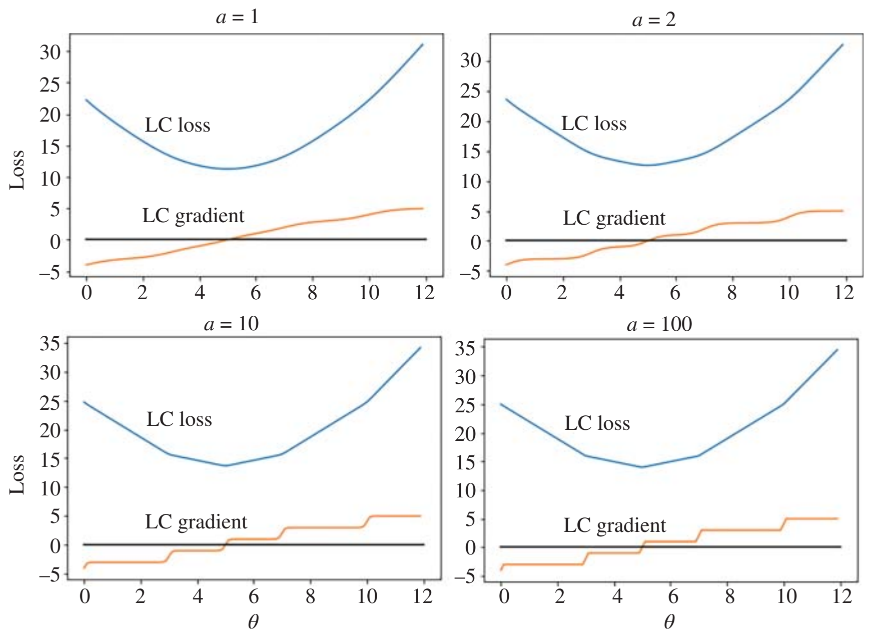


Figure 3.9 Comparison of LC loss and gradient with different values of a. For linear regression, a value of $a = 2$ is recommended.

3.9 Summary

The log-cosh loss function is perhaps one of the most important loss functions in machine learning because it offers robustness along with all of the attendant regularity conditions that allow for a thorough mathematical treatment. In this chapter, the distribution from which the loss function arises has been identified, followed by a maximum-likelihood derivation of the loss function. Its statistical properties were provided along with the computation of standard errors. The s.e.'s for log-cosh were compared to the least squares case and found to be smaller than least squares for datasets with outliers, and similar for datasets with no outliers. As a result of the detailed analysis, a strong case can be made for using the general log-cosh loss to replace many other loss functions in machine learning.

Problems

3.1 Show that the CDF associated with the log-cosh loss, which is given by

$$F_X(x) = \frac{1}{2} + \frac{1}{\pi}\arctan(\sinh(x)),$$

is correct by taking its derivative to produce the PDF. (Hint: you may need the identity $\sinh^2(x) + 1 = \cosh^2(x)$.)

3.2 Use the reverse MLE procedure to find the distribution associated with

$$\text{M-loss} = \sum_{i=1}^{n} \frac{1}{2} \log(\cosh(2x))$$

using the following steps.

a) Find the form of the distribution using $f(x) = e^{-\rho(x)}/\kappa$.

b) Find the normalizing constant κ by integrating the result from $-\infty$ to ∞.

3.3 Derive the Huber distribution using the reverse MLE procedure. The first few steps are given below starting with the term:

$$\rho_H(x) = \begin{cases} x^2/2 & \text{if } |x| \le \delta \\ \delta\left(|x| - \frac{\delta}{2}\right) & \text{if } |x| > \delta \end{cases}.$$

Application of the reverse MLE rule above gives

$$f_X(x) = \frac{1}{\kappa} e^{-\rho_H(x)} = \frac{1}{\kappa} \begin{cases} e^{-x^2/2} & \text{if } |x| \le \delta \\ e^{-\delta\left(|x| - \frac{\delta}{2}\right)} & \text{if } |x| > \delta \end{cases}.$$

The only remaining steps involve finding the value of κ by integrating the different regions and adding up the quantities. Use $\delta = 1.345$ to simplify the calculations. Plot the distribution and compare it to the one in Figure 3.5. Re-do the problem with $\delta = 0.1$ and $\delta = 10$. Which of the two plots has heavier tails? Which one is more robust to outliers?

3.4 (PROJECT) In this project, you will estimate the slope and intercept of a dataset using robust regression with the general log-cosh loss and compare it with Huber's method as implemented in the Python library `sklearn.linear_model`.

a) Load Python libraries.

```
import numpy as np
from sklearn.linear_model import HuberRegressor
import matplotlib.pyplot as plt
from scipy.optimize import minimize
```

b) Create the Belgium telephone dataset manually.

```
X = np.linspace(1950, 1973, 24).reshape(-1, 1)
y = np.array([ 0.44,  0.47,  0.47,  0.59,  0.66, \
              0.73,  0.81,  0.88,  1.06, 1.2 , \
              1.35,  1.49,  1.61,  2.12, 11.9 , \
              12.4 , 14.2 , 15.9 , 18.2 , 21.2 , \
              4.3 ,  2.4 ,  2.7 ,  2.9 ])
```

c) Define the general log-cosh loss.

```
logcosh = lambda x: np.logaddexp(x, -x) - np.log(2)
def residual(b,x,y):
        residual_y = y-np.matmul(x,b)
        return residual_y
    def LC(beta):
        ri = residual(beta,x,y)
        rho = (1/a)*logcosh(a*ri)
        loss = np.sum(rho)
        return loss
```

d) Create a column of 1's in the x matrix.

```
column_ones = np.ones((24, 1))
x = np.hstack((column_ones, X))
```

e) Find the slope and intercept using log-cosh.

```
initbeta = np.array([0.,0.])
a = 2
minval = minimize(LC,initbeta)
beta = minval.x
```

f) Plot the results.

```
abline = np.zeros(x.shape[0])
for i in range(x.shape[0]):
    abline[i] = beta[0]+beta[1]*x[i,1]
plt.plot(x[:,1],abline)
plt.scatter(x[:,1],y)
plt.show()
```

g) Rerun with different values of the robustness coefficient `a`. Comment on the results.

h) Run the Huber regressor. Adjust δ (which is called `epsilon`) starting from 4.0 down to 0.1 to determine its effect. Is this implementation of Huber's method satisfactory?

```
X = np.linspace(0, 23, 24).reshape(-1, 1)
huber = HuberRegressor(epsilon=4.0)
huber.fit(X, y)
y_pred = huber.predict(X)
plt.scatter(X+1950, y, color='blue', \
    label='Data points')
plt.plot(X+1950, y_pred, color='red', \
    label='Huber Regressor')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Huber Regressor with Outliers')
plt.show()
```

3.5 (PROJECT) In this project, you will find the estimates and standard errors for the Swiss dataset using the least squares approach. The data itself was provided in Table 3.1. A file called `swiss.csv` must be created to use with the Python code below.

a) The following code can be used to import libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.optimize import minimize
```

b) The Swiss dataset may be read in to create the $\boldsymbol{X}$ and $\boldsymbol{y}$ values as follows:

```
swiss = pd.read_csv("swiss.csv")
swiss['ones'] = 1
X = np.array(swiss.drop(["Fertility"],axis=1))
Y = np.array(swiss["Fertility"])
```

c) The LS code is as follows:

```
def LS(beta):
    ri = Y-np.matmul(X,beta)
    rho = ri*ri/2
    loss = np.sum(rho)
    return loss
```

d) The following code can be used to generate the estimates.

```
# Initialize betaLS
initbetaLS = np.array([-0.1,-0.1,-0.1,0.1,0.1,-0.1])
# Minimize the LS  loss function
model = minimize(LS, initbetaLS, method='BFGS')
# Extract the estimates
betaLS = model.x
col_names = swiss.columns
print("Estimates:")
for i in range(len(betaLS)-1):
    print(col_names[i+1],f"{betaLS[i]:.3f}")
print("(Intercept)",f"{betaLS[len(betaLS)-1]:.3f}")
```

e) To obtain the standard errors, we can use the formulation given in Eqs. (3.6.2)–(3.6.4).

```
resid = Y-np.matmul(X,betaLS)
n = X.shape[0]
p = X.shape[1]
d = p - 1
sigma2 = np.sum((resid-np.mean(resid))**2)/(n-d-1)
C = np.linalg.inv(np.matmul(X.T,X))
se2 = sigma2*np.diag(C)
```

```
seLS = np.sqrt(se2)
print("Standard errors:")
for i in range(len(seLS)-1):
    print(col_names[i+1],f"{seLS[i]:.3f}")
print("(Intercept)",f"{seLS[len(seLS)-1]:.3f}")
```

3.6 (PROJECT) In this project, you will find the estimates and standard errors for the Swiss dataset from the previous problem using the log-cosh approach.

a) The LC code and MADN(r) function are as follows:

```
logcosh = lambda x: np.logaddexp(x, -x) - np.log(2)
a = 2
def LC(beta):
    ri = Y-np.matmul(X,beta)
    loss = np.sum((1/a)*logcosh(a*ri))
    return loss
def MADN(r):
    return(1.4826*np.median(np.abs(r-np.median(r))))
```

b) The following code can be used to generate the estimates:

```
# Initialize betaLC
initbetaLC = np.array([-0.1,-0.1,-0.1,0.1,0.1,-0.1])
# Minimize the LC  loss function
model = minimize(LC, initbetaLC, method='BFGS')
# Extract the estimates
betaLC = model.x
col_names = swiss.columns
print("Estimates:")
for i in range(len(betaLC)-1):
    print(col_names[i+1],f"{betaLC[i]:.3f}")
print("(Intercept)",f"{betaLC[len(betaLC)-1]:.3f}")
```

c) To obtain the standard errors, we can use the formulation provided in Eqs. (3.6.5)–(3.6.10).

```
resid = Y-np.matmul(X,betaLC)
se = MADN(resid)
num = np.mean(np.tanh(a*resid/se)**2)
denom = (np.mean(a/np.cosh(a*resid/se)**2))**2
n = X.shape[0]
p = X.shape[1]
d = p - 1
vhat = (n/(n-d-1))*se*se*num/denom
C = np.linalg.inv(np.matmul(X.T,X))
se2 = vhat*np.diag(C)
```

```
seLC = np.sqrt(se2)
print("Standard errors:")
for i in range(len(seLC)-1):
    print(col_names[i+1],f"{seLC[i]:.3f}")
print("(Intercept)",f"{seLC[len(seLC)-1]:.3f}")
```

References

Maronna, R.A., Douglas Martin, R., Yohai, V. J., and Salibian-Barrera, M. (2019). *Robust Statistics, Theory and Methods (with R)*. Wiley.

Montgomery, D.C., Peck, E.A., and Vining G.G. (2012). *Introduction to Linear Regression Analysis*, 4e. Wiley.

4

Outlier Detection, Metrics, and Standardization

4.1 Introduction

This chapter addresses three important topics related to outliers and robust linear regression. First, we describe techniques for finding and removing outliers to produce outlier-free datasets, which are needed for many different types of statistical analysis. Included in this chapter are standard methods such as boxplots, histograms, and quantile–quantile plots. Each of these methods will be used to diagnose a number of datasets to demonstrate how outlier detection should proceed. This chapter also provides practical solutions to the problem. An algorithm is presented to find all the outliers, while leaving the true data intact, based on an iterative scheme using the boxplot. Traditional methods have not been adequate in this task because the least squares method forces the residuals to follow a Gaussian distribution. This tends to hide the outliers, making them hard to detect. In fact, we will show that robust methods provide the most reliable means to diagnose, detect, and remove outliers.

Second, the proper selection of evaluation metrics for tools that operate on datasets with outliers is presented. If an unsuitable metric is chosen to evaluate accuracy in the presence of outliers, the results can be misleading or hard to interpret. A detailed analysis of the mean square error (MSE), root mean square error (RMSE), and median absolute error (MAE) metrics is provided. Once established, these metrics can also be used to select the robustness hyperparameters, such as a in log-cosh methods (see Chapter 3) and δ for Huber's method (see Chapter 2).

Finally, to close out the chapter, procedures used to standardize datasets are discussed. Typically, machine learning methods work best with variables that have similar ranges of values. Some methods require that datasets be standardized in order to function properly. We present a robust standardization method which is needed when the explanatory variables have outliers.

4.2 Effect of Outliers

Outliers are a major concern in data science and machine learning. They can significantly impact statistical procedures beyond just the mean and standard deviation, as described in previous chapters. For instance, in the clustering problem, they can distort cluster shapes significantly leading to incorrect cluster centers (see Chapter 1). Linear regression slope and intercept estimates are skewed when outliers are present, as seen in Chapters 2 and 3. There are other issues caused by outliers in statistical applications. For example, they can lead to correlation coefficients that are either higher or lower than the true values (see Chapter 12). The analysis of variance (ANOVA) can be affected, making it harder to detect differences between groups. The process of imputation, that is replacing missing values in a dataset, can be skewed by outliers because it relies on the mean and variance of the data. Outliers can also affect *t*-tests (see Chapter 5) or chi-square tests in hypothesis testing, one of the mainstays of statistical analysis, potentially leading to incorrect conclusions. These are just a few of the many procedures requiring outlier-free datasets.

Unfortunately, outliers are generally viewed as nuisance observations in a dataset and so robust methods have not been widely embraced as yet. Their effects are often considered to be relatively benign, especially if there are only a few of them. As a result, they are usually overlooked or ignored completely. However, ignoring outliers is a risky proposition and can result in misleading conclusions or significant issues in data analysis. Instead, there are better options involving the removal of the outliers before using the statistical procedures. If done carefully, it can be a very effective way to circumvent the problem of outliers.

Before describing different strategies for outlier removal, we first demonstrate that even a single outlier can substantially impact the results when using the least squares (LS) approach. And second, we illustrate the impact as the number of outliers in the data is steadily increased. The results are contrasted with the robust log-cosh (LC) approach which, as detailed in Chapter 3, is quite tolerant to outliers.

To demonstrate the potential issues with only one outlier, we show four different cases in Figure 4.1 based on the Belgian telephone dataset of Chapter 2. The explanatory variable is `year` and the response variable is `calls`. In each case, one outlier is manually inserted in different locations. We will find that a single outlier can have unpredictable effects on LS estimates but little or no effect on LC estimates, depending on its position relative to other observations in the dataset. For example, in Figure 4.1(a), the insertion of one outlier at (`year` = 1963, `calls` = 12.5) produces only a small difference between LS and LC. Both produce acceptable results. This is a low-impact outlier that leads many to believe their effects are relatively benign. However, when the outlier is moved to (`year` = 1985, `calls` = 12.5), as in Figure 4.1(b), the LS and LC results begin to differ; hence, this is a medium-impact outlier in this dataset. When we try to explain any prediction results from LS, we have to somehow account for the fact that all the data points are not represented accurately by the model. In other words, we have to justify the inaccurate model in some plausible manner. In contrast, LC produces the same result in both cases.

Now consider the case of the outlier at (`year` = 1950, `calls` = 12.5) in Figure 4.1(c). This position constitutes a high-impact outlier for this dataset. In fact, LS tells us that there is no relationship between `calls` and `year`, which we know is incorrect. Any business decision made using this result will also be incorrect. In a multidimensional setting, this variable may be inadvertently removed during model reduction by a penalty function (see Chapter 5). Trying to explain the removal of a critical variable would be challenging and ultimately misleading to others. In the last case, Figure 4.1(d), the outlier placed at (`year` = 1940, `calls` = 12.5) is called a *high-leverage* point since the value of `year` is well outside the domain of the rest of the data. This type of outlier can have serious consequences on the results of LS. In fact, it shows a negative correlation between `year` and `calls` which is clearly wrong. High-leverage points can be detected and removed, but it requires additional effort. However, the LC method is not significantly affected in either case.

The point of all these examples is to illustrate the unpredictability of LS when even a single outlier is present, let alone a number of outliers and their effects. They show that outliers have varying impacts on

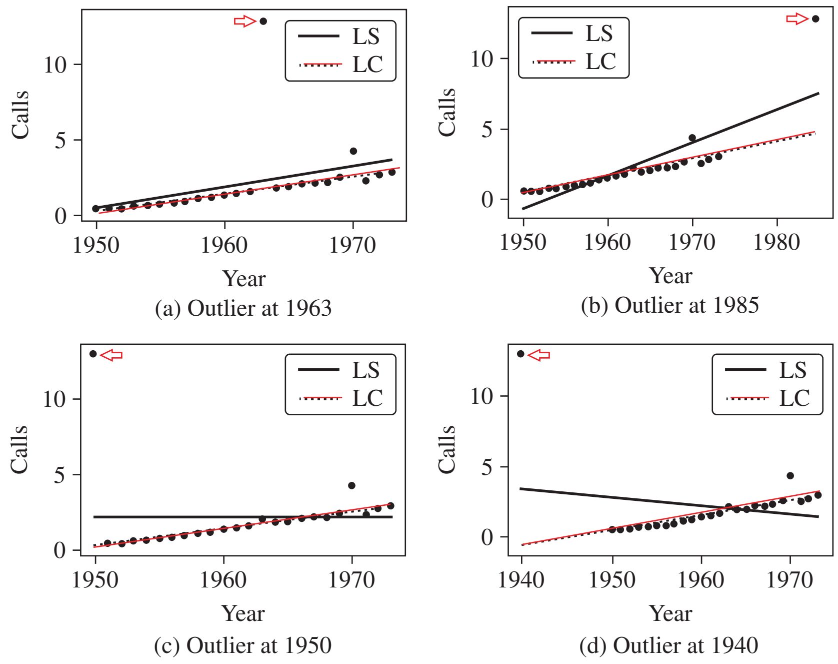


Figure 4.1 Effect of a single outlier (indicated with an arrow) on LS and LC estimators for (a) low-impact outlier, (b) medium-impact outlier, (c) high-impact outlier, and (d) high-leverage point.

the results when LS is used but the results are quite stable for all four cases when LC is used. It would be easy to explain the LC model for all four cases and make good decisions using this information, whereas trying to explain the results for LS would be complicated.

Next, consider multiple outliers in the dataset, as in Figure 4.2. There are four different cases to study the effects of increasing the number of outliers in the data. The datasets of Figure 4.2 (a), (b), (c) and (d) have 0, 2, 4, and 6 outliers, respectively. As more outliers are added, the line associated with LS shifts toward the outliers, while the line based on LC remains relatively stable. This figure gives us an indication of how easily LS estimates can shift due to a few outliers. The question is, why is this happening? The details were described earlier in the book but the simple answer is that the quadratic loss function associated with LS amplifies the error due to the outlier by squaring it. That makes outliers more important than the data itself. Therefore, in order to reduce the error and satisfy the LS condition (the residuals must sum to 0), the regression line is forced to be closer to the outliers. Unfortunately, this is the opposite of what should happen because we want the line to be closer to the inliers. Again, the results are different in all four cases for LS but stable for LC.

4.3 Outlier Diagnosis

The methods of dealing with outliers discussed so far in this book involve using robust machine learning tools. Another way to address the problem of outliers is to remove them. However, this is not

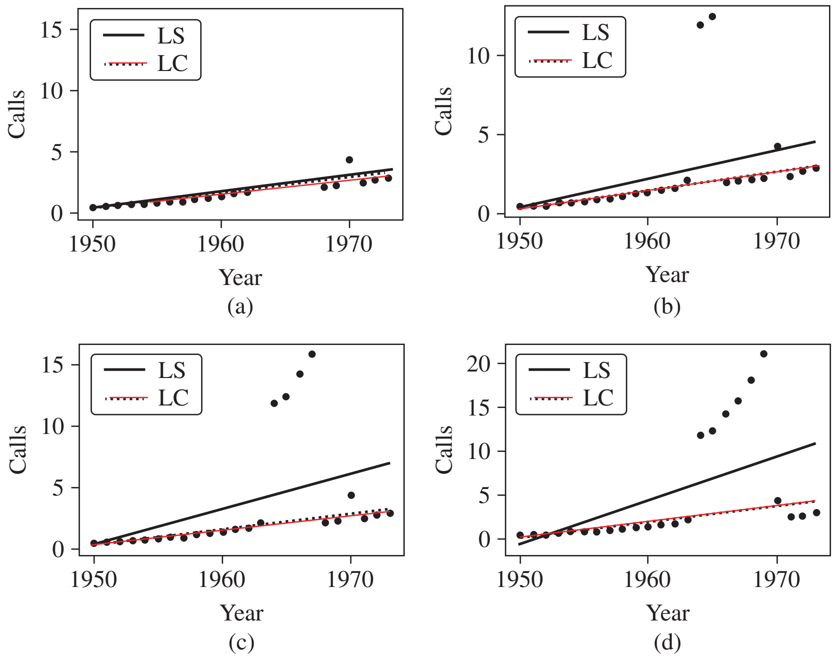


Figure 4.2 Four plots using different versions of the telephone dataset with fitted lines. The LC approach is robust and stable while the LS line varies depending on the number of outliers, (a) data with no outliers, (b) two outliers, (c) four outliers, and (d) six outliers.

as straightforward as it sounds. There is the risk of removing inliers along with outliers and there is the possibility of leaving some outliers in the dataset if they are not detected. Typically, the former occurs more frequently than the latter. In any case, it must be done carefully or the modified dataset may not be outlier-free or may be missing important inliers.

In this section, we discuss two common methods of outlier diagnosis where the objective is to determine if any outliers exist in the data. These methods are typically employed during exploratory data analysis (EDA). Without running any form of regression, only the explanatory and response variables are available for diagnosis. This is where the problem of inadvertently removing inliers occurs because it is difficult to tell outliers from inliers at this stage. However, after running a linear regression, the residuals are available for examination and they can be used to more accurately identify outliers. In a later section, we address detection and removal using residuals.

4.3.1 Boxplots

The easiest way to identify outliers is by using a boxplot. This compact graphical tool visually represents a one-dimensional set of data values, offering a quick snapshot of the inliers and outliers. The idea of a boxplot is to order the data values from smallest to largest and then identify where the bulk of the values reside to assess whether there are outliers beyond this range. Specifically, we identify the quartiles of the data (25th, 50th, and 75th percentiles) and determine if any data points lie well above the 75th percentile or well below the 25th percentile. If so, they are declared as outliers.

The process for creating a boxplot is as follows:

- First, find the median (Q2), the 25th percentile (Q1), and 75th percentile (Q3). The data between the 25th and 75th percentiles defines the inter-quartile range (IQR) where 50% of the data resides.
- For visualization purposes, place a bounding box between the 25th and the 75th percentiles and a line inside the box at exactly the 50th percentile (the median). The rest of the data lies outside this box.
- Next, we want to identify a reasonable range of values outside this box where the data would still be considered inliers. A rule-of-thumb is to use $1.5 \times \text{IQR}$ as the range on each side. We indicate this range in the boxplot with a line-and-whiskers demarcation.
- The whiskers extend from Q1 to the smallest data point above $\text{Q1} - 1.5 \times \text{IQR}$ and from Q3 to the largest data point below $\text{Q3} + 1.5 \times \text{IQR}$. This may lead to asymmetry in the whisker length on each side.
- All values outside this range are flagged as outliers and marked with circles above and below the whiskers, as the case may be.

Three examples of boxplots for three different variables are shown in Figure 4.3. In each case, the line in the middle of the box represents the median of the plotted data. The box itself shows the range from the 25th percentile (Q1) to the 75th percentile (Q3), which is the IQR. The extended lines terminate at the whiskers, and this represents the degree of dispersion of the given data. The lower and upper bounds are calculated as $\text{Q1} - 1.5 \times \text{IQR}$ and $\text{Q3} + 1.5 \times \text{IQR}$, respectively, with whiskers terminating at the lowest and highest data values in those regions, respectively.

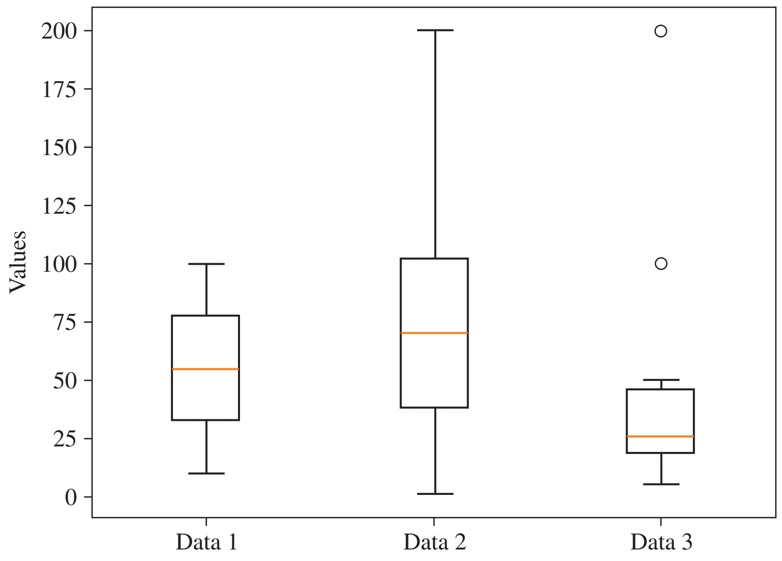


Figure 4.3 Example boxplots for different sample sets. Note that the rightmost plot clearly shows outliers outside the whiskers.

Examining the three boxplots shown in Figure 4.3 in greater detail, we find that:

- The leftmost boxplot is symmetric and represents data that is uniformly spaced so it has a nice compact representation with no outliers.
- The middle boxplot is asymmetric in terms of the whisker lengths due the values above the 75th percentile (but within the $Q3 + 1.5 \times IQR$ region) which are more spread out compared to those below the 25th percentile. There are no outliers in this case.
- The rightmost boxplot is also asymmetric and the median line is not centered in the box. The position of the median is data dependent so it may be located anywhere in the box. There are two outliers in the data (indicated with circles) due to the fact that they lie beyond the whiskers.

Boxplots are used on explanatory variables and the response variable during EDA to get a sense of the data and the likelihood of finding outliers in the dataset.

4.3.2 Histogram Plots

A histogram plot is a graphical representation of the approximate distribution of a given sample set. If properly generated, it enables a rapid assessment of the existence of outliers. To create a histogram, a set of equal-sized *bins* are defined based on the min–max range of the data. All values that lie within the bounds of a given bin are counted and the counts in each bin are plotted to form the histogram. Hence, the histogram is a frequency plot of the number of occurrences of values in each bin. The width of a bin or the number of bins can be controlled by the user. Outliers may be found in bins that are separated at a distance from the bulk of the data on either side with relatively low frequencies. The key issue is the proper selection of the *bin size*. If the bin size is too small, the frequency within each bin will be too low to identify outliers. If the bin size is too large, the outliers may be lumped in with inliers.

Figure 4.4 shows a sample histogram with frequency along the vertical axis and data values along the horizontal axis. When assessing histogram plots to find outliers, we are looking for data in bins that are

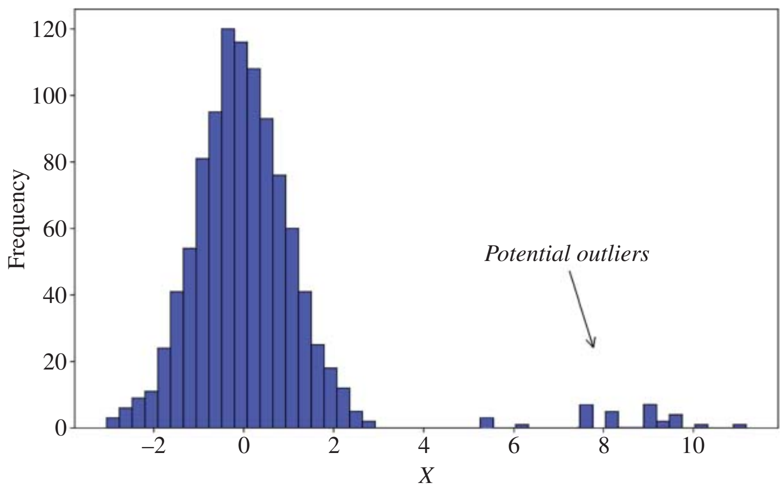


Figure 4.4 Example of histogram plots.

outside the bulk of the data. In this case, it does appear to have multiple outliers beyond a value of 4. On the other hand, a value of 7 may be a better choice. The criteria to decide if some of the data should be considered as outliers is referred to as the *threshold* and its selection is critically important in outlier detection. Further details are provided in Chapter 11 on anomaly detection.

4.3.3 Exploratory Data Analysis

EDA usually begins with boxplots and histograms of the data columns to familiarize oneself with the dataset. Early stages of outlier detection involve using only the explanatory variables and the response variable because no other information is available. Unfortunately, outlier diagnosis using traditional descriptive methods such as boxplots or histograms of the input/output variables can be misleading and inaccurate. To illustrate the issues, a well-known dataset, referred to as the Boston Housing dataset, will be used. It is appropriate to introduce it here since it is known to have many outliers. This dataset provides 13 explanatory variables that relate to the median price of 506 homes in the Boston area in the 1970s. Home prices in the dataset range from \$0 to \$50K. The goal is to build a model for prediction using linear regression. In this section, we consider only 11 explanatory variables, as 2 of the variables are categorical in nature and therefore not amenable to boxplots or histograms. Other details of the dataset are not pertinent at this point, but the dataset itself will be referenced throughout the book.

Consider the boxplots shown in Figure 4.5 (displayed horizontally here for convenience). They indicate that many of the variables may contain outliers. Only four variables (`INDUS`, `NOX`, `AGE`, and `TAX`) appear to be outlier-free according to their boxplots. The `MEDV` response variable and `ZN`, `CRIM`, `B`, `LSTAT`, `DIS`, `PTRATIO`, and `RM` explanatory variables all appear to have outliers. The estimated percentage of outliers is provided in each case. Here is the problem: if we declare all of them to be outliers, a total of 134 observations out of 506 will have to be removed. This is approximately 26% of the contents of the dataset. Clearly, it is not reasonable to remove such a large percentage. It is likely that some inliers will be deleted if this approach is used. Furthermore, some outliers may be undetected and remain lurking in the dataset. While this is often done in practice, there are much more effective methods that will be described in this chapter.

Another way to identify outliers is to generate histograms of the variables and examine their distributions. This is shown in Figure 4.6. If the distributions are Gaussian in nature, it is unlikely to have outliers. This is the case for variable `RM`. However, the rest of the variables do not appear to be Gaussian but it is hard to detect any outliers from the distributions. This is one of the problems with the histogram: it requires a separate bin size and threshold for each variable to find outliers and is not as informative as the boxplot. Standardization can help (as described later in Section 4.9) but if not done properly, it can also hide outliers. On the other hand, histograms do allow you to find any usual characteristics at a glance.

For example, the response variable `MEDV` is worth further investigation for potential outliers. Figure 4.7 shows the histogram in more detail using a smaller bin size. It indicates the presence of a number of homes priced at exactly \$50K. The reason for this unusual number of \$50K homes is not clear but it could be due to winsorization (see Section 4.5.2); that is, the values were so large that it skewed the data and they were simply assigned to \$50K. As a result, it is not clear if they are outliers or inliers or a mix of both. We cannot risk deleting them from the dataset for fear of "throwing the baby out with the bath water". Overall, the best we can say at this stage from the boxplots and histograms is that the dataset likely contains outliers.

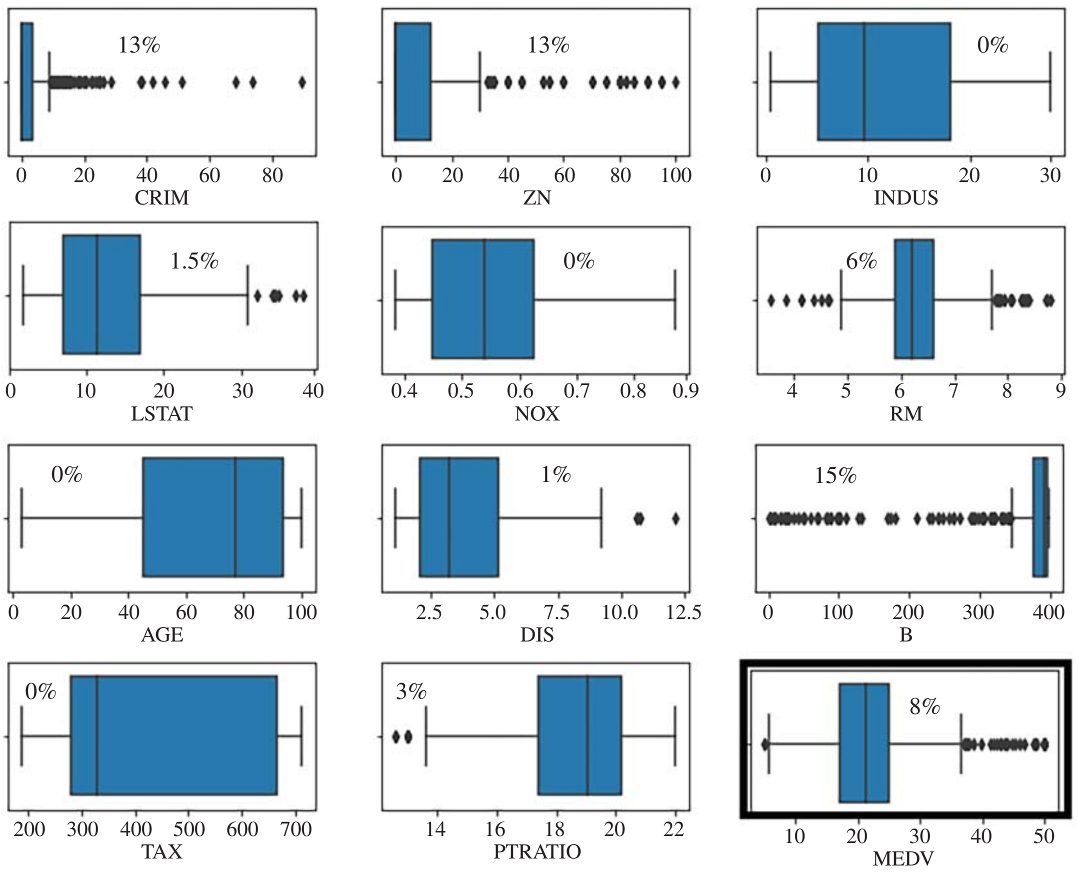


Figure 4.5 Boxplots can be used to check for outliers in all variables for the Boston housing dataset.

4.4 Outlier Detection

After the preliminary diagnosis phase is complete, if we know that the dataset has outliers, we can move to the outliers detection phase (see Aggarwal 2017). Here, we are interested in the number of outliers in the dataset. In this section, three well-known methods are described. Note that methods for counting outliers are described in this section, whereas removing the detected outliers will be described in the next section.

4.4.1 3-Sigma Edit Rule

The 3-sigma edit rule is probably the most widely known method of outlier detection for data distributed as a Gaussian. The idea is that any data points more than 3 standard deviations away from the mean are counted as outliers. This is a sensible approach since 99.7% of the data lies within $\pm 3\sigma$ of the mean, μ, for the normal distribution. Figure 4.8 shows a Gaussian distribution, labeled "true distribution," along

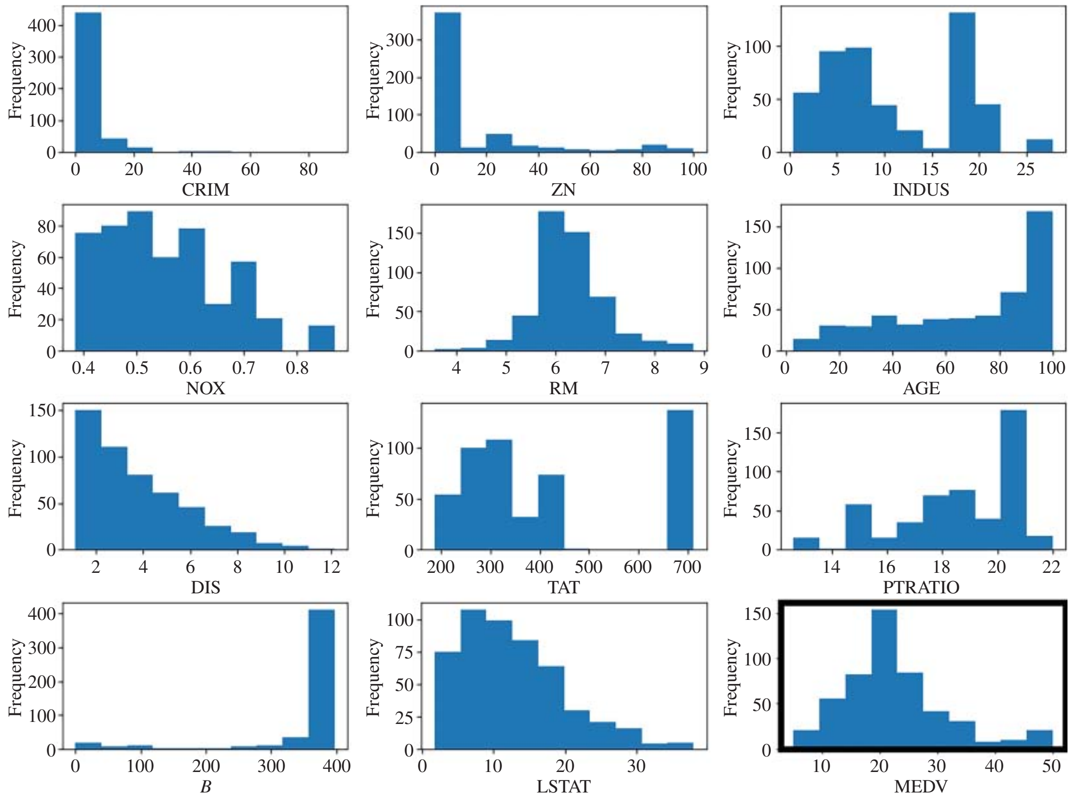


Figure 4.6 Histogram can be used to check for outliers in all variables for the Boston housing dataset.

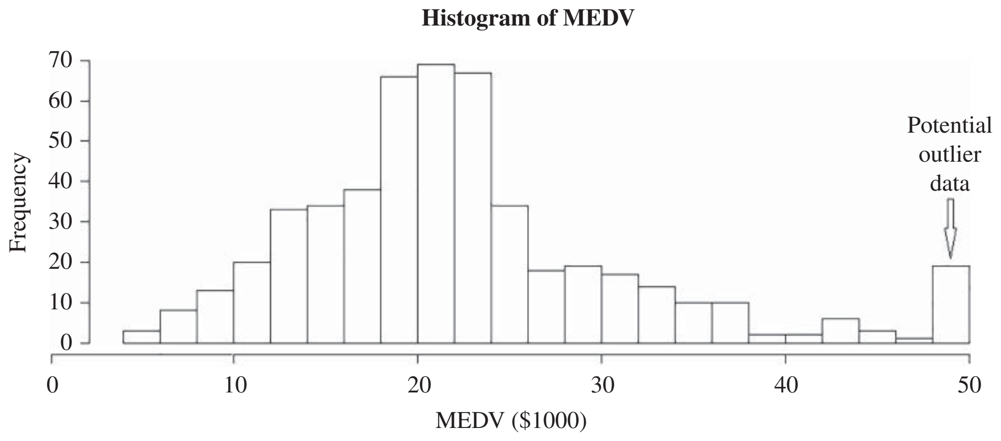


Figure 4.7 Distribution of MEDV showing unusual number of homes at $50K.

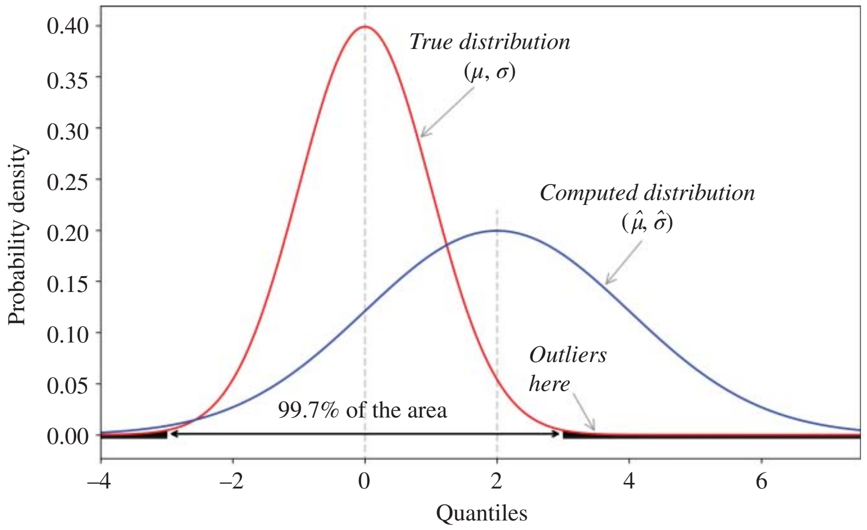


Figure 4.8 Gaussian distributions illustrating issues with the 3-sigma edit rule.

with a region to the right in which outliers reside. The values along the x-axis are sometimes referred to as **quantiles**.[1] We want to find outliers that occur above a quantile of 3 or below a quantile of −3. In order to apply the 3-sigma edit rule, one can standardize the data using the equation,

$$z_i = \frac{x_i - \hat{\mu}}{\hat{\sigma}}, \tag{4.4.1}$$

to obtain the z-scores so that all the data belongs to $\mathcal{N}(0, 1)$. Both $\hat{\mu}$ and $\hat{\sigma}$ are computed using the data values. Unfortunately, the $\hat{\mu}$ value will be shifted relative to μ, and $\hat{\sigma}$ value will be inflated relative to σ due to outliers. These effects are shown in the distribution labeled "computed distribution." Outliers tend to flatten and stretch out the computed distribution which hides the outliers. They are no longer beyond $\pm 3\sigma$ from $\hat{\mu}$. Therefore, the 3-sigma edit rule is of little or no value when outliers are present.

4.4.2 4.5-MAD Edit Rule

As usual, we turn to robust statistics to deal with outliers. A robust value of μ is the median($\boldsymbol{x}$). A robust value of σ for the 3-sigma edit rule can be obtained using:

$$\hat{\sigma} = \text{MADN}(\boldsymbol{x}) = 1.4826 \times \text{MAD}(\boldsymbol{x}), \tag{4.4.2}$$

where MAD($\boldsymbol{x}$) is the median absolute deviation (MAD) given by

$$\text{MAD}(\boldsymbol{x}) = \underset{1 \le i \le n}{\text{median}} \left(|x_i - \text{median}(\boldsymbol{x})| \right). \tag{4.4.3}$$

This gives rise to the 4.5-MAD edit rule which is a robust version of the 3-sigma edit rule. As the name implies, we check for outliers that are 4.5-MAD or more away from the median. The reason for the

1 The term quantile is typically associated with values between 0 and 1. Here, the quantiles lie between $-\infty$ and ∞ but their values are derived from the CDF which lies between 0 and 1.

4.5 multiplier is that we are using a 3-MADN edit rule which is roughly 4.5×MAD. In Figure 4.8, the computed distribution using robust statistics would look similar to the true distribution.

4.4.3 1.5-IQR Edit Rule

Another robust alternative is to make use of the boxplot parameters when detecting the number of outliers. It makes use of the quartiles of the data. In the earlier section, we showed that outliers reside outside the interval Q1−1.5×IQR and Q3+1.5×IQR according to the boxplot rules. This is much more effective than the 3-sigma edit rule for reasons described earlier. In fact, we can count the number of outliers in a straightforward manner using this simple approach.

There are some issues with this approach that could lead to an excessive number of outliers if used incorrectly with the explanatory variables or the response variable. We saw this earlier in Figure 4.5 for the variables of the Boston Housing dataset. If we apply the 1.5-IQR rule to this set of boxplots, we would inadvertently remove 134 observations, most of which are not outliers. Therefore, a factor higher than 1.5 is usually required to identify outliers exclusively.

4.5 Outlier Removal

Now that the different methods of outlier detection have been described, we can move to the final phase of outlier removal. There are two options available at this stage: the first is to delete all detected outliers, called *trimming*, and the second is to keep all outliers but adjust their location, a process called *winsorization*. Both are discussed in the sections to follow.

4.5.1 Trimming Methods

The conventional approach to outlier removal is called trimming. The idea is to simply remove observations associated with predictors and responses that are considered outliers. It is used in conjunction with the detection methods described above (3-sigma, 4.5-MAD, and 1.5-IQR). Often, a quantile-based approach, targeting a specific percentage of outliers (such as 5%), is used rather than the results of the detection methods. Trimming can be symmetric where both the highest and lowest values of potential outliers are removed, or asymmetric where only the highest or the lowest values of potential outliers are removed. This simple approach is illustrated in Figure 4.9. The original data is diagnosed for outliers and then those observations are removed from the dataset. While this method is widely used in data science, the main problem is that if it is applied using explanatory and response variables individually, then inliers may be thrown out with the outliers, while some outliers still remain in the dataset. This may eventually lead to inaccurate models. Also, the edit criteria are all heuristic in nature and the quality of the results depends on the specific application.

4.5.2 Winsorization

An alternative to trimming is to keep all observations but adjust the extreme values of the explanatory and response variables to be equal to some predefined maximum and minimum values. For example, if we

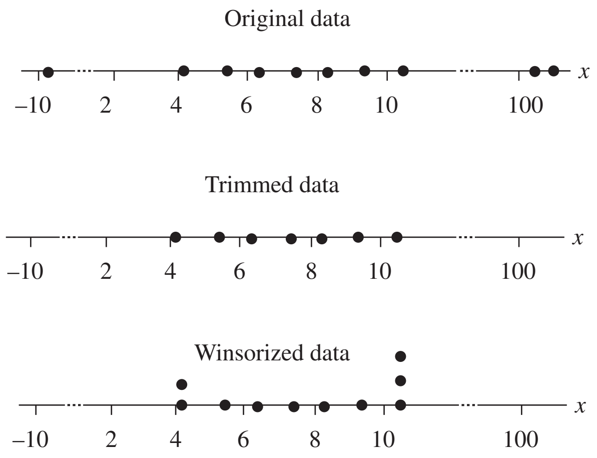


Figure 4.9 Trimmed data vs. winsorized data.

decide on a 90% winsorization of the variable under consideration, we set boundaries at the 5th and 95th percentiles. All values outside these bounds are moved to the boundaries. The procedure is illustrated in Figure 4.9 in the lower diagram. A brief outline of the steps is as follows:

1) Calculate percentiles: For each column, calculate the 5th and 95th percentiles.
2) Winsorize: Replace values below the 5th percentile with the 5th percentile value, and values above the 95th percentile with the 95th percentile value for each column.

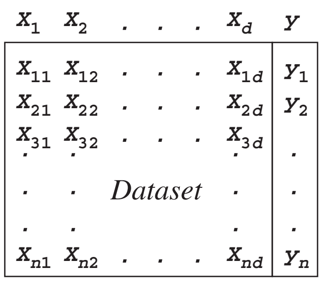
In the process, none of the data has been removed but instead their values have been reset closer to the bulk of the data. This was seen earlier in the Boston Housing dataset for response variable `MEDV` in Figure 4.7. The $50K homes have a higher frequency in the histogram than expected, perhaps due to winsorization of outlier values. This may introduce bias and some loss of information which may have some unintended consequences during linear regression. Therefore, great care should be taken using this approach.

4.5.3 Anomaly Detection Method

One additional technique that is perhaps more expensive but much more reliable is to use anomaly detection. The details of the different methods will not be covered here since Chapter 11 is devoted to the subject. However, it is important to mention that, to carry out this approach, the explanatory variables and the response variable must be combined together (rather than examining each one individually). The response vector $\boldsymbol{y}$ is attached to the $\boldsymbol{X}$ matrix to form the input to anomaly detection, as depicted in Figure 4.10. Instead of finding outliers by processing each individual variable separately, a holistic approach is taken to find them in an $\mathbb{R}^{d+1}$-dimensional space using anomaly detection. The role of an anomaly detector is to find extreme outliers in this composite dataset. It is not necessarily to find all outliers but rather the potentially troublesome ones. For example, high-leverage outliers will be far away from the inlier data and can be identified as anomalies.

The candidate tools for this purpose are DBSCAN and MADmax, both of which are described in Chapter 11. They use clustering techniques (see Chapter 1) to identify the inlier data and then detect

Figure 4.10 Combining explanatory variables and the response variable as input to anomaly detection.


x_1 x_2 . . . x_d y
x_{11} x_{12} . . . x_{1d} y_1
x_{21} x_{22} . . . x_{2d} y_2
x_{31} x_{32} . . . x_{3d}
Dataset
x_{n1} x_{n2} . . . x_{nd} y_n

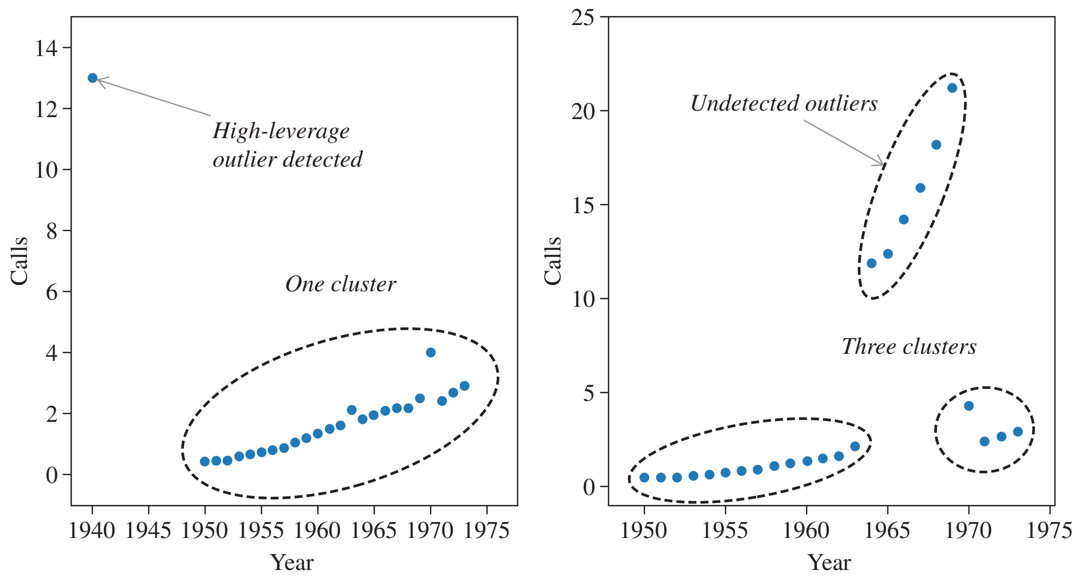


Figure 4.11 Two cases for anomaly detection. On the left, the outlier can be detected since it is isolated and far from the cluster. On the right, outliers form a cluster making them more difficult to detect.

outliers as those data points that are far from the clusters. Figure 4.11 illustrates the procedure for one explanatory variable and one response variable. The type of outlier shown on the left side of Figure 4.11 can be detected since it is isolated from the bulk of the data in a single cluster. However, if there are several outliers that form a cluster of the type shown on the right side of Figure 4.11, they may not be detected since three clusters will be formed and none of them will be considered as outliers. Nevertheless, it is important to remove the high-impact and high-leverage outliers of the type on the left side of the figure before linear regression is used, and anomaly detection has this capability.

4.6 Regression-Based Outlier Detection

If finding and removing all outliers is a critical part of a data science application, especially for statistical analysis that relies on outlier-free data, then the techniques described above may be unsuitable. In this section, we explore the options available after a linear regression is performed on the data. The difference is that we now have residuals to work with. Techniques involving quantile–quantile (QQ)-plots and ordered absolute residuals can be employed to accurately detect outliers. Additionally, more effective methods can be used to surgically remove them. In particular, a new method using an iterative boxplot approach is described to enable the removal of outliers while keeping inliers intact.

4.6.1 LS vs. LC Residuals

As mentioned earlier, using input/output variables may not be effective in outlier detection. In this section, we show that residuals have the needed information to diagnose outliers in a dataset. Recall that the residuals are simply the difference between the true response and the predicted response for all observations. The residuals for linear regression can be computed after estimation using a general equation given by

$$r_i = y_i - \hat{y}_i, \quad i = 1, \dots, n, \tag{4.6.1}$$

where y_i is the true response and $\hat{y}_i$ is the predicted value based on the regression model. Typically, outliers in the dataset have large residuals whereas inliers have small residuals. Therefore, it is possible to tell them apart by their relative positions in a residual histogram. Regardless of its size, outliers will always be positioned at one extreme or the other. The question is, should we use LS residuals or LC residuals for the task at hand?

To answer this question, consider Figure 4.12 which plots the histograms for the residuals of LS and LC of Figure 4.2(d) from the Belgian telephone dataset. For LS residuals, it is difficult to tell whether or not there are outliers since the distribution appears to be Gaussian. However, for the LC residuals, it is clear that there are six outliers. Therefore, the residuals from a robust regression contain the information needed to diagnose outliers using the histogram.

More evidence that residuals from LS should not be used to detect outliers can be found from boxplots. In Figure 4.13, we show the residuals of LS and LC from the diabetes dataset in boxplot form. The dataset

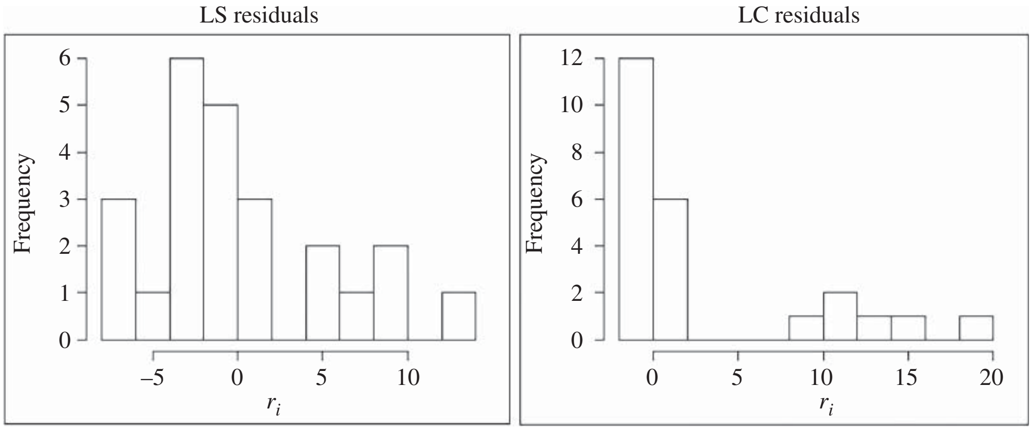


Figure 4.12 Histograms of LS and LC residuals based on Figure 4.2(d).

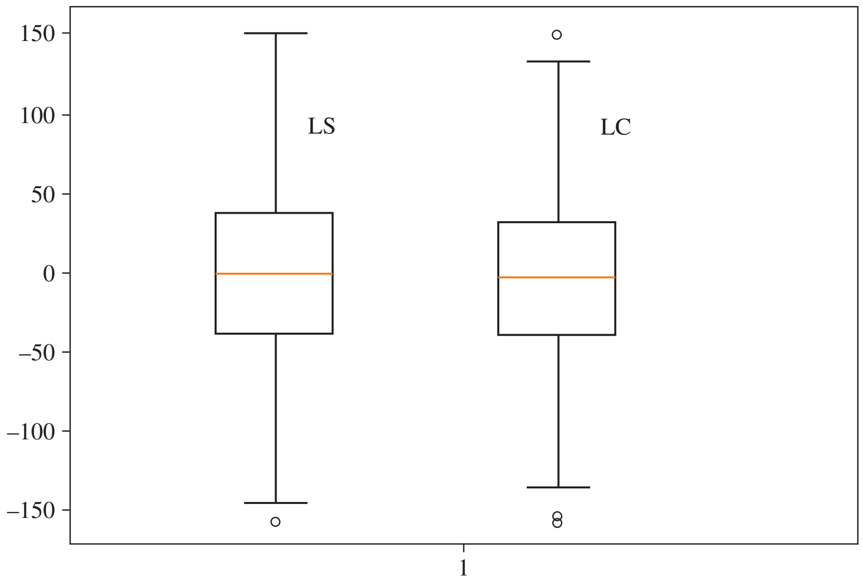


Figure 4.13 Boxplot form of 1.5-IQR edit rule using LS and LC residuals for the diabetes dataset.

has 404 observations relating 10 explanatory variables to a response variable. This dataset will also be used throughout this book but the focus in this chapter is on outlier detection. In the case of LS, only one outlier is detected when in fact there are three outliers. Notice that with LC residuals, the boxplot detects all three outliers. The key takeaway is that robust methods are needed to find the correct number of outliers, and they should be used in any mission-critical outlier diagnosis, detection, and removal methods. This was encountered in Chapter 1 for the clustering problem during outlier detection, and the same concepts apply here.

4.6.2 Comparison of Detection Methods

In this section, the three methods, namely, 3-sigma, 4.5-MAD, and 1.5-IQR edit rules, are applied to Swiss, diabetes, and Boston Housing datasets. The Swiss dataset does not have any outliers, while the diabetes and Boston Housing datasets do contain outliers. Table 4.1 shows the number of outliers obtained when

Table 4.1 Comparison of three outlier detection methods on three datasets using LC residuals.

	Swiss Fertility	**Diabetes Data**	**Boston Housing**
3-sigma	0	0	10
4.5-MAD	0	0	30
1.5-IQR	0	3	35

using the three methods. In all cases, LC regression was used to obtain the residuals. All three methods show that there are no outliers in the Swiss dataset. For the diabetes dataset, only the 1.5-IQR method produced the correct result. The other two methods could not determine the number of outliers accurately. For the Boston Housing data, the 3-sigma edit rule performed badly compared to the other methods. The 4.5-MAD and 1.5-IQR methods produced similar results, although we will find later that there are more outliers than these numbers indicate. The main conclusion here is that the 1.5-IQR approach is the most effective one when combined with LC residuals.

4.6.3 Ordered Absolute Residuals (OARs)

Aside from histograms and boxplots, there are a number of other ways to visually diagnose a dataset for outliers. One useful approach is to plot the absolute values of the residuals in sorted order from smallest to largest. A large residual is indicative of an outlier whether it be positive or negative. The notation used when ordering a set of values from smallest to largest is to place parentheses around the subscript of the variable. If we have a set of values in random order, such as $x_1, \dots, x_n$, then $x_{(1)}, \dots, x_{(n)}$ are referred to as the *order* statistics. When we take the absolute value of the quantities and then order them, we use the notation, $|x|_{(1)}, \dots, |x|_{(n)}$ for the absolute order statistics.

Exercise: Given $x_1 = 0.4$, $x_2 = -0.6$ and $x_3 = 1.7$, find the order statistics and then the absolute order statistics.

Ans.: The order statistics are given by $x_{(1)} = -0.6$, $x_{(2)} = 0.4$, and $x_{(3)} = 1.7$. The absolute order statistics are given by $|x|_{(1)} = 0.4$, $|x|_{(2)} = 0.6$, and $|x|_{(3)} = 1.7$. □

Consider Figure 4.14 showing the ordered absolute residuals from the Belgian telephone dataset based on Figure 4.2(d) for both LS and LC cases. There are a total of 24 residuals in each case. We plot the absolute order statistics of the residuals from $|r|_{(1)}$ to $|r|_{(24)}$, using an index from 1 to 24 along the x-axis. The LS residuals produce a plot that makes it difficult to separate outliers from the rest of the data points. This is because LS forces the residuals to have a Gaussian distribution which tends to blur the distinction

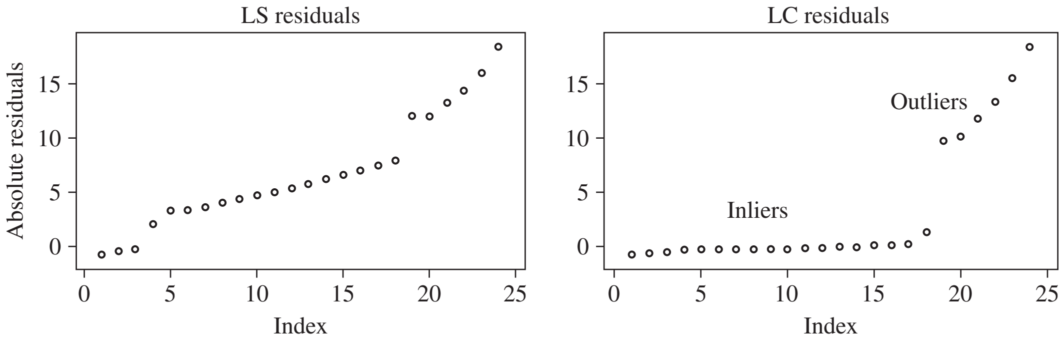


Figure 4.14 Ordered absolute residual plots of LS and LC estimators.

between inliers and outliers. Therefore, it is not easy to make a definitive statement about whether or not there are outliers based on the figure on the left panel.

On the other hand, the residuals for LC show a clear separation between the outliers and the rest of the data. A threshold of 5.0 provides a suitable horizontal dividing line to sift out the six outliers. Note that ordering does not help to separate the outliers from inliers for the LS case with a threshold of 5.0. At this threshold, some inliers would be declared as outliers, which is not correct. In sharp contrast, we see a clear distinction between the outliers and inliers in the LC case. Their respective absolute order statistics are $|r|_{(19)}, \dots, |r|_{(24)}$ since there are 24 points and the last 6 are outliers.

The six outliers may now be inspected to determine if they should be retained, modified or removed from the dataset. Some points may be easily corrected if they are due to manual entry errors or a measurement calibration error that can be adjusted. The manual removal of outliers is error prone and must be done carefully. An automatic approach using robust statistical procedures to detect and remove outliers in a systematic way will be described later in Section 4.7.

4.6.4 Quantile–Quantile Plot

The quantile–quantile (or QQ) plot (see Maronna et al. 2019) is perhaps the most important statistical tool in post-regression analysis. At a very basic level, it assesses whether the residuals belong to a Gaussian distribution. If so, then the LS method is a suitable linear regression method for the data since the residuals fit the associated distribution. Therefore, one of the first steps after LS regression should be to perform a QQ-plot on the residuals to determine if robust regression is required. Hence, it provides another qualitative way to determine whether there are outliers in a dataset. The only issue against this reasoning is that LS forces the residuals to be Gaussian, so even if the residuals appear to be Gaussian, there may still be undetected outliers. For reasons cited earlier, LC residuals may be preferable when generating a QQ-plot.

The QQ-plot is a way to visually compare the quantiles of the data to the theoretical quantiles. By default, the *theoretical quantiles are placed along the x-axis*. The quantiles were shown earlier in Figure 4.8 associated with the true distribution. To understand quantiles in the context of a QQ-plot, we start with the cumulative distribution function (CDF) for the Gaussian, denoted as $u = \Phi(x)$, where $u \in [0, 1]$. The CDF is simply the area under the Gaussian normal curve up to a point x. If the area is 0.5, then $u = 0.5$ and $\Phi^{-1}(0.5) = 0$. This quantile is at the center of the x-axis in Figure 4.8. We also note that $\Phi^{-1}(0) = -\infty$ and $\Phi^{-1}(1) = \infty$ so those values imply that the area under the curve as being 0 at $-\infty$ and 1 at ∞. All quantiles for a QQ-plot will lie between these two extremes.

To obtain the theoretical quantiles, we choose evenly spaced points between 0 and 1 along the vertical axis of the CDF. If there are n values in our data, then the quantiles are based on n uniformly spaced points between 0 and 1. In particular, the points would be located at $u = 1/(n+1), 2/(n+1), \dots, n/(n+1)$. The inverse CDF, $\Phi^{-1}(u)$, is performed on these values to obtain the theoretical quantiles.[2] The *sample order statistics are placed along the y-axis*. If the (x, y) points line-up along a diagonal, then we view the sample

2 The inverse CDF of the Gaussian distribution is typically found using lookup tables or specialized software programs, as it does not have a closed-form expression.

data as being drawn from a Gaussian distribution. Otherwise it comes from some unknown distribution. In either case, outliers are located at the extremes of the QQ-plot and are usually found off the diagonal.

Example: To illustrate the process of constructing a QQ-plot, we use a simple example with only four values. Let $y_1 = 0.4$, $y_2 = -0.6$, $y_3 = 1.7$, and $y_4 = -1.8$. First, we need the order statistics as follows: $y_{(1)} = -1.8, y_{(2)} = -0.6, y_{(3)} = 0.4$, and $y_{(4)} = 1.7$. In order to generate a QQ-plot, the theoretical quantiles must be obtained using a CDF plot. The theoretical quantiles, x_1, x_2, x_3, and x_4, are based on $u = 0.2, 0.4, 0.6$, and 0.8, respectively. We need the inverse CDF to obtain them, as follows: $\Phi^{-1}(0.2) = -0.84$, $\Phi^{-1}(0.4) = -0.25$, $\Phi^{-1}(0.6) = 0.25$, and $\Phi^{-1}(0.8) = 0.84$.

The process of obtaining the theoretical quantiles is illustrated in Figure 4.15. It shows the CDF, $\Phi(x)$, for a standard normal Gaussian. The sample values are indicated with "■" along the x-axis. The theoretical quantiles are labeled from x_1 to x_4. The x_1 quantile is obtained from the inverse CDF associated with $1/(n+1) = 1/5 = 0.2$. It is located at $\Phi^{-1}(0.2) = -0.84$. The same process holds for the other quantiles. Finally, we have the values needed for a QQ-plot which compares the theoretical quantiles with the ordered actual values. The pairwise points to be plotted are

$$(-0.84, -1.8), (-0.25, -0.6), (0.25, 0.7) \text{ and } (0.84, 1.7). \qquad \square$$

The previous example with four points is too small to obtain meaningful results, as a sample size of at least 30 points is generally recommended for reliable QQ-plots. Consider the three realistic QQ-plots in Figure 4.16. We see the theoretical quantiles on the x-axis and the sample quantiles on the y-axis. To check if the sample quantiles belong to the Gaussian distribution, we expect the points to lie along the diagonal line shown. On the left panel, we find that the data is Gaussian because the points lie along the

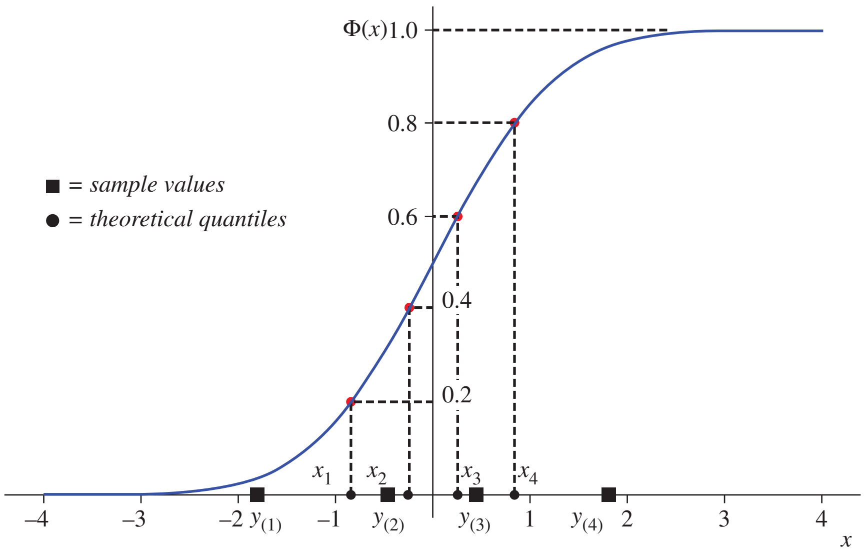


Figure 4.15 Obtaining theoretical quantiles from the Gaussian CDF. The $y_{(i)}$ values (shown as "■") are the order statistics of the sample values. The x_i values (shown as "•") are the theoretical quantiles.

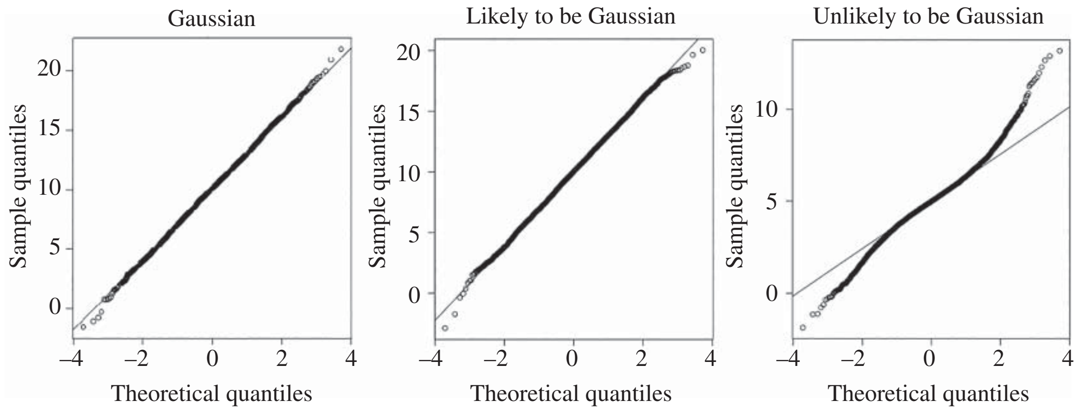


Figure 4.16 Example QQ-plots for different distributions.

diagonal. In the second case, the points mostly lie on the diagonal, but there are a number of points off the diagonal at the extreme ends. However, the distribution is considered to be Gaussian because nearly all the other points lie along the diagonal, which would not happen if it were not a Gaussian sample. It is a judgement call but this is well within an acceptable region around the diagonal. In the third QQ-plot, the sample distribution is not Gaussian, but rather some other unknown distribution. There are clearly outliers in the dataset based on this QQ-plot. A robust method based on the LC loss is suitable for linear regression in this case, or perhaps the removal of outliers before using LS.

4.6.5 Quad-Plots for Outlier Diagnosis

The four visualization methods for detecting outliers can be combined into a “quad-plot” for residuals. The quad-plot encapsulates all four diagnostic methods, namely, the OAR-plot, boxplot, histogram, and QQ-plot. By examining all four plots together, it is much easier to determine whether or not the dataset has outliers. To illustrate the process, we compare the Swiss fertility dataset (which has no outliers) with the Boston Housing dataset (which has over 50 outliers).

The quad-plot for the Swiss Fertility dataset based on LC residuals is given in Figure 4.17. If we examine each one separately, we may get conflicting results. For example, the boxplot indicates that there are no outliers. This is confirmed by the QQ plot. The histogram looks Gaussian in nature and does not appear to have outliers either. However, the OAR-plot indicates there may be three outliers. But the QQ-plot tells us that they are just three extreme values (one at the top and two at the bottom) that should not be considered as outliers.

For the Boston Housing dataset, the residuals of the LC estimator are presented as a quad-plot in Figure 4.18. These plots give a clear picture regarding the existence of outliers in contrast to the case for the Swiss dataset. The boxplot provides the most compelling evidence of outliers, but the QQ-plot and OAR-plot also support the claim of a large number of outliers. The histogram skews to one side which is also indicative of the potential for outliers. We now have an effective method to detect outliers using residuals from a robust estimator. However, surgically removing outliers without losing any inliers is another matter, as we will address in the next section.

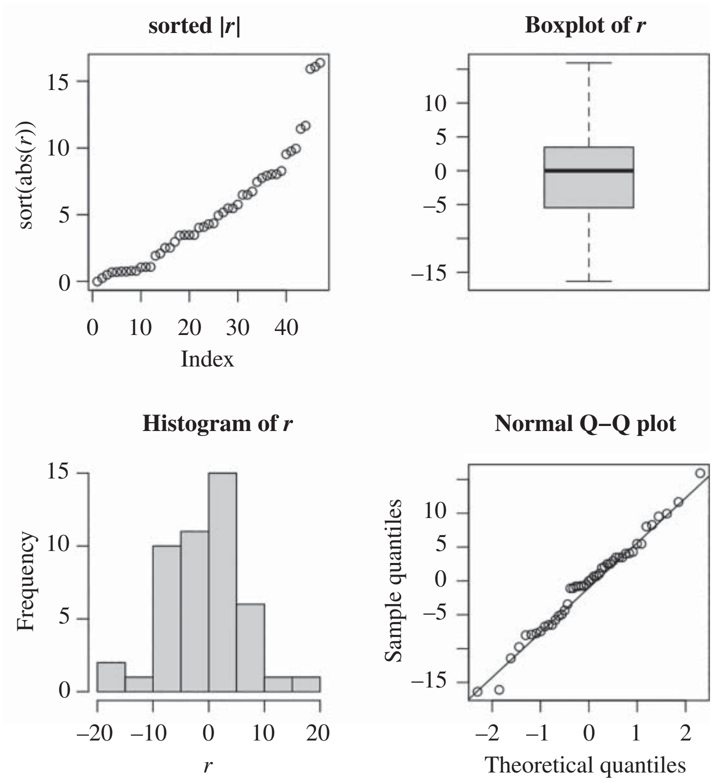


Figure 4.17 Quad-plot diagnosis of the Swiss LC residuals, **r**.

4.7 Regression-Based Outlier Removal

The task of actually removing outliers is fraught with peril. It is a risky business which is why we have been advocating the use of robust methods. However, many users of machine learning tools prefer to remove outliers rather than switch to robust methods. Some statistical methods require outlier-free data. There may also be no choice in some settings other than to remove outliers so we must develop methods that are fast and accurate for the intended purposes. We describe a method for surgically removing outliers using robust LC regression and the boxplot metrics.

4.7.1 Iterative Boxplot Method

Most methods in use currently are unable to separate the "wheat from the chaff" with any precision. Usually some outliers are missed and some useful data is thrown out. Here, we are trying to remove only

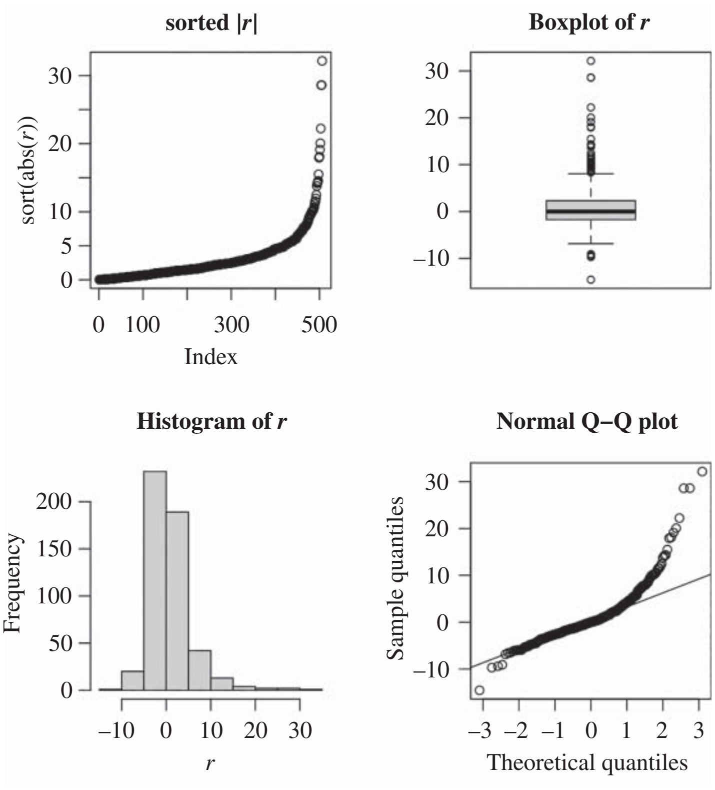


Figure 4.18 Quad-plot of the Boston Housing LC residuals, **r**.

outliers while leaving the valuable data intact. Note that standard methods for finding outliers are not as effective because most of the outliers can only be found *after a robust regression is performed and the residuals are computed*.

The Iterative Boxplot method is a fast and accurate alternative that can be used to remove outliers of a dataset. The idea is to use the boxplot method in an iterative loop to remove outliers at each step until none are left. It is derived from the 1.5-IQR method but uses `Ifactor`-IQR instead. The value of `Ifactor` starts at 1.5 and is steadily increased by $50/n$ in each iteration based on the data size, n. This is a heuristic value but it works well in practice over many datasets. Each additional pass takes the partially cleaned data and removes any remaining outliers. Eventually, all of the outliers can be detected and removed with enough iterations, typically around one to four iterations. The procedure given in Algorithm 4.1 is fast, reliable, and more precise than previous algorithms. For the diabetes dataset, it removes all three outliers. In the Boston Housing dataset, it finds and removes 52 outliers. In the Swiss dataset, it is unable to find outliers because there are none. The results for the Iterative Boxplot method are given in Table 4.2.

Table 4.2 Outliers detected and removed by the iterative boxplot method on three datasets.

	Swiss Fertility	Diabetes Data	Boston Housing
Number of outliers removed	0	3	52

Algorithm 4.1 Iterative boxplot method for outlier removal.

```
1: **Input:** entire dataset $\boldsymbol{X}, \boldsymbol{y}$;
2: set n=number of observations, d=number of variables,
       outliers= $n$, Ifactor=1.5
3: **while** outliers > 0 do
4:     run LC regression
5:     compute residuals $r_i = y_i - \boldsymbol{X}_i^\top \hat{\boldsymbol{\beta}}^{\mathrm{LC}}$ for $i = 1, \dots, n$
6:     compute U = Q3 + Ifactor×IQR for residuals
7:     compute L = Q1 - Ifactor×IQR for residuals
8:     compute outliers = $\sum_{i=1}^{n}(r_i > U) + \sum_{i=1}^{n}(r_i < L)$
9:     **for** $i = 1, \dots, n$
10:        remove row $i$ from dataset if $r_i > U$ or $r_i < L$
11:    **end for**
12:    Ifactor = Ifactor + 50/n
13: **end while**
14: run LS regression
15: use quad-plot of LS residuals on reduced dataset as a final check
16: **Output:** outlier-free dataset;
```

4.8 Regression Metrics with Outliers

A variety of metrics are used to evaluate the performance of various machine learning tools relative to one another. These metrics are statistical in nature and some are prone to be misleading if not used carefully, especially with outliers in datasets. In particular, if a robust metric such as MAE is not used, then it is possible that a non-robust regression such as LS will look better according to the MSE metric, when in reality a robust regression is performing better. How can we tell if a metric is not robust? As a general rule, every time the word “mean” appears in a metric, it should be interpreted as non-robust and somewhat unstable in the presence of outliers. The goal of this section is to compare and contrast MSE and MAE and provide recommendations for their proper usage.

4.8.1 Mean Square Error (MSE)

The MSE is one of the most popular metrics used with LS. It is given by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n} r_i^2, \tag{4.8.1}$$

where n is the number of data points in the dataset, $r_i = y_i - \hat{y}_i$, and y_i represents the ith true response while $\hat{y}_i$ is the predicted response. There is a problem using this metric with outliers that we will illustrate using the Belgium telephone dataset (see Section 4.2). For simple linear regression using least squares and log-cosh methods, which one provides a smaller error?

The results of the regressions and histograms of the associated residuals of LS and LC are plotted in Figure 4.19. If we compute the MSE for the two cases, we find that for LS regression, MSE = 29.0, and for LC regression, MSE = 42.4. This informs us that LS is better than LC. Unfortunately, the opposite is actually true. The six outliers in the dataset force the LS regression to produce a line that does not represent the true data. Consider the values of the two residuals shown in the figure as r_i. In the LS case, the residuals are all smaller due to the proximity of the line to the outliers. However, in LC, there is a large distance to the outliers from the regression line so the MSE will necessarily be higher. This is also shown clearly by examining the two histograms in Figure 4.19. In the LC case, the large residuals will lead to a large MSE. Therefore, we must not be tempted to use MSE as a reliable metric with outliers.

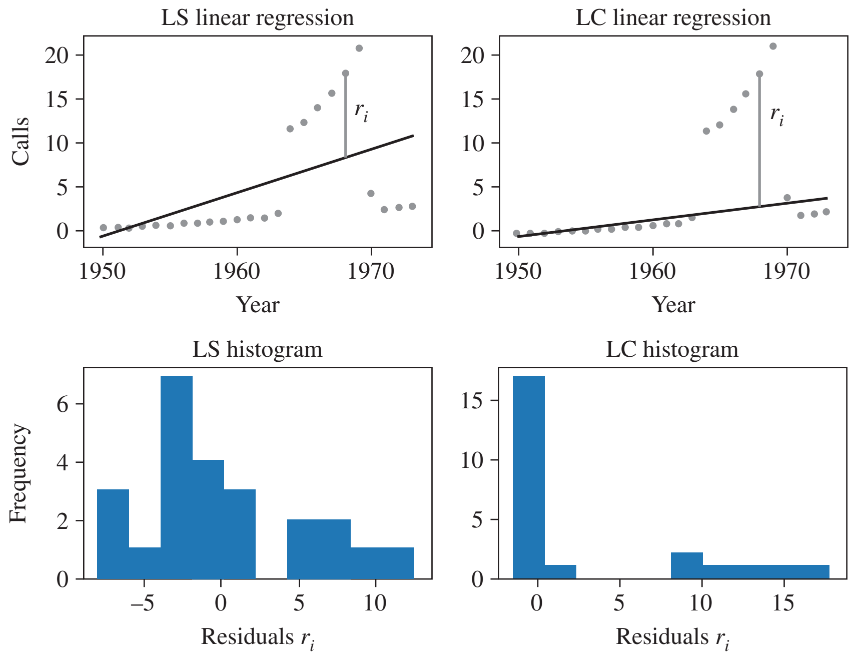


Figure 4.19 Regression lines and residual histograms for LS and LC on the Belgium telephone data with six outliers.

Another problem with the MSE metric is that the units are not the same as the response variable. This can be fixed by using the RMSE which produces the same units of measure as the response variable. The RMSE is given by

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} r_i^2}. \tag{4.8.2}$$

For the two cases of Figure 4.19, RMSE = 5.4 for LS and RMSE = 6.5 for LC. Of course, there is no change in the outcome of which method is better according to the RMSE metric. Again, LS is better than LC based on the RMSE, which is misleading. The reason is that the line fitted by LC is further away from the outliers and this gives rise to a higher RMSE. This will always be the case since the outliers will be far from the regression line or hyperplane in robust regression. On the other hand, LS will try to balance the residuals of the outliers and inliers to reduce the RMSE. Therefore, the MSE or RSME statistic should only be selected when working with outlier-free datasets.

4.8.2 Median Absolute Error (MAE)

We need to find a better evaluation statistic with outliers in mind. A robust metric is the residual MAE given by

$$\text{MAE} = \text{median}(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \dots, |y_n - \hat{y}_n|). \tag{4.8.3}$$

Here, the median is used so it is a robust error measure. If the MAE is computed on the same dataset as in Figure 4.19, then for LS, we have MAE = 3.45 and for LC, we have MAE = 0.49. Hence, the results favor LC by a substantial margin. This robust metric is much more suitable for datasets with outliers as it takes into account that the robust method would necessarily have larger residuals due to outliers. This was seen earlier in the residual histograms of Figure 4.19.

Another metric often used is the mean absolute error (MnAE) given by

$$\text{MnAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|r_i|. \tag{4.8.4}$$

Typically, this metric uses the same acronym as the MAE. Both are referred to in the literature as MAE. To avoid confusion between the two, we use MnAE.[3] Beware that this metric is not robust because it uses the mean. It is somewhat better than MSE since it is not squaring the errors so the difference is not exaggerated. In the case of Figure 4.19, the MnAE = 4.24 for LS and MnAE = 3.98 for LC. In this case, it slightly favors LC even though this metric is not robust. However, it can favor LS in other cases even when LC is better. Therefore, it is not as reliable a metric as MAE when comparing LS to LC. The MAE is the only truly robust metric.

Now consider Figure 4.20 in which there are no outliers. For this case, the MSE = 0.169 and MAE = 0.055 for LS, whereas MSE = 0.170 and MAE = 0.054 for LC. They are almost identical which

3 Readers should be careful when using the acronym "MAE" to ensure it is the median absolute error being used and not inadvertently the mean absolute error.

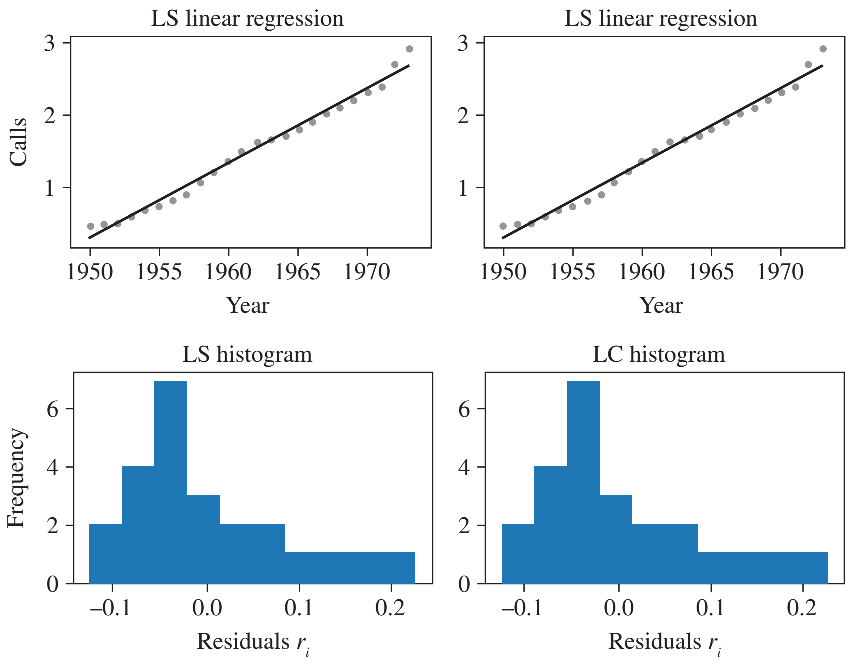


Figure 4.20 Regression lines and residual histograms for LS and LC on the Belgium telephone data (modified version with no outliers).

indicates that LS and LC perform about the same without outliers. This is important because we would like to use the MAE statistic even when there are no outliers. Usually, when comparing two or more different regression methods, both the MSE (or RMSE) and MAE should be computed. However, more weight should be placed on the MAE than MSE with outliers since the MAE is robust.

4.8.3 MSE vs. MAE on Realistic Data

The results from the analysis of the previous section using a simple dataset indicate that MAE is preferable over MSE for comparison purposes when robust methods are involved. This means that any regression method with the lowest MAE should be selected over other methods. This will now be tested on bigger datasets, namely, the Swiss Fertility (no outliers), diabetes (3 outliers), and Boston Housing (over 50 outliers) datasets. The bar charts in Figure 4.21 show the relative MSE and MAE values for the three different datasets. The values have been **normalized to 1** since they all have different orders of magnitude of error. In the left panel, we note that the MSE indicates that LS is better than LC in all three cases since the MSE is smaller. This may be true in a marginal sense in the case of the Swiss dataset. However, the diabetes and Boston Housing data both contain outliers so they should perform better with LC. Unfortunately, the MSE would lead us to believe that LS is better even in the presence of outliers.

The panel on the right has a more accurate view of the situation. The MAE ratio between LS and LC continues to increase as the number of outliers increases. Specifically, the Boston Housing data with

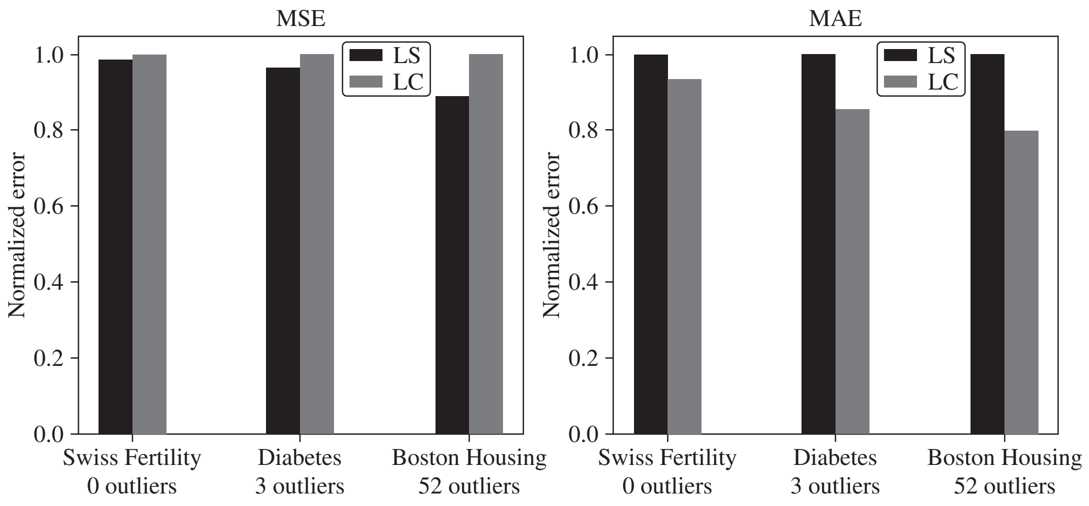


Figure 4.21 MSE vs. MAE for both LS and LC on Swiss, diabetes, and Boston Housing datasets.

52 outliers and diabetes data with 3 outliers are better served using LC rather than LS according to the MAE metric. More generally, if there are many robust methods to choose from, the one with the lowest MAE should be selected rather than the one with the lowest MSE.

4.8.4 Selecting Hyperparameters for Robust Regression

Now that the MAE metric and its proper use have been described, it can immediately be applied to the selection of the robustness coefficient a for linear regression using log-cosh and δ for Huber's method. Recall from Chapter 3 that the general log-cosh loss has the form

$$J^{\mathrm{LC}}(x; a) = \frac{1}{a}\sum_{i=1}^{n} \log(\cosh(ax)). \tag{4.8.5}$$

A suitable value for a can be obtained by selecting six different values and comparing the MAE values. In addition, since this loss function is meant to deliver a median solution, the number of points below the line or hyperplane should be 50% of the data. This criteria can also be used to select the value of a.

Table 4.3 provides the MAE and number of points below the line (and percentage) for the Boston Housing dataset using six different values of a. Effectively, we are comparing six different robust estimators from the same family. The results indicate that $a = 2$ is a suitable value. Below this value, the MAE is higher and the "points below the line" exceeds 50% which does not satisfy the definition of the median. Above this value, the "points below the line" drops below 50%. Clearly, there is a trade-off between the MAE and the need to satisfy the requirements of the median. The value of $a = 2$ is a good balance between the two objectives and is applicable with all datasets.

Table 4.3 Finding a suitable a-factor for log-cosh linear regression using the Boston Housing dataset.

	$a = 0.5$	$a = 0.7$	$a = 1.0$	$a = 1.5$	$a = 2$	$a = 10$
MAE	2.010	2.000	1.988	1.979	1.962	1.939
Points below line	263	259	255	254	253	252
Percentage of data	(52.0%)	(51.2%)	(50.4%)	(50.2%)	(50.0%)	(49.8%)

Table 4.4 Finding a suitable δ-factor for Huber linear regression using the Boston Housing dataset.

	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.5$	$\delta = 1.345$	$\delta = 2$
MAE	1.975	1.962	1.979	2.017	2.080
Points below line	251	253	256	274	283
Percentage of data	(49.6%)	(50.0%)	(50.6%)	(54.1%)	(55.9%)

Next, recall Huber's method from Chapter 2 as

$$\text{H-loss}(x; \delta) = \sum_{i=1}^{n} \rho_H(x), \tag{4.8.6}$$

where

$$\rho_H(x) = \begin{cases} x^2/2 & \text{if } |x| \le \delta \\ \delta\left(|x| - \frac{\delta}{2}\right) & \text{if } |x| > \delta. \end{cases} \tag{4.8.7}$$

Using the same procedure as above, we can find the most suitable value of δ for the Boston Housing dataset. The results are shown in Table 4.4. In this case, $\delta = 0.2$ is the proper compromise between MAE and points below the hyperplane. The MAE value is the same as for log-cosh with $a = 2$. Unfortunately, the value of δ is specific to the Boston Housing dataset. For the Swiss dataset, the suitable value is $\delta = 4$ whereas for the diabetes dataset, it is $\delta = 1.2$. A value of $\delta = 1.345$ is typically used as the default but ideally it should be adjusted on a case-by-case basis. This is one of the drawbacks of Huber's method.

4.9 Dataset Standardization

An essential preprocessing step in every data science problem is data transformation. When a dataset is initially received, various transformations are necessary to make it suitable for machine learning tools. One crucial transformation is standardization, which simplifies the tasks of downstream machine learning algorithms. During this step, explanatory variables with differing units and ranges are standardized to ensure consistency and comparability.

Standardization is required in certain machine learning tasks such as logistic regression, neural networks, anomaly detection, and determining variable importance. For example, during gradient descent in logistic regression or neural networks, the iterations may not converge if the data is not normalized[4] in some way. Certain anomaly detectors do not work well unless the data is standardized. It is also good practice to standardize the data (unless they are categorical data with discrete values) in order to obtain reliable information about variable importance.[5]

Standardization is applied to the data on a column-by-column basis using the formula for z-scores as follows:

$$\boldsymbol{z}_j = \frac{\boldsymbol{x}_j - \hat{\mu}_j}{\hat{\sigma}_j}, \quad \text{for} \quad j = 1, \dots, d, \tag{4.9.1}$$

where $\hat{\mu}_j$ and $\hat{\sigma}_j$ are the computed mean and standard deviation of the jth column, respectively. Essentially, we are making all the explanatory variables behave as standard normal variables such that $\boldsymbol{Z} \sim \mathcal{N}_d(\boldsymbol{0}, \boldsymbol{I}_d)$. This implies that the optimization space is much more uniform and leads to faster convergence.

The results of such a procedure can be visualized as shown in Figure 4.22 for the Swiss Fertility, diabetes, and Boston Housing datasets. In each case, the non-standardized and standardized values are shown side by side. The original variables (left panels) and the z-scores (right panels) are plotted horizontally and stacked one on top of the other for ease of comparison. The explanatory variables are labeled from $\boldsymbol{x}_1$ to $\boldsymbol{x}_d$ in each case. The unstandardized values could have a different ranges and completely different units of measure, as in the case of the Boston Housing.[6] Therefore, it is mandatory to standardize this data set. The values for Swiss Fertility are roughly in the same range so standardization may be optional here. The diabetes values are in the same range so standardization is not needed. For illustrative purpose, we apply standardization to all of them.

After standardization, a majority of the standardized values fall between −4 and 4. There are a few values that lie outside this range (indicated with a square) which we may be tempted to declare as outliers. They are potentially high-leverage points which could be removed. However, they may also be valuable data points which we may want to keep. Robust methods are better-equipped to handle these types of outliers properly, as described in Chapters 2 and 3.

4.9.1 Robust Standardization

The above approach is not suitable when the variables have outliers. This is because outliers will compromise the values of $\hat{\mu}_j$ and $\hat{\sigma}_j$. In fact, as mentioned throughout the book, when any variable is treated

4 Standardization and normalization are used interchangeably in machine learning although they may have slightly different meanings in other contexts.
5 Note that if standardization is performed in regression or classification problems, then we must unstandardize the coefficients to revert back to the original units.
6 One should be careful not to standardize binary variables such as `CHAS` in the Boston dataset.

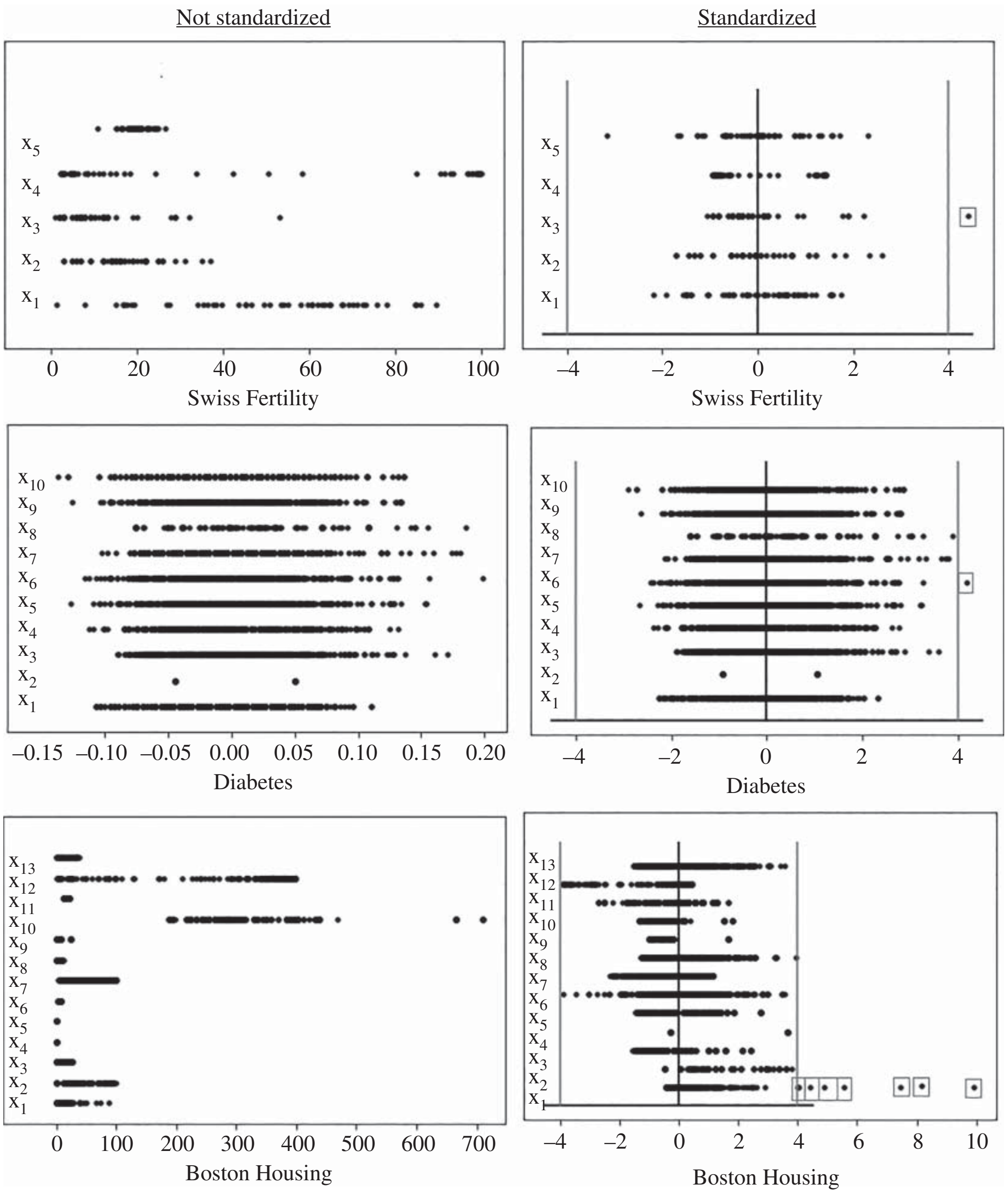


Figure 4.22 Non-standardized and standardized variable plots for Swiss Fertility, diabetes, and Boston Housing datasets.

as if it was drawn from a Gaussian distribution, the outliers (if any) tend to be hidden by a shift in $\hat{\mu}$ and an increase in $\hat{\sigma}$. In other words, the distribution is modified to accommodate outliers as if they were inliers. This was shown earlier in Figure 4.8. Therefore, we must turn to a robust alternative instead. In particular, the mean and standard deviation should be replaced by

$$\tilde{\mu}_j = \text{median}(\boldsymbol{x}_j) \tag{4.9.2}$$

and

$$\tilde{\sigma}_j = \text{MADN}(\boldsymbol{x}_j), \tag{4.9.3}$$

respectively. These are the appropriate choices for non-Gaussian distributions of explanatory variables. Then, the robust z-scores are obtained using

$$\boldsymbol{z}_j^{\text{robust}} = \frac{\boldsymbol{x}_j - \tilde{\mu}_j}{\tilde{\sigma}_j}, \quad \text{for} \quad j = 1, \ldots, d. \tag{4.9.4}$$

One caveat on the use of robust standardization is that it cannot be applied blindly to binary or categorical data. The reason is that it may result in $\text{MADN}(\boldsymbol{x}) = 0$ which presents a problem in Eq. (4.9.4) as it appears in the denominator. Instead, for these types of variables, the options are to use the standardization based on the non-robust equations presented earlier or to leave them alone as unstandardized variables.

Exercise: Given $\boldsymbol{x}_1 = (0, 1, 1, 0, 0, 1, 0)^\top$, compute $\text{median}(\boldsymbol{x}_1)$ and $\text{MADN}(\boldsymbol{x}_1)$. What problem exists if we attempt to compute the robust z-scores? To resolve this issue, compute the standard z-scores for this variable or suggest another approach.

Ans.: We find that $\text{median}(\boldsymbol{x}_1) = 0$ and $\text{MADN}(\boldsymbol{x}_1) = 0$ so Eq. (4.9.4) is not applicable. However, the mean is $\hat{\mu} = 0.429$, and the standard deviation is $\hat{\sigma} = 0.535$. Therefore, the z-scores of $\boldsymbol{x}_1$ are $\boldsymbol{z}_1 = (-0.802, 1.069, 1.069, -0.802, -0.802, 1.069, -0.802)^\top$. Or, we could skip the entire process and simply use $\boldsymbol{x}_1 = (0, 1, 1, 0, 0, 1, 0)^\top$.

The effects of robust standardization compared to Gaussian standardization are illustrated in Figure 4.23. For the Swiss and Diabetes datasets, the differences are not pronounced but it is clear that more outliers remain noticeable and influential in the case of robust standardization. On the other hand, the Boston dataset shows a stark difference between the two approaches. Most of the outliers have been confined to the region between −4 and 4 in the case of Gaussian standardization whereas most, if not all, of the outliers have been preserved after robust standardization. It is hard to detect outliers if the process of standardization reduces their "outlierness." In fact, most anomaly detectors require some form of standardization of the data before their application and it would be counterproductive if that process tended to hide the outliers. That is why Gaussian standardization is problematic and robust standardization is preferable.

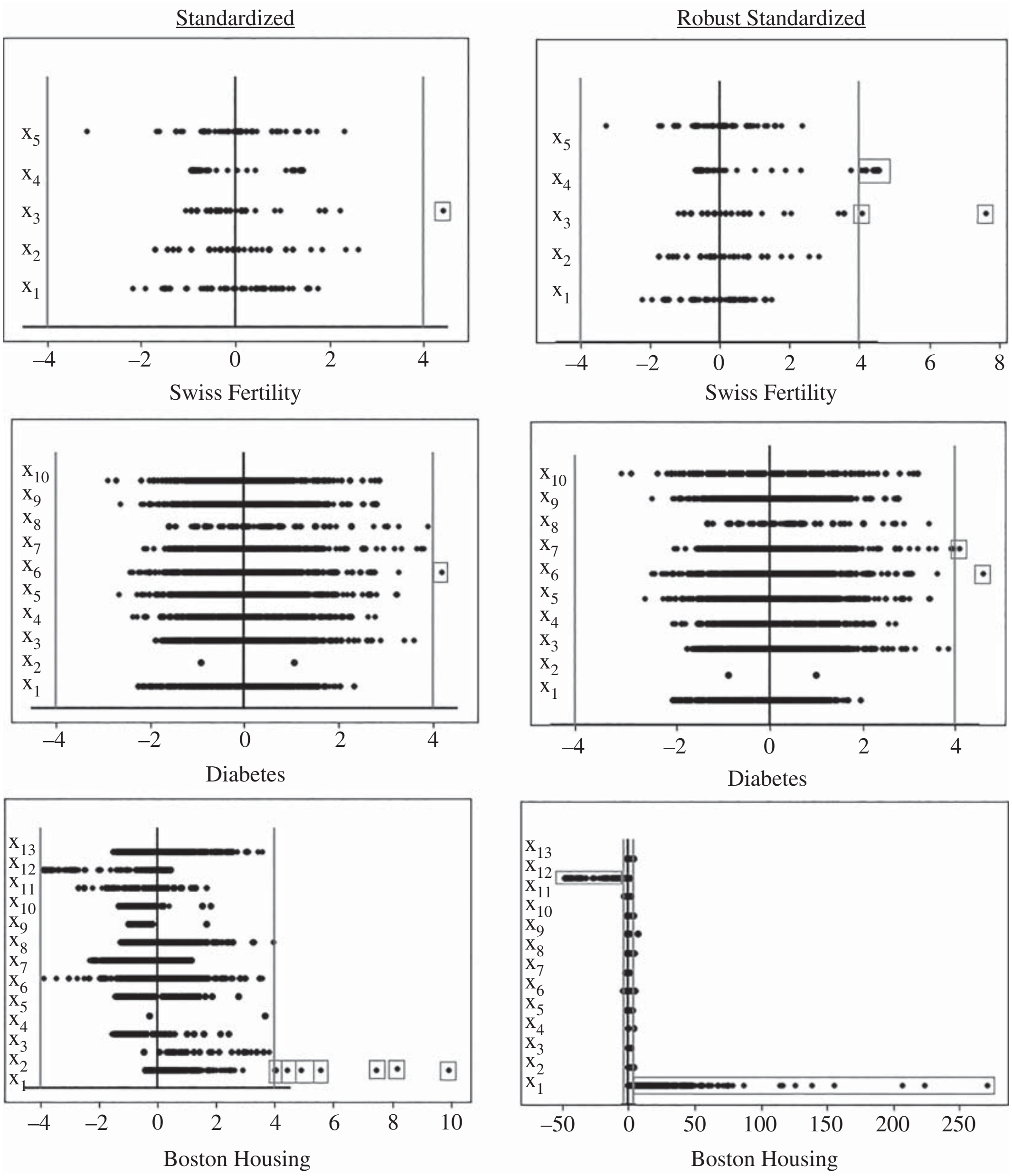


Figure 4.23 Gaussian standardized and robust standardized variable plots for Swiss Fertility, diabetes, and Boston Housing datasets.

4.10 Summary

This chapter began with methods for diagnosis, detection, and removal of outliers. Often, there is a need to have outlier-free datasets since much of the current infrastructure for machine learning is built on this assumption. There are also many statistical procedures that require clean datasets. One of the key points of this chapter is that the best way to deal with outliers is to examine the residuals of a robust regression rather than the explanatory and response variables.

The diagnosis phase involves checking whether any outliers exist in the dataset. Boxplots and histograms are useful for this purpose. In detection, the goal is to determine the approximate number of such outliers. Here, various edit rules can be applied. Robust rules, such as 4.5-MAD or 1.5-IQR, are most effective. A quad-plot for residuals that incorporates a boxplot, a QQ-plot, a histogram, and ordered absolute residuals was introduced. In the removal phase, the process of deleting outlier rows of the dataset must be done carefully or risk removing rows with valuable data. An iterative boxplot approach was described for this purpose. This is a simple but effective method of removing outliers without inadvertently removing inliers.

The chapter concluded with the topics of robust metrics and standardization. Robust metrics should be used alongside non-robust metrics in order to ensure that models are accurate. In particular, MAE is robust and should always be used alongside MSE or RMSE. In addition, standardization should be carried out using robust statistics to retain information about outliers, as traditional methods tend to hide them.

Problems

4.1 Let $x = \{0, 1, ..., 100\}$. Find the values of Q1, Q2, and Q3, associated with the boxplot. Find the upper and lower values of the whiskers of the boxplot. Draw the boxplot associated with this data. What is the minimum value of a new data point x_{new} that would be considered an outlier if $x_{\text{new}} > 0$? What is the maximum value of a new data point to be considered an outlier if $x_{\text{new}} < 0$?

4.2 The Python code below generates a boxplot for the given data. Run this code to examine the resulting boxplot. To understand the workings of a boxplot, change the existing data (without adding any new data) such that it produces roughly the same boxplot as the right-most boxplot of Figure 4.3. (Hint: consider changing outlier values so they fit into the box portion of the boxplot.)

```
import numpy as np
import matplotlib.pyplot as plt
data = np.array([  0.7, 1.9,   2.0,   9.4,  10., \
    10.5,  10.8,  11.2, 11.3,  \
    12.4,  12.6,  13.5,  14.6,  \
    15.7,  16.3,  18.1, 20.0,  \
    21.4,  37.,  38.,  45., \
    108, 111, 127, \
    143, 192  ])
```

```
fig = plt.figure(figsize =(10, 7))
plt.boxplot(data)
plt.show()
```

4.3 For the original data in the previous problem, generate a histogram using the following code.

```
plt.hist(data,bins=1)
plt.show()
```

What is a suitable value for the number of `bins` and the threshold to detect five outliers?

4.4 Let $y = \{-0.26, 0.62, -1.47, -0.56, -0.83, 0.59, 1.41, 0.18, -0.40\}$. Create a QQ-plot using y and determine if it is Gaussian. In order to do this, you will need to find the inverse CDF of the Gaussian distribution for the quantiles $q = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. For example, $x_1 = \Phi^{-1}(0.1) = -1.28$. The same operation is applied to all quantiles in q. To obtain a QQ-plot, simply plot $\boldsymbol{x}$ vs. $\boldsymbol{y}$.

4.5 (PROJECT) Consider solving a linear regression problem associated with the Belgium telephone dataset of Chapter 2 using the least squares, log-cosh, and Huber estimators in Python. The goal is to compare the two metrics, MSE and MAE, and their suitability for datasets with outliers. First, the least squares approach will be used to find the MSE and MAE using the following Python code.

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
X = np.linspace(1950, 1973, 24).reshape(-1, 1)
Y = np.array([ 0.44,  0.47,  0.47,  0.59,  0.66, \
              0.73,  0.81,  0.88,  1.06, 1.2 , \
              1.35,  1.49,  1.61,  2.12, 11.9 , \
              12.4 , 14.2 , 15.9 , 18.2 , 21.2 , \
              4.3 ,  2.4 ,  2.7 ,  2.9 ])
column_ones = np.ones((24, 1))
X = np.hstack((column_ones, X))
def LS(b):
    ri = Y-np.matmul(X,b)
    rhoLS = ri*ri/2
    loss = np.sum(rhoLS)
    return loss
initbetaLS = np.array([0.,0.])
minval = minimize(LS,initbetaLS)
betaLS = minval.x
Y_pred = betaLS[0]+ betaLS[1]*X[:,1]
print("Least squares estimation")
print("Mean squared error", \
      format(np.mean((Y - Y_pred)**2),‘.3f’))
print("Median absolute error", \
    format(np.median(np.abs(Y - Y_pred)),‘.3f’))
```

Next, the log-cosh loss can be used to perform robust regression. Run the following code and find the MSE and MAE.

```
logcosh = lambda x: np.logaddexp(x, -x) - np.log(2)
a = 2
def LC(b):
    ri = Y-np.matmul(X,b)
    loss = np.sum((1/a)*logcosh(a*ri))
    return loss
initbetaLC = np.array([0.,0.])
minval = minimize(LC,initbetaLC)
betaLC = minval.x
Y_pred = betaLC[0]+ betaLC[1]*X[:,1]
print("log-cosh estimation")
print("Mean squared error", \
      format(np.mean((Y - Y_pred)**2),‘.3f’))
print("Median absolute error", \
    format(np.median(np.abs(Y - Y_pred)),‘.3f’))
```

Finally, Huber’s method can also be used to perform robust regression. Run the following code and find the MSE and MAE.

```
def Huber(b):
    ri = Y-np.matmul(X,b)
    n = len(ri)
    rhoH = np.zeros(n)
    for i in range(n):
        if np.abs(ri[i]) <= delta:
            rhoH[i] = ri[i]*ri[i]/2
        else:
            rhoH[i] = delta*(np.abs(ri[i])-delta/2)
    loss = np.sum(rhoH)
    return loss
initbetaH = np.array([0,0])
delta = 1.345
minval = minimize(Huber,initbetaH)
betaH = minval.x
Y_pred = betaH[0]+ betaH[1]*X[:,1]
print("Huber estimation")
print("Mean squared error", \
      format(np.mean((Y - Y_pred)**2),'.3f'))
print("Median absolute error", \
    format(np.median(np.abs(Y - Y_pred)),'.3f'))
```

Based on the results of least squares, log-cosh, and Huber, which metric is better for datasets with outliers such as the Belgium telephone dataset? For Huber, how do the results compare when δ is set to 4.0 and 10.0? Explain why the MAE value is increasing as δ increases.

4.6 (PROJECT) In this project, the task is to generate quad-plots of the different visualization methods given in this chapter.

a) Import the needed libraries.

```
import matplotlib.pyplot as plt
import scipy.stats as stats
import pylab
import statsmodels.api as sm
%matplotlib inline
```

b) Generate the quad-plot for a synthetic set of residuals by sampling a Gaussian distribution. It has the same number of residuals as the Swiss dataset. Repeat the process several times and comment on the results.

```
#remove the following line for later projects
r1 = np.random.randn(47)

fig, ax = plt.subplots(nrows=2, ncols=2)
# row 0, col 0
ax[0, 0].boxplot(r1)
# row 1, col 0
ax[1, 0].plot(np.sort(np.abs(r1)), \
        linestyle='None', marker='o',markersize=2)
# row 0, col 1
ax[0, 1].hist(r1)
fig = stats.probplot(r1, dist="norm", plot=pylab)
plt.tight_layout()
pylab.show()
plt.show()
```

4.7 (PROJECT) In this project, the task is to develop a Python program to implement the iterative box-plot method to remove all outliers in the diabetes dataset.

a) Import the needed libraries.

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
```

b) Load the dataset and define the ***X*** and ***y*** values.

```
from sklearn.datasets import load_diabetes
diabetes_dataset = load_diabetes()

diabetes = pd.DataFrame(diabetes_dataset.data, \
      columns=diabetes_dataset.feature_names)
diabetes['Y'] = diabetes_dataset.target
diabetes["ones"] = 1
X = np.array(diabetes.drop(["Y"],axis=1))
Y = np.array(diabetes["Y"])
```

c) Define the needed function for the log-cosh loss (reuse the code given in the previous problem).

d) Implement the iterative IQR code.

```
n = X.shape[0]
d = X.shape[1]
outliers = n
num_outliers = 0
Ifactor = 1.5
while (outliers > 0):
    initbetaLC = np.random.randn(d)/10.
    result = minimize(LC,initbetaLC,method='BFGS')
    betaLC = result.x
    r1 = Y-np.matmul(X,betaLC)

    quantiles = np.percentile(r1,[25, 50, 75])
    m = quantiles[1]
    IQR = quantiles[2]-quantiles[0]
    IQR_upper = quantiles[2]+Ifactor*(IQR)
    IQR_lower = quantiles[0]-Ifactor*(IQR)

    outliers = np.sum(r1 < IQR_lower)+ \
        np.sum(r1 > IQR_upper)

    num_outliers += outliers
    Ifactor += 50/n
    if (outliers > 0):
        k = 0
        outliers = np.sum(r1 > IQR_upper) + \
            np.sum(r1 < IQR_lower)
        rows_to_delete =  np.zeros(outliers)
        for i in range(X.shape[0]):
            if(r1[i] \
                  < IQR_lower or r1[i] > IQR_upper):
                  rows_to_delete[k] = i
                  k += 1
          # delete rows from X
          X = np.delete(X, \
              rows_to_delete.astype(int), \
              axis=0)
          # delete rows from y
          Y = np.delete(Y, \
              rows_to_delete.astype(int), \
              axis=0)
  print("Number of Outliers detected and removed:", \
      num_outliers)
```

e) How many outliers are detected and deleted by this program?

4.8 (ADVANCED PROJECT) Run the Boston Housing dataset through the iterative boxplot code given in Problem 4.6. The input data is provided below. Generate the quad-plots before and after the removal of outliers. How many outliers are removed?

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_boston
boston_dataset = load_boston()
boston = pd.DataFrame(boston_dataset.data, \
      columns=boston_dataset.feature_names)
boston["MEDV"] = boston_dataset.target
boston["ones"] = 1
X = np.array(boston.drop(["MEDV"],axis=1))
Y = np.array(boston["MEDV"])
initbetaLC = np.random.randn(14)
minval = minimize(LC,initbetaLC)
betaLC = minval.x
r1 = Y-np.matmul(X,betaLC)
```

You can use the code from Problem 4.6 to generate the quad-plot with the new value of `r1` produced by the above code. Then run the outlier removal code of Problem 4.7 and produce a new quad-plot using a new value of `r1`.

References

Aggarwal, C.C. (2017). *Outlier Analysis*, 2e. Springer.

Maronna, R.A., Douglas Martin, R., Yohai, V.J., and Salibian-Barrera, M. (2019). *Robust Statistics, Theory and Methods (with R)*. Wiley.

5

Robustness of Penalty Estimators

5.1 Introduction

The loss functions L_1, L_2, Huber, and log-cosh described in Chapters 2 and 3 provide reasonable models for prediction but they can be improved through a procedure called *regularization*. In this procedure, a penalty is added to a loss function so that parameter values can be fine-tuned to produce a better model. Using a penalty function with a loss function provides a range of models to choose from and the goal becomes one of finding optimal model that produces the lowest prediction error. In effect, we seek a model that does not overfit or underfit a given training set, but rather one that strikes a balance between the two. Penalty functions can also be used to reduce model complexity. In this chapter, we explore the basic properties and characteristics of a variety of different penalty functions that can be used to regularize regression models. In the next chapter, the issues of overfitting and underfitting are explored in detail along with the procedures that use penalty functions to control model complexity.

5.2 Penalty Functions

Penalty functions (see Saleh et al. 2019, 2022; James et al. 2023) and their effective use are very important in machine learning, especially for linear and logistic regression models and, to some extent, neural networks. Unfortunately, they are not used as often as they should be because many of their key benefits are not well-understood. While the primary benefit stems from their use in regularization, they have a number of other advantages. For example, they can be used to mitigate the effects of collinearity, reduce parameter variance, identify important features, or simply keep the values of the parameters under control. These and other capabilities will be covered in this chapter. In the context of outliers, we will delve into whether or not penalty functions can help reduce the impact of outliers. In this section, we describe the origins of penalty functions, how they are derived, their functional form, and their main properties.

5.2.1 Multicollinearity

The original motivation for penalty functions was the problem of multicollinearity. To understand this concept, consider the closed-form least squares (LS) solution in matrix form given by

$$\hat{\boldsymbol{\beta}}^{\mathrm{LS}} = (\boldsymbol{X}^{\top}\boldsymbol{X})^{-1}\boldsymbol{X}^{\top}\boldsymbol{y}, \tag{5.2.1}$$

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.

Companion Website: www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

which was derived in Chapter 2. Here $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{y} \in \mathbb{R}^n$. The $\boldsymbol{X}$ matrix has been standardized (see Chapter 4) such that the intercept is 0. Therefore, the intercept is not included in Eq. (5.2.1). Hence, estimates of the regression coefficients are given by $\hat{\boldsymbol{\beta}}^{\text{LS}} = (\hat{\beta}_1^{\text{LS}}, \dots, \hat{\beta}_d^{\text{LS}})^\top$. A key requirement of this formulation is that the matrix $\boldsymbol{X}^\top\boldsymbol{X}$ be invertible to produce the solution. Recall that $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$ is also used in constructing the covariance matrix so the invertibility of $\boldsymbol{X}^\top\boldsymbol{X}$ deserves some discussion.

Two potential problems may arise when applying Eq. (5.2.1). First, if $d > n$, then the matrix $\boldsymbol{X}^\top\boldsymbol{X}$ is not invertible and the parameters are indeterminate. This is easy to check and can be resolved by either adding more data points or reducing the number of parameters. Second, if any of the variables are linearly related, then $\boldsymbol{X}^\top\boldsymbol{X}$ may not be invertible and the parameters are again indeterminate. This second case gives rise to the *collinearity* problem, which is harder to solve.

As a simple illustration, let $d = 20$ such that we have 20 column vectors in the $\boldsymbol{X}$ matrix given by $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{20}$. Further, assume that

$$\boldsymbol{x}_7 = 2\boldsymbol{x}_2,$$
$$\boldsymbol{x}_1 = 0.6\boldsymbol{x}_{11} + 0.4\boldsymbol{x}_4.$$

Then, we will encounter collinearity in $\boldsymbol{X}$ due to the first equation, since $\boldsymbol{x}_2$ and $\boldsymbol{x}_7$ are linearly related. Furthermore, we will encounter *multicollinearity*[1] in $\boldsymbol{X}$ due to the second equation, since $\boldsymbol{x}_1$, $\boldsymbol{x}_4$, and $\boldsymbol{x}_{11}$ are all linearly related. The respective variables in both equations are said to be highly correlated and, therefore, the matrix will not be invertible for reasons cited above. Strong correlations between variables are manifested as large off-diagonal values in the covariance matrix making it harder to invert.

Multicollinearity occurs when three or more variables exhibit a strong relationship even though any two variables appear not to have any correlation. If there are hundreds of variables, there is likely to be some form of multicollinearity. This is a more difficult condition to identify but may lead to instability in the parameter values and make interpretability of the results more challenging. Outliers could also play a role in this context. High-leverage observations, which are data points with extreme values in the predictor variables, can introduce apparent multicollinearity. This happens because these outliers can disproportionately influence the regression coefficients, making it seem like there is a linear relationship between variables when, in fact, this relationship is driven by the outliers. When the outliers are removed, the multicollinearity may disappear, revealing that the variables are not actually collinear in the absence of these high-leverage points. However, multicollinearity may still exist even in outlier-free datasets. Interestingly, the search for methods to address multicollinearity eventually led to the discovery and use of penalty functions in machine learning, often for reasons unrelated to this problem.

There are several ways to determine if multicollinearity exists which we will not detail here. The more important point is that it is crucial for variables to be relatively independent of one another for accurate regression modeling. If the covariance matrix is diagonal, then the variables are completely independent. Any large off-diagonal terms are indicative of correlations between variables. In regression problems, the standard method to address multicollinearity is to somehow force the matrix to become invertible by

1 The term multicollinearity, which involves multiple variables, is used more often than collinearity even though it is a special case of collinearity.

changing the diagonal values of the covariance matrix, that is, make it diagonally dominant. But how can one do that without violating the original LS problem? The key is that we are mostly interested in the prediction error on test sets so we are permitted to do anything sensible to fine-tune the model parameters as long as the prediction error of the resulting model is reduced.

The most well-known modification of the LS formation is

$$\hat{\boldsymbol{\beta}}^{\text{LS-ridge}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \tag{5.2.2}$$

where $\boldsymbol{I}$ is the identity matrix and λ is a tuning hyperparameter that is adjusted to make the matrix invertible. In this new formulation, all we have done is add λ to the diagonals of $\boldsymbol{X}^\top \boldsymbol{X}$ such that the matrix becomes diagonally dominant and hence invertible for a sufficiently large value of λ. The problem of multicollinearity can be mitigated by virtue of this simple change in formulation. However, this adjustment will not produce the same set of LS estimates as in Eq. (5.2.1) and, as a result, it is given a different name, specifically *ridge regression*.

A natural question to ask is, what happens to the estimates as a function of λ? In fact, the **estimates will be shrinking as λ is increased** relative to the LS estimates. The use of ridge acts as a penalty on the original LS formulation since it reduces the estimates. The follow-up question then arises regarding the quality of the estimates if the new ridge formulation is used. Is it better or worse? The answer lies in the fact that **the variance is reduced and the bias is increased as λ is increased**. Then λ is chosen to balance the two such that the prediction error is minimized. Hence, an overall benefit can be achieved using ridge. The procedure used to find the best λ is called *regularization*. With these basic facts understood, it is appropriate to take a deeper dive into penalty estimation.

5.2.2 Penalized Loss Functions

In machine learning, the closed-form LS solution is rarely used because it is not suitable for large problems, the matrix may not be invertible and other penalty functions are difficult to incorporate. Instead, the standard approach is to apply gradient descent to some objective function and minimize it to obtain the model. A number of loss functions of the form $\sum_{i=1}^{n} \rho(\boldsymbol{\beta})$ were presented in Chapters 2 and 3. With that in mind, we need to create an objective function composed of a loss function and a penalty function. In this context, it is better to view penalty estimation as a constrained minimization problem as follows:

$$\underset{\beta \in \mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{n} \rho(\boldsymbol{\beta}) \qquad \text{subject to} \quad P(\boldsymbol{\beta}) \leq c. \tag{5.2.3}$$

That is, the original loss function is penalized by a function $P(\boldsymbol{\beta})$ which is constrained by c. The constant c limits the range of the possible solutions during optimization. To apply the constraint in the form of a single objective function, we can reformulate the problem as a Lagrange multiplier as follows:

$$J^{\text{loss-penalty}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \rho(\boldsymbol{\beta}) + \lambda P(\boldsymbol{\beta}) = \sum_{i=1}^{n} \rho(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{d} p(\beta_j). \tag{5.2.4}$$

In the above, the first term is the loss function and the second term is the penalty function. In linear regression, the loss function can be seen as a **"coarse-grain tuner"** because it broadly adjusts the model to fit the data. The penalty function acts as a **"fine-grain tuner"** by making more subtle adjustments to the model parameters. The hyperparameter λ plays the role of c in the original constrained minimization problem of Eq. (5.2.3). The solution to the regression problem of the form

$$\hat{\boldsymbol{\beta}}^{\text{loss-penalty}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^d}{\operatorname{argmin}} J^{\text{loss-penalty}}(\boldsymbol{\beta}) \tag{5.2.5}$$

is called a penalty estimator.

Penalized loss functions can be used to solve regression problems even in the absence of multicollinearity, and that is what is done in practice. The notion of multicollinearity is more of legacy justification and is rarely mentioned because of the enormous benefits obtained when using penalty functions. For example, they can be used to prevent overfitting or for variable selection to reduce model complexity and improve model interpretability. These are major advantages that are achieved through penalized estimators, making them critical to many data science problems.

5.3 Ridge Penalty

The most widely used form of penalty estimation is the one we have encountered already called ridge regression. The objective function using the least squares loss and ridge penalty is given by

$$J^{\text{LS-ridge}}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^\top\boldsymbol{\beta})^2 + \lambda\sum_{j=1}^{d}\beta_j^2. \tag{5.3.1}$$

Surprisingly, it is equivalent to Eq. (5.2.2) although it seems to bear little or no resemblance to it except for the use of λ. But they are indeed both penalized least squares. Because of its formulation seen above, the ridge penalty function is sometimes referred to as an L_2 penalty since it is based on the L_2-norm. Its primary purpose is to shrink the parameters thus moving the estimates away from the unbiased value and toward $\boldsymbol{\beta} = \mathbf{0}$. By properly tuning λ, it is possible to apply enough shrinkage to the parameters such that the resulting biased estimates provide a lower variance and a more accurate model for prediction. Note that $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_d)^\top$ but the intercept is not included in the penalty function. That is, β_0 is typically left out of the shrinkage process and is instead adjusted to accommodate the overall level of shrinkage of the other parameters.

To simplify the function, if we let $r_i = y_i - \boldsymbol{x}_i^\top\boldsymbol{\beta}$, the objective function becomes

$$J^{\text{LS-ridge}}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n}r_i^2 + \lambda\sum_{j=1}^{d}\beta_j^2. \tag{5.3.2}$$

The objective function using the log-cosh (LC) loss and ridge penalty is given by

$$J^{\text{LC-ridge}}(\boldsymbol{\beta}) = \frac{1}{a}\sum_{i=1}^{n}\log(\cosh(ar_i)) + \lambda\sum_{j=1}^{d}\beta_j^2. \tag{5.3.3}$$

For linear regression, $a = 2$ as discussed in Chapter 4.

For the Huber case, we use

$$J^{\text{H-ridge}}(\boldsymbol{\beta}) = \sum_{i=1}^{n}\rho_H(r_i) + \lambda\sum_{j=1}^{d}\beta_j^2, \tag{5.3.4}$$

where the loss term is

$$\rho_H(r_i) = \begin{cases} r_i^2/2 & \text{if } |r_i| \le \delta \\ \delta\left(|r_i| - \frac{\delta}{2}\right) & \text{if } |r_i| > \delta \end{cases}. \tag{5.3.5}$$

One issue to resolve when using Huber is that of specifying the value of δ. In many datasets with differing number of outliers, the value of δ cannot be determined *a priori*. If there are no outliers, a large value of δ is appropriate (between 4 and 5) whereas for cases with lots of outliers, a smaller value is warranted (between 0.1 and 0.5). However, as we saw in Chapter 3, choosing small values of δ may lead to optimization problems. Therefore, it is common practice to use a fixed value of $\delta = 1.345$ with penalty functions.

Exercise: In a linear regression problem, let $d = 2$. The two parameters of interest are β_1 and β_2. Write the ridge penalty function in terms of these two parameters with a constraint c applied.

Ans.: The expanded form of the ridge penalty with the constraint is

$$P(\boldsymbol{\beta}) = \beta_1^2 + \beta_2^2 \le c.$$

□

For a two-dimensional case, the constrained optimization problem of Eq. (5.2.3) can be written as

$$\underset{\boldsymbol{\beta}\in\mathbb{R}^3}{\text{minimize}} \sum_{i=1}^{n} \rho(\boldsymbol{\beta}) \qquad \text{subject to} \quad \beta_1^2 + \beta_2^2 \le c. \tag{5.3.6}$$

This implies that as c decreases (which is equivalent to increasing λ), the two parameters are constrained to live in a tighter and tighter *circle* around the origin. Therefore, the parameters will shrink in value as the constraint c is made smaller.

5.4 LASSO Penalty

The ridge penalty is based on the L_2-norm so it is natural to consider devising a penalty based on the L_1-norm. Such a penalty function is referred to as LASSO (least absolute shrinkage and selection operator). The objective function combines the least squares loss with the LASSO penalty and is given by

$$J^{\text{LS-LASSO}}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n} r_i^2 + \lambda \sum_{j=1}^{d} |\beta_j|. \tag{5.4.1}$$

An interesting and unique property of LASSO is that it can set some parameters to 0 effectively removing them from the regression procedure. This can be useful in many different data science problems, but particularly in high-dimensional problems with a large number of variables. In those situations, many of the variables do not affect the response and therefore can be set to 0. This process of removing unimportant variables is sometimes called *feature selection* in machine learning.

The objective function with the log-cosh loss and LASSO is given by

$$J^{\text{LC-LASSO}}(\boldsymbol{\beta}) = \frac{1}{a}\sum_{i=1}^{n} \log(\cosh(a r_i)) + \lambda \sum_{j=1}^{d} |\beta_j|. \tag{5.4.2}$$

Finally, in the Huber case, we use

$$J^{\text{H-LASSO}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \rho_H(r_i) + \lambda \sum_{j=1}^{d} |\beta_j|, \tag{5.4.3}$$

where $\rho_H(r_i)$ was defined earlier in Eq. (5.3.5).

Exercise: In a linear regression problem, let $d = 2$. The two parameters of interest are β_1 and β_2. Write the LASSO penalty function using these two parameters with a constraint c applied.

Ans.: The expanded form of the LASSO penalty is

$$P(\boldsymbol{\beta}) = |\beta_1| + |\beta_2| \le c.$$

□

For a two-dimensional problem, the constrained optimization problem of Eq. (5.2.3) can be written as

$$\underset{\boldsymbol{\beta} \in \mathbb{R}^3}{\text{minimize}} \sum_{i=1}^{n} \rho(\boldsymbol{\beta}) \qquad \text{subject to} \quad |\beta_1| + |\beta_2| \le c. \tag{5.4.4}$$

This implies that as c decreases (which is equivalent to λ increasing), the two parameters are constrained to live in a tighter and tigher *diamond* around the origin. Therefore, the parameters will shrink in value as the constraint c is made smaller.

5.5 Effect of Penalty Functions

In this section, the effect of penalty functions on the unpenalized estimates is described. Assume we have two parameters, β_0 and β_1, representing the intercept and slope of a line, respectively. The penalty would only be applied to β_1 since the intercept is not typically penalized. Then, the penalty function would act to shrink the slope as λ is increased. There are two basic ways to shrink a parameter value:

1) subtract a value until it reaches 0, or
2) divide by a number greater than 1.

For example, assume that we have produced an estimate of the slope, $\hat{\beta}_1^{\text{Unpenalized}}$, using simple linear regression, and we want to reduce its value with a penalty. In the first approach, we can simply let

$$\hat{\beta}_1^{\text{Penalized}} = \hat{\beta}_1^{\text{Unpenalized}} - \lambda$$

and increase λ until we achieve the desired reduction in value. If $\lambda = \hat{\beta}_1^{\text{Unpenalized}}$, then $\hat{\beta}_1^{\text{Penalized}} = 0$ and no further reduction is allowed (otherwise, it would go negative).

A second approach is to set

$$\hat{\beta}_1^{\text{Penalized}} = \frac{\hat{\beta}_1^{\text{Unpenalized}}}{(1 + \lambda)}$$

and increase λ until the needed reduction is obtained. However, in this case, $\hat{\beta}_1^{\text{Penalized}}$ does not reach 0, except at $\lambda = \infty$. In both cases, we would pick the λ that produces the lowest prediction error. As simple

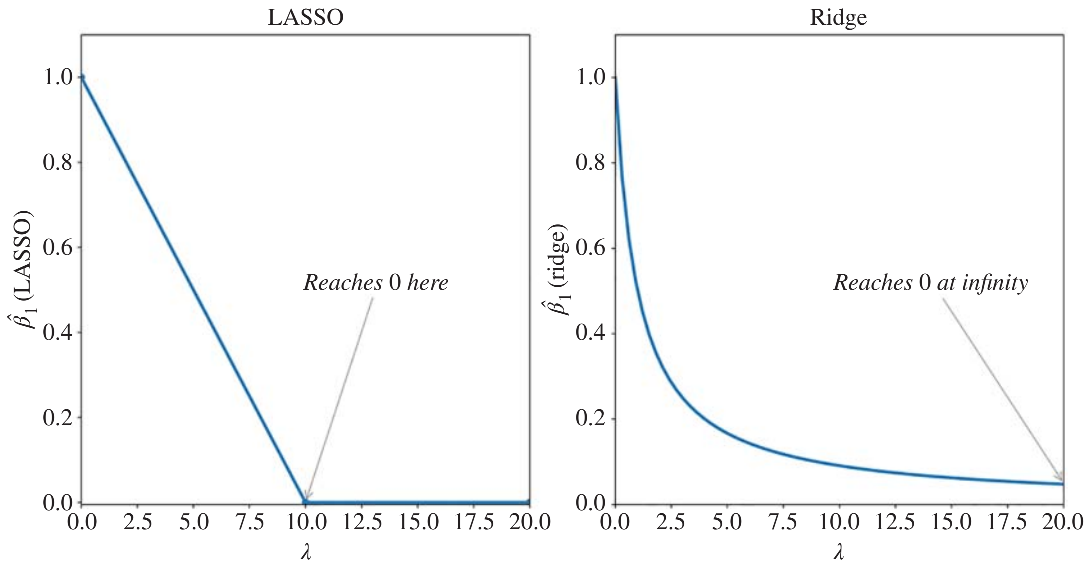


Figure 5.1 Key shrinkage characteristics of LASSO and ridge.

as these two cases may appear, they are the basis of the two most popular penalty estimators described above: LASSO and ridge.

The characteristics of these two important penalty functions are illustrated in Figure 5.1 for the one variable case. These types of plots are called *traces* because they trace the path of the estimates as λ increases. In the case of LASSO on the left, the estimate shrinks linearly as λ increases and eventually stops abruptly at 0. This is the result of the L_1 penalty. On the right side of the figure showing ridge, the estimate reduces in value gradually and reaches 0 at infinity. This is the result of the L_2 penalty. Since these plots are for a one-variable problem, they do not tell us about the behavior in a multidimensional setting. That will be presented in the sections to follow. However, it is worth keeping these first-order representations in mind when using the LASSO and ridge penalty functions: the ridge penalty reaches 0 gradually while the LASSO penalty reaches 0 abruptly as λ increases. In the figure, if $\lambda = 15$, LASSO would set $\hat{\beta}_1^{\text{Penalized}} = 0$ while ridge would set $\hat{\beta}_1^{\text{Penalized}} = 0.08$.

There is one other effect of penalty functions which has to do with the variance. When an estimate is reduced in size, the variance also decreases. That means that the uncertainty around the estimate is reduced so there is more confidence in the estimate. If we reduce all estimates to 0, the variance goes to 0, but the model is meaningless. The trade-off is that the bias increases (see Chapter 2). Estimates are initially unbiased but the bias steadily increases as λ increases. These two factors must be balanced which gives rise to the bias–variance trade-off, as described later in this chapter.

5.6 Penalty Functions with Outliers

We now address the two central questions in this chapter. First, can penalty functions mitigate the effects of outliers? And second, do penalty functions provide any benefits in terms of producing better models with robust loss functions? To illustrate the effect of outliers, we return to the Belgium telephone dataset from Chapter 2. It is a small dataset used throughout this book to illustrate some key features of linear regression with outliers. Here, we examine the ridge and LASSO penalties and show what happens to the slope of the regression line as the estimates shrink when λ is increased. Obviously, multicollinearity does not exist when there is only one variable, but the penalty function is still useful for reasons cited earlier.

Consider Figure 5.2. The two plots, one for ridge and one for LASSO, show the change in the slope as λ is adjusted using values 0, 50, 500, and 5000. We observe that the lines all rotate around a pivot point. For LS, the pivot point is the centroid. Both ridge and LASSO start at the unbiased estimate and then

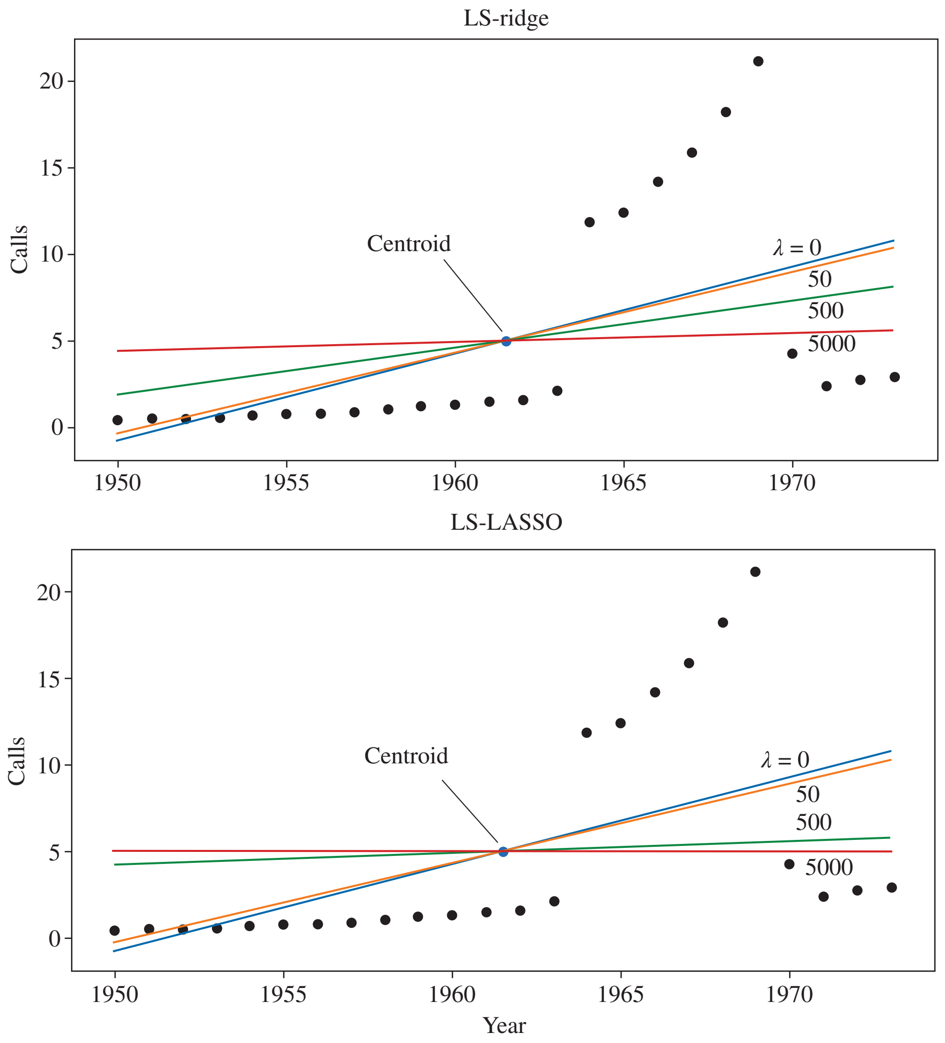


Figure 5.2 Ridge and LASSO effect on LS estimates.

flatten out to $\hat{\beta}_1^{\text{LS-ridge}} = 0$. If we examine the penalty function's ability to correct for outliers, we find that it cannot make the necessary adjustments because it is anchored to the centroid. Therefore, generally speaking, penalty functions will not help much when outliers are present. Nevertheless, the variance is reduced as the estimates shrink and the prediction error may be slightly reduced in the process.

Next, we address whether there are some gains to be had if a robust loss is used in conjunction with a penalty function. Consider Figure 5.3 where the log-cosh loss is combined with ridge and LASSO, respectively. Here, the pivot point is in the vicinity of the median of the data. There is some gain in choosing a nonzero value of λ as it may improve prediction accuracy. As we will see later, when there are many

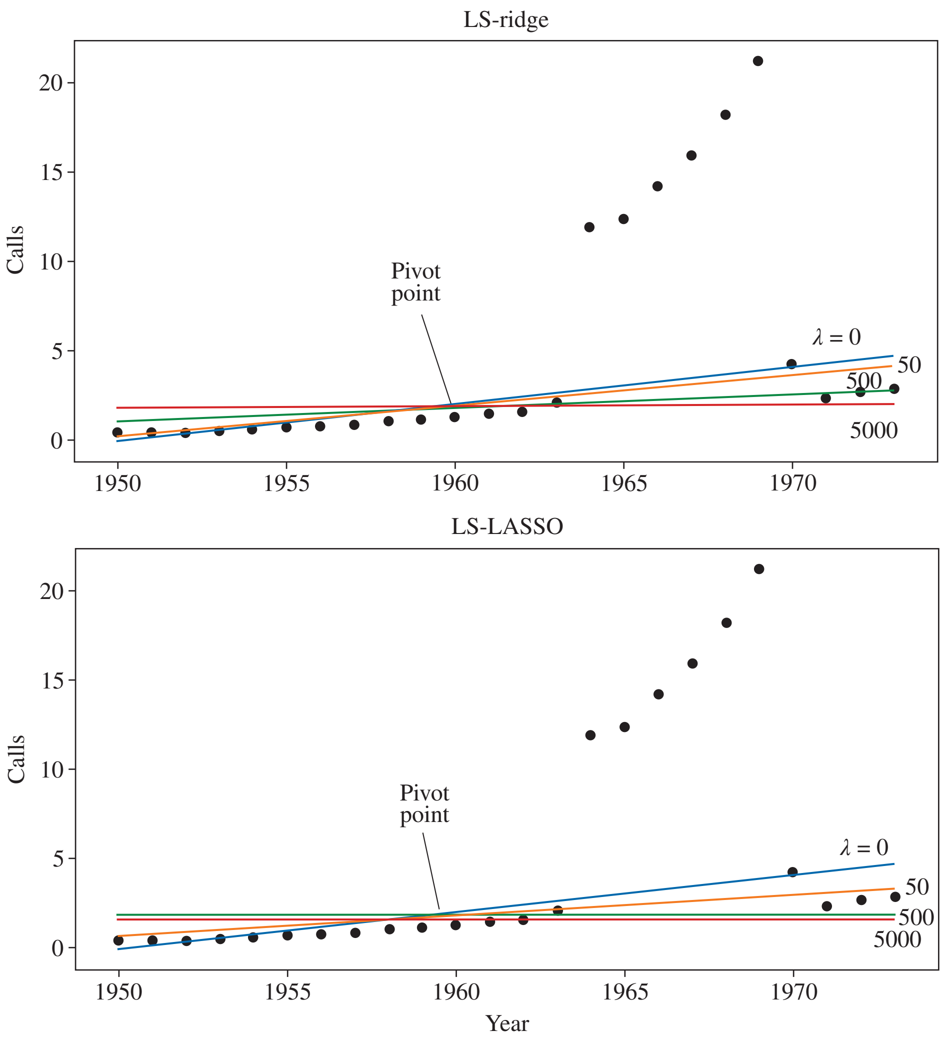


Figure 5.3 Ridge and LASSO effect on log-cosh estimates.

variables, ridge will improve the variance while LASSO allows for subset selection, but these two key advantages of penalty estimators are not observable with only one variable. The main point here is that penalties can improve the model and its variance in the robust case.

5.7 Ridge Traces

It is instructive to know what happens to each parameter as we vary λ in a multivariate setting. As we increase λ, the estimates tend to decrease but exactly how they decrease is not as easy to predict. However, in a previous section, we saw that the trajectory of one estimate as λ is increased is either a gradual decay to zero or an abrupt drop to zero. For the multivariate case, the estimates can be plotted as a function of λ to obtain the true trajectories which are called penalty traces.

In this and other chapters, we will be using these traces to illustrate many of the characteristics and properties of penalty estimators. We use ridge to introduce the traces and their meaning but the reader is encouraged to become very familiar with these traces as they will be used in the same way for other penalty functions. The process of creating the traces is very simple: choose a number of values of λ, compute the estimates at those values, and plot the results as a function of λ. Typically, the estimates are plotted as a function of $\log(\lambda)$ because it is more informative due to the large scales over which shrinkage occurs.

The Swiss dataset was introduced in Chapter 3. It is a linear regression problem with $d = 5$ explanatory variables (`Agriculture` (β_1), `Examination` (β_2), `Education` (β_3), `Catholic` (β_4), and `Infant Mortality` (β_5)) and one response variable (`Fertility`). The overall goal is to build a model to understand the relationship between the explanatory variables and the response variable. For our purpose, the objective is to understand the properties of penalty functions through the use of ridge traces. As described before, these traces are simply plots of the estimates for each variable as a function of λ.

To generate the ridge traces, we have to select many values of λ and compute the estimates for each one using Eq. (5.3.2). The traces for the Swiss dataset are plotted as a function of $\log(\lambda)$ in Figure 5.5. We see that all five variables have different trajectories in the figure. However, they all gradually arrive at 0. A key insight into the true nature of the traces is that the variance largely controls the rate of shrinkage. Estimates with large variances will shrink faster than those with small variances. This is exactly what we would like to happen. There is an important point worth mentioning here that may be clarifying in terms of the meaning of the traces: if we pick any vertical line associated with a specific λ value, the intersection of that line and the curves would give us the penalized estimates for that λ.

Exercise: Using Figure 5.4, find the approximate values of each of the parameters at $\lambda = 10^2$. (Hint: round to the nearest 0.05).

Ans.: If a vertical line is drawn on Figure 5.4, the approximate values of each parameter when $\lambda = 10^2$ are

$$\beta_1 = -0.20, \quad \beta_2 = -0.25, \quad \beta_3 = -0.85, \quad \beta_4 = 0.10, \quad \beta_5 = 0.80.$$

□

The ridge traces for LS-ridge, H-ridge, and LC-ridge are provided in Figure 5.5. They illustrate the similarities and differences between the different cases. In LC-ridge and H-ridge, `Examination` is clearly

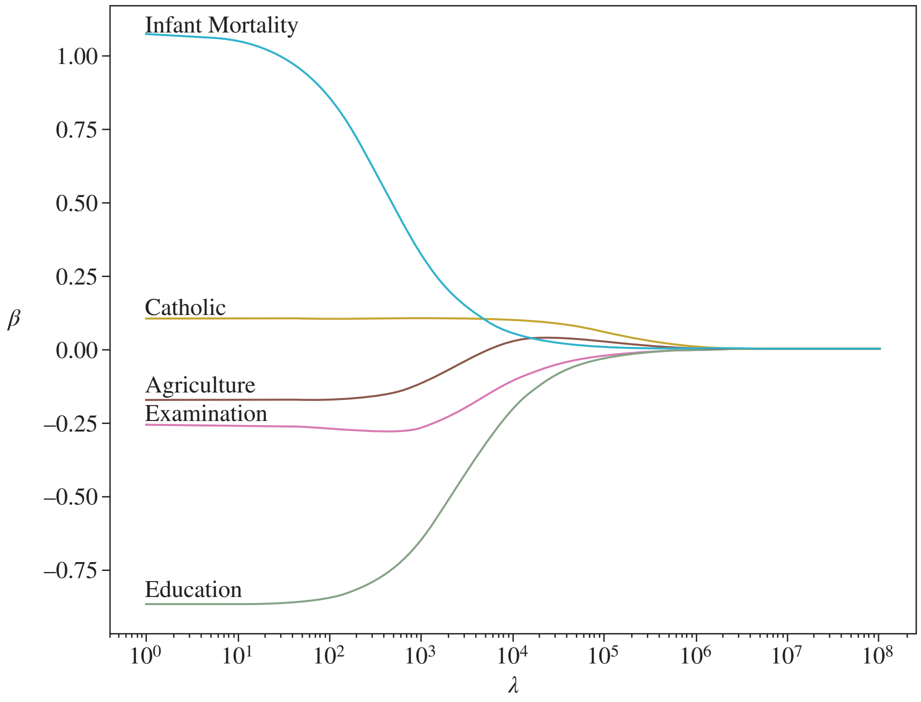


Figure 5.4 Ridge traces for the LS-ridge estimator.

non-monotonic but this is not as noticeable in the LS-ridge case. Recall that the Swiss dataset does not have outliers. In general, the traces of LS-ridge compared to LC-ridge and H-ridge will only differ slightly with no outliers, but the differences will be more pronounced when outliers are present. We should note that there is nothing magical about a particular λ value. This can be understood by examining the x-axis values of all three traces. Their ranges are all different because they are produced by three different loss functions. Therefore, LS-ridge, LC-ridge, and H-ridge may all produce different optimal λ values that lead to essentially the same set of estimates if no outliers exist.

5.8 Elastic Net (Enet) Penalty

A penalty function that combines the ridge and LASSO penalties into a single composite penalty function referred to as the *Elastic Net* as follows:

$$P(\boldsymbol{\beta}) = \lambda_1 \sum_{j=1}^{d} |\beta_j| + \lambda_2 \sum_{j=1}^{d} \beta_j^2. \tag{5.8.1}$$

There are now two tuning hyperparameters, λ_1 and λ_2. If $\lambda_1 = 0$, we obtain the ridge penalty, whereas if $\lambda_2 = 0$, we obtain the LASSO penalty. The two hyperparameters, λ_1 and λ_2, are adjusted until the desired bias/variance trade-off is achieved (i.e. balancing overfitting and underfitting).

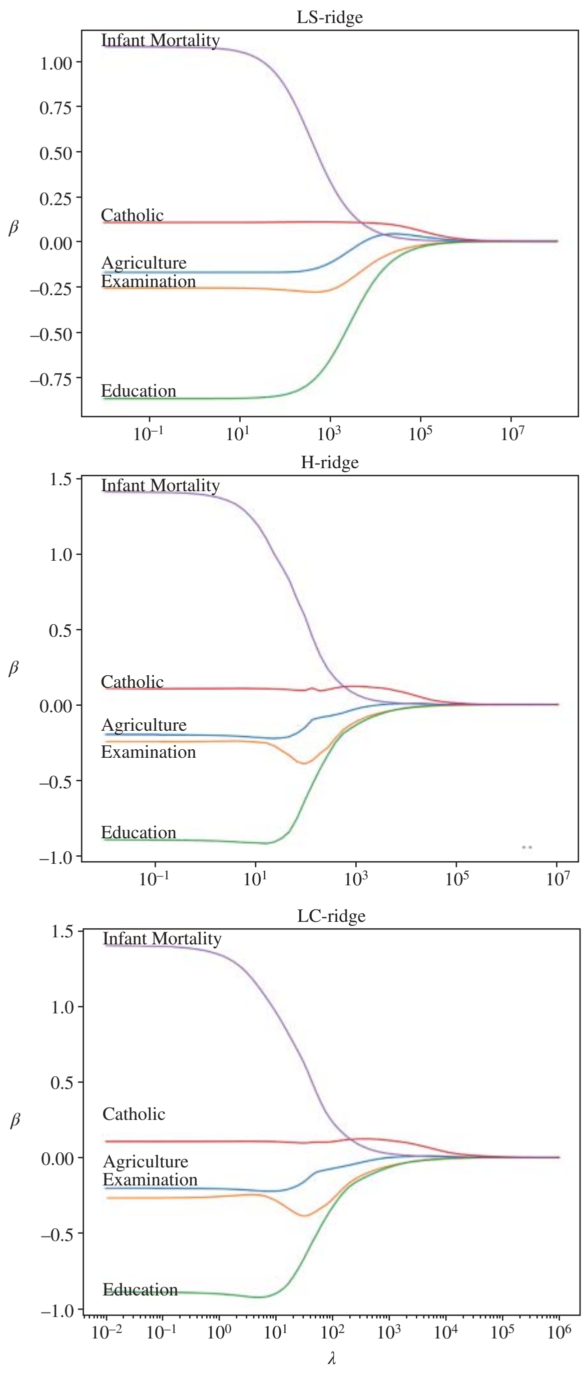


Figure 5.5 Traces for LS-ridge, H-ridge, and LC-ridge estimators.

In machine learning, it is formulated by letting $\lambda_1 = \lambda\alpha$ and $\lambda_2 = \lambda(1-\alpha)/2$ to produce

$$P(\boldsymbol{\beta}) = \lambda \left[\alpha \sum_{j=1}^{d} |\beta_j| + \frac{(1-\alpha)}{2} \sum_{j=1}^{d} \beta_j^2 \right]. \tag{5.8.2}$$

If $\alpha = 0$, we are using ridge while if $\alpha = 1$, we are using LASSO. In practice, we must seek optimal values of both λ and α by selecting a number of discrete values of each one and then trying all possible combinations of these values to find the best model—a procedure referred to as a *grid search*.

Exercise: Write the objective function for a log-cosh loss with an elastic net penalty function.

Ans.: The full expression is

$$J^{\text{LC-Enet}}(\boldsymbol{\beta}) = \frac{1}{a} \sum_{i=1}^{n} \log(\cosh(ar_i)) + \lambda \left[\alpha \sum_{j=1}^{d} |\beta_j| + \frac{(1-\alpha)}{2} \sum_{j=1}^{d} \beta_j^2 \right].$$

□

5.9 Adaptive LASSO (aLASSO) Penalty

Model reduction using LASSO is very important when the number of parameters is large. In a dataset with many variables, most of them will not be important in terms of their impact on the response variable. One can imagine a dataset with $d = 50$ being reduced to $d_0 = 30$ by variable selection, and ideally only the important explanatory variables would be retained in the compact model. A penalty function with an inherent ability to order the variables from most important to least important is said to have the *oracle property*.

Unfortunately, the LASSO penalty does not have this property. It is effective at removing variables but not necessarily in their order of importance. As a consequence, it is possible that the wrong variables may inadvertently be removed by setting them to 0. For example, let $d = 7$ and assume we know *a priori* that $\beta_4 = \beta_6 = 0$ (hence they should be removed) while all other parameters are nonzero. If LASSO is used, it might remove two variables but not necessarily β_4 and β_6. Thus, such a penalty estimator does not have the oracle property. However, it may still produce a compact model but the parameter values may not be as meaningful if the wrong variables have been removed. This is important because of model interpretability. We want to be able to eliminate parameters from the model that are not explanatory in terms of the response variable, leaving only those that are important. If an estimator possesses the oracle property, it behaves asymptotically (i.e. with enough data) as if it has prior knowledge of the true model. Whenever model reduction and interpretability are important, we require the oracle property to be present in the penalty function.

Improved methods have been developed that do indeed possess the oracle property. One in particular is relevant to the discussion here: aLASSO. In the original LASSO, the same λ value is used for all parameters. In aLASSO, an additional weight is applied to each parameter as follows:

$$P(\boldsymbol{\beta}) = \lambda \sum_{j=1}^{d} w_j |\beta_j|, \tag{5.9.1}$$

where w_j is the weight for each parameter. The only matter to resolve is to choose w_j.

If we set $w_j = 1/|\hat{\beta}_j^*|$, where $\hat{\beta}_j^*$ is a known quantity typically set to the unpenalized estimate for β_j, we obtain the new penalty function

$$P(\boldsymbol{\beta}) = \lambda \sum_{j=1}^{d} \frac{|\beta_j|}{|\hat{\beta}_j^*|} = \sum_{j=1}^{d} \lambda_j |\beta_j|, \tag{5.9.2}$$

where $\lambda_j = \lambda/|\hat{\beta}_j^*|$. Hence, each parameter has its own effective λ_j based on the unpenalized estimates. Then, small $\hat{\beta}_j^*$ values will lead to large weights which is equivalent to a high λ_j, while large $\hat{\beta}_j^*$ values would be equivalent to a small λ_j. This makes sense intuitively since the unpenalized solution has valuable information about the magnitude of the estimates, which in turn can be used as a weighting factor to guide the aLASSO estimation in the right direction for the removal of unimportant variables. Although somewhat more complicated than ridge or LASSO, the aLASSO approach provides a powerful mechanism for model reduction.

The objective function for a least squares loss with an aLASSO penalty function, assuming that the unpenalized estimates are known and given by $\hat{\beta}_j^{\text{LS}}$ for $j = 1, \ldots, d$, is

$$J^{\text{LS-aLASSO}}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n} r_i^2 + \lambda \sum_{j=1}^{d} \frac{|\beta_j|}{|\hat{\beta}_j^{\text{LS}}|}.$$

The objective function for a log-cosh loss with an aLASSO penalty function, assuming that the unpenalized estimates are known and given by $\hat{\beta}_j^{\text{LC}}$ for $j = 1, \ldots, d$, is

$$J^{\text{LC-aLASSO}}(\boldsymbol{\beta}) = \frac{1}{a}\sum_{i=1}^{n} \log(\cosh(ar_i)) + \lambda \sum_{j=1}^{d} \frac{|\beta_j|}{|\hat{\beta}_j^{\text{LC}}|}.$$

Finally, in the Huber case, assuming that the unpenalized estimates are known and given by $\hat{\beta}_j^{\text{H}}$ for $j = 1, \ldots, d$, we use

$$J^{\text{H-aLASSO}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \rho_H(r_i) + \lambda \sum_{j=1}^{d} \frac{|\beta_j|}{|\hat{\beta}_j^{\text{H}}|},$$

where $\rho_H(r_i)$ was defined earlier in Eq. (5.3.5).

5.10 Penalty Effects on Variance and Bias

In the next chapter, we will carry out a mathematical treatment and provide practical applications of the bias–variance trade-off in machine learning. In this section, we provide a more qualitative view of each quantity in preparation for that discussion.

5.10.1 Effect on Variance

We first study the effect of a penalty function on parameter variance. It was mentioned earlier that the variance is reduced as λ increases. This effect will now be illustrated through a series of histograms.

It should be well understood at this stage that the variance is a measure of the uncertainty in the estimates but closed-form expressions of variance may be hard to derive. An alternative is to generate a large number of estimates of each parameter using many datasets, but unfortunately we only have one to work with. However, we can use a well-known approach called the *bootstrap method* to generate the distribution of the estimates in histogram form using only one dataset.

The basic idea in bootstrapping is to sample a given dataset multiple times and compute estimates for each sample. That is, we can create what are called sampling distributions of the estimates. Ideally, we would like to do this with many different datasets but since we only have one to work with, we use a *sample with replacement* strategy. This means we will draw an observation from the original dataset randomly and then put that observation back into the original dataset and repeat the process until we have constructed the new dataset in full.

More concretely, in the Swiss dataset, there are 47 observations (i.e. rows). When we apply bootstrapping, we randomly select 47 observations (with replacement) to create a new dataset, which implies that we may have repeated observations in our synthesized dataset. We then compute the estimates using some method (LS in this case) and repeat this procedure multiple times to generate parameter distributions. The spread of the estimates around the mean captures the variance and this can be computed easily. Figure 5.6 shows the distribution of LS estimates ($\lambda = 0$) after sampling the dataset in this manner. We note that β_2, β_3, and β_5 have large variances. In addition, β_0 has a large variance but we will not be applying the ridge penalty to it; only $\beta_1, \ldots, \beta_5$ will be penalized because they are explanatory variables directly associated with the response.

Now what happens to the variance when we shrink the estimates by increasing λ when using a ridge penalty? Consider only β_5 using LS-ridge for a number of different values of λ as shown in Figure 5.7. With $\lambda = 0$, we obtain the original distribution given in Figure 5.6 which features a large variance. However, as λ is increased, the variance is reduced, and the estimates move toward 0. This is exactly what the theory predicts. Also implied here is the fact that the bias is increasing as we increase λ since the mean

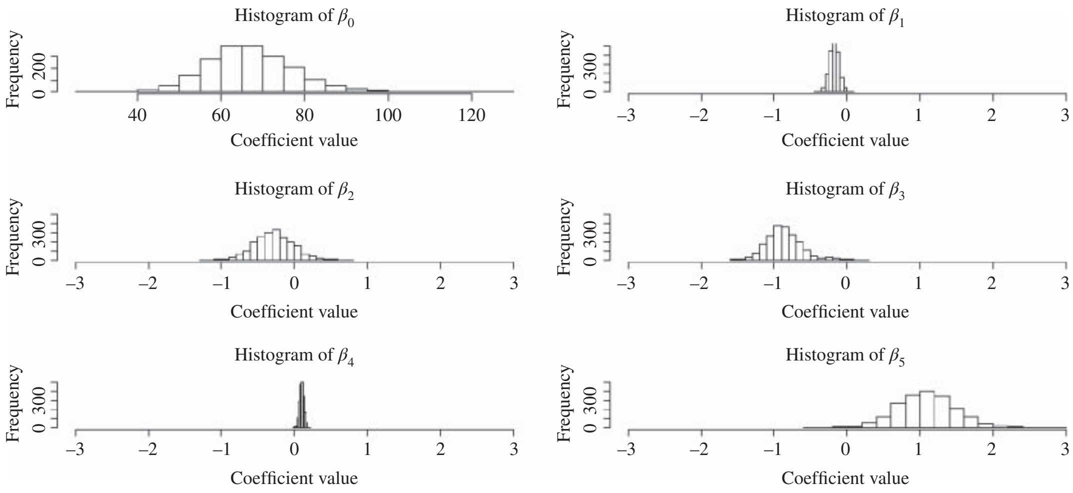


Figure 5.6 Sampling distributions of LS estimates.

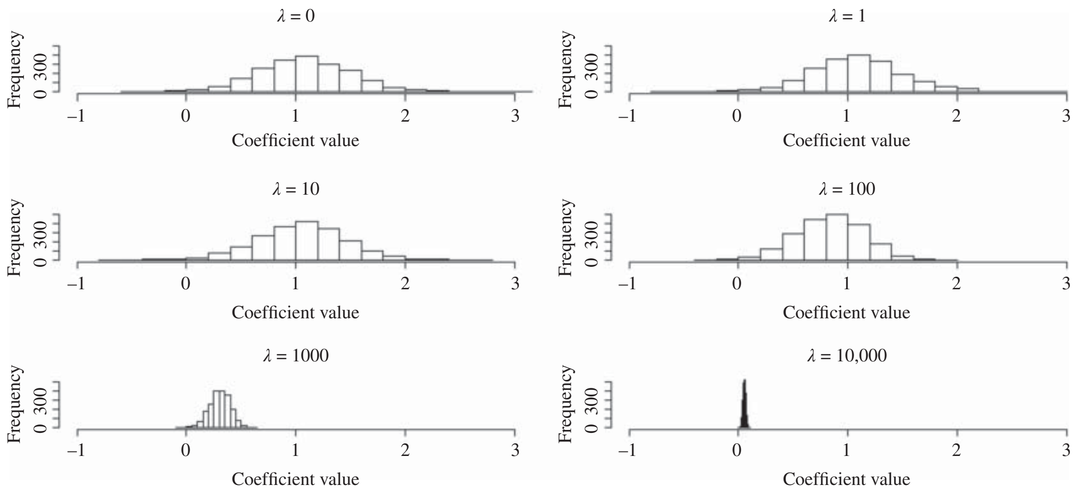


Figure 5.7 Sampling distributions of LS estimates of β_5 for different λ values.

is moving away from its original unbiased value (i.e. the mean of the distribution with $\lambda = 0$). So the variance is decreasing while the bias is increasing. But when do we stop? The answer is to choose the λ that gives us the lowest prediction error; ideally, this occurs when the bias and variance are balanced. Again, we will explore this trade-off more fully in the next chapter to answer this question properly.

5.10.2 Geometric Interpretation of Bias

In order to understand the bias of a penalty estimator, we now consider geometric interpretations of the three penalty functions, ridge, LASSO, and aLASSO, using two variables. These interpretations also provide insight into the inner workings of each penalty estimator but are only meant as a qualitative view of their differences. They are based on a multiple linear model given by

$$\boldsymbol{y} = \beta_0 + \beta_1\boldsymbol{x}_1 + \beta_1\boldsymbol{x}_2 + \boldsymbol{\epsilon}, \tag{5.10.1}$$

where least squares estimation is used for linear regression. The concepts to be described apply to robust regression in the same manner.

We begin with the ridge penalty. One can show that the L_2 penalty function effectively places a circular constraint on the two parameters, β_1 and β_2, given by

$$\beta_1^2 + \beta_2^2 \leq c, \tag{5.10.2}$$

where c is a constant. These circular regions centered at the origin are shown in Figure 5.8. The figure represents the two-dimensional space of possible β_1 and β_2 values. The circles represent the constraint $\beta_1^2 + \beta_2^2 = c$ due to the ridge penalty function for different values of c, which corresponds to λ in the penalty function. The point marked "×" represents the *unbiased* LS estimates. Imagine that the unbiased solution sits at the bottom of a valley with contours around it. The elliptical contours represent lines of equal value of the original loss function while the circles represent the constraints due to the penalty function.

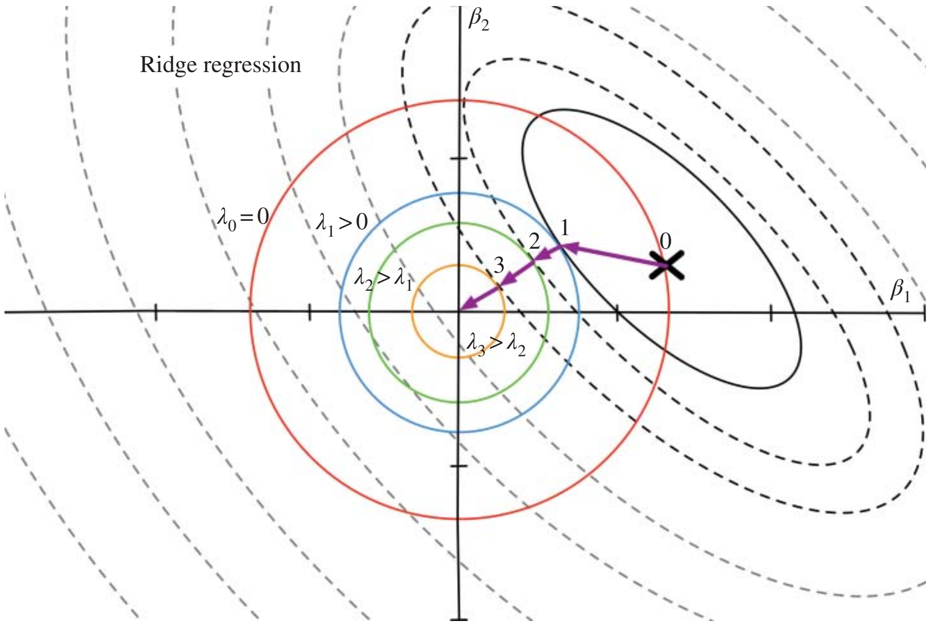


Figure 5.8 Geometric interpretation of ridge.

Starting at the unbiased solution on the circle labeled 0, with $\lambda_0 = 0$, when we begin to apply the penalty function, the estimates shrink and follow some hypothetical trajectory depicted in the figure until the edge of circle 1 is reached at $\lambda = \lambda_1$. As λ increases to $\lambda = \lambda_2$ (circle 2), we shrink the estimates by a greater degree. At $\lambda = \lambda_3$, we reach circle 3. This continues until all estimates reach 0 at $\lambda = \infty$ at the origin. As we do this, the bias increases. For example, the bias at point 2 is the vector distance from point 0 to point 2. Essentially, we are shrinking the estimates toward the origin where the bias is the largest and the variance is the smallest. In practice, we seek some intermediate point between the unbiased estimate at "×," where we have overfitting, and the origin, where we have underfitting.[2] We can try a number of different values of λ until a proper balance is struck between the two. The process of finding the optimal value, λ_{opt}, to balance this *bias–variance trade-off* is called regularization.

Next, we examine the LASSO penalty in the same manner. The constraint imposed by the L_1 penalty function is

$$|\beta_1| + |\beta_2| \leq c. \tag{5.10.3}$$

The resulting constraint geometry is a diamond shape in two dimensions rather than the circle, as shown in Figure 5.9. If $\lambda_0 = 0$, we obtain the unbiased estimates at "×." As λ is increased, the diamond shrinks in size, as do the resulting parameters. Eventually, when $\lambda = \infty$, the origin is reached. However, in the case of LASSO, one of the two parameters may reach 0 well before the other.

2 The definitions of overfitting and underfitting will be elaborated in the next chapter. Briefly, overfitting refers to the tendency to try to fit every point in the dataset as closely as possible so that the model loses its generality, whereas underfitting is inadvertently not fitting enough points to create a usable model.

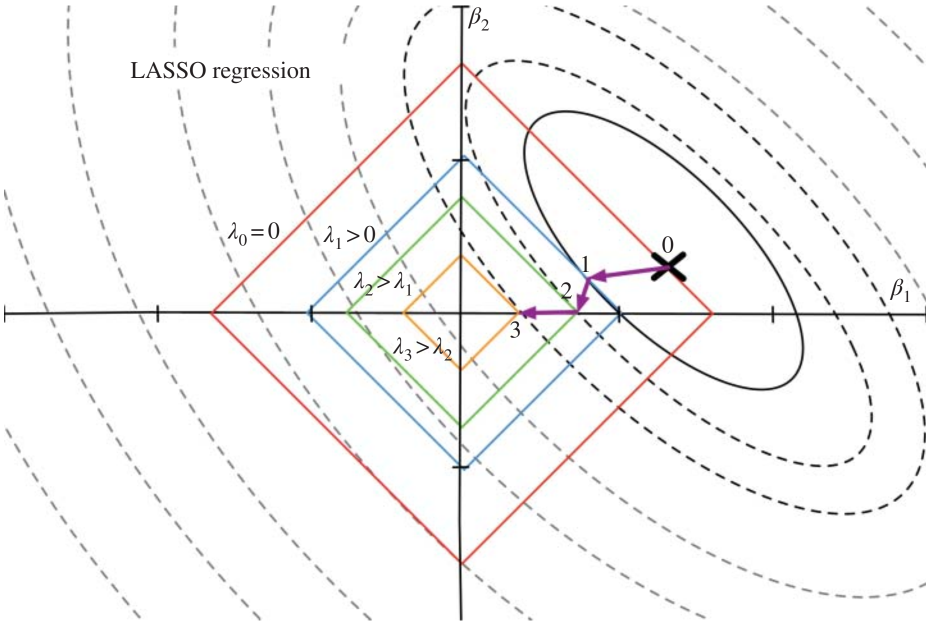


Figure 5.9 Geometric interpretation of LASSO.

The hypothetical trajectory for LASSO starts at the unbiased LS estimates ("×") at point 0 and terminates at point 1 with $\lambda = \lambda_1$. Both estimates shrink but neither one is 0 yet. However, when $\lambda = \lambda_2$, the path lands on one corner of diamond 2, where $\beta_2 = 0$. Then, we are left with a model where only $\beta_1 \neq 0$. We can now remove β_2 and keep only β_1. This is the way LASSO performs feature selection. That is, it has selected β_1 over β_2 because it is considered more important (with the caveat that LASSO does not possess the oracle property). Further increases in λ eventually forces $\beta_1 = 0$ at the origin.

Finally, consider the aLASSO penalty in the same manner. Recall that, in aLASSO, we introduced a weight w_j to the LASSO penalty. The effect of individual weights on each parameter can be seen in the geometric interpretation of Figure 5.10. Weights effectively reshape the diamond from a symmetrical one in LASSO (Figure 5.9) to an asymmetrical one in aLASSO (Figure 5.10). The constraint for aLASSO is given by

$$\frac{|\beta_1|}{|\hat{\beta}_1^{\text{LS}}|} + \frac{|\beta_2|}{|\hat{\beta}_2^{\text{LS}}|} \leq c. \tag{5.10.4}$$

The asymmetry is due to the division by the unpenalized estimates. In the case of LASSO in Figure 5.9, the trajectory starting from "×" could reach either $\beta_1 = 0$ or $\beta_2 = 0$ since the diamond is symmetric. However, aLASSO would likely reach the $\beta_2 = 0$ corner first, which is the nearest corner, assuming the geometry in Figure 5.10. Then, the asymmetric diamond would continue to shrink in size as indicated by λ_1 and λ_2, etc., and eventually reach $\beta_1 = 0$ at or before $\lambda = \infty$.

In this case, both LS-LASSO and LS-aLASSO reached the same conclusion that β_1 is more important than β_2 but this was a simple two-variable problem. In larger problems, they tend to find different

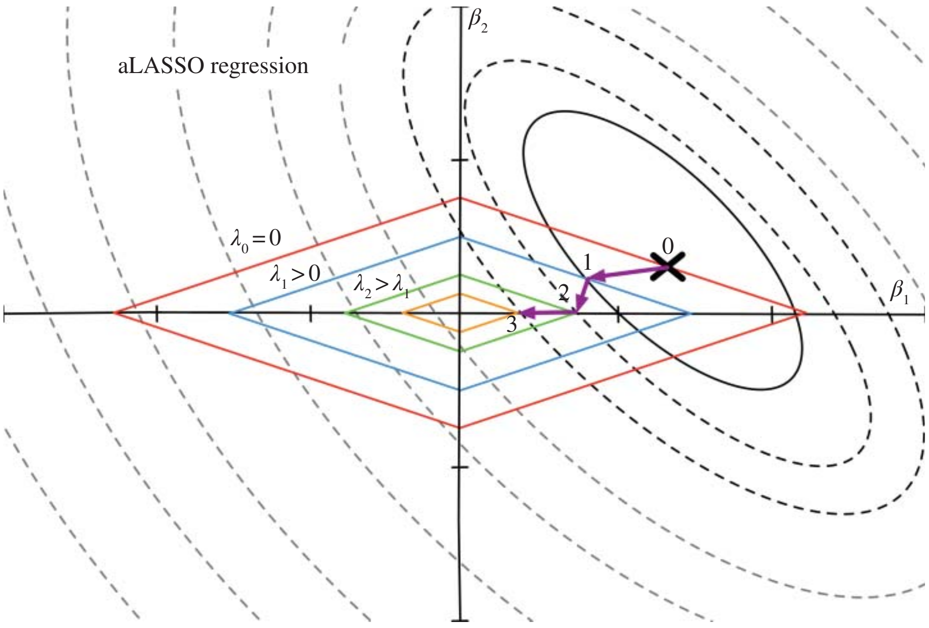


Figure 5.10 Diamond-warping effect of weights in aLASSO estimator.

orderings of the variables. Because of the oracle property, the aLASSO penalty captures an oracle trajectory of each parameter whereas LASSO, in general, does not. We will explore this further in the next chapter.

5.11 Variable Importance

One of the key objectives in data science is to determine the importance of different explanatory variables relative to the response variable. There are many tools and techniques used to obtain the variable importance, two of which will be described in this section. For linear regression, we could look at the size of the estimates to gain some insight into its importance. However, this requires that the dataset be standardized and does not take the variance into account. Instead, it is common practice to use the *t*-statistic to obtain a first-order assessment of the relative importance of different variables. Another approach is to use LASSO and aLASSO traces to provide an ordering of the variables from most important to least important. These two penalty functions perform better if combined with robust methods, especially if there are outliers in the dataset. In this section, we compare and contrast these different approaches.

5.11.1 The *t*-Statistic

The *t*-statistic is associated with the *t*-distribution which is a more general version of the Gaussian distribution in that it can handle small sample sizes of less than 30 observations. Formally, the *t*-statistic for a given estimate, $\hat{\theta}$, with standard error, s.e.($\hat{\theta}$) is obtained using the formula

$$t_{\hat{\theta}} = \frac{\hat{\theta} - 0}{\text{s.e.}(\hat{\theta})} = \frac{\hat{\theta}}{\text{s.e.}(\hat{\theta})}. \tag{5.11.1}$$

The t-statistic is used in hypothesis testing to determine the statistical significance of an estimated value of a parameter. We want to know if the estimate should be trusted. In this case, the hypothesis test assumes the target value is 0. The t-statistic can be used to obtain a p-value to quantify the significance of the estimate. For example, in Figure 5.11, a t-distribution is shown for some estimate. The t-statistic computed as above turns out to be, say $t_{\hat{\theta}} = -2.3$. The p-value is the area under the curve between $-\infty$ and -2.3 plus the area between 2.3 and ∞ (the two shaded areas). For this case, the p-value is 0.02 which means that the estimate is significant at the $\alpha = 0.05$ level (95% confidence), but not at the $\alpha = 0.01$ level (99% confidence). The maximum p-value is 1 and the minimum is 0. Usually we want to have very small p-values for all of our parameters so we are confident about the model.

The p-value is one of the most frequently cited quantities in statistical analysis. For our purposes, the magnitude of the t-statistic is of greater interest rather than the p-value itself since it can be informative about the relative importance of each variable. The further away the t-statistic is from 0 in either direction, the greater is the parameter importance. Therefore, if we consider the absolute value of the t-statistic and order the variables accordingly, we obtain a good idea of the variable importance relative to one another.

As mentioned in Chapter 4, if there are outliers in the dataset, the values of both $\hat{\theta}$ and s.e.($\hat{\theta}$) can be affected. The computed distribution may be quite different from the true distribution, as shown in Figure 4.8. Therefore, it is important to note that the t-test assumes an outlier-free dataset that conforms to the true Gaussian distribution. Otherwise, the results may be misleading. With this caveat in mind, we continue with the description of the t-statistic for variable importance.

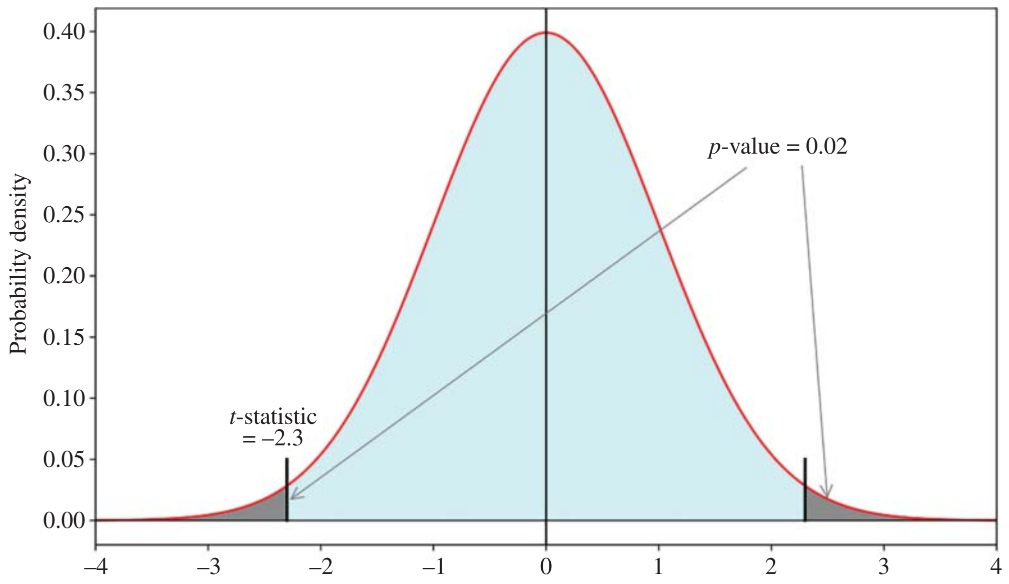


Figure 5.11 Using the t-statistic to find the p-value.

Table 5.1 Swiss fertility t-statistics for least squares (LS) and log-cosh (LC).

Feature	$\frac{\hat{\beta}^{LS}}{\text{s.e.}(\hat{\beta}^{LS})}$	Var. order	$\frac{\hat{\beta}^{LC}}{\text{s.e.}(\hat{\beta}^{LC})}$	Var. order
Agriculture	−2.43	4	−2.96	4
Examination	−1.03	5	−1.08	5
Education	−4.75	1	−4.94	1
Catholic	2.88	2	3.01	3
Infant Mortality	2.83	3	3.72	2

Consider the analysis of the t-statistics for the Swiss dataset shown in Table 5.1. This dataset does not have outliers so the t-test can be applied. The t-statistic and variable ordering are provided in the table using LS and LC linear regression. From these results, it appears that `Education` is the most important factor to determine `Fertility`, and `Examination` and `Agriculture` are the least important. The parameters `Catholic` and `Infant Mortality` are of medium importance. Both LS and LC produce roughly the same ordering since the Swiss data does not have outliers. This is a simple and effective method to get a sense of the relative importance of the parameters, so long as there are no outliers.

5.11.2 LASSO and aLASSO Traces

An interesting aspect of both LASSO and aLASSO is their feature selection capability based on their traces. However, one is better than the other as we will demonstrate by examining their traces. To begin, consider the traces for LS-LASSO, H-LASSO, and LC-LASSO provided on the left side of Figure 5.12 for the Swiss dataset. They illustrate the fact that, unlike the ridge traces, these traces terminate abruptly on the λ-axis at some finite values. They are all similar in nature due to the fact that there are no outliers. Again, the shape of LS-LASSO compared to LC-LASSO and H-LASSO will only differ slightly with no outliers, but it will be more pronounced when outliers are present.

Perhaps more importantly, the order in which they reach an estimate of 0 has some significance. It can be interpreted as a variable ordering from least important to most important as λ increases. Note that all variables eventually have estimates of 0 (the zero point) when λ is large enough. There is valuable information in the order in which they reach the zero point. It would be of great value if the order was related to its relative importance. Then, the first variable to reach 0 would be deemed the least important, while the one that reached 0 last would be considered the most important. But it has been found that the LASSO penalty does not offer this desirable feature because it lacks the oracle properly. However, the aLASSO penalty possesses this property, given enough data, and the traces can provide a visualization of variable importance.

Consider the right-hand side of Figure 5.12 showing the traces for LS-aLASSO, H-aLASSO, and LC-aLASSO. They look similar to the LASSO traces with one notable difference: the order in which the variables reach 0 has changed. This is a subtle but important difference. Table 5.2 provides a summary of the orderings for the Swiss dataset using LS-LASSO, LC-aLASSO, and the t-statistic. Both LC-aLASSO and the t-statistic have the same order of the variables, but not LS-LASSO. From the consensus of

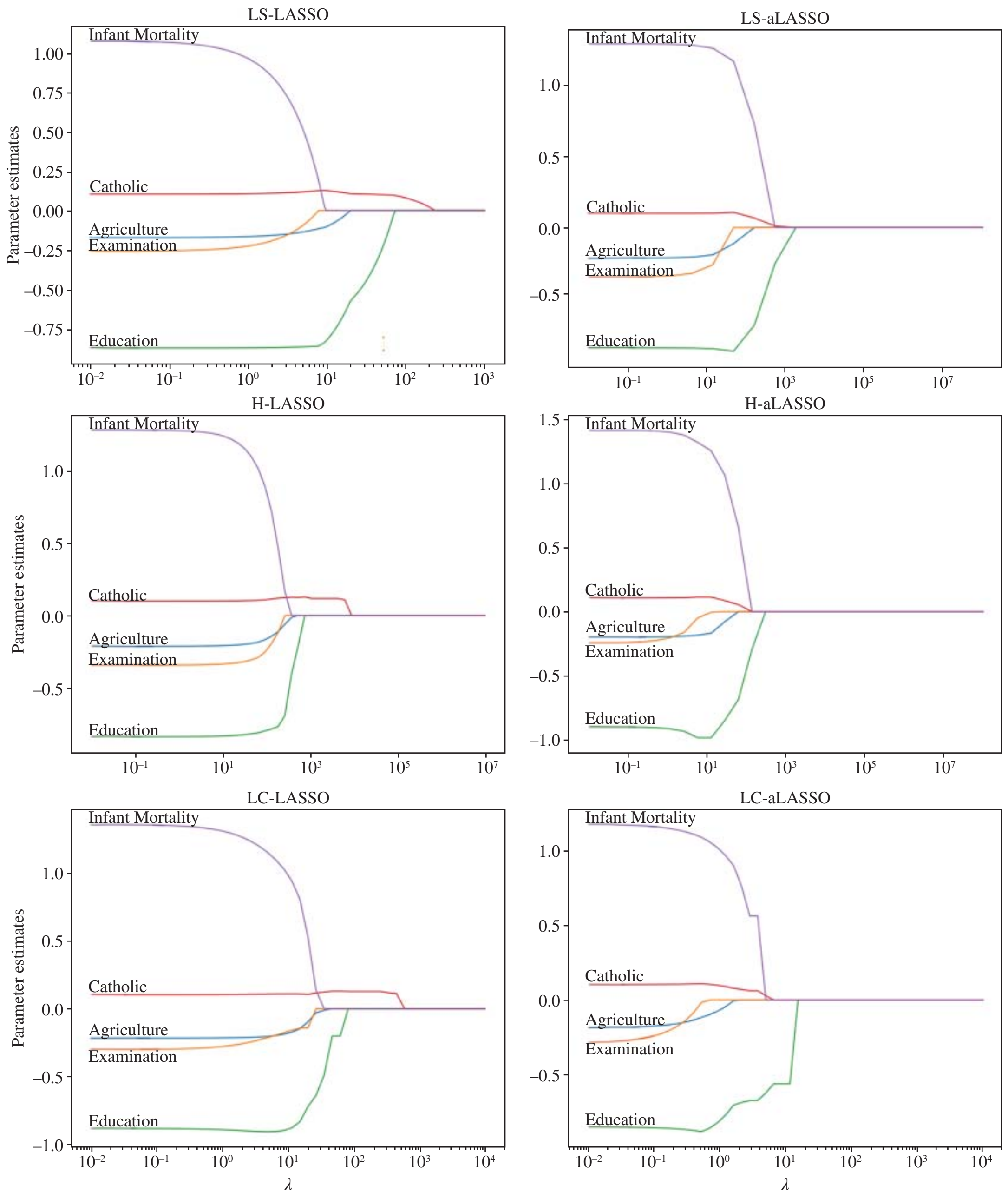


Figure 5.12 Trace characteristics of LASSO and aLASSO.

Table 5.2 Variable importance of the Swiss fertility dataset.

Feature	LS-LASSO	LC-aLASSO	t-Statistic
Agriculture (β_1)	3	4	4
Examination (β_2)	5	5	5
Education (β_3)	2	1	1
Catholic (β_4)	1	2	2
Infant Mortality (β_5)	4	3	3

LC-aLASSO and the t-statistic, `Education` is the most important. Then, `Catholic` and `Infant Mortality` are next, then `Agriculture`, and then finally `Examination` (for which all three agree). This ordering is also logically sound. While the t-statistic is correct for this example, it may not be correct in general as it is only considering $\lambda = 0$. aLASSO is considering the entire range of λ values when determining variable ordering and is much more reliable.

To summarize, LASSO does not have the oracle property, whereas aLASSO does possess this property assuming there is enough data. Further, the t-statistic provides first-order information about variable importance but it should be used with a robust loss function. If there are outliers, the LS estimates may be inaccurate and the standard errors may be inflated. Therefore, the ordering may be incorrect. The most reliable option is a robust loss function with aLASSO to produce the correct ordering.

5.12 Summary

In this chapter, we have presented properties and practical aspects of penalized robust estimators and its application to machine learning. Several new ideas and concepts have been introduced to enable robust linear regression and model building that deserve further investigation. These topics will be covered in the next chapter. We have demonstrated the importance of penalty estimators in reducing variance and identifying variable importance. The ridge penalty is a smooth shrinkage mechanism, whereas aLASSO has the oracle property and is best-suited for variable ordering. One can view the LASSO as being somewhere in between. The LASSO penalty typically sets some parameters to zero, but not necessarily in the proper order since it does not have the oracle property. There are some numerical challenges in the implementation of LASSO and aLASSO due to the L_1 nature of their respective penalty functions, especially for large λ.

The motivation for the use of robust penalty estimators in machine learning is due to the presence of outliers in datasets. The LS method is not outlier-tolerant so penalty estimation with an LS loss is equally problematic. This is why robust methods should supersede the least squares approach where appropriate. In the next chapter, methods of regularization with penalty estimators and k-fold cross-validation techniques will be described which will complete the picture on the value of these methods.

Problems

5.1 Consider the following dataset with $d = 2$ to study collinearity.

$\boldsymbol{x}_1$	132.2	501.5	904.0	227.6	66.6	43.4	1253.0
$\boldsymbol{x}_2$	11.7	17.9	21.1	14.7	7.7	8.4	32.8
$\boldsymbol{y}$	404.6	1180.6	1807.5	470.0	151.4	93.8	3293.4

Assume the linear regression model to be

$$\boldsymbol{y} = \beta_0 + \beta_1\boldsymbol{x}_1 + \beta_2\boldsymbol{x}_2 + \boldsymbol{\varepsilon}.$$

Use Python code to solve the following.

a) Plot $\boldsymbol{x}_2$ vs. $\boldsymbol{x}_1$. Qualitatively assess whether there is a potential of collinearity, that is, a linear relationship between the two variables. (Hint: if there is a strong correlation between the two variables, there is likely to be collinearity in the dataset.)

b) Center and standardize the data by subtracting the mean and dividing by the standard deviation for each variable.

c) Find the least squares estimates for β_1 and β_2 using centered data as follows:

$$\hat{\boldsymbol{\beta}}^{\text{LS}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}.$$

d) Find the LS-ridge estimates and plot the ridge traces using the closed-form expression given here (I_p is a 2×2 identity matrix)

$$\hat{\boldsymbol{\beta}}^{\text{LS-ridge}} = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I}_p)^{-1}\boldsymbol{X}^\top\boldsymbol{y}.$$

Use $\lambda = 0, 1, 2, 5, 10, 20, 50, 100, 200, 500,$ and 1000. What are the values of β_1 and β_2 for $\lambda = 50$? How should we decide which λ is best? (Hint: you will need to read Chapter 6 to answer this question.)

5.2 (PROJECT) This problem uses Python libraries to obtain results for both least squares with ridge and least squares with LASSO. In both cases, the traces will be plotted and compared.

a) Load the needed libraries.

```
import numpy as np
import math
import scipy
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from scipy.optimize import minimize
import matplotlib.pyplot as plt
```

b) Read in the data file (which you should already have in your directory using the data provided in Chapter 3).

```
swiss = pd.read_csv("swiss.csv")
swiss['ones'] = 1
X = np.array(swiss.drop(["Fertility"],axis=1))
Y = np.array(swiss["Fertility"])
x = X - np.mean(X, axis=0)
y = Y - np.mean(Y)
colnames = swiss.columns
```

c) Run ridge regression from $\lambda = 10^0$ to 10^7.

```
n_lambdas = 200
lambdas = np.logspace(0, 7, n_lambdas)
coefs = []
d = X.shape[1]
for a in lambdas:
    ridge = Ridge(alpha=a)
    ridge.fit(X, Y)
    coefs.append(ridge.coef_[0:d-1])
```

d) Plot the ridge traces.

```
plt.figure(figsize=(10, 8))
ax = plt.gca()
ax.plot(lambdas, coefs)
ax.set_xscale('log')
plt.xlabel('lambda')
plt.ylabel('beta 1-5')
plt.title('LS-ridge coefficients as a function ' \
    'of lambda')
beta1 = coefs[0]
pos = np.zeros(len(beta1))
for i in range(len(beta1)):
    plt.annotate(xy=[lambdas[0],beta1[i]+pos[i]], \
        text=colnames[i+1])
plt.axis('tight')
plt.show()
```

e) Replace the ridge code segment above with LASSO:

```
n_lambdas = 200
lambdas = np.logspace(-2, 3, n_lambdas)
coefs = []
d = X.shape[1]
for a in lambdas:
        lasso = Lasso(alpha=a)  #LASSO
        lasso.fit(x,y)
        coefs.append(lasso.coef_[0:d-1])
```

Plot the results again using the plotting code above with a change in the title to reflect the use of LS-LASSO. Compare and contrast the results from LS-ridge and LS-LASSO. What value of λ reduces the number of variables to 2 for LS-LASSO?

5.3 (PROJECT) This project involves generating the traces for log-cosh with ridge and log-cosh with LASSO.

a) Define the log-cosh function and all the routines needed for LC-ridge and LC-LASSO.

```
logcosh = lambda x: np.logaddexp(x, -x) - np.log(2)
def residual(b,x,y):
    residual_y = y-np.matmul(x,b)
    return residual_y
def LCRidge(b):
    d = len(b)
    diff = residual(b,X,Y)
    #RIDGE
    loss = np.sum(logcosh(diff))+ \
        lam*np.sum(b[0:d-1]*b[0:d-1])
    return loss
def LCLASSO(b):
    d = len(b)
    diff = residual(b,X,Y)
    #LASSO
    loss = np.sum(logcosh(diff))+ \
        lam*np.sum(np.abs(b[0:d-1]))
    return loss
```

b) Run LC-ridge with the following code.

```
n_lambdas = 25
beta = np.random.randn(d)/10.
lambdas = np.logspace(-3, 5, n_lambdas)
coefs = []
for lam in lambdas:
    res = minimize(LCRidge, beta, method='BFGS')
    beta = res.x
    coefs.append(beta[0:d-1])
```

Plot the results using the code in the previous problem with a change in the title to reflect the use of LC-ridge.

c) Run LC-LASSO with the following code.

```
n_lambdas = 25
beta = np.random.randn(d)/10.
lambdas = np.logspace(-3, 5, n_lambdas)
coefs = []
for lam in lambdas:
    res = minimize(LCLASSO, beta, method='BFGS')
    beta = res.x
    coefs.append(beta[0:d-1])
```

Plot the results using the code in the previous problem with a change in the title to reflect the use of LC-LASSO. What value of λ reduces the number of variables to 2 for LC-LASSO?

References

James, G., Witten, D., Hastie, T. et al. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer.

Saleh, A.K. Md. E., Arashi, M., and Kibria, B.M.G. (2019). *Theory of Ridge Regression Estimation with Applications*. New York: John Wiley and Sons.

Saleh, A.K. Md. E., Arashi, M., Saleh, R., and Norouzirad, M. (2022). *Rank-Based Shrinkage and Selection: With Application to Machine Learning*. New York: John Wiley and Sons.

6

Robust Regularized Models

6.1 Introduction

When building a model using machine learning, there is a concerted effort to balance the possibilities of overfitting and underfitting on a given dataset. This issue has its roots in the bias–variance trade-off. In this chapter, we explore the details of this trade-off and then examine the effects of outliers on the standard procedures used in model building to generalize the results. This chapter is a companion to the previous chapter and covers a number of practical regularization techniques that can be used to build better models. One of the key questions to be answered is regarding the selection of the λ hyperparameter associated with the penalty functions. The overall goal is to produce a compact, robust model that generalizes to unseen test datasets when outliers are present. This involves feature selection, penalty functions, and/or the application of k-fold cross-validation, all of which will be described.

The chapter begins with illustrations of overfitting and underfitting in linear regression problems. Then, an in-depth study of the bias–variance trade-off is provided as it pertains to choosing the optimal λ. This is followed by the effect of outliers in datasets when they are divided into training and test sets. Next, penalty estimation with various cross-validation schemes will be described as methods of generalization. We also address the question of whether generalization procedures can mitigate the effects of outliers. The application of penalty functions to the task of feature selection is also described. In this case, the objective is to select the best subset of the available features. A methodology is described that combines robust penalty estimation with feature selection to find a compact model with good overall prediction accuracy.

6.2 Overfitting and Underfitting

An illustration of overfitting and underfitting in linear regression is provided in Figure 6.1. A simple way to understand overfitting is "too many parameters" for the given data. For underfitting, it can be viewed as "too few parameters" for the given data. For example, in the left panel of Figure 6.1, the data fitting is carried out using linear regression with a polynomial expansion of variable x. In particular, if we expand the parameter space using $x, x^2, \dots, x^9$, then a solution can be produced that follows the data quite closely leading to a small error. The regression model for this case is

$$\boldsymbol{y} = \beta_0 + \beta_1\boldsymbol{x} + \beta_2\boldsymbol{x}^2 + \cdots + \beta_9\boldsymbol{x}^9 + \boldsymbol{\varepsilon}. \tag{6.2.1}$$

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.

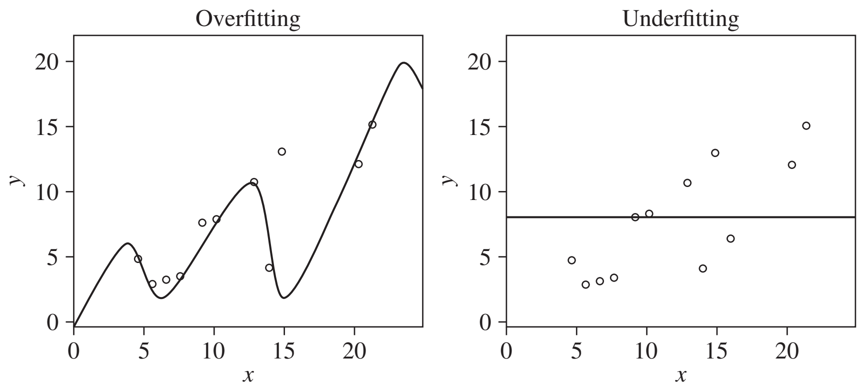


Figure 6.1 Overfitting and underfitting in linear regression.

There are a total of 10 parameters including the intercept (too many for this data) which allows the regression model to capture every nuance of the data but does not lead to a general model for prediction purposes. Hence, having a large number of parameters can lead to overfitting.

At the other extreme, a model can be built with just one parameter (not enough parameters) using

$$\boldsymbol{y} = \beta_0 + \boldsymbol{\varepsilon}. \tag{6.2.2}$$

The results are shown on the right-hand panel of Figure 6.1. This model is also unsatisfactory because it does not capture the general trend of the data and it incurs large errors. Hence, having fewer parameters than necessary can lead to underfitting.

If we were to blindly carry out the two options, we would likely select the overfitted case because the errors are small. But there is a better solution between the overfitted and underfitted cases that is more general. Here, a simple linear model would suffice. We could try several models in between and choose the best one but the problem is that if we use the errors as a guide, we would always pick the overfitted case. Another approach is needed to address the problem of selecting the proper **model complexity**, and it is based on the bias–variance trade-off.

Note: The next section provides details of the bias–variance trade-off for those who want to understand it at a fundamental level. It contains mathematical analysis to clarify and reinforce the concepts. It is possible to skip this section without any loss of continuity, although it is highly recommended as it is an important concept in machine learning.

6.3 The Bias–Variance Trade-Off

Since the error of a model between the true value and predicted value, $y - \hat{y}$, cannot help us to decide which model is better, a different metric is needed. For example, the error between the estimated parameter and the true parameter, $\hat{\theta} - \theta$, is a viable option. This quantity can be positive or negative for different datasets so we could simply square it to obtain $(\hat{\theta} - \theta)^2$ and define a metric called the **quadratic risk** (QR) (see Saleh et al. 2022) which is given by[1]

$$\text{QR} = \mathbb{E}[(\hat{\theta} - \theta)^2]. \tag{6.3.1}$$

The idea behind this metric is to monitor the expected value of the squared difference between the estimated value $\hat{\theta}$ and the true value θ. We can use this quantity to compare various estimators. In practice, the expected value is replaced with the average of the squared error.

It is difficult to use the equation in this form because we do not know θ, but there are two fundamental quantities that make up the QR: the bias and variance. These two quantities tend to go in opposite directions to one another and this becomes the basis for the trade-off in the QR. We can obtain the bias–variance relationship by starting with the equation for the variance of a random variable X (see Chapter 2),

$$\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2, \tag{6.3.2}$$

and rearranging it to obtain

$$\mathbb{E}[X^2] = \text{Var}(X) + \mathbb{E}[X]^2. \tag{6.3.3}$$

Then letting $X = \hat{\theta} - \theta$, we obtain

$$\mathbb{E}[(\hat{\theta} - \theta)^2] = \text{Var}(\hat{\theta} - \theta) + (\mathbb{E}[\hat{\theta} - \theta])^2 = \text{Var}(\hat{\theta}) + (\mathbb{E}[\hat{\theta}] - \theta)^2. \tag{6.3.4}$$

The first term is obviously the variance of $\hat{\theta}$. The second term is simply the bias squared. Recall from Chapter 2 that the bias is given by

$$\text{bias} = \mathbb{E}[\hat{\theta}] - \theta. \tag{6.3.5}$$

This is a measure of how close the expected value of our estimator is to the true value. Ideally, the bias should be 0 in which case the estimator is said to be **unbiased**.

More concretely, if we have 100 datasets and compute 100 estimates of θ using those datasets, we can measure *how close the average of those estimates is to the true value of θ*. Now imagine an infinite number of datasets and carrying out the same measurements. Surely, the average will be equal to the true value. Not necessarily. It is only true if the estimator is unbiased. Perhaps an equivalent way to think about it would be "how close is the average estimate to the true value if we had an infinite number of datasets?."

With this understanding of the QR, we can now write that

$$\text{QR}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \theta)^2] = \text{Var}(\hat{\theta}) + \text{bias}^2(\hat{\theta}). \tag{6.3.6}$$

This equation lies at the heart of the bias–variance trade-off, a key part of many machine learning procedures.

Exercise: An estimator is used on $N = 5$ datasets and each dataset produces a different estimate as follows: 0.25, 0.27, 0.29, 0.23, and 0.24. The true value is 0.25. Calculate the bias. (Hint: replace the expectation with an average.)

Ans.: $\mathbb{E}[\hat{\theta}] \approx \frac{1}{5}\sum_{i=1}^{5} \hat{\theta}_i = (0.25 + 0.27 + 0.29 + 0.23 + 0.24)/5 = 0.256$. Therefore, $\mathbb{E}[\hat{\theta}] - \theta \approx 0.256 - 0.25 = 0.006$. □

1 The quadratic risk is often referred to as the mean-squared error but we will continue to use quadratic risk to avoid confusion with the MSE metric of Chapter 4.

Exercise: An estimator is used on $N = 5$ datasets and each dataset produces a different estimate as follows: 0.25, 0.27, 0.29, 0.23, and 0.24. Estimate the variance.

Ans.: The mean is $\overline{\theta} = 0.256$. The unbiased estimate of variance uses $N - 1$ instead of N as the divisor. Then, $\widehat{\text{Var}(\hat{\theta})} = \frac{1}{4}\sum_{i=1}^{5}(\hat{\theta}_i - \overline{\theta})^2 = ((0.25 - 0.256)^2 + (0.27 - 0.256)^2 + (0.29 - 0.256)^2 + (0.23 - 0.256)^2 + (0.24 - 0.256)^2)/4 = 0.00058$. □

We generally seek estimators with the lowest QR and Eq. (6.3.4) tells us there will be a different bias–variance trade-off in each estimator. Ideally, we would like the bias to be 0 and the variance to be 0 but the equation does not permit this because if we reduce the variance, the bias will increase, and vice versa. The best we can do is strike a balance between the two quantities.

Example: The bias–variance trade-off was derived using a shortcut earlier in this section. The proper derivation proceeds as follows. Starting with the definition of the quadratic risk, we use the linearity of expectations to produce the desired form of the equation:

$$\begin{aligned}
\text{QR} &= \mathbb{E}[(\hat{\theta} - \theta)^2] \\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}] + \mathbb{E}[\hat{\theta}] - \theta)^2] \\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2 + 2(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\mathbb{E}[\hat{\theta}] - \theta) + (\mathbb{E}[\hat{\theta}] - \theta)^2] \\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2] + 2\mathbb{E}\boldsymbol{[}(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\mathbb{E}[\hat{\theta}] - \theta)\boldsymbol{]} + \mathbb{E}[(\mathbb{E}[\hat{\theta}] - \theta)^2] \\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2] + 2\mathbb{E}\boldsymbol{[}(\hat{\theta} - \mathbb{E}[\hat{\theta}])\boldsymbol{]}(\mathbb{E}[\hat{\theta}] - \theta) + \mathbb{E}[(\mathbb{E}[\hat{\theta}] - \theta)^2] \\
&= \text{var}(\hat{\theta}) + 0 + \text{bias}^2(\hat{\theta}) \\
&= \text{var}(\hat{\theta}) + \text{bias}^2(\hat{\theta}).
\end{aligned}$$

In the fifth line of the above derivation, the first term is the definition of the variance, the last term is the bias squared, and the middle term is 0. Since $(\mathbb{E}[\hat{\theta}] - \theta)$ is a number, it can be moved outside the expectation of the middle term, and then we have that $\mathbb{E}[\hat{\theta} - \mathbb{E}[\hat{\theta}])] = \mathbb{E}[\hat{\theta}] - \mathbb{E}[\mathbb{E}[\hat{\theta}]] = \mathbb{E}[\hat{\theta}] - \mathbb{E}[\hat{\theta}] = 0$. Note that the QR equation is true in the limit, but not necessarily at finite values of N. □

6.4 Regularization with Ridge

The term *regularization of a model* refers to the adjustment of model parameters to obtain the lowest quadratic risk using the hyperparameter λ associated with a penalty function. First we develop a simple strategy in a well-defined context with closed-form expressions for the bias and variance. Then, based on this experience, we can formalize the procedure in a real-world setting for machine learning.

Our starting point is the least squares estimator,

$$\hat{\boldsymbol{\beta}}^{\text{LS}} = (\boldsymbol{X}^{\top}\boldsymbol{X})^{-1}\boldsymbol{X}^{\top}\boldsymbol{y}, \tag{6.4.1}$$

where $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{y} \in \mathbb{R}^n$. This is the best linear unbiased estimator as discussed in Chapter 2. However, we want to add flexibility to see if a better model can be found. One way to do this is to apply a penalty function, namely, ridge (see Saleh et al. 2019). The LS-ridge estimator from Chapter 5 had the following closed-form expression:

$$\hat{\boldsymbol{\beta}}^{\text{LS-ridge}}(\lambda) = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_d)^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \tag{6.4.2}$$

where $\boldsymbol{I}_d$ is a $d \times d$ identity matrix and λ is the ridge hyperparameter. We can build a number of different models just by changing λ and then choose the model associated with the optimal λ. This is how we intend to regularize the model.

To understand the regularization procedure, we start by reviewing the aforementioned bias–variance trade-off at a more basic level. We know the meaning of variance—it is the degree of uncertainty around an estimate. In a multidimensional setting, there will be one variance for each parameter. We need to construct a covariance matrix (see Chapter 3, Section 3.6) and then add up the terms along the diagonal to obtain the sum of the variances. This is called the *trace of the covariance matrix*. Notationally, the trace of a matrix $\boldsymbol{C}$ is represented as $\text{tr}(\boldsymbol{C})$.

For the bias, a working definition in a multidimensional case is the L_2 distance between the expected value of a given estimator and the true value, $||\mathbb{E}(\hat{\boldsymbol{\beta}}) - \boldsymbol{\beta}||_2$. An estimator is said to be unbiased if $\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$. Although one would think it is desirable to have an unbiased estimator, there are no guarantees that the variances will be low for unbiased estimators. In fact, we are more interested in models with a smaller variance at the cost of a slightly higher bias by choosing the appropriate λ.

Exercise: Let $\mathbb{E}[\hat{\boldsymbol{\beta}}] = (-3.2, 0.1, 1.1)^\top$ and $\boldsymbol{\beta} = (-3, 0, 1)^\top$. Compute the bias. Then, given $\sigma^2 = 0.25$ for the covariance matrix,

$$\sigma^2(\boldsymbol{X}^\top \boldsymbol{X})^{-1} = \sigma^2 \begin{pmatrix} 3 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix},$$

compute the trace of the covariance matrix. Finally, compute the quadratic risk. What can you say about the relative sizes of the bias and variance terms?

Ans.: First, $||\text{bias}(\hat{\boldsymbol{\beta}})||_2^2 = ||\mathbb{E}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta}||_2^2 = (-0.2)^2 + 0.1^2 + 0.1^2 = 0.06$. Next, the variance is $\text{Var}(\hat{\boldsymbol{\beta}}) = \text{tr}(\sigma^2(\boldsymbol{X}^\top \boldsymbol{X})^{-1}) = 3(0.25) + 2(0.25) + 2(0.25) = 1.75$. Therefore, $\text{QR} = 0.06 + 1.75 = 1.81$. Note that the variance is high and the bias is low in this case. □

6.4.1 Selection of Hyperparameter λ

To select the optimal value of λ, there are two possible options. One option is to use the mean-squared error (MSE) as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{6.4.3}$$

and choose the λ that minimizes this metric. There is one major problem with this approach: the optimal value is $\lambda = 0$ because the MSE is minimized using LS. It can only grow larger as λ is increased.

The other option is to use the quadratic risk which is the sum of the variance and the bias squared. In short, the proper value of λ should be based on balancing overfitting and underfitting to build our model. Note that an optimal λ value is unique to each estimator and generally cannot be compared or transferred between estimators. Additionally, the most suitable λ usually lies in a range of values in practice, as opposed to being one unique value, so it is not critical that an ideal value be found. A near-optimal value is sufficient in machine learning.

By applying a penalty function, we can increase the bias and decrease the variance until the estimates minimize the QR statistic. But how do we know this is happening as we change λ? This was already illustrated in Chapter 5 in a qualitative way. Here, we provide quantitative evidence of these effects. For LS-ridge, there are closed-form expressions available (see Montgomery et al. 2012) that allow us to gain insight into what is happening.

The QR of estimates in a multidimensional setting is given by

$$\mathrm{QR}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}}) = \mathrm{tr}(\mathrm{Var}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}})) + \|\mathrm{bias}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}})\|_2^2 \tag{6.4.4}$$

where $\mathrm{tr}(\mathrm{Var}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}}))$ is the trace of the covariance matrix and involves adding up its diagonal terms. For Eq. (6.4.2), the formulas for bias and variance for LS-ridge are given by[2]

$$\|\mathrm{bias}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}})\|_2^2 = \lambda^2\, \boldsymbol{\beta}^\top (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_d)^{-2} \boldsymbol{\beta}, \tag{6.4.5}$$

which is a scalar quantity, and

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}}) = \sigma^2 (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_d)^{-1} (\boldsymbol{X}^\top \boldsymbol{X}) (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_d)^{-1}, \tag{6.4.6}$$

which is a matrix quantity (for which the trace is taken to produce a scalar).

The above expressions look rather unwieldy and are too complicated to provide any understanding of the bias–variance trade-off. To simplify the analysis and develop intuition, let $\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{I}_d$. This yields

$$\|\mathrm{bias}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}})\|_2^2 = \frac{\lambda^2}{(1+\lambda)^2} \boldsymbol{\beta}^\top \boldsymbol{\beta}, \tag{6.4.7}$$

and

$$\mathrm{tr}(\mathrm{Var}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}})) = \frac{d\sigma^2}{(1+\lambda)^2}. \tag{6.4.8}$$

Hence, the quadratic risk is given by

$$\mathrm{QR}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}}) = \frac{d\sigma^2}{(1+\lambda)^2} + \frac{\lambda^2}{(1+\lambda)^2} \boldsymbol{\beta}^\top \boldsymbol{\beta}. \tag{6.4.9}$$

We see that if $\lambda = 0$, then the bias term is zero and the variance term is given by $d\,\sigma^2$. As we increase λ, the bias increases while the variance decreases. In the limit, as $\lambda \to \infty$, the variance approaches zero while the bias converges to $\boldsymbol{\beta}^\top \boldsymbol{\beta}$, provided this quantity is bounded. This situation is also undesirable because the bias is now too large. The trade-off between the bias and variance is clearly evident in this equation. This search for the best λ is what we call regularization in the context of penalty estimators.

2 For full derivation of variance and bias, see Montgomery et al. (2012).

6.4.2 Example: Diabetes Dataset

We now consider using the closed-form expressions of Eqs. (6.4.4)–(6.4.6) on the diabetes[3] dataset to obtain the optimal λ. It has $n = 442$ observations and $d = 10$ explanatory variables. The reason for the selection of this dataset is that the estimates have large variances so that the benefits of penalty estimation are more pronounced. We proceed by using LS-ridge and progressively picking larger and larger values of λ to find a better set of estimates. For the moment, other details of the dataset are not pertinent to the discussion here, and we will use the full dataset in the regularization process to find out if we can do better than the standard LS estimates.

The results are shown in Figure 6.2. If we use the MSE metric of Eq. (6.4.3) to find the optimal λ, we obtain the results shown in the upper portion of Figure 6.2. Unfortunately, the MSE increases as λ increases. Therefore, $\lambda = 0$ is the optimal value if MSE is used as the metric. This makes sense since LS is designed to find the minimum MSE but does not offer a way to improve the model.

Our goal is now clear: use the QR metric instead of MSE to find the optimal value λ that balances the bias and variance. We have plotted the results for the bias squared, variance, and QR in the lower portion of Figure 6.2. We see that the bias is initially zero when $\lambda = 0$ and the variance is large. This is the overfitted situation.

Next, consider what happens as λ is increased. Initially the variance alone determines the value of the QR. As we increase λ, the variance drops while the bias increases exactly as we would expect.[4] Eventually, the bias alone determines the QR. This is the underfitted situation. Between the two extremes, the QR reaches a minimum at $\lambda_{\text{opt}} = 0.0025$ for this dataset. It may appear that the QR has not been reduced by much but remember that the variances have all been reduced significantly which implies a more general model with less uncertainty.

The LS estimates ($\lambda = 0$) and the resulting LS-ridge estimates ($\lambda = 0.0025$) are provided in Table 6.1 along with their corresponding variances. Notice that the estimates and variances are all smaller for LS-ridge compared to LS for the variables that started with large variances (`s1, s2, s3, s4`, and `s5`). However, the remaining variables (`age, sex, bmi, bp`, and `s6`) did not change much in both their estimates and their variances. That is because they already had good estimates for the general model.

It is also instructive to visualize the regularization process graphically. In the upper panel of Figure 6.3, the ridge traces for the diabetes dataset are shown with a vertical line in the plot at the optimal value. Each intersection of the line with one of the traces is the corresponding estimate for that parameter.

To complete the picture, we also plot the variances as a function of λ in the lower panel of Figure 6.3. Note the large drop in the variances for `s1, s2, s3, s4`, and `s5`. This is exactly what we would like to happen since they are exceedingly high. Further reduction beyond the optimal point acts to increase the bias so it is not effective to go beyond it. Meanwhile the variables `age, sex, bmi, bp`, and `s6` all begin and end with essentially the same variances. Through this example, it is clear that using the ridge penalty is very effective at reducing the variances of certain offending parameters while leaving other parameters with low variances about the same.

3 This dataset can be obtained from the sklearn Python library of datasets and accessed using `diab = datasets.load_diabetes()`.

4 Note that we are using $\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}^{\text{LS}}$ in Eq. (6.4.5).

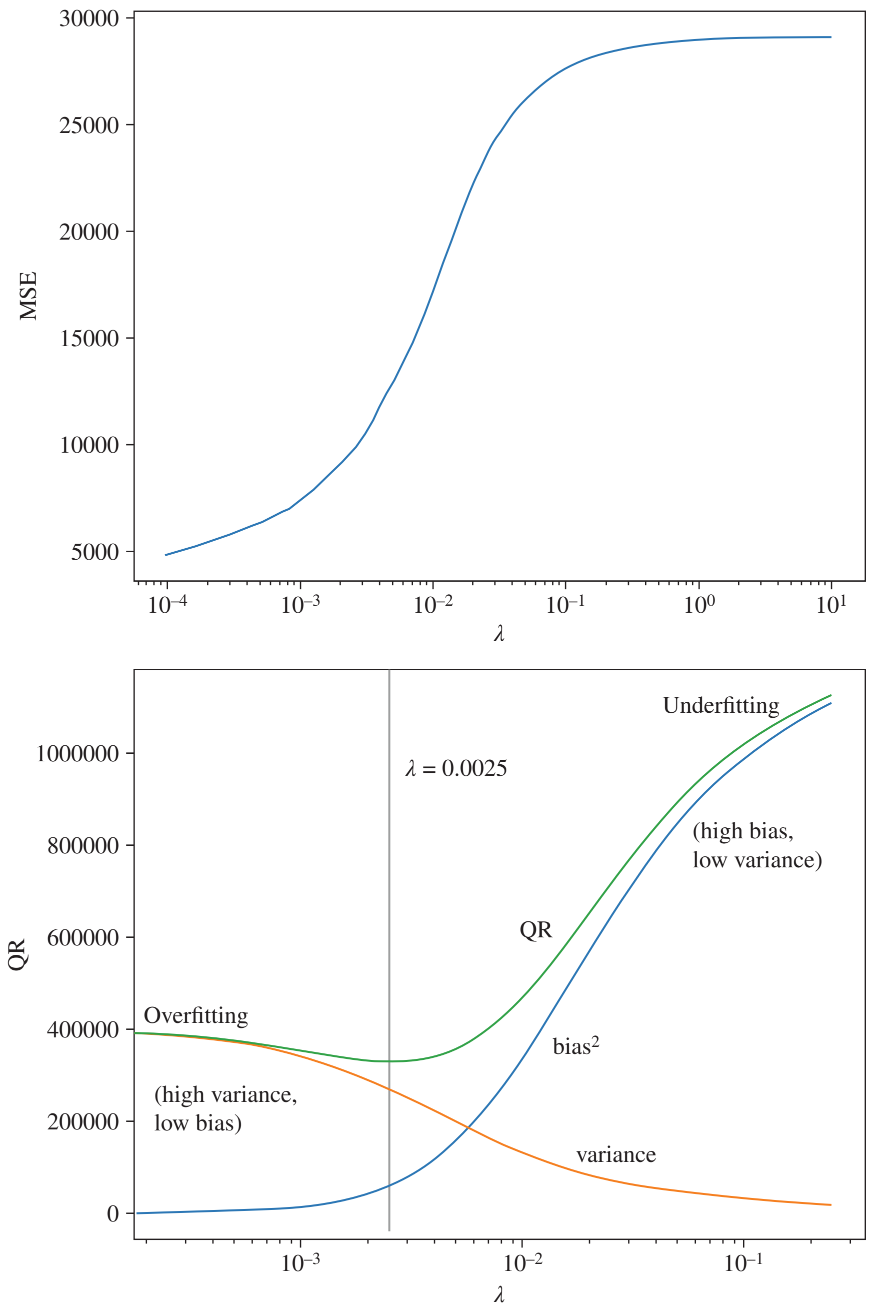


Figure 6.2 MSE compared to QR for diabetes dataset (optimal $\lambda = 0.0025$).

Table 6.1 Diabetes estimates and variances for LS ($\lambda = 0$) and LS-ridge ($\lambda = 0.0025$).

Feature	$\hat{\beta}^{\text{LS}}$	Var($\hat{\beta}^{\text{LS}}$)	$\hat{\beta}^{\text{LS-ridge}}$	Var($\hat{\beta}^{\text{LS-ridge}}$)
age	−10.0	3570.0	−8.9	3536.5
sex	−239.8	3745.7	−238.1	3709.0
bmi	519.8	4422.3	520.8	4365.9
bp	324.3	4277.2	323.0	4230.5
s1	−792.1	173622.2	−618.3	104173.8
s2	476.7	114944.7	338.8	70905.5
s3	101.0	45156.3	24.4	30714.3
s4	177.0	26082.3	156.4	23521.5
s5	751.2	29549.6	685.0	19774.7
s6	67.6	4353.9	68.7	4306.7

6.5 Generalization using Robust Estimators

The procedures described above are useful for comprehension of the bias–variance trade-off but cannot be used directly in machine learning. This is because the bias and variance are not usually available in closed form. What we actually take away from that exercise is an understanding of the important trade-off which will show up time and time again in different forms. In fact, we are more interested in generalizing the model in terms of prediction accuracy on a test set whereas in the previous section we optimized for quadratic risk on a training set. Furthermore, if there are outliers in the dataset, LS is not effective so different loss functions and metrics are needed.

6.5.1 Training and Test Sets

The first issue to resolve is that of not having any test data to evaluate a candidate model against other possible models. The approach used in machine learning is to divide the given dataset into two parts: a training set (say, 80% of the data) and a test set (the remaining 20% of the data), as in Figure 6.4. The training set contains $n - m$ rows while the test set contains m rows. Then, the model can be built using only the training data and then applied to the test data to determine the prediction error.

The error measure on the test set can be any suitable metric, such as the MSE, root mean square error (RMSE), or median absolute error (MAE), depending on the situation at hand. For example, the test error could be measured as

$$\text{MSE}_{\text{test}} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2, \tag{6.5.1}$$

where y_i and $\hat{y}_i$ are the true and predicted values of the ith response on the test set, respectively. Alternatively, the RMSE given by

$$\text{RMSE}_{\text{test}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}, \tag{6.5.2}$$

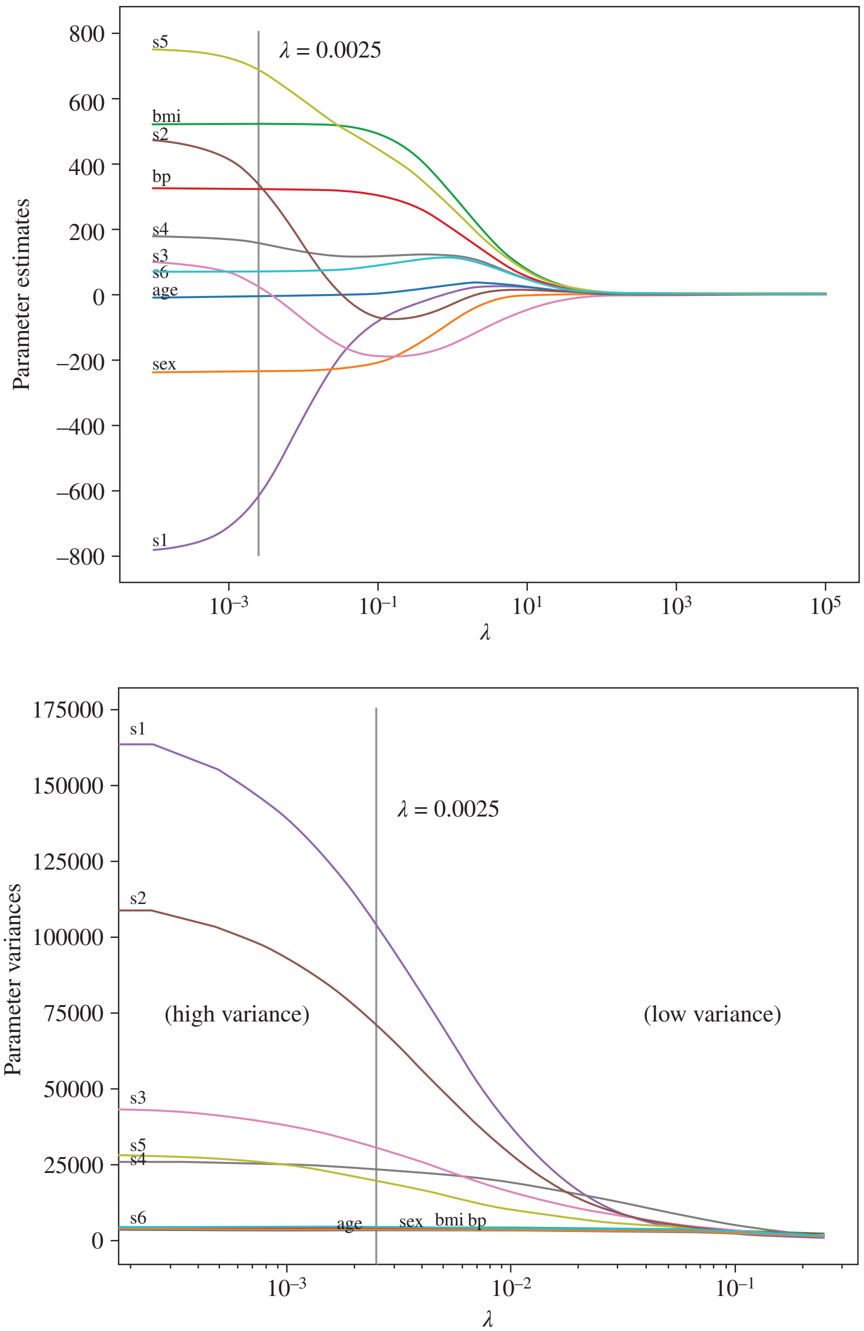
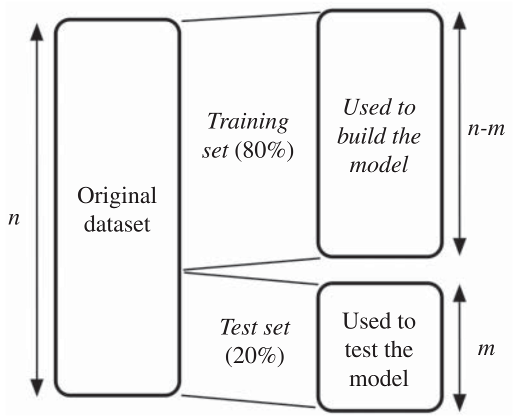


Figure 6.3 LS-ridge trace (upper panel) and variance (lower panel) vs. λ for diabetes dataset (optimal $\lambda = 0.0025$).

Figure 6.4 Splitting a dataset into training and test sets.



can be used. For a robust test error, the MAE given by

$$\text{MAE}_{\text{test}} = \text{median}(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \dots, |y_m - \hat{y}_m|) \tag{6.5.3}$$

is appropriate.

There is still a problem with this overall approach. If we tune the model to one test set, we run the risk of not being able to handle other test sets. We need more training sets and test sets to produce a general model. There is a simple way to resolve this issue. Any number of training and test sets can be generated using randomized splits of the original dataset. This is referred to as the *k-random validations* approach. It produces a more general model because of the randomized nature of the validation scheme. When k training sets are used to build k models and validated on k test sets, the average of those models is considered to be generalized. Typically, k in the range of 5–10 is sufficient to generalize the model.

6.5.2 *k*-Fold Cross-validation

In standard machine learning applications, a more coherent scheme is used called k-fold cross-validation (see James et al. 2023). The previous approach may have overlapping training and test sets. However, once we build the final model with the whole dataset, there is no longer a test set to validate the overall results. If our goal is to produce a generalized model, then we should use the dataset in a way that reflects this goal by creating several "independent" datasets. But how do we do this?

Given a dataset, we first divide it into a training set and a test set (which we now view as an unseen test set called the "holdout" set). The test set can be used to assess our final model prediction accuracy. We do not want to inadvertently tune λ to a specific test set or even a randomized (and potentially overlapping) subsets as described above since the prediction accuracy may not completely generalize to arbitrary test sets.

Instead, it is common practice to further subdivide the training set into k disjoint sets, called cross-validation (CV) sets or "folds." The decomposition of the dataset into a 5-fold cross-validation scheme is illustrated in Figure 6.5. The partitioning on the right represents one of the five folds. The four unshaded areas collectively form the CV training set for one fold while the one shaded area is the CV test set for that fold. Similarly, each of the other segments represents a different fold. At the end of the

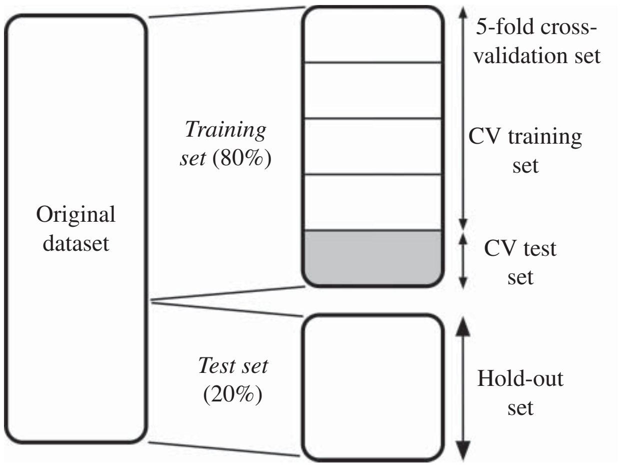


Figure 6.5 *k*-fold cross-validation.

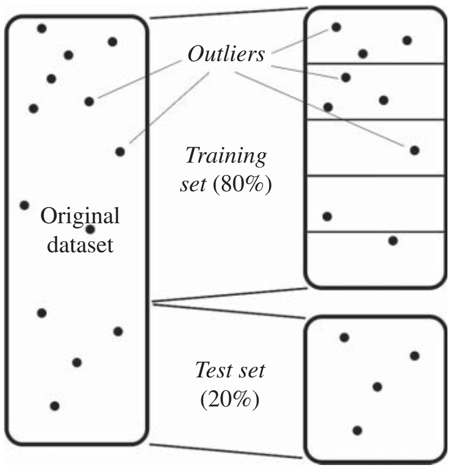


Figure 6.6 Addressing the issue of outliers in *k*-fold CV.

5-fold training process, we will have five sets of estimates that we can average to obtain the generalized model. The final model can be tested on the holdout set to confirm the results.

One often-neglected issue concerning *k*-fold cross-validation is the impact of outliers. The question that arises is whether or not a *k*-fold CV can be used to mitigate the effects of outliers since it produces a more general solution. To answer this question, consider the diagram in Figure 6.6. Each dot represents a potential outlier in the dataset. When applying *k*-fold CV, it is not clear where the outliers will land. They could be in the holdout set or in the CV test set or in the CV training set or sprinkled throughout all of them, depending on the number of outliers. Without knowing the exact location, one cannot blindly use a non-robust method. Therefore, the *k*-fold CV method should not be viewed as a solution to the problem of outliers. However, a robust method in conjunction with *k*-fold CV will certainly be more appropriate for model generalization in the presence of outliers.

Second, the use of MSE with outliers may not be suitable either. It is better to use RMSE and MAE together because they will have the same units of measure. If these two metrics differ significantly, there may be outlier issues in the dataset. The final test with the holdout set should also make use of both RMSE and MAE to validate the model. The reasons for the selection of these two metrics are discussed in Chapter 4.

6.6 Robust Generalization and Regularization

The combination of robust generalization and regularization will now be applied to the diabetes dataset to select λ. Before applying any of these methods, one should check to see if there are outliers in the dataset. One approach is to use a boxplot on the residuals following a linear regression, as shown in Figure 6.7. Here, both least squares (LS) and log-cosh (LC) losses are used on the full dataset to obtain residuals to create the boxplots (see Chapter 4). The boxplot of the LS residuals indicates only one outlier (below the boxplot). The boxplot of LC residuals indicates three outliers (one above and two below). The latter is of course correct since LS is not robust and therefore not able to find all the outliers. This is a key point to remember: detecting outliers requires the use of a robust method since LS will try to distribute the residuals as a Gaussian which tends to hide some of the outliers.

Having found three outliers, it is prudent to use robust methods even though the inclination would be to ignore the outliers and forge ahead with LS. It is not possible to tell how the three outliers will impact the estimates. For the diabetes dataset, these three outliers may not have a significant impact but we will not take the risk and utilize robust regularization with ridge. One thing to note is that the ridge penalty acts in roughly the same manner for robust and non-robust methods for linear regression.

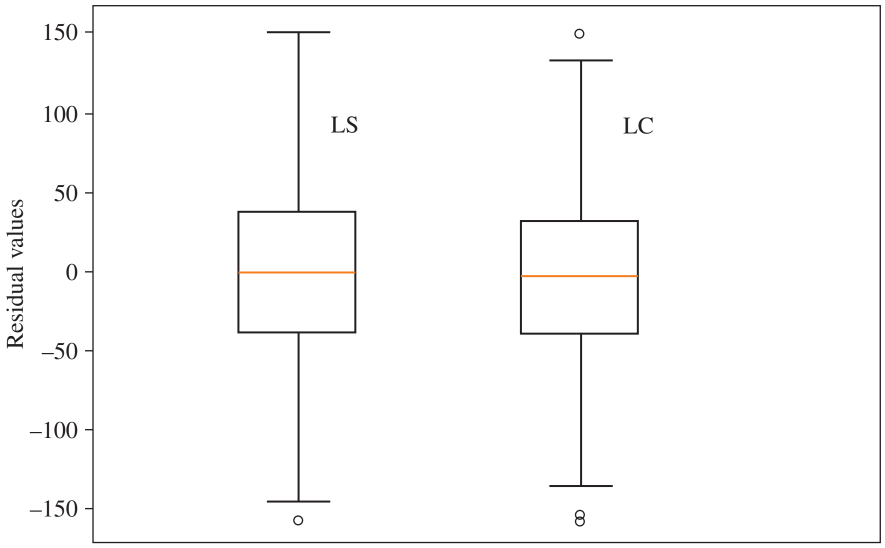


Figure 6.7 Boxplots of residuals from LS and LC for diabetes dataset.

6.6.1 Regularization with LC-Ridge

Knowing that outliers exist in the diabetes dataset, we will be using LC-ridge to build a general regularized model that balances bias and variance. Since we no longer have access to closed-form expressions for the bias or variance, the dataset is divided into training and test sets for the k-randomized validations technique. This will provide visibility into the internal workings of cross-validation. To evaluate the performance, the RMSE on the test set will be used. This is feasible because the outliers do not significantly affect the results in this case. More generally, we should consider both RMSE and MAE. For each generated training/test pair, a range of λ values is selected, and the one with the minimum RMSE determines the best λ for LC-ridge.

The first step in the process is to find the range over which to sweep the hyperparameter λ. This range can vary greatly depending on the loss function being used so a few attempts at generating the LC-ridge traces may be required but it is important to find the proper range to obtain any meaningful results. For LC-ridge using the diabetes dataset, the optimal value of λ lies somewhere between 10^{-8} and 10^{0}.

Next, an 80–20 train/test split of the data is performed. Then, an LC-ridge regression is performed for a number of λ values in the given range. For each value of λ, the prediction RMSE of Eq. (6.5.2) is computed on the test set based on the model from the training set. The results are plotted in Figure 6.8 on the left panel for one randomized validation set. Note the characteristic dip in the curve until the minimum is reached, after which it rises to the saturation value as λ increases. At one end, there is a low bias and a high variance due to overfitting and, at the other end, there is a high bias and a low variance due to underfitting. Even though we have not explicitly computed the bias or variance, it is clear that the RMSE plot obeys the basic characteristics of the bias–variance trade-off.

We could accept the minimum at approximately $\lambda = 3 \times 10^{-6}$ but this is only one randomized run on one test set. It may not generalize to other test sets. Therefore, we should run this experiment a few times on several randomized train/test combinations until a more general result is produced. This is shown in the right panel of Figure 6.8. The gray curves are five different cases of the randomized trials. The black curve is the mean of the five curves. It indicates a minimum at $\lambda = 8 \times 10^{-6}$. This is a more general

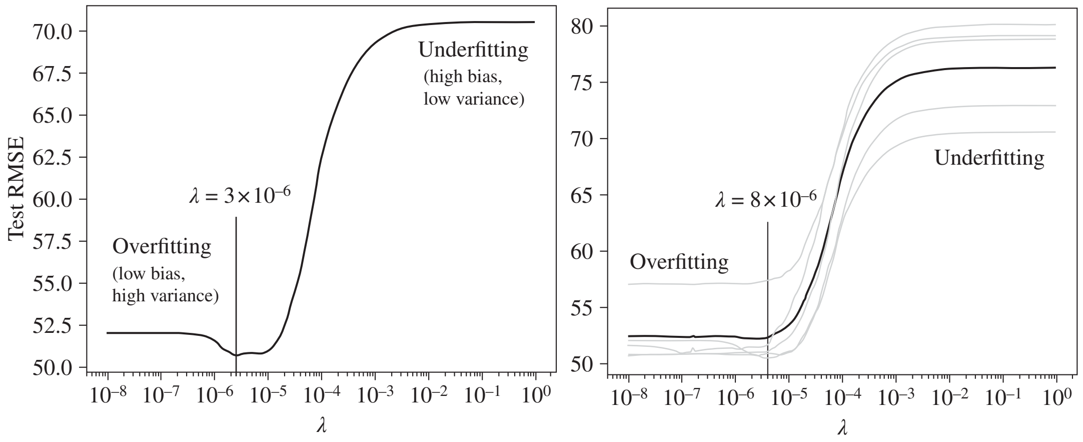


Figure 6.8 Test RMSE for one trial (left panel) and five trials (gray, left panel) with embedded mean curve (black).

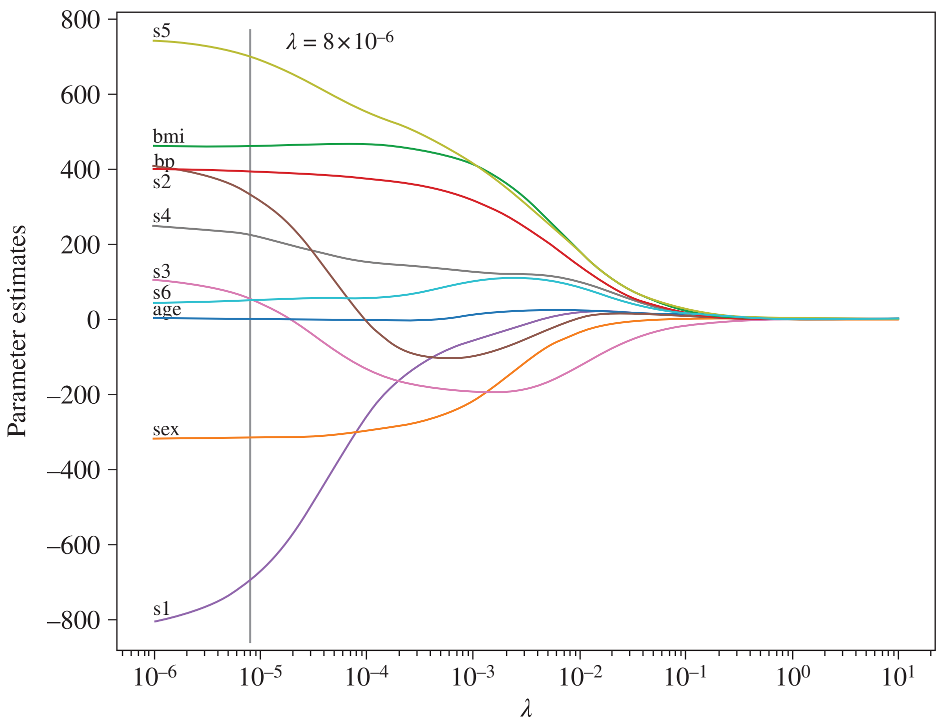


Figure 6.9 Traces of LC-ridge estimates as a function of λ.

result based on five trials. We could continue to do more runs but roughly the same result would be obtained.

The last step is to perform estimation using LC-ridge and $\lambda = 8 \times 10^{-6}$ to obtain the model. Consider the LC-ridge traces of Figure 6.9 with a vertical line representing the resulting model. Note that it has similar characteristics to the LS-ridge traces in Figure 6.3 and the vertical line is roughly in the same place. One line was produced using the quadratic risk and the other using the test RMSE. In this case, with only three outliers the results are expected to be similar. However, in other datasets, with a larger percentage of outliers, the LS-ridge results may not be acceptable since the estimates will not be accurate and the variances will be large. Therefore, datasets with outliers will require robust regularization to build an accurate and general model.

6.7 Model Complexity

The complexity of a model generally refers to the number of parameters used by the model. A model with only a few parameters is a low-complexity model, and one with a large number of parameters is a high-complexity model. It is a goal in data science to produce a model that uses the fewest variables while still retaining high predictive power. There are many reasons for this. One is that fewer variables produce a more compact model which is highly desirable to avoid overfitting. It is also easier to assess their relative importance if there are fewer of them. The second issue is that there are variables that add noise to the model, and it is preferable to remove these variables. Third, and perhaps more subtly, the variance of the residuals (which ideally should be small) can be large if there are too many variables in the model. To understand this point, consider the unbiased estimator of residual variance which is given by

$$\hat{\sigma}^2 = \frac{1}{n-(d+1)}\sum_{i=1}^{n}(r_i - \overline{r})^2. \tag{6.7.1}$$

As $(d+1)$ approaches n, the residual variance goes to infinity. Hence, keeping d small will reduce the variance of all parameters because they all depend on $\hat{\sigma}^2$.

In effect, we have another bias–variance trade-off involving the number of variables in our model. If we have too many variables, we risk overfitting the dataset and that leads to a high variance. If we have too few variables, we risk underfitting which creates a high bias situation. Our next objective will be to seek the optimal number of variables using some form of feature selection.

6.7.1 Variable Selection Using LS-LASSO

One useful application of penalty functions involves using the LASSO penalty to select the optimal number of parameters. The diabetes dataset will be used to demonstrate the procedures. It has 10 variables, namely, `s1`, `s2`, `s3`, `s4`, `s5`, `s6`, `age`, `bmi`, `bp`, and `sex`. With domain knowledge, the relative importance of each variable may already be known. However, the approach to be described can be used with or without domain knowledge. The first step is to generate the LS-LASSO traces as shown in Figure 6.10. In contrast to the LS-ridge traces, the LS-LASSO traces all terminate abruptly on the λ-axis. The zero points are indicated with a "•" for each parameter. By selecting a suitable λ, a subset of the variables can be removed. This is how it is able to perform variable selection by setting some variables to 0.

To clarify the procedure, if a vertical line is drawn at $\lambda = 0.25$ on the LS-LASSO traces, then any variable to the left of the line can be removed, while those on the right are kept. In this case, `s1`, `age`, `s2`, `s4`, and `s6` are all set to 0, meaning that they are removed from the model. By doing so, a compact model is produced for the diabetes dataset involving only `s3`, `s5`, `bp`, `bmi`, and `sex`. The LS-ridge traces do not provide this ability since all variables tend to reach 0 gradually rather than abruptly (cf. see Figure 6.3).

6.7.2 Variable Ordering Using LC-aLASSO

There are many different ways to assess variable importance but most of them rely on an outlier-free dataset. For example, the adjusted-R^2, BIC, AIC, and C_p metrics used for subset selection (see James et al. 2023)[5] require either the mean or the standard deviation, which are not robust. Therefore, we suggest removing outliers (see Chapter 4) before using these techniques. Another approach is to use the absolute value of the t-statistic (see Chapter 5) obtained from an LC linear regression. In this section, we describe an alternative method using an oracle penalty function, specifically aLASSO. This method provides a visualization of variable ordering using penalty traces (see Saleh et al. 2022).

5 See James et al. (2023) for details on these metrics and the associated subset selection techniques.

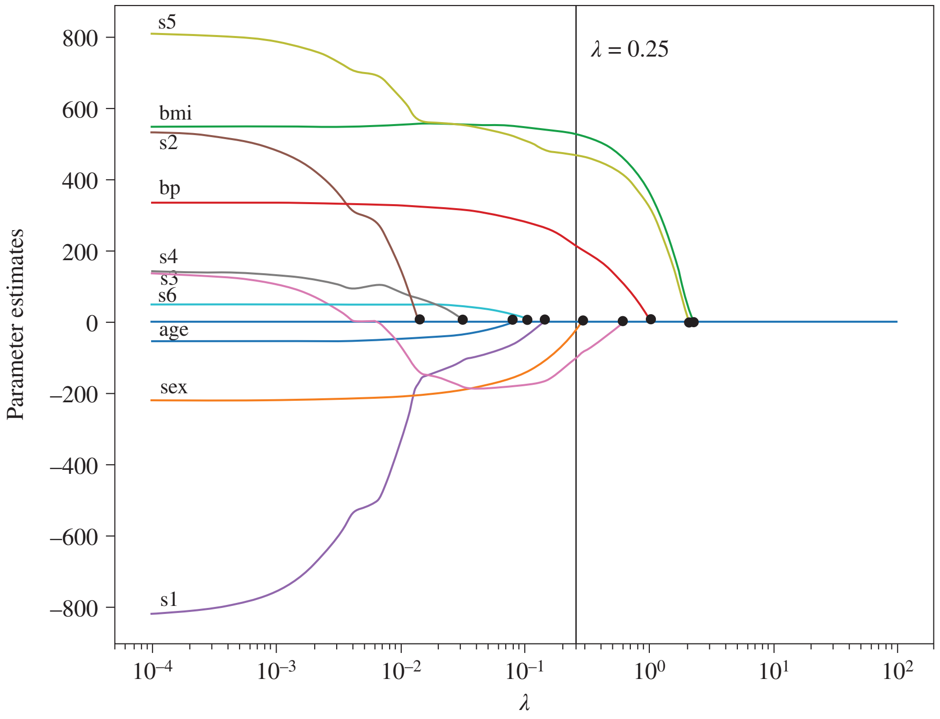


Figure 6.10 Traces of LS-LASSO estimates as a function of λ.

Before proceeding further, we note that LS-LASSO is extremely valuable in high-dimensional applications where there are a large number of variables, say $d = 10,000$, because it will remove most of the unimportant variables. But here we are concerned with a proper ordering of a smaller number of variables. In this situation, there are two notable problems with LS-LASSO. The first is obviously the lack of robustness of LS. The second is the lack of the oracle property in LASSO (see Chapter 5). We can remedy both by utilizing LC-aLASSO.

The first step is to generate the LC-aLASSO traces as shown in Figure 6.11. The figure provides the LC-aLASSO variable ordering labeled from 1 to 10. According to LC-aLASSO, the most important variable is `s5`, then `bmi` and then `bp`. If we compare it against Figure 6.10, it is clear that the variables reach 0 in a different order compared to LS-LASSO. In the case of LS-LASSO, `bmi` is the most important, slightly more than `s5` and then a distant third is `bp`. The complete orderings of all 10 variables are provided in Table 6.2 along with another ordering based on the t-statistic of the unpenalized LC estimates. Recall from Chapter 5 that, by ranking the |*t*-statistic|, we obtain an approximate ordering of the variables. The *t*-statistic itself is the ratio of the estimate to the standard error. Note that all three methods (*t*-statistic, LC-aLASSO, and LS-LASSO) produce different orderings (see columns 5, 6, and 7). Which is correct? The *t*-statistic method and LC-aLASSO are the same for the last five variables. However, the only one with the oracle property is LC-aLASSO and we know that asymptotically, it will produce the correct order. According to the theory, this is a good ordering of the variables based on the given data.

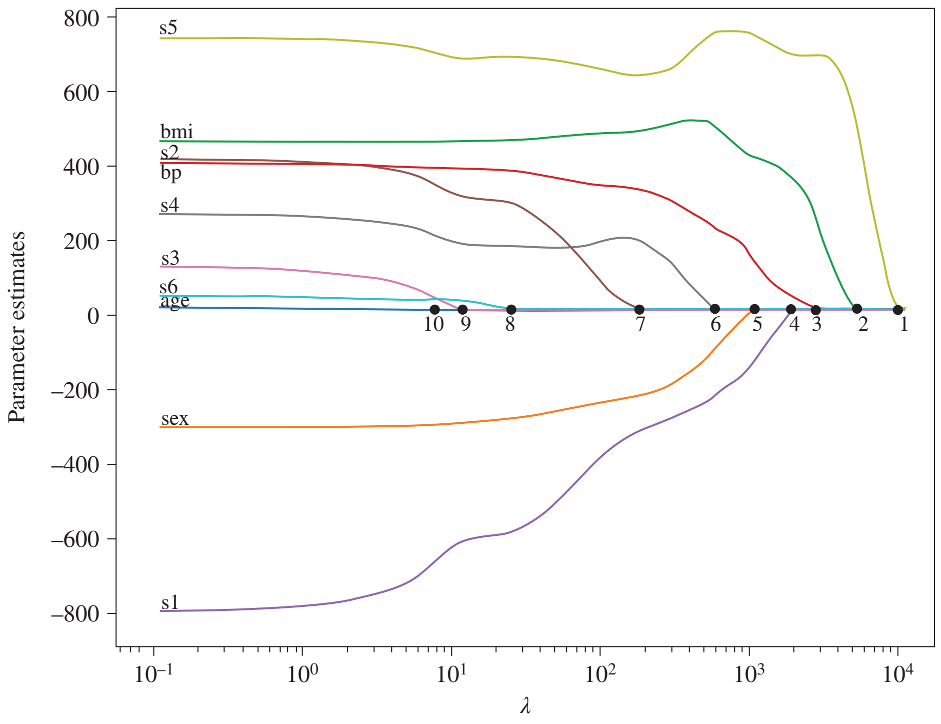


Figure 6.11 Traces of LC-aLASSO estimates as a function of λ, with variable ordering labeled from most important (1) to least important (10).

But how can this be verified? There are many ways to do this but one way is to simply create a training set and test set and then run 10 different LC regressions without a penalty function. The first run will have only one variable; the second with two variables, and so on. The starting variable is the most important one, followed by the addition of the next most important and so on until all 10 variables are included. The variable order is provided in the LC-aLASSO column of the table. For each case, we obtain the estimates on the training set and monitor the prediction error on the test set. The resulting plot should have a convex shape according to an inherent bias–variance trade-off. If there are any spikes in the plot, there may be a problem with the ordering, perhaps due to the limits of a finite amount of data.

To proceed, we first define a training set and a test set, but this time we choose an even split between the two. If we do not use a 50–50 split on the diabetes dataset, the estimates may not be accurate or the test MSE may not be reliable. In general, there needs to be sufficient data in both the training and test sets to obtain acceptable results. At each step, the number of regression variables will be different. The training and testing loops can be performed using k-random validations and the test MSE is averaged over all the trials. Note that the effect of outliers is not significant in the diabetes dataset so the use of the MSE is acceptable.

We start with only variable `s5` in the model to be built using the training set and then compute the test MSE. As we incrementally add `bmi` and then `bp` to the regression model, we expect the test MSE to decrease because the model accuracy is improving. However, at some point, noise variables will be added

Table 6.2 Estimates and standard errors (s.e.) and t-statistic for log-cosh (LC), followed by variable ordering for LC t-statistic, LC-aLASSO, and LS-LASSO.

	$\hat{\beta}^{LC}$	s.e. ($\hat{\beta}^{LC}$)	t-Statistic ($\hat{\beta}^{LC}$)	LC t-statistic order	LC aLASSO order	LS LASSO order	Category
(Intercept)	151.52	2.7	55.78	0	0	0	
`s5`	745.31	181.2	4.11	4	1	2	
`bmi`	461.58	70.1	6.57	1	2	1	Critical
`bp`	401.75	68.9	5.82	2	3	3	
`s1`	−829.82	439.4	−1.88	5	4	7	
`sex`	−322.71	64.5	−4.99	3	5	5	Important
`s4`	261.88	170.2	1.53	6	6	8	
`s2`	411.21	357.5	1.15	7	7	9	
`s6`	41.52	69.5	0.59	8	8	6	Unimportant
`s3`	119.52	224.1	0.53	9	9	4	(noisy)
`age`	6.46	63.0	0.10	10	10	10	

and therefore the test MSE will increase. The results of this validation process are plotted in Figure 6.12. The MSE decreases at each step until six variables are used. Then it begins to increase according to the underlying bias–variance trade-off.

The results validate the aLASSO ordering and also tell us that six variables will be sufficient to build a compact but accurate model. We can now go back to Table 6.2 and understand the categorizations in the final column. The critical and important variables are identified in the first two groups. They are `s5`, `bmi`, `bp`, `s1`, `sex`, and `s4`. These variables must be included in the model. The remainder may be optionally included but they are noise variables and should be considered for removal. Note that none of this is possible without an initial ordering from the aLASSO traces.

6.7.3 Building a Compact Model

Now that the key variables in the diabetes dataset have been identified, a compact and accurate model can be built using LC-ridge with a k-fold cross-validation scheme. Typically $k = 5$ or $k = 10$ is used depending on the size of the dataset. A 5-fold scheme is used here. The first step involves subdividing the training set into five smaller CV training and test sets (hence, 5-folds) using only the variables `s5`, `bmi`, `bp`, `s1`, `sex`, and `s4`.

The estimates from each of the five CV training sets are used to compute the prediction MSE on each of their associated five CV test sets. This is done for each selected value of λ. That is, a set of five estimates is produced and then averaged for each chosen λ. Similarly, the average prediction MSE is computed for each case. It is useful to keep track of the maximum and minimum MSE for each λ for plotting purposes. Finally, we choose the λ that minimizes the average MSE among all selected values of λ used during

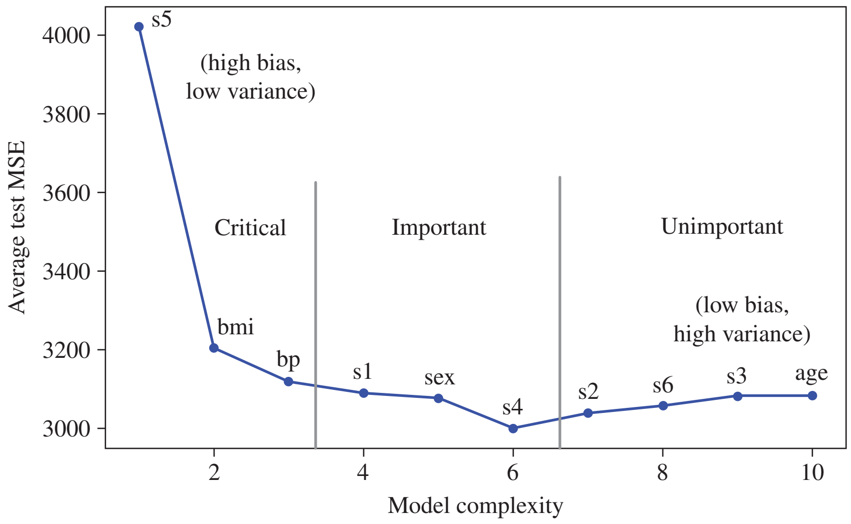


Figure 6.12 Test MSE as a function of model complexity.

cross-validation. Of course, we do not explicitly compute the bias or variance at any time. Rather, the prediction MSE serves as a proxy for the quadratic risk since it can be readily computed for any dataset.

The results are shown in Figure 6.13. On the top panel, the average CV MSE is plotted along with the holdout test MSE. The minimum of the CV MSE occurs at roughly $\lambda_{\text{opt}} = 3 \times 10^{-6}$. The holdout test set requires a higher $\lambda = 10^{-5}$ but this should not be taken into account. In fact, the λ_{opt} value found by 5-fold CV is intended to be located in such a way as to provide acceptable accuracy for all possible test sets. This is the meaning of the term "generalized model." It is the best overall value since some test sets would prefer a smaller λ, while others a larger λ. The estimates are indicated by the vertical line in the LC-ridge traces in the lower panel.

It is important to note that the CV test MSE plots show the signature bias–variance trade-off. The vertical bars indicate the range of the min/max MSE values of the CV test set. The holdout test set MSE plot is included only to show that its minimum value may not occur at the same point as the CV test MSE. Normally, it would not be shown. Notice the long seemingly flat region of the MSE plot. One may wonder why we choose $\lambda = 3 \times 10^{-6}$ when the MSE is essentially constant for the entire range of 10^{-8} to 10^{-5}. The answer is that the variance is reduced as λ is increased and this means less uncertainty in the estimates, which is a good thing. Therefore, a larger λ provides an enormous benefit. However, we cannot go too far, otherwise the bias will increase.

What has effectively been carried out in the above sequence is depicted in Figure 6.14. The original dataset with $n = 442$ and $d = 10$ was used with LC-aLASSO to extract the variable ordering. A model complexity plot was used to reduce the number of variables to $d_o = 6$. At this point, the noise variables have been removed. Then a 5-fold CV scheme can be used with LC-ridge to find the optimal λ. Since a robust method is used, the three outliers are effectively ignored implying that $n_o = 439$ in this case. We could have manually removed the outliers but we used a robust method to avoid this unnecessary step. The final model is obtained using the optimal λ and then verified on the holdout set to complete the task.

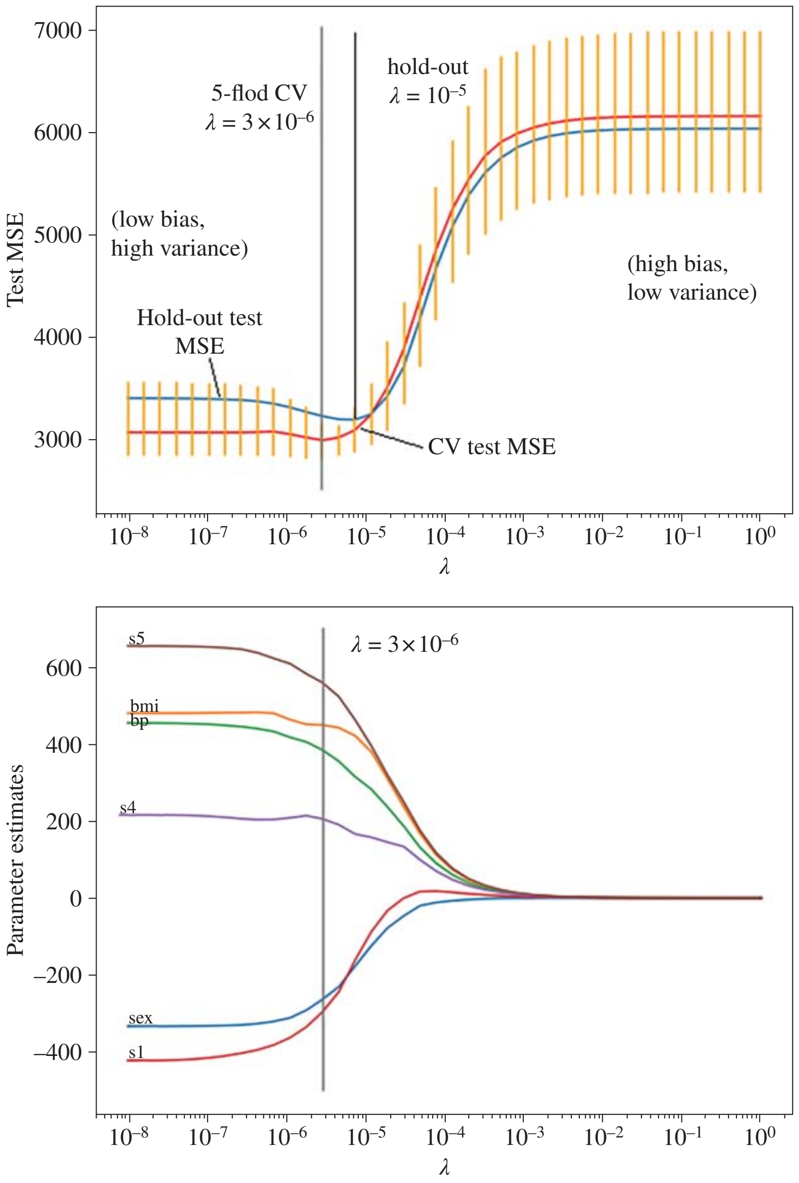


Figure 6.13 Test MSE (upper panel) and LC-ridge traces (lower panel) for diabetes dataset.

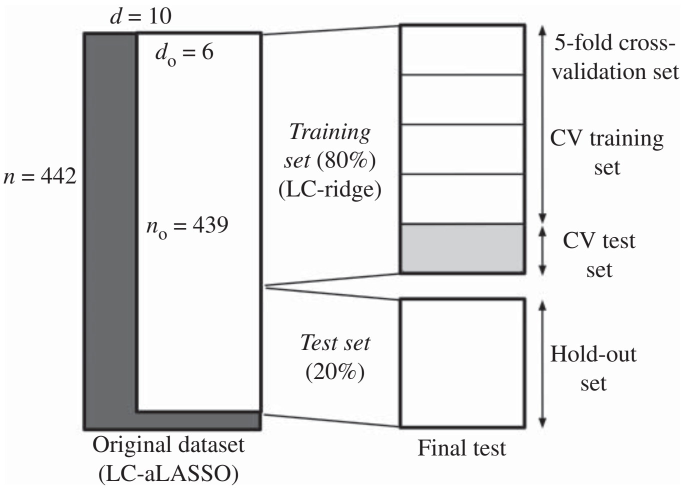


Figure 6.14 Splitting the diabetes dataset into a CV training and test sets and a holdout test set.

6.8 Summary

This chapter focused on the bias–variance trade-off and its use in building compact and accurate prediction models. The mathematical origins of the trade-off in the quadratic risk formula were detailed along with its application to the diabetes dataset. Methods to generalize and regularize models using a k-fold cross-validation scheme with a penalty function were described. Notably, the effect of outliers cannot be mitigated using these techniques. Therefore, robust methods must be used for linear regression. A robust methodology for the development of a compact prediction model for linear regression was described using the log-cosh loss function. This involved the use of penalty functions, specifically ridge and aLASSO. Variable ordering and regularization procedures were explained in detail so that data science methodologies could be developed for use in other applications. In particular, these methods can also be applied to logistic regression, as described in Chapter 8.

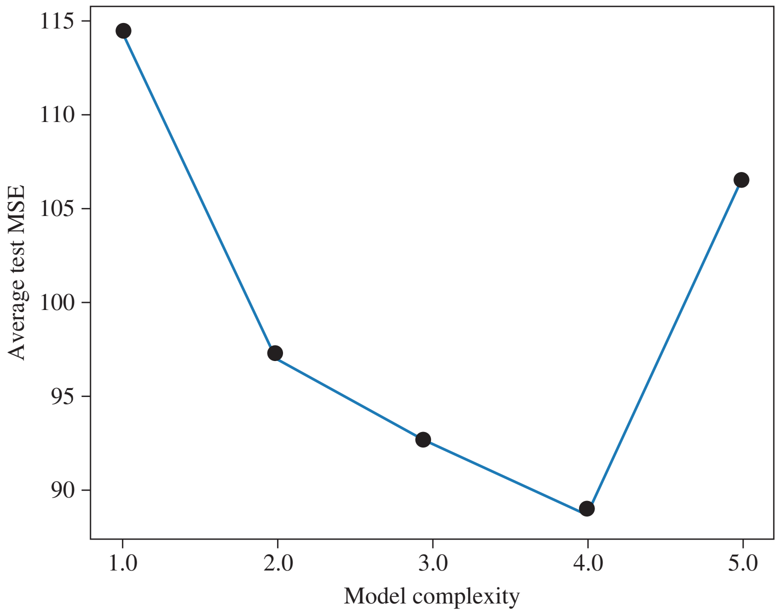
Problems

6.1 The equation for the quadratic risk (QR) in the orthogonal case was derived to be:

$$\mathrm{QR}(\hat{\boldsymbol{\beta}}^{\text{LS-ridge}}) = \frac{d\sigma^2}{(1+\lambda)^2} + \frac{\lambda^2}{(1+\lambda)^2}\boldsymbol{\beta}^\top\boldsymbol{\beta}.$$

a) What is the formula for the optimal value of λ in terms of d, σ^2, and $\boldsymbol{\beta}$? (Hint: take the derivative and set it to 0.)
b) Let $d = 3$, $\sigma^2 = 1$, and $\hat{\boldsymbol{\beta}} = (1, 2, 3)^\top$. Plot the variance, bias^2, and QR.
c) Estimate the λ associated with the minimum value of QR according to the plots.
d) Use the formula from part (a) to compute the optimal λ.

6.2 The figure below was generated by ordering the 5 variables from the Swiss Fertility dataset according to the LC-aLASSO results given in Chapter 5, Table 5.2. Then each data point was produced by running 1 variable, then 2, 3, 4, and 5, as described in Section 6.7.2.



Answer the following questions. Assume the following abbreviations: `Ex=Examination`, `Ed=Education`, `C=Catholic`, `A=Agriculture`, and `IM=Infant Mortality`. What is the ordering of the variables among the following four options? That is, what labels should be applied to the vertices shown in the plot and in what order?

a) `Ex, Ed, C, A, IM`
b) `Ed, Ex, C, A, IM`
c) `Ed, C, IM, A, Ex`
d) `Ed, IM, C, A, Ex`

Which variable is most important and which is least important? Would it make sense to remove the least important variable in this case?

6.3 (PROJECT) In this first project, you will use the k-random validations approach to find λ_{opt} for an LC-ridge regression. This will be done using the Swiss Fertility dataset which was given in Chapter 3. You will need a `swiss.csv` file to run the given code below. In this version, the `Examination` variable is removed based on the previous problem. Enter this Python code and then run the program. What is a suitable value of λ based on the CV test MSE? (Hint: a higher value of λ reduces the variance.)

```
import numpy as np
import pandas as pd
```

```
from scipy.optimize import minimize
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

swiss = pd.read_csv("swiss.csv")
plt.figure(figsize=(8, 6))
n_points = 20
lambdas = np.logspace(-2, 7, n_points)
num_trials = 8
avg_mse = np.zeros(n_points)
for i in range(num_trials):
    mses = []
    mse = np.zeros(n_points)
    j = 0
    train, test = train_test_split(swiss, \
        test_size=0.2,random_state=5*i)
    for lambda1 in lambdas:
        x = np.array(train.drop(["Fertility", \
            "Examination"],axis=1))
        y = np.array(train["Fertility"])
        ridge = Ridge(alpha=lambda1)      #RIDGE
        ridge.fit(x, y)
        intter = ridge.intercept_
        betaRidge = ridge.coef_
        X_test = np.array(test.drop(["Fertility", \
            "Examination"],axis=1))
        y_test = test["Fertility"]
        yhat = intter + np.matmul(X_test, betaRidge)
        test_MSE = np.mean((y_test-yhat)**2)
        mses.append(test_MSE)
        mse[j] = test_MSE
        j += 1
    avg_mse += mse/num_trials
    #plt.plot(lambdas,mse)    #Use to plot all mse
ax = plt.gca()
ax.set_xscale('log')
plt.xlabel('lambda')
plt.ylabel('test MSE')
plt.plot(lambdas,avg_mse, color = 'black')
plt.show()
```

6.4 (PROJECT) In this second project, you will use the *k*-fold CV approach to find λ_{opt} for an LC-ridge regression. This will be done using the Swiss Fertility dataset as in the previous problem. Enter this Python code and then run the program. What value of λ produces the minimum CV test MSE? How does it compare with the one from the previous project?

```
from sklearn.model_selection import KFold
swiss = pd.read_csv("swiss.csv")
numfolds = 5
kf = KFold(n_splits=numfolds)
train, test = train_test_split(swiss, \
    test_size=0.25,random_state=20)
target = swiss['Fertility']
n_points = 20
lambdas = np.logspace(-3, 7, n_points)
mses = []
msetests = []

for lambda2 in lambdas:
    i = 0
    betaAvg = np.zeros(train.shape[1]-2)
    interceptAvg = 0.0
    mse = np.zeros(numfolds)
    kf = KFold(n_splits=5)
    for train_index, test_index in kf.split(train):
        X_tr, X_cv = train.iloc[train_index], \
            train.iloc[test_index]
        Y_tr, Y_cv = target.iloc[train_index], \
            target.iloc[test_index]
        x = np.array(X_tr.drop(["Fertility", \
            "Examination"],axis=1))
        y = X_tr["Fertility"]
        ridge = Ridge(alpha=lambda2)      #RIDGE
        ridge.fit(x, y)
        intter = ridge.intercept_
        betaRidge = ridge.coef_
        X_te = np.array(X_cv.drop(["Fertility", \
            "Examination"],axis=1))
        y_te = X_cv["Fertility"]
        yhat = intter + np.matmul(X_te, betaRidge)
        cv_test_MSE = np.mean((y_te-yhat)**2)
        mse[i] = cv_test_MSE
        i += 1
```

```
            betaAvg += betaRidge/numfolds
            interceptAvg += intter/numfolds

        mse_avg = np.mean(mse)
        x_test = np.array(test.drop(["Fertility", \
            "Examination"],axis=1))
        y_test = test["Fertility"]
        yhatt = interceptAvg+np.matmul(x_test, betaAvg)
        msetest = np.mean((yhatt-y_test)**2)
        msetests.append(msetest)
        mses.append(mse_avg)
    plt.figure(figsize=(8, 6))
    ax = plt.gca()
    ax.plot(lambdas, mses, color='red')
    ax.plot(lambdas, msetests,color='green')
    ax.set_xscale('log')
    plt.xlabel('lambda')
    plt.ylabel('test MSE')
    plt.title('CV (red) and Test MSE (green) as ' \
        'a function of lambda')
    plt.axis('tight')
    plt.show()
```

6.5 (Advanced Project) Using the given code above, convert it to run LC-ridge. To do this, you need to replace the LS-ridge statements above with the corresponding LC-ridge statements and then run the minimizer. You will need the code given in the Problems section of Chapter 5.

References

James, G., Witten, D., Hastie, T. et al. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer.

Montgomery, D.C., Peck, E.A., and Vining G.G. (2012). *Introduction to Linear Regression Analysis*, 4e. Wiley.

Saleh A.K. Md. E., Arashi, M., and Kibria, B.M.G. (2019). *Theory of Ridge Regression Estimation with Applications*. New York: John Wiley and Sons.

Saleh, A.K. Md. E., Arashi, M., Saleh, R., and Norouzirad, M. (2022). *Rank-Based Shrinkage and Selection: With Application to Machine Learning*. New York: John Wiley and Sons.

7

Quantile Regression Using Log-Cosh

7.1 Introduction

We have studied the log-cosh loss function in Chapter 3 as a way of resolving the problems of the L_1 loss function. It was shown to be a robust approach for linear regression and useful for other tasks in machine learning such as outlier detection. In simple linear regression, the log-cosh loss establishes a median line through the points in a dataset. As we know, the median line represents the 50th percentile of a given set of data points, dividing it in half: 50% of the points lie above the line and the remaining 50% below it. However, what if we are interested in other quantiles? For instance, rather than the 50th percentile, what if we want to find the line associated with the 25th percentile (also known as the first quartile)? In this case, 25% of the points would lie below the line, while the remaining 75% would be above it. Similarly, what about the lines associated with the 10th or 20th percentiles (called deciles)?

Methods for obtaining such lines associated with a desired percentile, decile, or quartile are referred to collectively as quantile regression (see Koenker and Bassett 1978; Koenker 2005). This chapter focuses on quantile regression which has been successfully applied to problems in econometrics, as well as in survival analysis and computational biology. The method is well established in these areas and is naturally robust to outliers. However, there exists a known problem with the original approach to quantile regression: the so-called crossing problem. This issue is widely recognized and understood to be troublesome. Specifically, it refers to the lack of monotonicity in the estimation of the conditional and structural quantile functions. Unfortunately, the crossing problem has confounded researchers and practitioners for over four decades. Numerous attempts have been made to find a simple and general solution.

To tackle this challenge, an intriguing alternative lies in the log-cosh function. This flexible function offers a solution that is easy to understand and implement. The basic idea is to replace the piecewise linear quantile loss function with a smooth log-cosh counterpart. This approach, akin to what was done in Chapter 3 for the L_1 loss, greatly reduces or even eliminates the crossing problem entirely.

7.2 Understanding Quantile Regression

At a very basic level, quantile regression serves as a natural extension of robust linear regression. Unlike the usual form of robust regression, which seeks a single line dividing the data in half, quantile regression allows us to divide the data using any desired percentile, decile, or quartile. The term "quantile" encompasses all possible cases and is typically specified as a value between 0.0 and 1.0.

Consider Figure 7.1, which displays a scatter plot of a dataset along with different regression lines representing various quantiles. Notably, the data exhibits *heteroscedasticity*–a phenomenon where the spread of data widens like a cone as the explanatory variable increases. This implies that the variance increases proportionally. Such heteroscedasticity has implications for the quality and reliability of statistical procedures. In standard regression, we assume *homoscedasticity*, meaning that the standard errors remain consistent as we move along the x-axis. However, this assumption may not hold in cases like the one in Figure 7.1. Quantile regression, with its flexibility, becomes well suited for addressing such issues, as a single regression line may not adequately capture the true characteristics of the data.

Each set of estimates from a quantile regression produces a line in Figure 7.1 that divides the data based on the specified value of a quantile parameter $\tau \in [0, 1]$.

- For $\tau = 0.0$, we obtain the 0th percentile line located just below all the data points.
- For $\tau = 1.0$, the 100th percentile line is located just above all the points.
- For $\tau = 0.5$, the 50th percentile line divides the data in half.
- Other lines shown are for the 10th ($\tau = 0.1$), 25th ($\tau = 0.25$), 75th ($\tau = 0.75$), and 90th ($\tau = 0.9$) percentiles.

Note that as τ increases from 0 to 1, the slope of the line gets steeper as more points fall below it. Essentially, the line sweeps through the points in the radial direction (indicated by the arrow) as τ goes

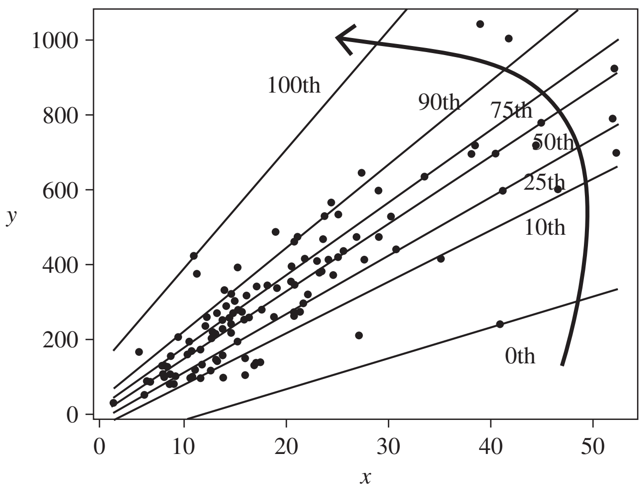


Figure 7.1 Quantile regression for different percentiles, deciles, and quartiles.

from 0 to 1. The key to the quantile regression procedure is to find a suitable loss function to optimize in order to find the quantile line or hyperplane associated with a given τ.

7.3 The Crossing Problem

To introduce the crossing problem, it is useful to consider what would be a fundamental requirement of any reasonable quantile regression method. It would be a property referred to as **monotonicity between quantiles**. As the value of τ increases, so should the number of points below the associated line or hyperplane. Therefore, one natural approach to monitor the quality of a quantile regression method is to simply count the number of data points below a given quantile regression line (or hyperplane in multiple dimensions) to ensure that it satisfies the requirements.

- The median (50th percentile) should result in a line with half the points below it.
- The third quartile should result in 75% of the data below that line.
- The 90th percentile should have 90% of the points below it, and so on.

Unfortunately, this is often not the case for two reasons:

1) Non-integer number of points: If there are, say, 47 points in a given dataset, then what exactly constitutes 75%? It would be 35.25 points. But since this is not an integer number, we will have either 35 or 36 points below the line. That implies 74.5% or 76.6%, respectively. So we have a problem due to matching the exact percentile.
2) Inadvertent point placement: The quantile regression method itself may inadvertently place more or fewer points than required below the line. In the above example with 47 points, there may be 34 or 37 points below the line for the 75th percentile instead of the expected 35 or 36 points, which can be confusing to the user. But it does happen in traditional quantile regression.

One rule that should be enforced in any reasonable quantile regression method is that a higher percentile should have more points below it than a lower percentile. It could also have an equal number but certainly not less. That is, during quantile regression, there should not be more points below the 75th percentile than at the 76th percentile. Consider again the example above with 47 total points. Assume that after running standard quantile regression, we find that:

- there are 37 points below the 75th percentile, and
- there are 36 points below the 76th percentile.

This situation would violate the basic definition of quantiles, which is not desirable. This is the type of non-monotonic behavior that can occur with quantile regression and is referred to as the *crossing problem*.

The issue is further illustrated in Figure 7.2. We see that the two lines intersect, i.e. cross, and this has implications in the resulting monotonicity of the percentiles. There are more points below the 75th percentile than the 76th percentile because the lines cross. Typically, this occurs frequently with nearby quantiles but it can also occur between quantiles that are further apart. Not all cases where lines cross actually present a problem, but the implication is that when two lines cross in the convex hull (i.e. the interior region) of the data, an inconsistency has occurred. Therefore, the crossing problem always

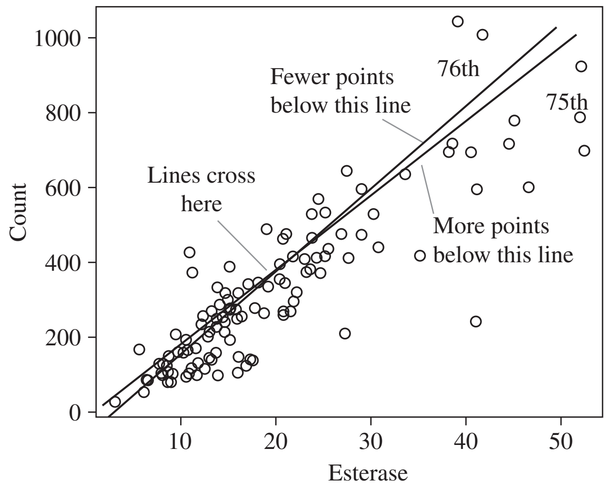


Figure 7.2 The crossing problem occurs when a higher quantile (76th percentile) has fewer points below it compared to a lower quantile (75th percentile).

manifests itself as a non-monotonicity problem as percentiles increase. But why would this occur? We will explore this question in the next section.

Note: We emphasize here that the lines and hyperplanes from quantile regression will always intersect in some way if they are not parallel. However, the model can still be viewed as acceptable if the crossing occurs outside the convex hull. Lines crossing within the convex hull often leads to non-monotonicity, but not always, whereas if non-monotonicity is detected, there is implicitly a crossing problem inside the convex hull. Therefore, we need a way to monitor any non-monotonic behavior in quantile regression to evaluate the performance of different approaches.

7.4 Standard Quantile Loss Function

Quantile regression is described in this section starting with the original method for studying the effect of explanatory variables on the entire conditional distribution of the response variable. It has been used extensively since the initial concepts were developed in the late 1970s and early 1980s. It is based on the so-called *check function* whereby the quantile of interest is obtained by setting a parameter, τ.

Different regression quantiles (RQ) represented by lines or hyperplanes are obtained by minimizing the conditional loss function in M-estimator form given by,

$$J^{\mathrm{RQ}}(\boldsymbol{\beta}|\tau) = \sum_{i=1}^{n} \rho_{\mathrm{RQ}}(r_i;\tau), \tag{7.4.1}$$

with

$$\rho_{\mathrm{RQ}}(r_i;\tau) = \begin{cases} -(1-\tau)r_i & \text{if } r_i < 0 \\ \tau r_i & \text{if } r_i \geq 0 \end{cases}, \tag{7.4.2}$$

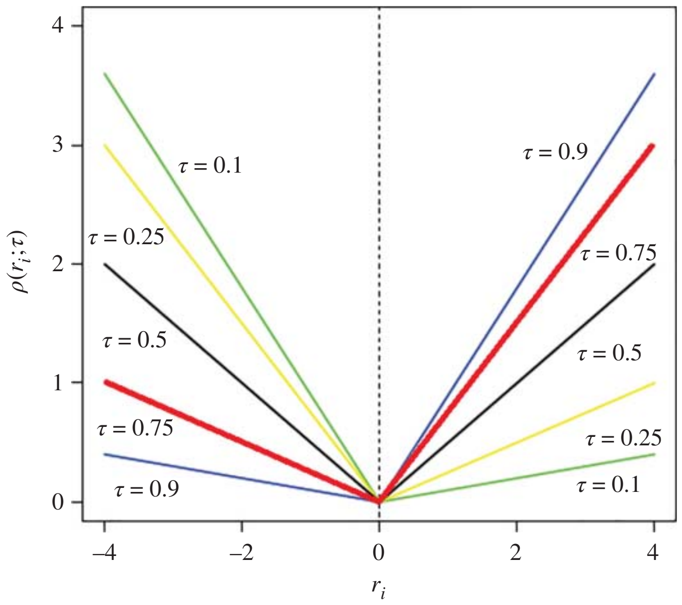


Figure 7.3 The standard check function used in quantile regression showing a number of cases with different values of τ.

where the parameter $\tau \in [0, 1]$ is the quantile and r_i is the ith residual given by $r_i = y_i - \boldsymbol{x}_i^\top \hat{\boldsymbol{\beta}}^{\text{RQ}}$. To understand this loss function, it is useful to examine a plot of $\rho_{\text{RQ}}(r_i;\tau)$ for different values of τ as provided in Figure 7.3.

Suppose we are interested in the median, which is also the 50th percentile and the 2nd quartile. We would set $\tau = 0.5$ and use the following function:

$$\rho_{\text{RQ}}(r_i; 0.5) = \begin{cases} -(0.5)r_i & \text{if } r_i < 0 \\ 0.5r_i & \text{if } r_i \geq 0. \end{cases} \tag{7.4.3}$$

This is equivalent to the least absolute deviation (LAD) function or the L_1 loss term. We see from Figure 7.3 for $\tau = 0.5$ that it is symmetric about $r_i = 0$ but has a kink at 0. This is the same characteristic that causes problems for the L_1 function. Because of this kink, the derivative of this function is discontinuous at 0.

For further illustration, suppose we are interested in the 3rd quartile or 75th percentile. Then we should set $\tau = 0.75$ as follows:

$$\rho_{\text{RQ}}(r_i; 0.75) = \begin{cases} -(0.25)r_i & \text{if } r_i < 0 \\ 0.75r_i & \text{if } r_i \geq 0. \end{cases} \tag{7.4.4}$$

This produces the asymmetric plot that looks like a *check mark* as indicated for the lines labeled $\tau = 0.75$ in Figure 7.3. This is where the function gets its name; the two lines for $\tau = 0.75$ taken together resemble a check mark so it is called the check function. For other percentiles, deciles or quartiles, it simply requires setting τ to a different value, as shown in Figure 7.3. The cases shown for $\tau = 0.1, 0.25, 0.75$, and 0.9, all exhibit a kink and are asymmetric in one direction or the other. Unfortunately, their derivatives are not continuous either. We saw this earlier in Chapter 2 for the L_1 loss function except in this case it occurs with the check function for every τ.

The derivative of the check function is

$$\psi_{\text{RQ}}(r_i;\tau) = \begin{cases} -(1-\tau) & \text{if } r_i < 0 \\ \tau & \text{if } r_i \geq 0. \end{cases} \tag{7.4.5}$$

If $r_i = 0$, then one side of the equation has a value of τ while the other side has a value of $-(1-\tau)$. Hence, there is a discontinuity at $r_i = 0$. It is clearer if we set $\tau = 0.5$. Then

$$\psi_{\text{RQ}}(r_i;0.5) = \begin{cases} -(0.5) & \text{if } r_i < 0 \\ 0.5 & \text{if } r_i \geq 0. \end{cases} \tag{7.4.6}$$

This discontinuity is also the same problem exhibited by the L_1 loss term, as described in Chapter 2.

As before, the kinks in the check function (one for each τ) imply discontinuous derivatives leading to a number of mathematical and numerical problems when solving for quantiles. When minimizing a loss function constructed from this check function, one must resort to linear programming techniques, such as the simplex-type or interior point methods, to obtain a solution. These methods are beyond the scope of this book but suffice it to say that it can lead to solutions that are non-monotonic. There is also the *possibility of an infinite number of solutions in a bounded region* and therefore the selection of any one solution for a given τ may not be consistent with a neighboring quantile which leads to the crossing problem. Whatever be the reason for the crossing problem, a different approach is required to resolve this troublesome issue.

7.5 Smooth Regression Quantiles (SMRQ)

One way to circumvent the crossing problem is to smooth out the kinks in the check function by using the flexibility of the log-cosh function. To develop the idea further, we use the same sequence of steps as in Section 3.2 of Chapter 3 where the L_1 loss term was smoothed out using log-cosh. However, for quantile regression, we must repeat the procedure for all values of τ. For convenience, consider $\tau \in [0,1]$ in increments of 0.1. The check function $\rho_{\text{RQ}}(x;\tau)$ of Eq. (7.4.2) is plotted as a function of a dummy variable x for each τ value in Figure 7.4 in the top-left panel. The highlighted function is associated with $\tau = 1.0$. The derivative of the check function, $\psi_{\text{RQ}}(x;\tau)$, is given in the lower-left panel based on Eq. (7.4.5). The discontinuous derivative for $\tau = 1.0$ is also highlighted.

To remove the kink and discontinuity, we replace the derivative function with

$$\psi(x;\tau) = \frac{1}{2}\tanh(10x) + \left(\tau - \frac{1}{2}\right) \tag{7.5.1}$$

as shown in the lower-right panel. In the first term of Eq. (7.5.1), the coefficient of $1/2$ is used to match the range of the derivative of the check function. The second term is used to shift the $\frac{1}{2}\tanh(10x)$ function vertically up and down using τ to track the derivative of the check function. The $10x$ term is used temporarily rather than x to make the slope of the function rather sharp but it will be adjusted later when

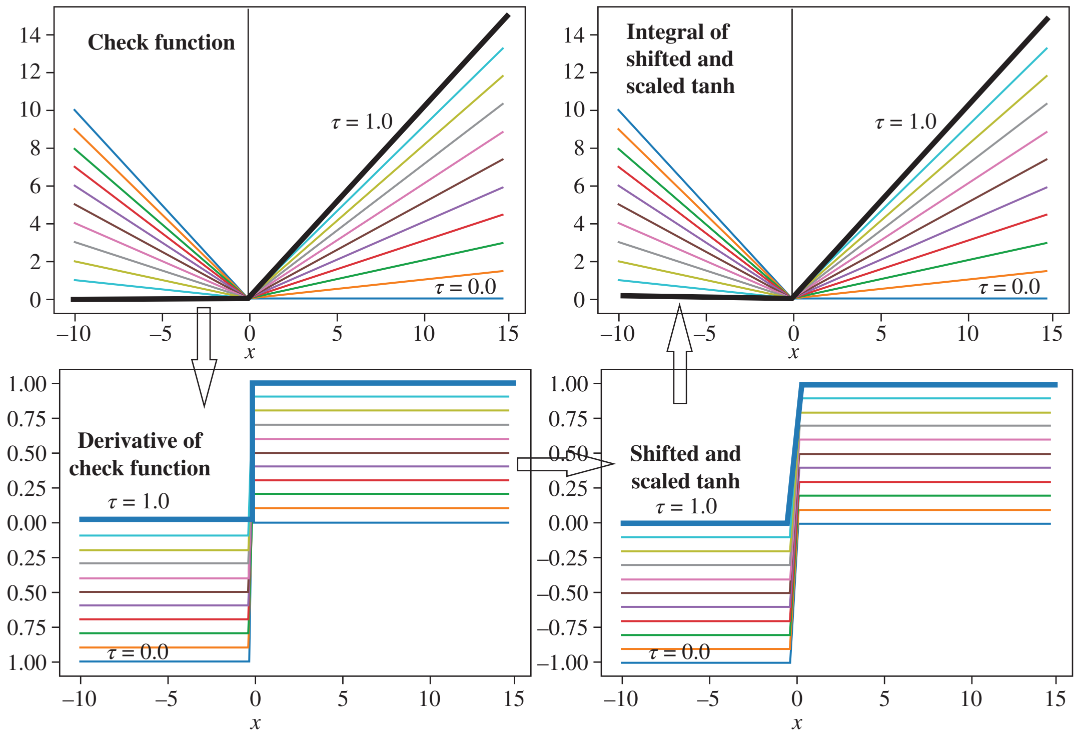


Figure 7.4 Illustrating the steps to go from the standard check function to the smooth check function. In each plot, 11 values of τ are selected from 0.0 to 1.0 in steps of 0.1.

the more general form is introduced. The function for $\tau = 1.0$ in the bottom-right panel is highlighted with a noticeable slope compared to the bottom-left panel.

Exercise: Using Eq. (7.5.1), express this derivative term with $\tau = 1.0$. How much of a shift does this introduce relative to $\tau = 0.5$?

Ans.: Eq. (7.5.1) with $\tau = 1$ is

$$\psi(x; 1.0) = \frac{1}{2}\tanh(10x) + \frac{1}{2}.$$

The result is a vertical shift of 1/2 (relative to $\tau = 0.5$), as seen in the left-bottom corner of Figure 7.4. □

All that remains is to take the integral of this function to obtain a smooth check function which is given by

$$\rho(x; \tau) = \frac{1}{20}\log(\cosh(10x)) + \left(\tau - \frac{1}{2}\right)x. \tag{7.5.2}$$

This smoother function follows the original check function closely but does not have any kinks or discontinuous derivatives, as shown in the top-right panel of Figure 7.4.

We can now reformulate the quantile regression problem using this new loss term. It involves minimizing a new convex loss function which is the conditional quantile function for each τ. By replacing the dummy variable x with r_i, we obtain

$$J(\boldsymbol{\beta}|\tau) = \sum_{i=1}^{n} \rho(r_i;\tau). \tag{7.5.3}$$

This loss function has a unique solution and can be optimized using gradient descent. Unfortunately, the new formulation may still exhibit the crossing problem if the degree of smoothness is not sufficient to eliminate it. We need a more general form of this loss function involving the robustness coefficient a and then we have to carefully select the value of a (rather than the 10 used above) to make the function smooth enough to avoid the crossing problem while delivering the required model accuracy.

The general M-estimator based on log-cosh for quantile regression is given by:

$$\hat{\boldsymbol{\beta}}^{\text{SMRQ}} = \underset{\beta}{\operatorname{argmin}}\, J^{\text{SMRQ}}(\boldsymbol{\beta}|\tau), \tag{7.5.4}$$

where

$$J^{\text{SMRQ}}(\boldsymbol{\beta}|\tau) = \sum_{i=1}^{n} \rho_{\text{SMRQ}}(r_i;\tau), \tag{7.5.5}$$

and the loss term is

$$\rho_{\text{SMRQ}}(r_i;\tau,a) = \frac{1}{2a}\log(\cosh(ar_i)) + \left(\tau - \frac{1}{2}\right) r_i, \tag{7.5.6}$$

where $r_i = y_i - \boldsymbol{x}_i^{\top}\boldsymbol{\beta}^{\text{SMRQ}}$ and a is the robustness coefficient which controls the desired level of curvature of the function (i.e. the smoothness at the kink). This formulation is referred to as the "smooth regression quantiles" (SMRQ). We will use SMRQ to refer to this loss function whereas quantile regression using Eq. (7.4.1) will be referred to simply as "regression quantiles" (RQ). Both of these methods will be compared qualitatively and quantitatively in the sections to follow.

The value of a requires some discussion. For $a = 10$ used above, there is still a sharp corner in the loss term. The value needs to be reduced further within the range of $0.5 < a < 10$ for optimal performance. We recommend the use of $a = 1.5$ for reasons to be elaborated in Section 7.6.2. The resulting loss term is

$$\rho_{\text{SMRQ}}(r_i;\tau) = \frac{1}{3}\log\left(\cosh\left(\frac{3}{2}r_i\right)\right) + \left(\tau - \frac{1}{2}\right) r_i. \tag{7.5.7}$$

This loss term is plotted as a function of r_i for different values of τ in Figure 7.5. At the top, we note that the two cases for $\tau = 0.0$ and $\tau = 1.0$ are mirror images of one another. The same holds true for the middle pair of cases with $\tau = 0.25$ and $\tau = 0.75$. The case for $\tau = 0.5$ is the only symmetric case. The bottom right panel includes all five cases plotted together. Note that the kinks in the standard check function have all been smoothed out. This will eventually lead to quantile estimates that no longer exhibit the crossing problem. The reasons have to do with the strictly convex property of the SMRQ loss function (in contrast to RQ), which produces unique solutions for each τ, and the smoothness of the log-cosh check function, which leads to monotonic quantiles.

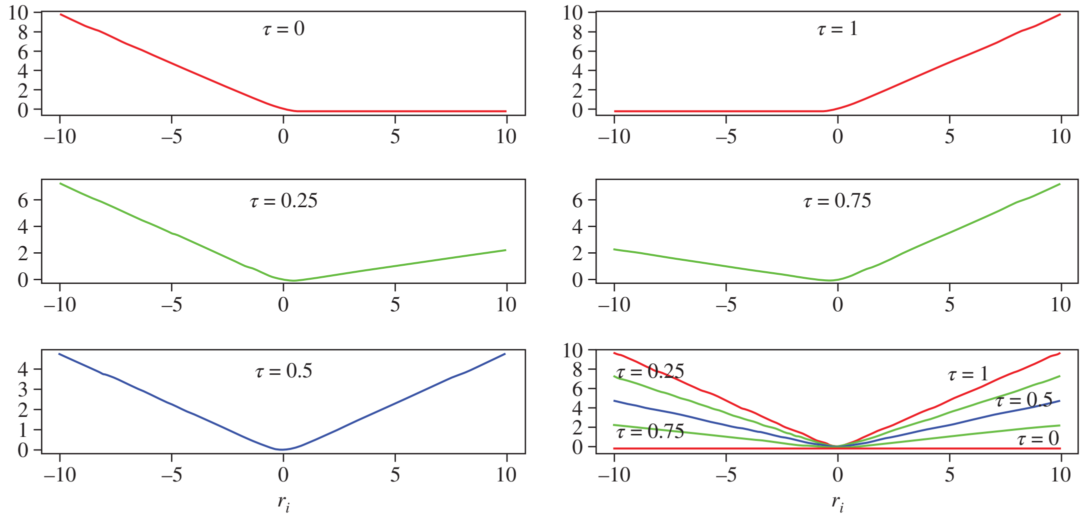


Figure 7.5 SMRQ loss term with $a = 1.5$ for different τ values. Bottom-right plot combines all other plots into one plot.

7.6 Evaluation of Quantile Methods

Consider Figure 7.6 which shows the results of the two different methods for quantile regression. The data and quantile regression lines are shown for $\tau = 0.0, 0.25, 0.5, 0.75$, and 1.0. The question is, which one is better? The two plots look the same visually, but one has multiple crossing problems while the other has none. Unfortunately, it is difficult to determine which option is superior based on a side-by-side

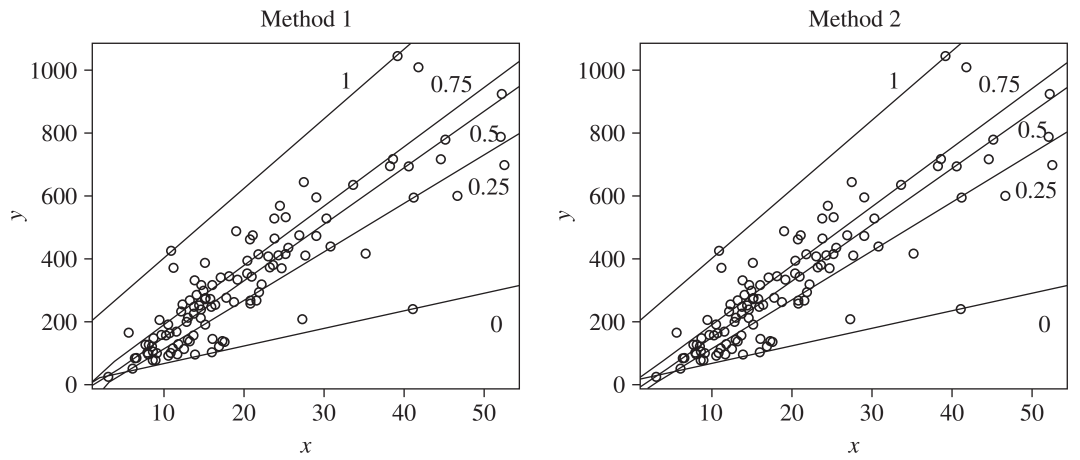


Figure 7.6 Comparing two methods for quantile regression. It is difficult to detect the crossing problem even in the case of one variable using this type of plot.

comparison of this sort. For higher dimensional problems, it is impossible to render the results in this way. We need better qualitative and quantitative measures to compare two or more approaches, especially in higher dimensions.

This section describes ways to properly compare and evaluate different methods for quantile regression. The most important criteria should be related to monotonicity, but others related to the severity of crossing problems are also important. In Section 7.9, the standard errors will be used as another comparison measure. However, if a quantile regression method has crossing problems, then all the other error measures would seem to be less important.

7.6.1 Qualitative Assessment

One way to check for the crossing problem is to count the number of points below a line or hyperplane against the target number implied by the τ parameter. The larger the value of τ, the more points we expect below the line. This way, the overall performance of different methods can be evaluated against the ideal case or by comparing the total number of crossing problems encountered by each method. This method scales easily to multiple dimensions since it reduces to a counting exercise.

For example, the 0th percentile ($\tau = 0$) should have 0 points below the corresponding regression line. As τ increases from 0.0 to 1.0, the number of points should increase monotonically from 0 to n. The 100th percentile ($\tau = 1.0$) should have all n points below the regression line. If these requirements are not satisfied, then the desired monotonicity property of quantiles has been violated. In effect, the quantile method has encountered the crossing problem.

Rather than plotting the data and the regression lines, the procedure is to plot the "points below line" or "points below hyperplane" (as the case may be) against τ. We call this type of plot a quantile-points (QP) plot. Such a plot is reminiscent of the QQ plot of Chapter 4 and can be interpreted in a similar way. Hence, the ideal line is a diagonal line from corner to corner as in a QQ plot. Likewise, we want the QP plot of a quantile regression method to closely follow the ideal one when plotted together.

To illustrate the nature of a QP plot, consider Figure 7.7. In this small dataset with $n = 10$, the x-axis is between 0.0 and 1.0 and the y-axis is between 0 and 10. Ideally, the points should lie directly on the diagonal line, as shown in the left panel, which means that the quantile method is working perfectly. On the right panel, a more realistic QP plot is shown which clearly does not trace a path along the diagonal. Each point on the plot is the result of a quantile regression with a specific value of τ followed by a counting of the points below the line or hyperplane. This provides a qualitative assessment of the associated method for quantile regression. The QP plot can have peaks and valleys as well as flat regions.

We define a "dip" as any point along the QP plot where there is a decrease in value, rather than an increase or a flat region. To understand the definition of a dip, consider the following example. We compute the number of points below the line beginning at the 0th percentile and ending at the 100th percentile, visiting each percentile in between. Assume that, after running standard quantile regression up to the 76th percentile, we obtain the results shown in Table 7.1.

In the table, the first row indicates 35 points below the line. Then, there is a flat region from the 72nd percentile to the 73rd percentile, as the number of points below the line remains at 35. This is not considered problematic. From the 73rd to the 74th percentile, the number of points below the line increases by one, and the same occurs from the 74th to the 75th percentile. This is the expected behavior. However, from the 75th to the 76th percentile, we encounter a dip: the number of points below the line drops from

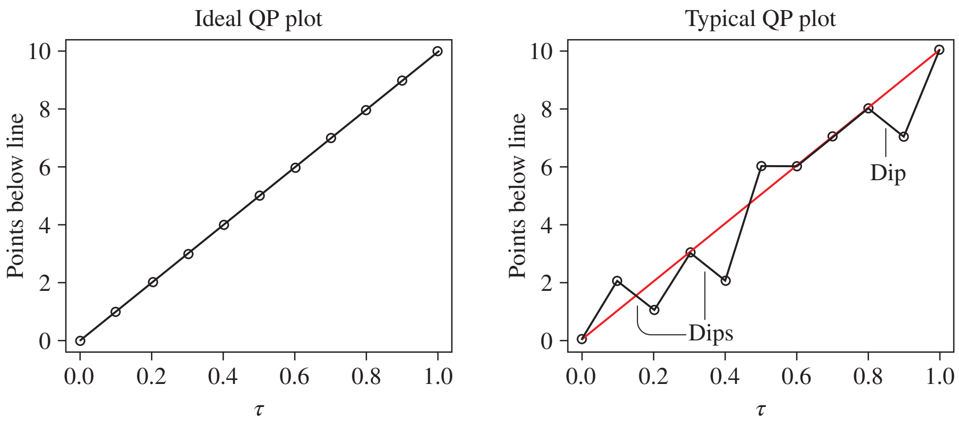


Figure 7.7 Basic concepts of a QP plot. The ideal plot is shown on the left where all points lie along the diagonal. A more realistic plot is shown on the right illustrating several dips which indicate non-monotonic behavior.

Table 7.1 A sequence of quantile regression results.

Percentile	**Points below line**	**QP plot feature**
72nd	35	—
73rd	35	Flat
74th	36	Increase
75th	37	Increase
76th	36	Dip

37 to 36, which is unacceptable. Such a dip indicates a crossing problem, as dips imply non-monotonic behavior in the quantile regression method. Thus, we can immediately determine whether a quantile method is behaving properly if there are no dips.

This methodology can now be applied to Figure 7.6 to determine which is the preferred quantile method. Method 1 is the standard quantile regression (RQ) using Eq. (7.4.2). Method 2 is based on log-cosh quantile regression (SMRQ) using Eq. (7.5.7). To produce the results in Figure 7.8, 100 quantile regressions were performed between 0.0 and 1.0, with steps of 0.01. The number of points below the resulting lines is counted in each case.

Examining the two QP plots, five crossing problems were detected in RQ but not a single one in SMRQ. In the case of RQ, the graph follows the diagonal but there are five dips produced, as indicated by the circles. One of the dips has been enlarged in the inset for clarity, since the resolution of the plot is too low to observe them. In the adjacent panel, the SMRQ plot is given with a similar diagonal characteristic but this time with no dips. However, there are several flat regions, one of which has been enlarged in the inset. These are quite acceptable in quantile regression. Clearly SMRQ performs better than RQ. Furthermore, it illustrates that the use of the log-cosh loss has the ability to remove the crossing problem in quantile regression.

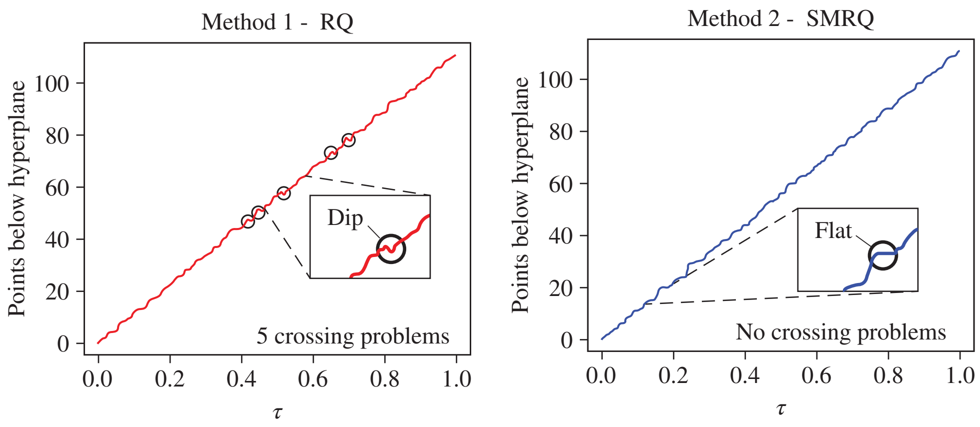


Figure 7.8 Comparison of RQ and SMRQ using data of Figure 7.6.

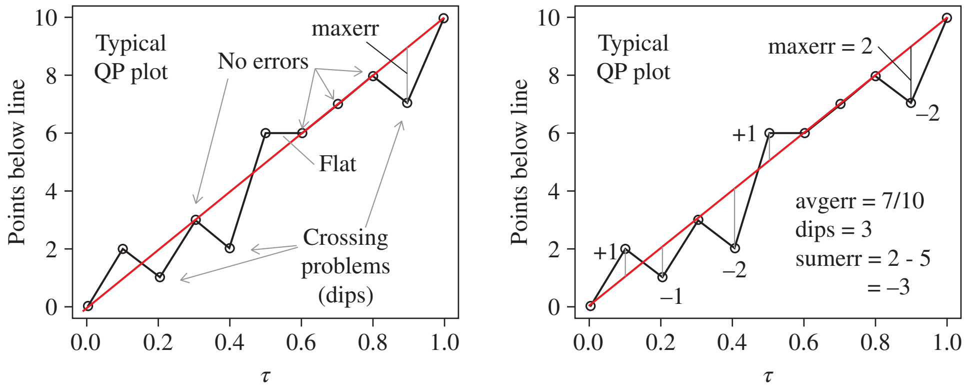


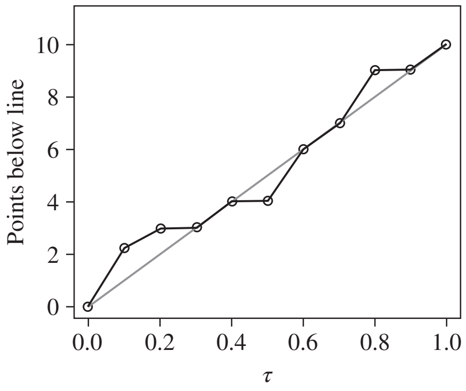
Figure 7.9 Key metrics of a QP plot. Left panel defines *flats*, *dips*, and *maxerr*. Right panel shows computation of *maxerr*, *avgerr*, *sumerr*, and number of dips.

7.6.2 Quantitative Assessment

Now we move to quantitative measures for quantile regression to ensure accuracy as well as monotonicity. The characteristics of a hypothetical QP plot are shown in Figure 7.9 annotated with the features to be measured. If the number of data points at or below the line is exactly the expected number, they will appear as dots on the diagonal (labeled “no errors”). However, if a dot is situated above or below the diagonal line, there is an error in the quantile regression. It may not be a serious error unless it causes a crossing problem. The three points indicated on the left panel (labeled “crossing problem”) arise due to the non-monotonic behavior relative to a neighboring quantile, and each one generates a dip. There may also be “flat” regions, one of which is indicated in the figure.

To quantify the errors, four metrics can be computed: the maximum error (*maxerr*), the average error (*avgerr*), the sum of the errors (*sumerr*), and the number of non-monotonic events (dips). The *maxerr* is the absolute worst-case error in the QP plot which happens to be 2, as shown in the right panel of Figure 7.9. The *avgerr* is the average of the sum of the absolute errors along the diagonal. In this case, the sum is $|1| + |-1| + |-2| + |1| + |-2| = 7$ across 10 data points along the diagonal (excluding the origin). Therefore, $avgerr = 7/10$. The *sumerr* tells us whether most of the errors are positive or negative. Ideally, the *sumerr* should be 0 but in this case, it is $+1 - 1 - 2 + 1 - 2 = -3$. This means that the errors are biased in the negative direction. The number of dips is 3 from a visual examination. There is usually a trade-off between these parameters. For example, when reducing the number of dips to 0, the *maxerr* and *avgerr* may increase, or the *sumerr* may increase in one direction or the other. When comparing two quantile methods quantitatively, these four values can be used to assess their overall performance.

Exercise: For the QP plot below, compute the number of dips, and the *maxerr*, *avgerr*, and *sumerr* quantities.



Ans.: The number of dips is 0 since the graph is monotonically increasing. Then, $maxerr = 1$, $avgerr = 4/10$, and $sumerr = 3 - 1 = 2$. □

As a realistic example, consider the *diabetes* dataset with 442 observations and 10 explanatory variables. The four metrics are given in Figure 7.10. Based on this information, SMRQ (0 dips) is preferable over RQ (10 dips). Furthermore, the other three measures are much better for SMRQ. Note that *sumerr* is particularly high for RQ due to the number and severity of the crossing problems.

Next, RQ and SMRQ are compared on the Swiss Fertility dataset using a QP plot. As mentioned above, we would expect to observe a monotonic trajectory of the graph of the number of points below the hyperplane vs. quantiles. Figure 7.11 shows RQ on the right and SMRQ on the left. Notably, RQ is wildly non-monotonic in its behavior, which is clearly observable, whereas SMRQ is uniformly monotonic. There are numerous instances of the crossing problem in RQ whereas none are detected in SMRQ. Furthermore, all quantitative measures are better in SMRQ.

The interesting thing to notice in SMRQ is that it does not strictly follow the diagonal line but rather traces a stepwise continuous path from the 0th to the 100th percentile. This implies that the quantiles from regression are not perfectly aligned with the true quantiles. Sometimes the graph stray above the

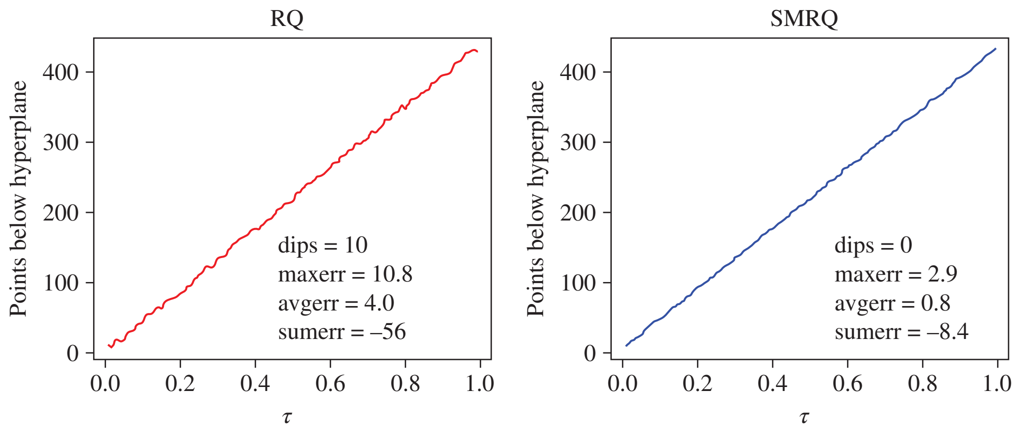


Figure 7.10 Comparison of QP plots of RQ and SMRQ on the diabetes data.

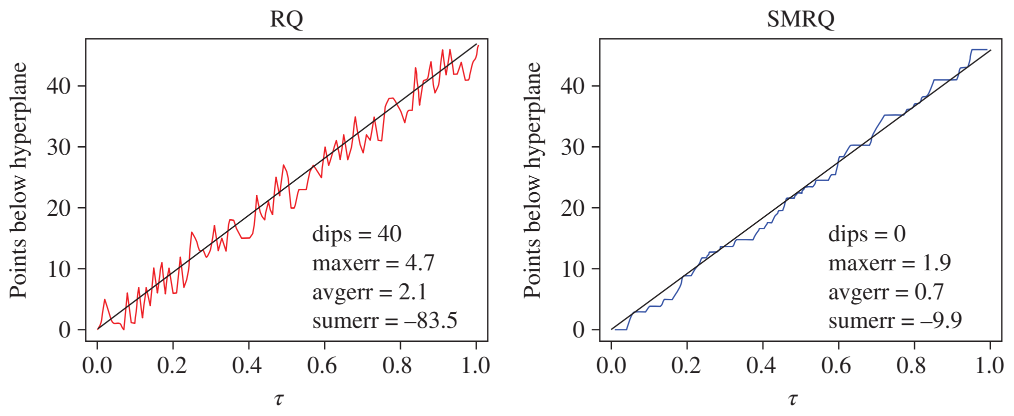


Figure 7.11 Comparison of RQ and SMRQ on Swiss Fertility data.

diagonal and other times below the diagonal. However, for data science purposes, it is sufficient that the quantiles be monotonic as opposed to the non-monotonic behavior exhibited by RQ.

7.7 Selection of Robustness Coefficient

We now address the issue of selecting the robustness coefficient, a, in detail. We saw in Chapter 3 that this factor controls the steepness of the slope of the tanh(ax) function as it crosses the origin and the smoothness around the kink region of the loss term $\frac{1}{a}\log(\cosh(ax))$. As $a \rightarrow 100$, the function approaches the L_1 loss term. The same applies to the quantile loss term for exactly the same reasons. The difference here is that we need to select the robustness coefficient to remove the crossing problem while at the same time keeping the errors as small as possible.

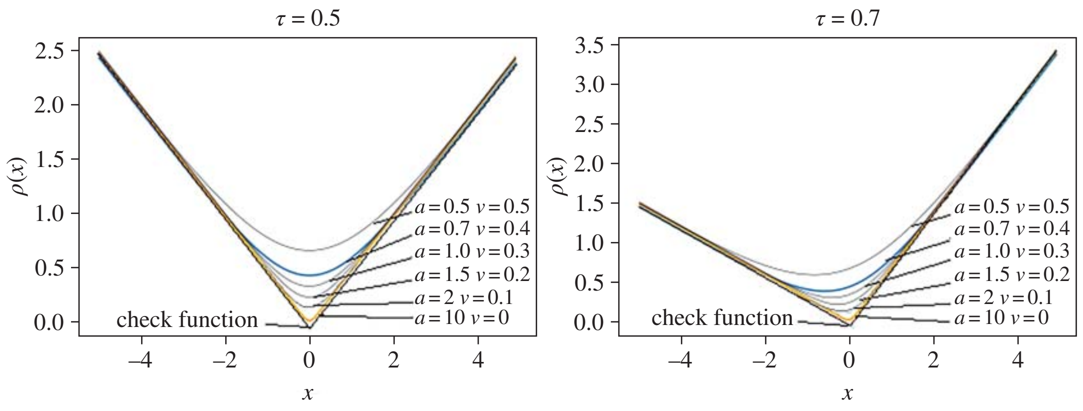


Figure 7.12 Loss functions for SMRQ ($\tau = 0.5$ and $\tau = 0.7$ cases) for different a and v values.

Consider the smooth quantile loss term of the form

$$\rho_{\text{SMRQ}}(x; \tau, a) = \frac{1}{2a}\log(\cosh(ax)) + \left(\tau - \frac{1}{2}\right)x + v. \tag{7.7.1}$$

A new variable v has been introduced for convenience to adjust the vertical position of the loss term to make it align with the corresponding term of the check function (but otherwise has no effect on the optimization since it is a constant). Figure 7.12 shows six different cases of Eq. (7.7.1) by varying parameters a and v for $\tau = 0.5$ and $\tau = 0.7$. One can observe the different levels of smoothness offered by the SMRQ function, which can be varied easily as the need arises. The lower bound of the different cases is the original check function featuring the kink. The next curve above it is associated with $a = 10$ and it follows the check function quite closely. As a decreases in value to $a = 0.5$, the function becomes smoother. The question is, which of the given values of a should be used as a default setting?

Table 7.2 provides an answer. If we run all these cases on the Swiss Fertility dataset and compute the number of dips, *maxerr*, *avgerr*, and *sumerr* from a QP plot, we find that $a = 1.5$ is a reasonable value to use. This is the highest value that has no dips. Above this value, we encounter crossing problems. Below this value, the errors tend to increase. Similar results will be obtained on other datasets. More generally, for a given dataset, the QP plot metrics can be used to obtain a finer resolution of the robustness coefficient, if desired.

Table 7.2 Error metrics as a function of the robustness coefficient values between 0.5 and 10 from Figure 7.12.

	$a = 0.5$	$a = 0.7$	$a = 1.0$	$a = 1.5$	$a = 2$	$a = 10$
dips	0	0	0	0	2	4
maxerr	2.52	2.52	2.05	1.88	1.88	1.87
avgerr	0.89	0.82	0.69	0.67	0.62	0.59
sumerr	−25.9	−27.0	−15.9	−9.9	−2.0	5.0

7.8 Maximum-Likelihood Procedure for SMRQ

Since the SMRQ loss function is a smoother version of the check function, it is possible to use maximum-likelihood estimation (MLE), as described in Chapter 3. To derive the SMRQ loss using the maximum-likelihood procedure, we need a probability density function (PDF), $f_X(x)$. Since we only have a loss function as a starting point, the reverse MLE rule can be used here (see Section 3.5). That is, we know that $\rho(x) = -\log f_X(x)$. Then, it follows from the reverse MLE procedure that

$$f_X(x) = \frac{1}{\kappa} e^{-\rho_{\text{SMRQ}}(x;\tau)}, \tag{7.8.1}$$

where

$$\rho_{\text{SMRQ}}(x;\tau) = \frac{1}{3}\log\left(\cosh\left(\frac{3x}{2}\right)\right) + \left(\tau - \frac{1}{2}\right)x, \tag{7.8.2}$$

using $a = 1.5$ as in Eq. (7.5.7). Applying the procedure, the form of the PDF is given by

$$f_X(x) = \frac{e^{-(\tau-\frac{1}{2})x}}{\kappa \cosh(3x/2)^{\frac{1}{3}}}, \tag{7.8.3}$$

where the normalizing constant κ will vary as a function of the selected τ. For $\tau = 0.5$, we obtain κ as follows

$$\kappa = \int_{-\infty}^{\infty} \frac{e^{-(\frac{1}{2}-\frac{1}{2})x}}{\cosh(3x/2)} = 4.857,$$

and for $\tau = 0.25$, we find κ using

$$\kappa = \int_{-\infty}^{\infty} \frac{e^{-(\frac{1}{4}-\frac{1}{2})x}}{\cosh(3x/2)^{\frac{1}{3}}} = 6.546.$$

For $\tau = 0.75$, we find κ using

$$\kappa = \int_{-\infty}^{\infty} \frac{e^{-(\frac{3}{4}-\frac{1}{2})x}}{\cosh(3x/2)^{\frac{1}{3}}} = 6.546.$$

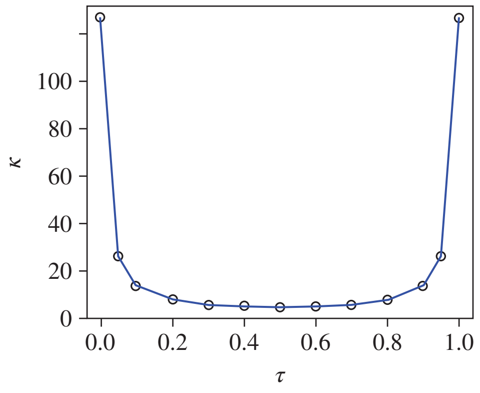


Figure 7.13 A plot of the normalizing constant κ for different τ.

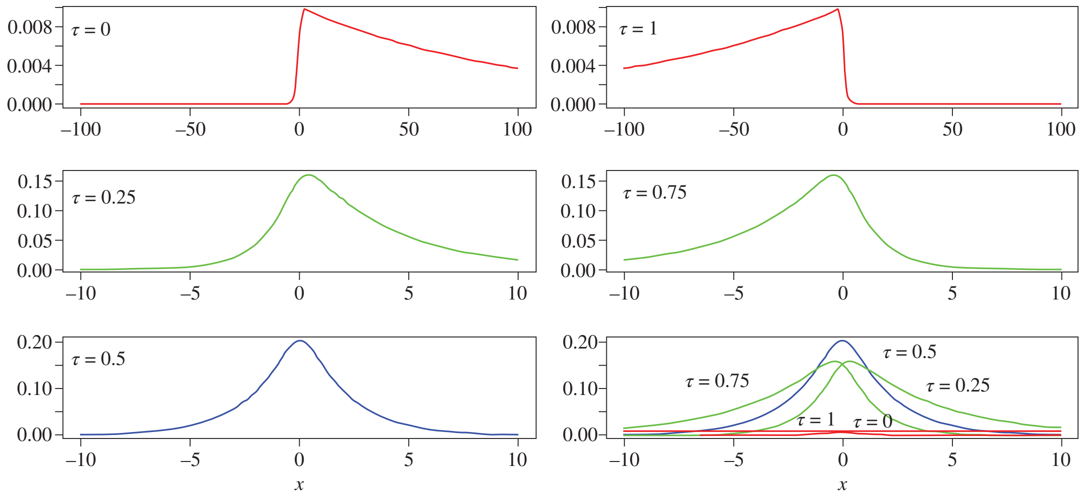


Figure 7.14 PDFs of the quantile function for five different values of τ. The five plots are all combined in the bottom-right panel.

A graph of κ vs. τ is provided in Figure 7.13 using a number of different values of τ. The distributions can now be plotted to reveal their characteristics.

Starting with Eq. (7.8.3) and the normalizing constants κ given in Figure 7.13, different values of τ were selected and their associated distributions plotted in Figure 7.14. The cases shown are for $\tau = 0, 0.25, 0.5, 0.75$, and 1.0. There is a one-to-one correspondence between Figures 7.14 and 7.5. The distributions for $\tau = 0.25$ and $\tau = 0.75$ are mirror images, as are distributions for $\tau = 0.0$ and $\tau = 1.0$. The only symmetric distribution is associated with $\tau = 0.5$.

Since we now have the equations for the PDFs, we can find the expected values and variances. For example, setting $\tau = 0.5$, we find that

$$\mathbb{E}[X] = 0 \tag{7.8.4}$$

and noting that $\mathbb{E}[X^2] = 8.29$ via integration, we have

$$\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = 8.29 - 0^2 = 8.29. \tag{7.8.5}$$

However, for $\tau = 0.75$, we obtain

$$\mathbb{E}[X] = -3.7 \tag{7.8.6}$$

and

$$\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = 34.4 - (-3.7)^2 = 20.7. \tag{7.8.7}$$

All of the distributions encountered in this book thus far have a mean that is 0 but in quantile regression, the associated PDFs do not have a mean of 0 except for $\tau = 0.5$. From the above, the mean of the distribution for $\tau = 0.75$ is negative rather than 0 and for $\tau = 0.25$ it would be positive rather than 0. This can be seen in Figure 7.14. Furthermore, the variance is not constant but increases as τ approaches either 1 or 0.

The MLE can be derived for a given τ using the following steps. First, the likelihood function is

$$L(x_1, \dots, x_n|\tau) = \prod_{i=1}^{n} \left(\frac{e^{-(\tau-\frac{1}{2})x_i}}{\kappa \cosh(3x_i/2)^{\frac{1}{3}}} \right). \tag{7.8.8}$$

Then, the negative log-likelihood is

$$-\ell(x_1, \dots, x_n|\tau) = \sum_{i=1}^{n} \left[\frac{1}{3}\log\left(\cosh\left(\frac{3x_i}{2}\right)\right) + \left(\tau - \frac{1}{2}\right)x_i \right] + n\log(\kappa). \tag{7.8.9}$$

Therefore, after removing the constant term, we obtain the loss function as

$$J^{\text{SMRQ}}(\boldsymbol{\beta}|\tau) = \sum_{i=1}^{n} \left[\frac{1}{3}\log\left(\cosh\left(\frac{3x_i}{2}\right)\right) + \left(\tau - \frac{1}{2}\right)x_i \right]. \tag{7.8.10}$$

The estimates are obtained by selecting τ and then carrying out the optimization as follows:

$$\hat{\boldsymbol{\beta}}^{\text{SMRQ}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{n} \left[\frac{1}{3}\log\left(\cosh\left(\frac{3r_i}{2}\right)\right) + \left(\tau - \frac{1}{2}\right)r_i \right]. \tag{7.8.11}$$

In the above, the dummy variable x_i is replaced with residuals $r_i = y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}^{\text{SMRQ}}$ for $i = 1, \dots, n$.

7.9 Standard Error Computation

Standard errors (s.e.'s) represent the uncertainty around the estimates, as described in Chapter 3. It is essentially the computed value of the standard deviation for each estimate. In the case of quantile regression using log-cosh, the same general formulation of Chapter 3 can be used to obtain the standard errors. However, there will be a different set of s.e.'s for each τ so they must be computed separately for each case. For large n, we treat the estimates as normally distributed so that we can use

$$\text{Var}(\hat{\boldsymbol{\beta}}^{\text{SMRQ}}) = \nu^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}, \tag{7.9.1}$$

and then define the term ν^2 as follows:

$$\nu^2 = s^2 \frac{\mathbb{E}[\psi_{\text{SMRQ}}(z)^2]}{\mathbb{E}[\psi'_{\text{SMRQ}}(z)]^2}, \tag{7.9.2}$$

where $z = (r - \mu)/s$, which has an unbiased estimate that can be obtained by replacing expectations with averages (as described in Chapter 3):

$$\hat{\nu}^2 = \hat{s}^2 \frac{\frac{1}{n}\sum \psi_{\text{SMRQ}}(r_i/\hat{s})^2}{[\frac{1}{n}\sum \psi'_{\text{SMRQ}}(r_i/\hat{s})]^2} \frac{n}{n-d-1}. \tag{7.9.3}$$

In the above, we assume $\tau = 0.5$ with

$$\psi_{\text{SMRQ}}(r_i/\hat{s}) = \frac{1}{2}\tanh\left(\frac{3r_i}{2\hat{s}}\right) + \left(\tau - \frac{1}{2}\right), \tag{7.9.4}$$

and

$$\psi'_{\text{SMRQ}}(r_i/\hat{s}) = \frac{3}{4}\text{sech}^2\left(\frac{3r_i}{2\hat{s}}\right), \tag{7.9.5}$$

Table 7.3 Swiss Fertility estimates and standard errors for RQ and SMRQ quantile regression with $\tau = 0.5$.

Feature	$\hat{\beta}^{\text{RQ}}$	s.e.($\hat{\beta}^{\text{RQ}}$)	$\hat{\beta}^{\text{SMRQ}}$	s.e.($\hat{\beta}^{\text{SMRQ}}$)
(Intercept)	63.49	15.60	62.14	10.41
Agriculture	−0.20	0.10	−0.20	0.07
Examination	−0.46	0.37	−0.27	0.25
Education	−0.79	0.27	−0.88	0.18
Catholic	0.10	0.05	0.10	0.03
Infant mortality	1.46	0.56	1.42	0.37

where the residuals are given by $r_i = y_i - \boldsymbol{x}_i^\top \hat{\boldsymbol{\beta}}^{\text{SMRQ}}$, $i = 1, \ldots, n$. And finally, for the needed $\hat{s}$ and $\hat{s}^2$ terms, we first obtain a robust estimate of s as follows (see Eq. (1.4.6)):

$$\hat{s} = \text{MADN}(\boldsymbol{r}), \tag{7.9.6}$$

and then square it. The s.e.'s are computed as the square root of the diagonal elements of the covariance matrix, $\hat{v}^2(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$.

The estimates and s.e. values for RQ and SMRQ are provided in Table 7.3 using the Swiss Fertility dataset. In this case, the quantile of interest is $\tau = 0.5$. The table indicates that, while the estimates are similar, the standard errors are slightly larger in RQ as compared to SMRQ.

In order to compare both methods over a range of quantiles, a table is not sufficient. Since this dataset has $d = 5$ explanatory variables, it is appropriate to show the aggregate s.e.'s for each τ based on the L_2-norm. The following formula is used:

$$\|\text{s.e.}\|_2 = \sqrt{\text{s.e.}_1^2 + \cdots + \text{s.e.}_d^2}. \tag{7.9.7}$$

Exercise: Compute the L_2-norm of the s.e.'s of the explanatory variables for the Swiss Fertility dataset with $\tau = 0.5$ using Table 7.3.

Ans.: For RQ, $\|\text{s.e.}\|_2 = \sqrt{0.10^2 + 0.37^2 + 0.27^2 + 0.05^2 + 0.56^2} = 0.732$.
For SMRQ, $\|\text{s.e.}\|_2 = \sqrt{0.07^2 + 0.25^2 + 0.18^2 + 0.03^2 + 0.37^2} = 0.487$. □

Figure 7.15 illustrates the relationship between s.e.'s and the quantile parameter τ for the Swiss Fertility dataset. The L_2-norm of the standard errors for SMRQ and RQ is plotted in Figure 7.15. In the case of SMRQ in the top panel, several graphs are shown for different values of the robustness coefficient, a. Each one has a different error trajectory as τ moves from 0 to 1. The graph associated with $a = 1.5$ provides a relatively low standard error for all values of τ. This is yet another reason why $a = 1.5$ is a suitable value for SMRQ. The other graphs begin to stray from their mid-range values as τ approaches either 0 or 1, especially for those with a low value of a.

In the lower panel, the standard error graphs for RQ and SMRQ ($a = 1.5$) are compared. The graph for RQ indicates a higher overall standard error and it is not a smooth function of τ due to the numerous

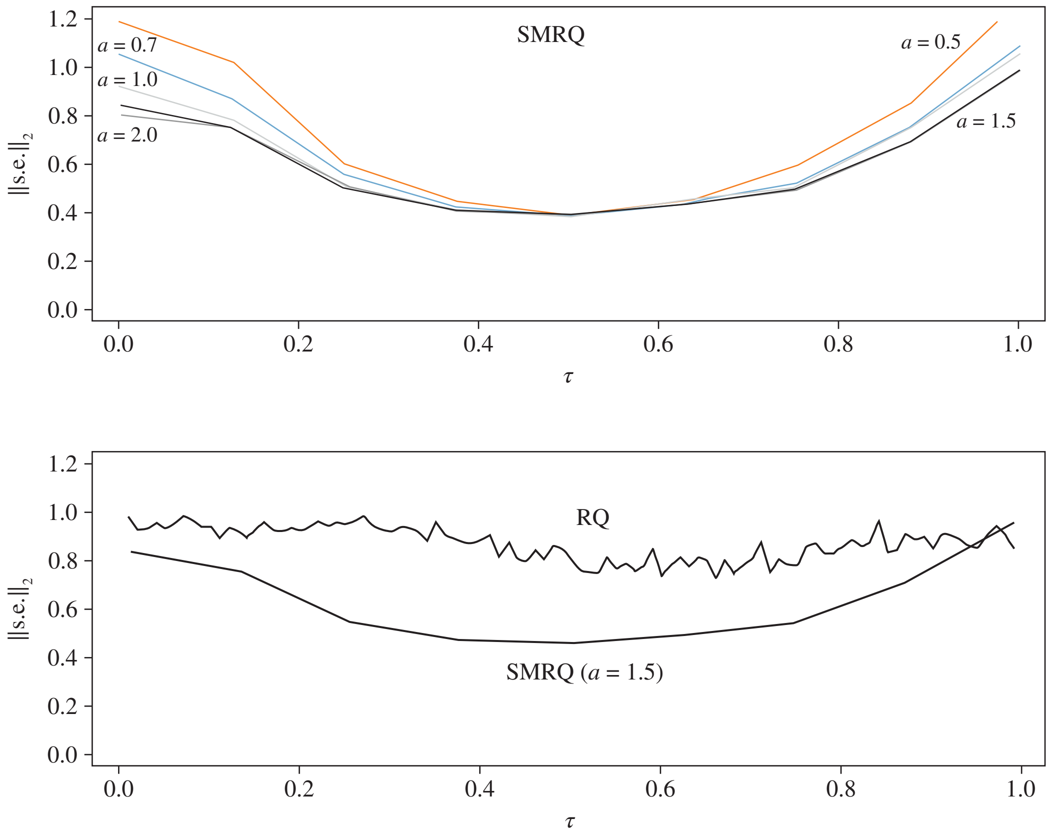


Figure 7.15 Aggregate standard errors for quantile regression on the Swiss Fertility dataset.

crossing problems shown earlier. However, the SMRQ graph is much smoother and features a much lower standard error. Based on these comparisons, and others earlier in the chapter, it is clear that the SMRQ approach to quantile regression offers a significant advantage over the RQ approach which uses the standard check function.

7.10 Summary

This chapter described methods for quantile regression which is a popular machine learning approach in econometrics, survival analysis, and computational biology. It focused on the replacement of the standard check function with a flexible log-cosh loss function as a way to resolve the crossing problem encountered in quantile regression. The basic idea is to smooth out the kinks and remove the dips due to the standard check function by replacing it with a smooth counterpart based on the log-cosh function. By adopting this approach, the crossing problem can be reduced or removed entirely depending on the selection of the robustness coefficient, a. Notably, the loss function can be optimized using gradient descent, which is routinely used in machine learning.

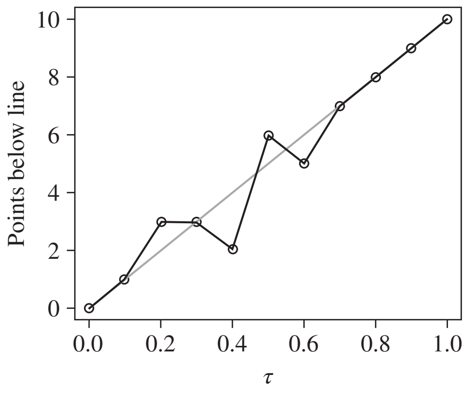
A number of qualitative and quantitative techniques were presented to compare different methods of carrying out quantile regression. The number of non-monotonic events, called dips, can be detected using a QP plot as a measure of the severity of the crossing problem. Other metrics such as the maximum error, average error, and sum of errors can also be computed. These values can be used to select the robustness coefficient suitable for the smooth check function since there is an inherent monotonicity-error trade-off which needs to be balanced.

The use of the log-cosh formulation also leads to a deeper understanding of quantile regression. The source of the crossing problem can be identified as partially due to the kinks in the check function and partially due to infinite solutions possible when solving the standard quantile loss function using linear programming methods. In addition, the maximum-likelihood procedure can be applied to explore the distributions associated with the quantiles in the log-cosh case. Furthermore, the PDFs of different quantiles can be plotted and statistical quantities, such as expectation and variance, can be computed. Indeed, this formulation is a promising approach for use in robust machine learning for quantile regression.

Problems

7.1 Plot the loss terms for L_1, the check function of Eq. (7.4.2) using $\tau = 0.5$ and the log-cosh quantile term of Eq. (7.5.7) using $\tau = 0.5$. Compare and contrast the three cases.

7.2 For the QP plot below, compute the number of dips (non-monotonic events), the maximum error, the average error, and the sum of errors.



7.3 Let $a = 10$ in Eq. (7.5.6). Plot the $\rho_{\text{SMRQ}}(x;\tau)$ loss term for the same τ values as in Figure 7.5. Plot $\psi_{\text{SMRQ}}(x;\tau)$ for the same cases. Compare the results obtained for $a = 10$ with the results for $a = 1.5$.

7.4 Consider the distribution of Eq. (7.8.3) with $\tau = 0.25$. What is the expected value, $\mathbb{E}[X]$, and variance, $\text{Var}(X)$, of this distribution? (Hint: as a short cut, consider Eqs. (7.8.6) and (7.8.7) in conjunction with Figure 7.14 where the PDFs are mirror images of each other.)

7.5 (PROJECT) In this project, the task is to develop a Python program to implement the smooth quantile regression method to avoid the crossing problem.

a) Import the needed libraries.

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
import matplotlib.pyplot as plt
%matplotlib inline
```

b) Load the Swiss dataset (assumed to be stored in the file `swiss.csv` using the data of Chapter 3, Section 3.6.1) and define the $\boldsymbol{X}$ and $\boldsymbol{y}$ values.

```
swiss = pd.read_csv("swiss.csv")
swiss['ones'] = 1
X = np.array(swiss.drop(["Fertility"],axis=1))
y = np.array(swiss["Fertility"])
```

c) Define the needed functions for the quantile log-cosh loss.

```
logcosh = lambda x: np.logaddexp(x,-x)-np.log(2)
def SMRQ(b):
    beta = b
    ri = y-np.matmul(X,beta)
    LCsum = np.sum((1/(2*a))*logcosh(a*ri)+
                   (tau-1/2)*ri)
    return LCsum
```

d) Run the code with $a = 1.5$ and $\tau = 0.5$.

```
a = 1.5
tau = 0.5
#Initialize beta with small random values
initbetaSMRQ = np.random.randn(X.shape[1])/10.
#Run the minimizer to obtain the estimates
res = minimize(SMRQ, initbetaSMRQ, method='BFGS')
betaSMRQ = res.x
formatted_values = [f"{value:.3f}" for value \
    in betaSMRQ]
print("Optimized estimates are:", ', ' \
    .join(formatted_values))
```

e) Re-run the code for $\tau = 0.75$ and then again with $\tau = 0.25$.

7.6 (Project) In this project, the task is to generate the QP plot for the Swiss dataset for $a = 1.5$.

a) Define the code for the QP plot and run it.

```
m = 100
a = 1.5
n = X.shape[0]
taus = np.zeros(m-1)
ptsbelow = np.zeros(m-1)
for i in range(m-1):
    # Set value of tau
    tau = (i+1)/(m)
    taus[i] = tau
    # Initialize betaSMRQ
    initbetaSMRQ = np.random.randn(X.shape[1])
    # Perform SMRQ regression
    res = minimize(SMRQ, initbetaSMRQ, \
        method='BFGS')
    # Save estimates
    betaSMRQ = res.x
    # Determine points below line
    count = np.sum(y < np.matmul(X,betaSMRQ))
    ptsbelow[i] = count
```

b) Generate a QP plot using the results.

```
plt.plot(taus,ptsbelow,color='black')
plt.title("QP plot of Swiss dataset")
plt.xlabel("tau")
plt.ylabel("points below hyperplane")
plt.show()
```

7.7 (Advanced Project) In this project, the task is to write a Python program that takes the information from the previous problem to generate the number of dips, *maxerr*, *avgerr* and *sumerr* of the QP plot. Use the `taus` and `ptsbelow` arrays and compute these quantitative metrics. The results should match Figure 7.11 for SMRQ.

References

Koenker, R. (2005). *Quantile Regression*. Cambridge University Press.
Koenker, R. and Bassett, G. (1978). Regression quantiles. *Econometrica* 46: 33–50.

8

Robust Binary Classification

8.1 Introduction

This chapter addresses machine learning algorithms for binary classification. The fundamental concept involves analyzing a dataset where the combined values of the explanatory variables lead to one of two possible outcomes, which are referred to as classes. The objective is to establish a boundary that separates the two classes. Predictions are made by adding new data points and assessing their positions relative to the boundary to determine class affiliation. While the idea may seem straightforward conceptually, there are many ways to address the problem, and the theory and its practical implementation can be quite complex. However, it becomes interesting at this stage because logistic regression (see James et al. 2023), one of the techniques employed, shares many characteristics with linear regression, which has been thoroughly explored earlier in this book.

We begin this chapter with a detailed treatment of logistic regression which is a well-known approach to binary classification. Then, we modify it to use robust techniques based on the log-cosh loss function, unveiling its untapped potential for data science and machine learning applications. Specifically, we compare the robust log-cosh method with the conventional cross-entropy approach, highlighting their advantages and inherent limitations. This discussion sets the stage for a subsequent exploration of neural networks in Chapter 9, making it crucial to grasp the foundational concepts presented here.

There are several other well-known approaches to binary classification. Some are robust to a certain degree (but not relative to outliers) while others are truly robust (but still require some tuning to obtain acceptable results). Our main interest here is to evaluate their degree of robustness. We first develop the insights that lead to the support vector classifier (SVC). Then we extend the concepts learned to the support vector machine (SVM). SVMs are considered to be among the best large-margin classifiers. This method, developed by the computer science community, is not based on fundamental statistics or any known distributions, but it has emerged as an important classification method. Then, we take a look at the robustness characteristics of k-NN and finish with tree-based methods.

All the traditional methods have been widely studied and documented in the literature. Therefore, we will not go into great depth here. The primary goal of this chapter is to broaden the reader's understanding of the importance and advantages of robust methods in binary classification.

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.

8.2 Binary Classification Problem

Binary classification is a machine learning problem whereby a given set of data falls into one of two possible classes with the goal of finding a suitable decision boundary, either linear or nonlinear, that acts to separate members of one class from the other. As an example, let us consider a bank that is issuing loans and wants to classify each potential customer as either eligible for a loan or not, based on data from existing and previous customers. This is a classic binary classification problem since we want to identify the characteristics of customers who tend to pay back loans as the criteria for giving out future loans. Our target group would be those who received loans and paid them back, labeled as being in Class 1, and the alternate group would be composed of those who received loans and did not pay back, labeled as being in Class 0. The goal is to build a model that predicts whether a new customer belongs to Class 1 or Class 0.

For data science, logistic regression is one of the workhorse methods for binary classification and we will focus most of this chapter on robustifying this approach. One of the main advantages of logistic regression over other methods is that it is possible to compute probabilities of being in one class or the other (if needed) for all points in the dataset, or any new points. Furthermore, it allows us to access feature importance in ways similar to what is done in linear regression. These two aspects of logistic regression make it preferable in many applications in data science.

8.2.1 Why Linear Regression Fails

Without knowing about logistic regression, our first inclination may be to use linear regression to see what happens. Consider the two datasets shown in Table 8.1. In Set 1, the values for x and y are quite suitable for linear regression. The (x_i, y_i) pairs consist of real values which fits the necessary requirements. We can try to fit the data using the robust linear regression of previous chapters, noting here that an outlier is present in the data table (due to data point $(5.4, 9.73)$). However, in Set 2, the response variable is binary; i.e. $y_i \in \{0, 1\}$. Therefore, the rules of the game must change. Although it is possible to attempt some sort of fitting using linear regression, this type of data does not inherently lend itself to this approach. This is why we turn to alternative methods such as logistic regression.

For binary responses, we talk in terms of two classes of observations, Class 0 and Class 1. Each class is typically represented in plots with a different symbol. We are still seeking a line (more generally, a plane or hyperplane) but now *we want the line that divides the data into two classes rather than one that fits the data*. Often, the data is almost separable but not quite. There may also be some responses that are out of place relative to others in the same class, the so-called outliers. Note that most of the other approaches to binary classification do not perform well when outliers are present. We will describe robust methods for binary classification that can handle outliers of the type given in Set 2 (due to data point (1.2, 1)).

Exercise: If a simple linear regression model is used on Set 2 of Table 8.1, we obtain $\hat{\beta}_0 = 0.035$ and $\hat{\beta}_1 = 0.098$. Plot the data and regression line. Can you see any problems using this type of approach for binary classification? If so, list the problems. (Hint: consider the predicted values for each of the given x values.)

Ans.: See Figure 8.1 for the plot. This is not a good approach because the line does not separate the points properly and therefore does not produce a suitable decision boundary. Further, the predicted value at

Table 8.1 Examples of the use of real and binary numbers representing the response data and their implications.

Set												
Set	x	0.1	1.2	2.1	3.3	4.1	5.4	6.2	7.3	8.0	9.2	10.4
1	y	0.44	1.47	2.47	2.59	4.66	9.73	5.81	7.88	7.06	9.21	10.35
Set	x	0.1	1.2	2.1	3.3	4.1	5.4	6.2	7.3	8.0	9.2	10.4
2	y	0	1	0	0	0	0	1	1	1	1	1

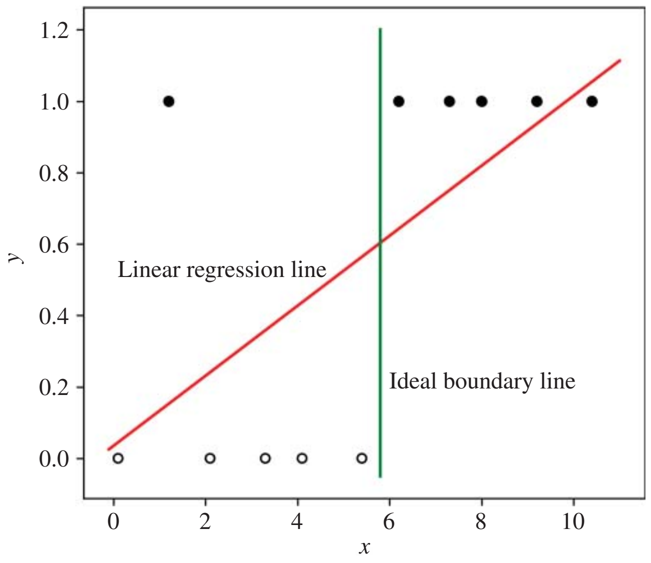


Figure 8.1 Using linear regression for Set 2 of Table 8.1.

$x = 10.4$ exceeds 1.0 which is not desirable. Predictions should be in the range [0,1]. A vertical line at $x = 5.8$ is actually the best decision boundary for this dataset, as demonstrated in the next section. □

8.2.2 Outliers in Binary Classification

Before describing various methods for binary classification, the notion of an outlier in the context of binary classification should be defined. If we return to Table 8.1, we noted earlier that there is an outlier in Set 2. Reviewing the above exercise involving the use of linear regression on Set 2, it seems that a simple horizontal line at $y = 0.5$ would make a suitable boundary between the two classes and this would imply that no outliers exist in the data. However, this is not correct. While it is a two-dimensional problem for linear regression, it is only a one-dimensional problem in binary classification and, in that setting, there is an outlier present in the dataset.

In Figure 8.1, we incorrectly represented the responses in the y-dimension. The problem is actually one-dimensional and the y_i values should be represented as symbols, such as "o" and "•," along the

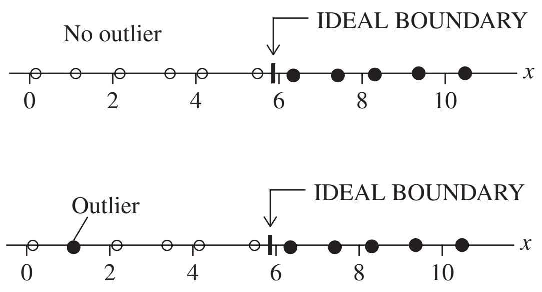


Figure 8.2 One-dimensional binary classification problem without and with an outlier. A method is robust if outliers do not greatly affect the position of the decision boundary.

x-axis. This is done in Figure 8.2. The diagram in the upper half is a clean dataset free of outliers and it is obvious where the ideal boundary point should be located. The diagram in the lower half represents Set 2 of Table 8.1 and we can now see that there is indeed an outlier present. However, this should not change the location of the ideal boundary. If a method is robust, the decision boundary remains the same regardless of the presence of outliers. But this trivial example creates problems for standard logistic regression as it would be influenced by the outlier, causing the boundary to move towards it, due to lack of robustness. Of course, one-dimensional problems are not considered in practice but the same concept applies in higher dimensions.

Now consider a more realistic two-dimensional problem with a linear decision boundary. While most logistic problems have dimensions much higher than two, using two variables is sufficient to gain insight into the way that logistic regression is carried out. In Figure 8.3, we see binary data values associated with y represented with symbols "x" and "•." The two classes appear to be linearly separable on the left panel and the ideal linear decision boundary is shown in the plot. Logistic regression computes the slope and intercept of such a line. Note that the boundary lies between the two classes, at a safe distance from points on either side. This is the highly desirable *large-margin property* that is characteristic of some, but not all, binary classifiers.

On the right-hand panel of Figure 8.3, there is an outlier in the data. Except for this one data point, the same decision boundary could be established to separate the "x" class from the "•" class. But standard logistic regression does not produce this line. Why? Because *an outlier presents an attractive force*. We saw this earlier for linear regression in Chapter 2. The effect of outliers on the least squares method is to change the slope of the line toward the outliers. The same type of effect occurs in standard logistic regression: the decision boundary would be shifted toward the outlier. However, robust logistic regression keeps the same boundary line as if outliers did not exist.

One question to ask is whether this is an error in classification or an outlier in the data. The point on the left side of the boundary in the right-hand panel is *an outlier only in the context of the proposed separation boundary*. We could consider it as an error in classification but that would be incorrect. Instead, it should be treated as an outlier because it lies in the midst of elements from the other class and not in the vicinity of the boundary. *Errors in classification occur near the boundary*, not in the middle of a region dominated by the opposite class. This can be seen in Figure 8.3 for the outlier in the right-hand panel. Its nearest neighbors are all of the opposite type and it is not near the boundary.

An outlier in binary data can be produced in a variety of ways but we can only find it once we establish a decision boundary. Only then is it clear which points are inliers and which ones are outliers. Nonlinear boundaries may also change our view of outliers and inliers. Whatever be the designation, we would like

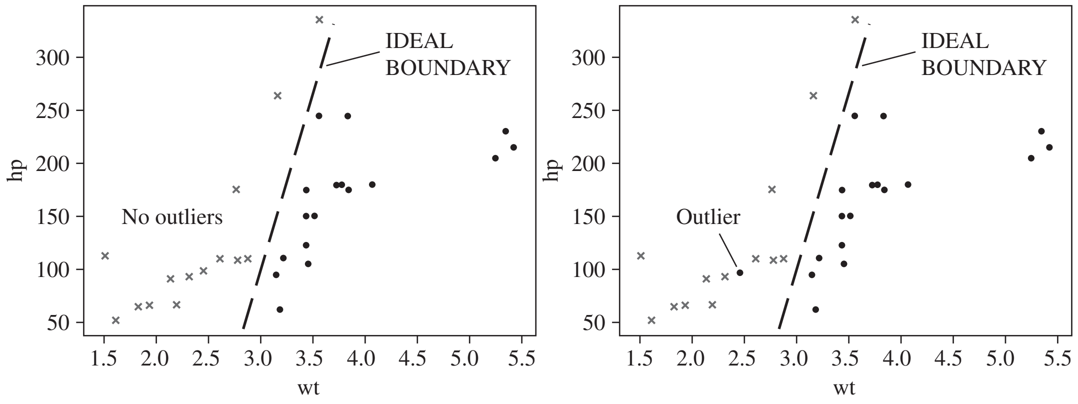


Figure 8.3 Outlier definition and robustness in a two-dimensional setting.

to have a method that is robust enough to produce a good model in the presence of a small percentage of such outliers. In fact, the word *robust* implies that a certain amount of tolerance to outliers, meaning that outliers will not greatly affect the estimation of the true underlying model. Hence, the model building process should withstand a small number of these outliers and produce results as if they did not exist, as shown in the right panel of Figure 8.3.

Unfortunately, standard cross-entropy methods are strongly affected by outliers. They act like an attractive force that moves the decision boundary in their direction. In essence, these methods are based on the mean of the residuals, much like least squares for linear regression. In the robust approach, the median is used to build the model. Note that it does not have to be precisely the median. Any quantile in the vicinity of the median effectively ignores outliers.

Non-robust methods currently in use may produce misleading or erroneous results if there are outliers present in the data, and this fact would be hidden from the user. Oftentimes, the accuracy of predictions on a training set may reach 100%, which gives the user confidence in the results. Unfortunately, overfitting has likely occurred. This means that the outliers were perfectly captured by the parameters which can occur if the feature space is large enough. The effect of outliers will be felt only when the accuracy on the test set is significantly lower than expected. With this background, the details of the standard and robust methods will now be described.

8.3 The Cross-Entropy (CE) Loss

The concept of "cross-entropy" originated in information theory as a way of quantifying the difference between two distributions. It is now used widely in machine learning in the form of a loss function to compare a known distribution of the true labels $\boldsymbol{y}$ against an estimated distribution $\boldsymbol{p}$ over the same set of observations. Following the procedure similar to that of Chapter 2 on linear regression, we will first derive this loss function which is used in logistic regression. We will eventually end up with the cross-entropy loss function and an equation to compute $\boldsymbol{p}$ so we might as well get a glimpse of these equations before setting off to derive them.

The CE loss is given by

$$\text{CE loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log p_i + (1-y_i)\log(1-p_i)], \tag{8.3.1}$$

where the y_i's are the true responses and the p_i's are the predicted values, for $i = 1, \dots, n$. It is also referred to as the *log loss* function.

This loss function does not have the same form as any of the loss functions we have encountered thus far, but remember that we are now doing logistic regression, not linear regression. The term p_i is a probability that depends on the explanatory variables $\boldsymbol{x}_i$ and the model parameters $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$ in the following way:

$$p_i = \frac{1}{1+\exp(-\boldsymbol{x}_i^\top \boldsymbol{\beta})}. \tag{8.3.2}$$

Rather than predicting 0 or 1, the goal of logistic regression is to predict the probability of a response of 1 for each observation using this **logistic function**. If the probability is high, the predicted response is considered to be 1, whereas if the probability is low, it is considered to be 0. Because of its importance, we will also derive this equation, but consider it to be known for the moment. The estimate of $\boldsymbol{\beta}$ associated with the minimum of the CE loss function defines a point, line, plane, or hyperplane (depending on the dimensions of $\boldsymbol{\beta}$) that forms the decision boundary.

Note: While the goal of the next section is to derive the cross-entropy loss from first principles in some easy to understand form, it may require several passes to fully grasp it. This is an important derivation to follow carefully. One way to deal with the derivation in d-dimensional space is to imagine that we are trying to find a linear boundary in a two-dimensional space. This may be helpful in keeping things clear in your mind. The section may also be skipped without loss of continuity.

8.3.1 Deriving the Cross-Entropy Loss

To set up the derivation, we need to specify the distribution associated with the response variable Y as a starting point. Since we can only have a response of 1 or 0, the random variable Y follows a Bernoulli distribution with some unknown probability, p. That is, $Y \sim \text{Ber}(p)$. The probability mass function (PMF) of the Bernoulli distribution is shown in Figure 8.4. This discrete distribution describes our view of $Y \in \{0, 1\}$, where the probabilities of being 0 or 1 are $1-p$ and p, respectively. This particular distribution can be written in an elegant form as follows:

$$f_p(y) = p^y(1-p)^{(1-y)}. \tag{8.3.3}$$

To check if this is the correct formulation, we note that if $y = 1$, then $f_p(y) = p$, and if $y = 0$, we have that $f_p(y) = 1 - p$. So it is indeed the Bernoulli distribution.

To derive the CE loss function, suppose for the moment that we have access to the (y_i, p_i) pairs. Since we know the distribution is Bernoulli, we can use the maximum likelihood procedure, as outlined in Chapter 2. Step one is to define

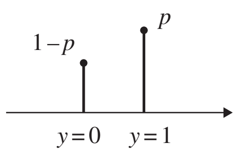


Figure 8.4 Bernoulli distribution of random variable Y.

the likelihood function,

$$L(y_1, \dots, y_n; p_1, \dots, p_n) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{(1-y_i)}. \tag{8.3.4}$$

Then, the negative log-likelihood expression is given by:

$$-\ell(y_1, y_2, \dots, y_n; p_1, \dots, p_n) = -\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]. \tag{8.3.5}$$

This is the **cross-entropy loss function** and is expressed as follows:

$$\text{CE loss} = -\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]. \tag{8.3.6}$$

The equation computes the total cross-entropy between y_i and p_i over n observations. During the optimization process, the loss function of Eq. (8.3.6) will do its best to match each p_i to each y_i because when they match, the loss function is identically 0. It is easy to check this condition for the case of, say $y_1 = 1$, $p_1 = 1$ or $y_1 = 0, p_1 = 0$. In either case, the loss is 0. Of course, in practice, we do not reach 0, but hope to obtain some minimum loss value. At the minimum value, we find the corresponding $\hat{\boldsymbol{\beta}}^{\text{CE}}$ that becomes our model.

To summarize, the average cross-entropy loss is

$$\text{CE loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)] \tag{8.3.7}$$

and the CE estimator is given by

$$\hat{\boldsymbol{\beta}}^{\text{CE}} = \underset{\beta \in \mathbb{R}^{d+1}}{\text{argmin}} \left\{ -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)] \right\}. \tag{8.3.8}$$

It can be shown that the CE loss function is convex which is a highly desirable property for convergence to a global minimum. Therefore, the way we obtain $\hat{\boldsymbol{\beta}}^{\text{CE}}$ is the same as in Chapter 2: we use gradient descent in one of its advanced forms so that convergence will occur smoothly and in a timely manner. The estimates are used to establish a decision boundary between the two classes and new data can be classified using the boundary line or plane. Essentially, logistic regression involves minimizing the CE loss function to obtain estimates that define a decision boundary between two classes.

Now that the loss function is derived and understood, the logistic function will be derived based on the *generalized linear model* (GLM) formulation. The procedure involves expressing the conditional probability that $Y_i = y_i$ given p_i in an exponential form and identifying the canonical link function associated with it. We again start with the Bernoulli distribution as follows:

$$\text{P}(Y_i = y_i | p_i) = p_i^{y_i}(1-p_i)^{(1-y_i)}. \tag{8.3.9}$$

Next, we express it in the form of an *exponential family* by taking the log and then the exponential as follows[1]:

$$\text{P}(Y_i = y_i | p_i) = \exp(y_i \log p_i + (1-y_i)\log(1-p_i)). \tag{8.3.10}$$

1 Here, exp() is the exponential function expressed this way to improve readability.

Collecting all terms with y_i (i.e. taking y_i as a common factor from both terms) and then rearranging in the GLM canonical form, we obtain

$$P(Y_i = y_i|p_i) = \exp\left(y_i \log \frac{p_i}{1-p_i} + \log(1-p_i)\right). \tag{8.3.11}$$

The term next to y_i in this formulation is called the link function. We take that term and postulate that it is representable by a linear model as follows:

$$\log \frac{p_i}{1-p_i} = \beta_0 + x_{i1}\beta_1 + \cdots + x_{id}\beta_d. \tag{8.3.12}$$

The right-hand side is the linear model we seek. The left-hand side is referred to as the **logit link function**. This form is particularly useful since it will eventually lead to a convex loss function. Rearranging Eq. (8.3.12) and solving for p_i, we obtain the logistic function,

$$p_i = \frac{1}{1+\exp(-\boldsymbol{x}_i^\top \boldsymbol{\beta})}. \tag{8.3.13}$$

This is same the equation we used earlier but now we have derived it from first principles so we can proceed from here using the maximum-likelihood procedure outlined before to obtain the CE loss function.

8.3.2 Understanding Logistic Regression

Now, suppose we find the estimates $\hat{\boldsymbol{\beta}}$ using the estimator described in Eq. (8.3.8), or any other similar classifier. What is the meaning of the estimates and what can we do with them? We know the estimates establish the decision boundary, but since we cannot plot such a boundary in high dimensions, how do we use them? The answer is that we can use a **probability test** for some new data point $\boldsymbol{x}$ to decide if it should be categorized as being in Class 0 or Class 1. A second option is to use a **sign test** on $\boldsymbol{x}^\top \hat{\boldsymbol{\beta}}$ to determine if it belongs to Class 1 or Class 0.

Conceptually, any point that satisfies $\boldsymbol{x}^\top \hat{\boldsymbol{\beta}} = 0$ represents a point on the boundary line or hyperplane with a 50% probability of belonging to Class 0 or Class 1. In Eq. (8.3.13), $p_i = 0.5$ when $\boldsymbol{x}^\top \hat{\boldsymbol{\beta}} = 0$. Thus, the linear boundary in d dimensions (i.e. the hyperplane boundary) is defined where

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_d x_d = 0. \tag{8.3.14}$$

Note that $\hat{\beta}_0$ serves as the intercept while the rest of the coefficients, $\hat{\beta}_1, \ldots, \hat{\beta}_d$, establish the angle of the hyperplane in each dimension.

Using a sign test, we can tell if $\boldsymbol{x}$ falls on one side of the boundary or the other. Therefore, we would predict Class 1 if

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_d x_d > 0, \tag{8.3.15}$$

and Class 0 if

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_d x_d < 0. \tag{8.3.16}$$

Example: Consider the one-dimensional problem back in Figure 8.2. In that case, we have two parameters to estimate, β_0 and β_1 (even though it seems like one parameter would be enough). Some classifier has produced estimates of $\hat{\beta}_0 = 58$ and $\hat{\beta}_1 = -10$ for this problem. With this information, we can establish the boundary point. It occurs where $\boldsymbol{x}^{\top}\hat{\boldsymbol{\beta}} = 0$ which means that $\hat{\beta}_0 + \hat{\beta}_1 x = 0$, i.e. $x = -\hat{\beta}_0/\hat{\beta}_1 = 5.8$ as shown in the figure. For some new data $x = 2$, since $\hat{\beta}_0 + \hat{\beta}_1 x = 58 + (-10)(2) > 0$, then x is considered to be in Class 1; similarly, if $x = 7$, then $\hat{\beta}_0 + \hat{\beta}_1 x = 58 + (-10)(7) < 0$ and therefore x is assigned to Class 0. □

Example: Consider the two-dimensional problem back in Figure 8.3. A classifier has produced estimates of $\hat{\beta}_0 = 10$, $\hat{\beta}_1 = -4$, and $\hat{\beta}_2 = 0.02$ for this problem. The decision boundary occurs where $10 + (-4)x_1 + 0.02x_2 = 0$. It corresponds to the line $x_2 = -\hat{\beta}_0/\hat{\beta}_2 - \hat{\beta}_1/\hat{\beta}_2 x_1 = -500 + 200x_1$. Therefore, for some new data point at $(4, 100)$, we find that $10 + (-4)(4) + 0.02(100) < 0$ and it is considered to be in Class 0. Likewise, a new point at $(2, 300)$ is considered to be in Class 1 since $10 + (-4)(2) + 0.02(300) > 0$. □

A graphical view of the two ways to use $\hat{\boldsymbol{\beta}}$ is provided in Figure 8.5. Assume that $\boldsymbol{x}_i$ is to be assigned to a specific class. Starting with Figure 8.5(a), if $\boldsymbol{x}_i^{\top}\hat{\boldsymbol{\beta}} > 0$, we declare the data point as belonging to Class 1, which is on one side of the boundary. If $\boldsymbol{x}_i^{\top}\hat{\boldsymbol{\beta}} < 0$, it belongs to Class 0, which is on the other side of the boundary. Alternatively, we can use a probability test by computing $p_i = \mathrm{P}(Y_i = 1|\boldsymbol{x}_i)$ using

$$p_i = p(z_i) = \frac{1}{1 + e^{-z_i}}, \tag{8.3.17}$$

where $z_i = \boldsymbol{x}_i^{\top}\hat{\boldsymbol{\beta}}$. The way to determine the class of $\boldsymbol{x}_i$ is depicted in Figure 8.5(b). On one side of the plot is the region where $p_i < 0.5$, which is Class 0, and on the other side is where $p_i > 0.5$, which is Class 1. When $z_i = 0$, the probability is 0.5. So the boundary line is indeed where $p_i = 0.5$. This is a key point to understand about the decision boundary. It is defined as the linear or nonlinear boundary where the probability is 0.5.

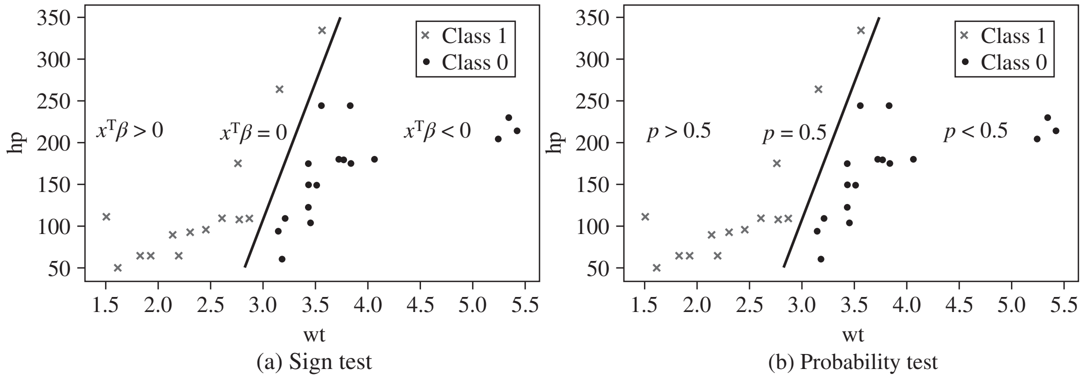


Figure 8.5 Two ways to interpret the results of logistic regression either using (a) the sign test or (b) the probability test.

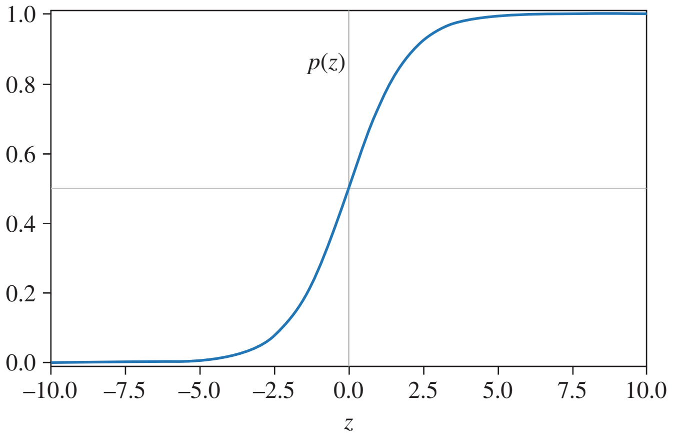


Figure 8.6 Logistic function of Eq. (8.3.17).

It is instructive to examine the logistic function with the computation of these probabilities in mind. A graph of the logistic function $p(z)$ is shown in Figure 8.6. Note that for large positive values of z, the probability asymptotically approaches 1, whereas for large negative values, it asymptotically approaches 0. The ground truth (i.e. true value) of the dependent variable y is either 0 or 1 whereas the predicted probability $p(z)$ lies between 0 and 1.

Exercise: Let the estimates $\hat{\beta}_0 = -1.0$, $\hat{\beta}_1 = -1.0$, $\hat{\beta}_2 = -1.0$, and $\hat{\beta}_3 = -1.0$ be derived from a training set for a logistic regression problem where the possible logical response is either 0 or 1. Using the sign test, what is the predicted logical value (0 or 1) of the response for a new point $\boldsymbol{x}_1 = (1.0, 2.0, 3.0)^\top$? What about $\boldsymbol{x}_2 = (-1.0, -2.0, -3.0)^\top$?

Ans.: For $\boldsymbol{x}_1 = (1.0, 2.0, 3.0)^\top$, we have that $-1.0 - 1.0(1.0) - 1.0(2.0) - 1.0(3.0) = -7.0 < 0$. Therefore, this element belongs to Class 0. In the case of $\boldsymbol{x}_2 = (-1.0, -2.0, -3.0)^\top$, we have that $-1.0 - 1.0(-1.0) - 1.0(-2.0) - 1.0(-3.0) = 5.0 > 0$. Therefore, this element belongs to Class 1. □

Exercise: Given $\hat{\beta}_0 = -1.0$, $\hat{\beta}_1 = -1.0$, $\hat{\beta}_2 = -1.0$, and $\hat{\beta}_3 = -1.0$, find a vector $\boldsymbol{x}_3$ such that the probability of being in Class 0 or Class 1 is 0.5. Note that for this case, there is no unique solution for $\boldsymbol{x}_3$ since it can lie anywhere on the decision boundary.

Ans.: $\boldsymbol{x}_3 = (2.0, 0.0, -3.0)^\top$ would be one of many solutions since $-1.0 - 1.0(2.0) - 1.0(0.0) - 1.0(-3.0) = 0.0$ and $p = 1/(1 + e^0) = 0.5$. □

Exercise: Given $\hat{\beta}_0 = -1.0$, $\hat{\beta}_1 = -1.0$, $\hat{\beta}_2 = -1.0$, and $\hat{\beta}_3 = -1.0$, compute the probability $\mathrm{P}(Y_i = 1|\boldsymbol{x}_i)$ for three different cases, $\boldsymbol{x}_1 = (1.0, 2.0, 3.0)^\top$, $\boldsymbol{x}_2 = (-1.0, -2.0, -3.0)^\top$, and $\boldsymbol{x}_3 = (2.0, 0.0, -3.0)^\top$ using the logistic function.

Ans.: $p_1 = 1/(1 + e^7) \approx 0.0$. $p_2 = 1/(1 + e^{-5}) \approx 1.0$. $p_3 = 1/(1 + e^0) = 0.5$. □

8.3.3 Gradient Descent

The numerical optimization technique used to find $\hat{\boldsymbol{\beta}}$ is usually some form of gradient descent. The objective function to be minimized in logistic regression with cross-entropy, written as $J^{\mathrm{CE}}(\boldsymbol{\beta})$ to emphasize its dependence on $\boldsymbol{\beta}$ through p_i, is given by

$$J^{\mathrm{CE}}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}[y_i \log p_i + (1-y_i)\log(1-p_i)]. \tag{8.3.18}$$

We will describe only the simplest form of gradient descent for our purposes here. Basically, we compute the loss function for an initial guess, $\boldsymbol{\beta}^{(0)}$, and then take a step in the opposite direction of the gradient to reduce the value of $J^{\mathrm{CE}}(\boldsymbol{\beta})$. The process is repeated until the minimum of $J^{\mathrm{CE}}(\boldsymbol{\beta})$ is found. This iterative method requires partial derivatives of $J^{\mathrm{CE}}(\boldsymbol{\beta})$ in Eq. (8.3.18) with respect to each of the parameters in the vector $\boldsymbol{\beta}$.

We begin with the case involving only one observation, $\boldsymbol{x}$, and one known binary response, y, and a predicted value, p. Starting with the loss function, $J^{\mathrm{CE}}(\boldsymbol{\beta})$, we can apply the chain rule to obtain the needed gradients. For some parameter, β_j, we have that

$$\frac{\partial J^{\mathrm{CE}}(\boldsymbol{\beta})}{\partial \beta_j} = \frac{\partial J^{\mathrm{CE}}(p)}{\partial p}\frac{\partial p}{\partial \beta_j}. \tag{8.3.19}$$

Then, the partial derivative in the first term of Eq. (8.3.19) is

$$\frac{\partial J^{\mathrm{CE}}(p)}{\partial p} = -\frac{y}{p} + \frac{1-y}{1-p}, \tag{8.3.20}$$

and the second term of Eq. (8.3.19), which involves the partial derivative of the logistic function, is

$$\frac{\partial p}{\partial \beta_j} = \frac{\exp(-\boldsymbol{x}^{\top}\boldsymbol{\beta})}{(1+\exp(-\boldsymbol{x}^{\top}\boldsymbol{\beta}))^2}x_j = p(1-p)x_j. \tag{8.3.21}$$

Combining the two and simplifying, we obtain

$$\frac{\partial J^{\mathrm{CE}}(\boldsymbol{\beta})}{\partial \beta_j} = -(y-p)x_j = (y-p)(-x_j). \tag{8.3.22}$$

Next, consider the case with n observations. We have to sum up such quantities in Eq. (8.3.19) associated with each observation for every parameter. Using the steps above, an update equation for each β_j, $j = 0, \dots, d$, to compute the next iteration value is given by

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \ell \times \frac{\partial J^{\mathrm{CE}}(\boldsymbol{\beta})}{\partial \beta_j} = \beta_j^{(t)} - \ell\sum_{i=1}^{n}(y_i - p_i)(-x_{ij}), \tag{8.3.23}$$

where i is the row index of the dataset and ℓ is a tuning parameter called the **learning rate** that controls the step size needed in each iteration to produce a smooth yet rapid convergence to the global solution.

8.4 The Log-Cosh (LC) Loss Function

In this section, we explore log-cosh logistic regression (LCLR) and compare it against the aforementioned cross-entropy approach. We motivate the use of log-cosh because it possesses some desirable properties in terms of robustness to outliers. For our purposes here, robust estimation involves the replacement of the cross-entropy loss function with the log-cosh loss function. This loss function has been implemented in `keras` and `tensorflow`, which are open-source machine learning platforms (see Chapter 12). Therefore, it is worthwhile to study the log-cosh approach in detail.

For the robust procedure, we start with the distribution of residuals of the form $r_i = y_i - p_i$. We postulate that the residuals are distributed according to the Cosh distribution and then use the maximum-likelihood procedure (see Chapter 3). The likelihood function based on the Cosh distribution is given by:

$$L(r_1, r_2, \dots, r_n; \beta) = \prod_{i=1}^{n} \frac{1}{\pi \cosh(r_i)}. \tag{8.4.1}$$

The negative log-likelihood expression is given by:

$$-\ell(r_1, r_2, \dots, r_n; \beta) = n \log \pi + \sum_{i=1}^{n} \log(\cosh(r_i)). \tag{8.4.2}$$

The first term is constant for a given n and ignored during minimization. This gives rise to the log-cosh loss function in terms of residuals,

$$\text{LC loss} = \sum_{i=1}^{n} \rho_{\text{LC}}(r_i) = \sum_{i=1}^{n} \log(\cosh(y_i - p_i)). \tag{8.4.3}$$

Following the steps outlined earlier for gradient descent, we obtain

$$\frac{\partial J^{\text{LC}}(\boldsymbol{\beta})}{\partial \beta_j} = \sum_{i=1}^{n} p_i(1 - p_i) \tanh(y_i - p_i)(-x_{ij}). \tag{8.4.4}$$

The update equation for each β_j to compute the new parameter values is given by

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \ell \times \sum_{i=1}^{n} p_i(1 - p_i) \tanh(y_i - p_i)(-x_{ij}), \tag{8.4.5}$$

where ℓ is the learning rate that controls the step size needed in each iteration, t, to produce a smooth yet rapid convergence to the optimum solution.

To compare CE and LC losses in a consistent framework, we start the loss function for cross-entropy as

$$\text{CE loss} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \tag{8.4.6}$$

and somehow introduce r_i into the equation to get it into the M-estimation form (refer to Chapter 2). First, we recognize that $r_i = y_i - p_i$. Therefore, $p_i = y_i - r_i$. Next, we note that when $y_i = 1$, only the first term of Eq. (8.4.6) contributes to the loss. And when $y_i = 0$, only the second term of Eq. (8.4.6) contributes to the loss. Therefore, we can redefine the CE loss as

$$\text{CE loss} = \sum_{i=1}^{n} \rho_{\text{CE}}(r_i), \tag{8.4.7}$$

where

$$\rho_{\text{CE}}(r_i) = \begin{cases} -\log(1 - r_i) & \text{if } y = 1 \ (\text{CE}_1) \\ -\log(1 + r_i) & \text{if } y = 0 \ (\text{CE}_0) \end{cases}. \tag{8.4.8}$$

If $\rho_{\text{CE}}(r_i)$ and $\rho_{\text{LC}}(r_i)$ are plotted on the same graph, we obtain the results given in Figure 8.7. Note that the residuals can only be in the range $[-1, 1]$ because of the nature of the logistic response variable p and

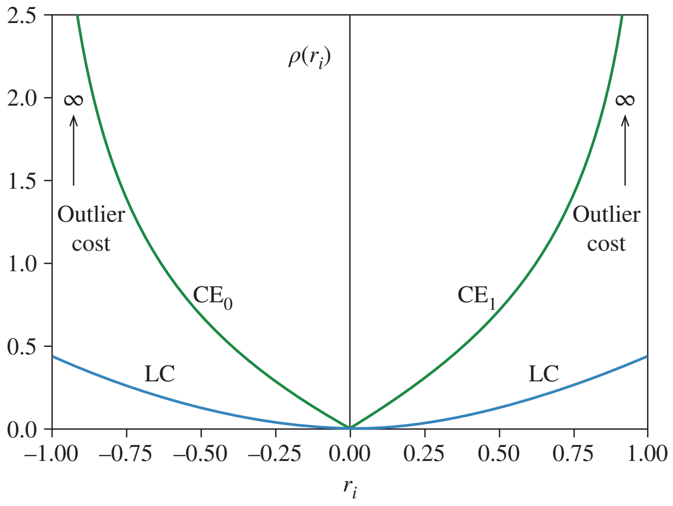


Figure 8.7 Comparison of $\rho(r_i)$ for CE and LC.

the binary variable y which are both in the range of 0 and 1. Therefore, only this region is plotted along the x-axis. There are several important points to make at this stage. First, the two halves of the plot for $\rho_{\text{CE}}(r_i)$ are independent so the lack of a smooth transition at the origin is not an issue. In the figure, the two sides are labeled separately as CE$_1$ and CE$_0$, respectively. Second, the curves for both halves go to infinity at −1 and 1, respectively, which is highly undesirable with outliers. It implies that outliers will have an infinite weight attached to them and force solutions that are far from the desired general model. In fact, this *explains why there is a strong influence of outliers* on the resulting model. And third, the LC loss term does not have any of these issues and uses a weight of roughly 0.43 on outliers, which is quite low. This is why it produces results that are robust to outliers.

On the other hand, the gradients associated with the log-cosh loss function are generally smaller, which implies that convergence may require more iterations. The smoothness of log-cosh leads to an optimization space that is flatter compared to the cross-entropy loss, which is typically steeper, allowing gradient descent to make more progress in each iteration. This can be understood by examining Eqs. (8.3.23) and (8.4.5). For any term in the summation, it can be shown that

$$|p_i(1-p_i)\tanh(y_i-p_i)| \leq |(y_i-p_i)|. \tag{8.4.9}$$

For a given learning rate, the smaller gradients and flatter landscape of log-cosh imply slower convergence compared to cross-entropy. However, this is a small price to pay for a more robust and general solution.

8.4.1 General Formulation

A more general formulation for the LC loss is one that contains the robustness coefficient, a (as described in Chapter 3), as follows:

$$J^{\text{LC}}(\boldsymbol{\beta}) = \frac{1}{a}\sum_{i=1}^{n}\log(\cosh(a(y_i-p_i))). \tag{8.4.10}$$

The general log-cosh logistic regression estimator is then given by

$$\hat{\boldsymbol{\beta}}^{\mathrm{LC}} = \underset{\beta \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \left\{ \frac{1}{a} \sum_{i=1}^{n} \log(\cosh(a(y_i - p_i))) \right\}. \tag{8.4.11}$$

Typically, a value of a in the range of 5–10 is appropriate for logistic regression (see Chapter 12). The optimization technique to solve Eq. (8.4.11) is gradient descent but we will need a set of partial derivatives to utilize it. Following the steps outlined earlier, we obtain

$$\frac{\partial J^{\mathrm{LC}}(\boldsymbol{\beta})}{\partial \beta_j} = \sum_{i=1}^{n} p_i(1 - p_i) \tanh(a(y_i - p_i))(-x_{ij}). \tag{8.4.12}$$

The update equation for each β_j to compute the new parameter values is given by

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \ell \times \frac{\partial J^{\mathrm{LC}}(\boldsymbol{\beta})}{\partial \beta_j}, \tag{8.4.13}$$

where ℓ is the learning rate that controls the step size needed in each iteration, t, to produce a smooth yet rapid convergence to the optimum solution.

8.5 Algorithms for Logistic Regression

This section describes the algorithms for binary classification problems using cross-entropy and log-cosh loss functions. Recall that the cross-entropy loss function is

$$J^{\mathrm{CE}}(\boldsymbol{\beta}) = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)], \tag{8.5.1}$$

and the log-cosh loss function is

$$J^{\mathrm{LC}}(\boldsymbol{\beta}) = \frac{1}{a} \sum_{i=1}^{n} \log(\cosh(a(y_i - p_i))). \tag{8.5.2}$$

Simplified algorithms in the form of pseudo-code for the two methods, referred to as CELR and LCLR, are provided in Algorithms 8.1 and 8.2, respectively. Actual implementations can be much more involved but, perhaps more importantly, they are almost identical except for lines 5 (loss update) and 7 (gradient calculation). Therefore, the implementation of the log-cosh approach in existing software is rather straightforward.

The algorithms begin by reading in the $\boldsymbol{X}$ and $\boldsymbol{y}$ data, along with the settings of the desired number of iterations, T, and learning rate, ℓ. The two hyperparameters will be different for CELR and LCLR. Then the $\boldsymbol{\beta}$ estimates are initialized to some small values. Next, a **while** loop is executed until the specified number of iterations is reached. In the loop, the loss function is computed to monitor convergence, the derivatives for gradient descent are computed and the parameters are updated. The final estimates are produced when the program exits the **while** loop.

The question of when robust methods should be used depends on whether the dataset is believed to contain outliers. Many binary classification problems have two classes that are largely separable, but most

Algorithm 8.1 Cross-entropy logistic regression (CELR).

```
1: Input: training dataset $\boldsymbol{X}, \boldsymbol{y}$; number of iterations ($T$); learning rate ($\ell$).
2: Set $n$=number of observations, $d$=number of parameters.
3: Set $t$=0. Initialize $\beta^{(0)}$ to small nonzero values.
4: while $t < T$ do
5:    Compute loss function:
      $J^{\text{CE}}(\beta) = -\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$
6:    for j = 0 to $d$ do
7:       Compute gradient of the loss w.r.t. $\beta_j$:
         $\frac{\partial J^{\text{CE}}(\beta)}{\partial \beta_j} = \sum_{i=1}^{n}(y_i - p_i)(-x_{ij})$
8:       Update estimate: $\beta_j^{(t+1)} = \beta_j^{(t)} - \ell \times \frac{\partial J^{\text{CE}}(\beta)}{\partial \beta_j}$
9:    end for
10:   Update iteration: $t = t + 1$
11: end while
12: Output: model $\hat{\boldsymbol{\beta}}^{\text{CE}}$
```

Algorithm 8.2 log-cosh logistic regression (LCLR).

```
1: Input: training dataset $\boldsymbol{X}, \boldsymbol{y}$; number of iterations ($T$); learning rate ($\ell$).
2: Set $n$=number of observations, $d$=number of parameters.
3: Set $t$=0. Initialize $\beta^{(0)}$ to small non-zero values.
4: while $t < T$ do
5:    Compute loss function: $J^{\text{LC}}(\beta) = \frac{1}{a}\sum_{i=1}^{n}\log(\cosh(a(y_i - p_i)))$
6:    for j = 0 to $d$ do
7:       Compute gradient of the loss w.r.t. $\beta_j$:
         $\frac{\partial J^{\text{LC}}(\beta)}{\partial \beta_j} = \sum_{i=1}^{n}\tanh(a(y_i - p_i))p_i(1-p_i)(-x_{ij})$
8:       Update estimate: $\beta_j^{(t+1)} = \beta_j^{(t)} - \ell \times \frac{\partial J^{\text{LC}}(\beta)}{\partial \beta_j}$
9:    end for
10:   Update iteration: $t = t + 1$
11: end while
12: Output: model $\hat{\boldsymbol{\beta}}^{\text{LC}}$
```

tend to have outliers. Therefore, it is best to select a robust method. One can show that if there are no outliers, the results of the two approaches are very similar. Therefore, log-cosh offers acceptable results with or without outliers.

Our next step is to explore the advantages and disadvantages of CELR and LCLR through a number of examples using a program written in Python. The results in the sections to follow are based on the implementation of both algorithms in one program with the option of selecting CELR or LCLR to allow fair comparisons to be carried out, including issues related to regularization to improve the model.

8.6 Example: Motor Trend Cars

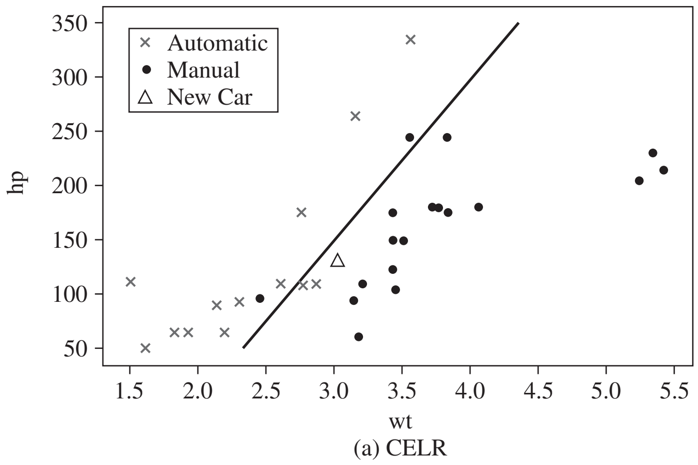
We begin with a simple illustration of a binary classification problem with a single outlier. The dataset to be used is called **mtcars**. It contains information about whether or not a car has automatic or manual transmission (am) based on its horsepower (hp) and weight (wt). Hence, the binary response is am, while the explanatory variables are hp and wt. The dataset was shown earlier as a scatter plot in Figure 8.3 along with an ideal decision boundary line. Now, imagine a hypothetical setting as an employee in a car company. Your manager gives you the mtcars dataset and asks you to determine whether a soon-to-be-designed car with hp = 125 and wt = 3.0 Mg (where 1 Mg = 1000 kg) should use an automatic or manual transmission. How can you decide which type of transmission the car should use? Based on the fact that the response variable is binary, logistic regression is a suitable choice.

If we apply the cross-entropy method using CELR, we obtain the separation boundary shown in Figure 8.8(a). This is a somewhat surprising result at first glance because it is visually clear where the separating line should go and CELR does not produce it because of the influence of one outlier. Unfortunately, this is the main issue with CELR—the decision boundary is altered by the presence of outliers. As a result, the linear boundary actually incurs four misclassifications in this example. There are also two points straddling the line that are close enough to be called a 1 or 0. Therefore, even in this ideal setting where a clear delineation exists between 0 and 1 responses, CELR may make a few mistakes if outliers are present. To answer the manager's question about the newly designed car with 125 hp and a weight of 3.0 Mg (shown using a the symbol △ in the figure), the decision would be to make it a manual transmission based on CELR.

Now consider the results with LCLR shown in Figure 8.8(b). This is exactly what one would expect without any outliers present, and visually this seems like the correct result as discussed previously. The use of log-cosh effectively ignores the outlier in this case. It also behaves like a large margin classifier, similar to SVCs (to be described in a later section). As an answer to the manager's question, LCLR recommends an automatic transmission (indicated by the △ located on the other side of the boundary). This is the correct decision. Clearly, the robust solution ignored the outlier and produced a higher-quality result (which could save your job in the end!).

One important point to note is that we can obtain probability information using logistic regression. The probability can be used to decide whether the point is a 0 or a 1, or can be used unaltered to provide more information for a business decision. The reason for its importance is 2-fold. First, in a high-dimensional setting, we cannot plot the decision boundary associated with $p = 0.5$ and examine it visually. The probability is more informative than simply stating the class. Second, the threshold for determining whether a point belongs to Class 0 or Class 1 is nominally set to 0.5, but if the more appropriate setting for a particular application, say in the case of deciding on whether to give out a loan, may be much higher, such as 0.7. In many data science problems, we may be interested in the probability value rather than the class and logistic regression provides this information.

A contour map is shown in Figure 8.9 based on the probabilities computed using the logistic function. The decision boundary line is the case where $p = 0.5$. Other probability contours are parallel to this line and reach $p = 1.0$ on the left-hand side and $p = 0.0$ on the right-hand side. It is only out of convenience that we plot only the $p = 0.5$ line and call it the decision boundary. As a practical matter, it can be set to any desired value between 0 and 1.



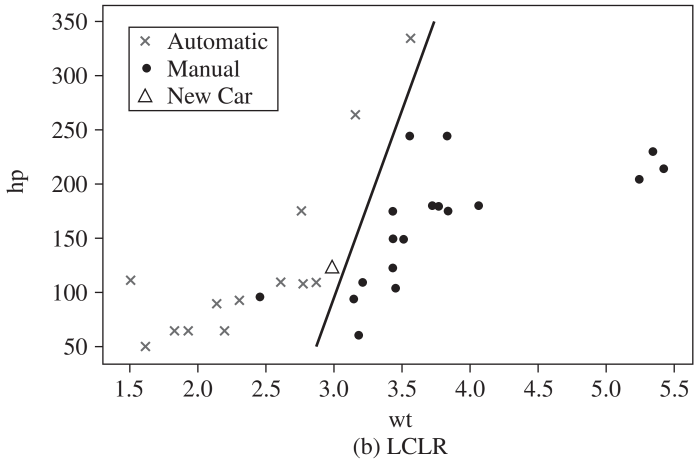


Figure 8.8 Comparing the decision boundaries of CELR vs. LCLR with one outlier.

8.7 Regularization of Logistic Regression

Binary classification problems also face issues related to the bias–variance trade-off (see Chapter 6), more commonly known as overfitting and underfitting. In overfitting, we may produce a model that predicts the training set with high accuracy but fails to reach the same level of accuracy on the test set. Such a model is said to have a high variance and low bias, which implies overfitting. In underfitting, the model with high bias and low variance is unable to attain sufficient training accuracy due to the lack of enough features. In machine learning, we always seek a general model to balance overfitting and underfitting using a variety of different techniques.

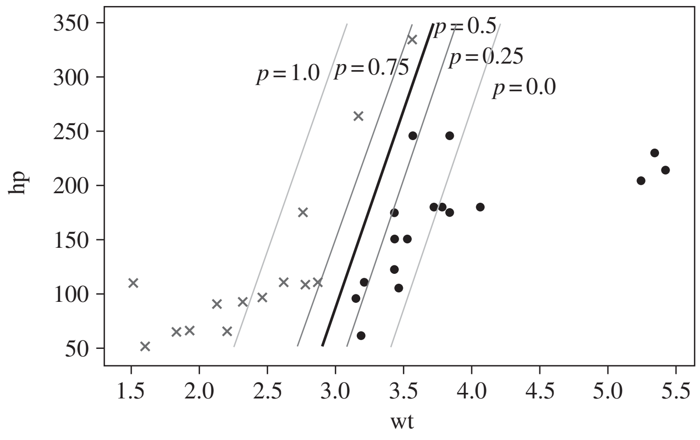


Figure 8.9 Probability contours from logistic regression.

8.7.1 Overfitting and Underfitting

In Figure 8.10, the concepts of overfitting and underfitting in binary classification are illustrated in a two-dimensional space. The 1's represent Class 1 and the 0's represent Class 0. The model parameters define the decision boundary between the two classes. In Figure 8.10(a), the decision boundary is shown to be some complex path through the data to separate 1's and 0's. This overfitted situation produces high accuracy on the training data. Note that if there are outliers present, as shown in the figure, they will not be predicted correctly. Therefore, 100% accuracy may not be achieved in training sets with outliers even if the model is overfitted. In any case, the nonlinear decision boundary may not be general enough to obtain good accuracy on test data. The reason is that there are overfitted regions that are stretched out to capture 0's and this may cause errors in classification on a test set.

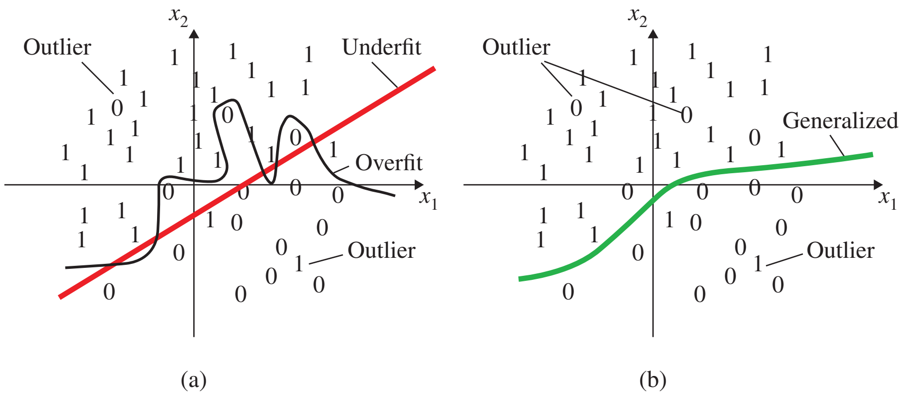


Figure 8.10 Effect of overfitting, underfitting, and generalization. (a) overfitting and underfitting, and (b) generalized result.

The linear boundary shown in the same plot is an example of underfitting. In this case, the decision boundary is restricted to being linear so it is difficult to find a good solution. Figure 8.10(b) shows a more general result which, in this case, produces a nonlinear decision boundary. It will likely produce good accuracy on a wide range of test sets. When building a model, the goal is to balance overfitting and underfitting in some manner using cross-validation, penalty functions, feature selection, or a variety of other techniques. In the next two sections, we examine the use of cross-validation and penalty functions for binary classification.

8.7.2 *k*-Fold Cross-Validation

The standard approach for generalization in logistic regression is to use k-fold cross-validation. This subject was described in detail in Chapter 6 and will not be repeated here. However, its use in connection with outliers in datasets is relevant to our discussion. The main issue is the presence of outliers in the training set and the test set and how to interpret the results. First, the training set will be partitioned into k folds and each one may have outliers. Therefore, the CELR approach will be influenced by outliers as described in the previous section, whereas LCLR will exhibit its characteristic robustness to outliers. Therefore, the results obtained using k-fold cross validation will usually be better for LCLR with outliers present.

Second, there is a problem with the test set accuracy when outliers are present. It is difficult to control the number of outliers in the test set without any knowledge of the decision boundary since the test set is constructed using random selection from the dataset (before any training). However, after the decision boundary has been established, it may result in a number of outliers in the test set which would be manifested as classification errors. These outliers can be detected and removed using the techniques described in Section 8.9, but this is not generally done in practice. Recall that the unseen test set is mainly used to confirm the results of the trained model. We are not permitted to adjust its contents to achieve better results. The test accuracy might be higher or lower than what was achieved during training, but we expect it to be in the same ballpark to validate the model. In Figure 8.10, imagine if only outliers appeared in the test set. The accuracy would be 0% which is not reflective of the actual performance of the robust model. Therefore, it is important to use both the cross-validation scores after training and the test scores to evaluate the model due to unpredictable effects of outliers in the unseen test set. In general, the two results tend to line up if there is enough data to work with using LCLR.

8.7.3 Penalty Functions

Penalty functions, such as ridge, LASSO, aLASSO, and Enet, were detailed in Chapters 5 and 6. They are useful for fine-tuning a linear regression model to balance overfitting and underfitting—a process called regularization. The cross-entropy loss for logistic regression can be regularized using penalty functions in the same manner. Regularization in this context can be viewed as a way to adjust the parameters until a high accuracy is achieved during cross-validation. This is particularly important for CELR since it requires tuning a hyperparameter λ to obtain acceptable results. However, for robust logistic regression, there is some amount of "auto-regularization" built-in to the loss function so that penalty functions are not as essential. For that reason, we present results only for the regularization of CELR.

Regularization can be applied to cross-entropy logistic regression using ridge as follows:

$$J^{\text{CELR-ridge}}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)] + \frac{\lambda}{2}\sum_{j=1}^{d}\beta_j^2, \tag{8.7.1}$$

where $\lambda > 0$ is the regularization hyperparameter. The hyperparameter λ can be moved to the first term by defining a hyperparameter C, where $C = 1/\lambda$ to conform to notation used in logistic regression packages in Python. Therefore,

$$J^{\text{CELR-ridge}}(\boldsymbol{\beta}) = -C\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)] + \frac{1}{2}\sum_{j=1}^{d}\beta_j^2. \tag{8.7.2}$$

Often, the equation is written in the form

$$J^{\text{CELR-ridge}}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{j=1}^{d}\beta_j^2 - C\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]. \tag{8.7.3}$$

For LCLR, the use of ridge may act to offset its robustness and produce misleading results. However, k-fold cross-validation is important for generalizing both CELR and LCLR independent of the use of ridge.

8.7.4 Effect of Outliers

We now apply a ridge penalty to CELR for cases with and without outliers and compare it against LCLR. For this purpose, we return to the **mtcars** dataset. An obvious first experiment is to remove all outliers and compare the two methods. This is shown in Figure 8.11. Note that in Figure 8.11(a), a good solution on linearly separable data is not obtained by CELR with $C = 1$, whereas LCLR produces the ideal boundary in Figure 8.11(b). However, using $C = 1000$ does improve the result bringing it in line with LCLR. Therefore, hyperparameter tuning of C is required in CELR to obtain satisfactory results.

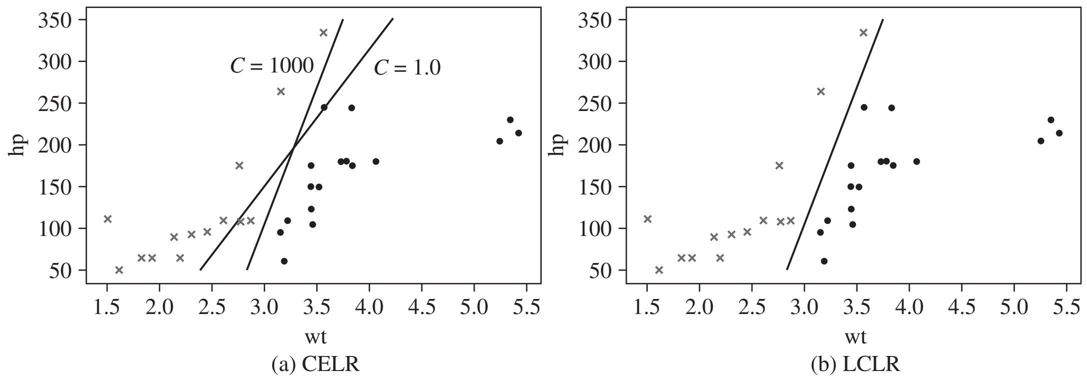


Figure 8.11 Comparing CELR and LCLR with no outliers. CELR is unable to find a proper boundary on linearly separable data until $C = 1000$. LCLR finds the proper decision boundary.

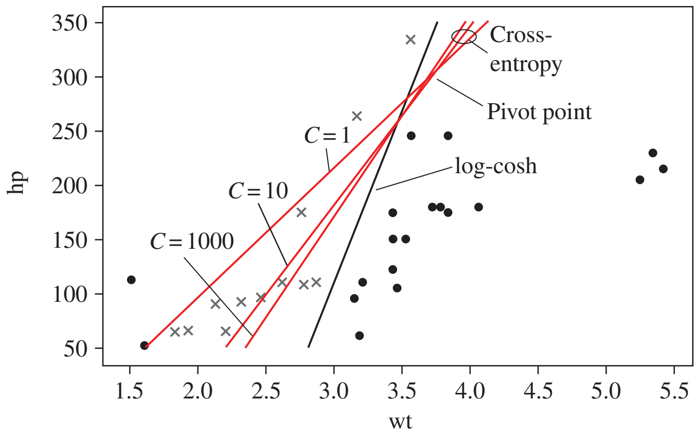


Figure 8.12 Comparing CELR ($C = 1, 10$, and 1000) vs. LCLR with two outliers. The decision boundary for CELR is controlled by the two outliers whereas LCLR is unaffected, as if there were no outliers.

In Figure 8.12, the results for mtcars with two extreme outliers are shown with different levels of hyperparameter tuning using $C = 1, 10$, and 1000. Each value of C changes the slope of the line. All such lines are anchored at a *pivot point* and, as a result, they are not able to produce the robust solution of LCLR. In fact, an extremely large value of C does not change the result much beyond the line for $C = 1000$. Clearly, regularization via hyperparameter tuning can only go so far in producing general models in CELR with outliers present. In reality, the alternative solutions provided are essentially different amounts of rotation around the pivot point. Tuning does not solve the outlier problem for CELR but can mitigate it to some degree. However, each new value of C in CELR requires a k-fold cross-validation step which can be expensive. This cost is avoided completely in LCLR for which only one round of k-fold CV is needed to generalize the model.

8.8 Example: Circular Dataset

In this section, an example is used to compare CELR and LCLR in which the data requires a *nonlinear decision boundary*. A synthetic dataset that requires a circular/elliptical boundary is shown in Figure 8.13. The two-dimensional set of points is either gray (0) or black (1) and the goal of binary classification is to find a suitable separation boundary. To connect this example to a toy data science problem, let us imagine that Bank ABC has given out 337 loans to customers in its local vicinity. After some analysis, the bank manager finds that 266 loans were not repaid in full (gray points) but 71 loans were completely repaid (black points). This is not an acceptable ratio so the manager decides to use machine learning to make all future loans. The two features to be used are the normalized `age` of the customer and normalized `size` of the loan. Normalizing the columns in the data matrix ensures that each feature contributes equally to the optimization process, thereby enhancing the performance and accuracy of machine learning algorithms. In this case, the range is $[-2, 2]$, although generally the normalized range is $[-1, 1]$.

The scatter plot of the data shows that middle-aged customers with mid-size loans have all paid back completely, along with some folks with high value loans. Note that there is a separation region

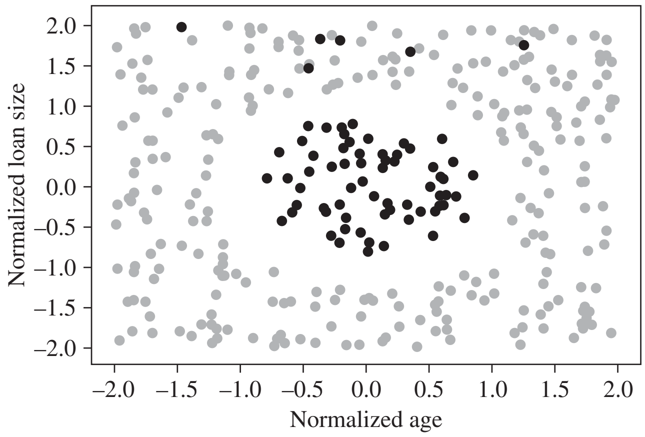


Figure 8.13 Binary classification problem with nonlinear decision boundary.

between the gray and black points. There are also six outliers (in black) in the upper region of the figure representing the high-value loans that were paid back. If we ignore these points, the data is clearly separable but not linearly separable. The two variables, `age` and `size`, by themselves are insufficient to produce a nonlinear boundary. We need to carry out a polynomial expansion of these variables. In particular, if we have two variables, x_1 and x_2, we must use polynomial expansion to obtain x_1, x_2, x_1x_2, x_1^2, and x_2^2 as our new feature set since the solution has a circular boundary. For this case, the term x_1x_2 is not needed. Therefore, the number of variables is 4 (i.e. $d = 4$), which implies that $\boldsymbol{\beta} \in \mathbb{R}^5$ since the intercept, β_0, must also be included.

When we expand the feature space in this manner, i.e. via polynomial expansion, we are projecting the data into a higher-dimensional space with the hope that there is a linear boundary to be found in that space. We can illustrate the linear boundary that lies in the `age`^{2} and `size`^{2} space by replotting the data, as shown in Figure 8.14. There is indeed a linear boundary and therefore we are likely to find a reasonable solution. Obviously when the dimensions become larger than 2–3, it is difficult to determine the degree of polynomial expansion needed but this is part of the art of feature engineering.

We can now proceed knowing that a good solution is close at hand using polynomial expansion. To compare the robust approach to the non-robust approach, we can execute CELR and LCLR followed by a comparison of the nonlinear decision boundaries. The results are provided in Figure 8.15. The CELR approach (with $\lambda = 0$) is not able to find a satisfactory solution because of the influence of the outliers in the upper region. In fact, the separation boundary captures all the black points in the central region (as desired), but inadvertently captures a number of gray points (which is undesirable). From the perspective of our loan problem, this implies that some high value loans will be given out in the future even though customers are unlikely to pay it back.

The LCLR approach finds a much better decision boundary that separates the data in a manner as if the outliers did not exist. It also obtains a solution similar to a large-margin classifier. In fact, it achieves 331 correct classifications—the maximum possible given that there are 6 outliers. One lingering question

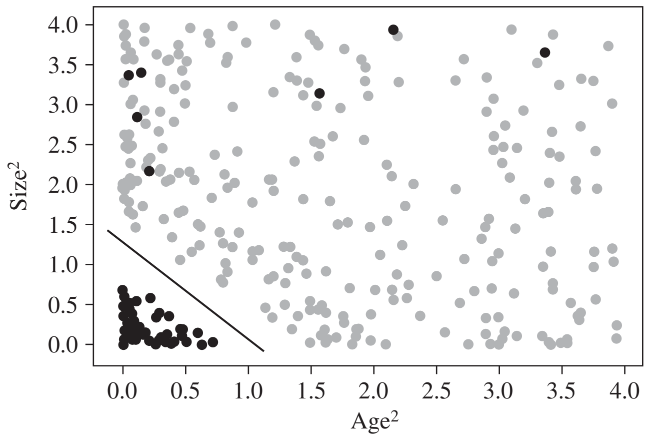


Figure 8.14 Scatter plot of the circle data for `size`^{2} against `age`^{2}.

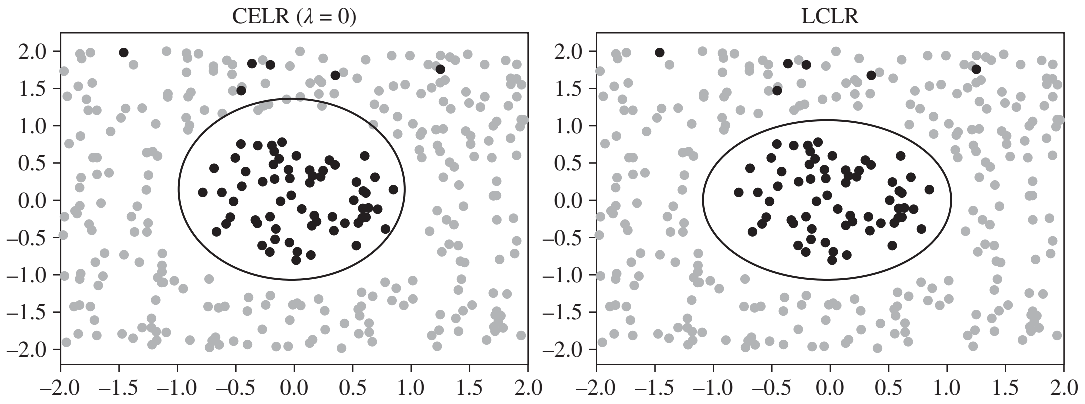


Figure 8.15 Binary classification comparison—nonlinear boundary.

is whether a penalized version of CELR would improve the situation. We have already seen the case of $\lambda = 0$. The evolving decision boundaries for several nonzero values of λ using CELR-ridge are shown in Figure 8.16. An optimal choice is $\lambda = 10$. The boundary first shifts downward as λ increases and then begins to shrink in size as λ is increased towards infinity. Therefore, regularization is important for CELR. For LCLR-ridge, the only effect is to worsen the decision boundary so it is not as useful as in the case of CELR-ridge.

These two simple examples, one with a linear decision boundary for the mtcars dataset, and the other with a nonlinear decision boundary for the circular dataset, demonstrate the inherent value of LCLR over CELR. While these are toy examples, they provide the needed insight to understand the differences. A larger example will be presented in Chapter 12 involving the Titanic dataset to showcase additional advantages of the robust approach.

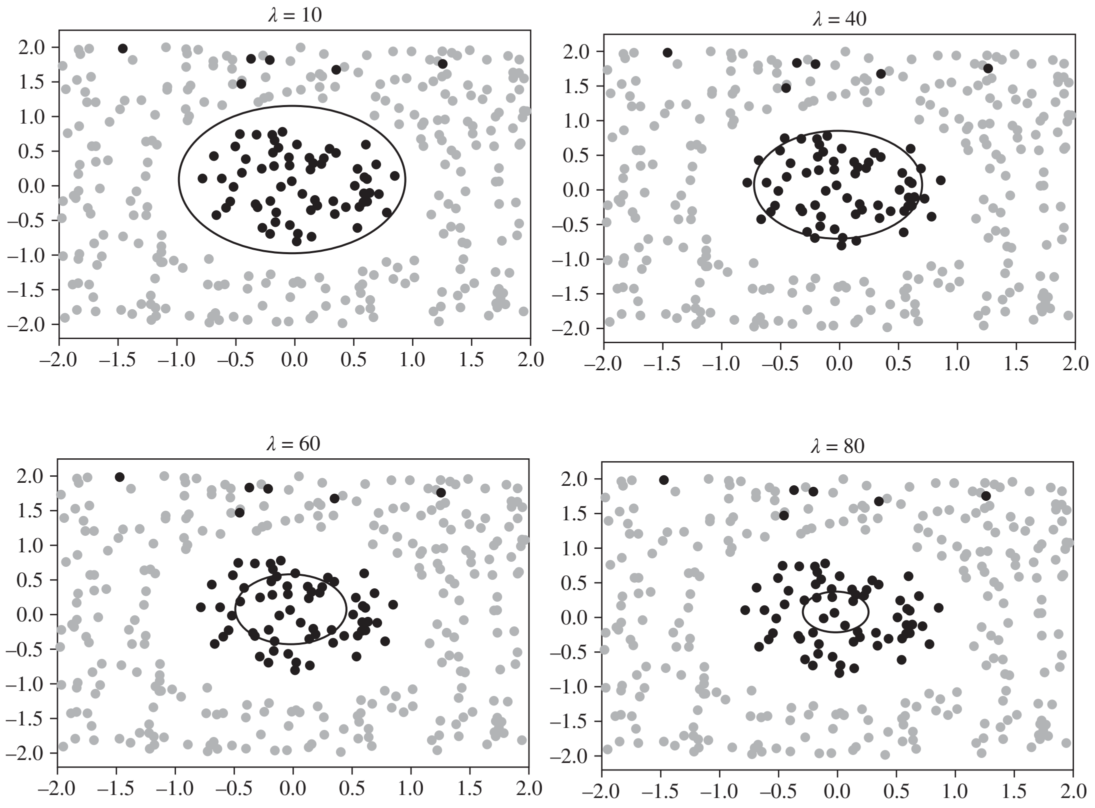


Figure 8.16 CELR-ridge regularization showing a suitable result for $\lambda = 10$ after which the shrinking decision boundary collapses to 0 as $\lambda \to \infty$.

8.9 Outlier Detection

In binary classification problems, it is possible to identify potential outliers in the dataset after logistic regression is performed. The basic idea is to examine the histogram of the residuals. Since this is a binary classification problem, the difference between the predicted probabilities and the ground truth for all points lies between −1 and +1. Outlier diagnosis simply requires a plot of the histogram of the residuals from a robust regression. Residuals located in the vicinity of −1 and +1 are outliers. Detection simply involved counting the number of outliers in those two regions.

Consider the histograms shown in Figure 8.17. The left panel is the histogram for residuals of CELR while the right panel shows the residuals using LCLR. The outliers in this case all reside at −1, as detected by LCLR. Note that in the case of CELR, the errors are more distributed compared to LCLR. In general, LCLR is more likely to distinguish outliers from inliers as will be demonstrated in Chapter 12.

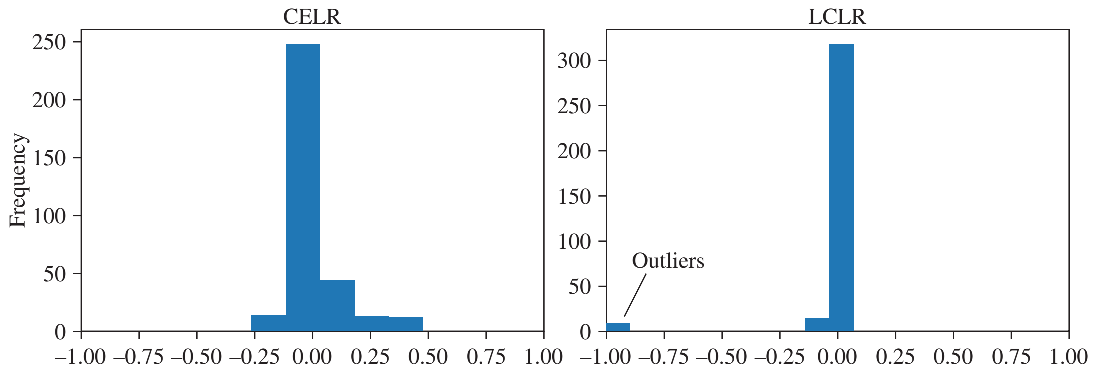


Figure 8.17 Histograms of residuals for CELR and LCLR on the circular dataset.

8.10 Robustness of Binary Classifiers

In the realm of binary classification, a number of other methods are available that have been routinely used in practice for several decades. However, they have not been fully investigated in terms of their robustness to outliers. In this section, we review these methods, including SVMs, k-nearest neighbors (k-NN), and decision trees to illustrate to what degree they may be impacted by outliers. We begin with the SVC and conclude the section by examining the random forest method.

8.10.1 Support Vector Classifier (SVC)

A large margin classifier is one that places a decision boundary that is maximally distant from points in Class 0 and Class 1, assuming the data is linearly separable and no outliers exist. We already saw this property in LCLR but not in CELR. Other binary classifiers also possess this property, specifically the SVC which is the subject of this section. To develop intuition about SVCs, recall the regularized version of CELR given earlier as

$$J^{\text{CELR-ridge}}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{j=1}^{d}\beta_j^2 - C\sum_{i=1}^{n}[y_i \log p_i + (1-y_i)\log(1-p_i)], \tag{8.10.1}$$

where C is the regularization hyperparameter. If we want to change the characteristics of this classifier, we simply need to choose a different loss function, which is the term after C (i.e. the cross-entropy loss in this case).

To understand SVC, consider the diagram in Figure 8.18 where a linearly separable dataset is shown. If the goal were to find a decision boundary with the maximum margin size, the result would be the dark line shown. The so-called margins are represented by the dotted lines on each side. Any points that lie on the margins are called *support vectors*,[2] labeled as **SV**. The size of the margin is $2/||\boldsymbol{\beta}||_2$, where $\boldsymbol{\beta}$ is value

2 The term "vector" is derived from the simple fact that all $\boldsymbol{x}_i$ can be viewed as vectors in the d-space.

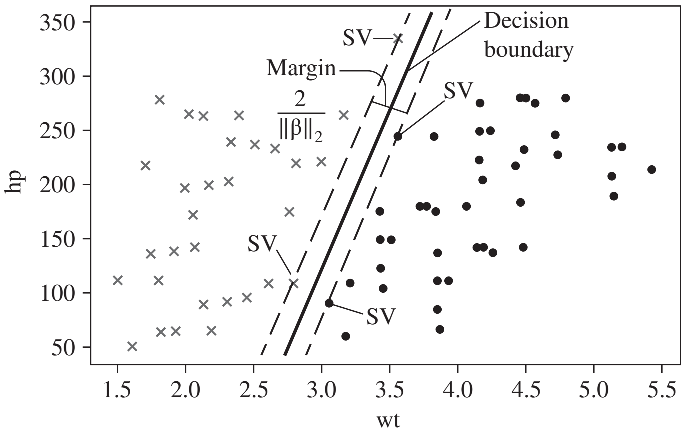


Figure 8.18 A demonstration of a support vector classifier showing the decision boundary, support vectors (SV) and large margins.

that we seek through optimization. To obtain the largest margin, the term $2/||\boldsymbol{\beta}||_2$ is to be maximized. Alternatively, we can instead minimize $||\boldsymbol{\beta}||_2/2$, or equivalently minimize $||\boldsymbol{\beta}||_2^2/2$ (for mathematical convenience). However, minimizing this function would lead to the trivial solution $\boldsymbol{\beta} = 0$ which sets the margins at infinity. We need to apply some constraints to obtain nontrivial results.

To formulate the optimization problem, let $y_i \in \{-1, 1\}$, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_d)^\top$, and call β_0 the bias. For logistic regression, we said that $\boldsymbol{x}_i$ belongs to Class 1 if

$$(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) > 0$$

and Class 0 if

$$(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) < 0.$$

Now, we insist on a larger margin such that $\boldsymbol{x}_i$ belongs to Class 1 only if

$$(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) > 1$$

and Class 0 only if

$$(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) < -1.$$

Instead of keeping track of two conditions, we could simply write that $\boldsymbol{x}_i$ is properly classified if

$$y_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) > 1.$$

Otherwise it is improperly classified. The added constraint in choosing a suitable $\boldsymbol{\beta}$ is that all the data be properly classified, which avoids the trivial solution.

The constrained optimization problem can be stated as

$$\underset{\boldsymbol{\beta} \in \mathbb{R}^d}{\text{minimize}} \frac{1}{2}||\boldsymbol{\beta}||^2 \quad \text{s.t.} \quad y_i(\boldsymbol{x}_i^\top \boldsymbol{\beta} + \beta_0) > 1, \;\; \forall\, i = 1, \dots, n. \tag{8.10.2}$$

Using a Lagrange multiplier formulation, we obtain the following objective function:

$$J^{\text{SVC}}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{j=1}^{d}\beta_j^2 + C\sum_{i=1}^{n}\max\left(0, 1 - y_i(\boldsymbol{x}_i^{\top}\boldsymbol{\beta} + \beta_0)\right), \tag{8.10.3}$$

where C is the regularization hyperparameter and the second term uses the **hinge loss** to enforce the requirement, $y_i(\boldsymbol{x}_i^{\top}\boldsymbol{\beta} + \beta_0) > 1$. Effectively, this is just a replacement of the cross-entropy loss of Eq. (8.10.1) with the hinge loss. This convex optimization problem can be minimized in a straightforward manner. Not all points may be properly categorized at the minimum but it will be optimal relative to any given problem.

Consider the two plots in Figure 8.19. The results shown for the case with no outliers in the left-hand panel would be produced using SVC. The decision boundary lies in the middle of two dotted lines that define the margins. This approach is very effective as a large-margin linear classifier. The two support vectors lie on the margins running in parallel to the decision boundary. They are the only points involved in defining the decision boundary. None of the other points matter in the solution. In that sense, this is a robust method.

However, it is not robust to outliers, and that is one of the weak points for standard SVCs. This is shown in the plot on the right-hand side of Figure 8.19. When two outliers are introduced, the results clearly shift to the left. The reason for this should be clear: the loss function penalizes errors in classification and each outlier adds a large error to the objective function. If we look back at Eq. (8.10.3), we note that if the classification of a point is correct, the term inside the summation of the loss function is identically 0. However, if it is incorrect, as will always occur with outliers on the wrong side of the margins, the term will be greater than 1 and make a large contribution to the total loss. Points inside the margins will contribute to the loss but these values will be small.

The complete picture is illustrated in Figure 8.20 where the hinge loss is plotted with various segments indicating the cost for each type of data point. Regularization may help somewhat in this situation but it depends on the location and impact of the outliers during cross-validation. Therefore, we can conclude that outliers present a problem to the SVC method, although it is robust in terms of the support vectors at the margins of the decision boundary.

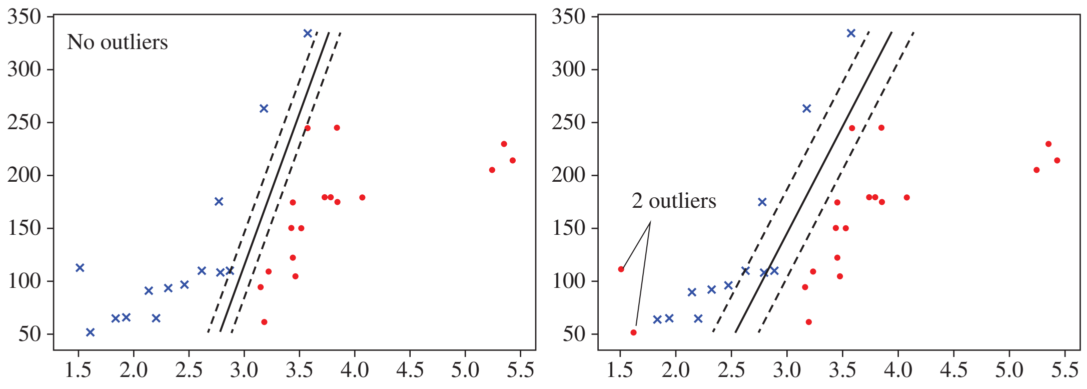


Figure 8.19 Results of support vector classifier for the case of no outliers and two outliers.

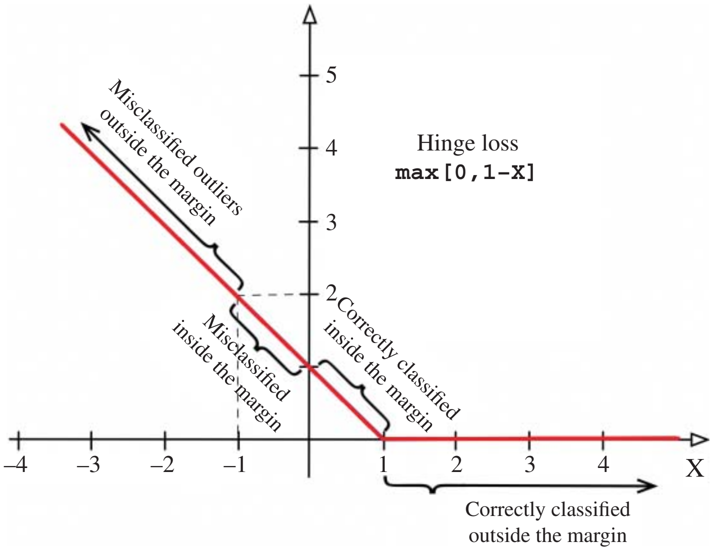


Figure 8.20 Hinge loss with segments showing data points contributing to the loss calculations. Notice that the largest contributors to the total cost are misclassified outliers.

8.10.2 Support Vector Machines (SVMs)

The basic SVC approach described above can only produce linear boundaries with large margins. Polynomial expansion can be used to produce nonlinear boundaries. However, the SVM is a more powerful formulation based on *duality* to allow for nonlinear boundaries without resorting to polynomial expansion. First, recall that $\boldsymbol{\beta}$ represents the model parameters for each data column in the training set. Next, let $\boldsymbol{\alpha}$ be the model parameters associated with each row of the training data, as shown in Figure 8.21. This is not something we have considered thus far but it is quite feasible to formulate the problem in a dual form. Now, we want to relate $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ in our formulation as follows:

$$\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i. \tag{8.10.4}$$

In other words, there is a relationship captured above between the optimal column coefficients and the optimal row coefficients based solely on the data. Notice that we now have n parameters, essentially one

$$\begin{array}{c|ccccc|c}
 & \beta_1 & \beta_2 & . & . & \beta_d & y \\
\hline
\alpha_1 & X_{11} & X_{12} & . & . & X_{1d} & 1 \\
\alpha_2 & X_{21} & X_{22} & . & . & X_{2d} & 0 \\
\alpha_3 & X_{31} & X_{32} & . & . & X_{3d} & 1 \\
. & . & . & & & . & . \\
. & . & . & \textit{Dataset} & & . & . \\
. & . & . & & & . & . \\
\alpha_n & X_{n1} & X_{n2} & . & . & X_{nd} & 0 \\
\hline
\end{array}$$

Figure 8.21 Relationship between α's and β's in SVMs.

for each of the n rows, rather than one for each of the d columns. But it turns out most of the α_i values are 0 since all points outside the margins will have no contribution to the sum. Only the support vectors have nonzero coefficients and it is this sparsity that makes SVMs efficient from a computational point of view. The support vectors comprise only a small number of points in the training set.

We now have to find the values of the α_i's using some optimization method on a different problem with linear constraints. The dual setup is a maximization problem given by

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\text{maximize}} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \text{ s.t. } \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } 0 \le \alpha_i < C. \tag{8.10.5}$$

The details of the derivation of the above are not important here but rest assured that the α_i's can be computed in an efficient manner using it. Once found, the support vectors due to nonzero α_i's come in two flavors: some will be located exactly at the two margins while others will be located within the margins. This is controlled by the regularization factor C. The new criterion for the decision on some test point $\boldsymbol{x}$ is based on a sign test by replacing $\boldsymbol{\beta}$ with $\boldsymbol{\alpha}$, as follows:

$$\text{sgn}(\boldsymbol{x}^\top\boldsymbol{\beta} + \beta_0) = \text{sgn}\left(\sum_{i=1}^{n}\alpha_i y_i \boldsymbol{x}_i^\top \boldsymbol{x} + \alpha_0\right), \tag{8.10.6}$$

where $\beta_0 = \alpha_0$. If the sign test is positive, then $\boldsymbol{x}$ belongs to Class 1, otherwise it belongs to Class 0. Only linear boundaries are possible with this formulation, as mentioned before regarding SVCs. The interesting part is that it depends on $\boldsymbol{x}_i^\top\boldsymbol{x}$ where $\boldsymbol{x}_i$ is from the training data and $\boldsymbol{x}$ is a new test data point. This can be written more generally as $k(\boldsymbol{x}_i, \boldsymbol{x})$, which is called a *kernel function*.

To obtain different classifiers, we simply choose different kernel functions and then compute the associated α_i's. Three useful kernel functions are the **linear kernel**,

$$k(\boldsymbol{x}_i, \boldsymbol{x}) = \boldsymbol{x}_i^\top\boldsymbol{x}, \tag{8.10.7}$$

the **polynomial kernel**, with expansion up to the d^{th} power, given by

$$k(\boldsymbol{x}_i, \boldsymbol{x}) = (1 + \boldsymbol{x}_i^\top\boldsymbol{x})^d, \tag{8.10.8}$$

and the **radial basis function (RBF) kernel** or Gaussian kernel,

$$k(\boldsymbol{x}_i, \boldsymbol{x}) = \exp(-\gamma||\boldsymbol{x}_i - \boldsymbol{x}||^2). \tag{8.10.9}$$

The latter one, the Gaussian kernel, is the most popular and produces a nonlinear decision boundary that is quite adept at capturing complex boundaries between the two classes. As a result, SVM is a powerful alternative to logistic regression.

The RBF kernel has the property of decreasing in value as a function of the distance from a support vector (i.e. it is an exponential decay that depends on the γ hyperparameter). The SVM has an added advantage over CELR and LCLR in that it does not require specification of all the various polynomial expansions of the variables to create the needed features. It will figure that out for itself from the original variables and data using the RBF. However, the user must adjust two hyperparameters, γ and C, to obtain satisfactory results. After computing $\boldsymbol{\alpha}$ using the training set, the decision of whether $\boldsymbol{x}$ belongs to Class 1 is based only on

$$\text{sgn}\left(\sum_{i=1}^{n}\alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + \alpha_0\right) > 0 \tag{8.10.10}$$

and to Class 0 based on

$$\operatorname{sgn}\left(\sum_{i=1}^{n}\alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + \alpha_0\right) < 0. \tag{8.10.11}$$

Now, by proper selection of the kernel function, this approach allows nonlinear and complex decision boundaries to be created for a given dataset, as opposed to the linear boundary in the previous cases. However, in this form, we can no longer directly extract probabilities, except through additional expensive procedures. In contrast, LCLR can, in principle, provide probabilities as well as a large-margin boundary. Furthermore, the direct connection to the explanatory variables is lost in the SVM since we are estimating parameters for rows rather than columns. Aside from these limitations, the SVM is a very effective method for binary classification.

Figure 8.22 illustrates the capability of the SVM with an RBF kernel. The original dataset is shown in the top-left corner. The data is not linearly separable but a suitable linear decision boundary can be established between the two regions. If we consider the case for $\gamma = 1$ and $C = 1$ for SVM, the panel on

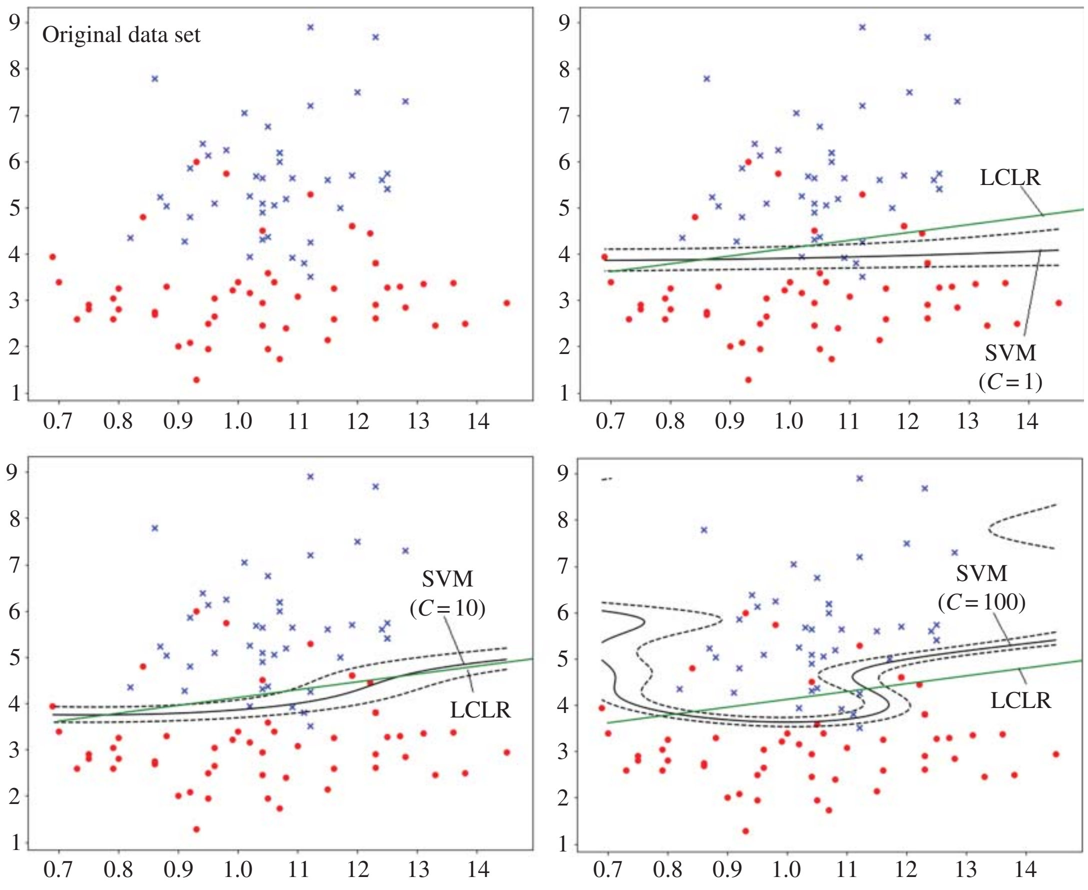


Figure 8.22 Comparison of SVM with RBF kernel ($\gamma = 1$) and $C = 1$, 10 and 100 against the log-cosh loss.

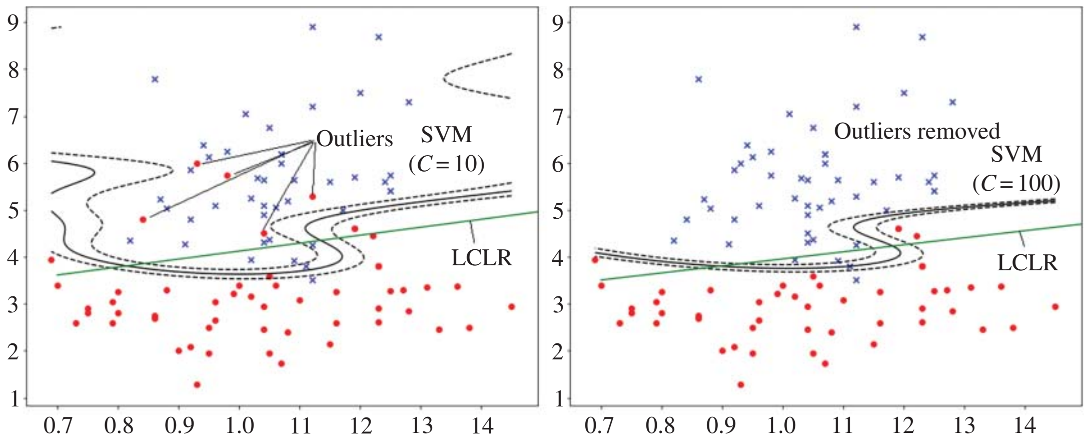


Figure 8.23 Comparison of SVM with RBF kernel ($\gamma = 1$) and $C = 100$ against the log-cosh loss with and without outliers present.

the top-right corner is produced. The LCLR solution is also plotted in the same figure. Note that they do not match very well. However, when we use $C = 10$ for SVM, there is a better match between the two solutions. The nonlinearity of the SVM boundary is clear in the plot. When $C = 100$, there is a certain amount of overfitting in the SVM solution which is not desirable. Overall, the case for $C = 10$ is the most appropriate for this dataset, which is likely to be found via cross-validation, and it matches closely with the LCLR solution.

One of the advantages of LCLR is that no hyperparameter tuning is necessary. It has an auto-regularization feature due to its robustness. It will produce similar results with or without outliers. Consider Figure 8.23. On the left side, the results with outliers are shown for LCLR and SVM from the previous figure. On the right side, the outliers are removed. LCLR hardly changes while there is a noticeable change in SVM. However, LCLR can only produce a nonlinear boundary using polynomial expansion. Here, some knowledge of the application and its characteristics is needed. On the other hand, SVM itself is not robust to outliers for reasons similar to those given in the section on SVCs. Finally, while SVMs provide effective solutions to binary classification, it is not as informative in terms of explainable AI (see Chapter 12).

8.10.3 *k*-Nearest Neighbors (*k*-NN)

The *k*-nearest neighbors (*k*-NN) approach is rather simple and straightforward. It offers good results in a short amount of time and has robust characteristics. It operates by first computing the distance from a given test point to all points in the training data. Then, as its name implies, the nearest k neighbors are used to predict the binary outcome based on a simple majority vote of the neighbors, or a weighted combination based on the distance from the test point. For example, if $k = 3$, then three nearest neighbors of the test point are used to determine the class of the test point. If two neighbors are from Class 0 and one neighbor is from Class 1, the test point is assigned to Class 0. Odd numbers of k are preferred to avoid ties. The distance can be computed in a variety of ways including the Euclidean distance, the Manhattan distance, or any other suitable measure. The dataset should be standardized to obtain good results.

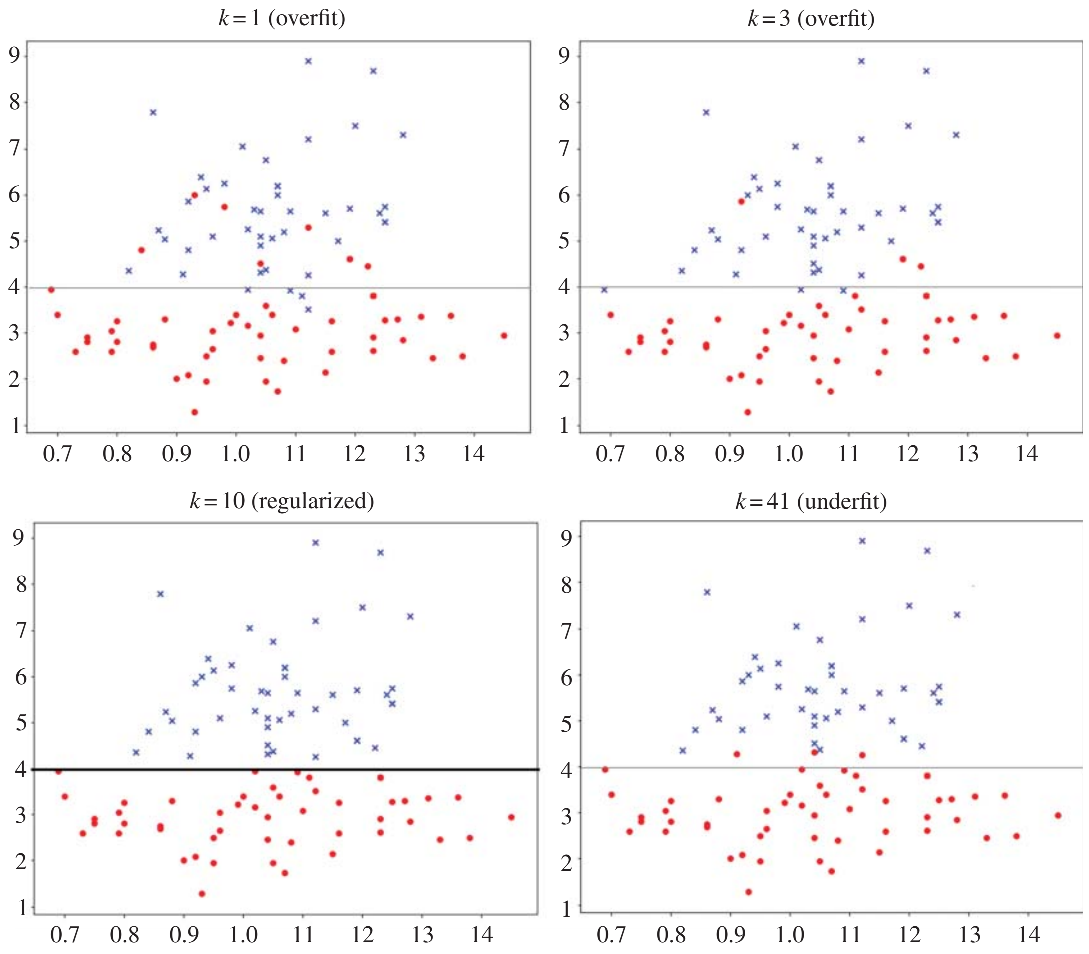


Figure 8.24 Comparison of k-NN binary classification with $k = 1$, 3, 21, and 41.

The value of k also determines the amount of overfitting and underfitting as illustrated in Figure 8.24. If $k = 1$, then the test point is simply assigned to the nearest neighbor class. Clearly, this is an overfitted situation. It does not generalize because any outliers will affect the class of the test point. This can be seen in the top-left panel which uses the data of the previous section. The *training set is used as the test set* to illustrate this effect. For the special case of $k = 1$, the nearest neighbor is itself so the outliers will remain unaltered and the training set is simply replicated. When $k = 3$, as shown in the top-right panel, the three nearest neighbors of the outliers are usually of the opposite class, so most of the outliers are removed. However, three points are still misclassified because there are two outliers in their neighborhood. Therefore, a certain amount of overfitting remains.

When $k = 21$ (which is rather large for this dataset), the results are quite acceptable. All of the outliers are gone meaning that the approach is robust to outliers for a proper setting of k. This is the regularized solution for this dataset. Any point above 4.0 on the y-axis is considered Class 1 and those below 4.0 are in Class 0. The dividing line is shown in all four cases to illustrate how the regularization proceeds relative to different values of k. Finally, with $k = 41$, the method begins to underfit the data. In fact, if $k = n$ (the maximum possible value of k), then the majority of the overall data rules the decision and all data points will be assigned to Class 0.

The main issue with k-NN is that the proper value of k must be found in a multi-dimensional setting that balances overfitting and underfitting using cross-validation. It is not clear how outliers will affect the results but it will have some impact. And second, the results may not be as accurate as SVM or LCLR, especially near the boundaries between the two classes. Third, the interpretability of the results is not as easy as for LCLR but much better than SVM. The issue of explainable AI will be discussed further in Chapter 12.

8.10.4 Decision Trees and Random Forest

In this last section, we briefly describe tree-based methods for binary classification. Generally speaking, both decision trees and random forest methods are robust to outliers. Figure 8.25 shows a decision tree on the left for three variables, a, b, and c. The tree is built by starting at the root node and sequencing through the variables. For each selected variable, a threshold t is chosen such that the rows in the training set with that variable value above t follow one branch of the tree while those below t follow the other branch. The thresholds for a, b, and c are found separately for each variable and are labeled t_a, t_b, and t_c, respectively. Since the threshold is typically selected at some midpoint in the data, the outliers do not affect this part of the process. The number of levels in the tree is referred to as the tree *depth*. Once built, the tree is used for prediction. Each new data point starts at the root node and follows a path through the tree based on the thresholds until some leaf node is reached. At that point, the prediction is based on the points in that leaf node of the tree.

Decision trees can be overfit by having too many levels and only one point in each leaf node. In this case, outliers may cause problems in the prediction phase. The depth of the tree can be controlled by restricting the number of levels in the tree and the minimum number of samples in a leaf node. For regularization purposes, the tree is often constrained by two hyperparameters, `max_depth` and `min_samples`. Assuming the decision tree has been regularized properly, outliers may not play a significant role in the prediction of new points in binary classification. Furthermore, decision trees have much more explanatory power than k-NN and SVMs. Data scientists are usually interested in the relative importance of each variable and they prefer methods for which the results can be rationalized. In the decision tree of Figure 8.25, the binary outcome of any new point can be easily interpreted based on the path it takes through the tree from the root node at the top to some leaf node at the bottom because a clear decision is made at each intermediate node.

Random forests are quite robust to outliers but for a different reason. Consider the illustration on the right side of Figure 8.25. It shows m random trees created by selecting different subsets of the rows and columns of the training set. For prediction, the results from many trees are combined to produce the result. When combining the trees for prediction, it is more tolerant to outliers since a bad prediction in one tree is usually compensated by good predictions in the other trees. Also, random forests are regularized by design so they are not prone to overfitting which causes the sensitivity to outliers. This is a very effective method in machine learning for classification problems. The main issue is interpretability of the results.

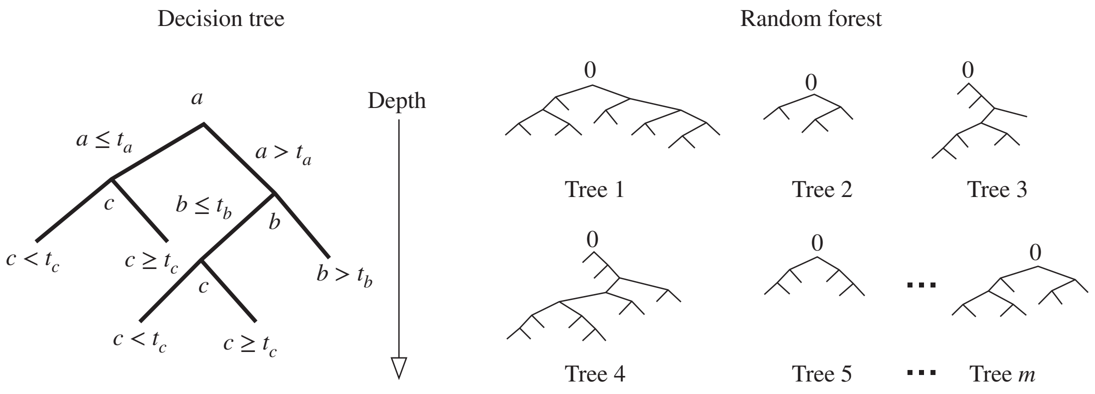


Figure 8.25 Robust decision trees and random forest.

It is very difficult to correlate a decision with the explanatory variables. In spite of this, the random forest continues to be one of the more popular tools in machine learning today.

8.11 Summary

In this chapter, we explored various robust approaches to binary classification. Initially, we discussed the cross-entropy loss in the context of logistic regression, namely CELR, which lacks tolerance for outliers. This was followed by an introduction to an outlier-tolerant logistic regression method using the log-cosh loss, namely, LCLR. Even in the absence of outliers, LCLR's robust estimates can outperform the standard cross-entropy method in prediction accuracy, albeit potentially requiring additional iterations. Notably, robust techniques like LCLR serve as large-margin classifiers, offering both probability values and insights into variable importance (more details are given in Chapter 12). Implementing these robust methods is straightforward, involving minimal code alterations within an existing ML framework. Furthermore, data scientists can leverage this approach for outlier detection. In Chapter 12, further comparisons of CELR and LCLR are carried out on the Titanic dataset, including outlier detection and removal.

Next, we explored the robustness of existing binary classifiers and found that only decision tress and the random forest methods were deemed to be robust. This implies that the other methods are prone to generate models that may not be accurate enough for the data science task at hand when outliers are present. However, random forest does not allow for model interpretation and therefore may not be suitable in all applications. Overall, LCLR provides all the key requirements for binary classification problems.

Problems

8.1 Derive Eq. (8.3.23) of this chapter and Eq. (2.7.8) of Chapter 2. Compare the two equations. How do they compare to the gradient of the LCLR loss function? Plot the two terms of Eq. (8.4.9) as a function of $r_i = y_i - p_i$ in the interval $[-1,1]$ for different values of p_i. Compare the results. Which gradient is larger?

8.2 Find the partial derivatives with respect to β_0 and β_1 of the log-cosh loss function for logistic regression given by

$$J^{\text{LCLR}}(\boldsymbol{\beta}) = \frac{1}{a}\sum_{i=1}^{n} \log(\cosh(\mathrm{a}(y_i - p_i))) \quad \text{where} \quad p_i = \frac{1}{1+\exp(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta})}.$$

(Hint: you need to compute the partial derivative for the slope and intercept, i.e. two cases. See Eq. (8.4.12) for guidance.)

8.3 (Project) This project involves the creation of a circular dataset for use in subsequent problems. You will build a circular dataset similar to Figure 8.13 using Python. There will be six data points defined to be outliers as shown in the upper region of the figure. Run the Python code below and check the results.

a) Import the needed libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
from sklearn.linear_model import LogisticRegression
```

b) Randomly select points to define the binary classification problem.

```
m = 386
np.random.seed(4)
setx = np.random.random(m)*4.0-2.0
np.random.seed(6)
sety = np.random.random(m)*4.0-2.0
y = np.zeros(m)
for i in range(m):
    if (setx[i]**2 + sety[i]**2) < 0.8:
        y[i] = 0
    elif (setx[i]**2 + sety[i]**2) > 1.5:
        y[i] = 1
    else:  #remove these points
        y[i] = -1
    counter = 0
outliers = 6
for i in range(100):
    if sety[i] > 1.6 and counter < outliers:
        y[i] = 0
        counter += 1
```

c) Plot the data to examine the possible decision boundary and the location of the six outliers.

```
setx_df = pd.DataFrame(setx,columns=["xval"])
sety_df = pd.DataFrame(sety,columns=["yval"])
setr_df = pd.DataFrame(y,columns=['flag'])
```

```
setx_df['yval'] = sety_df['yval']
setx_df['flag'] = setr_df['flag']
set1 = setx_df[setx_df['flag']==1]
set0 = setx_df[setx_df['flag']==0]
plt.scatter(set1['xval'],set1['yval'],color='red')
plt.scatter(set0['xval'],set0['yval'],color='blue')
plt.show()
```

8.4 (Project) In this project, you will run CELR using the `LogisticRegression()` module of the `sklearn` library. The code below assumes that the previous project was completed.

a) Create a data frame with a polynomial expansion of order 2 of the two variables.

```
setx_df = pd.DataFrame(setx,columns=["xval"])
sety_df = pd.DataFrame(sety,columns=["yval"])
setr_df = pd.DataFrame(y,columns=['flag'])
setx_df['yval'] = sety_df['yval']
setx_df['flag'] = setr_df['flag']
setx_df['x2'] = setx_df['xval']**2
setx_df['y2'] = setx_df['yval']**2
setx_df["Ones"] = 1
train_df = setx_df[setx_df['flag']!=-1]
train_df.head()
```

b) Run logistic regression based on CELR.

```
Xd = train_df.drop(["flag","Ones"],axis=1)
X = np.array(Xd)
y1 = train_df["flag"]
y = np.array(y1)
logreg = LogisticRegression()
logreg.fit(X, y)
```

c) Plot the nonlinear decision boundary and comment on the results.

```
b1 = [logreg.coef_[0,0],logreg.coef_[0,1],
     logreg.coef_[0,2], logreg.coef_[0,3],
     logreg.intercept_]
plt.figure() # Create a new figure window
set1 = train_df[train_df['flag']==1]
set0 = train_df[train_df['flag']==0]
plt.scatter(set1['xval'],set1['yval'], \
    color='red')
plt.scatter(set0['xval'],set0['yval'], \
    color='blue')
```

```
xlist = np.linspace(-2.0, 2.0, 100)
ylist = np.linspace(-2.0, 2.0, 100)
X1,Y1 = np.meshgrid(xlist, ylist)
F = b1[0]*X1 +b1[1]*Y1+b1[2]*X1**2 + \
    b1[3]*Y1**2 + b1[4]
plt.contour(X1, Y1, F, [0], colors = 'k',
            linestyles = 'solid')
plt.show()
```

d) Find the number of correct predictions by CELR which ideally should be 331 out of 337 due to the 6 outliers. Explain any discrepancies.

```
Y_pred = logreg.predict(X)
print("Number of samples predicted correctly is ", \
    format(np.sum(Y_pred == y),'d'))
```

8.5 (PROJECT) Adjust the code above to include regularization using the ridge penalty.

a) Try using the following code in place of their counterparts in the above project. This is the default setting for *C*.

```
logreg = LogisticRegression(C=1.0)
logreg.fit(X, y)
```

b) Next, try different values of *C* such as 4.0, 1.0, 0.4, and 0.1. You can plot each of the decision boundaries using the plot routine of the previous project. What value of *C* provides the best results?

8.6 (PROJECT) In this project, the goal is to write and execute the code for LCLR. It requires the logistic function, a routine for LCLR, and an iterative loop to implement gradient descent. The partial derivatives of the form given in Eq. (8.4.12) will be needed. Also, the code below assumes that all the previous projects have been executed.

a) Define the log-cosh function and the LCLR routine.

```
logcosh = lambda x: np.logaddexp(x, -x) - np.log(2)
def LCLR(b):
    n = X.shape[0]
    p = 1./(1.+np.exp((-np.dot(X.T,b))))
    diff = y-p
    dbi = -p*(1-p)*np.tanh(y-p)/n
    db = np.dot(X,dbi.T)
    cost = np.sum(logcosh(y-p))/n
    return cost,db
```

b) Define the $\boldsymbol{X}$ and $\boldsymbol{y}$ data and then run the iterative loop for gradient descent to convergence. Plot the results.

```
Xd = train_df.drop(["flag"],axis=1)
X = np.array(Xd.T)
y1 = train_df["flag"]
y = np.array(y1)
learning_rate = 0.1
b = np.array([ 0.01,0.02,0.03,0.04,0.05])
T = 30 # 30 iterations of gradient descent
iterations = np.zeros(T)
for t in range(T):
    loss,db = LCLR(b)
    iterations[t] = loss
    b = b - learning_rate * db.T
b2 = b
plt.plot(iterations)
plt.show()
```

c) Plot the results of both CELR and LCLR. Compare the results noting that no regularization was used on LCLR.

```
plt.figure() # Create a new figure window
set1 = train_df[train_df['flag']==1]
set0 = train_df[train_df['flag']==0]
plt.scatter(set1['xval'],set1['yval'],color='red')
plt.scatter(set0['xval'],set0['yval'],color='blue')
xlist = np.linspace(-2.0, 2.0, 100)
ylist = np.linspace(-2.0, 2.0, 100)
X1,Y1 = np.meshgrid(xlist, ylist)
F = b1[0]*X1 +b1[1]*Y1+b1[2]*X1**2 + \
    b1[3]*Y1**2 + b1[4]
plt.contour(X1, Y1, F, [0], colors = 'k',
            linestyles = 'solid')
F = b2[0]*X1 +b2[1]*Y1+b2[2]*X1**2 + \
    b2[3]*Y1**2 + b2[4]
plt.contour(X1, Y1, F, [0], colors = 'green',
            linestyles = 'dashed')
plt.title("LCLR = green, CELR = black")
plt.show()
```

d) Find the number of correct predictions by LCLR which ideally should be 331 out of 337 due to the 6 outliers. Explain any discrepancies.

```
p = 1./(1.+np.exp((-np.dot(X.T,b2))))
Y_pred = p > 0.5
print("Number of samples predicted correctly is ", \
    format(np.sum(Y_pred == y),'d'))
```

Reference

James, G., Witten, D., Hastie, T. et al. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer.

9

Neural Networks Using Log-Cosh

9.1 Introduction

Our objective in this chapter is to build on Chapter 8, which addressed logistic regression using the log-cosh loss function, and investigate the feasibility of applying the log-cosh loss to neural networks. The question here becomes: Can log-cosh methods be effectively used in conjunction with neural networks? Since we already know that log-cosh methods are applicable to logistic regression, it naturally follows that they should be immediately applicable to neural networks. Therefore, the issue at hand is to determine the advantages of log-cosh neural networks over cross-entropy neural networks for machine learning and what new capabilities they may provide, if any. This is the main question to be explored in this chapter.

We begin with a brief history of neural networks (see Haykin 2009; James et al. 2023) and then describe the basic units and architecture in the next section, including the activation functions, which are the non-linear part of the units that allow neural networks to learn patterns and characteristics of the input data. Then, details of training neural networks will be described, along with the steps involved in computing gradients needed for training. The chapter concludes with examples of binary classification and a number of metrics typically used to assess accuracy.

9.2 A Brief History of Neural Networks

Work on neural networks (NNs) began in earnest in the mid-to-late 1980s and early 1990s by computer scientists in the pursuit of artificial intelligence (AI) systems. Neural networks were initially touted as a way to mimic the behavior and operation of neurons in the brain, although that connection is not fully embraced today. There was much excitement at the time regarding the potential of NNs to solve many unsolvable problems due to breakthroughs such as the use of the sigmoid function for activation in conjunction with the development of efficient methods for back propagation. But there were many technological barriers to its success at that time and it quickly fell out of favor. In particular, computers were relatively slow and the era of big data had not yet arrived. As skepticism grew, NNs were relegated to being a stagnant technology with no future and, as a result, lay dormant for many years. In the intervening years, the speed of computers increased significantly, new and efficient algorithms for neural networks

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.

Companion Website: www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

were developed, and the Internet emerged as a way to deliver large datasets and open-source software to users around the world.

By 2010, the neural network had emerged from its dormant state and made an impact in many important applications, most notably speech recognition, image processing, and natural language processing. In medical applications, it was able to quickly and accurately identify cancerous tissue and gained a strong foothold in radiology. Other advances in optimization methods and NN architectures with many layers of neurons were made in the same period. Some of the most difficult problems became tractable and were solved using these advanced techniques. The promise of AI systems of the 1980s was finally being realized. Collectively, these methods were rebranded as "deep learning." This term caught the attention of academics and practitioners alike and is now used to describe almost all neural networks in use today. The key driver from the early use of NNs to its ubiquitous presence today is the availability of big data. Images, video files, audio files, etc. are well-known forms of data for deep learning, but there are an uncountable number of other datasets that are also being processed using deep NNs. Eventually, NNs led to the development of large language models (LLMs) which are at the heart of recent generative AI systems used by hundreds of millions of people today.

Our purpose in this chapter is rather limited to a comparison of traditional cross-entropy NNs (CENNs) against robust log-cosh NNs (LCNNs). Here, we view NNs as simply multiple layers of logistic regression modified only by the activation functions used in each layer. If we have a single layer (i.e. no hidden layers), it is standard logistic regression. If we have two or three layers it is a shallow NN. If there are many layers of logistic regression, it is called a deep NN, which is used for deep learning. This view may be somewhat oversimplified but it is something to keep in mind as a good starting point to understand neural networks. The meaning of layers and networks will become clear later in this chapter but the basic idea is that complex relationships between the independent variables and the response variable can be realized by increasing the parameter space of the problem. Training of these types of networks is carried out using gradient descent but they are usually accelerated using *momentum* and *rmsprop*, which will be described when the Adam optimization technique is detailed in Chapter 10. However, the method of creating the gradient terms through back propagation will be described in this chapter. Then, a simple circular dataset from Chapter 8 will be used to compare and contrast LCNN and CENN. A more advanced application using the MNIST dataset will be used to compare the two methods in Chapter 10. Finally, in the last section, binary classification metrics such as the F_1 score and AUROC will be described.

Nomenclature: You may have heard the terms **sigmoid function** and **logistic function** and wondered about the difference between them. The logistic function and the sigmoid function are equivalent. Statisticians typically use the term "logistic function," while those in machine learning prefer "sigmoid function." Generally, the term "logistic function" is used when referring to equations or computing probabilities. In contrast, "sigmoid function" is used when discussing activation functions in neural networks.

9.3 Defining Neural Networks

In this section, we describe the basic building blocks of neural networks. These building blocks will be used to construct a prototypical neural network. As a simple way of thinking about neural networks, one can view it as a natural extension of logistic regression. Recall that in logistic regression (as described in Chapter 8), one of the problems was that in order to increase the feature space, a technique called polynomial expansion is needed. However, it is not clear how many terms are required in the expansion to obtain acceptable results. This is where neural networks can play a key role. Think of NNs as a way to increase the feature space by introducing many layers of logistic regression between the inputs and outputs. By doing so, we have a better chance to fit the data because there will be many parameters in the neural network to work with. But how many layers should be used and how should the NN be constructed? These and other questions will be addressed in this section.

9.3.1 Basic Computational Unit

Consider the diagram in Figure 9.1. This is another way to represent the operation of logistic regression. In this case, there are four independent variables, x_1, x_2, x_3, and x_4, plus the intercept term, b, and one dependent variable, y. Each input, x_i, is multiplied by its respective parameter, w_i, and summed together to form z as follows:

$$z = b + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4. \tag{9.3.1}$$

Then z is applied to the logistic function to produce the probability

$$a(z) = \frac{1}{1 + \exp(-z)}, \tag{9.3.2}$$

which is the same as the output, hence, $y = a(z)$.

For neural networks, we simply need to change the terminology. The computation involves a linear combination of the inputs and the application of an activation function. We say that, in this computational unit, the inputs are multiplied by their edge **weights**, w_i, and then a **bias**, b, is added before applying the **sigmoid** activation function, $a(z)$, to produce the output. The **activation function** is not necessarily the sigmoid function but can be any nonlinear function that operates on the linear combination of inputs to produce useful results. What we have just described is the basic unit of all neural networks: **an activated sum-of-products unit**. The only step remaining is to replicate this unit many times to construct a fully connected neural network.

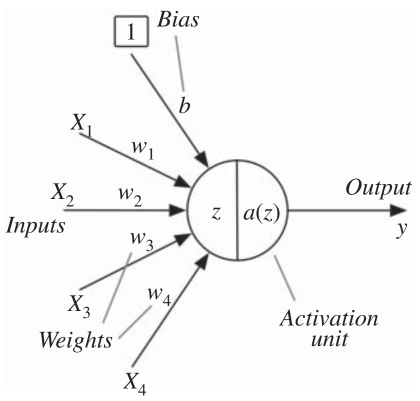


Figure 9.1 Computational unit (one neuron) in neural networks.

9.3.2 Four-Layer Neural Network

We are now ready to take the next step toward neural networks by increasing the number of layers, L, from $L = 1$ to some number $L > 1$. But how many layers should be added? This is where the art of designing neural networks comes into play. A variety of experiments may be needed to find the right architecture for a given application. There are some well-known architectures for certain problems but each data science problem may require some amount of tweaking to obtain a suitable architecture. We will not delve deeply into this aspect but rather focus on one specific architecture and how it operates.

First, we need to understand why we have layers in neural networks. To begin the discussion, we start with a training set with four real variables, x_1, x_2, x_3, and x_4, and one binary response, y, as shown in Figure 9.2. In the rest of this chapter, each entry in the main table will be referred to as x_{ij} and each response in the last column as y_i, for $i = 1, \dots, n$ and $j = 1, \dots, d$. In this case, $d = 4$. Suppose we know that a linear boundary may not be sufficient to separate the two classes for this binary classification problem. Then, for a nonlinear boundary, we could simply carry out a polynomial expansion for logistic regression until we achieved sufficient accuracy. Unfortunately, it is not straightforward to do this without some prior domain knowledge. For example, should we use a polynomial expansion of order 4 or 5? Should we include all powers and all the cross terms? Should we take various ratios of different variables? It is a matter of trial-and-error to obtain the right set of features if we wish to proceed in this manner.

A better approach is to use a neural network. If we take 12 computational units of the type shown in Figure 9.1 and arrange them as shown in Figure 9.3 and add one more at the output, the result is a four-layer neural network (with the bias terms not included to reduce clutter in the diagram). The units between the inputs and output are called *hidden units* and each column of hidden units is called a **hidden layer**. This NN has three hidden layers and one output layer comprising the four layers.[1] The architecture of this particular neural network is symmetric but actual networks can be highly asymmetric. That is,

i	x_1	x_2	x_3	x_4	y
1	0.306	0.898	0.239	0.902	1
2	0.294	0.863	0.176	0.871	1
3	0.361	0.835	0.125	0.863	1
4	0.310	0.902	0.235	0.741	0
5	0.298	0.871	0.173	0.314	1
6	0.365	0.863	0.122	0.302	1
7	0.314	0.741	0.235	0.173	1
8	0.302	0.733	0.173	0.122	0
9	0.369	0.745	0.122	0.235	0
10	0.318	0.757	0.239	0.173	0
⋮	⋮	⋮	⋮	⋮	⋮
n	0.235	0.173	0.310	0.902	0

Figure 9.2 Hypothetical training set for neural network.

1 Note that we are not counting the input as a layer since it is fixed, or alternatively it could be called layer 0. There is a minor debate in the NN community about how to count layers. For our purposes, the number of layers L is determined by the number of activation layers. In other words, the input does not count as a layer.

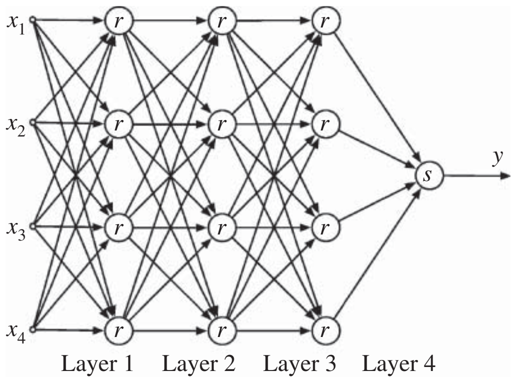


Figure 9.3 Four-layer neural network with relu activation (denoted as "*r*") for hidden layers and sigmoid activation (denoted as "*s*") for the output layer (bias terms are not shown). This is a fully connected architecture.

the number of units in each layer may vary considerably, and the number of layers used depends on the characteristics of the application and the associated dataset.

The question that arises is, what is the benefit of doing this? First, the number of parameters has increased dramatically. This allows the NN to learn more about inherent relationships in the data and to look for potential solutions in very high dimensions. Second, every output of the computational units is a nonlinear function of their respective inputs. The nonlinearity in the different layers is the mechanism by which the neural network is able to produce complex solutions. Without the nonlinearity, it would be one big linear function with no real advantage over a logistic regression approach. In other words, the key to the neural network is parameter-space expansion and nonlinearity in all computational units.

We call this a *fully connected* neural network because every output of a prior layer connects to every input of a subsequent layer. In some sense, it seems like a waste of resources. But when the initial values are set (usually to very small random non-zero values), each one produces a different result. Therefore, we prefer to keep the fully connected structure and let the neural network figure out which ones to use. We could remove some of them if we wanted to but it is not clear which ones to keep and which ones to kill. *Dropout* is a heuristic approach that randomly keeps and kills neurons in each cycle for regularization purposes.

Each arrow in the diagram of Figure 9.3 represents a different parameter. We can easily count the number of parameters associated with the four-layer neural network. There are 3 layers with 20 parameters per layer (16 arrows plus 4 more for the bias terms, which are not shown), and the output layer has $4 + 1$ parameters, for a total of 65 parameters. We have greatly increased the parameter space from 5 to 65 using this neural network architecture. Unfortunately, this architecture with 65 parameters could lead to overfitting. As we increase the depth of the neural network and the number of hidden units in each layer, the number of parameters in the network (and the likelihood of overfitting) increases considerably. Therefore, regularization using dropout is needed to counteract this effect.

9.3.3 Activation Functions

The nonlinearity of the neural network is provided by the activation functions at the output of each unit. However, the sigmoid function is not the only option. In fact, there are a host of other choices for this function: relu(z), leaky-relu(z), tanh(z), elu(z), linear(z), softmax(z), maxout(z), and so on. We will focus on the most widely used activation functions in this chapter, namely, the sigmoid and relu functions.

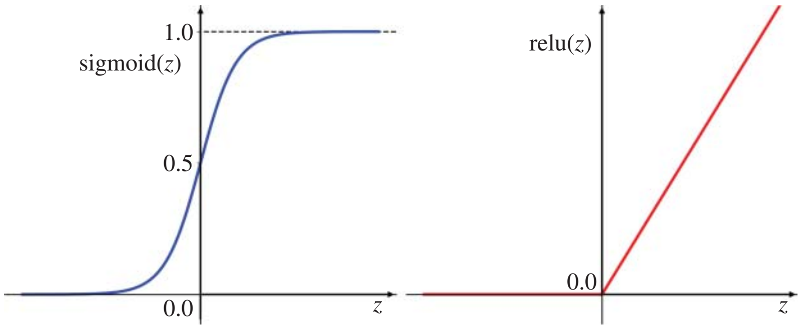


Figure 9.4 Sigmoid and relu activation functions. Typically, relu is applied to hidden units to speed up convergence whereas sigmoid is applied to the output layer to provide probabilities.

The sigmoid function is given by

$$\text{sigmoid}(z) = a(z) = \frac{1}{1+\exp(-z)}, \tag{9.3.3}$$

as shown on the left side of Figure 9.4. It takes the linear combination of the inputs, which can be any real value, and forces it to lie in the range [0,1]. We will need the derivative of the activation function for use in gradient descent during optimization. The derivative of the sigmoid function is

$$\frac{\partial \text{sigmoid}(z)}{\partial z} = \frac{\exp(-z)}{(1+\exp(-z))^2} = a(z)(1-a(z)). \tag{9.3.4}$$

We could use the sigmoid function for activation throughout the neural network but unfortunately convergence will be very slow during optimization. This is due to the two flat regions far away from the origin as shown in Figure 9.4. These near-zero derivatives present major convergence problems in cases when large values of z (positive or negative) arise in the network. Convergence will be very slow since the gradients are said to "vanish" when the magnitude of z is large. The structures in the optimization landscape that lead to slow convergence are often characterized as either "plateaus" or "ravines" (see Chapter 10 for further details). However, visualizing them is a challenge as they occur in a multidimensional space. In any case, a different approach is needed to speed up the rate of convergence.

The most popular alternative to the sigmoid function is the **rectified linear unit** (relu) function.[2] It is widely used in the hidden layers of neural networks. In Figure 9.3, all nodes labeled "r" use the relu activation, while those labeled "s" use the sigmoid activation. The relu function is composed of two linear segments, as shown on the right in Figure 9.4, that can be represented as follows:

$$\text{relu}(z) = zI(z) = \begin{cases} 0 & \text{if } z \le 0 \\ z & \text{if } z > 0 \end{cases},$$

where $I(z)$ is the indicator function, which is often used for mathematical convenience, given by

$$I(z) = \begin{cases} 0 & \text{if } z \le 0 \\ 1 & \text{if } z > 0 \end{cases}.$$

2 Often, the acronym ReLU is used rather than relu. We will use relu in this chapter.

For gradient calculation, the relu function has a derivative that reduces to

$$\frac{\partial \text{relu}(z)}{\partial z} = I(z) = \begin{cases} 0 & \text{if } z \le 0 \\ 1 & \text{if } z > 0 \end{cases}.$$

The relu function has a slope of 1 in cases where $z > 0$. So the optimization method does not get trapped in the aforementioned plateaus and ravines as readily as in the case of the sigmoid function.

There may be a concern that this relu function is too simple to be useful. In contrast, relu works extremely well with gradient descent and is straightforward to implement. The relu function is used instead of the sigmoid function in all neurons except the output layer. The problem with using the relu function at the output is that it can take on values greater than 1. Once the a relu output exceeds 1, it can no longer be viewed as a probability. Since classification outputs must lie in the range [0,1], a sigmoid activation is required at the output.

Note: The subject matter in the next two sections is very important. If you can follow the descriptions in detail, you will have a deeper understanding of how neural networks build a model through a process called **training**: i.e. finding an appropriate a set of values for the weights and biases via optimization. It is the core idea that allows training of neural networks to proceed in an orderly and efficient manner.

9.4 Training of Neural Networks

In order to train multilayered neural networks, an approach based on gradient descent is used. This is a numerical method that involves finding the minimum of a convex region of the objective function using an iterative approach. Essentially, given an initial guess of the weights and biases, the slope of the objective function at that point is used to decide which direction to take for the next step. So we will need to take partial derivatives of the objective function with respect to every parameter in the network. The size of the step taken is controlled by the learning rate, ℓ. If the step is too small, many iterations will be required. If too large, the process may not converge at all. After a proper selection of ℓ, the objective function is evaluated, gradients are computed, and parameters are updated in a repeated loop until the iterations converge to some nearby minimum.

While this approach is theoretically appealing, the landscape of the optimization space will dictate the number of iterations needed to converge. The true nature of the solution space is hard to grasp since the number of dimensions is very high. In particular, it is not a convex optimization problem since there may be many local minima. Adaptive learning rate adjustments can only go so far in navigating a complex configuration space. Many different learning rates must be tried before a suitable one is found. But even with a suitable learning rate adjustment strategy, the optimization procedure for neural networks may get stuck in ravines with sharp edges on each side or plateaus with flat regions in all directions. It may take hundreds of thousands of iterations to escape the ravine or find the edge of a plateau, if it happens at all.

To address this type of problem, a well-known improvement to gradient descent called **momentum** has been developed to speed up convergence in difficult solution spaces. It can be enhanced with another method called **rmsprop**. The Adam optimizer contains these methods as well as other heuristic techniques to improve convergence of neural networks. We will explore the details of the Adam optimizer in Chapter 10. For now, the goal is simply to understand the basics of neural network optimization.

The two objective functions to be considered are the cross-entropy loss and the log-cosh loss. The cross-entropy loss function is given by

$$J^{\mathrm{CE}}(\boldsymbol{w}, b) = -\sum_{i=1}^{n}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]. \tag{9.4.1}$$

The log-cosh loss function is given by

$$J^{\mathrm{LC}}(\boldsymbol{w}, b) = \frac{1}{a}\sum_{i=1}^{n}\log(\cosh((a(y_i \; - p_i))). \tag{9.4.2}$$

Both of these loss functions were derived for logistic regression in Chapter 8. In this chapter, the two will be compared relative to their application in the training of neural networks. The two loss functions may be represented as functions of p_i, $\boldsymbol{w}$ and/or $\boldsymbol{b}$. Therefore, we may use any of $J(\boldsymbol{w}, \boldsymbol{b})$, $J(\boldsymbol{w})$, $J(\boldsymbol{b})$, or $J(p_i)$, depending on the circumstances.

Assume that we want to train a one-layer neural network. Then, $\boldsymbol{w}$ is a vector of weights and b is a scalar bias. For gradient descent, we start with some initial values, $\boldsymbol{w}^{(0)}$ and $b^{(0)}$, then take a step toward the minimum, and then repeat this in each iteration until we reach the minimum. The weights and biases are updated at each iteration using some fraction of the gradient direction, controlled by ℓ, as follows:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \ell \times \left(\frac{\partial J(\boldsymbol{w}, b)}{\partial \boldsymbol{w}}\right)^{(t)}, \tag{9.4.3}$$

$$b^{(t+1)} = b^{(t)} - \ell \times \left(\frac{\partial J(\boldsymbol{w}, b)}{\partial b}\right)^{(t)}. \tag{9.4.4}$$

Here, t is the iteration count that starts at 0. At the end of the training process, we obtain the final weights and biases which represents the complete model. For multi-layer neural networks, the process is much more involved, as described in the next section.

9.5 Forward and Backward Propagation

The best way to understand the details of training a multilayered neural network is through a worked example. Suppose we have a training set with n rows and two data columns for the explanatory variables and one column for the binary response variable. We decide to build a model using a neural network with one hidden layer comprised of 4 relu units and an output layer with 1 sigmoid unit, as depicted in Figure 9.5. The training begins with some initial random values for the model parameters and then proceeds with forward and backward propagation to update the model in an iterative fashion until satisfactory convergence is achieved. Each pass through this network involves six steps for this small example: four steps in forward propagation to compute a new output, and then two steps for back propagation where the parameters are adjusted using gradients that are based on the size of the errors at the output. Before sequencing through the steps, the issue of gradient calculation is addressed.

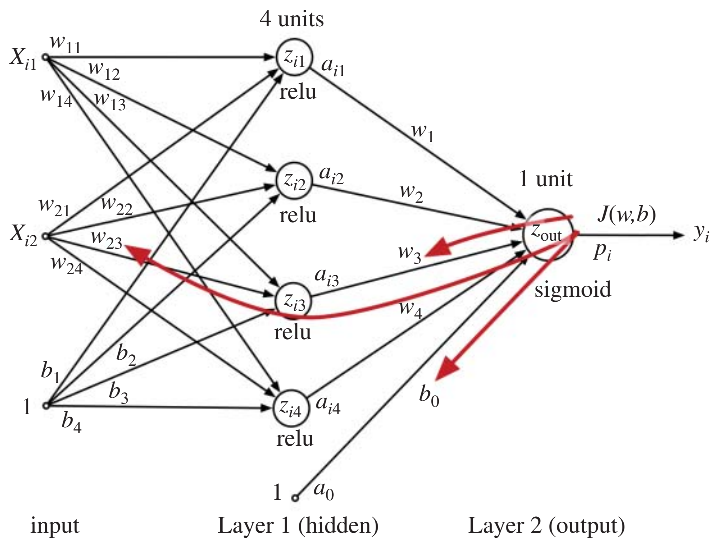


Figure 9.5 Back propagation in a two-layer neural network. In this example, w_3, w_{23}, and b_0 are to be computed using backprop.

In order to produce the full set of gradients during backward propagation, we must compute a gradient for every parameter inside the network with respect to the loss function. More specifically, starting with the loss function at the output, we must produce five gradients for layer 2, i.e. one for each of the weights $w_1, \dots, w_4$, and one for the bias, b_0. For layer 1, eight gradients must be computed for weights $w_{11}, w_{12}, w_{13}, \dots, w_{24}$, and four gradients for biases b_1, b_2, b_3, and b_4. The weights and biases are organized as matrices and vectors for ease of storage and retrieval, and to facilitate vectorized computations. A total of 17 gradients are needed for this small network but the question becomes, how do we scale this up for a large neural network with many layers to enable deep learning?

Back propagation is a formalized procedure to compute the vectors and matrices of gradients by starting at the output of the neural network and working backwards to the input, hence its nickname, **backprop**. The basic idea of backprop is to use the chain rule to systematically compute the gradients. We simply need to apply it recursively as we traverse the entire network backwards. However, the number of paths grows exponentially as the number of layers increases and this type of computation along all paths would make deep learning unfeasible. To address this problem, the backprop algorithm stores data during a forward propagation step, and then re-uses that stored data, and any newly computed data, during backward propagation to compute the gradients. In other words, the backprop algorithm is a very efficient way to store and retrieve partial data when computing gradients. This is what makes deep learning computationally feasible.

9.5.1 Forward Propagation

During forward propagation, an input $\boldsymbol{x}_i$ is applied to the neural network and propagated to the output to produce p_i, as in Figure 9.5. This is accomplished through a series of matrix–vector computations using the most recent set of parameter values. Note that all weights and biases are initialized to small random numbers before training begins. The entire sequence of operations will now be described for the neural network of Figure 9.5. In the description to follow, we will use j as the index for the input (two inputs), k as the index for hidden units (four units), and i for the row index of the training set (n rows). Consider the case when we apply input vector $\boldsymbol{x}_i = (x_{i1}, x_{i2})^{\top}$ with a known response y_i. In forward propagation, the following four steps are executed.

Step 1: First, the input values are multiplied by a weight matrix $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$ is added to produce $\boldsymbol{z}_i$ as follows:

$$\boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}, \tag{9.5.1}$$

where

$$\boldsymbol{z}_i = \begin{bmatrix} z_{i1} \\ z_{i2} \\ z_{i3} \\ z_{i4} \end{bmatrix}, \quad \boldsymbol{W} = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

Step 2: The resulting vector $\boldsymbol{z}_i$ is "activated" with relu to produce vector $\boldsymbol{a}_i$ as follows:

$$\boldsymbol{a}_i = \text{relu}(\boldsymbol{z}_i). \tag{9.5.2}$$

Step 3: To produce the raw output, $z_{\text{out,i}}$, we simply take the dot product of $\boldsymbol{a}_i = (a_{i1}, a_{i2}, a_{i3}, a_{i4})^{\top}$ and $\boldsymbol{w} = (w_1, w_2, w_3, w_4)^{\top}$ and add the bias b_0 as follows:

$$z_{\text{out,i}} = \boldsymbol{w}^{\top}\boldsymbol{a}_i + b_0 = w_1 a_{i1} + w_2 a_{i2} + w_3 a_{i3} + w_4 a_{i4} + b_0. \tag{9.5.3}$$

Step 4: The raw output is passed through the sigmoid activation function to produce p_i as follows:

$$p_i = \text{sigmoid}(z_{\text{out,i}}) = \frac{1}{1 + \exp(-z_{\text{out,i}})}. \tag{9.5.4}$$

Exercise: In Figure 9.5, what are the expressions for z_{i1} and a_{i1}, given an input of $(x_{i1}, x_{i2})^{\top}$?

Ans.: The expression for z_{i1} is given by $z_{i1} = w_{11}x_{i1} + w_{21}x_{i2} + b_1$. Then a_{i1} is the output of the relu unit given by $a_{i1} = \text{relu}(z_{i1}) = z_{i1}I(z_{i1})$. □

9.5.2 Backward Propagation

The procedure used in backward propagation is somewhat more elaborate than forward propagation but it is conceptually straightforward. Initially, the value of the predicted output, p_i, produced by forward propagation, may deviate considerably from the target value of y_i. A loss function, defined with y_i and p_i, is minimized to produce the final model by adjusting the weights and biases in the entire network using gradient descent.

To adjust the parameter values, we must compute the gradients of the loss function relative to all 17 parameters of the network and then apply one step of gradient descent. The gradients are computed using the chain rule backward through the network. Rather than computing all the gradients for this example, which would be lengthy and somewhat repetitive, we focus on three specific cases that are representative of all the other cases: w_3, b_0, and w_{23}. Each one is highlighted in the figure with an arrow that starts from the loss function at the output and traces a path backwards to the weight or bias of interest. The arrows represent the sequence of partial derivatives that are needed for the gradient computation, as will be seen shortly.

The process begins by finding the partial derivatives of the cross-entropy loss function with respect to each of the parameters. The loss function for the ith row of the training set is given by

$$J^{\text{CE}}(p_i) = -y_i \log(p_i) - (1 - y_i)\log(1 - p_i). \tag{9.5.5}$$

Consider weight w_3 as the first parameter of interest to us since it is directly related to the output. The question is, how should we change w_3 to reduce the loss? The answer is to find its associated gradient, which provides information about the amount and direction of the change. To obtain the gradient for w_3, we use the chain rule:

$$\frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_3} = \frac{\partial J^{\text{CE}}(p_i)}{\partial p_i}\frac{\partial p_i}{\partial z_{\text{out,i}}}\frac{\partial z_{\text{out,i}}}{\partial w_3}. \tag{9.5.6}$$

The arrow associated with w_3 in the diagram of Figure 9.5 refers to this chain rule of partial derivatives. Using the cross-entropy loss and sigmoid functions, the needed partial derivatives are multiplied to obtain

$$\frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_3} = \left(-\frac{y_i}{p_i} + \frac{1 - y_i}{1 - p_i}\right) p_i(1 - p_i)a_{i3} = -(y_i - p_i)a_{i3}. \tag{9.5.7}$$

Then, considering all n rows of the training set,

$$w_3^{(t+1)} = w_3^{(t)} - \ell \times \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_3} = w_3^{(t)} - \ell \times \sum_{i=1}^{n}(y_i - p_i)(-a_{i3}). \tag{9.5.8}$$

Now we know exactly how to change w_3 in each iteration.

The general expression for all four weights connected to the output is

$$w_k^{(t+1)} = w_k^{(t)} - \ell \times \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_k} = w_k^{(t)} - \ell \times \sum_{i=1}^{n}(y_i - p_i)(-a_{ik}), \tag{9.5.9}$$

for $k = 1, 2, 3, 4$, which are the indices for the hidden units.

The update equation for the bias term b_0 can be obtained with minor changes to the above equation. Specifically, the arrow in Figure 9.5 associated with b_0 has a fixed input value of $a_0 = 1$. After using this value in the above equation, the update equation for b_0 becomes

$$b_0^{(t+1)} = b_0^{(t)} - \ell \times \frac{\partial J^{\text{CE}}(b_0)}{\partial b_0} = b_0^{(t)} - \ell \times \sum_{i=1}^{n}(y_i - p_i)(-1). \tag{9.5.10}$$

In summary, the weight and bias update equations are (Step 5)

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}^{(t+1)} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}^{(t)} - \ell \begin{bmatrix} \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_1} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_2} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_3} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_4} \end{bmatrix},$$

and

$$b_0^{(t+1)} = b_0^{(t)} - \ell \times \frac{\partial J^{\mathrm{CE}}(b_0)}{\partial b_0}.$$

This completes the update equations for the second layer.

The next goal is to derive the update equations for the first layer. We use w_{23} for illustration purposes but the same steps apply to all the other gradients in that layer. We pose the same question as before: How should we change w_{23} to reduce the loss? The answer is somewhat more complicated because we must account for all the sum-of-products and activation functions between w_{23} and the loss function at the output. Equivalently, we can work backward from the output to w_{23}, as indicated by the long back arrow of Figure 9.5. In particular, it takes a path backward from the output loss function through unit z_{i3} to w_{23}. The corresponding chain rule for the ith row is as follows:

$$\frac{\partial J^{\mathrm{CE}}(\boldsymbol{w})}{\partial w_{23}} = \frac{\partial J^{\mathrm{CE}}(p_i)}{\partial p_i} \frac{\partial p_i}{\partial z_{\mathrm{out,i}}} \frac{\partial z_{\mathrm{out,i}}}{\partial a_{i3}} \frac{\partial a_{i3}}{\partial z_{i3}} \frac{\partial z_{i3}}{\partial w_{23}}. \tag{9.5.11}$$

The first two terms have been computed already during the calculation of the second layer gradients so this information would be saved for reuse. The third term is just the weight associated with a_{i3} which is w_3, as evident from Eq. (9.5.3). The fourth term depends on the type of activation function in layer 1, which in this case is a relu function. The relu derivative is either 1 or 0, since it is a “keep or kill” term that depends on the value of z_{i3}. We need an expression for a_{i3} to obtain this derivative. From Figure 9.5, the forward arrows pointing into z_{i3} in layer 1 imply that

$$z_{i3} = w_{13}x_{i1} + w_{23}x_{i2} + b_3 \tag{9.5.12}$$

and, therefore, using the definition of relu given earlier, we can write

$$a_{i3} = \mathrm{relu}(z_{i3}) = z_{i3}\, I(z_{i3}), \tag{9.5.13}$$

where $I(\cdot)$ is the indicator function. Then, using the fact that $\partial a_{i3}/\partial z_{i3} = I(z_{i3})$ and $\partial z_{i3}/\partial w_{23} = x_{i2}$, which is the fifth and final term of Eq. (9.5.11), we can compute the terms

$$\frac{\partial J^{\mathrm{CE}}(p_i)}{\partial p_i} \frac{\partial p_i}{\partial z_{\mathrm{out,i}}} \frac{\partial z_{\mathrm{out,i}}}{\partial a_{i3}} = -(y_i - p_i)w_3 \quad \text{and} \quad \frac{\partial a_{i3}}{\partial z_{i3}} \frac{\partial z_{i3}}{\partial w_{23}} = I(z_{i3})x_{i2}. \tag{9.5.14}$$

The equation on the left can be precomputed and saved while processing layer 2, while the equation on the right is a new computation specifically for w_{23}. At a basic level, this is how backprop saves time in the gradient computations—by storing precomputed data for later use.

Multiplying the terms together, the final result for all n rows is

$$\frac{\partial J^{\mathrm{CE}}(\boldsymbol{w})}{\partial w_{23}} = \sum_{i=1}^{n} (y_i - p_i) w_3 I(z_{i3})(-x_{i2}). \tag{9.5.15}$$

This looks complicated because of the indicator function but recall that it is either 0 or 1. In either case, the computation is rather straightforward.

The other gradients in the first layer are computed in the same manner. Typically, a gradient matrix would be assembled for the first layer, with each weight between input j and hidden unit k, given by

$$\frac{\partial J^{\mathrm{CE}}(\boldsymbol{w})}{\partial w_{jk}} = \sum_{i=1}^{n} (y_i - p_i) w_k I(z_{ik})(-x_{ij}), \tag{9.5.16}$$

where $j = 1, 2$ and $k = 1, 2, 3, 4$. Using the steps outlined above, all eight gradients can be calculated to update the $\boldsymbol{W}$ matrix (Step 6) as follows:

$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}^{(t+1)} = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}^{(t)} - \ell \begin{bmatrix} \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{11}} & \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{21}} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{12}} & \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{22}} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{13}} & \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{23}} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{14}} & \frac{\partial J^{\text{CE}}(\boldsymbol{w})}{\partial w_{24}} \end{bmatrix},$$

and likewise the four gradients to update the bias vector

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}^{(t+1)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}^{(t)} - \ell \begin{bmatrix} \frac{\partial J^{\text{CE}}(\boldsymbol{b})}{\partial b_1} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{b})}{\partial b_2} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{b})}{\partial b_3} \\ \frac{\partial J^{\text{CE}}(\boldsymbol{b})}{\partial b_4} \end{bmatrix}.$$

This constitutes one iteration of gradient descent. The whole process is repeated starting with Step 1 and concluding with Step 6 until convergence is reached.

9.5.3 Log-Cosh Gradients

For the log-cosh loss, a similar set of operations are needed for back propagation. We note that, aside from the objective function used at the output, the rest of the partial derivative calculations are identical in the network. This makes it very easy to replace the CE loss with LC loss in existing machine learning programs. Referring back to Figure 9.5, to obtain b_0, we would use the gradient

$$\frac{\partial J^{\text{LC}}(b_0)}{\partial b_0} = \frac{\partial J^{\text{LC}}(p_i)}{\partial p_i} \frac{\partial p_i}{\partial z_{\text{out,i}}} \frac{\partial z_{\text{out,i}}}{\partial b_0} = \sum_{i=1}^{n} p_i(1 - p_i) \tanh(a(y_i - p_i))(-1), \tag{9.5.17}$$

To obtain w_k, we would use the gradient

$$\frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_k} = \frac{\partial J^{\text{LC}}(p_i)}{\partial p_i} \frac{\partial p_i}{\partial z_{\text{out,i}}} \frac{\partial z_{\text{out,i}}}{\partial w_k} = \sum_{i=1}^{n} p_i(1 - p_i) \tanh(a(y_i - p_i))(-a_{ik}). \tag{9.5.18}$$

Even though these expressions look more expensive, the terms need only be computed once and then passed backward through the network. The update equation for the log-cosh weights is then (Step 5):

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}^{(t+1)} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}^{(t)} - \ell \begin{bmatrix} \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_1} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_2} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_3} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_4} \end{bmatrix}.$$

and the bias update equation is:

$$b_0^{(t+1)} = b_0^{(t)} - \ell \frac{\partial J^{\text{LC}}(b_0)}{\partial b_0}.$$

For w_{23}, the same type of formulation is used as for CE except for the use of the LC loss function. The needed partial derivatives for the ith row are

$$\frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{23}} = \frac{\partial J^{\text{LC}}(p_i)}{\partial p_i} \frac{\partial p_i}{\partial z_{\text{out,i}}} \frac{\partial z_{\text{out,i}}}{\partial a_{i3}} \frac{\partial a_{i3}}{\partial z_{i3}} \frac{\partial z_{i3}}{\partial w_{23}}. \tag{9.5.19}$$

Then, as before, we can assemble the matrix update equation associated with the first layer parameters as follows (Step 6):

$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}^{(t+1)} = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}^{(t)} - \ell \begin{bmatrix} \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{11}} & \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{21}} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{12}} & \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{22}} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{13}} & \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{23}} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{14}} & \frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial w_{24}} \end{bmatrix}$$

and the biases are updated using

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}^{(t+1)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}^{(t)} - \ell \begin{bmatrix} \frac{\partial J^{\text{LC}}(\boldsymbol{b})}{\partial b_1} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{b})}{\partial b_2} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{b})}{\partial b_3} \\ \frac{\partial J^{\text{LC}}(\boldsymbol{b})}{\partial b_4} \end{bmatrix}.$$

When we examine the expressions for gradients required for either CE or LC in this example, it is clear that there is a natural structure and repeated use of partial derivatives as the network is traversed in the reverse direction. This is the core idea of backprop: it relies on the fact that each layer requires partial derivative terms from a previously computed layer. Simply stated, backprop is an efficient implementation of gradient calculations facilitated by the storage and reuse of precomputed partial derivatives. Furthermore, with the advent of graphics processing units (GPUs) in recent years, the matrix–vector operations of forward and backward propagation can be vectorized to greatly reduce computation time. The tasks can also be parallelized to a great extent to further reduce the training time. The combination of backprop, GPUs, and parallelism has made deep learning possible and eventually led to the generative AI capabilities embodied in ChatGPT.

9.6 Cross-entropy and Log-Cosh Algorithms

We can now combine all the elements discussed thus far into simple algorithms for the cross-entropy neural network (CENN) and the log-cosh neural network (LCNN). The algorithms are provided in Algorithms 9.1 and 9.2.

In the setup phase, the algorithms begin by reading in the training data, $\boldsymbol{X}$ and $\boldsymbol{y}$. If the dataset is small, we can process all n rows together as one entity in each iteration. This is called **batch gradient descent** and each iteration is referred to as an **epoch**. For large datasets, especially those encountered in deep learning problems, the whole dataset is not usually processed all at once. Instead, we create small batches of equal-sized subsets of rows of the data and process them one small batch at a time.

Algorithm 9.1 Training procedure for cross-entropy neural network (CENN).

```
1: Input: training dataset $X \in \mathbb{R}^{n \times d}$, $y$;
2: Network: # of inputs = $d$, # of outputs=1,
3:    # of Layers($L$), # of units in each layer ($U_k$), $k = 1, ..., L$
4: Activation: layers $1, ..., L-1$ = relu, layer $L$ = sigmoid
5: Set # of epochs ($maxIter$), learning rate($\ell$)
6: Initialize weights $w_k^{(0)}$ and biases $b_k^{(0)}$ $k = 1, ..., L$
7: Set $t$=0
8: while $t < maxIter$ do
9:     Use forward propagation through network to compute $p_i$, $i = 1, ..., n$
10:    Use backprop to compute gradients, as follows:
11:    for k = L downto 1 do
12:        Compute gradients $\nabla w_k^{(t)}$ and $\nabla b_k^{(t)}$ of $J^{CE}(w, b)$
13:        Update weight estimates $w_k^{(t+1)} = w_k^{(t)} - \ell \times \nabla w_k^{(t)}$
14:        Update bias estimates $b_k^{(t+1)} = b_k^{(t)} - \ell \times \nabla b_k^{(t)}$
15:    end for
16:    Compute loss function
           $J^{CE}(w, b) = -\sum_{i=1}^{n}[y_i \log((p_i)) + (1 - y_i)\log(1 - p_i)]$
17:    Update $t = t + 1$
18: end while
19: Output: model $w_k, b_k$ $k = 1, ..., L$;
```

This is called **mini-batch gradient descent** with a batch size of m. Typically, m is set to 32, 64, or 128, depending on the application. In each epoch, the mini-batches are processed sequentially. The process is repeated until convergence, which may require hundreds or thousands of epochs. In this context, m is a tunable hyperparameter.

In the extreme case, if we process one row of data at a time, it is referred to as **stochastic gradient descent** with a batch size of $m = 1$. Here, the word "stochastic" is reflective of randomness seen in the convergence behavior since the gradients may fluctuate wildly from row to row at the beginning of the training process. Eventually, the gradients settle down as the model stabilizes. This approach has been shown to be very effective in many deep learning applications.

In Algorithms 9.1 and 9.2, batch gradient descent is employed (i.e. all data is processed together in each epoch). The neural network architecture is defined by the number of hidden layers, L, and the number of units in each layer, U_k, $k = 1, 2, \dots, L$. The activation of the unit at the output is the sigmoid function while all hidden layer units use the relu function. Then, the two hyper-parameters are set: number of epochs (*maxIter*) and the learning rate (ℓ). Finally, the weights and biases are **initialized to some small random values**. This defines the starting point of the optimization and the randomization allows the neurons to pursue different learning objectives. Some initial conditions may lead to a local minima that results in an unsatisfactory model. Subsequent runs may be needed to obtain a better model.

Both algorithms enter a `while` loop and execute a series of forward and backward propagation steps to determine the parameter values in each epoch. The output is determined by a forward propagation

Algorithm 9.2 Training procedure for log-cosh neural network (LCNN).

```
1: **Input:** training dataset $\boldsymbol{X} \in \mathbb{R}^{n \times d}, \boldsymbol{y}$;
2: Network: # of inputs = $d$, # of outputs=1,
3:    # of Layers($L$), # of units in each layer ($U_k$), $k = 1, ..., L$
4: Activation: layers $1, ..., L-1$ = relu , layer $L$ = sigmoid
5: Set # of epochs ($maxIter$), learning rate($\ell$)
6: Initialize weights $\boldsymbol{w}_k^{(0)}$ and biases $\boldsymbol{b}_k^{(0)}$ $k = 1, ..., L$
7: Set $t$=0
8: **while** $t < maxIter$ **do**
9:     Use forward propagation through network to compute $p_i$, $i = 1, ..., n$
10:    Use backprop to compute gradients, as follows:
11:    for k = L **downto** 1 **do**
12:      Compute gradients $\bigtriangledown \boldsymbol{w}_k^{(t)}$ and $\bigtriangledown \boldsymbol{b}_k^{(t)}$ of $J^{\text{LC}}(\boldsymbol{w}, \boldsymbol{b})$
13:      Update weight estimates $\boldsymbol{w}_k^{(t+1)} = \boldsymbol{w}_k^{(t)} - \ell \times \bigtriangledown \boldsymbol{w}_k^{(t)}$
14:      Update bias estimates $\boldsymbol{b}_k^{(t+1)} = \boldsymbol{b}_k^{(t)} - \ell \times \bigtriangledown \boldsymbol{b}_k^{(t)}$
15:    **end for**
16:    Compute loss function
         $J^{\text{LC}}(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{a} \sum_{i=1}^{n} \log(\cosh(a(y_i - p_i)))$
17:    Update $t = t + 1$
18: **end while**
19: **Output:** model $\boldsymbol{w}_k, \boldsymbol{b}_k$ $k = 1, ..., L$;
```

step. The backprop operation involves computing the gradients for the weights and biases in the neural network and then taking one step of gradient descent. Each backprop pass requires starting at the output layer L and traversing the network backwards to layer 1 while computing the gradients, as specified in the algorithms (note: for convenience, gradients are represented using the notation $\bigtriangledown \boldsymbol{w}_k$, $\bigtriangledown \boldsymbol{b}_k$, $k = 1, \dots, L$). Finally, the loss function is computed before proceeding to the next epoch. The only difference between the two algorithms is whether CE or LC is used as the loss function, and the corresponding gradient computations. Otherwise, the algorithms are identical.

After the `while` loop is completed, the loss function can be plotted graphically for each epoch to ensure that proper values of ℓ and *maxIter* have been selected. The convergence plot should be smooth for batch gradient descent and should settle at a small value in the latter stages of the iterative process. If not, the process must be repeated with different values of initialization, ℓ and/or *maxIter*.

9.7 Example: Circular Dataset

Having established the two methods to be compared, along with the details of their implementation, we can proceed to the next step of using a synthetic dataset to understand the advantages and disadvantages of each approach. For demonstration purposes, the nonlinear classification problem from Chapter 8 will be used, in which the binary data is largely separable (except for a few outliers) by a circular or elliptical decision boundary. It is a two-dimensional problem, with variables x_1 and x_2, and a binary response, y. In that chapter, we compared cross-entropy logistic regression (CELR) and log-cosh logistic regression (LCLR). However, for both CELR and LCLR, we needed to specify x_1, x_2, x_1^2, and x_2^2 as input variables since the solution has a nonlinear boundary. In other words, in logistic regression, a suitable polynomial expansion of the variables must be provided to obtain satisfactory results.

Recall from Chapter 8 that, for support vector machines (SVMs), polynomial expansion is unnecessary because SVMs will find a complex boundary using the RBF kernel controlled by a hyperparameter called γ. Therefore, SVMs have a distinct advantage over CELR and LCLR in this regard. But SVMs have a shortcoming since they do not directly report probabilities which may be needed in many applications, and they do not easily extend to multi-class problems. Both logistic regression and neural networks provide probabilities and easily extend to multi-class problems, as will be shown later in Chapter 10.

The key advantage of neural networks over logistic regression is that they require only the input variables because the parameter space is very large due to the hidden layers and the hidden units in those layers. There may be many different combinations of x_1 and x_2 required to create a highly nonlinear boundary using logistic regression but neural networks will likely find such a boundary, given the right number of layers and hidden units. The trade-off here is that it will not be easy to determine the best architecture for the application, or what relationships exist between the variables and the trained model. If the neural network used x_1^2 and x_2^2 internally, for example, we will not be able to tell from the final model. The parameter values are related to the variables but it will be difficult to interpret the results. Furthermore, variable importance cannot be directly assessed from the trained model.

In any case, our objective at this stage is to compare fully connected neural networks based on CENN and LCNN methods on a dataset with a known circular/elliptical boundary. For this purpose, we will only examine the performance on a training set in this section, as we did in Chapter 8. The circular dataset can be handled using a NN architecture with one hidden layer with 24 units fed only by x_1 and x_2. We do not have to explicitly deal with the bias term either since it is handled internally. The 24 hidden units use the relu activation function while the unit in the output layer uses the sigmoid function.

The choice of 1 hidden layer with 24 units is based on knowledge about the problem itself. In order to obtain a round shape, a large number of units is required. In general, it is not easy to determine the number of hidden units needed, or the number of layers for that matter.[3] In this case, it is possible to capture the data in the center region with 24 hidden units. In practice, a number of different NN architectures should be attempted with different numbers of layers and hidden units, and then the best architecture can be selected among all the combinations tried. This can be done using trial-and-error or a grid search or advanced hyperparameter tuning algorithms.

The original dataset is shown in Figure 9.6(a). There are 337 points of which 6 are outliers. Note the six outliers at the top-left corner of the figure. As a result, the classifier should get at most 331 correct out of 337. The nonlinear decision boundary produced by a neural network is not easy to express in closed form so that it can be rendered in a plot. However, there is a practical solution to the problem. If thousands of random points are selected in the region of interest and used to predict the outcomes as either 0 or 1, the results can be plotted with two different colors. This is the approach taken for this case, as shown in Figure 9.6.

3 The website playground.tensorflow.org provides facilities to develop intuition about the number of layers and hidden units needed for the circular dataset.

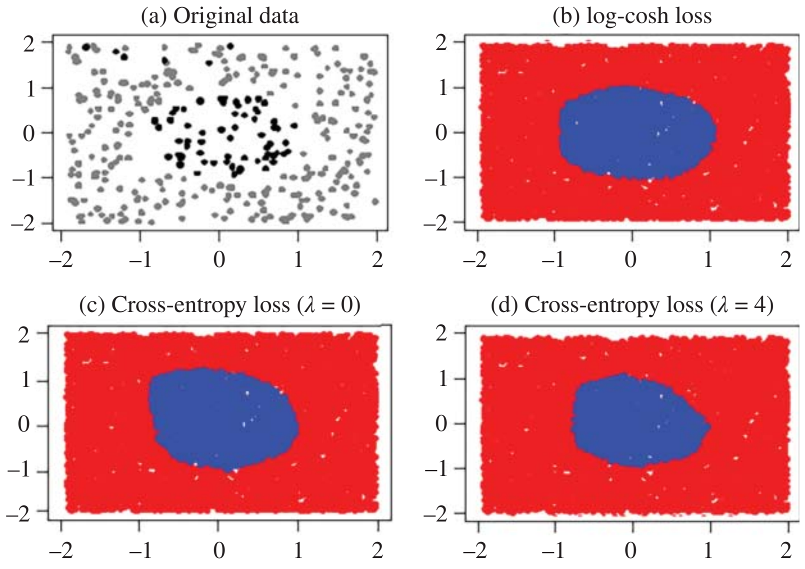


Figure 9.6 The circular dataset ($n = 337$, with 6 outliers). Part (a) shows the original dataset. Parts (b) LCNN, (c) CENN ($\lambda = 0$), and (d) CENN ($\lambda = 4$) show the nonlinear boundary of each method.

We see that LCNN finds a solution that separates the data in a manner as if there were no outliers, as shown in Figure 9.6(b). However, the same cannot be said for CENN as it distorts the decision boundary in the direction of the outliers, as shown in Figure 9.6(c). In other words, for this example, CENN is influenced by the outliers so the boundary is stretched upwards and to the left. At first glance, it may appear that CENN may not have converged to a final solution because of the irregularity of the boundary. However, if we check the convergence plots for LCNN and CENN (loss vs. epoch) shown in Figure 9.7, we see that both have properly converged to their solutions smoothly and monotonically.

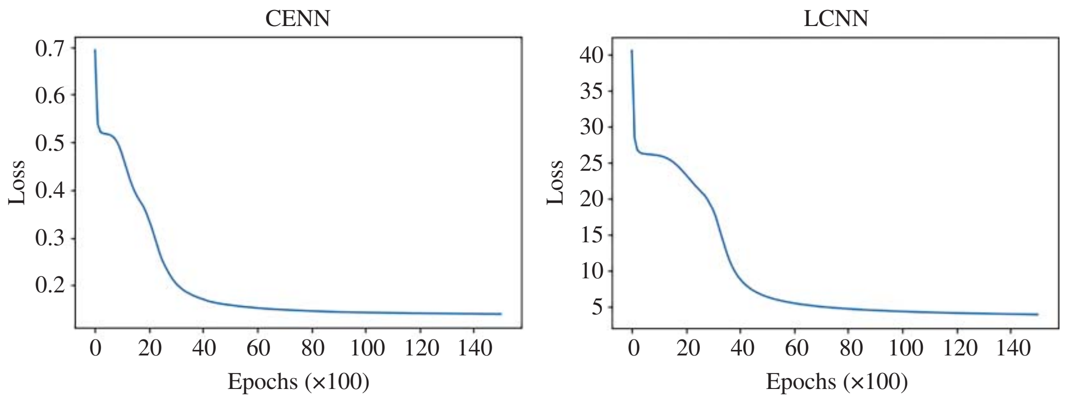


Figure 9.7 Convergence plots for CENN and LCNN for circular dataset.

The characteristics of the two convergence plots are rather interesting. In the case of CENN, there is a steep slope initially, followed by a small plateau, and then finally a gradual descent to the final value. Many convergence plots for CENN exhibit this type of behavior whereby the gradient descent method will get stuck in a narrow ravine and then eventually begin the drop to the minimum. A total of 15,000 epochs (i.e. iterations) were needed to reach convergence.

The LCNN convergence plot is also typical of log-cosh methods. Initially there is a sharp drop followed by a stall along a plateau and finally a more gradual descent to the minimum. LCNN also required about 15,000 epochs to reach convergence, although usually more are required compared to CENN. From the plots, we know that both CENN and LCNN did indeed reach their minimum values, but that CENN was not able to produce a satisfactory boundary in the same number of epochs.

The last step is to determine if a ridge penalty (see Chapter 5) would improve the results for CENN. If ridge is applied to CENN, the following objective function is obtained:

$$J^{\text{CE-ridge}}(\boldsymbol{w}) = -\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)] + \lambda\sum_{j=1}^{d} w_j^2, \tag{9.7.1}$$

where w_j represents each weight in the network and d is the total number of weights. Using the techniques described in Chapter 6, a suitable value of the regularization parameter is $\lambda = 4.0$. The improved result is shown in Figure 9.6(d). Beyond this value of λ, the solution gets worse. Hence, for CENN, ridge regression can be used to find a better model whereas LCNN does not require any regularization.

9.8 Classification Metrics and Outliers

We close this chapter by examining the metrics associated with binary classification problems. Assume we have a machine learning tool that performs binary classification on a training set to create a model which predicts the responses on the same training set. Although it is desirable to obtain a high level of prediction accuracy on the training set, the more important result is the accuracy on an unseen test set. In either case, to assess the accuracy of a binary classification model, we could simply take the number of correct predictions and divide by the total number of observations to determine the success rate, i.e.

$$\text{Accuracy} = \frac{\text{Total number of correct predictions}}{\text{Total number of observations}}.$$

However, this type of analysis, although useful as a first-order assessment, may be misleading if only a few of the true responses are, say 1, and all the remaining are 0. Then by guessing 0 in all cases, a high accuracy can be obtained but it would not be a proper indication of the accuracy of the model. Alternative methods are available to assess the quality of prediction in binary classification, as described below.

9.8.1 Precision, Recall, F_1 Score

When dealing with binary responses, it is common practice to compute the F_1 score on both the training set and test set. For this purpose, we need to build a table of outcomes based on whether or not the model predicted the correct label of 1 or 0. There are four possible scenarios as listed in Table 9.1. For example, if the true value is 1 and the prediction is 1, it is referred to as a true positive (tp). Similarly, a true value

Table 9.1 Interpretation of confusion matrix. The four possible outcomes of tn, fn, tp, or fp are accumulated in this tabular format from the ground truth and predicted binary values.

	True-0	True-1	Interpretation of Matrix Entries
Predicted-0	tn	fn	tn = true negative, fn = false negative
Predicted-1	fp	tp	fp = false positive, tp = true positive

of 0 and a predicted value of 0 is called a true negative (tn). On the other hand, an incorrect prediction is either a false positive (fp) when the true value is 0 and the model predicts 1, or a false negative (fn) when the true value is 1 and the model predicts 0. We say that the model is confused under these conditions.

One interesting question at this point is, how do outliers factor into these definitions? The answer at a conceptual level is provided in Figure 9.8. It shows a binary classification problem with a boundary generated by a classifier. Some of the points have been labeled to denote their type. Ideally, all points on the right of the boundary should be labeled tp and all those on the left should be labeled tn. However, this is not the case in this example. In fact, we can make a general statement that all outliers are either fp or fn. However, not all fp and fn are outliers. For example, the fp near the boundary line is a classification error. This can be corrected using a better classifier. Unfortunately, all the outliers will remain as fp or tn regardless of the classifier used. In this example, the best classifier will still produce two fp's and one fn. Therefore, outliers set an upper limit on the accuracy of classification methods. This is something to keep in mind when trying to produce better results using different classifiers with no apparent improvement: it could be that the upper limit due to outliers has been reached.

As shown in Table 9.1, the combination of true values along the diagonal and false values on the off-diagonals forms a confusion matrix. This matrix is very useful in telling us what type of error we are making using a particular classifier. One classifier may be prone to making a lot of false positive errors while another may make more false negative errors. Choosing the right classifier depends on the data science problem being solved. This is where the confusion matrix comes in handy because it contains the needed information. To obtain it, we can simply discretize each predicted probability using a threshold p_T and enter the totals into each of the four categories in the confusion matrix. Typically, $p_T = 0.5$ but it can be adjusted to improve performance. Using the discretized information, we compute the recall and precision, which are both standard metrics in binary classification. Finally, the F_1 score is computed using these two quantities to assess the overall effectiveness of the model.

In particular, the *recall* value, sometimes called the *sensitivity*, is given by

$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}, \tag{9.8.1}$$

while the *precision* value is given by

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}. \tag{9.8.2}$$

The recall quantifies what fraction of the predicted 1's are correct. The precision quantifies the fraction of true 1's that were predicted correctly. Then the F_1 score is computed using a combination of recall and

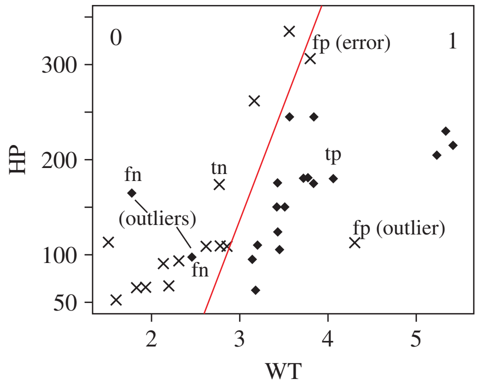


Figure 9.8 Examples of different logical cases involving outliers and errors.

precision as follows:

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \tag{9.8.3}$$

Exercise: Compute the accuracy, recall, precision, and F_1-score for the confusion matrix below.

Training set	**True-0**	**True-1**
Predict-0	485	108
Predict-1	64	234

Ans.: Accuracy = (tp + tn)/(tp + tn + fp + fn) = 0.807. Recall=tp/(tp + fn) = 0.684. Precision = tp/(tp + fp) = 0.785. Then F_1 = 2(0.684)(0.785)/(0.684 + 0.785) = 0.731. □

9.8.2 Receiver Operating Characteristics (ROCs)

It is possible to improve the performance of a classifier by simply adjusting the threshold, p_T. For example, in Figure 9.8, the decision boundary can be shifted a bit to the right by increasing p_T. Then, only outliers would not be properly classified, which is the best that can be achieved. Therefore, different values of p_T should be selected to find the optimal one for a given dataset to maximize the F_1 score.

An interesting issue arises when we have two classifiers, A and B, and we want to carry out a fair comparison between them. What if we adjusted p_T to improve the quality of classifier A while ignoring such adjustments for classifier B? That would be unfair but it may also be time consuming to adjust the p_T of both until all possible values are exhausted to find the best value for each one. The receiver operating characteristics (ROC) is an effective measure when comparing of the performance of two classifiers for different p_T settings. It is a plot of the true positive rate (TPR) against the false positive rate (FPR) for different values of p_T as follows. For each p_T, the TPR is given by

$$\text{TPR} = \frac{\text{tp}}{\text{tp} + \text{fn}}, \tag{9.8.4}$$

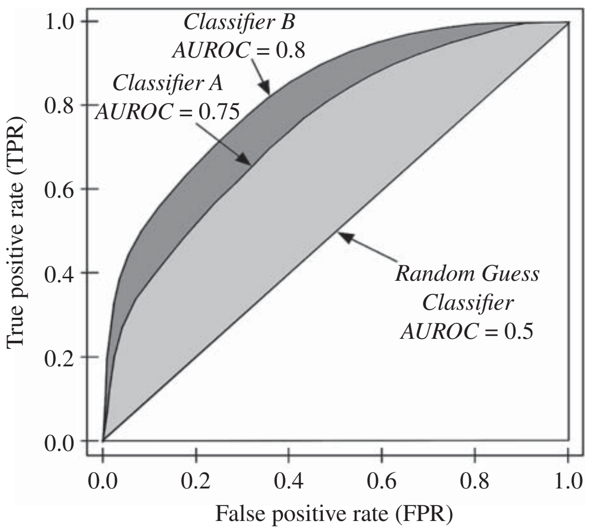


Figure 9.9 ROC curves of classifiers A and B, and a random guess classifier.

and the true negative rate (TNR), sometimes called the *specificity*, is given by

$$\text{TNR} = \frac{\text{tn}}{\text{tn} + \text{fp}}. \tag{9.8.5}$$

Then, the FPR is given by

$$\text{FPR} = 1 - \text{TNR} = 1 - \frac{\text{tn}}{\text{tn} + \text{fp}} = \frac{\text{fp}}{\text{tn} + \text{fp}}. \tag{9.8.6}$$

In Figure 9.9, we show a few hypothetical plots of TPR against FPR to illustrate the concepts discussed above. The trajectory of the plot starts from (0,0) and traces a path to (1,1). At (0,0), the classifier predicts all 0's, meaning no false positives, while at (1,1), the classifier predicts all 1's, meaning no false negatives. All other cases fall in between these two extremes as the ROC traces a path from one corner to the other. Three such paths are shown. First, the diagonal line represents the case of a random value classifier. That is, a classifier that simply guesses the outcomes would produce a diagonal line which represents a worst-case classifier. Second, the curve labeled A is the plot for classifier A that is clearly better than the random guess classifier since its trajectory lies above the diagonal line. Third, the curve labeled B is the plot for classifier B that is better than A because its curve is above A.

Example: Let p_T be the threshold for deciding whether a probability value from a classifier should be considered a 0 or a 1. Next, select different values of p_T starting at 0 and ending at 1 in steps of 0.01—a total of 101 points. For each p_T, the TPR and FPR are plotted to produce the ROC curve. The quality of a classifier can be established by finding the area under the ROC curve (AUROC). Three classifiers are compared in Figure 9.9. Specifically, the AUROC of a random guess classifier is 0.5. However, the total AUROC of classifier A is 0.75 whereas the total area under the ROC of classifier B is 0.8. Therefore, B is considered to be better than A. The AUROC can be loosely interpreted as the probability that the classifier has assigned the correct class to the data points. The ideal value under this interpretation is of course AUROC = 1.0. □

When AUROC is used with robust methods, the results can be misleading to some extent. For example, the AUROC of LCNN tends to be similar to CENN even though the F_1 results are better than CENN. In fact, the AUROC may not be a suitable metric in this type of comparison. Chapter 12 describes an alternative approach which is better suited for such a comparison.

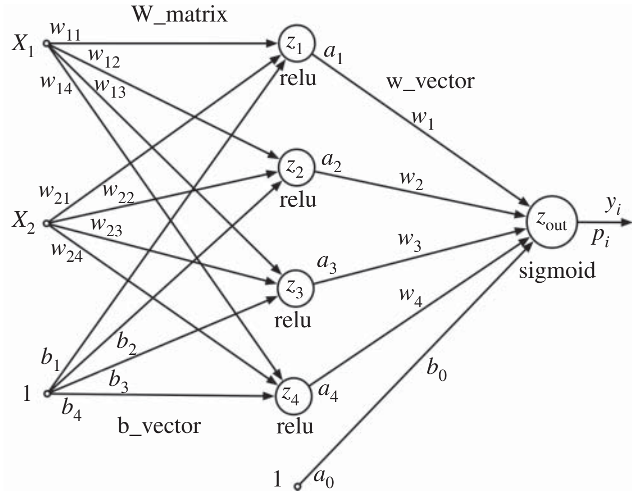
9.9 Summary

In this chapter, it has been shown that the log-cosh loss function is effective when used with neural networks. In fact, it is robust to outliers for binary classification problems. The cross-entropy loss commonly used in neural networks is affected by outliers but it can be made more tolerant with the use of the ridge penalty for regularization. However, the log-cosh loss was shown to have an auto-regularization feature such that penalty functions are not required. Chapters 10 and 12 take these concepts to several data science problems to illustrate their use in real-world applications. A number of other capabilities provided by using this robust loss function in neural networks will also be described in those chapters.

To close out this chapter, various metrics used to compare different binary classification tools were presented. Prediction accuracy by itself is not sufficient as an evaluation metric. Instead, metrics such as recall, precision, and F_1 score are more appropriate. In addition, the AUROC is also useful when comparing different classifiers. In Chapter 12, the effect of outliers on the AUROC metrics is considered in detail.

Problems

The following two-layer neural network will be used in all problems to follow. The labels are associated with Python code provided in the projects.



9.1 How many total parameters must be learned during training for the above network?

9.2 Find the expression for $\partial J^{\mathrm{LC}}(b_0)/\partial b_0$ for the log-cosh loss function. (Hint: derive Eq. (9.5.17) noting that the bias term is fixed at 1.)

9.3 Apply the chain rule to find the partial derivative term $\partial J^{\mathrm{LC}}(\boldsymbol{w})/\partial w_1$ for the log-cosh loss function. (Hint: this is similar to the derivation of Eq. (9.5.18).)

9.4 Apply the chain rule as in Eq. (9.5.19) to find the complete expression for $\partial J^{\mathrm{LC}}(\boldsymbol{w})/\partial w_{13}$ using the log-cosh loss function. (Hint: you will need to use the indicator function since the relu activation function is involved.)

9.5 (Project) This project involves the creation of a synthetic circular dataset and the implementation of CENN/LCNN algorithms to find a nonlinear decision boundary.

a) Import the libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

b) Build a circular dataset similar to Figure 9.6(a) using Python. Assign six data points to be outliers as shown in the upper region of the figure. The Python code below is suitable for this purpose. Enter this code and check the results in a Jupyter notebook.

```
m = 354
np.random.seed(42)
setx = np.random.random(m)*4.0-2.0
np.random.seed(30)
sety = np.random.random(m)*4.0-2.0
y = np.zeros(m)
for i in range(m):
    if (setx[i]**2 + sety[i]**2) < 0.9:
        y[i] = 0
    elif (setx[i]**2 + sety[i]**2) > 1.2:
        y[i] = 1
    else:
        y[i] = -1
    counter = 0
outliers = 6 # number of outliers
for i in range(100):
    if sety[i] > 1.6 and counter < outliers:
        y[i] = 0
        counter += 1
setx_df = pd.DataFrame(setx,columns=["xval"])
```

```
sety_df = pd.DataFrame(sety,columns=["yval"])
setr_df = pd.DataFrame(y,columns=['flag'])
setx_df['yval'] = sety_df['yval']
setx_df['flag'] = setr_df['flag']
set10 = setx_df[setx_df['flag']==1]
set00 = setx_df[setx_df['flag']==0]
plt.scatter(set10['xval'],set10['yval'], \
    color='red')
plt.scatter(set00['xval'],set00['yval'], \
    color='blue')
plt.show()
```

c) Create the ***X*** and ***y*** dataset for the neural network.

```
# Create a DataFrame
df = pd.DataFrame({'x': setx, 'y': sety, \
    'flag': y})
# Filter DataFrame to exclude rows where z is -1
X_df = df[df['flag'].isin([0, 1])]
Xd = X_df.drop(["flag"],axis=1)
X = np.array(Xd)
y1 = X_df["flag"]
y = np.array(y1)
```

d) The code below is for a specific neural network that performs CENN/LCNN using the detailed example provided in this chapter. The architecture of the NN was provided earlier in the figure. The forward propagation uses four steps. The backward propagation uses two steps and requires the partial derivatives to compute gradients for weights and biases, which are then stored in matrices and vectors for ease of use. The learning rate is used to obtain the new estimates.
Note: The code below is written in a form that is easy to follow rather than for efficiency or general-purpose use. Enter the code and run it on the circular dataset just created. It will initially run CENN as the default setting.

```
#Define logcosh function
logcosh = lambda x: np.logaddexp(x, -x) - \
    np.log(2)
#Define sigmoid function
def sigmoid(z):
    a = 1./(1.+ np.exp(-z))
    return a
#Initialize weights and biases
np.random.seed(9)
W_matrix = np.random.randn(4,2)*0.001
w_vector = np.random.randn(4)*0.001
b_vector = np.random.randn(4,1)*0.001
```

```
pick_random = np.random.randn(5)*0.001
b0 = pick_random[4]
n = X.shape[0]
#IMPORTANT:  CE=1 and LC=0, OR CE=0 and LC=1
CE = 1
LC = 0
if (CE):
    print("Cross-entropy")
else:
    alpha = 5.  # hyperparameter a in log-cosh
    print("Log-cosh")
learning_rate = 0.2
maxIter = 7500
epochs = np.zeros(maxIter)
#MAIN LOOP
for t in range(maxIter):
# FWD PROP
    z = np.matmul(W_matrix,X.T)+b_vector #Step 1
    a = z*(z>0) # ReLU function          #Step 2
    zout = np.matmul(w_vector,a)+b0      #Step 3
    p = sigmoid(zout)                    #Step 4
#BACK PROP
    delW_matrix = np.zeros([4,2])
    delw_vector = np.zeros([4])
    delb_vector = np.zeros([4,1])
    delb0 = 0.0
    wvec = np.array([w_vector])
    ones_matrix = np.ones((n, 1))
    for i in range(n):
        Indicator = np.array([z[:,i]>0,z[:,i]>0])
        if (CE):
            fullW_matrix = np.matmul(wvec.T, \
                [X[i,:]])*(p[i]-y[i])
            fullb_vector = np.matmul(wvec.T, \
                [ones_matrix[i,:]])*(p[i]-y[i])
            delw_vector += (y[i]-p[i])*(-a[:,i])
            delb0 += (y[i]-p[i])*(-1.)
        else:
            fullW_matrix = -np.matmul(wvec.T, \
                [X[i,:]])*np.tanh(alpha*(y[i]-p[i]))
            fullb_vector = -np.matmul(wvec.T, \
                [ones_matrix[i,:]])* \
            np.tanh(alpha*(y[i]-p[i]))
```

```
                delw_vector += p[i]*(1.-p[i])* \
                    np.tanh(alpha*(y[i]-p[i]))*(-a[:,i])
                delb0 += p[i]*(1.-p[i])* \
                    np.tanh(alpha*(y[i]-p[i]))*(-1.)
            delW_matrix += fullW_matrix*Indicator.T
            Indicator = np.array([z[:,i]>0])
            delb_vector += fullb_vector*Indicator.T
        w_vector -=learning_rate*delw_vector/n #Step 5
        b0 -=learning_rate*delb0/n             #Step 5
        W_matrix -=learning_rate*delW_matrix/n #Step 6
        b_vector -=learning_rate*delb_vector/n #Step 6
     # LOSS
        if (CE):
            loss = -np.sum(y*np.log(p)+(1.-y)* \
                np.log(1.-p))
        else:
            loss = np.sum(logcosh(alpha*(y-p)))/alpha
        epochs[t] = loss
        if t % 1000 == 0:
            print ("Cost after epoch %i of %i: \
                %f" %(t, maxIter, loss))
    print("Done!")
    plt.plot(epochs)
    plt.show()
```

9.6 (Project) Plot the nonlinear decision boundaries.

a) In order to obtain the nonlinear decision boundary, one option is to create thousands of points in the region of interest (either as a grid or randomly selected points) and then run the resulting `Xtest` matrix through the forward propagation steps to get the predicted values. The results can be color-coded to obtain the resulting boundary. First define the prediction code.

```
def predict(X,prob):
    pT = 0.5
    z = np.matmul(W_matrix,X.T)+b_vector
    a = z*(z>0)
    zout = np.matmul(w_vector,a)+b0
    if (prob):
        Y_prediction = sigmoid(zout)
    else:
        Y_prediction = sigmoid(zout) > pT
    return Y_prediction
```

b) Next, the predicted values, either 0 or 1, can be obtained using the following code.

```
m = 4000
setx = np.random.random(m)*4.0-2.0
sety = np.random.random(m)*4.0-2.0
setx_df = pd.DataFrame(setx,columns=["xval"])
sety_df = pd.DataFrame(sety,columns=["yval"])
setx_df['yval'] = sety_df['yval']
Xtest = np.array(setx_df)
predictions = predict(Xtest,False)
```

c) Finally, to observe the nonlinear decision boundary for CENN, use the following code fragment.

```
print("CENN Correct Predictions:", \
    np.sum(predict(X,False) == y))
setr_df = pd.DataFrame(predictions, \
    columns=['flag'])
setx_df['flag'] = setr_df['flag']
set1 = setx_df[setx_df['flag']==1]
set0 = setx_df[setx_df['flag']==0]
plt.scatter(set1['xval'],set1['yval'], \
    color='red')
plt.scatter(set0['xval'],set0['yval'], \
    color='blue')
if (CE):
    plt.title("Cross-entropy Loss")
else:
    plt.title("Log-cosh Loss")
plt.scatter(set10['xval'],set10['yval'],color='black')
plt.show()
```

9.7 (Project) In this project, you will replace CENN with LCNN. For this purpose, you need to go back to the main routine of Problem 9.5(d) and change to code to use the following:

```
CE = 0
LC = 1
```

With these changes, we see that only a few lines of code separate LCNN from CENN. Therefore, existing programs can easily be modified to include the log-cosh loss. Repeat the same steps in the previous problem for LCNN and compare the results with CENN.

```
m = 4000
setx = np.random.random(m)*4.0-2.0
sety = np.random.random(m)*4.0-2.0
setx_df = pd.DataFrame(setx,columns=["xval"])
sety_df = pd.DataFrame(sety,columns=["yval"])
```

```
setx_df['yval'] = sety_df['yval']
Xt = np.array(setx_df)
predictions = predict(Xt,False)
print("LCNN Correct Predictions:", \
    np.sum(predict(X,False) == y))
setr_df = pd.DataFrame(predictions, \
    columns=['flag'])
setx_df['flag'] = setr_df['flag']
set1 = setx_df[setx_df['flag']==1]
set0 = setx_df[setx_df['flag']==0]
plt.scatter(set1['xval'],set1['yval'], \
    color='red')
plt.scatter(set0['xval'],set0['yval'], \
    color='blue')
if (CE):
    plt.title("Cross-entropy Loss")
else:
    plt.title("Log-cosh Loss")
plt.scatter(set10['xval'],set10['yval'], \
    color='black')
plt.show()
```

9.8 (Project) Using the results from CENN and LCNN in the previous projects, compute the accuracy, recall, precision, F_1 score, and AUROC metrics for the two methods. Compare the results. You may need to run CENN again in order to obtain both sets of metrics.

a) Predict the outcomes.

```
pred = predict(X,False)  # Get logical outputs
```

b) Compute the metrics.

```
from sklearn import  metrics
if (CE):
    print("Cross-entropy metrics")
else:
    print("Log-cosh metrics")
print("Confusion Matrix:")
table = metrics.confusion_matrix(pred,y)
print(table)
accuracy = np.mean(pred == y)
precision = metrics.precision_score(y,pred)
recall = metrics.recall_score(y,pred)
f1 = metrics.f1_score(y,pred)
pred = predict(X,True)
```

```
    auroc = metrics.roc_auc_score(y, pred)
    print("accuracy =",format (accuracy, '.3f'))
    print("precision =",format (precision, '.3f'))
    print("recall =",format (recall, '.3f'))
    print("F1 score =",format (f1, '.3f'))
    print("AUROC score =",format (auroc, '.3f'))
```

9.9 (Advanced Project) The goal of this advanced project is to modify the code for the CENN/LCNN program in Problem 9.5(d) to use stochastic gradient descent. Every row will be processed sequentially, rather than all rows at once. In order to do this, move the lines labeled Steps 5 and 6 so that they reside inside the inner for loop (insert one tab space before each of the four lines). Then change the learning rate to 0.006 and `maxIter` to 1000 (roughly one-eighth of the prior number of epochs). Rerun CENN and LCNN to observe the results. Adjust `maxIter` and the learning rate as needed to obtain the desired results.

References

Haykin, S. (2009). *Neural Networks and Machine Learning*, 3e. Prentice-Hall.

James, G., Witten, D., Hastie, T. et al. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer.

10

Multi-class Classification and Adam Optimization

10.1 Introduction

This chapter addresses multi-class classification problems using neural networks (NNs) and the details of their optimization during training. First, the loss functions for multi-class problems are described which require categorical versions of the cross-entropy and log-cosh loss functions. Then, the sigmoid function is extended to incorporate multiple outputs leading to the softmax activation function. The implementation of the softmax activation function is described in detail. It involves a slightly more complicated calculation of the gradients than the sigmoid function used in Chapters 8 and 9. This is followed by an example of a multi-class problem solved using neural networks. Specifically, the MNIST dataset is used as the demonstration vehicle to identify the advantages of using a robust loss function for this type of problem.

To close the chapter, a number of different algorithms that comprise the Adam optimization technique will be described. Advanced algorithms are required in deep learning to navigate through the complicated optimization landscapes posed by neural networks, such as plateaus and ravines. These techniques include momentum, rmsprop and warm-up procedures. It should be noted that deep learning (see Haykin 2009; James et al. 2023) would not be possible without these advanced methods. While the primary aim of this chapter is to compare the cross-entropy neural network (CENN) with the log-cosh network (LCNN), it is important to understand that the foundational elements such as multi-class classification, categorical loss functions, the softmax activation function, and the Adam optimizer are crucial for the development of large language models (LLMs) used in today's generative AI tools. These components will be explained in detail, and it is highly recommended that readers study this material carefully.

10.2 Multi-class Classification

In the last two chapters, we focused on classification problems with binary outcomes—either 0 or 1. However, there are more complex classification scenarios where multiple outcomes are possible, but only one is the correct response. For instance, consider predicting which of three stores a customer should visit to purchase a certain type of product. The outcomes can only be Store A, Store B, or Store C. In this case,

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.

Companion Website: www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

i	x_1	x_2	x_3	x_4	Store A	Store B	Store C
1	0.306	0.898	0.239	0.902	1	0	0
2	0.294	0.863	0.176	0.871	0	1	0
3	0.361	0.835	0.125	0.863	0	0	1
4	0.310	0.902	0.235	0.741	0	0	1
5	0.298	0.871	0.173	0.314	1	0	0
6	0.365	0.863	0.122	0.302	1	0	0
7	0.314	0.741	0.235	0.173	1	0	0
8	0.302	0.733	0.173	0.122	0	1	0
9	0.369	0.745	0.122	0.235	0	1	0
10	0.318	0.757	0.239	0.173	0	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	0.235	0.173	0.310	0.902	0	0	1

Figure 10.1 A dataset with three one-hot encoded classes.

we refer to the set of outcomes as classes and the type of data as **categorical**. Typically, each class is assigned a numerical label: Store A = 1, Store B = 2, and Store C = 3. To represent these labels in a dataset for processing by a neural network (NN), we use **one-hot encoding** whereby each label is converted to a binary vector where only one element is set to 1, while the others are set to 0. Specifically, we can use the following one-hot encoding: Store A = 100, Store B = 010, and Store C = 001. More generally, for a response with c classes, a one-hot coding requires c bits with only one bit set to 1 to indicate the class. By doing so, the $\boldsymbol{y}$ vector used in binary classification now becomes a $\boldsymbol{Y}$ matrix for multi-class classification.

Consider the hypothetical dataset for the store selection problem depicted in Figure 10.1 that captures customer purchasing behavior for each product. The four key variables are product price, product rating, amount of discount available, and store distance from the customer. They are listed as column headings x_1, x_2, x_3, and x_4 in the table. In row 1 of the table, only one of the three possible responses is set to 1, reflecting the one-hot encoding of the label associated with Store A. Row 2 refers to Store B, rows 3 and 4 to Store C, and so on.

With the dataset in hand, the next step is to build a neural network with four inputs and three outputs. The NN architecture will involve three hidden layers and four hidden units in each layer. The output activation will be performed using the softmax function. Next, a loss function is needed that is specifically designed for categorical outputs. Finally, an optimization technique is required to minimize the loss function to produce the model that can be used for inference. Each of these steps will be outlined in the rest of this chapter.

10.2.1 Multi-class Loss Functions

In order to carry out machine learning with the dataset of Figure 10.1, we need to modify the two loss functions discussed in earlier chapters because we now have a large matrix of responses that are either 0 or 1. In particular, we want to use loss functions that focus on the 1's in the responses which have been conveniently provided by the one-hot encoding. A categorical version of the cross-entropy (CE) loss for use with multi-class problems will be described first. The categorical CE loss function is

$$\text{categorical CE loss} = -\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik} \log p_{ik}, \tag{10.2.1}$$

where c is the number of classes and n is the number of rows in the dataset. Similar to binary classification terminology, y_{ik} is the ground truth and p_{ik} is the predicted probability. Keep in mind that all but one of the y_{ik}'s are 0 due to one-hot encoding so only one term will contribute to the loss in each row of the dataset.

Recall from Chapter 9 that, in stochastic gradient descent, only one row of the dataset is processed at a time. Let us focus on this case to simplify the understanding of the modified loss function. The categorical CE loss for a single row is

$$\text{categorical CE loss} = -\sum_{k=1}^{c} y_k \log p_k. \tag{10.2.2}$$

For the first row of the dataset in Figure 10.1, we would obtain

$$-\sum_{k=1}^{c} y_k \log p_k = -(y_1 \log p_1 + y_2 \log p_2 + y_3 \log p_3). \tag{10.2.3}$$

Since Store A was selected, we have that $y_1 = 1, y_2 = 0$, and $y_3 = 0$. Therefore,

$$-\sum_{k=1}^{c} y_k \log p_k = -\log p_1. \tag{10.2.4}$$

Using the categorical loss and one-hot encoding, we obtain a very simple loss term. Each row of the training set will contribute one such term. The total loss is then the sum of the loss terms for all rows, as prescribed by Eq. (10.2.1).

Exercise: Determine the expression for the categorical CE loss for the first four rows of the dataset in Figure 10.1 using Eq. (10.2.1).

Ans.: For the first row, the loss is $-\log p_{11}$. For the second row, the loss is $-\log p_{22}$. For the third row, the loss is $-\log p_{33}$. For the fourth row, the loss is $-\log p_{43}$. □

The categorical loss function for log-cosh (LC) is given by

$$\text{categorical LC loss} = \sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik} \operatorname{logcosh}(y_{ik} - p_{ik}). \tag{10.2.5}$$

Exercise: Determine the categorical LC loss terms separately for rows 1, 2, 3, and 4 of the dataset in Figure 10.1 using Eq. (10.2.5).

Ans.: For the first row, the loss is $\operatorname{logcosh}(y_{11} - p_{11})$. For the second row, the loss is $\operatorname{logcosh}(y_{22} - p_{22})$. For the third row, the loss is $\operatorname{logcosh}(y_{33} - p_{33})$. For the fourth row, the loss is $\operatorname{logcosh}(y_{43} - p_{43})$. □

Note: The softmax function is perhaps the most important activation function in deep learning. It is important to understand this topic in detail because you will encounter it frequently in many applications including generative AI tools such as ChatGPT.

10.2.2 Softmax Activation Function

One of the key aspects of multi-class problems involves the use of the **softmax** function. This is essentially a multi-dimensional version of the sigmoid function. Recall that the sigmoid function generates a probability value for a single output in binary classification. In the multi-class problem, there are k outputs which is equal to the number of classes. It is possible to use the sigmoid activation at each of the k outputs but one inconsistency would arise. Since the outputs of the neural network are interpretable as probabilities, the total sum of these probabilities at the outputs should be equal to 1. The sigmoid activation cannot guarantee this probability axiom whereas the softmax function is designed such that it is always true. That is why, with multiple classes and multiple outputs, the softmax activation function is required.

Assume we have k outputs that are one-hot encoded for training purposes. After forward propagation, we will have k raw outputs, $z_1, z_2, \dots, z_k$. In the softmax operation, given by $\mathbb{S}(z_i)$, the raw outputs of the neural network are used in the following way to compute the probability for the i^{th} output:

$$\mathbb{S}(z_i) = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_k}} = p_i. \tag{10.2.6}$$

Effectively, the probabilities of all k outputs are scaled such that they will always add up to 1. The scaling could have been done using the raw outputs, $z_1, z_2, \dots, z_k$, but there is a good reason why the form of normalization in Eq. (10.2.6) is used—it is shift invariant. That is, a constant can be added to all raw outputs and their probabilities will not change.

The gradients of softmax have simple expressions that include the effects of all k outputs rather than just a single output. We will show (through an example) that

$$\frac{\partial\, \mathbb{S}(z_i)}{\partial z_i} = \frac{\partial p_i}{\partial z_i} = p_i(1 - p_i) \quad \text{(regular term)}$$

and

$$\frac{\partial\, \mathbb{S}(z_i)}{\partial z_j} = \frac{\partial p_i}{\partial z_j} = -p_j p_i \quad \text{(cross terms),}$$

for all $j = 1, \dots, k, \quad j \neq i$. Instead of one term for the derivative, there will now be k terms: one regular term and $k - 1$ cross-terms.

To understand how the softmax activation function works, consider Figure 10.2. It is the four-layer neural network of Figure 9.3 with the fourth layer replaced by the softmax function since there are three outputs in the store selection problem. A slight modification has been made to show the dependencies between the fourth layer outputs, i.e. the raw z_i values, and the softmax function. For example, when z_1 is generated as a sum-of-products quantity, it contributes to all three softmax outputs, p_1, p_2, and p_3. In the figure, the three lines emanating from z_1 are used to indicate this dependence. Likewise, z_2 affects p_1, p_2, and p_3 so three more lines are used for the same purpose. The same holds for z_3. These arrows do not represent weights as they do in the rest of the network; they only indicate the dependencies between the raw outputs and the softmax probabilities.

Example: To solidify the concepts presented above, we will go through a softmax example in a step-by-step manner using row 1 of the dataset in Figure 10.1 and the neural network diagram of Figure 10.2. We focus on the gradients for weights w_1, w_2, and w_3 for illustration purposes. The first step is to determine the softmax probabilities of the three outputs along with their partial derivatives with respect to the raw output of interest z_1:

$$p_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} \quad \Rightarrow \quad \frac{\partial p_1}{\partial z_1} = \frac{e^{z_1}(e^{z_2} + e^{z_3})}{(e^{z_1} + e^{z_2} + e^{z_3})^2} = p_1(1 - p_1),$$

$$p_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}} \quad \Rightarrow \quad \frac{\partial p_2}{\partial z_1} = -\frac{e^{z_1+z_2}}{(e^{z_1} + e^{z_2} + e^{z_3})^2} = -p_1 p_2,$$

$$p_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}} \quad \Rightarrow \quad \frac{\partial p_3}{\partial z_1} = -\frac{e^{z_1+z_3}}{(e^{z_1} + e^{z_2} + e^{z_3})^2} = -p_1 p_3.$$

We also know that $p_1 + p_2 + p_3 = 1$ which we will make use of later. Using the categorical CE loss for row 1, represented as $\mathbb{L}_{r1}^{\text{CE}}$, we find that

$$\mathbb{L}_{r1}^{\text{CE}} = -y_1 \log p_1 - y_2 \log p_2 - y_3 \log p_3.$$

Next, we need to select a weight and find its associated gradient. Initially, we select w_1 which is multiplied by a_1 to produce one component of z_1 as indicated by the arrow in Figure 10.2. Hence, $\partial z_1/\partial w_1 = a_1$. The full gradient will be composed of three terms, one for each output (see three arrows leaving z_1) as follows:

$$\begin{aligned}
\frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial w_1} &= \frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial p_1}\frac{\partial p_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} + \frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial p_2}\frac{\partial p_2}{\partial z_1}\frac{\partial z_1}{\partial w_1} + \frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial p_3}\frac{\partial p_3}{\partial z_1}\frac{\partial z_1}{\partial w_1} \\
&= -\frac{y_1}{p_1} p_1(1 - p_1)a_1 - \frac{y_2}{p_2}(-p_1 p_2)a_1 - \frac{y_3}{p_3}(-p_1 p_3)a_1 \\
&= -y_1(1 - p_1)a_1 - y_2(-p_1)a_1 - y_3(-p_1)a_1 \\
&= -y_1 a_1 - (y_1 + y_2 + y_3)(-p_1)a_1 = -(y_1 - p_1)a_1.
\end{aligned}$$

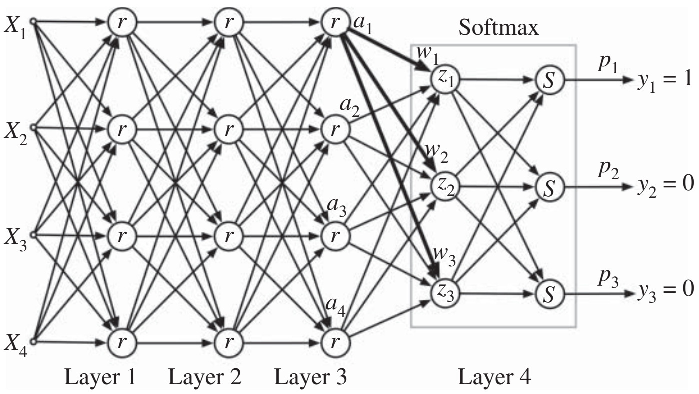


Figure 10.2 A four-layer neural network with a softmax activation.

It is somewhat remarkable that after all the calculus and algebra used, the final form is a simple quantity that we have seen several times throughout this book. This is due to the form of the softmax function and the fact that $y_1 + y_2 + y_3 = 1$ which conveniently reduces the equation into a compact form. As a final step, we can use the values associated with the one-hot encoding in the first row of the dataset. Since $y_1 = 1$, $y_2 = 0$ and $y_3 = 0$, the final result is

$$\frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial w_1} = -(1 - p_1)a_1.$$

Interestingly, this is the same result that would be obtained by applying one-hot encoding of 100 to an earlier step in the derivation as follows

$$\frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial w_1} = -1(1 - p_1)a_1 - 0(-p_1p_2)a_1 - 0(-p_1p_3)a_1 = -(1 - p_1)a_1.$$

The next gradient of interest is associated with w_2. It proceeds in the same manner as the above but with different partial derivatives involving the cross-terms. We begin with the equation for the loss as before,

$$\mathbb{L}_{r1}^{\text{CE}} = -y_1 \log p_1 - y_2 \log p_2 - y_3 \log p_3.$$

Again, the gradient will be composed of three terms, since there are three arrows leaving z_2, but they will be different for the case of w_2 compared to w_1 as follows:

$$\begin{aligned}\frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial w_2} &= \frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial p_1}\frac{\partial p_1}{\partial z_2}\frac{\partial z_2}{\partial w_2} + \frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial p_2}\frac{\partial p_2}{\partial z_2}\frac{\partial z_2}{\partial w_2} + \frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial p_3}\frac{\partial p_3}{\partial z_2}\frac{\partial z_2}{\partial w_2} \\ &= -\frac{y_1}{p_1}(-p_1p_2)a_1 - \frac{y_2}{p_2}p_2(1 - p_2)a_1 - \frac{y_3}{p_3}(-p_3p_2)a_1 \\ &= y_1p_2a_1 - y_2(1 - p_2)a_1 + y_3(p_2)a_1 \\ &= -y_2a_1 + (y_1 + y_2 + y_3)(p_2)a_1 = -(y_2 - p_2)a_1 = p_2a_1.\end{aligned}$$

Finally, for w_3, we follow the same steps and obtain

$$\frac{\partial \mathbb{L}_{r1}^{\text{CE}}}{\partial w_3} = -(y_3 - p_3)a_1 = p_3a_1.$$

□

It is abundantly clear why the categorical cross-entropy loss works well with softmax: the terms cancel in such a way as to produce simple and familiar gradients. Therefore, the implementation of softmax with categorical CE is common practice in deep learning applications.

Example: This example carries out a similar process to find the gradients for the categorical LC loss function when used in conjunction with softmax activation. We select w_1, w_2, and w_3, and repeat the same steps as above. Using the categorical LC loss for row 1, represented as $\mathbb{L}_{r1}^{\text{LC}}$ below, we find that

$$\mathbb{L}_{r1}^{\text{LC}} = y_1\text{logcosh}(y_1 - p_1) + y_2\text{logcosh}(y_2 - p_2) + y_3\text{logcosh}(y_3 - p_3).$$

Since $y_1 = 1, y_2 = 0$, and $y_3 = 0$, we can write the gradient as simply

$$\begin{aligned}\frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial w_1} &= \frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial p_1}\frac{\partial p_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} \\ &= -\tanh(1 - p_1)p_1(1 - p_1)a_1.\end{aligned}$$

Likewise, for the case of w_2, we have

$$\begin{aligned}\frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial w_2} &= \frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial p_1}\frac{\partial p_1}{\partial z_2}\frac{\partial z_2}{\partial w_2}\\ &= -\tanh(1-p_1)(-p_1p_2)a_1,\end{aligned}$$

and for w_3, we have

$$\begin{aligned}\frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial w_3} &= \frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial p_1}\frac{\partial p_1}{\partial z_3}\frac{\partial z_3}{\partial w_3}\\ &= -\tanh(1-p_1)(-p_1p_3)a_1.\end{aligned}$$

An efficient implementation of softmax with LC requires that the one-hot encoded values be used to compute only one term for the loss and one term for the gradient for each of the weights. All of the other terms are identically 0 and need not be computed. □

Exercise: For the general log-cosh loss function with hyperparameter a (not to be confused with the activation output), let the categorical version be given by

$$\text{categorical LC loss} = \sum_{k=1}^{c}\frac{y_k}{a}\text{logcosh}\,(a(y_k - p_k)).$$

Find the gradients for w_1, w_2, and w_3 for the softmax function in the diagram of Figure 10.2 using the first row of the dataset in Figure 10.1.

Ans.: Following the steps above, the gradient component for w_1 is

$$\frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial w_1} = -\tanh(a(y_1-p_1))p_1(1-p_1)a_1.$$

Likewise, for the case of w_2, we will have

$$\frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial w_2} = -\tanh(a(y_1-p_1))(-p_1p_2)a_1$$

and for w_3, we will have

$$\frac{\partial \mathbb{L}_{r1}^{\text{LC}}}{\partial w_3} = -\tanh(a(y_1-p_1))(-p_1p_3)a_1.$$

□

All the pieces are now in place to build a model for the store selection problem. Key elements of the dataset are given in Figure 10.1, and the architecture of the neural network in Figure 10.2. The gradients associated with softmax activation were provided in the examples and exercises for 3 of the 12 weights in the final layer. The other 9 weights are derived in the same manner along with the associated 3 biases. Backprop can be used to compute the gradients for all weights and biases, as described in Chapter 9. Finally, the algorithms given in that chapter for CENN and LCNN can be used to train the NN using gradient descent to produce the final model, which concludes our study of the store selection problem.

10.3 Example: MNIST Dataset

In this section, we take on a more challenging problem in the domain of multi-class classification with neural networks to compare LCNN against CENN (refer to Chapter 9) using the MNIST dataset.[1] Note that we consider fully connected networks rather than the advanced convolutional neural network (CNN). A CNN is more appropriate for image recognition but our goal is to compare CENN and LCNN so a fully connected architecture is sufficient for our purposes. As such, we will use the same NN architecture for both methods and the same learning rate. The objective is to tackle a large multi-class problem to determine if LCNN is suitable for this purpose.

The MNIST dataset consists of 42,000 handwritten numbers from 0 to 9 (total of 10 digits) in the form of 28×28 grayscale images with the objective of correctly recognizing as many numbers as possible using neural networks. For a fully connected architecture, each 28×28 image must first be unrolled to 784 pixels, each of which represents an input variable. Associated with each image is a corresponding label of 0-9, i.e. the digit being represented by the image. The dataset has no known outliers but the robust approach has a habit of finding outliers even when none are expected.

A sampling of 10 such digits is provided in Figure 10.3. They are hand-drawn digits such as would be used in a US zip code or street address on letters to be mailed. Recognizing digits is challenging since every person writes each digit in a slightly different way. The NN is given a subset of these images for training purposes, while the rest are used for testing purposes.

The MNIST dataset falls into the category of multi-class classification. Rather than one output of the network, we need 10 outputs—one for each digit to be recognized. The standard training approach is to apply a given image at the input and then set the corresponding output to 1 while setting all other outputs to 0. For example, an image of the number 3 would be encoded at the output as 0001000000, the

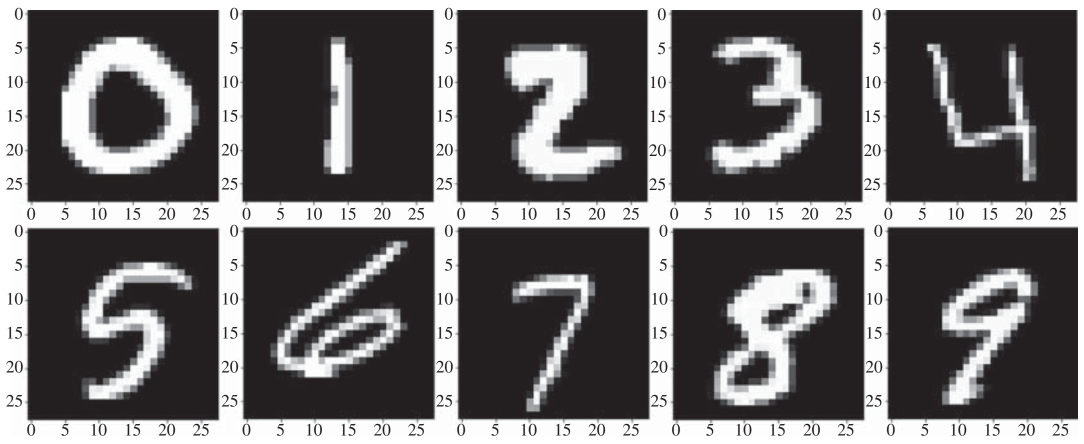


Figure 10.3 Ten representative images from the MNIST dataset.

1 This dataset is available on the website kaggle.com in the Competitions section.

number 7 would be encoded as 0000000100, and so on. Hence, the response is now a matrix $\boldsymbol{Y}$ whereby each element $y_{ij} \in [0, 1]$. The neural network is then trained on a dataset of images, each with 784 inputs and 10 outputs with one-hot encoded labels.

10.3.1 Neural Network Architecture

The NN architecture to be used is as follows: 784 inputs, 25 units in the hidden layer, and 10 units in the output layer as shown in Figure 10.4. Each image is unrolled and normalized into 784 pixel values to be used as input. The output activation is based on softmax. The 25 hidden units all use relu activation. Roughly speaking, this amounts to 20,000 parameters. Each image is labeled from 0 to 9 and then one-hot encoded to form a binary output for the NN. These are all standard steps in mapping an image and label for training purposes onto a fully connected neural network.

Exercise: How many total parameters are there in the neural network of Figure 10.4. Include all weights and biases.

Ans.: For the first layer, there are $784 \times 25 + 25 = 19,625$ parameters. For the second layer, there are $25 \times 10 + 10 = 260$. Thus, the total number of parameters 19,885. □

10.3.2 Comparing Cross-Entropy with Log-Cosh Losses

With this setup, the overall objective is to determine whether there are outliers in the data and if robust methods can outperform the standard approach. A dataset of 42,000 images of handwritten characters is used to compare LCNN and CENN. To make the problem tractable for demonstration purposes, the training set is reduced to 2000 labeled images. Before examining the results, it is important to ensure that both methods converge to a solution using batch gradient descent with momentum. The convergence plots are shown for CENN and LCNN ($a = 10$), respectively, in Figure 10.5. Notice that both methods have similar convergence patterns. LCNN generally requires a larger number of iterations to converge. Specifically, CENN required 50,000 iterations while LCNN required 90,000 iterations.

Consider the first set of results in Table 10.1 associated with training. We find that CENN is able to fit to 100% of the training data while LCNN is able to fit only 97.5% of the data. At first glance, one may

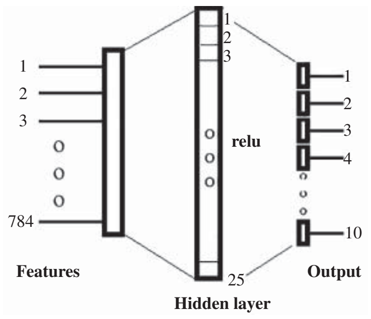


Figure 10.4 Neural network architecture for training on the MNIST dataset.

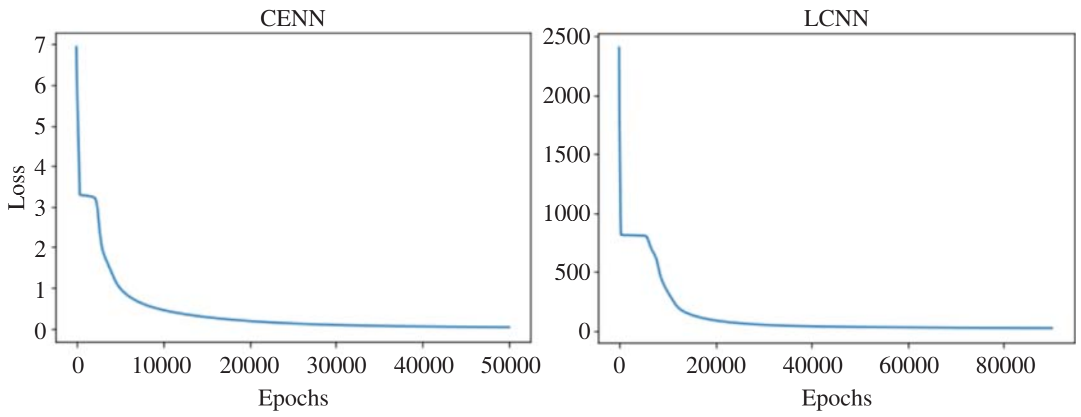


Figure 10.5 CENN and LCNN convergence traces—loss vs. iterations.

Table 10.1 MNIST training set size is $n = 2000$. 50 outliers detected by LCNN. 0 outliers detected by CENN. The resulting models were used to predict on a test set with $n = 40000$.

Train	**Accuracy**	**Matched**	**Outliers**
CENN	100%	2000/2000	0
LCNN	97.5%	1950/2000	50
Test	**Accuracy**	**Matched**	**Outliers**
CENN	89.8%	35,930/40,000	—
LCNN	90.0%	36,009/40,000	—

believe that CENN is doing a better job and, in some respects, that is true. However, there are so many parameters (i.e. approx. 20000) and only 2000 images that overfitting is likely to occur in both CENN and LCNN. In any case, both methods achieve convergence, as shown in Figure 10.5, so there must be some other reason for this discrepancy. The reason has to do with outliers, as described in the next section.

Results of the two models from CENN and LCNN when used to predict the label on 40,000 images in the test set are provided in the same table. During the prediction phase, we want to select the output with the largest probability to determine the class association. This is normally done with the **argmax** function which translates the largest probability to a label for a given image. We find that CENN is able to predict 89.8% of test images correctly (i.e. 35,930 out of 40,000) while LCNN is able to predict 90.0% correctly (i.e. 36,009 out of 40,000). This is due to the fact that LCNN produced a slightly better model than CENN. Even though the training set accuracy was lower for LCNN, it was able to achieve a higher prediction accuracy on the test set. This demonstrates that the log-cosh loss may be more effective than cross-entropy in certain applications where outliers are present.

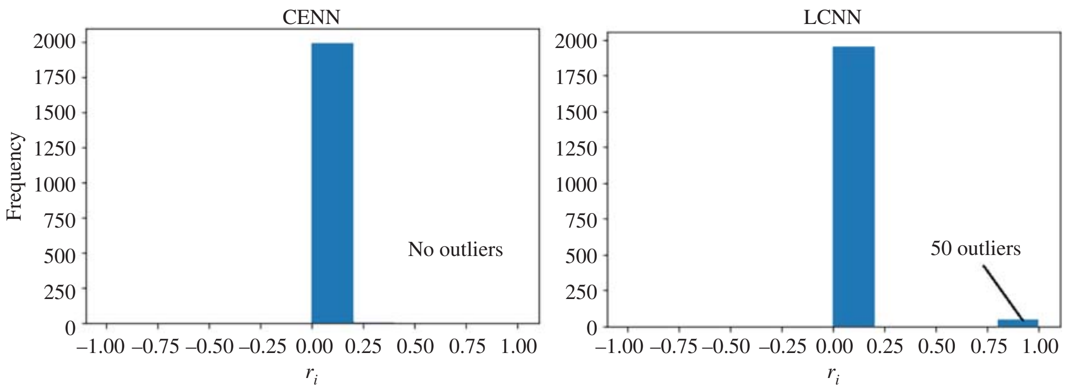


Figure 10.6 Histograms of residuals for CENN (0 outliers) and LCNN (50 outliers).

10.3.3 Outliers in MNIST

The reason why LCNN performed better than CENN can be linked to outliers detected in the MNIST dataset by the robust method. A histogram of LCNN residuals in Figure 10.6 indicates that 50 outliers exist in the data while the histogram for CENN does not show any evidence of outliers. In effect, CENN was able to fit 100% because there are enough parameters to overfit the data, whereas in the case of LCNN, only 97.5% of the digits were recognized and fitted. The remaining 50 images were designated as potential outliers by LCNN. This serves as a form of auto-regularization because the model built by LCNN effectively ignored 50 images. These outlier images are shown in Figure 10.7. Note that many of the circled images are not recognizable as numbers and therefore can be considered as outliers. This is a potentially valuable capability arising from the use of robust methods on the MNIST dataset.

10.4 Optimization of Neural Networks

To conclude this chapter, we shift our focus to the critical aspect of optimization of neural networks. Optimization lies at the heart of training deep learning models. It determines how quickly and how effectively the neural network adapts its weights and biases to minimize the loss function. While gradient descent serves as the foundation, recent advancements have introduced more sophisticated optimization algorithms. Among these, the Adam optimizer stands out as a powerful and widely used technique. In this section, the goal is to bridge the gap between neural networks and optimization by describing the Adam optimizer and its role in deep learning. We cover gradient descent with momentum and rmsprop to provide a complete picture for the Adam optimizer. This optimization algorithm can also be used for linear and logistic regression, in addition to neural networks for which it was originally intended.

To review the basic operation of gradient descent, recall that the parameters are updated at each iteration using some fraction of the gradient based on the learning rate, ℓ, shown symbolically as follows, with t as the iteration count

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \ell \times \left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^{(t)}. \tag{10.4.1}$$

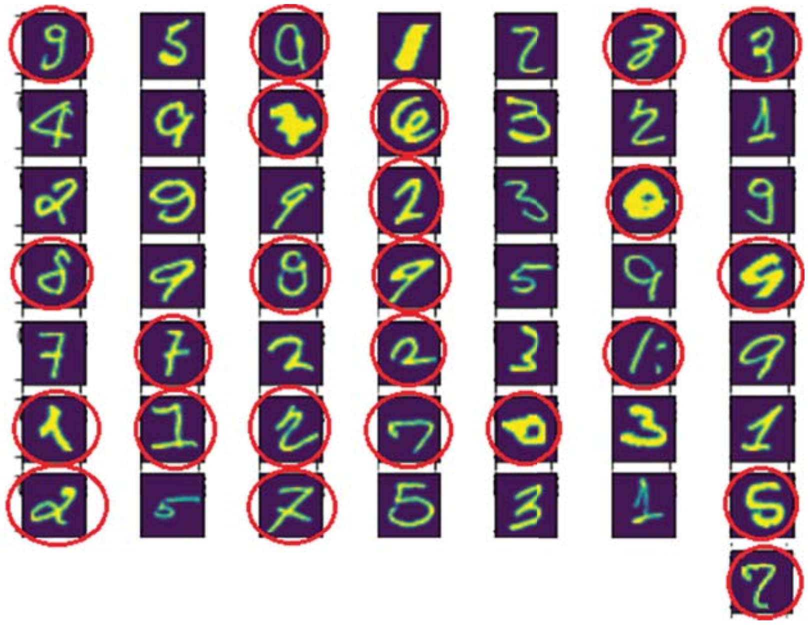


Figure 10.7 These are 50 potential outlier images reported by LCNN. Of these, the circled digits are not easily recognized by humans as a number. The data scientist can decide if they should be kept or removed.

Here, the objective function, $J(\boldsymbol{w})$, is being minimized and we seek the weight vector $\boldsymbol{w}$ associated with the minimum. This iterative method requires partial derivatives of $J(\boldsymbol{w})$ with respect to all the weights represented by $\boldsymbol{w}$. For ease of understanding, we focus on optimizing the weights but a similar formulation is used for the bias. The descriptions are carried out using vectors and scalars for simplicity but will often involve matrices and vectors in practice.

The optimization of neural networks can be quite difficult because the parameter space is extremely large. The actual configuration space of the loss function in such high dimensions is hard to navigate and harder to visualize. To give a sense of the complexity, we offer two representations of different landscapes associated with two different loss functions in Figure 10.8. In the figure, the y-axis is the loss function value and the two x-axes are the weights of interest, in this case w_1 and w_2. On the left, the landscape exhibits a steeper terrain, characterized by plateaus, ridges, ravines and a myriad of twists and turns. The gradients are large which tends to help the optimizer move more rapidly toward the minimum. A hypothetical desirable path is shown beginning on a plateau, where convergence would be slow, and then winding its way across the rugged terrain to eventually reach a minimum. This is representative of the optimization space for the CE loss function.

On the right side of the figure, the landscape is much flatter with shallower valleys and ridges. The gradients are smaller than in the CE case, which implies slower progress to the minimum (assuming the same learning rate for both). The reasons for this were discussed in Chapter 8 using Eq. (8.4.9). A hypothetical desirable path is shown in this case but does not reflect the speed at which the minimum is reached. This type of landscape is representative of the LC loss function. An optimizer must be able to handle both types of loss functions, with momentum, rmsprop and adaptive learning rates, in order

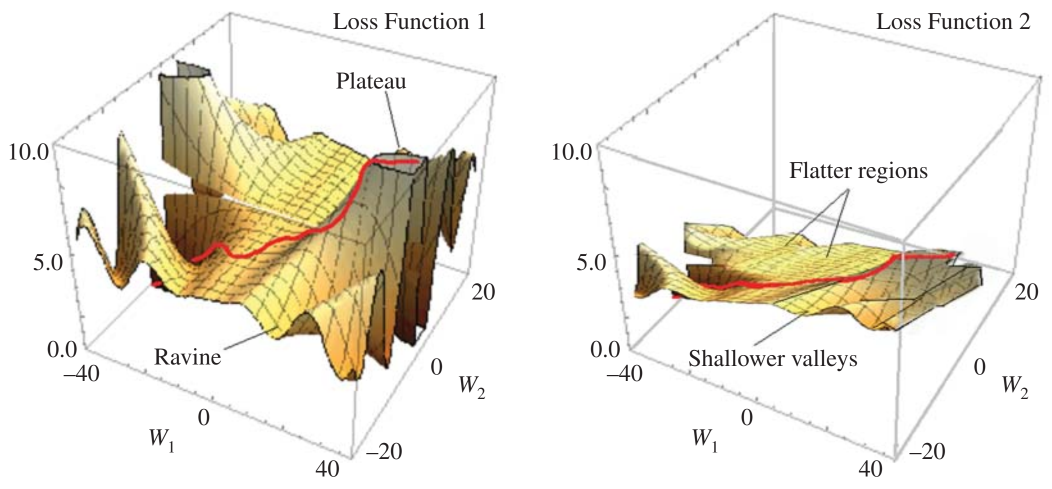


Figure 10.8 Conceptual view of optimization landscapes for two different loss functions.

to converge in a timely manner. And it must do this reliably in a very high-dimensional space. This is the reason that advanced methods are required for deep learning.

10.4.1 Momentum

There are a number of different ways to speed up the training process. One simple way is to adjust the learning rate, ℓ. The number of iterations needed to reach an acceptable level of accuracy is controlled by the learning rate. If ℓ is too large, convergence may not occur due to oscillations from one iteration to the next. If ℓ is too small, the rate of convergence can be very slow which implies a large number of iterations. Therefore, adaptive learning rate adjustment is a requirement in neural networks. A general rule is to reduce the learning rate as the minimum is approached.

The more important issue to be addressed is oscillations in the gradients in one or more dimensions in the optimization space, which slows down convergence. If we damp out the oscillations by averaging the gradients over several past iterations, it would allow for a more rapid convergence. A method called gradient descent with momentum does exactly that. It makes use of the history of gradients for each parameter using a weighted average of some number of past gradients. It is a smoothing mechanism that prevents wild fluctuations in the gradients which allows larger learning rates to be used.

Momentum works by taking a combination of the past gradients and the latest gradient to produce **an exponentially weighted moving average** using the following equation:

$$\boldsymbol{m}_{dw}^{(t+1)} = \alpha_1 \boldsymbol{m}_{dw}^{(t)} + (1-\alpha_1)\left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^{(t)}. \tag{10.4.2}$$

Basically, the latest gradient is scaled by $(1-\alpha_1)$ and added to the previous history of gradients $\boldsymbol{m}_{dw}^{(t)}$ which is scaled by α_1, where $\alpha_1 \in [0,1]$. The rolling exponentially weighted average is used to update the parameters. The smoothing parameter, α_1, controls the number of gradients used from past iterations in the averaging process and is typically set to 0.9. For example, with $\alpha_1 = 0.9$, the last 9 gradients are averaged and added to the latest gradient. The resulting smoothed out composite gradient is then used to establish the direction of the next step. The new update equation for $\boldsymbol{w}$ after applying momentum is

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \ell \times \boldsymbol{m}_{dw}^{(t+1)}. \tag{10.4.3}$$

Initially, $\boldsymbol{m}_{dw}^{(0)} = \boldsymbol{0}$ in Eq. (10.4.2). Then, it increases as the gradients are computed and a history is developed. The use of momentum in conjunction with gradient descent has been found to be highly effective in neural networks and is the foundation of the Adam optimizer.

Exercise: Given that $(\partial J(w)/\partial w)^{(0)} = 5$, $m_{dw}^{(0)} = 0$, and $w^{(0)} = 0.1$, compute the value of $w^{(1)}$ using the momentum equation with $\alpha_1 = 0.9$ and $\ell = 0.05$.

Ans.: The momentum term is given by

$$m_{dw}^{(1)} = \alpha_1 m_{dw}^{(0)} + (1 - \alpha_1)\, dw^{(0)} = 0.9 \times 0 + 0.1 \times 5 = 0.5.$$

Therefore, $w^{(1)} = 0.1 - 0.05 \times 0.5 = 0.075$. (Without momentum, it would be $w^{(1)} = 0.1 - 0.05 \times 5 = -0.15$.) □

10.4.2 rmsprop Approach

While the momentum adjustment to gradient descent is quite effective, it does not account for the fact that different parameters will produce gradients that range in size from the very small to the very large. Ideally, we would like to have **a different learning rate for each parameter** but as the parameter space increases, this can become an unwieldy and expensive task, especially if the learning rate is varied from one iteration to another. Instead, this problem can be handled by simply dividing the learning rate by the absolute value of the derivative as follows:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \ell \left(\frac{\partial J(\boldsymbol{w})/\partial \boldsymbol{w}}{|\partial J(\boldsymbol{w})/\partial \boldsymbol{w}|} \right)^{(t)}. \tag{10.4.4}$$

Therefore, the final term is essentially the sign of each of the gradients rather than the actual gradient. This is similar in principle to the aLASSO penalty function of Chapter 5.

This approach, while simple in nature, has a number of problems, one of which has to do with the absence of the magnitudes of the gradients and the second having to do with absence of the exponentially weighted aspect of the gradients seen in the momentum approach. The rmsprop approach resolves both issues. Rather than using the absolute value term in the denominator, it is more effective to replace it by a root-mean-square value, where the mean is actually the exponentially weighted value. That is, take the square of the gradients of past iterations and then average them and take the square root to obtain the correct scale. In particular, let us define a new term as follows:

$$\boldsymbol{r}_{dw}^{(t+1)} = \alpha_2 \boldsymbol{r}_{dw}^{(t)} + (1 - \alpha_2) \left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} \right)^{(t)} \times \left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} \right)^{(t)}, \tag{10.4.5}$$

where $\alpha_2 \in [0, 1]$ and is typically set to 0.999. Then, the updated equation becomes

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \ell \frac{(\partial J(\boldsymbol{w})/\partial \boldsymbol{w})^{(t)}}{\sqrt{(\boldsymbol{r}_{dw})^{(t+1)}}}. \tag{10.4.6}$$

In this form, the derivative is being scaled by $\sqrt{\boldsymbol{r}_{dw}}$ rather than the absolute value of the gradient. Since the new formulation is intended to modify the learning rate, we could conveniently rewrite it for better understanding as follows:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \frac{\ell}{\sqrt{(\boldsymbol{r}_{dw})^{(t+1)}}} \left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} \right)^{(t)}. \tag{10.4.7}$$

Now the square root of the exponentially weighted squared gradients is in the denominator of the learning rate and implies that each parameter adjusts the learning rate based on its own exponential history of gradients. This has been found to be a very effective update equation. However, if the derivative is 0, the equation goes to infinity. To resolve this issue, we can add a small value, such as $\varepsilon = 10^{-8}$, to the denominator as follows:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \frac{\ell}{(\sqrt{(\boldsymbol{r}_{dw})^{(t+1)}} + \varepsilon)} \left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} \right)^{(t)}. \tag{10.4.8}$$

Exercise: Given that $dw^{(0)} = (\partial J(w)/\partial w)^{(0)} = 5$, $r_{dw}^{(0)} = 0$, and $w^{(0)} = 0.1$ compute the value of $w^{(1)}$ using the rmsprop equation with $\alpha_2 = 0.999$, $\varepsilon = 0$, and $\ell = 0.05$.

Ans.: The rmsprop term is given by

$$r_{dw}^{(1)} = \alpha_2 r_{dw}^{(0)} + (1 - \alpha_2)\, dw^{(0)} \times dw^{(0)} = 0.999 \times 0 + 0.001 \times 5^2 = 0.025$$

Therefore, $w^{(1)} = 0.1 - \frac{0.05}{\sqrt{0.025}} \times 5 = -1.48$. □

10.4.3 Optimizer Warm-Up Phase

When applying momentum or rmsprop, there is a startup problem due to the fact that there is no history to draw from for the exponentially weighted gradients at the beginning. In fact, it is not until roughly 10 iterations have been computed that the requisite amount of history is available (assuming $\alpha_1 = 0.9$). Therefore, some mechanism is needed to "warm-up" the two methods until enough iterations of history exist. One way to do this is to divide the corresponding terms for each method by a quantity that starts with a large fraction of the initial gradients and then reaches the steady-state value when enough history exists; i.e. the corrected values are

$$\boldsymbol{m}_{dw}^{\text{corr}} = \frac{\boldsymbol{m}_{dw}^{(t+1)}}{1 - \alpha_1^{t+1}}$$

and

$$\boldsymbol{r}_{dw}^{\text{corr}} = \frac{\boldsymbol{r}_{dw}^{(t+1)}}{1 - \alpha_2^{t+1}},$$

respectively. Assuming $t = 0$ at the start, the denominator for momentum is initially $(1 - \alpha_1)$. Then for $t = 1$, we use $(1 - \alpha_1^2)$, for $t = 2$, we use $(1 - \alpha_1^3)$, and so on. The same holds for rmsprop with α_2. Eventually, the denominator approaches 1 in both cases (given that α_1 and α_2 are always less than 1), thereby ending the warm-up phase.

Exercise: Compute the correction terms for momentum and rmsprop for the previous two problems.

Ans.: The corrected terms are

$$m_{dw}^{\text{corr}} = \frac{0.5}{1 - 0.9} = 5$$

and

$$r_{dw}^{\text{corr}} = \frac{0.025}{1 - 0.999} = 25.$$

□

Exercise: Given that $dw^{(0)} = 5$, compute $w^{(1)}$ using the corrected values of m_{dw} and r_{dw} of the previous two exercises.

Ans.: Using the corrected terms above, we obtain for momentum,

$$w^{(1)} = 0.1 - 0.05 \times 5 = -0.15$$

and for rmsprop,

$$w^{(1)} = 0.1 - 0.05 \times 5/\sqrt{25} = 0.05.$$

They are different even in the warm-up phase. □

10.4.4 Adam Optimizer

Each of the three methods above was presented as separate approaches. However, the Adam (ADAptive Minimizer) optimizer combines all of them into one tool. It applies rmsprop to the momentum update equation along with the warm-up correction to produce a very powerful method that has proven to be extremely efficient at finding a regional minimum in deep neural networks. The combination of the methods produces the following update equation:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \ell \times \frac{\boldsymbol{m}_{dw}^{\text{corr}}}{\sqrt{\boldsymbol{r}_{dw}^{\text{corr}}} + \varepsilon}. \tag{10.4.9}$$

Exercise: Given that $(\partial J(w)/\partial w)^{(0)} = 5$, $r_{dw}^{(0)} = 0$, and $w^{(0)} = 0.1$, compute the value of $w^{(1)}$ using Eq. (10.4.9) with $\alpha_1 = 0.9$, $\alpha_2 = 0.999$, $\varepsilon = 0$, and $\ell = 0.05$.

Ans.: Using the results from prior exercises, we find that

$$m_{dw}^{\text{corr}} = 5 \quad \text{and} \quad r_{dw}^{\text{corr}} = 25.$$

Therefore, $w^{(1)} = 0.1 - \frac{0.05}{\sqrt{25}} \times 5 = 0.05.$ □

As shown in Algorithm 10.1, the Adam optimizer begins by initializing α_1, α_2, and $\boldsymbol{w}^{(0)}$ to the appropriate values. In addition, the starting values for $\boldsymbol{m}_{dw}$ and $\boldsymbol{r}_{dw}$ are set and the learning rate is specified. The algorithm then enters the convergence loop where the momentum, rmsprop, and warm-up terms are

Algorithm 10.1 Adam optimization algorithm.

```
1: Input: training dataset $\boldsymbol{X}, \boldsymbol{y}$;
2: Set $\alpha_1 = 0.9,\ \alpha_2 = 0.999,\ \varepsilon = 10^{-8}$, Initialize $\boldsymbol{w}^{(0)}$.
3: Set $\boldsymbol{m}_{dw}^{(0)} = 0,\ \boldsymbol{r}_{dw}^{(0)} = 0, t = 0$, learning rate $= \ell$.
4: while $\boldsymbol{w}$ not converged do
6:     Compute momentum term
           $\boldsymbol{m}_{dw}^{(t+1)} = \alpha_1 \boldsymbol{m}_{dw}^{(t)} + (1-\alpha_1)\left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^{(t)}$
7:     Compute rmsprop term
           $\boldsymbol{r}_{dw}^{(t+1)} = \alpha_2 \boldsymbol{r}_{dw}^{(t)} + (1-\alpha_2)\left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^{(t)} \times \left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^{(t)}$
8:     Add warm-up adjustment
           $\boldsymbol{m}_{dw}^{corr} = \frac{\boldsymbol{m}_{dw}^{(t+1)}}{1-\alpha_1^{t+1}}, \quad \boldsymbol{r}_{dw}^{corr} = \frac{\boldsymbol{r}_{dw}^{(t+1)}}{1-\alpha_2^{t+1}}$
9:     Update estimate $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \ell \times \frac{\boldsymbol{m}_{dw}^{corr}}{\sqrt{\boldsymbol{r}_{dw}^{corr}}+\varepsilon}$
10:    Update $t = t + 1$
11: end while
12: Output: trained model;
```

applied. Because of the use of momentum and rmsprop, a larger learning rate can often be used as part of an adaptive scheme. Using the learning rate, the $\boldsymbol{w}$ update phase is executed and the iteration count is incremented. These steps continue until convergence.[2]

10.5 Summary

This chapter addressed multi-class classification problems using neural networks with two different loss functions. First, the categorical cross-entropy loss was compared to the categorical log-cosh loss. Their implementation with the softmax activation function was detailed. Next, the MNIST dataset was used as the demonstration vehicle to identify outlier images which were a set of unrecognizable hand-written numbers. This is a unique capability arising from the use of a robust loss function with neural networks. To conclude, the Adam optimization technique was presented in detail, including momentum, rmsprop, and warm-up techniques.

Problems

10.1 Using the categorical CE loss function, find the expression for the loss associated with the last row of the dataset depicted in Figure 10.1.

2 The algorithm shows the updates for the weights only. The biases are processed in the same manner. See Problem 10.6 for an example Python code.

10.2 Using the categorical LC loss function in its general form, find the expression for the loss associated with the last row of the dataset depicted in Figure 10.1.

10.3 Assume we have the choice to buy a product from one of three stores. Let the three stores be represented using one-hot encoding as follows: Store A = 100, Store B = 010, and Store C = 001. The features of the product are applied to a trained neural network. After prediction, let $z_1 = -4$, $z_2 = 0$, and $z_3 = 3$ be the raw outputs of the neural network associated with the categorical response. Compute the softmax probability for each output. Which store is predicted by the neural network to be the most suitable place to buy the product?

10.4 In this problem, the shift invariance property of the softmax function will be explored. Consider the following candidate for normalization:

$$\mathbb{N}(z_i) = \frac{z_i}{z_1 + z_2 + \cdots + z_k} \quad i = 1, \ldots, k.$$

The sum of all the normalized outputs is 1, which is the same as for the softmax function. However, if a shift value of b is applied to all of the raw outputs, what is the new expression for $\mathbb{N}(z_i + b)$? Now, perform the same operation with the softmax function, $\mathbb{S}(z_i)$. Compare the results and comment on the shift invariance of each one.

10.5 Gradient descent with momentum uses an exponential weighting of previous derivative terms using a momentum factor which is typically set to $\alpha_1 = 0.9$. The update equation is given by

$$\boldsymbol{m}_{dw}^{(t+1)} = \alpha_1 \boldsymbol{m}_{dw}^{(t)} + (1 - \alpha_1)\left(\frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^{(t)}.$$

Compute this term for 10 iterations and plot the coefficients on the previous 10 derivatives as a function of iteration number. Initially, $\boldsymbol{m}_{dw}^{(0)} = 0$. Set $\left(\frac{\partial J^{\text{LC}}(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^{(t)} = \Delta^{(t)}$ to make it easier to work out the recursion formula. (Hint: The first iteration would be $\boldsymbol{m}_{dw}^{(1)} = 0.9 \times 0 + 0.1\Delta^{(0)} = 0.1\Delta^{(0)}$. The second iteration would be $\boldsymbol{m}_{dw}^{(2)} = 0.9 \times \boldsymbol{m}_{dw}^{(1)} + 0.1 \times \Delta^{(1)}$. Plug in the quantity for $\boldsymbol{m}_{dw}^{(1)}$. Now let t = 2 and continue the recursion from there. You will end up with a long expression with $\Delta^{(1)}$ to $\Delta^{(10)}$ in the final iteration.)

10.6 (Project) The following uses the Adam optimizer to solve a simple linear regression problem similar to the telephone dataset of Chapter 2 using the log-cosh and least squares loss functions. Run the code and plot the loss vs. iteration, and the linear model with the data, to validate the results.

a) Import the libraries.

```
import numpy as np
import matplotlib.pyplot as plt
```

b) Enter the code for the Adam optimizer and convergence checking.

```
# Define class for Adam optimizer
class AdamOptim():
    # Set default values
    def __init__(self, learning_rate=0.1,
         alpha1=0.9, alpha2=0.999, \
            epsilon=1e-8):
        self.m_dw, self.r_dw = 0, 0
        self.m_db, self.r_db = 0, 0
        self.alpha1 = alpha1
        self.alpha2 = alpha2
        self.epsilon = epsilon
        self.learning_rate = learning_rate
    def update(self, t, w, b, dw, db):
        # Momentum
        self.m_dw = self.alpha1*self.m_dw + \
             (1-self.alpha1)*dw
        self.m_db = self.alpha1*self.m_db + \
             (1-self.alpha1)*db
        # rmsprop
        self.r_dw = self.alpha2*self.r_dw + \
             (1-self.alpha2)*(dw*dw)
        self.r_db = self.alpha2*self.r_db + \
             (1-self.alpha2)*(db*db)
        # Warm-up corrections
        m_dw_corr = self.m_dw/ \
            (1-self.alpha1**(t+1))
        m_db_corr = self.m_db/ \
            (1-self.alpha1**(t+1))
        r_dw_corr = self.r_dw/ \
            (1-self.alpha2**(t+1))
        r_db_corr = self.r_db/ \
            (1-self.alpha2**(t+1))
        # Gradient descent
        w = w - self.learning_rate*(m_dw_corr/ \
            (np.sqrt(r_dw_corr)+self.epsilon))
        b = b - self.learning_rate*(m_db_corr / \
            (np.sqrt(r_db_corr)+self.epsilon))
        return w, b
# Convergence of w and b
def check_convergence(dw, db):
    if (np.abs(dw) <= 0.00001) and (np.abs(db) \
        <= 0.00001):
        return True
    else:
        return False
```

c) Enter the log-cosh and least squares loss functions and gradient routines.

```
logcosh = lambda x: np.logaddexp(x, -x) - np.log(2)
# Define loss for LC and LS
def Loss_function(w_0,b_0,LC):
    diff = y-(x*w_0+b_0)
    if(LC):
        loss = np.sum(logcosh(a*diff)/a)
    else:
        loss = np.sum(diff*diff)/2.
    return loss

# Define gradients for LC and LS
def dLoss_function(w_0,b_0,LC):
    diff = y-(x*w_0+b_0)
    if(LC):
        dw = np.sum(np.tanh(a*diff)*(-x))
        db = np.sum(np.tanh(a*diff)*(-1))
    else:
        dw = np.sum((diff)*(-x))
        db = np.sum((diff)*(-1))
    return dw, db
```

d) Enter the dataset.

```
# Define x and y data
x = np.arange(0, 2.4, 0.1)
y= np.array([0.44, 0.47, 0.47, 0.59, 0.66, 0.73,
             0.81, 0.88, 1.06, 1.2, 1.35, 1.49,
             1.61, 2.12, 11.9, 12.4, 14.2, 15.9,
             18.2, 21.2, 4.3, 2.4, 2.7, 2.9 ])
```

e) Enter the code for the main loop and then run the code. By default, it uses the log-cosh loss function.

```
# Find w and b
LC = True
LS = False
reg_type = LC #Choose LC or LS
if(reg_type):
    print("Running LC linear regression")
else:
    print("Running LS linear regression")
a = 1.0
w, b  = 10, 10
```

```
adam = AdamOptim(learning_rate=0.1)
t = 0
loss = []
print("Iteration",t, "w =",format (w, '.3f'), \
      "b =",format (b, '.3f'))
converged, iter_count  = False, 2000
while not converged:
    loss.append(Loss_function(w,b,reg_type))
    dw, db = dLoss_function(w,b,reg_type)
    w_old = w
    w, b = adam.update(t,w=w, b=b, dw=dw, db=db)
    if check_convergence(dw, db):
        print("Iteration",t, "w =", \
                format (w, '.3f'),"b =", \
                format (b, '.3f'))
        break
    else:
        t+=1
        if (t%200 == 0):
            loss_at_t = Loss_function(w,b,reg_type)
            print("Loss =",format (loss_at_t, '.3f'))
            print("Iteration",t, "w =", \
                format (w, '.3f'),"b =", \
                format (b, '.3f'))
        if (t > iter_count):
            converged = True

print("Final result on iteration",t)
print("w =",format (w, '.3f'),\
      "b =",format (b, '.3f'))
```

f) Plot the loss and the linear model with the data.

```
plt.plot(loss)
plt.title("Loss as a function of iteration")
plt.xlabel("Loss")
plt.ylabel("iterations")
plt.show()
plt.scatter(x,y,color='black')
y_pred = w * x + b
plt.plot(x,y_pred)
plt.ylim(-5,25)
if(reg_type):
    plt.title("LC linear regression model")
    else:
        plt.title("LS linear regression model")
    plt.xlabel("Year")
    plt.ylabel('Calls')
    plt.show()
```

g) Change the code to run least squares and compare the results.

References

Haykin, S. (2009). *Neural Networks and Machine Learning*, 3e. Prentice-Hall.
James, G., Witten, D., Hastie, T. et al. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer.

11

Anomaly Detection and Evaluation Metrics

11.1 Introduction

Anomaly detection (see Aggarwal 2017) is a form of outlier detection where the outliers are at extreme locations and represent unusual and unexpected occurrences in the dataset. Identifying such anomalies is typically an unsupervised activity. There are a wide variety of anomaly detection methods reported in the literature and in common use in machine learning. The techniques to be described in this chapter generally involve repurposing well-known machine learning methods, such as k-nearest neighbors (k-NN), clustering methods, and tree-based approaches. The details of k-NN, DBSCAN, and Isolation Forest will be described as they are popular techniques in this category. This will be followed by a new method based on robust statistics and k-medians clustering called MADmax which is shown to provide better results than current methods.

For our purposes here, the datasets to be considered are those that would be used for clustering. In these datasets, there are data points near each other forming one or more clusters, along with other isolated data points far from all clusters that we seek to identify using an anomaly detector. Sometimes, a few points will be declared as anomalies when they are actually inliers. These are called false positives. In other cases, some anomalies are left undetected. These are the false negatives. Methods are required to evaluate different anomaly detectors based on their ability to limit the number of false designations. The chapter includes an examination of various metrics to compare different anomaly detectors.

11.2 Anomaly Detection Methods

In terms of definitions, the term "outlier" is an all-encompassing term whereas an "anomaly" is a specific type of outlier that is far from the rest of the data. In other words, an anomaly is an extreme outlier that is considered a rare or unusual event associated with suspicious activity such as bank fraud, spam, and network intrusion. More generally, anomalies are data points in a dataset that are somewhat isolated from all other data and occur infrequently. While the difference between outliers and anomalies is somewhat subjective, the terms are often used interchangeably since many of the techniques find both outliers and anomalies depending on the settings of various hyperparameters. However, keep in mind that the overall objective here is to find the isolated points in the dataset.

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.
© 2025 John Wiley & Sons, Inc. Published 2025 by John Wiley & Sons, Inc.
Companion Website: www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

11.2.1 *k*-Nearest Neighbors

The k-NN algorithm is a supervised learning technique that was originally developed for classification and regression problems (see Section 8.10.3). In this section, it will be redirected toward solving the anomaly detection problem as an unsupervised learning technique. The basic idea is to use the average distance of a given data point to its k nearest data points to determine whether it should be considered as an outlier. If the average distance is large, it is an outlier; if it is small, then it is an inlier.

The quality of the results depends on the number of points, k, used in the computation. In the supervised k-NN algorithm, k is some small odd number in binary classification so that a majority vote of the neighbors determines the prediction of a new point. In regression, it is the average of a small number of k neighbors. In anomaly detection, the average distance needs to be computed over many neighbors, so k is typically much larger. Usually, the Euclidean distance is used in the distance calculation. In some cases, the Manhattan distance can be used as well. Once the average distances for all n points in the dataset are calculated, a *threshold distance* is needed to determine if a given point is an inlier or an outlier. Both k and the *threshold* are user-specified hyperparameters. Their proper selection is essential in obtaining useful results.

After choosing k, it is common practice to plot the average distances as a histogram to identify the cutoff point between outliers and inliers. The histogram can be very useful in identifying the *threshold* beyond which the outliers can be easily identified. However, it is somewhat of a subjective decision since the locations of points in a multidimensional space cannot be visualized and therefore the line between outliers and inliers is often based on experience. The selection of k is somewhat easier. A number of different values of k could be used to find a suitable value that clearly separates outliers from inliers. Typically, a larger value of k such as $k = \sqrt{n}$ produces better results since the average is taken over many points. This makes the average distance more stable for the purposes of *threshold* selection.

Figure 11.1 shows a sample dataset on the left with four anomalies strategically positioned for demonstration purposes. In this example, we assume two features, namely $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The figure also shows what appears to be three separate clusters. Forming these clusters using k-means clustering could be troublesome due to the presence of the four anomalies (see Chapter 1). If we found these anomalies and removed them, then a clean dataset could be provided to the k-means approach. This is one of the reasons to use anomaly detection in data science. As shown, the anomalies are far from the clusters and are isolated from the inlier data. Therefore, this is a suitable example to illustrate the inner workings of a k-NN anomaly detector.

The k-NN algorithm can be used to identify these anomalies assuming a particular size of the neighborhood, say $k = 3$. First, the average distance of the three nearest neighbors is calculated for all data points. Then a histogram plot, shown on the right of Figure 11.1, is produced. A suitable number of bins must be selected to provide the necessary fidelity for selecting a threshold. In this example, 10 bins are chosen. The majority of the average distances are squeezed into the first two bins which is expected. The anomalies are identified as the extreme cases far away from the majority. These outliers can be separated out using a *threshold*, such as 0.3, in which case any data point with a distance above this threshold is considered an anomaly.

Figure 11.2 on the left shows the same dataset with the anomalies identified in gray and the inliers in black after using the k-NN method with $k = 3$ and *threshold* $= 0.3$. After removal of four detected anomalies from the dataset, the histogram on the right is produced. Note that the distribution of distances

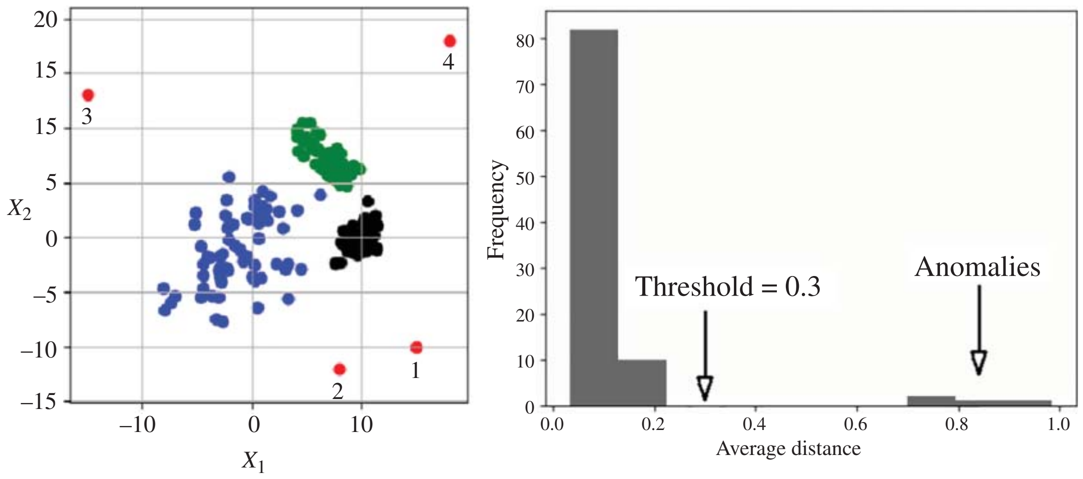


Figure 11.1 Example of a dataset with anomalies numbered from 1 to 4, with two high-density and one low-density clusters. The histogram on the right shows anomalies between the distances 0.7 and 1.0.

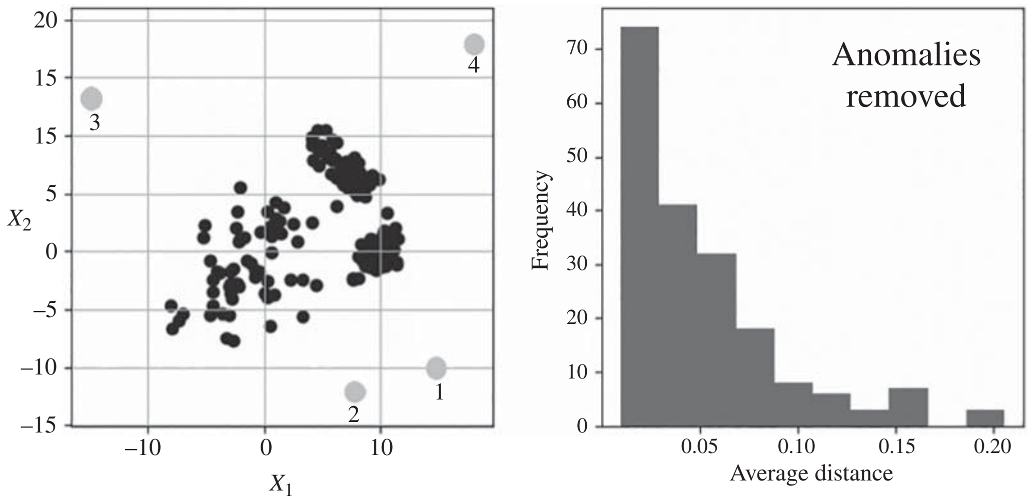


Figure 11.2 On the left, the same dataset as Figure 11.1 with the outliers detected using k-NN is shown in gray. The histogram with anomalies removed on the right is different from Figure 11.1. The average distances all lie between 0 and 0.25 instead of 0.0 and 1.0.

is more contiguous with no extreme cases. This is what is expected after outlier removal. The dataset after the anomalies are removed would be appropriate for a k-means clustering algorithm.

The approach above relies on the manual selection of k and the *threshold*. Consider the automated selection of these hyperparameters. The choice of k is an important step in obtaining satisfactory results, as mentioned earlier. In the supervised k-NN algorithm, a small value of k is appropriate. However, in the unsupervised algorithm for anomaly detection, a larger value is needed because there could be small groups of outliers near each other that would remain undetected if k were small. The *threshold* can be selected only if a suitable k is found. Hence, the development of an automated k-NN requires studying the characteristics of a range of k values and *thresholds*.

The upper diagram of Figure 11.3 shows a contrived dataset with one cluster containing 190 inliers and 11 scattered outliers, each of which has been numbered. The average distance from each point to all other points in its neighborhood as a function of k is shown in the plot in the lower portion of the figure. A total of 201 lines are plotted in that figure, one for each point in the dataset. The purpose of this figure is to devise a method to automatically select the k value, which is a vertical line, and the *threshold*, which is a horizontal line.

To understand Figure 11.3 in more detail, consider point 1 in the upper diagram. It is an anomaly that is relatively far from the other points. In the lower panel, the curve associated with this point is labeled with 1. It begins with a large distance to its nearest neighbor for $k = 1$ and continues to increase as k gets larger, meaning more neighbors are included in the average distance. A similar behavior is observed by point 2, the curve for which is given by label 2.

However, when examining points 3–11, it is a different matter. Their initial distances to nearest neighbors (with $k = 1$) are relatively small because there are other outliers in their immediate vicinity. It is only when k is increased to a large enough value that their average distances begin to increase significantly. These points emerge from the average distances associated with the inliers to join the average distances of the other outliers (points 1 and 2). Clearly, a large value of k is needed to separate inliers from outliers when there are groups of outliers.

As an example, a value of $n/3$ would be sufficient to separate out all the outliers from inliers as indicated by the vertical line at $k = 60$, which represents a suitable value of k. More generally, a value of $k = n/5$ gives good results. This fixed value can be used with most datasets, assuming that fewer than 20% of the data are outliers.

Next, if we examine the vertical axis, it represents the range of average distances in our dataset. We need to define a horizontal line that separates the outliers from the inliers. A suitable choice would be 10.0. In fact, the threshold for this example could be set to any value between 10 and 12.5 to obtain the same results. A value less than 10.0 would incorrectly make point 12 an anomaly. The *threshold* value (horizontal line) is usually obtained from a histogram of the distances produced by binning the values along the vertical line at $k = 60$, as shown in Figure 11.3.

It is possible to take the histogram data and find an appropriate threshold by identifying the distances that are large (outliers) from the ones that are small (inliers). This can be done automatically with a suitable algorithm that takes the frequency counts of the average distances, with say 30 bins, and looks through the bins to select the threshold. Using this strategy, the k-NN anomaly detection method can be fully automated. It is referred to as **auto-*k*NN** later in this chapter.

The k-NN algorithm is simple and can be very effective in identifying anomalies and outliers. However, one of the main drawbacks is the amount of computation needed to calculate the average distances for all points. If the number of features and/or number of data points is large, the computational cost may be prohibitive. The total number of distance calculations is $\frac{n^2-n}{2}$, thus the computational complexity is $O(n^2)$, where n is the number of data points.[1] Therefore, it can only handle small to medium-sized datasets.

1 The Big O(·) notation describes the upper bound of an algorithm's running time, indicating its efficiency as the input size grows.

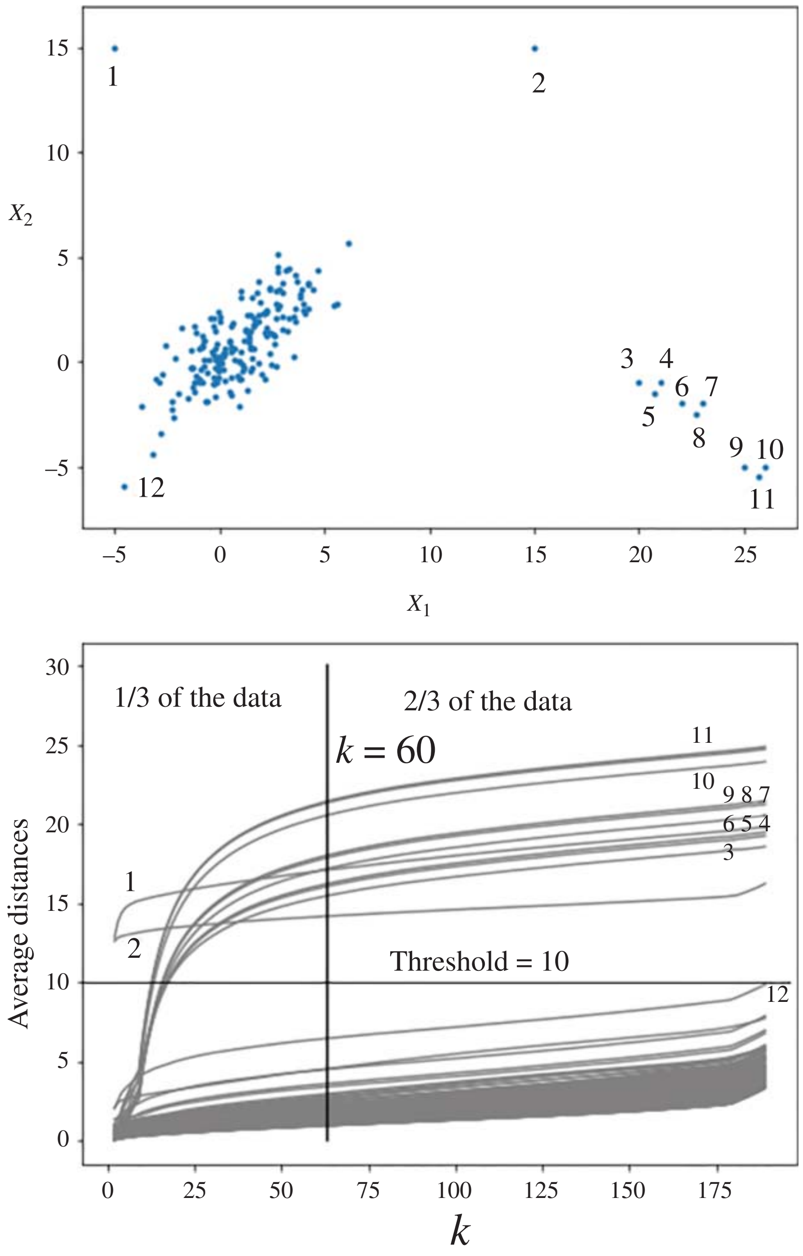


Figure 11.3 Average distance values as a function of *k*.

11.2.2 DBSCAN

To handle very large datasets, an approach based on clustering can be very effective. The idea is that if clusters can be formed using the data, then any points not in the clusters must be anomalies. This is the rationale behind an approach called DBSCAN which was originally developed for robust clustering. As such, it is an alternative to the k-medians clustering approach described in Chapter 1. DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. This density-based clustering method attempts to identify high-density regions of the feature space to create many clusters that can take on arbitrary sizes and shapes. The regions between clusters are defined as low-density areas where outliers will be found.

Every data point in the dataset is placed into one of three categories. A **core point** is one that is in close proximity to a given number of other points in its immediate vicinity defined by a radius, ε. A **border point** is defined as a point with not enough neighbors within this radius. The remaining points that are not identified as core or border points are declared as **outliers**. Hence, outliers are found as a side effect of the clustering process. The user-defined hyperparameters in DBSCAN are the radius, (referred to as `eps`), and the minimum number of points to identify a core point (referred to as `MinPts`).

Figure 11.4 illustrates the DBSCAN definitions on a small dataset showing the core points, border points, and outliers. The radius value used to decide on the core vs. border points is `eps` = 0.2 in this example, and the minimum number of neighboring points is `MinPts` = 10. Regardless of their shapes, DBSCAN will be able to find all clusters in the dataset. Originally, DBSCAN was intended to be used as a clustering method but it inherently detects outliers so it is often repurposed as an anomaly detector. For our purposes, core and border points will be considered as inliers. The rest are declared as anomalies.

To understand the effect of `eps` and `MinPts` on the results, consider Figure 11.5 which shows DBSCAN used as an anomaly detector on a dataset given earlier in Figure 11.1. Each plot shows the

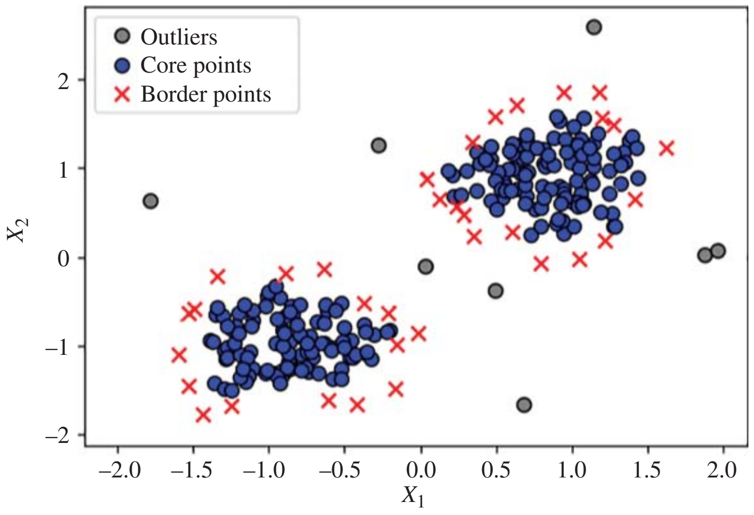


Figure 11.4 Example of DBSCAN applied to a randomly generated dataset showing the border and core points, and the outliers, given that `eps` = 0.2 and `MinPts` = 10.

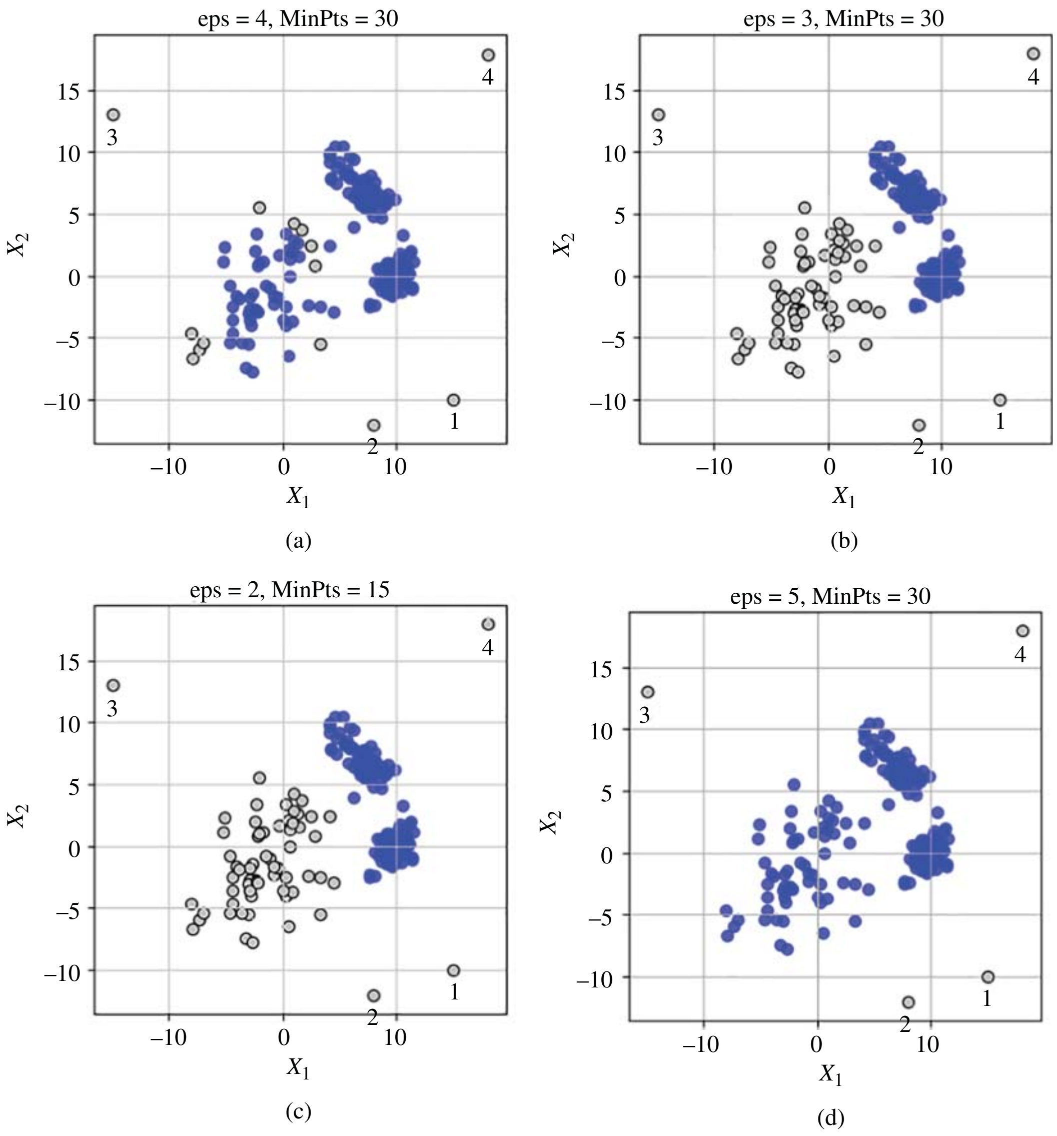


Figure 11.5 DBSCAN applied to the same dataset given in Figure 11.1 with (a) `eps` = 4 and `MinPts` = 30, (b) `eps` = 3 and `MinPts` = 30, (c) `eps` = 2 and `MinPts` = 15, and (d) `eps` = 5 and `MinPts` = 30. Detected outliers in gray. True outliers are numbered 1–4.

DBSCAN results for different values of `eps` and `MinPts`. As mentioned earlier, the dataset has three clusters, two that are high density and one of low density. For the different settings listed in the figure, all four of the true anomalies are detected in only case (d). In fact, more than four anomalies are found in cases (a), (b), and (c). As depicted in the figure, all the false positives are from the low-density cluster and none are reported from the high-density clusters. Although this example is contrived for demonstration purposes, it is clear that the DBSCAN might not produce an optimal solution when the dataset has density differences among clusters since only one value of `eps` is specified. Furthermore, in a multidimensional setting, it is not easy to determine the proper settings without trial-and-error or domain knowledge.

Generally speaking, DBSCAN provides acceptable results in most cases as long as `eps` and `MinPts` are set properly. To illustrate this further, consider the four plots given in Figure 11.6 where the inliers are highlighted in gray and the outliers found by DBSCAN are highlighted in black. In the top-left corner, all 12 anomalies were found. In the second example on the top-right, there are 9 anomalies but 10 outliers were found. Only one data point is at the boundary of being an inlier or an outlier. The others are correctly reported as anomalies. In the bottom-left case, 11 anomalies were correctly detected. Finally, in the bottom-right case, 10 anomalies were found, although it seems there are several undetected anomalies. We note that by adjusting `eps` and `MinPts`, the desired set of anomalies can be produced but this would be difficult in high dimensions where visualization is not possible.

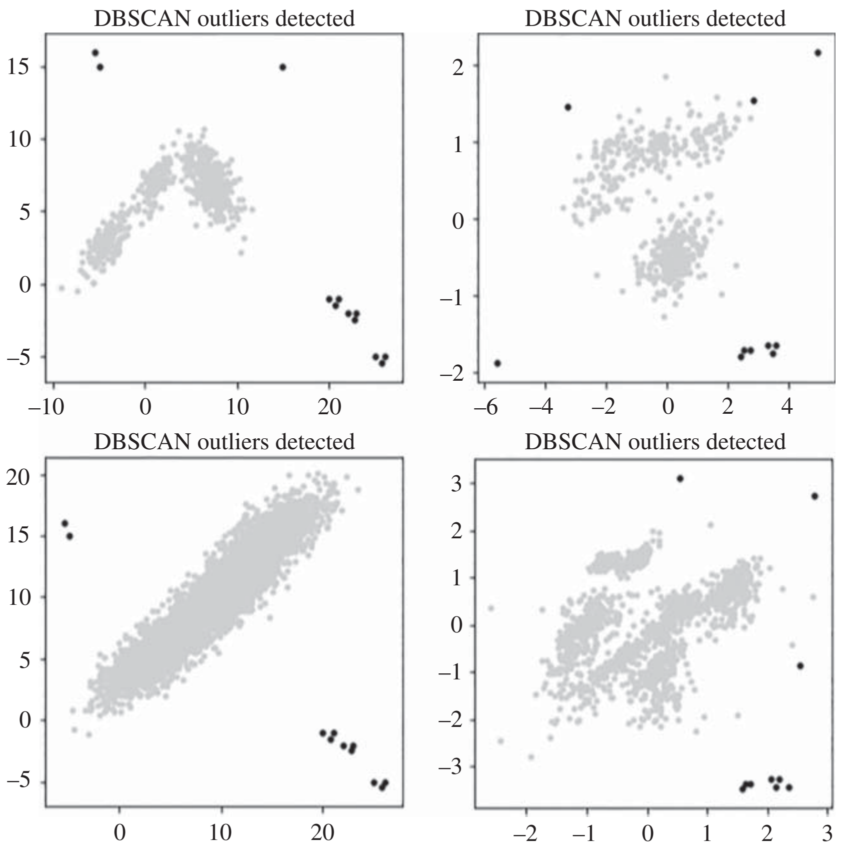


Figure 11.6 DBSCAN (`eps` = 4.5, `MinPts` = 30) applied to four synthesized datasets with inliers (gray) and detected outliers/anomalies (black).

DBSCAN is much faster than *k*-NN for large problems and produces the desired results if `eps` and `MinPts` are selected properly. Its computation time is similar to *k*-means clustering since it is performing a similar task. However, the proper selection of the two hyperparameters is not straightforward and may require a few iterations which increases the overall computation time. Hence, it could be rather expensive for extremely large datasets.

11.2.3 Isolation Forest

One of the fastest methods available for anomaly detection is referred to as Isolation Forest (IF). It is a popular and effective method for very large problems based on decision trees (see Chapter 8). A large number of random trees are built and then used to identify anomalies by examining the average tree depth of each data point. Those with the shortest average depth are declared as anomalies, based on an anomaly score. The approach can be viewed as an unsupervised version of *Random Forest* (RF), a robust supervised learning method for regression and classification problems. However, the goals of IF and RF are completely different.

The basic idea in IF is that anomalies tend to be "loners," isolated from the rest of the data. Because of this characteristic, if a random tree is constructed from the data, outliers tend to end up near the root of the tree. However, building one gigantic tree using the data would be expensive. Instead, the approach in IF is to subsample the dataset multiple times (like RF) and create a tree for each subsample. Figure 11.7 is a representation of 100 possible trees generated based on different selected subsets of the data. Not all points may appear in all trees due to the effect of subsampling (see data points 1 and 2 in the figure). Outliers are identified if their depth, on average, is shallow across all of the generated trees. For example, point 1 in the figure is embedded deep in many of the trees so it is unlikely to be an anomaly. However, point 2 has a shallow depth so it is likely to be an anomaly. To classify each point, an anomaly score based on the average depth is used along with a threshold to declare it an outlier or inlier.

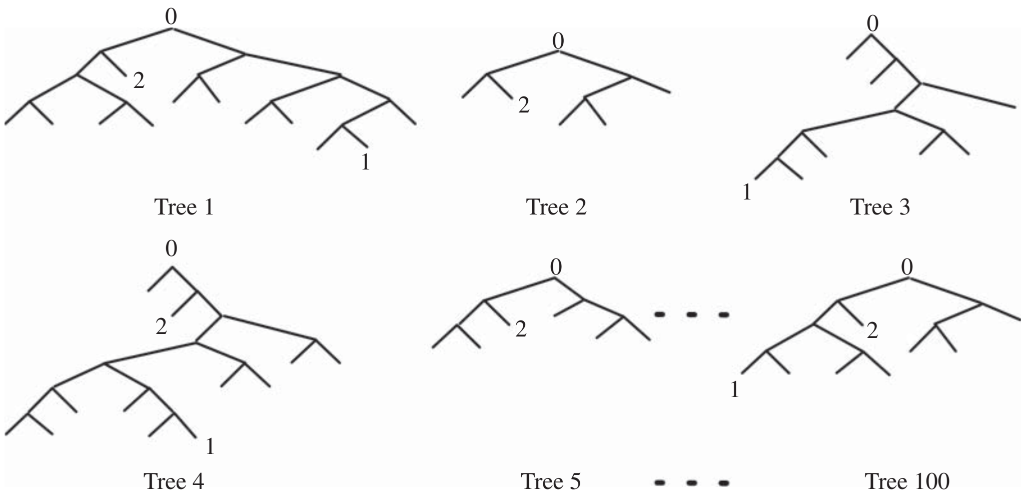


Figure 11.7 Illustration of random tree generation for Isolation Forest.

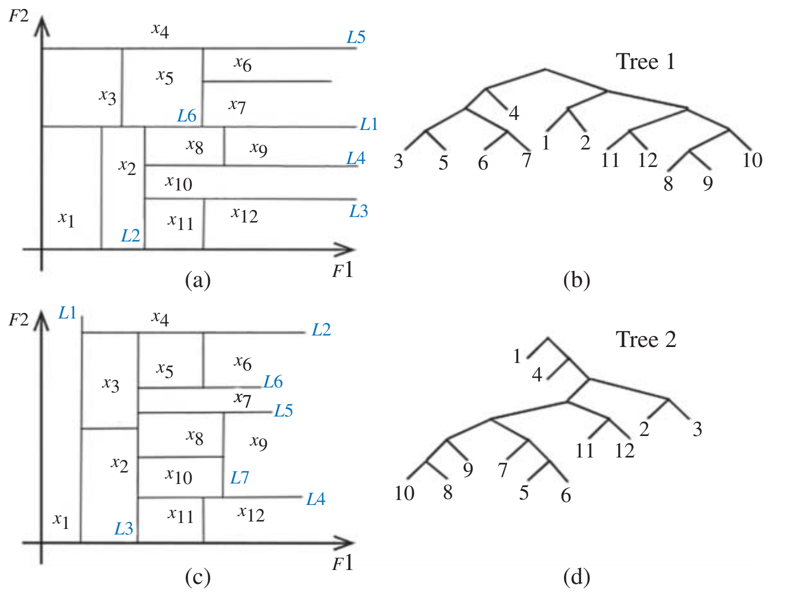


Figure 11.8 Example of Isolation Trees: (a) plot showing the splitting of the sub-sample dataset, (b) the corresponding Tree1 based on part (a), (c) another splitting of the same subsample set with different features, and (d) the corresponding Tree2 based on part (c).

Figure 11.8 shows a simplified example of how the trees are constructed so that anomaly scores can be calculated. In this example, the construction of two trees is shown considering a dataset with four features, F1, F2, F3, and F4, to illustrate the nondeterministic nature of IF. To begin, assume there are 12 data points in the first subsample (denoted as "x" with a label number) and two features, namely, $F1$ and $F2$. In Figure 11.8(a), $L1$ divides the subsample into two regions using a random division of the $F2$ feature (unlike RF which uses specific criteria such as Gini value or entropy function). Then, the lower region is further subdivided randomly along $L2$ using the $F1$ feature this time, followed by $L3$ and $L4$ considering the $F2$ feature, and so on and so forth. The final tree, Tree1, is shown in Figure 11.8(b).

In Figure 11.8(c), the subsample for features $F1$ and $F4$ is selected with 12 data points.[2] The first split begins using feature $F1$ and then $F4$ and continues in a similar fashion as above to generate Tree2. It should be stressed that rows and columns selected from the dataset, and the splitting locations are all done at random for each tree. It is unlikely that any two trees will be the same. On top of this, the selections change from one run to the next which makes IF highly nondeterministic such that the results are not repeatable from run to run.

Now consider the process of scoring based on these two trees, assuming they were generated in the same execution of IF. Examining Tree1 in Figure 11.8(b), the depth for point 1 is 3, while in Tree2 in Figure 11.8(d), the depth is 1. The average depth is 2 for this point. Let $\boldsymbol{x}$ be a specific point of interest in the dataset. In mathematical notation, we can define $h(\boldsymbol{x})$ as the random variable associated with the depth of $\boldsymbol{x}$ within a tree and we want to find $\mathbb{E}[h(\boldsymbol{x})]$, essentially the average of all the $h(\boldsymbol{x})$ values. For point 1,

2 Although there can be any random number of points in a subsample, the same 12 are selected in this example for convenience in anomaly scoring to be described shortly.

$h(\boldsymbol{x}_1) = 3$ for Tree 1 and $h(\boldsymbol{x}_1) = 1$ for Tree 2. Therefore, $\mathbb{E}[h(\boldsymbol{x}_1)] = (3 + 1)/2 = 2.0$. After all the trees are created, the average depth of every point is computed in this way. Each data point can now be assigned an anomaly score. The following equation is used to calculate the anomaly score:

$$s(\boldsymbol{x}, n) = 2^{-\frac{\mathbb{E}[h(\boldsymbol{x})]}{c(n)}}, \tag{11.2.1}$$

where $\mathbb{E}[h(\boldsymbol{x})]$ is replaced with the average depth of a given data point $\boldsymbol{x}$ across all trees. For a dataset with n points, $c(n)$ is a normalizing factor based on the average of the path lengths of all data points. It can be computed using

$$c(n) = 2(\log(n - 1) - 0.577) - \frac{2(n - 1)}{n}. \tag{11.2.2}$$

Table 11.1 shows the anomaly scores based on Figure 11.8. Points 1 and 4 seem to have the highest scores. Examining Figure 11.8, both points seem to be at the edge of the dataset. On the other hand, point 8 is deeply embedded in the data so it has the lowest score. This provides some insight into the inner workings of the IF approach in identifying anomalies. Points on the periphery of the data are candidates for being outliers while points embedded inside the bulk of the data are not. Using a threshold of 0.4 would be effective at finding the two anomalies in this example (although the example is too small to declare any points as anomalies).

Exercise: For the Isolation Forest shown in Figure 11.8, compute the anomaly scores for points 4 and 10.

Ans.: Since $n = 12$, we find that $c(n) = 2(\log(n - 1) - 0.577) - \frac{2(n-1)}{n} = 1.8$. For point 10, $\mathbb{E}[h(\boldsymbol{x})] \approx (4 + 7)/2 = 5.5$. Therefore, we have

$$s(\boldsymbol{x}, n) = 2^{-\frac{\mathbb{E}[h(\boldsymbol{x})]}{c(n)}} = 2^{-\frac{5.5}{1.8}} = 0.121.$$

Table 11.1 Anomaly score calculations for Isolation Forest.

Point	**Tree1 depth**	**Tree2 depth**	$\mathbb{E}[h(\boldsymbol{x})]$	**Anomaly score**
1	3	1	2.0	0.465
2	3	4	3.5	0.261
3	4	4	4.0	0.216
4	2	2	2.0	0.465
5	4	7	5.5	0.121
6	4	7	5.5	0.121
7	4	6	5.0	0.147
8	5	7	6.0	0.100
9	5	6	5.5	0.121
10	4	7	5.5	0.121
11	4	5	4.5	0.178
12	4	5	4.5	0.178

For point 4, we follow the same steps as above to obtain

$$s(\boldsymbol{x}, n) = 2^{-\frac{\mathbb{E}[h(\boldsymbol{x})]}{c(n)}} = 2^{-\frac{2.0}{1.8}} = 0.465. \qquad \square$$

Based on the calculations, point 4 is more likely to be an anomaly compared to point 10. Again, in this small example, there are no outliers or anomalies but the purpose here is to illustrate the basic operation of the Isolation Forest.

The required number of random trees needed in IF is based on when the average depth of the points converges to a consistent set of values. Typically 100 trees are sufficient to obtain acceptable results. The shapes and sizes of the trees will all vary but after about 100 trees, the average depth does not change significantly. Furthermore, the values used to split the features are randomized. In this way, the average of all the trees is regularized and suitable for scoring and anomaly detection.

To evaluate the performance of IF, we use the same dataset shown earlier in Figure 11.1. IF uses 100 subsampled trees for this dataset with 201 points. Figures 11.9(a), (b), (c), and (d) show the different anomalies obtained with the anomaly score threshold set to 0.87, 0.86, 0.78, and 0.5, respectively. In Figure 11.9(a), only three anomalies were detected using 0.87 as the threshold. Lowering it to 0.86 identifies four anomalies, as shown in Figure 11.9(b). Figures 11.9(c) and (d) show the behavior if the threshold is reduced further. It appears that the IF approach always focuses on the peripheral points of the dataset, as we found earlier. These examples demonstrate the fact that IF can occasionally fail to identify anomalies if the *threshold* hyperparameter is set incorrectly. In practice, there is a `contamination` hyperparameter that is accessible to the user rather than the *threshold* hyperparameter, but it acts in a similar manner. It is the fraction of the dataset that is expected to be anomalies and is typically set to a low value.

Figure 11.10 shows four cases (the same cases used to evaluate DBSCAN in the previous section) when IF is applied with `contamination`=0.02 in all cases. Note that in the first case in the top-left corner, the results are quite good. However, if the parameter is not adjusted on a case-by-case basis using domain knowledge, the results are unpredictable as seen in the other three plots. In each case, the level of `contamination` needs to be adjusted to obtain only the anomalies. If this is not done, the IF approach may declare that some data are outliers as shown in the bottom-left plot. Notice that IF "eats away at the data" from both ends in this case.

The main advantage of IF is its ability to handle high-dimensional data and very large datasets in a way that is much faster than other methods. This is due to the subsampling feature of the approach which naturally reduces the dimensionality of the data used to generate each tree. On the other hand, the random nature of subsampling implies that results tend to vary from one run to another. This nondeterministic behavior may require multiple runs to gain confidence in the results. Another issue is that clusters of outliers may present some difficulty in detection since IF attempts to isolate individual anomalies. There may also be some issues with binary and categorical features. Finally, the manual setting of the level of `contamination` controls the quality of the results. Regardless of these shortcomings, it is one of the most widely used detection methods today.

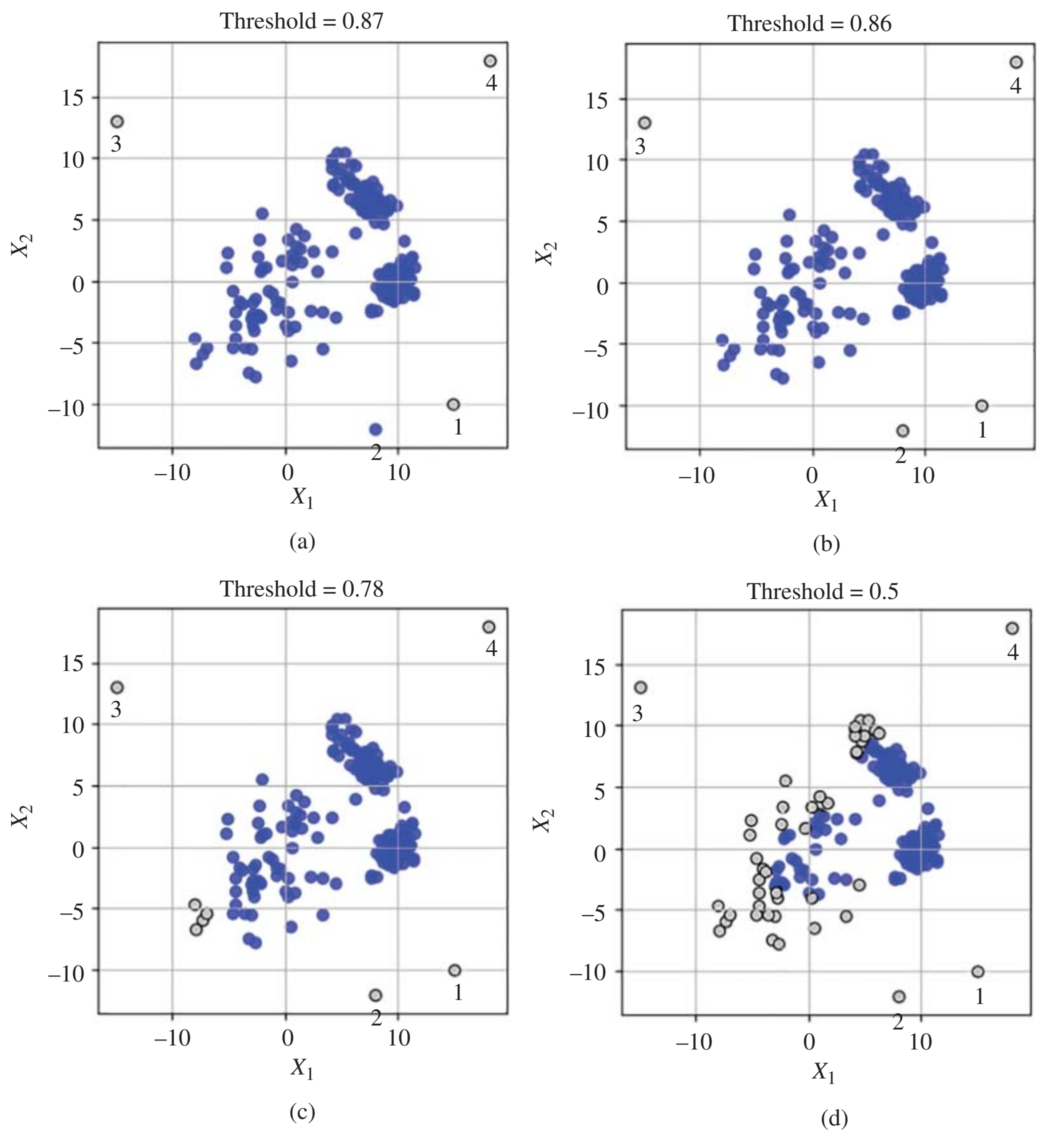


Figure 11.9 IF approach applied to the same dataset given in Figure 11.1 with different anomaly score thresholds. (a) *threshold* = 0.87, (b) *threshold* = 0.86, (c) *threshold* = 0.78, and (d) *threshold* = 0.50. Detected outliers in gray. True outliers are numbered 1–4.

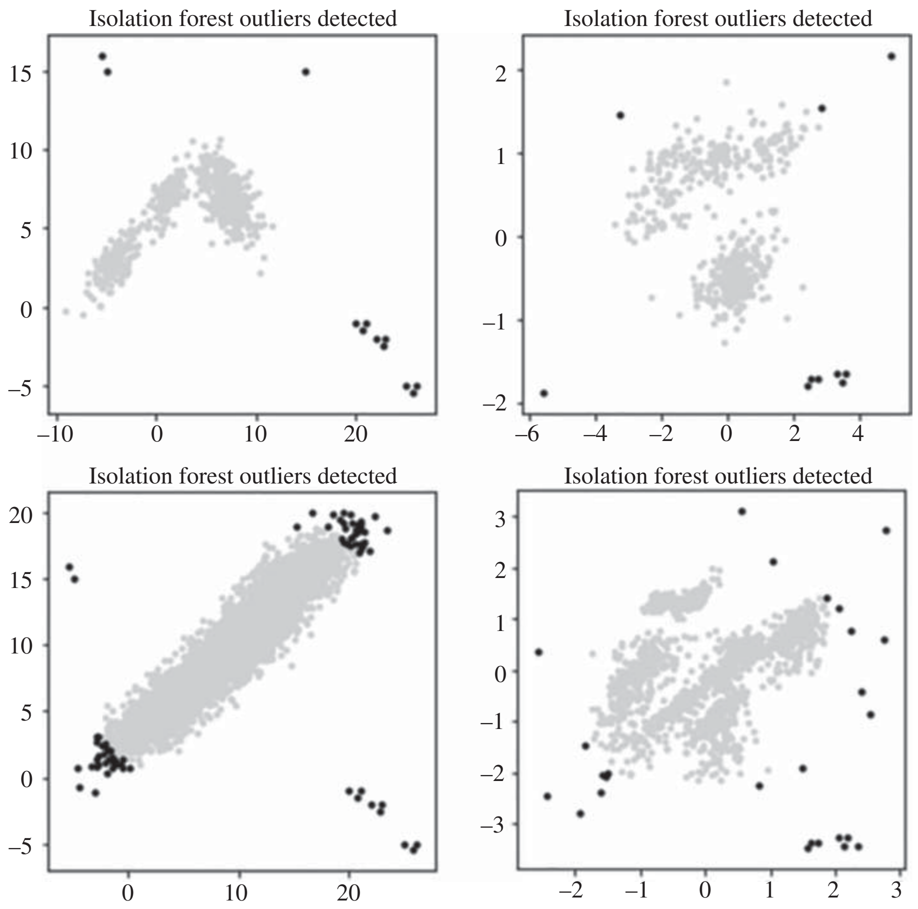


Figure 11.10 Isolation Forest (`contamination`=0.02) applied to four synthesized datasets with inliers (gray) and detected outliers/anomalies (black).

11.3 Anomaly Detection Using MADmax

The methods described previously require the specification of certain hyperparameters in order to obtain reasonable results. Furthermore, they do not directly make use of existing robust techniques or robust statistics. In this section, we describe an anomaly detection method that does not require user assistance and is based on various robust methods described throughout this book. It is called MADmax for reasons that will be clear shortly. The method utilizes a combination of the k-medians clustering algorithm of Chapter 1 and k-NN anomaly detection described earlier in this chapter. The automatic detection of anomalies without user intervention is a distinct advantage over other methods since they are usually prone to error if the hyperparameters are not set properly. In the case of MADmax, the parameters are automatically determined based on robust statistics, although external tuning is still possible if desired. The steps leading to the MADmax approach will be outlined in the sections to follow.

11.3.1 Robust Standardization

The first step in any anomaly detection method is to carry out some form of standardization of the data. This produces a dataset where all features are centered and scaled to lie in the same range of values. Typically, the mean and standard deviation are used to compute z-scores (see Chapter 4) which are then used in place of the original data. However, these statistics are not robust and they tend to hide outliers. Hence, in the pre-processing of data containing outliers, it is prudent to compute robust z-scores. The robust equivalent involves the use of the median and the MAD (see Chapters 1 and 4). In particular, the robust z-scores for each column j can be computed using

$$\boldsymbol{z}_j = \frac{\boldsymbol{x}_j - \text{median}(\boldsymbol{x}_j)}{\text{MADN}(\boldsymbol{x}_j)}, \quad j = 1, \dots, d. \tag{11.3.1}$$

Recall from Chapter 1 that $\text{MADN}(\boldsymbol{x}_j) = 1.426 \times \text{MAD}(\boldsymbol{x}_j)$, where

$$\text{MAD}(\boldsymbol{x}_j) = \underset{1\le i\le n}{\text{median}} \left(|x_{ij} - \text{median}(\boldsymbol{x}_j)|\right). \tag{11.3.2}$$

In the MADmax approach, it is essential that all data be robustly standardized, and it is also good practice for all other anomaly detection methods.

11.3.2 *k*-Medians Clustering

The second step in MADmax is to perform k-medians clustering, as described in Chapter 1. This is a robust version of k-means clustering. The number of clusters is usually specified by the user based on domain knowledge and experience. The difference here is that the number of clusters is computed automatically as

$$K = \left\lfloor \frac{n}{m} \right\rfloor, \tag{11.3.3}$$

where n is the number of points in the dataset and m is the average size of a cluster. The notation used above is the *floor* function to define K as the integer portion of n/m. In MADmax, the use of k-medians clustering is not to find clusters but rather to create as many *mini-clusters* as needed containing roughly m points in order to facilitate the process of finding a suitable threshold for anomaly detection. Typically, $m = 30$ for MADmax. For a dataset with 91 points, 3 mini-clusters will be created. For a larger dataset with 907 points, 30 mini-clusters will be created even if the dataset contains only 3 actual clusters. In contrast, DBSCAN will create three clusters in this case because there are only three clusters in the dataset. The large number of mini-clusters allows MADmax to handle any random shapes that the actual clusters may have, and any variations in density across clusters.

Example: The reason for k-medians clustering in MADmax is best understood through a detailed example. Consider Figure 11.11 which illustrates the manner in which a suitable threshold is determined. The original dataset shown in the top-left corner contains 420 points. It appears there are three clusters and four anomalies. The results of k-medians clustering are shown in the top-right corner. A total of $K = \lfloor n/m \rfloor = \lfloor 420/30 \rfloor = 14$ mini-clusters are created even though there are only three observable clusters. Each one has its own MAD value which is related to the density of the mini-cluster. We want to create circular regions around the median of each mini-cluster to try to encompass all the points in each mini-cluster. A good rule-of-thumb is to use the 4.5-MAD edit rule to compute the radius.

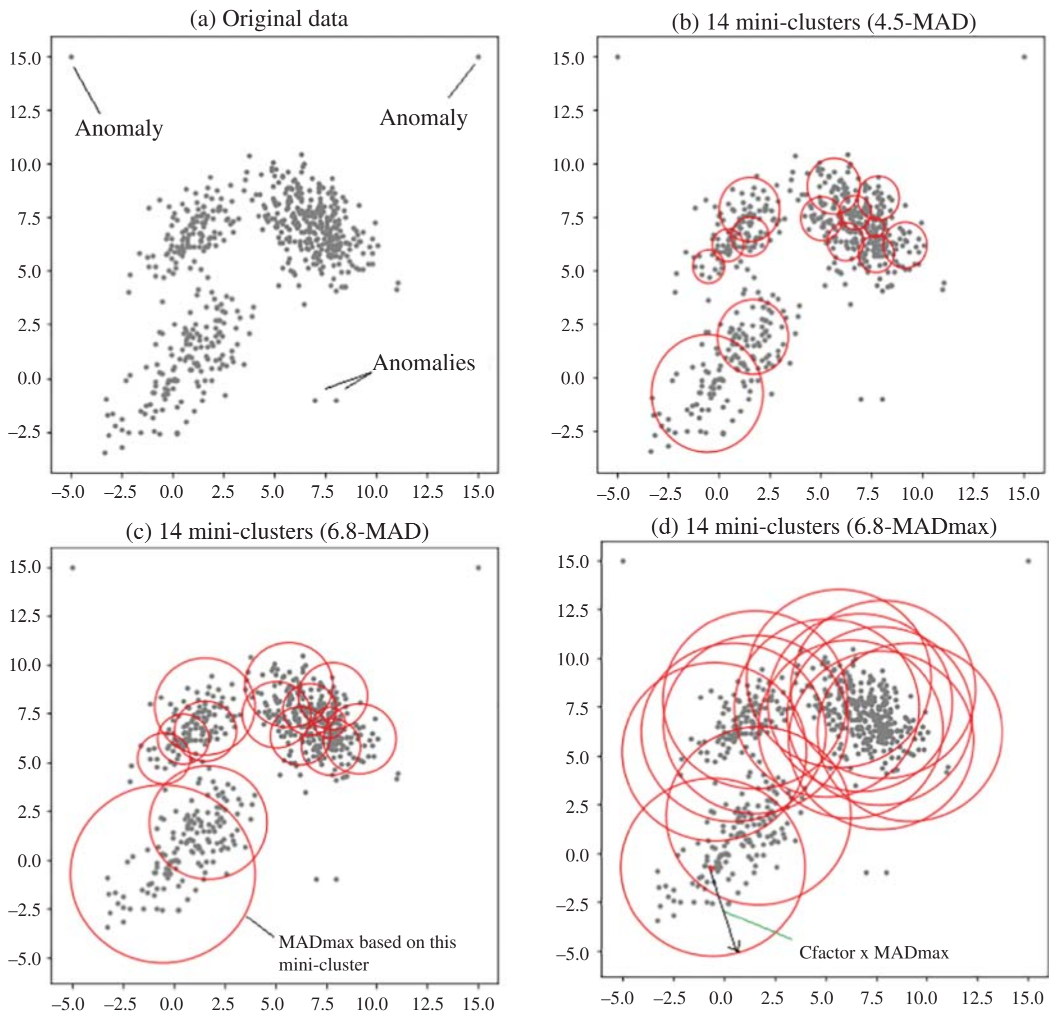


Figure 11.11 Development of the MADmax anomaly detection method. (a) Original dataset, (b) 14 mini-clusters indicated with circles of radius 4.5-MAD around the medians, (c) 14 mini-clusters with radius 6.8-MAD circles around the medians, and (d) 14 mini-clusters with radius 6.8-`MADmax` circles around the medians.

This is a robust statistic that was described in Chapter 4. Then, circular regions around each median can be produced, as shown in the top-right corner. Unfortunately, they do not cover all the points in the three clusters. Therefore, we need to increase the radius by a factor larger than 4.5.

For that purpose, a radius of 6.8-MAD is better at capturing more inlier data in the region around each median, as in the bottom-left figure. However, even 6.8-MAD does not cover all the points. This is true in this example and also for most datasets. Rather than using individual MAD values for each mini-cluster, it is better to use a single value for all mini-clusters. A sensible choice is the largest MAD value among the set of 14 clusters. The maximum of all 14 of the MAD values, called `MADmax`, is used to define a radius of 6.8×`MADmax` to create circular regions (or hyperspherical regions in higher dimensions) around all mini-clusters.[3] This covers all the inliers, as shown in the bottom-right corner. The mini-clusters all use the same `MADmax` value which establishes the boundary between inliers and outliers. Note that a cloud-like shape covers the inliers while the outliers and anomalies all lie outside the cloud. □

As mentioned before, one of the advantages of creating many mini-clusters is that it allows any cluster shape and density to be accommodated in the procedure. For example, a cluster in the form of a U-shape or S-shape can be decomposed into a number of mini-clusters of approximately 30 points each. The original shape of the clusters in the dataset becomes irrelevant, as does any variation in density within or between clusters.

11.3.3 Selecting MADmax

We now provide specific details of the third step in the approach involving the selection of `MADmax`. After k-medians clustering is performed on robustly standardized data to create K mini-clusters, the median and MAD are computed for each mini-cluster. The maximum of the mini-cluster MAD values is computed as follows:

$$\texttt{MADmax} = \max\left\{\mathrm{MAD}(\boldsymbol{d}_1), \mathrm{MAD}(\boldsymbol{d}_2), \dots, \mathrm{MAD}(\boldsymbol{d}_K)\right\}, \tag{11.3.4}$$

where $\boldsymbol{d}_i,\ i = 1, \dots, K$, represents the distances from all points in mini-cluster i to its median. The maximum MAD value across all mini-clusters, `MADmax`, can then be multiplied by a given amount, `Cfactor`, to establish a threshold for finding anomalies. That is, any point further than a threshold $t_{\mathrm{MC}} = \texttt{Cfactor} \times \texttt{MADmax}$ away from the median of its own mini-cluster will be considered an anomaly candidate. In our earlier example, we have been using `Cfactor=6.8` which is a good default setting. This method is quite reliable since it uses robust statistics to determine the threshold.

To understand why this approach is reasonable, recall that MAD is simply a robust version of the standard deviation which is a measure of the dispersion or spread of the data. If the data spread is large, then MAD will be large. However, if the data is tightly packed in a region, then the MAD will be small. Therefore, the MAD for each mini-cluster represents the local density of that mini-cluster and can be used to define an edit rule to trim data. However, one MAD value for all mini-clusters is required. Rather than using the smallest MAD or an average MAD, it is best to use the largest MAD in order to cover all the inliers. The data cloud associated with `Cfactor x MADmax` defines the region where outliers will not exist. In particular, anything outside the cloud of say 6.8×`MADmax` is a potential anomaly. Unfortunately, we cannot guarantee that it is an anomaly so further checking is necessary, as described next.

11.3.4 *k*-Nearest Neighbors (*k*-NN)

The fourth step is to take the candidate anomalies that lie outside the inlier cloud and apply the k-NN algorithm. Used in this way, there are two advantages over the k-NN method described earlier. The first is that only a few points are subjected to the k-NN test since only a small number of candidate anomalies are expected. The second is that the threshold and k are also determined automatically. For clarity, we use k_{NN} to refer to the number of nearest neighbors to be considered in the algorithm and t_{NN} as the threshold. Typically, k_{NN} is set to a high number such as $k_{\mathrm{NN}} = 25$. This means that the distances to the nearest 25 neighbors are averaged. The threshold is automatically set to $t_{\mathrm{NN}} = \texttt{Afactor} \times \texttt{MADmax}$,

3 To avoid confusion, keep in mind that "MADmax" is the algorithm and "`MADmax`" is the parameter.

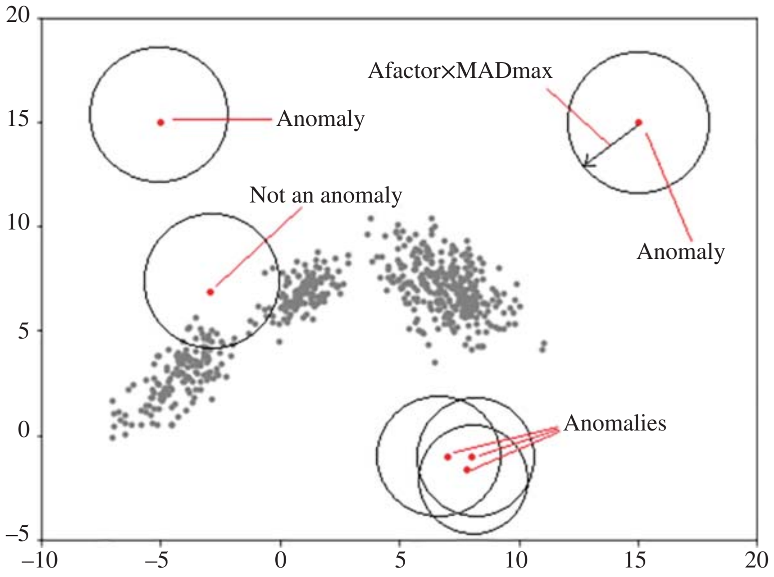


Figure 11.12 Post-processing of anomaly candidates using k-NN with acceptance criteria determined by a circular region defined using the radius `Afactor` × `MADmax`.

where `Afactor = 11.0` by default for reasons similar to those given for `Cfactor`. In doing so, all of the limitations of the basic k-NN approach are circumvented. In fact, this is a more effective use of k-NN since only a few points are involved (i.e. only anomaly candidates) and the threshold is automatically computed using `MADmax`.

The procedure is shown in Figure 11.12. The candidate anomalies have a circular region around them with a radius given by `Afactor` × `MADmax`. If the average distance of 25 nearest neighbors of a candidate anomaly is greater than this threshold, then it is declared an anomaly. In the figure, five of the candidates will be declared as anomalies, but the one near the inlier data will be not, because its radius encroaches on nearby inliers. Therefore, the average distance to the 25 nearest neighbors will be less than `Afactor` × `MADmax`. The three points in a group (in the lower part of the figure) will all be declared anomalies because each of their average distances to their nearest 25 neighbors will greatly exceed the size of the individual circles. In fact, all points in any small mini-clusters (those with less than five points) will be subjected to the k-NN test. In this manner, it is straightforward to check each candidate one by one using k-NN without incurring a large penalty.

11.3.5 *k*-Nearest Medians (*k*-NM)

With the basic ideas of MADmax covered, there are some special cases that need to be addressed. For example, creating mini-clusters in the manner described may produce some cases of mini-clusters that consist only of anomalies. If these points are far from each other, it results in a large MAD value which impacts the chosen `MADmax`. There may also be cases where we may have mini-clusters containing both inliers and outliers. These types of cases have to be identified and treated separately.

One approach to handling such special cases is to use the k-nearest medians (k-NM) approach which is a variant of k-NN that uses medians of mini-clusters rather than the original data points. In some sense, the medians are representatives of each mini-cluster and we can inspect their proximity to one another to assess whether or not the median is associated with the inliers or anomalies.

Assume that we have identified the K medians of the mini-clustered data. Then, when applying k-NM, we have effectively reduced the dimensionality of the problem space from $\mathbb{R}^n$ to $\mathbb{R}^K$. The problem is $O(K^2)$ instead of $O(n^2)$ which is a much lower computational complexity. In doing so, if we find any medians to be outliers, their associated mini-clusters will not be included in the specification of `MADmax`. Then, all data associated with that median are potential anomalies and they all need to be checked using k-NN as described in the previous section. The value of the threshold for the mini-cluster median test is dictated by $t_{NM} = \texttt{Mfactor} \times \texttt{MADmax}$, where `Mfactor` is another internal parameter used in the MADmax algorithm with a default value of 14.8.

Figure 11.13 illustrates this case with a scatter plot of medians for the inlier data and one median in the bottom-right corner for the outlier data. The mini-cluster associated with the outlier median may contain many anomalies, each of which must be checked using the k-NN algorithm. Of course, there may be many outlier medians to deal with. Therefore, one last issue involves choosing the proper value of k for k-NM, which we refer to as k_{NM}. In this case, a small value such as $k_{NM} = 1$ to 5 is appropriate. It specifies how many mini-clusters containing outliers are allowed to be in the same vicinity of one another. In particular, if we decide that three outlier mini-clusters are permitted next to each other as neighbors, then we set $k_{NM} = 3$. Beyond the overall steps of the MADmax algorithm that have been described, additional heuristics can be introduced to address any corner cases that may arise. The key aspect to note is that once the internal parameters are set, the MADmax approach is able to find anomalies in a variety of datasets without user intervention.

To summarize, the definitions of the internal parameters and multiplying factors for MADmax are:

- `MADmax`: the largest MAD value of all mini-clusters.
- t_{MC}: this defines a circle of radius $\texttt{Cfactor} \times \texttt{MADmax}$ around mini-cluster centers that contains all points associated with the mini-cluster.
- t_{NN}: this defines a circle of radius $\texttt{Afactor} \times \texttt{MADmax}$ around anomaly candidates to declare them as anomalies or inliers.

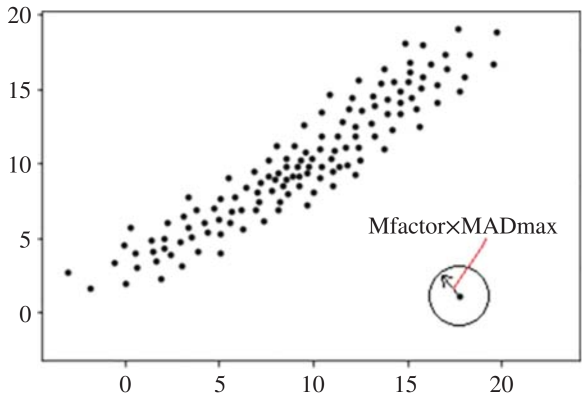


Figure 11.13 Scatter plot of medians to check if there are any mini-clusters of anomalies using a test criteria determined by $\texttt{Mfactor} \times \texttt{MADmax}$.

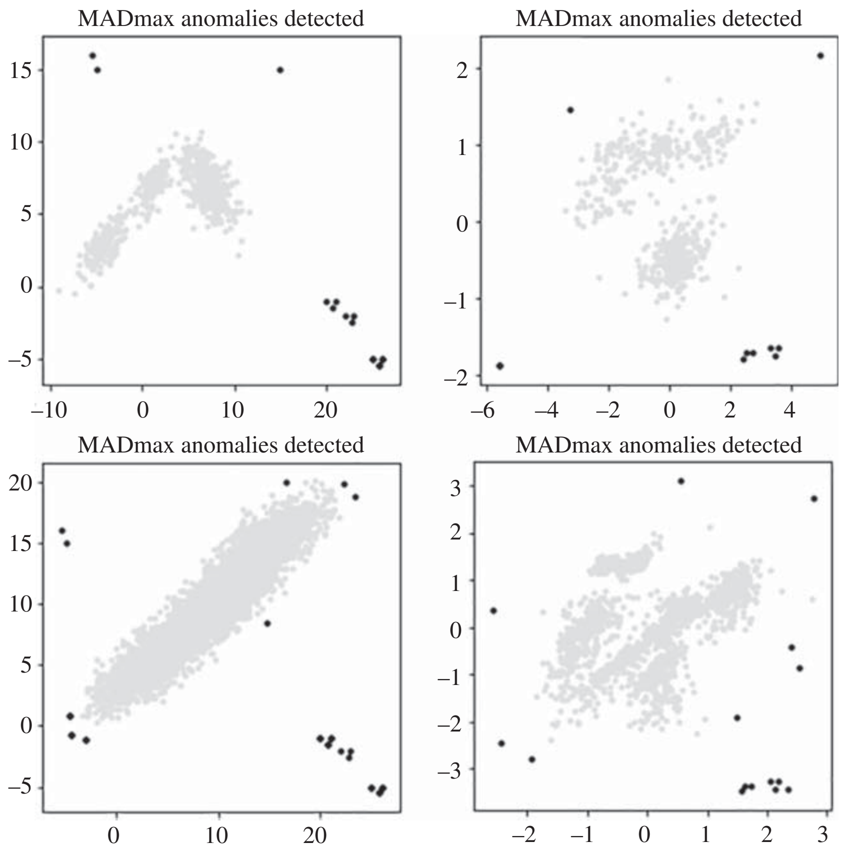


Figure 11.14 MADmax applied to four synthesized datasets with inliers (gray) and detected outliers/anomalies (black).

- k_{NN}: number of nearest neighbors to consider during the analysis of candidate anomalies.
- t_{NM}: this defines a circle of radius `Mfactor` × `MADmax` around the median of mini-clusters to determine if they are mini-clusters of anomalies.
- k_{NM}: number of nearest medians to consider in the k-nearest medians approach.

These parameters are set once and then used for all datasets. The reason that this is possible is due to the robust standardization carried out in the first step. Without it, the values would have to be adjusted for each dataset, which would be similar to other manually tuned anomaly detectors.

Figure 11.14 shows the four cases (used earlier to evaluate DBSCAN and IF) when MADmax is applied without modification of the parameters. The goal is to demonstrate the value and effectiveness of using robust statistics and robust techniques to detect anomalies. In all cases, the results are quite acceptable. There are a number of other refinements that can be applied to improve the overall performance on realistic datasets.

11.4 Qualitative Evaluation Methods

It is a challenge to compare different anomaly detection methods for a variety of reasons. For example, if there are four detection methods and three require manual adjustment of hyperparameters while the fourth does not, how should the comparison be carried out? Second, some methods are faster or can easily handle larger datasets compared to others but do not have the same level of accuracy as the others. Third, if all 4 methods detect 10 anomalies but they do not all agree on which ones are true positives and which ones are either false positives or false negatives, then how do we determine which one is best, especially in a multidimensional setting where visualization is not an option? And finally, if the correct number of anomalies is 10, what if each method obtains a different number of anomalies and only one predicts 10. How should we view the quality of the other detectors.

As a general statement, we have to accept the fact that anomaly detectors are hard to compare. Ideally, if a method automatically detects the anomalies without user intervention, it would be preferable. Then, if a method produces the correct number of anomalies, that makes it even more attractive. However, if it requires an inordinate amount of computer runtime, then its value may only be demonstrated on small to medium-sized datasets. The k-NN and MADmax approaches are accurate and can be made to work in an automated fashion. k-NN is more suitable for small or medium-sized problems. Both DBSCAN and MADmax can handle larger datasets although DBSCAN may require some tuning. Isolation Forest can operate on extremely large datasets but requires some amount of domain knowledge and user intervention.

With this backdrop, we now carry out qualitative comparisons of k-NN, DBSCAN, Isolation Forest and MADmax using a specific publicly available dataset. The four methods have noticeably different approaches to the problem but, at a basic level, they all measure some form of "distance" in their algorithms. These are converted into scores which are then used to rank data points. The rankings determine the designation of outliers or inliers based on a threshold. The overall methodology involves using each method as an unsupervised learning task on the same design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, with the option to tune certain hyperparameters if necessary. The goal is to assess their ability to find a set of known anomalies/outliers.

To illustrate the complications of comparing different methods, consider the wine dataset[4] which is known to contain outliers. The dataset consists of 130 observations ($n = 130$) and 13 features ($d = 13$). Therefore, it is not possible to visually examine results in a 13-dimensional space to compare different methods. In that case, how should we proceed and how do we adjudicate the identification of a data point as an outlier or inlier?

A good first step would be to invoke the auto-kNN approach to find the outliers and anomalies. This was mentioned in Section 11.2.1 whereby the values of k and the *threshold* are set automatically using heuristics. Although this method may be unsuitable for large datasets, it is quite suitable for the role of identifying true outliers on small datasets, which serves as a baseline when comparing other methods. Typically, a higher number of outliers are identified using this method, some of which are anomalies. If auto-kNN is applied to the wine dataset, a total of 14 outliers and anomalies are found. The next problem is to somehow visualize these outliers so that a comparison can be carried out.

4 The wine dataset is available in numerous places including on the site kaggle.com.

One way to do this is to select two features from the dataset and plot only the outliers in that two-dimensional space. Note that any two features can be used as long as the outlier positions are spread out enough to identify them visually. These outlier positions then become the targets for the other anomaly detectors to hit so that their accuracy can be assessed. This is shown in Figure 11.15 in the top-left corner. After running auto-*k*NN, the 14 shaded dots show the positions of the outliers using features `Alcohol` and `Malic Acid`. The other three methods must hit those targets to be considered as accurate.

The first anomaly detector to be evaluated is MADmax. This approach requires no user intervention. The dataset is simply fed to the algorithm and it reports its findings. The results are shown in the top-right corner of Figure 11.15. A total of nine anomalies are detected, each one hitting one of the targets set up by auto-*k*NN (see black dots). Of course, it misses five targets since only nine anomalies are found. For DBSCAN, a certain amount of tuning is necessary to obtain nine anomalies. The targets hit by DBSCAN

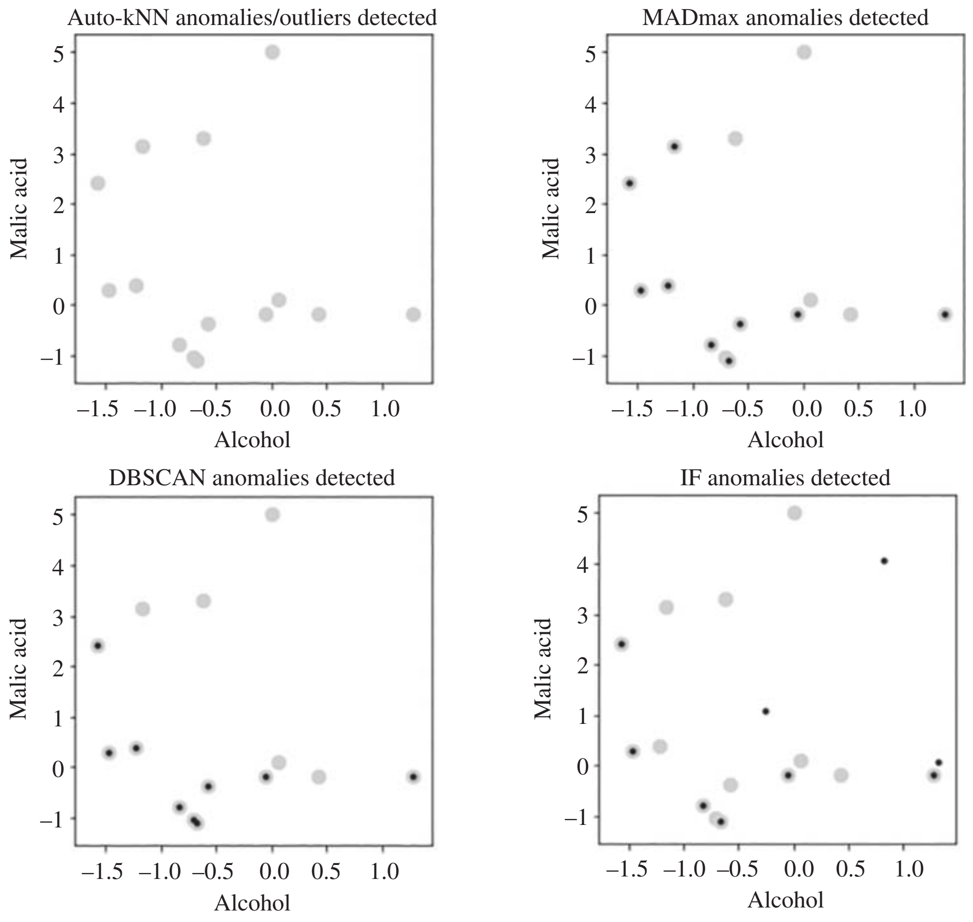


Figure 11.15 Comparing anomaly detection using the wine dataset with exactly nine anomalies allowed by each method.

are shown in the bottom-left corner of Figure 11.15. As mentioned above, DBSCAN can produce any number of outliers so, in the first attempt, the hyperparameters were adjusted to deliver nine anomalies. Of note is that fact that it does not hit all the same targets as MADmax. However, all 9 anomalies fall on one of the 14 targets and 5 targets are missed. The same cannot be said for IF. In that case, it produces three false positives and misses eight targets which are categorized as false negatives.

A second evaluation can be applied by forcing all the detectors to produce 14 outliers. This is shown in Figure 11.16. While MADmax does not require tuning, it can be adjusted manually if desired by changing the `Cfactor`. By doing so, all 14 outliers hit the targets set by auto-*k*NN. However, both DBSCAN and IF produce false negatives and false positives in this case. Using these values, the precision and recall can be computed (see next section). Furthermore, the area under the curve (AUC) of the receiver operating characteristics (ROC) could also be computed for the different methods. The reason is that, in an unsupervised setting, it is difficult to make claims about which points are inliers and which points are outliers

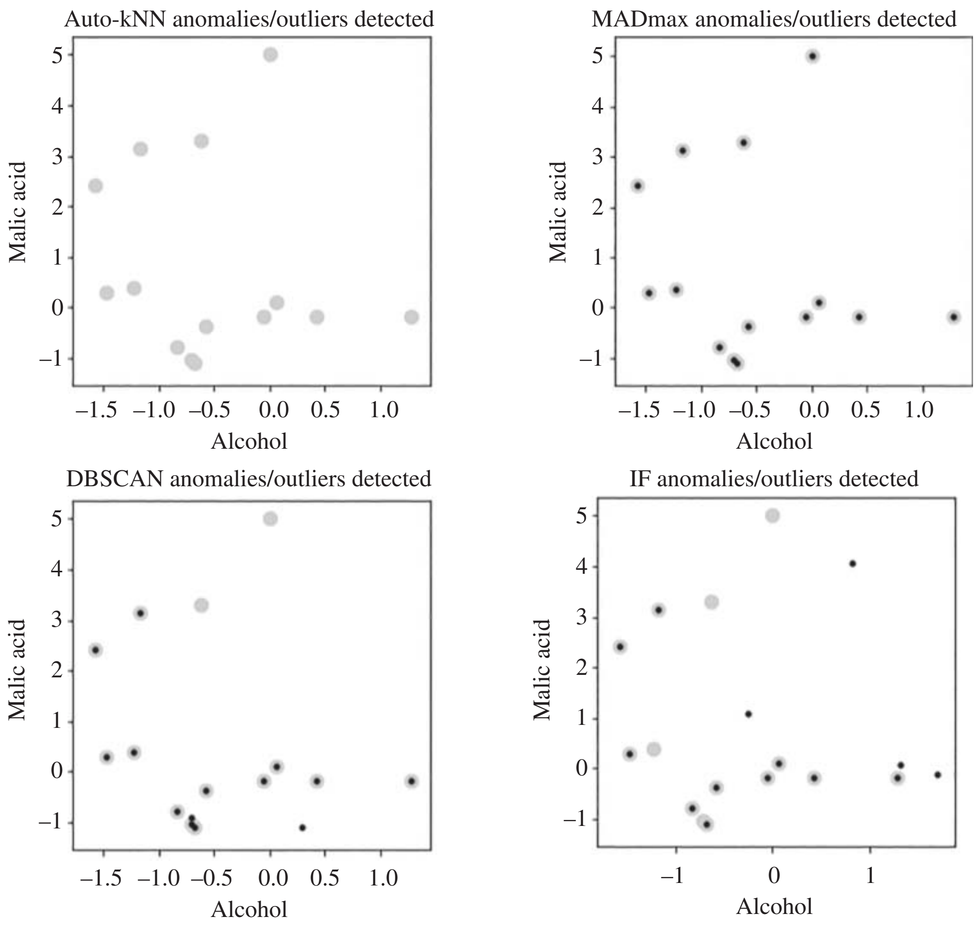


Figure 11.16 Comparing anomaly detection using the wine dataset with exactly 14 outliers required by each method.

and anomalies. Even with labeled data, there is no guarantee that the labeling is correct. However, the qualitative approach outlined here is very useful for multidimensional problems with $n \leq 20,000$ (due to limits on k-NN). Detailed analysis can be performed in this manner to build confidence in the tool that is finally selected for the anomaly detection task. Alternatively, comparisons across many benchmarks can be done in a similar fashion using row numbers of the dataset rather than the visualization approach shown here. In that way, the entire benchmarking process can be automated.

11.5 Quantitative Evaluation Methods

To develop quantitative metrics to compare different outlier detectors, the concepts described in Section 9.8 of Chapter 9 can be used as a starting point. The first step is to tabulate the results of an anomaly detector based on whether or not outliers and inliers were properly categorized. The definitions are provided in Table 11.2. If a true outlier is detected by an anomaly detector, it is considered a true positive (tp). If an inlier is erroneously considered an outlier, that is a false positive (fp). If an outlier is erroneously considered an inlier, that is a false negative (fn). Finally, if an inlier is declared to be an inlier, that is a true negative (tn). In evaluating the results of an anomaly detector, the recall and precision are both good metrics. The **precision** expresses what percentage of outliers detected by the anomaly detector are actually outliers. Typically, there is a preference for anomaly detectors with a higher precision. The **recall** expresses what percentage of the true outliers is detected by the anomaly detector. Together, these two metrics can be used as the initial quantitative assessment of the anomaly detector.

The **AUROC** can also be used to compare different approaches by plotting the true positive rate (TPR) against the false positive rate (FPR). Unlike the binary classification case, the meanings of true positive and false positive refer to whether or not an outlier was detected properly. As before, it is possible to plot the ROC curves to compare different detectors visually. However, for outliers, the plot is a sequence of steps (resembling an uneven staircase) with the number of steps equal to the number of outliers. Because of this, we use a different approach than for binary classification when computing the AUROC metric. Specifically, if we rank the outliers and inliers using their anomaly scores, we can formulate a compact equation for AUROC as follows,

$$\text{AUROC} = 1.0 - \frac{\sum_{i=1}^{M}(\text{rank of outlier } i) - M(M+1)/2}{M \times N}, \tag{11.5.1}$$

where N is the number of inliers and M is the number of outliers.

Table 11.2 Interpretation of confusion matrix for outliers. The four possible outcomes of tn, fn, tp, or fp are accumulated in this tabular format from the ground truth and detected outliers.

	True Inlier	True Outlier	Interpretation of Matrix Entries
Detected-Inlier	tn	fn	tn = true negative, fn = false negative
Detected-Outlier	fp	tp	fp = false positive, tp = true positive

Table 11.3 Computing the AUROC for different anomaly detectors using the ranks of inliers and outliers. 0 = inliers and 1 = outliers.

Rank	Ground truth	Detector 1	Detector 2	Detector 3
1	1	1	0	0
2	1	0	0	0
3	0	0	1	0
4	0	0	1	0
5	0	1	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	1
10	0	0	0	1

To understand this equation, consider Table 11.3 where the ground truth and the hypothetical results of three anomaly detectors are provided. There are only 10 data points in this small example of which two are outliers. The outliers are indicated with 1 and inliers with 0. The rankings of inliers and outliers are based on their respective anomaly scores, and rank 1 is the highest while rank 10 is the lowest. In the Ground Truth column, the two outliers are ranked 1 and 2. The AUROC for this case is always 1.0. A brute-force way to obtain this value is to carry out rank comparisons of outliers with inliers while counting the number of inliers above each outlier. A total of $M \times N$ comparisons will be performed. Starting with a count of 0, each time an inlier is ranked higher than an outlier, we add one to the count. Since there are no inliers above the outliers for the Ground Truth, the sum is 0. This is divided by $M \times N$ and then subtracted from 1.0. The same steps would be carried out for the three detectors to obtain their AUROC values.

There is an easier way to do this (without the tedious counting process) using Eq. (11.5.1). First, we sum the ranks of outliers. Then, we subtract the ideal value of this sum which, in general, for M outliers is $M(M+1)/2$. For most practical anomaly detectors, the sum of the outlier ranks will likely be higher than the ideal value. Finally, the difference between the two sums is divided by the total number of comparisons, and then subtracted from 1.0 to produce the AUROC. This a much easier method because it only requires the ranks of the outliers.

Exercise: What is the AUROC of each of detectors 1, 2, and 3 of Table 11.3 based on the formula given in Eq. (11.5.1)?

Ans.: We begin by noting that $M(M+1)/2 = 2(2+1)/2 = 3$, and $M \times N = (8)(2) = 16$. The ranks of the outliers for each detector can be summed easily by looking at the first column in the table. The AUROC of the three detectors are:

Detector 1: AUROC $= 1 - \frac{(1+5)-3}{16} = 1 - 3/16 = 13/16 = 0.8125$

Detector 2: AUROC $= 1 - \frac{(3+4)-3}{16} = 1 - 4/16 = 12/16 = 0.75$
Detector 3: AUROC $= 1 - \frac{(9+10)-3}{16} = 1 - 16/16 = 0$ □

Example: This example describes the steps used to produce an ROC plot for outliers, which resembles an uneven staircase as mentioned above. An example of an ROC curve is given in Figure 11.17. It is based on a dataset with $n = 100$ points of which 10 are outliers and 90 are inliers. In the ideal case (ground truth), the outliers are ranked as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, while inliers are ranked from 11 to 100. A candidate anomaly detector ranks the outliers as follows: 5, 8, 9, 11, 12, 13, 18, 23, 37, 57, with the remainder as inliers. To plot the ROC, we begin by assuming all data are inliers and place all 100 points in the Detected-Inlier row of Table 11.2. Therefore, tn $= 90$ and fn $= 10$, while fp $= 0$ and tp $= 0$. These values are used in the following equations:

$$\text{FPR} = \frac{\text{fp}}{\text{tn} + \text{fp}}, \qquad \text{TPR} = \frac{\text{tp}}{\text{tp} + \text{fn}}. \tag{11.5.2}$$

Initially, FPR $= 0$ and TPR $= 0$ which produces the first point in the ROC plot. Next, we sequence through the data in rank order. Going from rank 1 to rank n, if an inlier is encountered (indicated by 0 in the table), we decrease the value of tn by one and increase the value of fp by one. If an outlier is encountered (indicated by 1 in the table), then we decrease the value of fn by one and increase the value of tp by one. Then we recompute TPR and FPR. Each outlier produces a step in the staircase of the resulting ROC. For the outlier ranked 5, we have that FPR $= 4/(4+86) = 0.04$ and TPR $= 1/(1+9) = 0.1$. This constitutes the first step in Figure 11.17. There are a total of 10 steps, although the resolution does not allow all steps to be shown. The last point is given by TPR $= 10/(10+0) = 1.0$ and FPR $= 90/(0+90) = 1.0$. In this way, the TPR and FPR are computed (total of n data points) and then plotted to produce the ROC shown. □

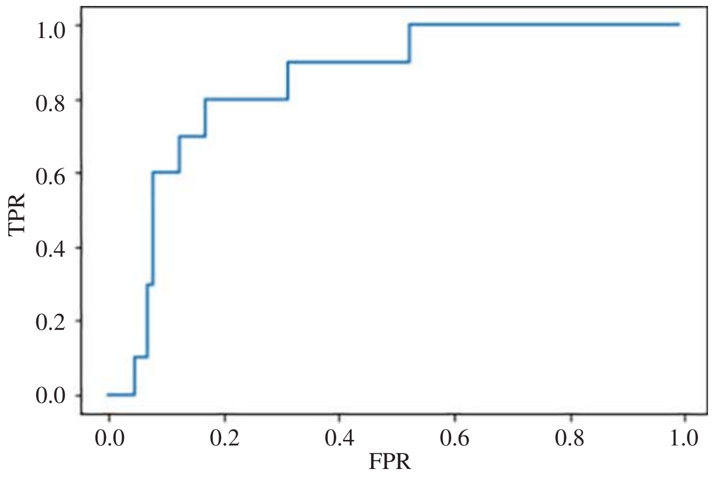


Figure 11.17 ROC plot for a candidate anomaly detector. There are a total of 10 steps (although it appears to have only 7 due to the resolution).

Exercise: What is the AUROC of the anomaly detector represented by the ROC curve in Figure 11.17?

Ans.: Here $N = 90$ and $M = 10$. Then $M(M+1)/2 = 55$. The sum of the ranks is $\sum \text{ranks} = 5+8+9+11+12+13+18+23+37+57 = 193$. Therefore, AUROC $= 1 - \frac{193-55}{10(90)} = 1 - 138/900 = 762/900 = 0.847$. □

If several detectors are plotted together, the ROC tells us how quickly an anomaly detector will find most of the outliers and which one is the first to find all outliers. Typically, the TPR will increase quickly from the origin. This is because at the origin all data are inliers so TPR = 0. Then, as we find outliers the TPR increases until we run out of outliers. In some sense, we only care about the early part of the ROC curve. After detecting all the outliers, one of the anomaly detectors can be declared the winner. Again, the AUROC value provides this information in one metric.

Exercise: Plot the ROC of Detectors 1 and 2 using the information in Table 11.3. Of the three detectors, which is the best detector?

Ans.: Detector 3 is ruled as being the worst by default. Then, as shown in Figure 11.18, detector 1 is better than detector 2 based on the AUROC. The results of manual area computations to obtain the AUROCs can be compared to the results of a previous exercise (which is 0.8125 for Detector 1 and 0.75 for Detector 2). Both methods produce the same values. Note that Detector 2 finds the two outliers ahead of Detector 1. □

The precision–recall curve can also be generated in a similar manner, as shown in Figure 11.19. This plot is not usually monotonic in its behavior and not as informative as the ROC curve. However, it does tell us where the largest value of precision occurs and what value of recall is associated with it. Since we seek high values of precision, we will tend to operate at the peak level of this curve.

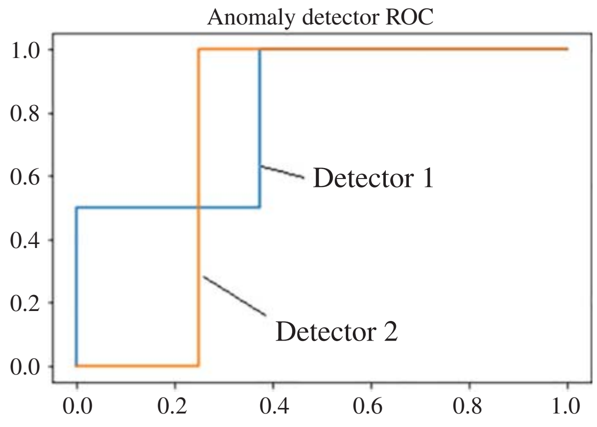


Figure 11.18 ROC curve for two anomaly detectors of Table 11.3.

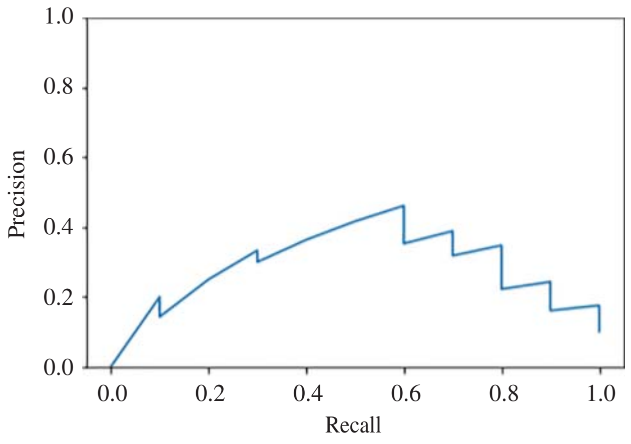


Figure 11.19 Precision–recall curve for a candidate anomaly detector.

11.6 Summary

In this chapter, the subject of anomaly detection was described in detail, focusing primarily on k-NN, DBSCAN, and Isolation Forest. Anomalies are an important subset of outliers. They are extreme outliers that have unusual characteristics compared to the rest of the data and oftentimes are symptomatic of suspicious activity such as spam or bank fraud. The three main methods for anomaly detection were compared and contrasted. The k-NN approach works well but is very expensive on large datasets. DBSCAN is a cluster-based method which can handle much larger datasets but requires careful hyperparameter tuning. Isolation Forest is among the fastest and most widely used for very large datasets. However, it also requires manual tuning and can generate more false positives and false negatives than the other methods due to the heuristic approach it takes for outlier detection. A new technique called MADmax was introduced which uses robust statistics and k-medians clustering to detect anomalies. Its speed is comparable to DBSCAN but has no requirement to tune any hyperparameter to obtain acceptable results. Finally, the techniques and metrics used to evaluate anomaly detectors were described, including precision, recall, and the AUROC metrics.

Problems

11.1 (Project) In this project, you will run DBSCAN using synthetically created data with anomalies inserted manually. Run the following Python code:

a) Load in the needed libraries.

```
# Standard imports
%matplotlib inline
from matplotlib.colors import ListedColormap
from itertools import groupby
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.ensemble import IsolationForest
from scipy.stats import norm, multivariate_normal
```

b) Generate the data with three clusters using a Gaussian mixture model.

```
# Generate data using Gaussian mixture model
dataset = 1
m1 = [-4,3]
m2 = [7,7]
m3 = [1,7]
cov1 = [[1.5, 1], [1, 1.5]]
cov2 = [[2, -1], [-1, 2]]
cov3 = [[0.8,0.5],[0.5,0.8]]
np.random.seed(np.random.randint(100))
x = np.random.multivariate_normal(m1, \
    cov1, size=(150,))
y = np.random.multivariate_normal(m2, \
    cov2, size=(300,))
z = np.random.multivariate_normal(m3, \
    cov3, size=(120,))
trainx = np.concatenate((x, y,z), axis=0)
print('The dimensions of the trainx are: ',trainx.shape)
```

c) Plot the raw data before anomaly detection.

```
plt.figure(figsize=(4,4))
plt.scatter(trainx[:,0], trainx[:,1], marker='.')
plt.axis('equal')
plt.xlabel('X-Axis', fontsize=16)
plt.ylabel('Y-Axis', fontsize=16)
plt.show()
```

d) Insert four anomalies into the data and plot the results.

```
#Insert anomalies to be detected
trainx[0] = [20,-1]
trainx[1] = [21,-1]
trainx[2] = [-5,15]
trainx[3] = [15,15]
plt.figure(figsize=(4,4))
plt.scatter(trainx[:,0], trainx[:,1], marker='.')
plt.axis('equal')
plt.xlabel('X-Axis', fontsize=16)
plt.ylabel('Y-Axis', fontsize=16)
plt.show()
```

e) Run DBSCAN and adjust the radius (`eps`) and the minimum number of samples (`min_samples`) until the results are correct.

```
# DBSCAN code starts here
print("Running DBSCAN on dataset", dataset)
eps, MinPts = 4.5, 10
db = DBSCAN(eps=eps, min_samples=MinPts).fit(trainx)
outlierDB = db.labels_
cats = pd.Series(outlierDB).value_counts()
print("Number of anomalies found:",cats[-1])
if(cats[-1] >4):
    print("Adjust eps and MinPts to improve results")
```

f) Plot the results to determine if the four anomalies were found. If not, adjust (`eps`) and (`min_samples`) as needed.

```
# Plot DBSCAN results
DBrows_with_anomalies = \
    np.where(db.labels_ == -1)[0]
xdbplot = []
ydbplot = []
for i in DBrows_with_anomalies:
    xdbplot.append(trainx[i,0])
    ydbplot.append(trainx[i,1])
plt.figure(figsize=(4,4))
plt.scatter(trainx[:,0], trainx[:,1], \
    marker='.',color='lightgray')
plt.scatter(xdbplot,ydbplot,marker='.', \
    color='black')
print("DBSCAN found", cats[-1],"anomalies")
plt.title("DBSCAN Outliers Detected")
plt.show()
```

11.2 (Project) In this project, you will run Isolation Forest using the same synthetically created data with anomalies inserted manually as in the first project.

a) Run Isolation Forest by setting the `contamination` parameter.

```
# Isolation Forest code starts here
print("Running IF on dataset ",dataset)
threshold = 0.1
IF = IsolationForest(contamination=threshold)
```

```
IF.fit(trainx)
anomaly_score = IF.decision_function(trainx)
outlierIF = IF.predict(trainx)
rows_to_delete = np.where(outlierIF <0)
count = 0
IFrows_with_anomalies = []
for i in range(len(outlierIF)):
    if outlierIF[i] == -1:
        count += 1
        IFrows_with_anomalies.append(i)
print("Number of anomalies found:",count)
if(count > 6):
    print("Adjust threshold to improve results")
```

b) Plot the results and adjust the `contamination` parameter, if needed.

```
# Plot IF results
xifplot = []
yifplot = []
for i in IFrows_with_anomalies:
    xifplot.append(trainx[i,0])
    yifplot.append(trainx[i,1])
plt.figure(figsize=(4,4))
plt.scatter(trainx[:,0], trainx[:,1], marker='.', \
    color='lightgray')
plt.scatter(xifplot,yifplot,marker='.', \
    color='black')
plt.title("Isolation Forest Outliers Detected")
plt.show()
```

11.3 (Advanced Project) In this project, you will run a prototype version of MADmax using the same synthetically created data with anomalies inserted manually as in the previous projects. The code uses only *k*-medians and *k*-NN algorithms to find outliers. The *k*-NM improvements are not included in this version.

a) To begin this project, you will need the *k*-medians code from Chapter 1 Problem 1.5(b). Enter that code into your notebook.

b) Next, run the following code to generate mini-clusters.

```
# Create mini-clusters for MADmax
m = 30
n = trainx.shape[0]
K = int(n/m)
print("Number of mini-clusters:",K)
cluster, cost, medians = KMedians(trainx,K)
```

c) Next, compute the value of `MADmax` from the mini-clusters.

```
# Find the value of MADmax
def MAD(x):
    return(np.median(np.abs(x-np.median(x))))
MADset = np.zeros(K)
for i in range(K):
    mini_cluster = trainx[cluster == i]
    distL2 = np.sqrt(((mini_cluster - \
        medians[i,:])**2).sum(axis=1))
    mini_cluster_MAD = MAD(distL2)
    MADset[i] = mini_cluster_MAD
MADmax = np.max(MADset)
print('The value of MADmax is ',format (MADmax, '.3f'))
```

d) Finally, enter the MADmax code to find and plot the outliers.

```
#Prototype MADmax code: kMedians+k-NN (without k-NM)
print("Running MADmax on dataset ", dataset)
plt.figure(figsize=(4,4))
plt.scatter(trainx[:,0], trainx[:,1], marker='.', \
    color='lightgray')
min_size = 5  #Deal with small mini-clusters
count_outliers = 0

kNN = 25        #internal parameter
Cfactor =  6.8  #internal parameter
Afactor = 11.0   #internal parameter

for i in range(K):
    mini_cluster = trainx[cluster == i]
    distL2 = np.sqrt(((mini_cluster - \
        medians[i,:])**2).sum(axis=1))
    mini_cluster_size = mini_cluster.shape[0]
    for j in range(mini_cluster_size):
        if (distL2[j] > Cfactor*MADmax \
            or mini_cluster_size < min_size ):
            out_distL1 = (np.abs(trainx - \
                mini_cluster[j,:])).sum(axis=1)
            sort_dist = np.sort(out_distL1)
            if (np.mean(sort_dist[1:kNN]) > \
                Afactor*MADmax):
                count_outliers += 1
                plt.scatter(mini_cluster[j,0], \
                    mini_cluster[j,1], marker='.', \
                        color='black')
print("Total outliers:",count_outliers)
plt.title('MADmax Anomalies Detected', fontsize=15)
plt.show()
```

e) Use the following new data to rerun all three anomaly detection methods: DBSCAN, IF, and MADmax. Do not adjust the parameter values. Compare the results.

```
dataset = 2
m1 = [0,3]
m2 = [7,7]
m3 = [3,7]
cov1 = [[6, 2], [2, 6]]
cov2 = [[5, -1], [-1, 5]]
cov3 = [[7,1.0],[1.0,7]]
np.random.seed(np.random.randint(100))
x = np.random.multivariate_normal(m1, \
    cov1, size=(150,))
y = np.random.multivariate_normal(m2, \
    cov2, size=(300,))
z = np.random.multivariate_normal(m3, \
    cov3, size=(120,))
trainx = np.concatenate((x, y,z), axis=0)
trainx[0] = [20,-1]
trainx[1] = [21,-1]
trainx[2] = [-5,15]
trainx[3] = [15,15]
print("Re-run DBSCAN, IF and MADmax")
```

f) Use the following new data to rerun all three anomaly detection methods: DBSCAN, IF, and MADmax. Do not adjust the parameter values. Compare the results.

```
dataset = 3
m1 = [3,5]
m2 = [10,10]
m3 = [15,15]
cov1 = [[6, 3], [3, 3]]
cov2 = [[6, 3], [3, 3]]
cov3 = [[6, 3], [3, 3]]
np.random.seed(np.random.randint(100))
x = np.random.multivariate_normal(m1, \
    cov1, size=(1200,))
y = np.random.multivariate_normal(m2, \
    cov2, size=(1500,))
    z = np.random.multivariate_normal(m3, \
        cov3, size=(1200,))
    trainx = np.concatenate((x, y,z), axis=0)
    trainx[0] = [20,-1]
    trainx[1] = [21,-1]
    trainx[2] = [-5,15]
    trainx[3] = [15,15]
    print("Re-run DBSCAN, IF and MADmax")
```

Reference

Aggarwal, C.C. (2017). *Outlier Analysis*, 2e. Springer.

12

Case Studies in Data Science

12.1 Introduction

In this chapter, we consider robust data science using the algorithms and procedures described in the previous chapters. The focus here is to compare the log-cosh (LC) methods with the traditional log-likelihood methods, namely, least squares (LS) and cross-entropy (CE), using two well-known datasets. In particular, we compare the competing approaches on linear regression, logistic regression, and neural networks. Examples include the Boston Housing and Titanic datasets. Data science problems of model building, prediction, and feature importance are addressed in the context of datasets with outliers. In each case, an outlier-centric approach is taken to demonstrate the enormous advantages of conducting data science in this manner. In addition, future directions for work in this area are described, namely, time series analysis of climate change and explainable artificial intelligence (XAI).

12.2 Example: Boston Housing Dataset

The Boston Housing dataset was used throughout this book to evaluate and compare different forms of machine learning. The dataset itself has 506 observations with 13 explanatory variables that factor into the median home prices in the Boston area, circa 1970–1975. The response variable and the 13 explanatory variables were all defined in Chapter 6. It is of particular interest to us in this book because it is also known to have many outliers. Housing prices vary considerably in an area or region and therefore the median price is often quoted, rather than the average price or a specific price, to account for outliers. For the moment, assume that the actual number of outliers is unknown but their existence is known. The question is, how do we handle a dataset when we know it may have potentially harmful outliers?

Three data science tasks will be carried out using this dataset. The first is to perform exploratory data analysis (EDA) to understand the dataset in some detail. The second is to build an accurate, compact model for prediction purposes, as was the case for most of the problems in this book. With the model, it will be possible to predict the median price of any home in the region provided that the needed information in terms of the explanatory variables is given. The model will be generated using linear regression

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.

Companion Website: www.wiley.com/go/HandlingOutliersandAnomaliesinDataScience1e

with a neural network. The third task addresses the problem of variable importance and correlations between them. With these data science objectives in mind, we follow a sequence of steps typically used in machine learning and data science, albeit in a streamlined fashion.

12.2.1 Exploratory Data Analysis

When we first obtain a dataset, we do not have any idea about its contents and characteristics so we need to perform EDA. Typically, the first step is to determine whether there are any missing values in any of the columns of the dataframe.[1] The Boston Housing dataset has missing values but here we will assume that some method of imputation is used to resolve this issue.

With a complete dataframe now in our possession, the next task is to examine the characteristics of the dataset. Some of these steps were shown in Chapter 4. Notably, we want to determine if there are outliers in the dataset and, if so, how many and where they are located. For example, the boxplots for the explanatory and response variables were presented in Chapter 4 to address these questions. The boxplots indicate the presence of outliers; so we will have to be extra careful about any operations performed that assume outlier-free data. Some preliminary analysis involves extraction of descriptive statistics such as the min, max, mean, median, quartiles, and standard deviation. These values are inspected to determine if standardization is required to enhance the performance and accuracy of downstream machine learning tools. The numerical ranges of the features were provided graphically in Chapter 4 which tells us that robust standardization will be needed.

Before going further, one useful step is to get a sense of the number of outliers for each variable in the dataframe. Another step is to examine the correlation coefficient matrix. The idea is to understand certain relationships between variables. Table 12.1 lists the variable names, the projected number of outliers using the 1.5-IQR edit rule (see Chapter 4), and the correlation coefficients relative to the response variable, `MEDV`. The number of outliers[2] indicates that we will likely have to resort to outlier removal. As a result, we cannot fully trust any of these correlation values. However, we find that variables `RM` and `LSTAT` are likely to have strong correlations to `MEDV`. Part of the data science problem here is to figure out how to accurately compute the number of outliers and correlations. We will explore the options in the sections to follow.

Next, we examine `MEDV` in the form of a boxplot and histogram in Figure 12.1. When the data is represented using this type of visualization, it is possible to observe both the frequency distribution and summary statistics of `MEDV`, making it easier to identify patterns, trends, anomalies, and unusual characteristics in the data. The boxplot identifies a total of 40 outliers in the response variable. The histogram indicates the presence of a number of homes priced at exactly \$50K which appears suspicious given the Gaussian-type distribution of the rest of the data. The reason for this is not clear at this stage but it could be due to censored data, or perhaps the values of these homes were high but unknown and they were simply assigned to \$50K as a default (i.e. winsorized data as described in Chapter 4). This will create

1 A dataframe is the standard format of a dataset, which may include columns of text, categorical values, real values, and the response variable. The data frame must eventually be converted into a design matrix, $\boldsymbol{X}$, and a response vector, $\boldsymbol{y}$, for use by a machine learning tool.
2 The number of outliers obtained this way cannot be trusted because it involves only individual variables and not regression residuals. Also, `CHAS` is a binary categorical variable so the number of outliers is not meaningful and therefore N/A is used in the table.

Table 12.1 Outlier counts for the Boston Housing variables using 1.5-IQR criteria, percentage of the total data, and correlation coefficients with respect to MEDV.

Feature set	Number of outliers	Percent (%)	Correlation coefficient (MEDV)
CRIM	65	12.9%	−0.39
ZN	68	13.4%	0.36
INDUS	0	0.0%	−0.48
CHAS	N/A	N/A	0.18
NOX	0	0.0%	−0.43
RM	30	5.9%	0.70
AGE	0	0.0%	−0.38
DIS	5	1.0%	0.25
RAD	0	0.0%	−0.38
TAX	0	0.0%	−0.47
PTRATIO	15	3.0%	−0.51
B	77	15.2%	0.33
LSTAT	7	1.4%	−0.74
MEDV	40	7.9%	1.00

headaches for the least squares (LS) method. We now have mounting evidence that we should resort to robust approaches.

One more check can be performed to determine the nature of the apparent outliers for two important variables. The RM variable has a strong positive correlation with MEDV (see Table 12.1) and a scatter plot confirms this, as shown in Figure 12.2. The LSTAT variable has a strong negative correlation with MEDV and a scatter plot in Figure 12.2 confirms this as well. We can observe the $50K homes along the top right and top left portions of the two plots, respectively. More outliers seem to be present for RM compared to LSTAT, but it is not clear if they are outliers or inliers at this stage. Any removal of the inliers would be a mistake so if we were to consider eliminating all $50K homes from the dataset, we may lose valuable information. Again, the idea of removing outliers before any regression is applied can be dangerous as it may lead to lower-quality models and higher prediction errors.

12.2.2 Neural Network Architecture

After the preliminary analysis of the data and potential outliers, it is time to decide how to proceed in building a model. For comparison purposes, LC can be used for the robust case, while for the non-robust case, LS can be used. It is possible to use standard linear regression methods but this is a particularly challenging dataset. The option of polynomial expansion of the variables exists to increase the feature space. A more powerful method to tackle this problem is to use a neural network (NN) for linear regression, as shown in Figure 12.3. It has 13 inputs, 1 output and one hidden layer comprised of 39 relu units.

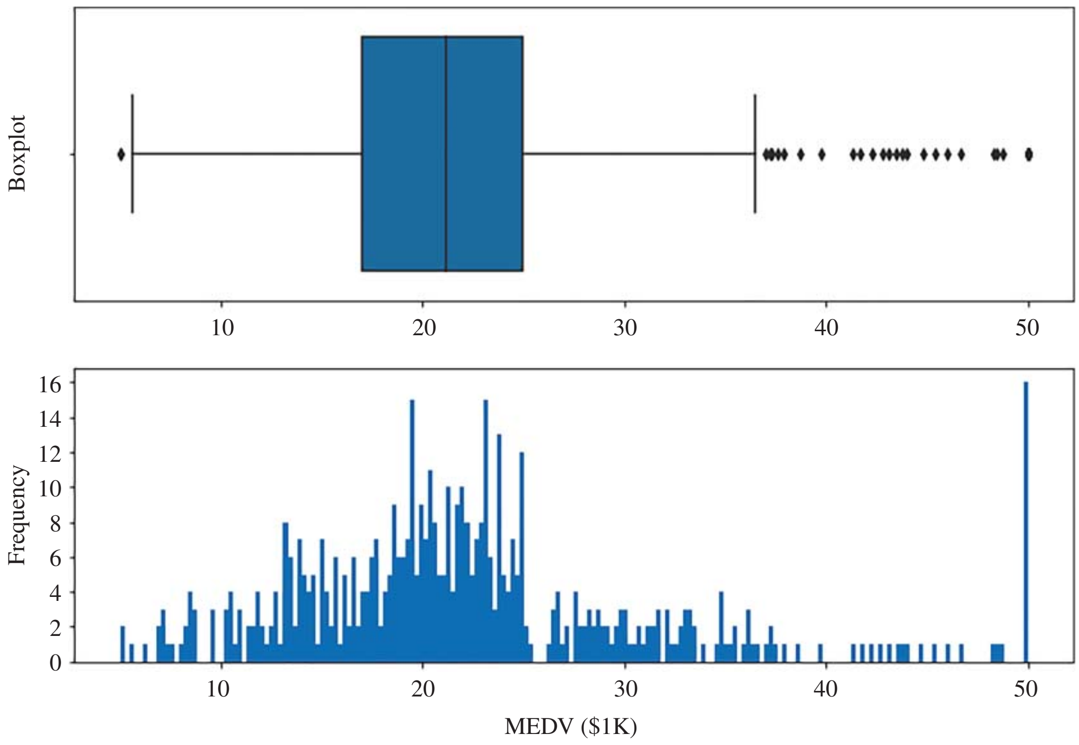


Figure 12.1 Relationship between boxplot and histogram of `MEDV` indicating suspected outliers and a large spike at $50K.

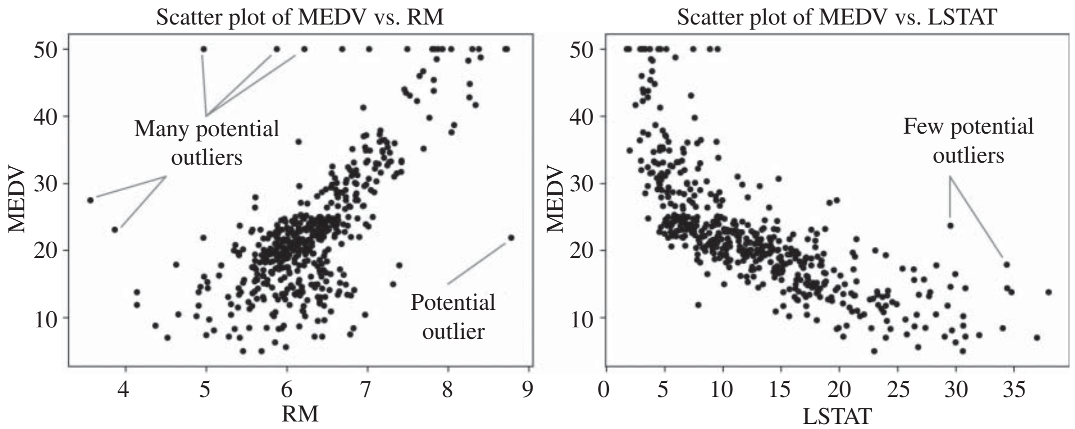


Figure 12.2 Scatter plot of MEDV vs. RM and LSTAT indicating suspected outliers.

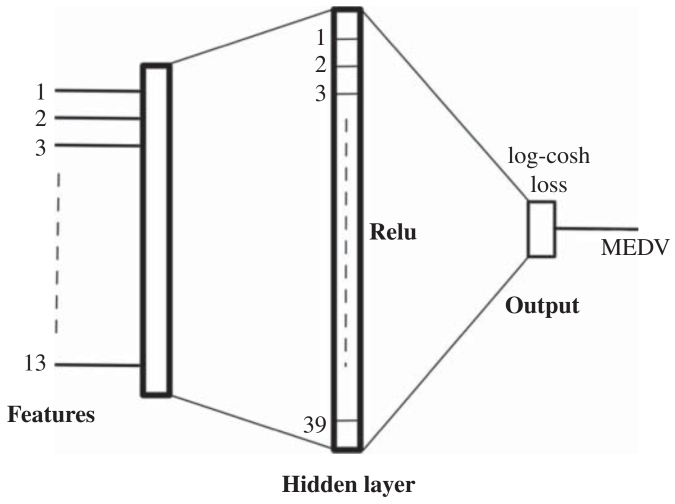


Figure 12.3 Neural network used for Boston Housing dataset.

As described in Chapter 9, this neural network creates $13 \times 39 + 39 = 546$ parameters which increases the parameter space dramatically. This will allow for a better fitting of the model to the data, although there is a high risk of overfitting. As a result, we need to establish a methodology to produce a regularized model to avoid overfitting.

Regularization was discussed in detail in Chapter 6. For example, reducing model complexity and the use of a ridge penalty function are effective methods to regularize models. Regularization can also be performed by controlling the number of iterations and this is how we will proceed here. Rather than letting the network converge to a set of final parameter values, an early-stopping criteria will be established so that a regularized model is produced. To accomplish this, we first divide the dataset into a training set and a test set using an 80–20 split. The resulting training set has 404 observations and the test set contains the remaining 102 observations. Then, we place the test set to one side and work with only the training set to build the model. To generalize the model, a 10-fold cross-validation (CV) scheme (as described in Chapter 6) can be used to find the optimal number of iterations with the smallest error.

The internal workings of a k-fold CV scheme are somewhat opaque to the user. An alternative, which is more transparent, is to use a related iterative method whereby the training set is randomly divided into CV training and test sets several times. In each pass, the two sets are used as if they were one fold of a k-fold CV scheme. This so-called *k-randomized validations* scheme is repeated a number of times, say 10 times, to get a similar effect as a 10-fold CV scheme. Accordingly, each CV training set can be processed over a large number of iterations to find the iteration at which the prediction error on the CV test set is minimized. Then, the optimal number of iterations across the 10-folds can be used to build a regularized model.

The results of this procedure are shown in Figure 12.4. The motivation behind this idea is based on the bias–variance trade-off. If we iterate too long, we end up overfitting. If we do not iterate enough, we are underfitting. The sweet spot between the two provides the optimal amount of fitting. The measurement criteria of the prediction error on each generated test set are the root-mean-square error (RMSE) and

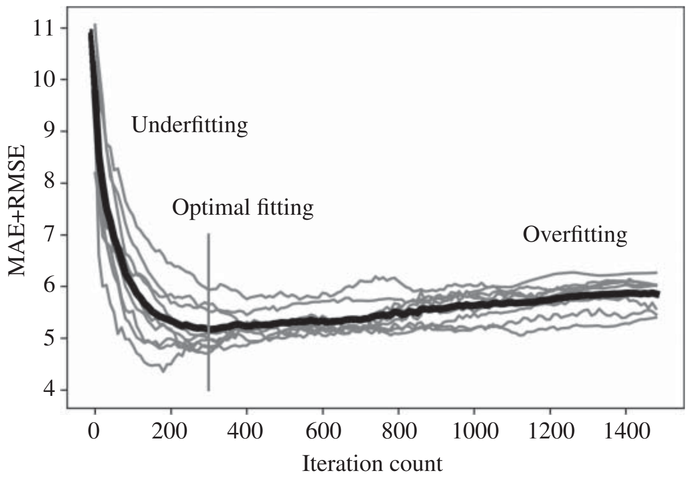


Figure 12.4 Using RMSE and MAE to find iteration count for regularization.

median-absolute error (MAE), as described in Chapter 4. The RMSE criteria is effective when outliers are **not** present in the test set whereas MAE is more suitable for test sets with outliers. When splitting the data into random CV training and test sets, there may be an unknown number of outliers in the test set. Therefore, it is better to use the sum of the RMSE and MAE to find the optimal number of iterations.

In Figure 12.4, each gray trajectory represents a different combination of CV training and test sets, as described above. The dark trajectory is the average of the 10 trajectories and it indicates that an iteration count in the neighborhood of 300 is suitable for this dataset. This value is now fixed for all further comparisons using the neural network.

12.2.3 Comparison of LSNN and LCNN

The next step is to use the above methodology to compare least squares NN (LSNN) and log-cosh NN (LCNN). For the purposes of the comparison, we consider four different cases involving outlier inclusion and removal. First, we remove all outliers that were found during EDA (see Table 12.1) and run a comparison between the robust and non-robust regressions. Second, we remove only the $50K data that look suspicious (see Figure 12.1) and rerun the regressions. These two cases will allow us to gain some insight into the inherent problems of the removal of outliers detected during EDA using only boxplots, histograms and quantiles of explanatory and response variables. Third, we keep all the outliers in the dataset and re-run the regressions. Finally, we remove all outliers using the iterative boxplot method (see Chapter 4), which is a very effective method, and rerun the regressions.

A comparison of the four cases, with the number of outliers removed in parentheses, is shown in Table 12.2. The comparison metrics are the RMSE and the MAE. There are several things to note in Table 12.2. First, the RMSE and MAE metrics are higher for LSNN compared to LCNN in all cases on the training set. However, the true measure of the quality of a model is the prediction accuracy on the test set. We see that the first column has the highest errors of the four cases. This is the problem with using EDA as the guide to outlier detection. It removes some inliers along with some outliers, but not necessarily all outliers.

Table 12.2 RMSE and MAE for the Boston Housing Dataset.

Removal Method (outliers)	**Remove All Potential Outliers (158)**	**Remove Only $50K Data (16)**	**Keep All Outliers (0)**	**Remove All True Outliers (52)**
LSNN				
CV train RMSE	2.66	2.13	2.42	1.72
CV train MAE	1.27	1.22	1.37	1.02
CV test RMSE	1.98	1.46	2.42	→1.29
CV test MAE	2.08	1.50	1.56	1.44
LCNN				
CV train RMSE	1.95	1.72	2.04	1.38
CV train MAE	0.40	0.51	0.53	0.38
CV test RMSE	1.66	1.46	2.78	→1.29
CV test MAE	1.94	1.44	1.28←	1.26←

The second column produces a noticeable improvement for LSNN just by removing the $50K data. In the case of LCNN, the training error is higher using MAE, but the test scores are lower. In the third column, we find that LCNN is much better than LSNN since LCNN is robust and no outliers have been removed. However, in the fourth column, only the true outliers (52 in total) have been removed so the dataset is outlier-free. Note the left-pointing arrows indicate where LCNN is producing almost identical results for MAE regardless of whether outliers are present or not. Similarly, the right-pointing arrows in the fourth column show that the RMSE values are the same for LCNN and LSNN when true outliers are removed. All the values are roughly the same at about 1.3, which is considered to be about the lowest value obtainable for this dataset.

It is an interesting exercise to take the 52 outliers extracted by the iterative boxplot method to see where they actually land in the original dataset. This is shown in Figure 12.5 for `MEDV` against the same two variables mentioned earlier (`RM` and `LSTAT`). The gray regions are the inlier data and the dark dots are

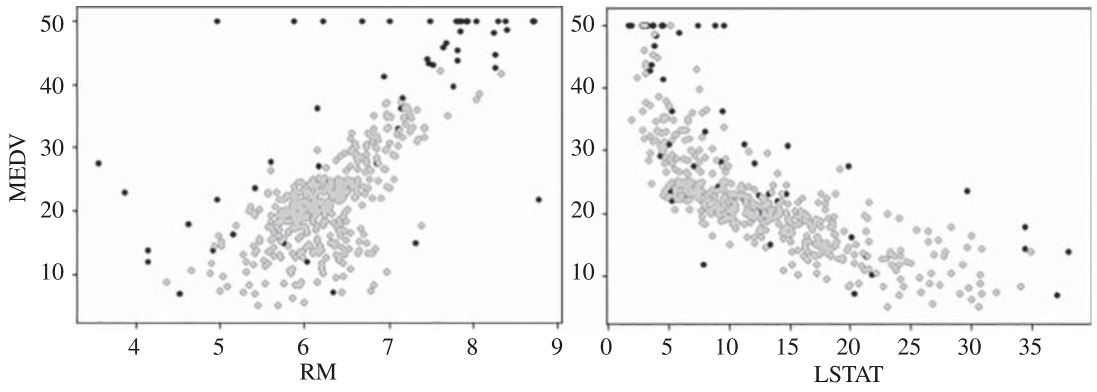


Figure 12.5 Actual outliers (black dots) based on residuals using the iterative boxplot method (see Chapter 4).

the outliers. Clearly, some of the true outliers would not be detectable prior to regression because they are only detectable in a 14-dimensional space, not a 2-dimensional space. This is the danger of trying to sift out the outliers during EDA. If we removed all 158 potential outliers, there would be 106 inliers that would be inadvertently removed. If we removed only the \$50K winsorized data, it would reduce the errors but we would still not catch all the outliers. The only viable options are to run robust regression without removing outliers, or use the iterative boxplot method to remove true outliers and use LSNN or LCNN. As noted earlier, outliers are having a noticeable effect on the overall accuracy of the LSNN model, which should not be used for this dataset unless outliers are removed.

12.2.4 Predicting Housing Prices

We now have a model based on the 546 parameters generated using the neural network. This model can be used to predict the price of any new house that is up for sale in the same region. For our purposes, we can predict the prices for the original dataset and compare the predictions against the original prices. The predictions are based on the model from the third column of Table 12.2. This is shown in Figure 12.6. Notice that the histogram is essentially the same but the suspicious \$50K homes are gone and the distribution is well-behaved. The error in the prediction, according to the MAE value in Table 12.2, is about \$1.3K. This is what we would expect of a very accurate model generated by a neural network using robust methods.

12.2.5 RMSE vs. MAE

The results presented in Table 12.2 were obtained for one specific split of the training set into its CV components. To solidify the understanding of how to interpret RMSE and MAE metrics when used with robust methods, we examine the effects of 10 different splits of the training set, one for the original data with outliers and one for outlier-free data (i.e. as done in the last two columns of Table 12.2). Before proceeding, we should remove the `INDUS` variable since it was found to be a nuisance variable

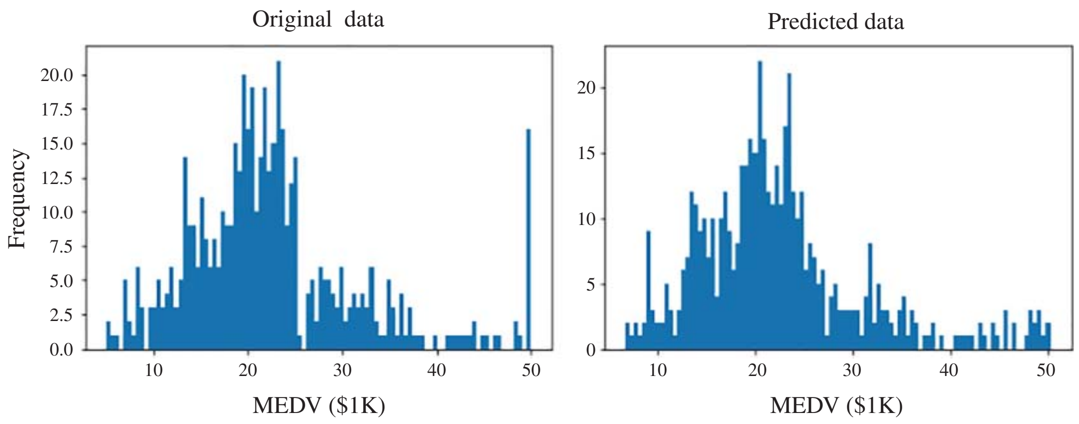


Figure 12.6 Original and predicted housing prices.

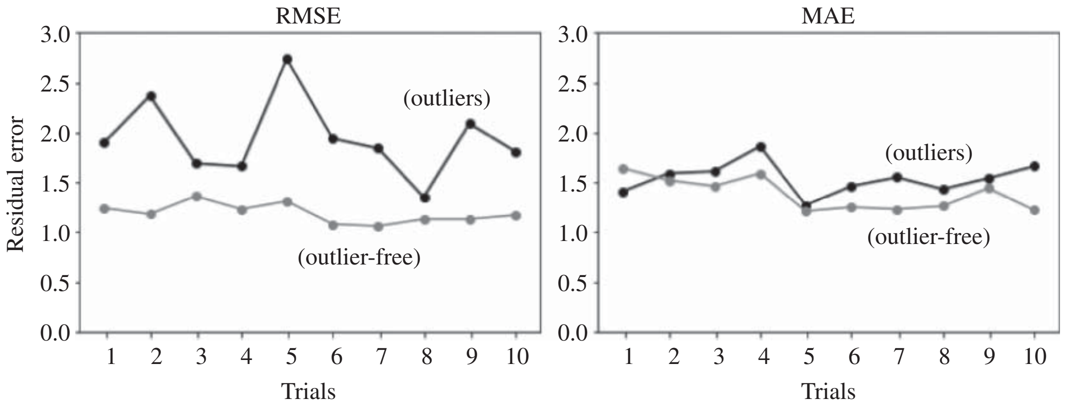


Figure 12.7 LCNN CV test set errors on 10 trials comparing RMSE and MAE on datasets with and without outliers.

in Chapter 6. Next, we run LCNN for 300 epochs using 12 inputs and 39 units in the hidden layer of the neural network.

The CV test results of the 10 trials on the two datasets, each using an 80–20 split, are given in Figure 12.7. On the left, we can clearly see that the RMSE is large in the dataset with outliers compared to the outlier-free data. This is why RMSE is not advisable for use on datasets with outliers. It causes an inflation of residuals which leads to large RMSE values. The outlier-free data produces a relatively constant low value across all 10 trials. On the right panel, we note that MAE is roughly the same whether or not outliers exist in the dataset. We know that the robust MAE metric will be roughly constant (at around 1.5 in this case) across many different configurations of the training and test sets if LCNN is used, regardless of the presence or absence of outliers. The recommendation is to observe both RMSE and MAE when using robust methods.

12.2.6 Correlation Coefficients

Finally, we return to the issue of computing the correlation coefficients. In statistics, a correlation coefficient is a numerical measure that quantifies the strength and direction of the linear relationship between two variables. The value of a correlation coefficient ranges from −1 (strong negative correlation) to 1 (strong positive correlation). We noted earlier in Table 12.1 that the correlation coefficients relative to the response variable `MEDV` could not be trusted due to the presence of outliers. But how do we obtain values that are trustworthy? We must use the outlier-free data. There is no choice here.

Figure 12.8 plots two sets of correlation coefficients for the explanatory variables relative to `MEDV`. One set is directly taken from Table 12.1. The other set is obtained in the same manner using outlier-free data. Note that most of the values change noticeably when the outliers are removed. Some are higher, while others are lower. For example, the original value of the correlation coefficient is 0.25 for the `DIS` variable. After removing the outliers, the value jumps to 0.50. This is a significant change in the value. For `AGE`, the value drops from −0.4 down to −0.6. If accurate values of the correlation coefficients are important for a particular data science application, then outliers must be removed before applying the necessary computations. Again, the iterative boxplot method (see Chapter 4) is the most reliable way to find and remove outliers.

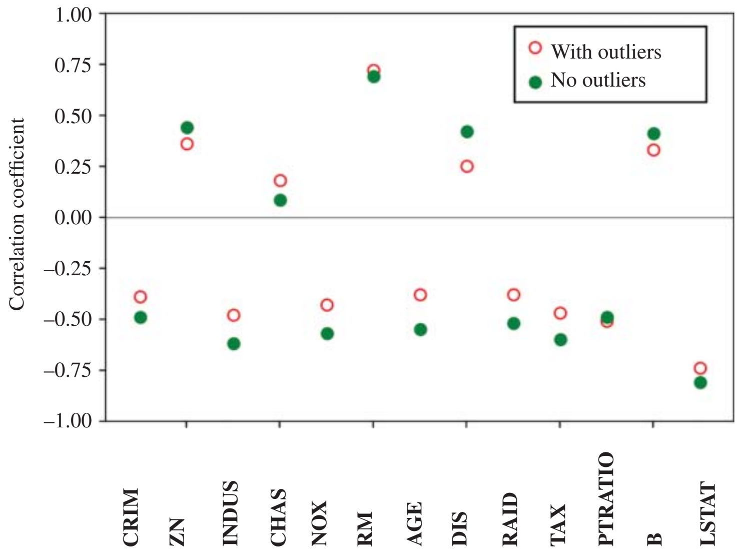


Figure 12.8 Correlation coefficients (with respect to `MEDV`) with and without outliers.

12.3 Example: Titanic Dataset

The Titanic dataset is well-known in data science circles as a standard binary classification problem. It contains information about the passengers who survived and those who did not on the ill-fated maiden voyage of the Titanic in April 1912, which struck an iceberg 400 miles off the coast of Canada. Our objective is to use a training set to analyze a random subset of Titanic passengers and their survival rates. By applying logistic regression, we can build a model and use it to predict which passengers would survive based on a given test set. This approach seeks to determine whether survival was random or if there were underlying patterns that could be captured in a predictive model.

We first illustrate basic elements of data science and EDA. Then, the dataset is processed to create the needed $\boldsymbol{X}$ and $\boldsymbol{y}$ values for both training and testing purposes. There are 891 rows in the training set and 418 rows in the test set.[3] Next, two approaches to binary classification are compared and contrasted, namely, CELR and LCLR, which were covered in Chapter 8. The Titanic dataset is expected to have many outliers by its very nature, leading us to believe that robust methods would be better suited for this application. Another goal of this section is to demonstrate that robust methods offer a number of other advantages, in addition to better prediction accuracy.

12.3.1 Exploratory Data Analysis

When a dataset is first encountered, there are a number of procedures that are usually followed to obtain a deeper understanding of the problem and gain some domain knowledge. It is rarely the case that you can run a machine learning tool directly on the dataset without any modifications. Many of the steps to follow

3 The Titanic dataset is available on the Kaggle website at www.kaggle.com in the Competitions section.

have been covered in different parts of this book. In this section, we carry out rudimentary elements of EDA to provide a more coherent picture.

To begin the analysis, we examine the first 10 rows of the Titanic dataset in Table 12.3. The original dataset contains much more information but this reduced form is sufficient for our purposes here. The first column is the passenger Id for the dataset and is only used for bookkeeping. The second column is the binary response indicating whether the passenger survived (1) or not (0). This tells us immediately that it is a binary classification problem which will require the use of the methods of Chapter 8. This column will be used to form the $\boldsymbol{y}$-vector. A subset of the remaining columns will be used to derive the $\boldsymbol{X}$ matrix, with one row for each passenger.

The definitions associated with other columns are provided in Table 12.4. The dataset is a mixture of strings, categorical data, and real and integer numbers. This is usually the nature of most datasets. They are a mix of all types of data in different formats. In order to process the data in Python, it should be expressed in the form of a **dataframe**, which is a two-dimensional, tabular data structure commonly used for data manipulation and analysis. Libraries like `Pandas` provide powerful tools for working with dataframes. Our task at hand is to convert all of this data into numbers so that it is suitable for logistic regression.

We will only provide a brief glimpse of EDA because there are so many ways to represent and visualize a dataset.[4] We start by examining the total number of missing elements in each column, as given in Table 12.5, noting that there are 891 rows in the Titanic training set. The `Cabin` and `Age` columns each have a significant number of missing entries with a label of NaN or NA, whereas `Embarked` has only two entries missing. Each case would be handled differently. The relevance of the `Cabin` number seems questionable and many entries are missing so this column can be deleted. As an aside, the `Name` column seems irrelevant too. It is not clear how a person's name would affect their survival, and any family or sibling/spouse relationships are captured in the `Parch` and `SibSp` columns. Therefore, the `Name` column can also be deleted. The same applies to the `Ticket` column.

The missing entries in the `Age` column pose a real problem because there may be some correlation between `Age` and `Survived`. The missing values can be simply replaced with the mean of the ages. A better approach is to compute the mean and standard deviation of the data in Figure 12.9(a) and sample `Age` values from the computed Gaussian distribution. This approach was taken to impute the missing values.

The next step is to investigate each variable in more detail. Consider the two columns represented with real numbers: `Age` and `Fare`. These are suitable for visualization using histograms as seen in Figure 12.9. For the `Age` histogram, one can observe a higher frequency of young people with a decline as `Age` increases. The `Fare` histogram indicates that most fares are in the same range, but some outliers exist. Both are skewed right. We would typically investigate which categories of `Age` and `Fare` have the most survivors to understand if `Age` and/or `Fare` are important features to retain. We skip those details here and keep them both because they will turn out to be important. Further, we will scale `Age` by 20 and `Fare` by 50 to bring them in line with other values in the table.

Following this, we examine the categorical data columns: `Sex`, `Embarked`, and `Pclass`. These columns can be cross-tabulated with the `Survived` column to get a snapshot of the relevance of the data. Consider, for example, the tabulated data in Table 12.6. It indicates that more females survived than males by a significant margin. If we simply guessed that all female passengers survived and all

4 For a thorough analysis, we refer the reader to the Kaggle website and the detailed explanations therein.

Table 12.3 Sample of first 10 rows of Titanic training data.

Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	Male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	Female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	Female	26.0	0	1	STON/ O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	Female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	Male	35.0	0	0	373450	8.0500	NaN	S
6	0	3	Moran, Mr. James	Male	NaN	0	0	330877	8.4583	NaN	Q
7	0	1	McCarthy, Mr. Timothy J	Male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	Male	2.0	3	1	349909	21.0750	NaN	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	Female	27.0	0	2	347742	11.1333	NaN	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	Female	14.0	1	0	237736	30.0708	NaN	C

Table 12.4 Definitions of variables in the Titanic dataset.

Variables	Description	Data type
`Pclass`	Passenger class (1, 2, 3)	Categorical
`Name`	Full name of passenger	String
`Sex`	Male or female passenger (0, 1)	Categorical
`Age`	Age of passenger	Real
`Parch`	Number of parents with children	Integer
`SibSp`	Number of siblings/spouse of passenger	Integer
`Fare`	Actual fare paid by passenger	Real
`Cabin`	Cabin number of passenger	String
`Embarked`	Starting point for passenger (S, C, Q)	Categorical

Table 12.5 Number of actual data entries in each column.

Data columns	Total number of valid entries
`Survived`	891
`Name`	891
`Sex`	891
`Age`	714
`SibSp`	891
`Parch`	891
`Fare`	891
`Cabin`	204
`Embarked`	889

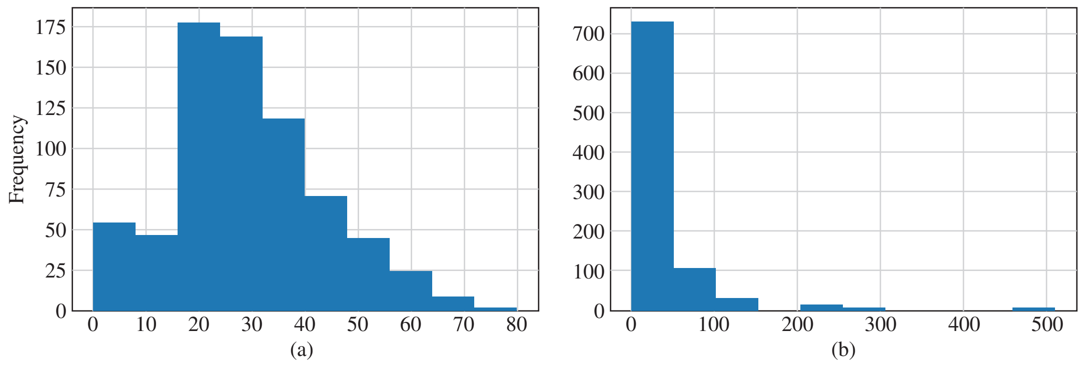


Figure 12.9 Histograms of passenger (a) `Age` and (b) `Fare`.

Table 12.6 Cross-tabulation of survivors based on `Sex`.

Sex	Survived	Did not survive	Totals
Female	233	81	314
Male	109	468	577

Table 12.7 Cross-tabulation using `Embarked` for the Titanic dataset.

Embarked	Survived	Did not survive	Totals
S	217	427	644
C	93	75	168
Q	30	47	77
Totals	342	549	889

males did not, we would have $233 + 468 = 701$ correct predictions out of 891. That is an accuracy of 78% without using machine learning. Clearly, the `Sex` of the passenger is a critical feature. Also, we should view 78% as a baseline accuracy for the evaluation of different classification methods set since it is easy to obtain this value using hand calculations.

We now turn our attention to the `Embarked` column. Passengers embarked from Southampton (S), Cherbourg (C), or Queenstown (Q). Table 12.7 lists the number of passengers embarking from each port. The number of survivors from each port seems to be roughly proportional to the total number who embarked from that port. This kind of assessment of the data is very useful because we noted earlier that two data entries are missing. The data scientist performing EDA must now decide how to fill those two missing entries. Since S and C are the most frequent, one entry could be set to S and the other to C. This will not change the results significantly so this simple solution is acceptable. The last step is to convert each category into a number. We should choose S=3, C=2, and Q=1 because it roughly correlates to the number of passengers embarking from each port, if for no other reason.

The `Pclass` column may be examined in the same way as `Embarked`, with some cross-tabulation work to assess its relative importance. We skip this step here. Since none of the data entries are missing, we choose to keep this column in case it has relevance. Finally, the `Parch` and `SibSp` columns are integer quantities indicating the number of Parent/Children and Sibling/Spouse totals associated with each passenger, so we choose to keep them both without modification. With EDA now completed for the Titanic dataset, the numerical data for the training and test sets can be generated.

After applying all the above steps, we produce $\boldsymbol{X}$ and $\boldsymbol{y}$ for the training data shown in the upper part of Table 12.8. Next, we apply the identical steps to the test set. A portion of the test data is shown in the lower part of Table 12.8. We must remove the same columns as in the training set, apply replacements for NaN or NA in the same manner, and scale or standardize the data in the same manner as the training set. Otherwise, the logistic regression model will be of no value during prediction.

Table 12.8 Sample of Titanic training and test data.

	Survived	Pclass	Sex	Age/20	SibSp	Parch	Fare/50	Embarked
Training set	1	1	0	1.90	1	0	1.425	2
	0	3	1	1.10	1	0	0.145	3
	0	3	1	1.10	1	0	0.145	3
	1	3	0	1.30	0	1	0.158	3
	1	1	0	1.75	1	0	1.062	3
	0	3	1	1.75	0	0	0.161	3
	0	3	1	1.45	0	0	0.169	1
	0	1	1	2.70	0	0	1.037	3
	0	3	1	0.10	3	1	0.421	3
	1	3	0	1.35	0	2	0.222	3
	1	2	0	0.70	1	0	0.601	2
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Test set	0	3	1	1.70	0	0	0.157	1
	1	3	0	2.35	1	0	0.140	2
	0	2	1	1.55	0	0	0.160	1
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

In the results to follow, we will not use k-fold cross-validation (see Chapter 6) but instead use the entire training set to generate the model parameters and then perform prediction using the test set. Regularization, if needed, will be performed using a penalty function. Recall, our goal here is to compare different methods using the same procedure. For this reason, we can skip the normal cross-validation scheme and simply illustrate the similarities and differences of the two approaches. Our overall objective is to provide evidence that the robust method can be used to produce superior results.

12.3.2 LCLR vs. CELR

In this section, we compare the results of CELR and LCLR (see Chapter 8 for details of the two methods). All the steps of the previous section are used to produce the $\boldsymbol{X}$ and $\boldsymbol{y}$ data. The task of machine learning is to generate model parameters for prediction. Model generation is done with the training set ($n = 891$, $d = 7$); prediction is performed using the test set ($n = 418, d = 7$). We can now use them to compare the two machine learning methods for binary classification.

In a multidimensional setting, the model defines a hyperplane that separates the survivors from the non-survivors. With the presence of so many outliers, the data is not easily separable with a linear or nonlinear boundary. Since the dataset represents a seven-dimensional problem, we are unable to plot or even visualize the hyperplane boundaries produced by CELR or LCLR. However, we can determine

Table 12.9 Number of correct predictions for Titanic training and test sets.

Dataset (Training = 891, Test = 418)	**CELR (C = 100)**	**LCLR (a = 10)**
train (# correct out of 891)	710	725
train (% correct)	79.6%	81.6%
Outliers detected	57	159
test (no. correct out of 418)	317	325
test (% correct)	75.8%	77.8%

Table 12.10 CELR (left) and LCLR (right) confusion matrices for Titanic dataset for training and test sets.

CELR (C = 100)			**LCLR (a = 10)**		
Training set	True-0	True-1	Training set	True-0	True-1
Predict-0	470	102	Predict-0	492	113
Predict-1	79	240	Predict-1	57	229
Test set	True-0	True-1	Test set	True-0	True-1
Predict-0	211	48	Predict-0	222	51
Predict-1	53	106	Predict-1	42	103

the number of correct responses for the training and test sets to compare the two methods against one another. This is shown in Table 12.9. The results indicate that LCLR outperforms CELR in both training and test sets. Of particular note is the 2% improvement in the test accuracy for LCLR over CELR. CELR used ridge regularization with $C = 100$, whereas LCLR was executed without any regularization. The selection of the robustness parameter ($a = 10$) for LCLR will be described in a later section.

A proper evaluation of the performance of a binary classifier involves the use of other metrics such as recall, precision, and F_1 score which account for false positives and false negatives in the results (see Chapter 8). The F_1 score is composed of the recall and precision so it will be used as an overall assessment of the two approaches. The confusion matrices for the two methods are shown in Table 12.10. For CELR, the scores for the training and testing sets are $F_1 = 0.726$ and $F_1 = 0.677$, respectively. For LCLR, the scores for the training and testing sets are $F_1 = 0.729$ and $F_1 = 0.689$, respectively. Therefore, LCLR is marginally better than CELR based on the F_1 metric.

Exercise: Compute the accuracy, recall, precision, and F_1-score for all four confusion matrices in Table 12.10. Which set of results are more important, those on the training set or those on the test set? Also, which method performs better overall?

Ans.: For CELR on the training set:

```
Accuracy = (tp+tn)/(tp+tn+fp+fn) = 710/891 = 0.797.
Recall = tp/(tp+fn) = 240/(240+102) = 0.701.
Precision = tp/(tp+fp)=240/(240+79) = 0.752.
Then F1 = 2(0.701)(0.752)/(0.701+0.752)  = 0.726.
```

For CELR on the test set:

```
Accuracy = (tp+tn)/(tp+tn+fp+fn) = 317/418 = 0.758.
Recall = tp/(tp+fn)=106/(106+48) = 0.688.
Precision = tp/(tp+fp)=106/(106+53) = 0.667.
Then F1 = 2(0.688)(0.667)/(0.688+0.667) =0.677.
```

For LCLR on the training set:

```
Accuracy = (tp+tn)/(tp+tn+fp+fn) = 721/891 = 0.801.
Recall = tp/(tp+fn)=229/(229+113) = 0.670.
Precision = tp/(tp+fp)=229/(229+57) = 0.801.
Then F1 = 2(0.670)(0.801)/(0.670+0.801)  = 0.729.
```

For LCLR on the test set:

```
Accuracy = (tp+tn)/(tp+tn+fp+fn) = 325/418 = 0.778.
Recall = tp/(tp+fn)=103/(103+51) = 0.669.
Precision = tp/(tp+fp)=103/(103+42) = 0.710.
Then F1 = 2(0.669)(0.710)/(0.669+0.710)  = 0.689.
```

The test set results are more important than the training set results because they are more representative of the model performance on unseen data. The LCLR results are superior to the CELR results due to outliers in the dataset. □

12.3.3 Outlier Detection and Removal

At this stage, it is natural to ask the question of how the two methods would perform if the outliers were removed. Returning to Table 12.9, note that it also contains the number of outliers detected. In Chapter 8, an outlier in the binary classification problem was defined as a data point whose label is the opposite of those in its immediate vicinity. The Titanic dataset contains many such outliers and LCLR is able to identify all potential outliers using a rather straightforward methodology. The basic idea is to compute the residuals of logistic regression and plot the associated histogram. The extreme values on either side are likely to be outliers. This was done for CELR and LCLR, as shown in the histograms of Figure 12.10. The histogram on the right for CELR shows that residuals are distributed across the range between −1 and +1, with a large peak around 0. LCLR also has a large peak around 0. However, many LCLR residuals are either located near +1 or −1. These are the potential outliers resulting from a choice of $a = 10$ in LCLR. For $a = 1$, the histogram for LCLR would look similar to CELR.

The number of LCLR and CELR outliers can be determined by setting a criteria of +0.85 and above for false negatives, and −0.85 and below for false positives. As listed in Table 12.9, CELR reports only

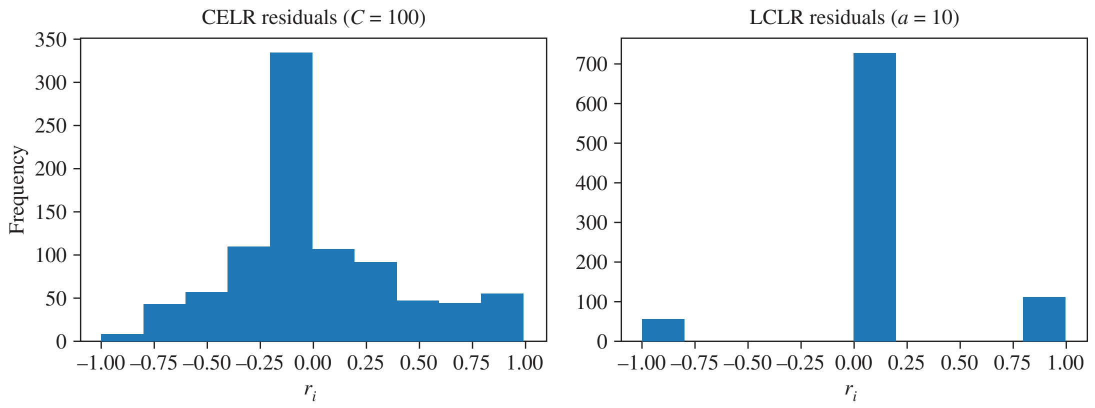


Figure 12.10 Histograms of residuals r_i associated with the CELR and LCLR cases for the Titanic dataset.

57 outliers out of 891 observations (6.3%). LCLR reports that 159 out of 891 observations (i.e. 17.8%) are potential outliers. Assuming LCLR is correct, this gives us an idea that the maximum achievable accuracy in the Titanic training set is about 82.2%. Since we know which data points are potential outliers, we are able to report and inspect each one. This may serve as an important tool for many data scientists attempting to clean their dataset by removing unwanted outliers in binary classification problems.

The next question to be asked is, how many of them are actual outliers and how many are errors made by the classifier due to the location of the decision boundary? To answer this question, the outliers detected by LCLR can be removed from the dataset and both CELR and LCLR can be executed on the clean dataset. The results on the outlier-free training set are shown in Table 12.11. What is remarkable, although completely expected, is that the two tables are almost the same. This is true for the training set as well as the test set. The results for LCLR did not change much (somewhat improved) over Table 12.10, whereas there is a great deal of improvement for CELR. The accuracy is now 99% and the F_1-score is 0.98, while the results on the test set remain relatively unchanged for LCLR. This is what the robust method is able to deliver: stable test results with or without outliers in the training set.

Table 12.11 CELR (left) and LCLR (right) confusion matrices for Titanic dataset for training (**outlier removed**) and test sets.

CELR ($C = 100$)			**LCLR ($a = 10$)**		
Training set	True-0	True-1	Training set	True-0	True-1
Predict-0	492	3	Predict-0	492	2
Predict-1	8	229	Predict-1	8	230
Test set	True-0	True-1	Test set	True-0	True-1
Predict-0	222	51	Predict-0	221	50
Predict-1	42	103	Predict-1	43	104

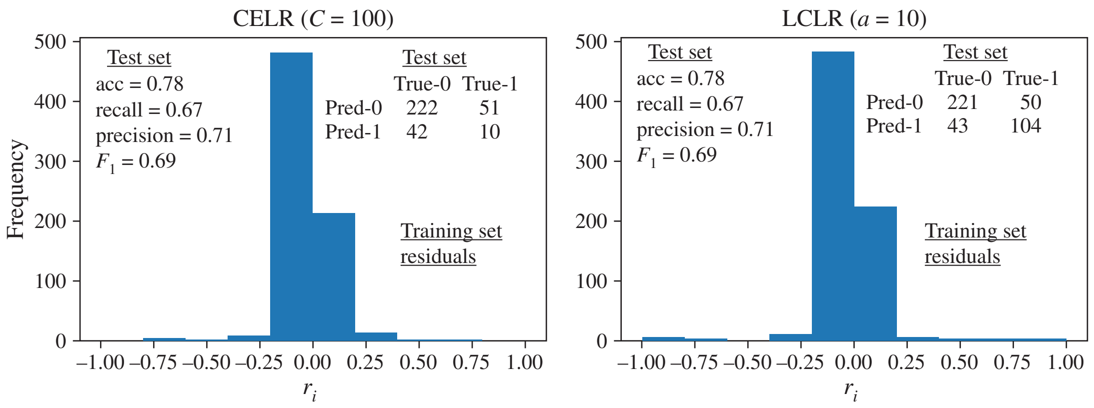


Figure 12.11 Histograms of training set residuals after outlier removal, and test set metrics for the Titanic dataset.

A more detailed investigation can now be carried out. There are a total of 153 true outliers. This implies that the classifier made only 10 actual errors (2 false negatives and 8 false positives). These are essentially classification errors that occur near the separation boundary. If we examine the new histogram of residuals as shown in Figure 12.11, the outliers are now gone, but there are a few lingering classification errors. After the outlier removal process, a new set of test metrics can be computed and they are provided in the figure. For LCLR, the metrics remain unchanged from the original training set with outliers. The CELR approach also produces the same results using the training set with outliers removed. This analysis gives us a lot of insight into what is actually going on in the classification problem and the ability for robust methods to find the proper decision boundary in the midst of both outliers and classification errors.

12.3.4 Robustness Coefficient for Log-Cosh

In this section, the goal is to select an appropriate value for the robustness coefficient, a, for LCLR. The loss function for logistic regression using the log-cosh function is given by

$$J^{\text{LCLR}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \frac{1}{a} \log(\cosh(a(y_i - p_i))) \tag{12.3.1}$$

where $p_i = 1/(1 + e^{-\mathbf{x}_i^\top \boldsymbol{\beta}})$ as described in Chapter 8.

To demonstrate how to select a suitable value of a, we use the Titanic dataset, but this value will be suitable for all other binary classification problems. Four values of a are considered for comparison purposes, specifically $a = 1, 2, 5$, and 10. Using these values, the histograms of the residuals are plotted in Figure 12.12 for the four cases along with all the metrics and confusion matrices described in the previous section. The histogram in the top-left corner is very similar to the one for CELR in Figure 12.10. In fact, the accuracy and F_1 scores are about the same as CELR. This is why $a = 1$ is not a suitable level of robustness. Also note that with $a = 1$ or 2, the outliers are not clearly visible based on the histograms. Some can be observed near +1 but only a few at −1. However, as a increases, the outliers begin to emerge

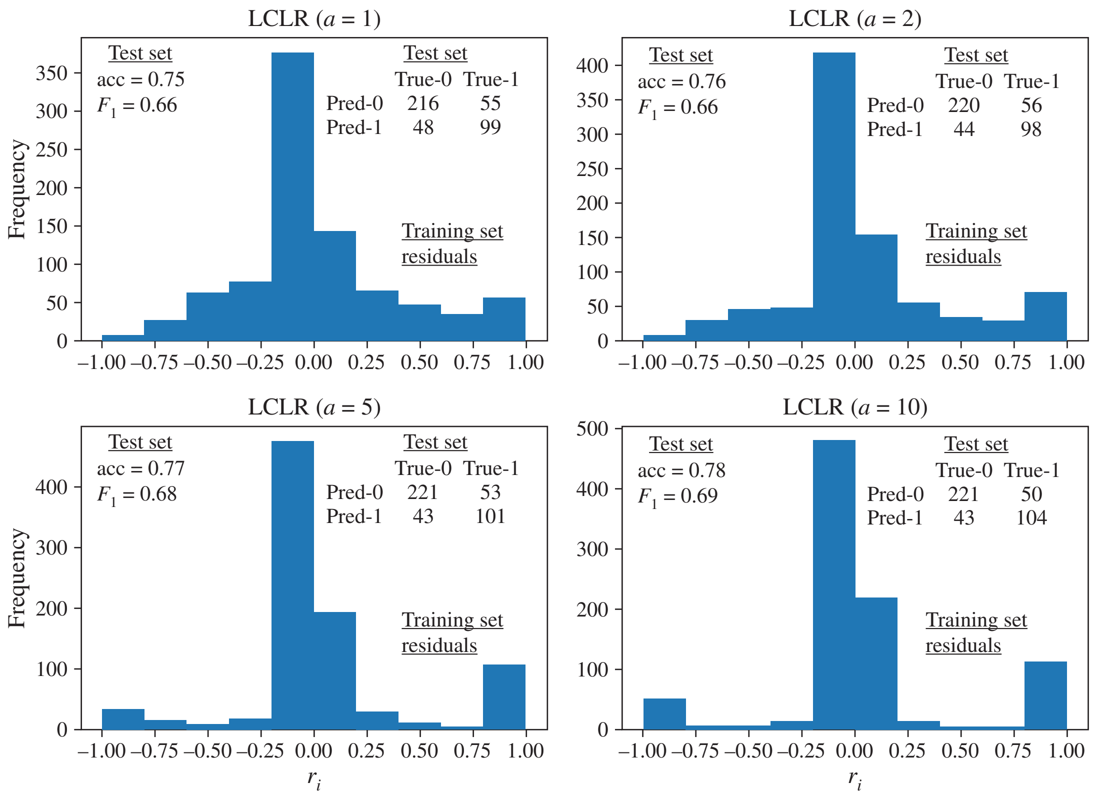


Figure 12.12 Test set metrics for the Titanic dataset for four selected robustness coefficients ($a = 1, 2, 5, 10$) for LCLR.

at $a = 5$ and are clearly visible at $a = 10$ around -1 and $+1$. In addition, all the metrics improve at $a = 10$. The inset provides the confusion matrices for the test set for each case, and $a = 10$ produces the best results. The outliers indicated by this case will limit the maximum accuracy possible so, optionally, they can be removed to produce outlier-free data. On the other hand, the case for $a = 2$ is more suitable if probabilities are desired as opposed to outlier detection and removal.

12.3.5 The Implications of Robustness

In this section, we explore the deeper implications of robustness. While it is true that the robust methods are outlier tolerant, they also provide stability to the results. To understand the nature of this stability, consider an underlying issue in logistic regression which involves the selection of a threshold, p_T, that determines whether a probability is discretized as 0 or 1. The range of possible values of p_T is between 0.0 and 1.0. The nominal value is 0.5. However, in practice, a different value can be selected if it produces better overall results. But how should such a threshold be selected? To answer this, we first review the metrics of Chapter 9.

Recall that the quality of a binary classification method is often evaluated using the area under the ROC curve (see Aggarwal 2017). To produce this curve, different values of p_T are selected between 0.0

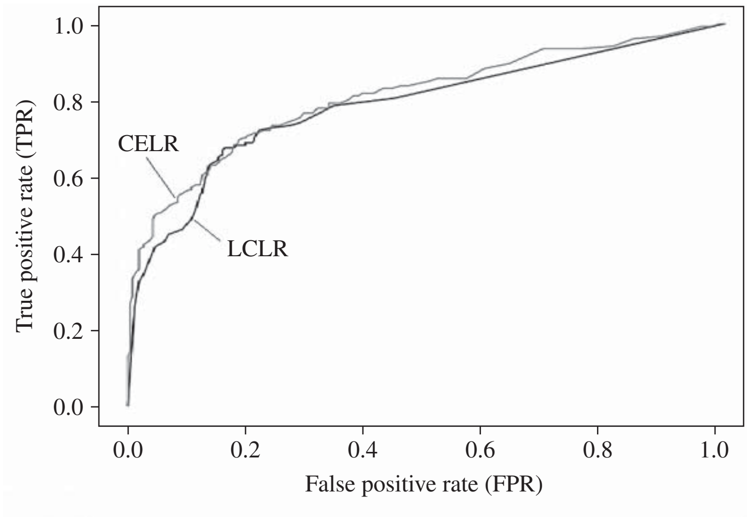


Figure 12.13 ROC curves for CELR and LCLR.

and 1.0. Then the true positive rate (TPR) is plotted against the false positive rate (FPR). The area under the resulting curve is the AUROC metric. An example of two ROC curves is given in Figure 12.13. Here, CELR and LCLR are compared in terms of their ROC. They are very similar and they cross over several times. The AUROC for CELR is larger than LCLR so one could conclude that CELR is better than LCLR in this context. However, this is not quite right.

The ROC curves can be misleading in cases such as this. We have already found that LCLR is superior to CELR with outliers present. These two curves are too close to make any definitive judgments. Furthermore, it does not suggest which value of p_T is best to use for a given test set. Ultimately, a particular value of p_T will be selected and the F_1 score will necessarily be a function of p_T. In fact, the best p_T is the one that has a point closest to the top-left corner of the plot, where TPR = 1 and FPR = 0. Instead of using AUROC, more insight can be gained by examining F_1 as a function of p_T.

The results of the F_1 score plotted as a function of p_T for the cross-entropy and log-cosh cases are shown in Figure 12.14. On the left side, we observe that CELR has one optimal value of p_T given by 0.5 in this case. On the right side, LCLR produces a flat characteristic implying that any value of p_T between 0.1 and 0.8 will produce roughly the same F_1 score. Therefore, LCLR is robust in terms of the selection of p_T, whereas CELR will require a certain amount of adjustment to obtain satisfactory results. In other words, the robust method is not as sensitive to the specific choice of p_T compared to the non-robust approach. This is why the ROC may not be as useful when used with outliers and robust methods. The average F_1 score would be a more appropriate metric in this case.

12.3.6 Ridge and aLASSO

Our final step in this section is to examine how penalty functions impact the results achieved through their application.[5] We know that LCLR produces the best results without the need for a penalty function.

5 Refer to Chapters 5 and 6 for a detailed treatment of penalty functions.

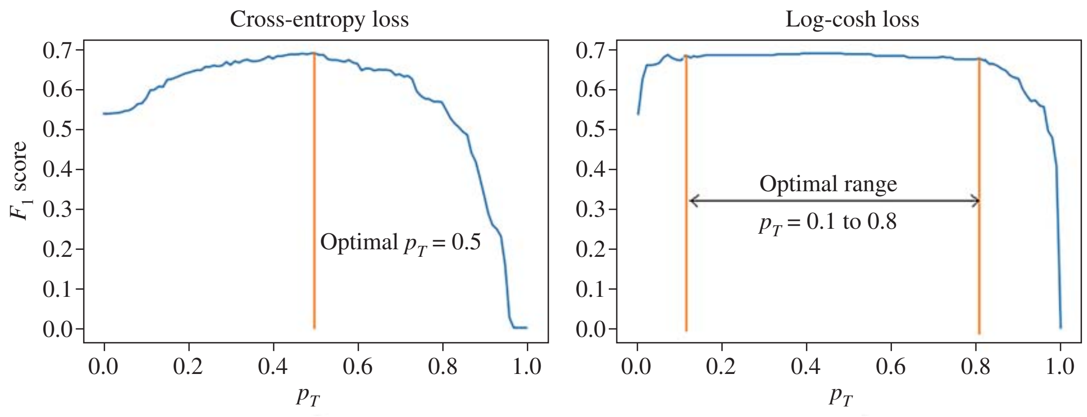


Figure 12.14 F_1 score as a function of p_T.

However, CELR-ridge can improve the results by adjusting the C hyperparameter, but it only provides a marginal improvement. The robust form of logistic regression achieves superior results without requiring regularization.

On the other hand, penalty functions, such as the adaptive LASSO (aLASSO), can still play an important role for LCLR in reducing variance and assessing variable importance. In particular, aLASSO traces provide useful information regarding variable ordering due to the fact that aLASSO has the oracle property. LCLR traces with aLASSO are shown in Figure 12.15. Based on the variable ordering, we find that

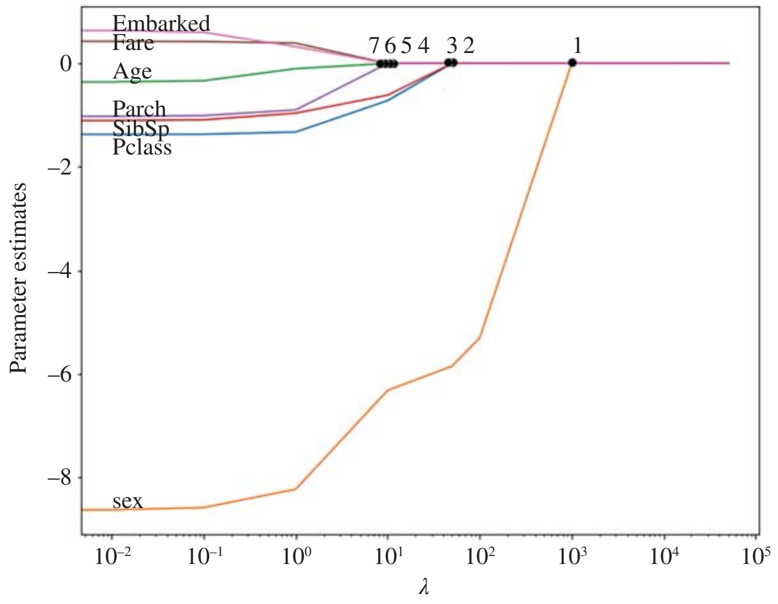


Figure 12.15 Visualization of feature importance using LCLR–aLASSO traces for the Titanic dataset.

`Sex` is a very important parameter in determining whether a passenger survives the sinking of the Titanic since its trace reaches 0 at the largest value of λ in Figure 12.15. In fact, it is the most important variable by far compared to the others. This confirms the analysis carried out during EDA.

Then, following the order given in the figure, we encounter `Pclass` and `SibSp`, followed by `Parch`, `Fare`, `Age`, and `Embarked`. So we can safely conclude that `Sex` is the most important, `Pclass` and `SibSp` are of intermediate importance, while `Parch`, `Fare`, `Age`, and `Embarked` are the least important.

To summarize, LCLR outperformed CELR and it gives us some idea of the maximum accuracy that can be obtained on the dataset. It also provides a list of potential outliers which can be removed to create a clean dataset. In addition, feature importance can be obtained using LCLR-aLASSO in the presence of outliers.

12.4 Application to Explainable Artificial Intelligence (XAI)

The robust machine learning techniques described in this book have a wide range of applications in data science, as detailed in this chapter. However, they also play an important role in an emerging field of research known as explainable artificial intelligence (XAI). The goal is to be able to explain the "why?" of a decision or prediction generated by an AI tool, not just the "how?." The machine learning tools presented in this book, along with many of the generative AI tools such as ChatGPT, can provide a predicted response but often lack transparency in their decision-making process. XAI aims to address this by providing insights into how AI tools arrive at their results to foster more trust and confidence.

Consider the conceptual diagram of Figure 12.16. It is a qualitative view of the accuracy of different machine learning tools against their ease of interpretability. Tools such as linear regression, logistic regression, and decision trees are **model-based techniques** that are largely considered to be interpretable. On the other hand, neural networks, random trees, support vector machines (SVMs), and k-nearest neighbors are called **post hoc techniques** and are more difficult to interpret. For example, in a deep neural network, there may be hundreds of thousands of parameters that are learned but it is very difficult to comprehend how a prediction is made with the final model. The same is true for random forest and SVMs. It is easy to create thousands of trees and make a decision but it is not easy to figure out how a specific decision was made, and if it is a reasonable decision. For SVMs, the decision boundary

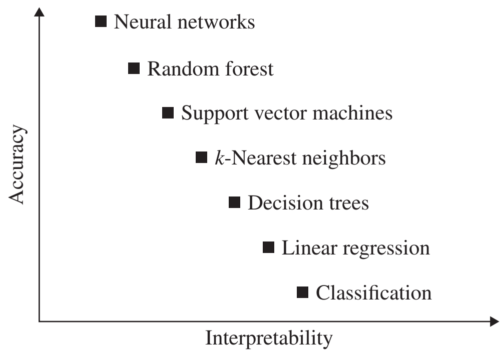


Figure 12.16 Accuracy vs. interpretability of mainstream machine learning tools.

is based on a limited number of support vectors, but very little insight can be gained in terms of the decision-making process. To highlight the nature of the research in XAI, we examine logistic regression as a case study because it is easier to understand than the advanced research being conducted today.

12.4.1 Case Study: Logistic Regression

In this section, the importance of using a robust version of logistic regression is described in the context of XAI. The objective is to show *why* the use of logistic regression based on the cross-entropy loss may lead to incorrect explanations of certain outcomes due to the presence of outliers. On the other hand, the explanations resulting from the robust counterpart can be justified more easily and with more confidence. This will be done using detailed calculations of the probabilities based on the estimates. Interestingly, we have not focused much on the actual estimates produced by the two competing approaches but in this section, we will scrutinize them in detail to determine if the values can be trusted. This is easier to do for logistic regression since it is model based and it will give us an idea of the challenges ahead for the post hoc techniques.

Recall that the cross-entropy loss function is given by (see Chapter 8),

$$J^{\mathrm{CELR}}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}[y_i \log p_i + (1-y_i)\log(1-p_i)],$$

and the log-cosh loss function is (see Chapter 8),

$$J^{\mathrm{LCLR}}(\boldsymbol{\beta}) = \frac{1}{10}\sum_{i=1}^{n}\log(\cosh(10(y_i - p_i))).$$

These two loss functions are minimized to produce estimates of $\boldsymbol{\beta}$. If CELR and LCLR are used on the Titanic training data, the parameter estimates given in two columns of Table 12.12 are obtained. Because LCLR is robust while CELR is not, the two sets of estimates are different. Both sets look reasonable and, in general, it is not possible to tell by simple inspection which is better and in what sense. What we can

Table 12.12 Parameter estimates after training on Titanic dataset using CELR and LCLR.

Parameters	$\hat{\beta}_{\mathrm{CELR}}$	$\hat{\beta}_{\mathrm{LCLR}}$
$\hat{\beta}_{\texttt{Pclass}}$	−1.03	−1.37
$\hat{\beta}_{\texttt{Sex}}$	−2.78	−8.65
$\hat{\beta}_{\texttt{Age}}$	−0.69	−0.30
$\hat{\beta}_{\texttt{SibSp}}$	−0.33	−1.10
$\hat{\beta}_{\texttt{Parch}}$	−0.11	−1.01
$\hat{\beta}_{\texttt{Fare}}$	0.13	0.42
$\hat{\beta}_{\texttt{Embarked}}$	0.16	0.63
$\hat{\beta}_0$	4.36	6.28

say is that LCLR believes that `Sex` is the most important variable because the coefficient is the largest by far. The more relevant questions to be raised are, which model is explainable and which one should we trust?

To answer these questions quantitatively requires elements from the test set along with an analysis of the total contributions of each variable toward the probability computation. Recall that p_i, used in the loss functions above, is given by

$$p_i = (1 + \exp(-\boldsymbol{x}_i^\top \boldsymbol{\beta}))^{-1}. \tag{12.4.1}$$

Therefore, the quantity $\boldsymbol{x}_i^\top \boldsymbol{\beta}$ determines the probability p_i. It is the sum of the products of the estimates with their corresponding data values. We can examine the detailed computations to identify the significant contributors to the probability to assess whether the results are explainable.

Two cases drawn from the Titanic test set of Table 12.8 are listed in separate columns in Table 12.13. The contributions of each parameter to the overall decision can be computed by taking the product of its estimate (Tables 12.12) and data value (Table 12.13). The results are shown in Table 12.14. For example, in the first row of Table 12.14, we have $\hat{\beta}_{\texttt{Pclass}} \times \texttt{Pclass} = -1.03 \times 3 = -3.09$ for CELR and $\hat{\beta}_{\texttt{Pclass}} \times \texttt{Pclass} = -1.37 \times 3 = -4.11$ for LCLR. The same calculations are carried out for all rows in the table. The sum of product values are provided at the end of the table along with the corresponding values of p_i (using Eq. (12.4.1)) and predicted response for each test case.

The tabular data of Table 12.14 can be converted to bar graphs as in Figure 12.17 to visualize the results. From the bar graph on the right, it is clear that the decision made by LCLR of whether the passenger *survived* (1) or *did not survive* (0) is largely due to the `Sex` of the passenger (*male* in this case), with some influence of `Pclass`. For CELR, it is a combination `Pclass`, `Sex`, and `Age` that leads to the decision. In both cases, the ground truth is that the passenger *did not survive* (0) and both methods produce the same decision, CELR with probability 0.075 and LCLR with probability 0.002 of survival, respectively. However, the robust solution is more easily explained and believable: it is mostly controlled by the `Sex` of the passenger.

Table 12.13 Two test cases from the Titanic test set used to compare CELR and LCLR.

Parameters	**Data for Test 1**	**Data for Test 2**
`Pclass`	3	3
`Sex`	1	0
`Age`	1.70	2.35
`SibSp`	0	1
`Parch`	0	0
`Fare`	0.157	0.140
`Embarked`	1	2
`Survived` (ground truth)	0	1

Table 12.14 Parameter × value for CELR and LCLR on Test cases 1 and 2 from Table 12.12.

	Test 1		Test 2	
Parameter × value	**CELR**	**LCLR**	**CELR**	**LCLR**
$\hat{\beta}_{\texttt{Pclass}} \times \texttt{Pclass}$	−3.09	−4.11	−3.09	−4.11
$\hat{\beta}_{\texttt{Sex}} \times \texttt{Sex}$	−2.78	−8.65	0	0
$\hat{\beta}_{\texttt{Age}} \times \texttt{Age}$	−1.17	−0.51	−1.62	−0.71
$\hat{\beta}_{\texttt{SibSp}} \times \texttt{SibSp}$	0	0	−0.33	−1.10
$\hat{\beta}_{\texttt{Parch}} \times \texttt{Parch}$	0	0	0	0
$\hat{\beta}_{\texttt{Fare}} \times \texttt{Fare}$	0.02	0.07	0.02	0.06
$\hat{\beta}_{\texttt{Embarked}} \times \texttt{Embarked}$	0.16	0.63	0.32	1.26
$\hat{\beta}_0 \times 1$	4.36	6.28	4.36	6.28
Sum of products, $\boldsymbol{x}_i^{\mathrm{T}} \hat{\boldsymbol{\beta}}$	−2.5	−6.3	−0.34	1.64
Probability, p_i	0.075	0.002	0.42	0.84
Predicted response	0	0	0	1

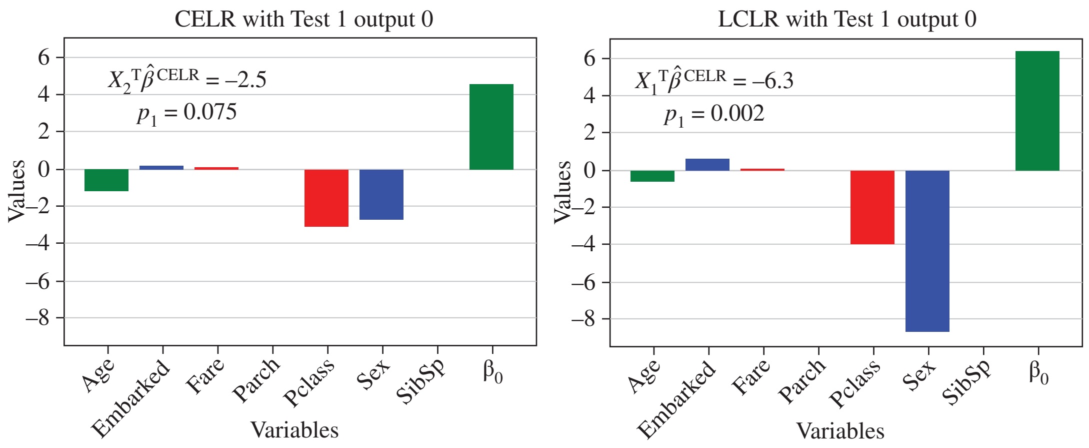


Figure 12.17 Bar graphs for CELR and LCLR depicting values of Table 12.14 using Test 1 of Table 12.13.

If the same steps are applied to Test 2, a different explanation is needed. The results are also tabulated in Table 12.14 and shown as bar graphs in Figure 12.18. Note that `Sex` is set to 0 for *female* so it does not play any direct role in the decision (there is no bar in Figure 12.18 for `Sex`). In this case, the ground truth is that the passenger *survived* (1) and LCLR did indeed arrive at this conclusion with a probability 0.84. Using $p_T = 0.5$ as the threshold for CELR, the computed probability is only 0.42 which implies that the passenger *did not survive* (0), which is wrong. We see a discrepancy here and only LCLR produces an acceptable result. When `Sex` is set to 0, all other variables control the outcome. In particular, the value of β_0 needs to be high enough to overcome the other components to obtain the correct result. The conjecture is that outliers are corrupting the estimates for CELR so that it is not able to handle this case.

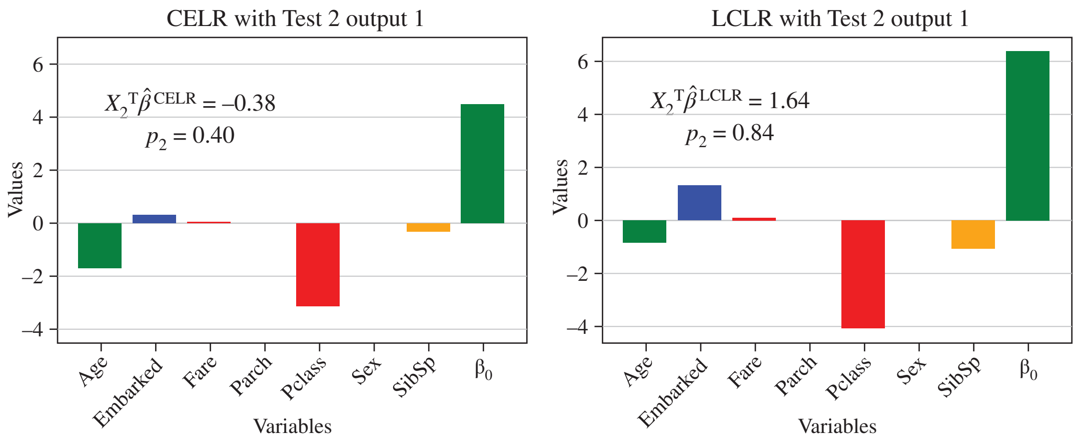


Figure 12.18 Bar graphs for CELR and LCLR depicting values of Table 12.14 using Test 2 of Table 12.13.

But how far is CELR from the ground truth, and what changes in variable values would be needed for LCLR to make the wrong decision? These types of questions are referred to as **counterfactuals**. It involves the adjustment of the test data values (typically one at a time) to obtain a desired outcome. For each variable, the data is increased and decreased to understand how it affects the response. In the case of LCLR, we want to know what specific changes in values would lead to an incorrect decision. Conversely, since CELR is not correct, we would ask what changes in data values would lead to a correct decision. These are unusual questions in that we are trying to coerce the opposite outcome from what the algorithm produces, hence the name counterfactuals.

When testing counterfactual hypotheses, it is important to bound the values of the variables to the appropriate ranges. For example, an `Age` value of 150 is unreasonable so it should not be allowed. Likewise, `Pclass` cannot be set to 4 since it can only hold values of 1, 2, or 3. Given these types of constraints on the Titanic dataset, it is difficult to flip the decision in LCLR unless `Parch` and `Sibsp` are increased. This implies that *the decision made by LCLR is quite robust and tolerant to small changes in the data*. However, the incorrect decision made by CELR can be flipped easily by adjusting either `Pclass`, `Age`, `Fare`, or `Embarked`. The decision is not robust and reflects the characteristics of CELR itself. Therefore, when faced with the choice of robust methods and non-robust methods, it is clear that robust methods offer the opportunity to make intuitive and reasonable explanations for binary decisions made in classification problems because they are quite stable. For CELR, the explanations are not as reliable in the presence of outliers.

Exercise: Consider test data from row 3 of the Titanic dataset in Table 12.8 and the estimates of Table 12.12. Compute the probability of survival using the CELR and LCLR estimates. Which is closer to the ground truth? What happens if `Pclass=1`?

Ans.: Using the steps given in this section, $p_3 = 0.2$ for CELR and $p_3 = 0.007$ for LCLR. In this case, LCLR is closer to the ground truth which is 0 (did not survive). If `Pclass=1`, then $p_3 = 0.4$ for CELR and $p_3 = 0.027$ for LCLR. It does not affect LCLR by much whereas CELR is greatly affected by this change. □

In addition to the choice of robust methods over non-robust methods, there is a similar choice in the penalty functions. In particular, for problems where many of the parameter values are ideally 0, the LASSO penalty function (see Chapters 5 and 6) is often coupled with CELR. The combined loss function is

$$J^{\text{CELR-LASSO}}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}(y_i \log p_i - (1-y_i)\log(1-p_i)) + \sum_{j=1}^{d}|\beta_j|.$$

Unfortunately, CELR is not robust and LASSO does not possess the oracle property and therefore may not asymptotically produce the proper variable ordering. On the other hand, using LCLR with aLASSO provides robustness and the oracle property. The combined loss function is given by

$$J^{\text{LCLR-aLASSO}}(\boldsymbol{\beta}) = \frac{1}{10}\sum_{i=1}^{n}\log(\cosh(10(y_i - p_i))) + \sum_{j=1}^{d}\frac{|\beta_j|}{|\beta_j^*|},$$

where β_j^* is a known quantity typically set to the unpenalized estimate, $\hat{\beta}_j^{\text{LC}}$.

The difference between the two approaches is illustrated in Figure 12.19 using the Titanic dataset. The CELR–LASSO and LCLR–aLASSO traces are shown and there are notable differences. In the CELR–LASSO case, `Pclass` is considered to be as important as `Sex`. However, we know that this is not true. In fact, the LCLR–aLASSO trace shows that the `Sex` of the passenger is the most important variable, far more important than the `Pclass`. In addition, CELR–LASSO indicates that the `Age` is as important as the remaining of the variable whereas LCLR–aLASSO indicates it is less important. The previous analysis of the contributions of each variable to the predicted outcome shows that `Age` should not be considered as important as `SibSp` and that is exactly what LCLR-aLASSO shows in its traces. Furthermore, many `Age` values were imputed as they were missing so its true importance would be hard to gauge. Hence, CELR and LASSO both fall short of the desired goals of XAI and lead to explanations that may not be correct. This is why the LCLR and aLASSO combination is preferable. Conclusions reached using this combination may have more credibility and stability, and hence explainability.

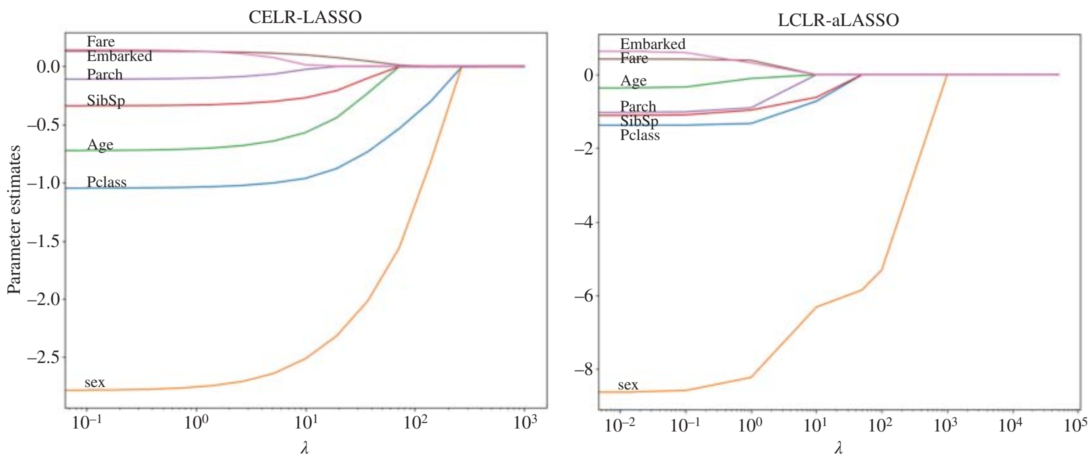


Figure 12.19 Comparison of CELR–LASSO with LCLR–aLASSO traces for the Titanic dataset.

To summarize, robust methods can play an important role in XAI. When explaining model predictions, robust logistic regression ensures that extreme data points do not disproportionately affect the interpretation of feature importance. By downplaying the influence of outliers, robust logistic regression provides more reliable estimates of coefficients. Users can trust the model's coefficients even when the dataset contains noisy or anomalous observations. By handling outliers effectively, it enhances the transparency of the model's decision boundaries. In some sense, robustness helps maintain fairness by preventing outliers from disproportionately affecting predictions.

12.4.2 Case Study: Neural Networks

In the previous section, the focus was on the issue of model interpretability in the context of logistic regression but this comprises only one component of the field of XAI. In a broader context, XAI refers to a set of processes and methods that make the results and workings of AI models understandable to humans. It involves interpretability, transparency, trust, and bias. Most of the activity is focused on neural networks where the challenges are much greater.

Consider Figure 12.20 which shows a 4-input, 1-output model for logistic regression, and a neural network. There are only 5 parameters in logistic regression but there are 65 parameters in the neural network. Interpretability is straightforward in logistic regression because there is a one-to-one correspondence between the variables and the parameters. This is not the case in the neural network. It is very difficult to sequence through the same steps as done in the previous section to understand how each weight in the neural network controls the outcome and how to interpret the results. There are so many parameters and nonlinearities in the network that the complex relationships between the input and output are hard to unwind.

Taking it to the extreme, in generative AI today, there are up to 32,000 inputs (referred to as tokens), and up to 50,000 probabilistic (i.e. softmax) outputs in the large language models (LLMs) of tools like ChatGPT. The number of trained parameters exceeds 1.5 trillion. The meaning of XAI is quite different in this setting. Due to the "black-box" nature of neural networks, especially for deep learning models,

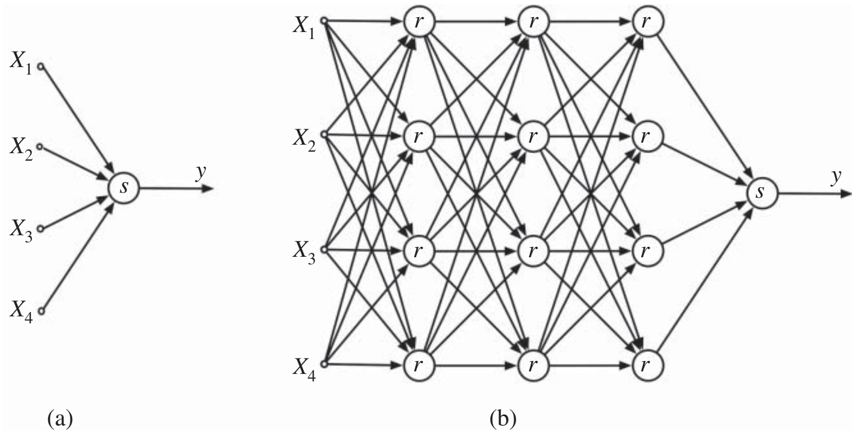


Figure 12.20 Comparing XAI in (a) logistic regression and (b) neural networks.

transparency about the inner workings of AI models, i.e. making it clear and understandable, has become a major area of focus for XAI. Techniques like feature importance, layer-wise relevance propagation, and attention mechanisms are used to explain their decisions. Of particular importance are whether the responses are trustworthy and whether there are systemic built-in biases.

Research is currently underway in a number of different directions. One approach is to use a Bayesian neural network, an extension of the traditional neural network that incorporates Bayesian statistics to estimate uncertainty in the predictions. Instead of weights that have fixed values, they are represented by probability distributions. The degree of uncertainty can give insights into the model's trustworthiness and make it more explainable. Other promising directions include SHapley Additive exPlanations (SHAP), which provides information about how much each feature contributes to a prediction, and Layer-wise Relevance Propagation (LRP), which provides relevance scores to each neuron using a back propagation scheme. While these issues are beyond the scope of this book, it is worth noting that much more study is needed about the role of robust methods in this emerging field. Ongoing research in this challenging area would involve finding links between models and outcomes for all aspects of machine learning, including random forest and SVMs. However, most of the future work in the field will focus on generative AI models.

12.5 Time Series Example: Climate Change

Thus far in this book, we have examined ways to build outlier-tolerant machine learning tools and identified methods for outlier detection and removal. Once the concepts described herein have been fully digested and mastered, the reader is encouraged to seek new opportunities to apply these methods. The basic ideas can be extended to many other application areas. To close out the book, we consider one such application to demonstrate how future work in data science can be conducted using the knowledge gained from this book.

Consider the problem of climate change. This is currently a topic of great concern around the world due to the potentially disruptive effects of high CO_2 levels heating the Earth's atmosphere. In 1968, the measured level of CO_2 was roughly 322 parts per million (ppm) whereas by 2025, it reached 425 ppm. That is an increase of about 30% in less than 6 decades. Atmospheric CO_2 creates a "greenhouse" effect which leads to the heating of the land and oceans around the Earth. As a consequence, the ice fields around the North and South poles are melting which increases sea levels and releases methane (another greenhouse gas) trapped in the ice. Climate disruption is the net result and it may have severe consequences if not dealt with in the near future.

As a measurable indicator of heating effects over time, the average land temperature from 1750 to 2015 is shown in Figure 12.21. From 1750 to 1900, there are wild fluctuations in temperature but the average is roughly 8°C. Beyond 1900, the start of the Industrial Revolution, there is a noticeable upward trend in the temperature. This is the effect of climate change. Going forward, we want to be able to predict the temperature increase using machine learning by treating this data as a time series. Thus, we need to consider using the techniques available in time series analysis.

A univariate **time series** is constructed by measuring some quantity of interest in uniform increments of time, e.g. each day, month, or year. Since the measurements are evenly spaced, the time series can be represented as a sequence of numbers occurring at different integer values of time. Based on this

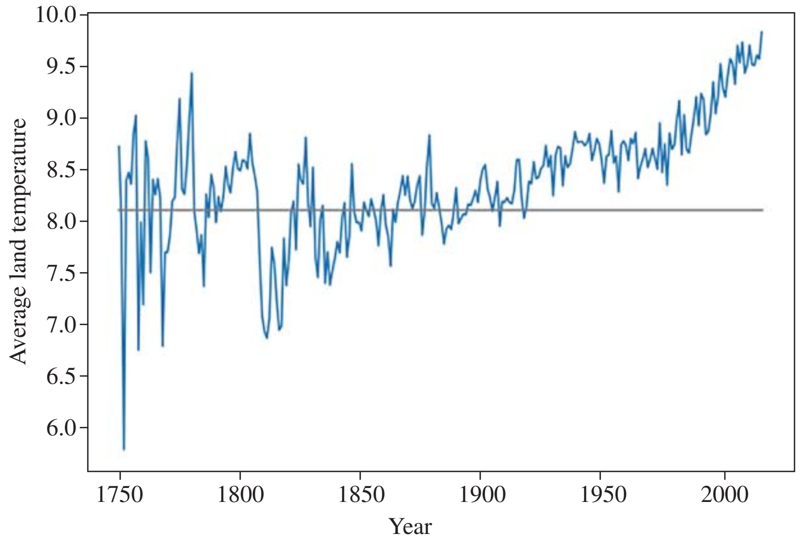


Figure 12.21 Time series of the average land temperature on Earth from 1750 to 2015.

assumption, a set of statistical procedures have been developed for modeling and prediction. Specifically, we will examine the ARIMA time-series modeling approach used in machine learning (see Cryer and Chan 2008). The acronym ARIMA stands for Autoregressive Integrated Moving Average. We will sequence through the different components of ARIMA and then apply it to the specific time series associated with climate change in Figure 12.21. Keep in mind that the purpose of this exercise is to illustrate how to robustify existing machine learning techniques.

12.5.1 Autoregressive Model

The first component of ARIMA is the autoregressive (AR) model. In such a model, the future value of the time series is constructed as a linear combination of past values of the time series. We use the notation AR(p) to refer to models that use a total of p previous values in time to predict the value at time $p + 1$. More concretely, assume we have a time series given by $\{x_1, x_2, \dots, x_n\}$. If we have a model with $p = 6$, we use the first 6 time points to predict the 7th time point. Likewise, the values from positions 2 to 7 are used to predict the value at the 8th time point, and so on. It is possible to create a matrix $\boldsymbol{X}$ with p values in each row and $n - p$ rows, and a response vector $\boldsymbol{y}$ with the $(p + 1)$st values associated with each row. Then, we can apply linear regression using least squares estimation to produce an AR(p) model. Alternatively, and perhaps more in line with our objective, the log-cosh loss can be used to robustify AR(p).

More generally, for a model of order p, we consider the linear regression problem of the form,

$$y_t = \varphi_0 + \varphi_1 x_{t-1} + \cdots + \varphi_p x_{t-p} + \varepsilon_t, \quad p + 1 \leq t \leq n, \tag{12.5.1}$$

where ε_t is the noise, and $\varphi = (\varphi_0, \varphi_1, \dots, \varphi_p)^\top \in \mathbb{R}^{p+1}$ is a vector of unknown autoregressive parameters with a response vector $y_t = x_t$. This is called the autoregression process of order p (≥ 1). It is a form of *self-regression* since it creates rows of the dataset by repeated use of the values in its own time series based on a sliding window of size $p + 1$.

As a more concrete example, consider an AR(p) model given by

$$\begin{array}{cccccccc} \boldsymbol{y} & = & \boldsymbol{X} & & \boldsymbol{\varphi} & + & \boldsymbol{\varepsilon} & \\ \begin{pmatrix} x_{p+1} \\ x_{p+2} \\ \vdots \\ x_n \end{pmatrix} & = & \begin{pmatrix} x_p & x_{p-1} & \cdots & x_1 & 1 \\ x_{p+1} & x_p & \cdots & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n-1} & x_{n-2} & \cdots & x_{n-p} & 1 \end{pmatrix} & & \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_0 \end{pmatrix} & + & \begin{pmatrix} \varepsilon_{p+1} \\ \varepsilon_{p+2} \\ \vdots \\ \varepsilon_n \end{pmatrix} & . \end{array} \tag{12.5.2}$$

The $\boldsymbol{y}$ column on the left is the time series starting at $p + 1$ and ending at n. Each row of the $\boldsymbol{X}$ matrix is a contiguous set of p values of the time series. The error term $\boldsymbol{\varepsilon}$ is assumed to be Gaussian. The goal is to estimate $\boldsymbol{\varphi}$ using robust linear regression.

Exercise: Given a time series $\boldsymbol{x} = \{x_1, x_2, x_3, x_4, x_5\}$, what is the matrix equation for an AR(p) model with $p = 1$?

Ans.: From Eq. (12.5.2), it is straightforward to write the AR(1) model as

$$\begin{array}{cccccc} \boldsymbol{y} & = & \boldsymbol{X} & \boldsymbol{\varphi} & + & \boldsymbol{\varepsilon} \\ \begin{pmatrix} x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} & = & \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \end{pmatrix} & \begin{pmatrix} \varphi_1 \\ \varphi_0 \end{pmatrix} & + & \begin{pmatrix} \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \end{pmatrix} . \end{array}$$

□

If we now apply AR(1) to the average land temperature time series of Figure 12.21 and plot the first column of the $\boldsymbol{X}$ matrix (which we call $\boldsymbol{x}$) vs. $\boldsymbol{y}$, we obtain Figure 12.22. Note that there are a few outliers so a robust regression method may be better suited to the problem. The results of the robust regression followed by a point-by-point prediction of the time series using the model are shown in Figure 12.23 along with the original time series. Here, the log-cosh loss was used rather than least squares. In some sense, this is a robustified AR(p) model.

Examining Figure 12.23, the original time series and the predicted time series are very close. But this is to be expected because the parameters φ_0 and φ_1 were learned using the original time series. So the fact that they are close is not surprising. On closer inspection, the predicted values lag the original values and the prediction gets worse as time evolves. Note that no future predictions are made as yet; only values within the span of the time series have been computed thus far.

The time lag and errors can be reduced by using higher orders of AR(p). For example, in Figure 12.24, we see the original time series and the AR(1), AR(4), and AR(12) predictions. Clearly, AR(12) is the closest to the original time series but there are two issues regarding this choice. The first is that the model has $p + 1 = 13$ parameters which could lead to overfitting. The second is that it may be more computationally expensive than necessary. A smaller value of p may be sufficient to obtain a more general model without the excessive computation time. But this is a good start in modeling the time series since all the predictions are close to the true values.

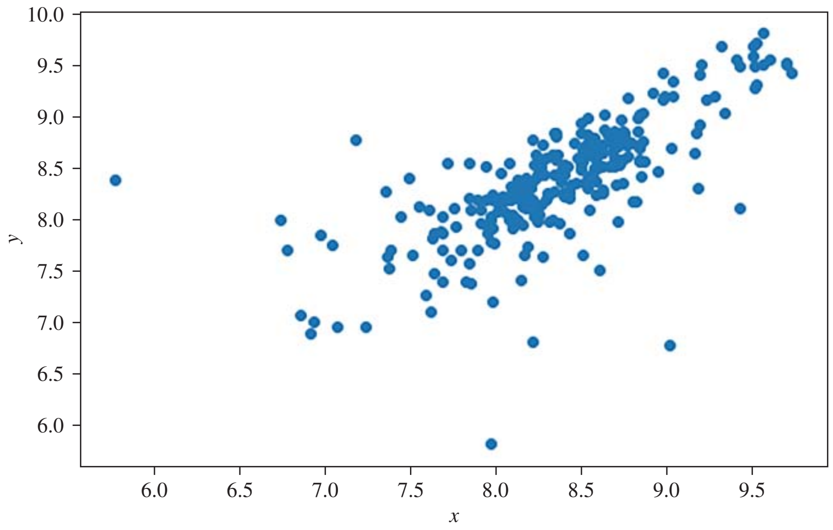


Figure 12.22 Scatter plot of ***y*** and ***x*** for an AR(1) model of the average land temperature time series indicating the presence of outliers.

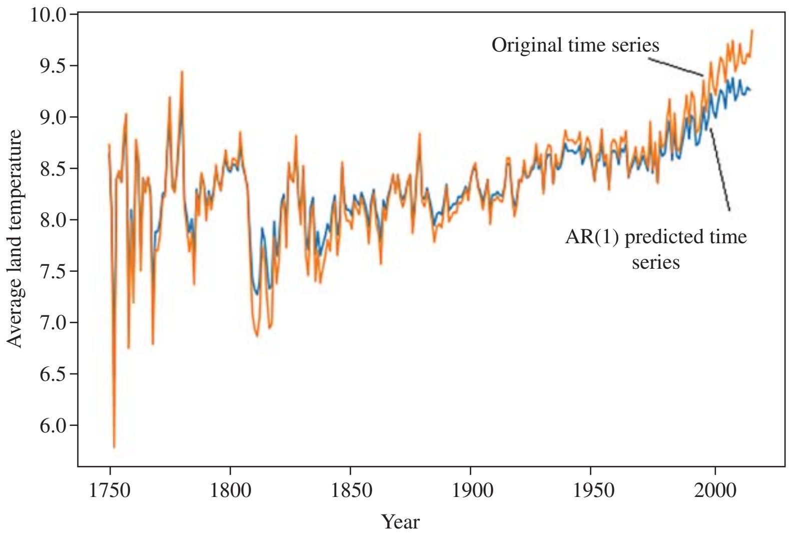


Figure 12.23 Applying AR(1) to the average land temperature time series.

12.5.2 Forecasting Using AR(p)

The purpose of machine learning is to be able to make predictions based on a given set of data, but the prediction is not of the type just described. We want to predict the future of a time series, i.e. produce a forecast, not simply replicate what we already know about its past. The way to do that is to

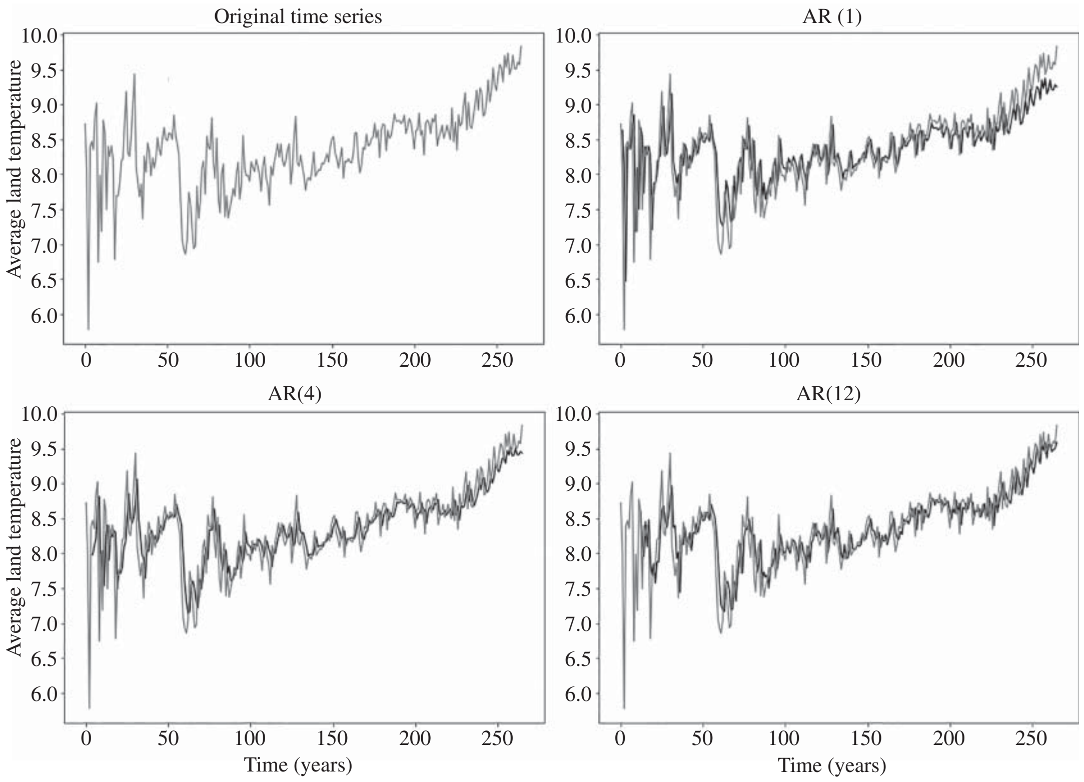


Figure 12.24 Comparison of AR(1), AR(4), and AR(12) on climate change time series.

define a training set and a test set, then build a model from the training set, and see how well it aligns with the time series in the test set. This is consistent with common practice in machine learning and seems rather straightforward but it is challenging to produce accurate predictions when dealing with time series.

To illustrate the difficulty, consider the forecasting problem using the climate data. Specifically, the time series for the average land temperature contains 266 values, of which 254 can be used to train the model and 12 can be placed in the test set for prediction purposes. Effectively, we are using 254 years of data to predict 12 years into the future. To simplify the explanation, we use AR(1) on the training set and build a model with only φ_0 and φ_1. The forecast can be carried out recursively using the estimates of φ_0 and φ_1 with the iterative equation,

$$x_{t+1} = \hat{\varphi}_0 + \hat{\varphi}_1 x_t, \tag{12.5.3}$$

where $t = 254, \dots, 266$ in our example. Using this equation, the forecast can be completed in 12 iterations.

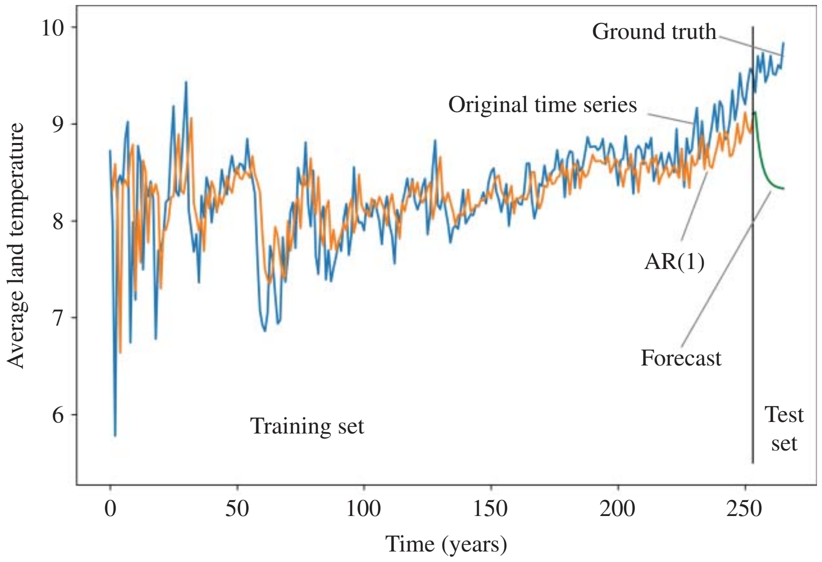


Figure 12.25 Creating training and test sets using the climate change time series. The test set and forecast are on the right side of the figure.

Exercise: An AR(1) model produces the estimates $\hat{\varphi}_0 = 2.0$ and $\hat{\varphi}_1 = 0.7$. Predict the next three values of the time series at $n + 1$, $n + 2$, and $n + 3$. Assume that $x_n = 9.0$.

Ans.:

$$x_{n+1} = \hat{\varphi}_0 + \hat{\varphi}_1 x_n = 2.0 + 0.7 \times 9.0 = 8.3$$

$$x_{n+2} = \hat{\varphi}_0 + \hat{\varphi}_1 x_{n+1} = 2.0 + 0.7 \times 8.3 = 7.8$$

$$x_{n+3} = \hat{\varphi}_0 + \hat{\varphi}_1 x_{n+2} = 2.0 + 0.7 \times 7.8 = 7.5$$

□

The results for the climate change data are shown in Figure 12.25 which identifies the time intervals for the training and test sets and provides plots of the original time series and the AR(1) predicted time series for the training set. The forecast is also shown in the test set region of the figure. It is not only inaccurate but it is going in the wrong direction. So clearly AR(1) by itself is not sufficient in this case. One would think that applying the same procedure using AR(4) or AR(12) would improve the forecast, but increasing p does not improve the forecast accuracy by much, if at all. Another approach is needed to obtain satisfactory results.

12.5.3 Stationary Time Series

To understand the source of the problem, it is important to recognize that the time series is not **stationary**. A stationary time series has a constant mean value across time about which the series fluctuates (i.e. moves above and below), and its range of fluctuation around that mean remains bounded over time. Neither of these requirements are satisfied by the time series of Figure 12.21. Because it is not stationary, it will be difficult to build a good model for prediction using AR(p) due to an underlying assumption in AR(p) that the time series is stationary. However, there is a well-known solution to this problem: we can apply a *difference operation* to the time series which increases the likelihood that it is stationary. If it is still not stationary, we can apply a second difference operation (usually two is sufficient) and then apply AR(p) on the resulting time series. This process is the *Integrated* part of ARIMA and uses the notation I(d), where the parameter d (unrelated to the dimensions of the problem) specifies the desired number of differencing operations (which can be set to 0, 1, or 2).

The effect of differencing is to *de-trend* the time series and make it stationary. That is, we want to remove any dependence of each point with its prior point or future point so that the points used in the regression are independent and come from the same distribution. The differencing operation itself is rather straightforward. The first difference is simply

$$\delta_t^{(1)} = x_t - x_{t-1} \quad t = 2, \dots, n. \tag{12.5.4}$$

The notation $\delta_t^{(d)}$ refers to d differencing operations at time t. In the process, the length of the new time series is reduced by one relative to the original series. The second difference is given by

$$\delta_t^{(2)} = \delta_t^{(1)} - \delta_{t-1}^{(1)} \quad t = 3, \dots, n. \tag{12.5.5}$$

Again, the length of the newer time series is reduced by two relative to the original series. The result after two differencing operations is shown in Figure 12.26 for the average land temperature time series. Note

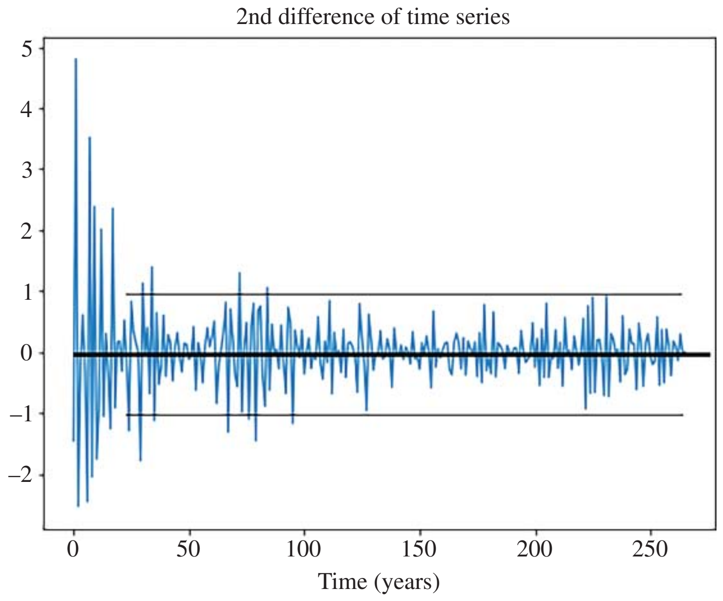


Figure 12.26 Results of second differences of the average land temperature time series which produces a stationary time series.

that there is a clear mean around 0 and the fluctuations mostly live within the two bounds indicated in the figure. Finally, we can apply AR(p) to this stationary time series and then reverse the whole process to predict the values of the average land temperatures. Only the last $n - p - d$ points in the time series can be predicted since a combination of AR(p) and the differencing operation removes the early points from the time series. However, this does not pose a problem when using the derived model to make a forecast of future values.

For the AR(1) case with $d = 2$, the forecasting equations for the next time point at $n + 1$ are

$$\delta_{n+1}^{(2)} = \hat{\varphi}_0 + \hat{\varphi}_1 \delta_n^{(2)},$$

$$\delta_{n+1}^{(1)} = \delta_n^{(1)} + \delta_{n+1}^{(2)},$$

and

$$x_{n+1} = x_n + \delta_{n+1}^{(1)}.$$

These equations can be used iteratively to construct the complete forecast for the desired future interval. Extensions to higher-order AR(p) models are straightforward.

The results of these two steps for AR(12) with I(2) are given in Figure 12.27. This time, the prediction is quite good and, in fact, it is trending in the right direction. Therefore, $p = 12$ and $d = 2$ are suitable values for this example. On the other hand, there is still room for improvement. The forecast tends to overshoot the test set in this example. The question is, can we do better?

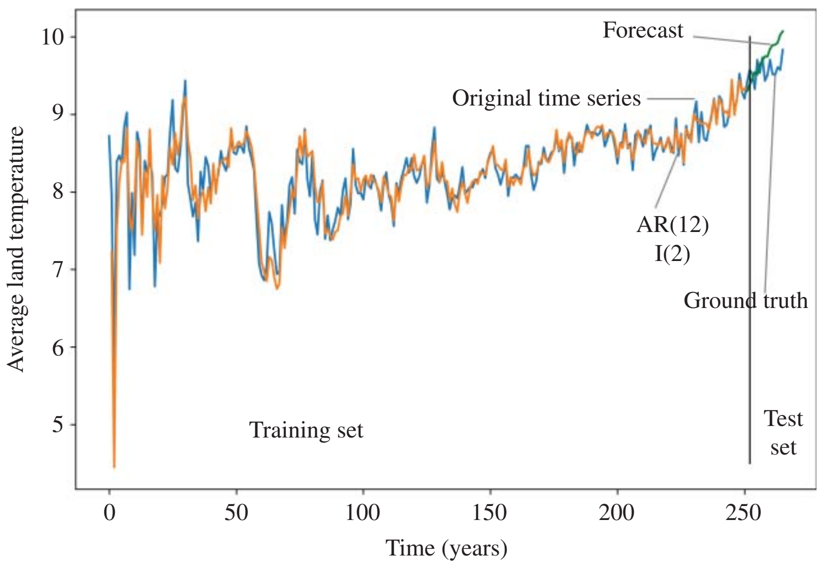


Figure 12.27 Using AR(12) with two differences ($d = 2$) to model and predict average land temperature time series.

12.5.4 Moving Average

There is one last part of ARIMA that involves the moving average (MA) which can further improve the results. The term itself is somewhat misleading because it involves creating an autoregression around the *residual errors at each time point* rather than on a true moving average of the time series. The notation used in ARIMA is MA(q), where q is the order of the MA model. If we wanted to use such a model, it would have the form

$$y_t = \theta_0 + \theta_1 r_{t-1} + \dots + \theta_q r_{t-q} + \varepsilon_t, \quad q+1 \le t \le n,\ q \ge 1, \tag{12.5.6}$$

where r_{t-j} are the residuals at each point in the time series (the original time series, the first difference, or the second difference, depending on the value of d). Strangely, the residuals of the past are being used to predict the time series value in the future. But if used in conjunction with AR(p) and I(d), it can produce more accurate models. In order to compute residuals, we must know the previous values and predicted values at each time point in the training set. Therefore, this process is iterative which adds much more computational complexity to the problem, but it may be needed in practice to obtain satisfactory results.

The complete ARIMA(p, d, q) model is given by

$$y_t = \varphi_0 + \sum_{i=1}^{p} \varphi_i \,\delta_{t-i}^{(d)} + \sum_{j=1}^{q} \theta_j \, r_{t-j} + \varepsilon_t. \tag{12.5.7}$$

Here $y_t = \delta_t^{(d)}$ by definition, and the $\delta_{t-i}^{(d)}$ terms in the first sum are the differences between adjacent time points (as determined by I(d)), whereas the r_{t-j} terms in the second sum are the residual differences at each time point $(t-j)$, respectively. In general, the choices of p, d, and q are based on experience, good judgment, or simply trial-and-error.

Exercise: Given a time series $\boldsymbol{x} = \{x_1, x_2, x_3, x_4, x_5\}$, what is the matrix equation for an ARIMA(1,1,1) model?

Ans.: First note that because $d = 1$, we will have to find the first difference of the given data. Therefore, $\delta_2^{(1)} = x_2 - x_1$, $\delta_3^{(1)} = x_3 - x_2$, $\delta_4^{(1)} = x_4 - x_3$, and $\delta_5^{(1)} = x_5 - x_4$. Furthermore, the residuals are $r_i = \delta_i^{(1)} - \delta_i^{(1)\text{old}}$, where $\delta_i^{(1)\text{old}}$ is the predicted value in the previous iteration. Then, we can write that

$$\begin{pmatrix} \delta_3^{(1)} \\ \delta_4^{(1)} \\ \delta_5^{(1)} \end{pmatrix} = \begin{pmatrix} \delta_2^{(1)} & r_2 & 1 \\ \delta_3^{(1)} & r_3 & 1 \\ \delta_4^{(1)} & r_4 & 1 \end{pmatrix} \begin{pmatrix} \varphi_1 \\ \theta_1 \\ \varphi_0 \end{pmatrix} + \begin{pmatrix} \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \end{pmatrix}.$$

□

Returning to the climate change problem, the previous example essentially used ARIMA(12,2,0), since q was not specified. It is time to revisit the settings of p, d, and q. First, large values of p may lead to overfitting and an increase in the computational cost of training. Therefore, the smallest value of p that produces acceptable results should be selected. Next, for d, a value of 1 or 2 is recommended. Finally, for q, a low value is usually sufficient. The results shown in Figure 12.28 use ARIMA(8,2,1) to avoid overfitting while still producing an acceptable prediction. The prediction of the time series is much improved and the forecast is now in line with expectations based on the test set. The autoregressive model can be solved using a log-cosh loss function making it a robust ARIMA model. Further work on robust ARIMA is necessary to identify the benefits and limitations of such an approach.

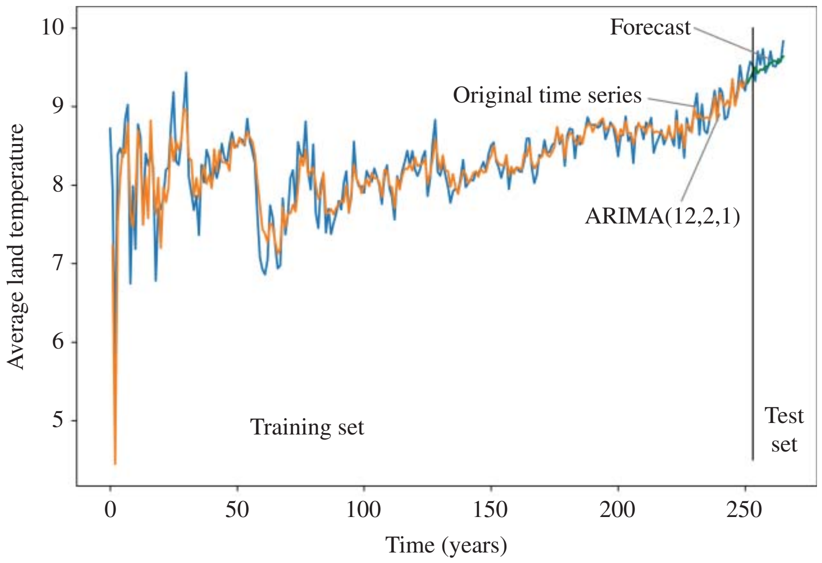


Figure 12.28 Using AR(8) with two differences I(d = 2) and MA(1) to accurately predict and forecast average land temperature time series.

12.5.5 Finding Outliers in Time Series

As with other topics in this book, the alternative to robust methods is to detect outliers and handle them in some manner. Outliers occur in time series because time series tend to be noisy and may have significant fluctuations, as seen in Figure 12.21. The average land temperature varies significantly in the early years of this figure, but it stabilizes as time progresses. The effect of outliers is dependent on the nature of the analysis being conducted. Statistical quantities such as mean, variance, and autocorrelation will all be skewed by outliers in the context of time series. They can also make forecasts less reliable. Therefore, outlier detection is an important part of time series analysis.

There are many ways to find outliers in a time series. Visualization is one way to quickly identify the existence of outliers, especially in the case of a one-dimensional time series. However, the robust statistical approaches described in Chapter 4 are better suited for outlier detection. In particular, box-plots applied to the first or second difference of the time series (since they are de-trended) would be very effective in detecting outliers. Robust z-scores can also be used in conjunction with an edit rule between 4.5-MAD and 15-MAD, depending on the time series of interest. Advanced methods, such as Isolation Forest and DBSCAN described in Chapter 11, can also be evaluated for this purpose. Furthermore, the robust clustering techniques of MADmax may be useful along with the associated outlier detection methodology.

For outlier removal, rather than trimming the data (which can be dangerous), it is more appropriate to winsorize the data since every time point requires some data. There are methods to place upper and lower bounds on the data, and then these bounds can be used to replace outliers. Different quantiles, such as the 5th/95th or 1st/99th combinations, are suitable for upper and lower bounds. Careful attention must be paid to their use on de-trended time series rather than the original time series. These and other details of outlier detection and removal described throughout the book can all be adapted for use with time series. This book has equipped you with the knowledge and "know-how" to solve these and other advanced problems in data science.

12.6 Summary and Conclusions

In this chapter, we have opened the door to robust data science by illustrating some of the novel capabilities available in robust machine learning. We have demonstrated that it is superior in many respects to the traditional methods. Two well-known datasets, such as Boston Housing and Titanic, were used to demonstrate its viability for linear regression and binary classification. Its viability for multi-class classification and neural networks was demonstrated in earlier chapters. The information provided in this book was also applied to time series analysis as a case study of how the knowledge gained from the book can be used. However, we have only scratched the surface of the possible data science problems that may be addressed using robust machine learning. Many research topics also remain unexplored in this area. We see a large potential for future work and expect there to be fruitful outcomes for those who pursue this topic. Our hope is that this book and all of the new concepts and ideas will resonate with readers and provide motivation to pursue some of these ideas in more detail in the near future. Greatness awaits those with deep knowledge.

Problems

At this stage, there is no better way to learn the material than to write Python code to produce the results in this chapter associated with the Titanic dataset. We recommend that you visit the kaggle.com website to gain access to the dataset and examine other Jupyter notebooks on the site to carry out the assignments below.

12.1 (PROJECT) Using the `tensorflow` library, write Python code to perform logistic regression using the general log-cosh loss function and the cross-entropy loss function, as described in Section 12.3. The name of the data file here is `titanic.csv`. However, you may need to modify it according to the location of the training file on your computer or on the website kaggle.com.

a) Import the needed libraries.

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

b) Read the data, drop certain unneeded columns.

```
#Read the titanic dataset
train_df = pd.read_csv("titanic.csv")
#Get rid of some unneeded columns
train_df = train_df.drop(['PassengerId', \
    'Ticket','Name','Cabin'],axis=1)
```

c) Process data to fill in missing data, scale certain columns, and convert strings to numbers.

```
# Process data as described in this chapter
if train_df.iloc[829]['Survived'] == 0:
    train_df['Embarked'] = \
        train_df['Embarked'].fillna('S')
else:
    train_df['Embarked'] = \
        train_df['Embarked'].fillna('C')
train_df['Sex'].loc[train_df['Sex'] == 'male'] = 1
train_df['Sex'].loc[train_df['Sex'] == 'female'] = 0
train_df['Embarked'].loc[train_df['Embarked'] \
    == 'Q'] = 1
train_df['Embarked'].loc[train_df['Embarked'] \
    == 'S'] = 2
train_df['Embarked'].loc[train_df['Embarked'] \
    == 'C'] = 3
# Get average, std, and number of NaN values
average_age_titanic   = train_df["Age"].mean()
std_age_titanic       = train_df["Age"].std()
count_nan_age_titanic = \
    train_df["Age"].isnull().sum()
rand_1 = np.random.randint(average_age_titanic - \
    std_age_titanic, average_age_titanic + \
    std_age_titanic, size = count_nan_age_titanic)
# Fill mssing values in Age column
train_df["Age"][np.isnan(train_df["Age"])] = \
    rand_1
train_df['Age'] = train_df['Age']/20.0
train_df['Fare'] = \
    train_df['Fare']/50.0
train_df.head()
```

d) Create the needed ***X*** and ***y*** data.

```
X = train_df.drop(["Survived"],axis=1)
y = train_df["Survived"]
```

```
X = np.array(X)
y = np.array([y])
y = y.T
print('The dimensions of X and Y are:')
print(X.shape,y.shape)
```

e) As a programmer in modern times, one of the advantages is that generative AI tools such as ChatGPT can produce Python code in seconds. The following code was generated by Bing Copilot with only a few prompts. It uses a custom loss function for the general log-cosh function to perform robust logistic regression. Run the AI-generated LCLR code and try different values of the robustness factor. Also, compare the results with CELR using the cross-entropy loss.

```
#AI generated code for LCLR
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.001, \
    random_state=12)
# Define the custom loss function
def custom_logcosh_loss(y_true, y_pred):
    return (1/10) * tf.math.log \
        (tf.math.cosh(10 * (y_pred - y_true)))
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)
y_train = y_train.astype(np.float32)
#y_test = y_test.astype(np.float32)
# Define the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, \
        activation='sigmoid', \
        input_shape=(X_train.shape[1],))
])

# Compile the model with Cross-entropy loss
#model.compile(optimizer='adam', \
    #loss='binary_crossentropy', \
    #metrics=['accuracy'])
```

```
# Compile the model with LogCosh loss
model.compile(optimizer='adam', \
    loss=custom_logcosh_loss, \
    metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=200, \
    batch_size=32, \
    validation_data=(X_train, y_train))

# Evaluate the model
loss, accuracy = model.evaluate(X_train, y_train)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

Compare the overall accuracy of the two models with the training set.

12.2 (Advanced Project) The previous project made use of components from the `tensorflow` library to carry out logistic regression. While it is very easy to generate and run such tools, the process is somewhat opaque since the inner workings are not readily available to the user. In this project, we will implement the equivalent Python code that allows for further experimentation by the user, especially with regard to reproducing the results from this chapter.

a) Begin by obtaining the code for the Adam optimizer from Problem 10.6(b) of Chapter 10 and run it in your notebook.
b) Enter the following code to define the log-cosh and cross-entropy loss functions and their gradients.

```
import pandas as pd
logcosh = lambda x: np.logaddexp(x, -x) - np.log(2)
#Define sigmoid function
def sigmoid(z):
    a = 1./(1.+ np.exp(-z))
    return a
#Define log-cosh and cross-entropy loss functions
def Loss_function(w_0,b_0,LC):
    p = sigmoid(np.dot(X,w_0)+b_0)
    if(LC):
        loss = (1/a)*np.sum(logcosh(a*(y-p)))
    else:
        loss = -np.sum(y*np.log(p)+ \
            (1.-y)*np.log(1.-p))
    return loss
#Define log-cosh and cross-entropy gradient functions
def dLoss_function(w_0,b_0,LC):
        p = sigmoid(np.dot(X,w_0)+b_0)
        if(LC):
            v = p*(1.-p)*np.tanh(a*(y-p))
            v = v.T
            v_stacked = np.tile(v[:, np.newaxis], \
                (1, X.shape[1]))
            X_multiplied = -X * v_stacked
            dw = np.sum(X_multiplied, axis=0)
            db = np.sum(p*(1.-p)*np.tanh(a*(y-p))*(-1))
        else:
            v = y-p
            v_stacked = np.tile(v[:, np.newaxis], \
                (1, X.shape[1]))
            X_multiplied = -X * v_stacked
            dw = np.sum(X_multiplied, axis=0)
            db = np.sum((y-p)*(-1))
        return dw, db
```

c) Run the following code to perform LCLR and CELR. Initially, it is set to run LCLR on the Titanic training set. The code assumes a successful completion of the Problem 1 of this chapter.

```
X = train_df.drop(["Survived"],axis=1)
y = train_df["Survived"]
LC = True
CE = False
reg_type = LC #LC or CE
if(reg_type):
    print("Running LC logistic regression (LCLR)")
    adam = AdamOptim(learning_rate=0.005)
else:
    print("Running CE logistic regression (CELR)")
    adam = AdamOptim(learning_rate=0.005)
a = 10.0
w = np.array([-0.1,-0.1,-0.1,0.1,0.1,0.1,-0.1])
b = 0.0
t = 0
loss = []
print("Iteration =", t, "Loss =", \
    format(Loss_function(w,b,reg_type), '.3f'))
converged, iter_count  = False, 4500
while not converged:
    loss.append(Loss_function(w,b,reg_type))
    dw, db = dLoss_function(w,b,reg_type)
```

```
        w_old = w
        w, b = adam.update(t,w=w, b=b, dw=dw, db=db)

        t+=1
        if (t%1000 == 0):
            print("Iteration =", t, "Loss =", \
                format(Loss_function(w,b,reg_type), '.3f'))
        if (t > iter_count):
            converged = True
    print("Iteration =", t, "Loss =", \
        format(Loss_function(w,b,reg_type), '.3f'))
    print(w)
    print("intercept  ",format(b, '.6f'))
```

d) Plot of the loss function against iterations to confirm that convergence has occurred.

```
plt.plot(loss)
plt.xlabel('Iteration')
plt.ylabel('loss')
plt.show()
```

e) Predict the outcomes.

```
pT = 0.5
z = np.dot(X,w)+b
p = sigmoid(z)
pred = p >pT
```

f) Run the following code to obtain the metrics for accuracy, precision, recall, F_1 score, and AUROC.

```
from sklearn import  metrics
if (reg_type):
    print("Log-cosh metrics")
else:
    print("Cross-entropy metrics")
print("Confusion Matrix:")
table = metrics.confusion_matrix(pred,y)
print(table)
accuracy = np.mean(pred == y)
precision = metrics.precision_score(y,pred)
recall = metrics.recall_score(y,pred)
f1 = metrics.f1_score(y,pred)
auroc = metrics.roc_auc_score(y, pred)
print("accuracy =",format (accuracy, '.3f'))
print("precision =",format (precision, '.3f'))
print("recall =",format (recall, '.3f'))
print("F1 score =",format (f1, '.3f'))
print("AUROC score =",format (auroc, '.3f'))
```

g) Rerun the sequence for CELR. Compare the results with LCLR.
h) With the working code for both methods, follow the steps through the chapter to reproduce the results of Sections 12.3 and 12.4.1.

   Assuming that you have successfully completed the projects, you can proceed to work on robust data science using these techniques and/or solve new research problems in robust machine learning. Congratulations and Good luck!

References

Aggarwal, C.C. (2017). *Outlier Analysis*, 2e. Springer.
Cryer, J.D. and Chan, S.-K. (2008). *Time Series Analysis, with Applications in R*, 2e. Springer.

Index

Fundamentals of Robust Machine Learning: Handling Outliers and Anomalies in Data Science,
First Edition. Resve Saleh, Sohaib Majzoub and A. K. Md. Ehsanes Saleh.