

CHAPMAN & HALL/CRC DATA MINING AND KNOWLEDGE DISCOVERY SERIES

DATA SCIENCE AND ANALYTICS WITH PYTHON

SECOND EDITION



Jesús Rogel-Salazar

A Chapman & Hall Book



CRC Press
Taylor & Francis Group

Data Science and Analytics with Python

Since the first edition of *Data Science and Analytics with Python* we have witnessed an unprecedented explosion in the interest and development within the fields of Artificial Intelligence (AI) and Machine Learning (ML). This surge has led to the widespread adoption of the book, not just among business practitioners, but also by universities as a key textbook. In response to this growth, this new edition builds upon the success of its predecessor, expanding several sections, updating the code to reflect the latest advancements in Python libraries and modules, and addressing the ever-evolving landscape of Generative AI (Gen AI).

This updated edition ensures that the examples and exercises remain relevant by incorporating the latest features of popular libraries such as Scikit-learn, Pandas, and NumPy. Additionally, new sections delve into cutting-edge topics like Gen AI, reflecting the advancements and the expanding role these technologies play. This edition also addresses crucial issues of explainability, transparency and fairness in AI. These topics have rightly gained significant attention in recent years. As AI integrates more deeply into various aspects of our lives, understanding and mitigating biases, ensuring fairness and maintaining transparency become paramount. This book provides comprehensive coverage of these topics, offering practical insights and guidance for data scientists and analysts.

Designed as a practical companion for data analysts and budding data scientists, this book assumes a working knowledge of programming and statistical modelling but aims to guide readers deeper into the wonders of data analytics and ML. Maintaining the book's structure, each chapter stands alone as much as possible, allowing readers to use it as a reference as well as a textbook. Whether revisiting fundamental concepts or diving into new, advanced topics, this book offers something valuable for every reader.

Chapman & Hall/CRC

Data Mining and Knowledge Series

Series Editor: Vipin Kumar

Data Science and Machine Learning for Non-Programmers

Using SAS Enterprise Miner

Dothang Truong

Healthcare Data Analytics

Chandan K. Reddy and Charu C. Aggarwal

Accelerating Discovery

Mining Unstructured Information for Hypothesis Generation

Scott Spangler

Event Mining

Algorithms and Applications

Tao Li

Text Mining and Visualization

Case Studies Using Open-Source Tools

Markus Hofmann and Andrew Chisholm

Graph-Based Social Media Analysis

Ioannis Pitas

Data Mining

A Tutorial-Based Primer, Second Edition

Richard J. Roiger

Data Mining with R

Learning with Case Studies, Second Edition

Luis Torgo

Social Networks with Rich Edge Semantics

Quan Zheng and David Skillicorn

Large-Scale Machine Learning in the Earth Sciences

Ashok N. Srivastava, Ramakrishna Nemani, and Karsten Steinhäuser

Data Science and Analytics with Python

Jesús Rogel-Salazar

Feature Engineering for Machine Learning and Data Analytics

Guozhu Dong and Huan Liu

Exploratory Data Analysis Using R

Ronald K. Pearson

Human Capital Systems, Analytics, and Data Mining

Robert C. Hughes

Industrial Applications of Machine Learning

Pedro Larrañaga et al

Automated Data Analysis Using Excel

Second Edition

Brian D. Bissett

Knowledge Guided Machine Learning: Accelerating Discovery using Scientific Knowledge and Data

Edited by Anuj Karpatne, Ramakrishnan Kannan, Vipin Kumar

For more information about this series please visit:

<https://www.routledge.com/Chapman--HallCRC-Data-Mining-and-Knowledge-Discovery-Series/book-series/CHDAMINODIS>

Data Science and Analytics with Python

Second Edition

Jesús Rogel-Salazar



CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Designed cover image: © Jesús Rogel-Salazar

MATLAB® and Simulink® are trademarks of The MathWorks, Inc. and are used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB® or Simulink® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® and Simulink® software.

Second edition published 2025

by CRC Press

2385 NW Executive Center Drive, Suite 320, Boca Raton FL 33431

and by CRC Press

4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

© 2025 Jesús Rogel-Salazar

Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, access www.copyright.com or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. For works that are not available on CCC please contact mpkbookspermissions@tandf.co.uk

Trademark notice: Product or corporate names may be trademarks or registered trademarks and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Names: Rogel-Salazar, Jesus, author.

Title: Data science, machine learning and analytics with Python / Jesús Rogel-Salazar.

Description: Second edition. | Boca Raton, FL : Taylor & Francis, 2025. |

Series: Chapman & Hall/CRC Press the python series | Includes bibliographical references and index. |

Identifiers: LCCN 2024040350 (print) | LCCN 2024040351 (ebook) | ISBN 9781032772523 (hardback) | ISBN 9781032772493 (paperback) | ISBN 9781003482062 (ebook)

Subjects: LCSH: Mathematical statistics--Data processing. | Python (Computer program language) | Information visualization.

Classification: LCC QA276.45.P98 R64 2025 (print) | LCC QA276.45.P98 (ebook) | DDC 519.50285/5133--dc23/eng/20241226

LC record available at <https://lcn.loc.gov/2024040350>

LC ebook record available at <https://lcn.loc.gov/2024040351>

ISBN: 978-1-032-77252-3 (hbk)

ISBN: 978-1-032-77249-3 (pbk)

ISBN: 978-1-003-48206-2 (ebk)

DOI: [10.1201/9781003482062](https://doi.org/10.1201/9781003482062)

Typeset in URWPalladioL-Roman
by KnowledgeWorks Global Ltd.

Publisher's note: This book has been prepared from camera-ready copy provided by the authors.

To Antony Johnson & Prof. Bowman

*Thanks to Alan M. Turing for
opening up my mind*



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Contents

List of Figures xv

List of Tables xxiii

Preface to the Second Edition xxv

Preface xxix

Reader's Guide xxxv

About the Author xxxix

1	<i>Trials and Tribulations of a Data Scientist</i>	1
1.1	<i>Data? Science? Data Science!</i>	2
1.1.1	<i>So, What Is Data Science?</i>	3
1.2	<i>The Data Scientist: A Modern Jackalope</i>	9
1.2.1	<i>Characteristics of a Data Scientist and a Data Science Team</i>	15
1.3	<i>Data Science Tools</i>	21
1.3.1	<i>Open-Source Tools</i>	25

1.4	<i>From Data to Insight: the Data Science Workflow</i>	27
1.4.1	<i>Identify the Question</i>	29
1.4.2	<i>Acquire Data</i>	30
1.4.3	<i>Data Munging</i>	30
1.4.4	<i>Modelling and Evaluation</i>	31
1.4.5	<i>Representation and Interaction</i>	32
1.4.6	<i>Data Science: an Iterative Process</i>	32
1.5	<i>Summary</i>	34
2	<i>Python: For Something Completely Different</i>	37
2.1	<i>Why Python? Why not?!</i>	39
2.1.1	<i>To Shell or Not To Shell</i>	42
2.1.2	<i>Jupyter Notebook</i>	44
2.2	<i>Firsts Slithers with Python</i>	45
2.2.1	<i>Basic Types</i>	46
2.2.2	<i>Numbers</i>	46
2.2.3	<i>Strings</i>	47
2.2.4	<i>Complex Numbers</i>	49
2.2.5	<i>Lists</i>	50
2.2.6	<i>Tuples</i>	55
2.2.7	<i>Dictionaries</i>	58

2.3	<i>Control Flow</i>	60
2.3.1	<i>Conditional Statements — if... elif... else</i>	61
2.3.2	<i>while Loop</i>	62
2.3.3	<i>for Loop</i>	63
2.3.4	<i>Exception handling — try... except</i>	64
2.3.5	<i>Functions</i>	67
2.3.6	<i>Scripts, Modules and the Standard Library</i>	72
2.4	<i>Computation and Data Manipulation</i>	76
2.4.1	<i>Matrix Manipulations and Linear Algebra</i>	76
2.4.2	<i>NumPy Arrays and Matrices</i>	78
2.4.3	<i>Indexing and Slicing</i>	82
2.5	<i>Pandas to the Rescue</i>	83
2.6	<i>Plotting and Visualising: Matplotlib</i>	90
2.7	<i>Virtual Environments</i>	92
2.8	<i>Summary</i>	99

3 *The Machine That Goes “Ping”: Machine Learning and Pattern Recognition* 101

3.1	<i>Recognising Patterns</i>	101
3.2	<i>Artificial Intelligence and Machine Learning</i>	104
3.2.1	<i>Challenges and Future Trends</i>	106
3.2.2	<i>Unleashing the Power: Generative Artificial Intelligence</i>	108
3.2.3	<i>The Art of Prompt Engineering</i>	118

3.3	<i>Narrowing it Down: Machine Learning</i>	123
3.4	<i>Data Is Good, but other Things Are also Needed</i>	124
3.5	<i>Learning, Predicting and Classifying</i>	127
3.6	<i>Machine Learning and Data Science</i>	131
3.7	<i>Feature Selection</i>	133
3.8	<i>Bias, Variance and Regularisation: A Balancing Act</i>	135
3.9	<i>Some Useful Measures: Distance and Similarity</i>	138
3.10	<i>Beware the Curse of Dimensionality</i>	143
3.11	<i>Scikit-Learn is our Friend</i>	149
3.12	<i>Training and Testing</i>	152
3.13	<i>Cross-Validation</i>	157
3.13.1	<i>k-fold Cross-Validation</i>	158
3.14	<i>Summary</i>	162
4	<i>The Relationship Conundrum: Regression</i>	165
4.1	<i>Relationships between Variables: Regression</i>	165
4.2	<i>Multivariate Linear Regression</i>	170
4.3	<i>Ordinary Least Squares</i>	172
4.3.1	<i>The Maths Way</i>	173
4.4	<i>Brain and Body: Regression with One Variable</i>	178
4.4.1	<i>Regression with Scikit-learn</i>	187

4.5	<i>Logarithmic Transformation</i>	189
4.6	<i>Making the Task Easier: Standardisation and Scaling</i>	194
4.6.1	<i>Normalisation or Unit Scaling</i>	195
4.6.2	<i>z-Score Scaling</i>	197
4.7	<i>Polynomial Regression</i>	198
4.7.1	<i>Multivariate Regression</i>	203
4.8	<i>Variance-Bias Trade-Off</i>	204
4.9	<i>Shrinkage: LASSO and Ridge</i>	207
4.10	<i>One of a Kind: Regression with Categorical Features</i>	214
4.11	<i>Summary</i>	223
5	<i>Jackalopes and Hares: Clustering</i>	225
5.1	<i>Clustering</i>	226
5.2	<i>Clustering with k-means</i>	227
5.2.1	<i>Cluster Validation</i>	231
5.2.2	<i>k-means in Action</i>	234
5.3	<i>Optimising the Number of Clusters: The Elbow Method</i>	238
5.4	<i>Density's Delight: DBSCAN Clustering</i>	241
5.4.1	<i>Finding Friendly Neighbours: Nearest Neighbours</i>	244
5.4.2	<i>DBSCAN in Action</i>	247
5.5	<i>Summary</i>	252

6	<i>Unicorns and Horses: Classification</i>	255
6.1	<i>Classification</i>	256
6.1.1	<i>Confusion Matrices</i>	258
6.1.2	<i>ROC and AUC</i>	262
6.2	<i>Classification with KNN</i>	265
6.2.1	<i>KNN in Action</i>	267
6.3	<i>Classification with Logistic Regression</i>	272
6.3.1	<i>Logistic Regression Interpretation</i>	276
6.3.2	<i>Logistic Regression in Action</i>	278
6.4	<i>Classification with Naïve Bayes</i>	287
6.4.1	<i>Naïve Bayes Classifier</i>	293
6.4.2	<i>Naïve Bayes in Action</i>	294
6.5	<i>Being Zen with Data: Dealing with Imbalanced Classes</i>	300
6.5.1	<i>Resampling: Oversampling or Undersampling</i>	304
6.5.2	<i>Synthetic Minority Oversampling Technique (SMOTE)</i>	309
6.6	<i>Summary</i>	311
7	<i>Decisions, Decisions: Hierarchical Clustering, Decision Trees and Ensemble Techniques</i>	313
7.1	<i>Hierarchical Clustering</i>	314
7.1.1	<i>Hierarchical Clustering in Action</i>	317
7.2	<i>Decision Trees</i>	321
7.2.1	<i>Decision Trees in Action</i>	328

7.3	<i>Ensemble Techniques</i>	337
7.3.1	<i>Bagging</i>	344
7.3.2	<i>Boosting</i>	345
7.3.3	<i>Random Forests</i>	346
7.3.4	<i>Stacking and Blending</i>	348
7.4	<i>Ensemble Techniques in Action</i>	349
7.5	<i>Explain Yourself: Explainable AI</i>	355
7.5.1	<i>SHAP — SHapley Additive exPlanations</i>	356
7.5.2	<i>SHAP in Action</i>	357
7.6	<i>Summary</i>	361
8	<i>Less is More: Dimensionality Reduction</i>	365
8.1	<i>Dimensionality Reduction</i>	366
8.2	<i>Principal Component Analysis</i>	371
8.2.1	<i>PCA in Action</i>	375
8.2.2	<i>PCA in the Iris Dataset</i>	380
8.3	<i>Singular Value Decomposition</i>	384
8.3.1	<i>SVD in Action</i>	386
8.4	<i>Recommendation Systems</i>	390
8.4.1	<i>Content-Based Filtering in Action</i>	392
8.4.2	<i>Collaborative Filtering in Action</i>	396
8.5	<i>Dealing Effectively with Dimensionality Reduction</i>	403
8.6	<i>Summary</i>	408

9	<i>Kernel Tricks up the Sleeve: Support Vector Machines</i>	411
9.1	<i>Support Vector Machines and Kernel Methods</i>	412
9.1.1	<i>Support Vector Machines</i>	415
9.1.2	<i>The Kernel Trick</i>	424
9.1.3	<i>SVM in Action: Regression</i>	427
9.1.4	<i>SVM in Action: Classification</i>	431
9.1.5	<i>Support Vector Machines — Use Cases</i>	437
9.2	<i>Transparency, Fairness and Beyond</i>	441
9.2.1	<i>Getting Support — Generative AI</i>	444
9.3	<i>Summary</i>	447
	 <i>Pipelines in Scikit-Learn</i>	 451
	 <i>Bibliography</i>	 457
	 <i>Index</i>	 467

List of Figures

- 1.1 Simplified diagram of the skills needed in data science and their relationship. 10
- 1.2 Jackalopes are mythical animals resembling a jackrabbit with antlers. 12
- 1.3 The various steps involved in the data science workflow. 28
- 2.1 A plot generated by Matplotlib. 92
- 3.1 Measuring the distance between points A and B. 139
- 3.2 The curse of dimensionality. Ten data instances placed in spaces of increased dimensionality, from one to three. Sparsity increases with the number of dimensions. 145
- 3.3 Volume of a hypersphere as a function of the dimensionality N . As the number of dimensions increases, the volume of the hypersphere tends to zero. 148
- 3.4 A dataset is split into training and testing sets. The training set is used in the modelling phase and the testing set is held for validating the model. 155

- 3.5 For $k = 4$, we split the original dataset into four and use each of the partitions in turn as the testing set. The result of each fold is aggregated (averaged) in the final stage. 159
- 4.1 The regression procedure for a very well-behaved dataset where all data points are perfectly aligned. The residuals in this case are all zero. 176
- 4.2 The regression procedure for a very well-behaved dataset where all data points are perfectly aligned. The residuals in this case are all zero. 177
- 4.3 Scatter plot of the brain (gr) versus body mass (kg) for various mammals. 179
- 4.4 Scatter plot and the regression line calculated for the brain (gr) versus body mass (kg) for various mammals. 186
- 4.5 Scatter plot in a log-log scale for the brain (gr) versus body mass (kg) for various mammals. 190
- 4.6 Log-log scale figure with a scatter plot and the regression line calculated for the brain (gr) versus body mass (kg) for various mammals. 192
- 4.7 Comparison of the simple linear regression model and the model with logarithmic transformation for the brain (gr) versus body mass (kg) for various mammals. 193
- 4.8 Comparison of a quadratic model, a simple linear regression model and a model with logarithmic transformation fitted to the brain (gr) versus body mass (kg) for various mammals. 201

- 4.9 Using GridSearchCV, we can scan a set of parameters to be used in conjunction with cross-validation. In this case we show the values of λ used to fit ridge and LASSO models, together with the mean scores obtained during modelling. 212
- 4.10 Linear regression using a categorical feature. The interpretation of the regression line is different from those obtained with continuous features. 219
- 4.11 Linear regression using a categorical feature with three levels. 222

- 5.1 The plots show the exact same dataset but in different scales. The panel on the left shows two potential clusters, whereas in the panel on the right the data may be grouped into one. 229
- 5.2 Diagrammatic representation of cluster cohesion and separation. 232
- 5.3 k -means clustering of the Wine dataset based on Alcohol and Colour Intensity. The shaded areas correspond to the clusters obtained. The stars indicate the position of the final centroids. 236
- 5.4 k -means clustering with the elbow method. The plot enables us to find a suitable number of clusters in our data. 240
- 5.5 Using the elbow method to determine the optimal value of the ϵ parameter to be used in our DBSCAN analysis. 249
- 5.6 DBSCAN clustering of the Wholesale customer dataset based on Detergent_Paper and Milk. 251

- 6.1 ROC for our hypothetical aircraft detector. We contrast this with the result of a random detector given by the dashed line, and a perfect detector shown with the thick solid line. 264
- 6.2 Accuracy scores for the KNN classification of the Iris dataset with different values of k . We can see that the best parameter found is 11 neighbours. 269
- 6.3 KNN classification of the Iris dataset based on sepal width and petal length for $k = 11$. The shaded areas correspond to the classification mapping obtained by the algorithm. We can see some misclassifications in the upper-right-hand corner of the plot. 270
- 6.4 Plot of the logistic function $g(z) = \frac{e^z}{1+e^z}$. 273
- 6.5 Heatmap of mean cross-validation scores for the Logistic Regression classification of the Wisconsin Breast Cancer dataset for different values of C with $L1$ and $L2$ penalties. 283
- 6.6 ROC curves obtained by cross-validation with $k = 3$ on the Wisconsin Breast Cancer dataset. 286
- 6.7 Venn diagrams to visualise Bayes' theorem. 289
- 7.1 A dendrogram is a tree-like structure that enables us to visualise the clusters obtained with hierarchical clustering. The height of the clades or branches tells us how similar the clusters are. 315
- 7.2 Dendrogram generated by applying hierarchical clustering to the Iris dataset. We can see how three clusters can be determined from the dendrogram by cutting at an appropriate distance. 319

- 7.3 Simple decision tree built with information from Table 7.1. 323
- 7.4 Comparison of impurity measures we can use for a binary classification problem. 326
- 7.5 Heatmap of mean cross-validation scores for the decision tree classification of the Titanic passengers for different values of maximum depth and minimum sample leaf. 334
- 7.6 Decision tree for the Titanic passengers dataset. 336
- 7.7 Decision boundaries provided by a) a single decision tree, and b) several decision trees. The combination of the boundaries in b) can provide a better approximation to the true diagonal boundary. 340
- 7.8 Diagrammatic view of the idea of constructing an ensemble classifier. 341
- 7.9 ROC curves and their corresponding AUC scores for various ensemble techniques applied to the Titanic training dataset. 354
- 7.10 Mean SHAP values providing us with a view of the average impact of each feature on the model. We can see how the gender of the passenger has a greater impact on the decision, compared to the travelling class and the age. 358
- 7.11 Impact on the outcome of the model. Each dot represents a record in the dataset.. 359
- 8.1 Simple illustration of data dimensionality reduction. Extracting features $\{u_1, u_2\}$ from the original set $\{x_1, x_2\}$ enables us to represent our data more efficiently. 370

- 8.2 Diagrammatic scree plot showing the eigenvalues corresponding to each of six different principal components. 374
- 8.3 A jackalope silhouette to be used for image processing. 376
- 8.4 Principal component analysis applied to the jackalope image shown in Figure 8.3. We can see how retaining more principal components increases the resolution of the image. 378
- 8.5 Scree plot of the explained variance ratio (for 10 components) obtained by applying principal component analysis to the jackalope image shown in Figure 8.3. 379
- 8.6 Scree plot of the explained variance ratio obtained by applying principal component analysis to the four features in the Iris dataset. 381
- 8.7 Illustration of the singular value decomposition. 385
- 8.8 Image of a letter *J* (on the left) and its column components (on the right). 387
- 8.9 Singular values obtained from applying SVD in a an image of a letter *J* constructed in Python. 388
- 8.10 Reconstruction of the original noisy letter *J* (left-most panel), using one to four singular values obtained from SVD. 389

- 9.1 The dataset shown in panel a) is linearly separable in the $X_1 - X_2$ feature space, whereas the one in panel b) is not. 413
- 9.2 Linearly separable dataset may have a large number of separation boundaries. Which one is the best? 415

- 9.3 A support vector machine finds the optimal boundary by determining the maximum margin hyperplane. The weight vector w determines the orientation of the boundary and the support vectors (marked in solid black) define the maximum margin. 417
- 9.4 Comparison of the regression curves obtained using a linear model, and two SVM algorithms: one with a linear kernel and the other one with a Gaussian one. 430
- 9.5 Heatmap of the mean cross-validation scores for the support vector machine algorithm with a Gaussian kernel for different values of the parameter C . 434
- 9.6 Comparison of the classification boundaries obtained using support vector machine algorithms with different implementations: SVC with a linear, Gaussian and degree-3 polynomial kernels, and LinearSVC. 436



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

List of Tables

2.1	Arithmetic operators in Python.	46
2.2	Comparison operators in Python.	62
2.3	Standard exceptions in Python.	66
2.4	Some built-in functions in Python.	68
2.5	Sample tabular data to be loaded into a pandas dataframe.	84
2.6	Some of the input sources available to pandas.	89
3.1	The art of prompt engineering.	119
3.2	Machine learning algorithms can be classified by the type of learning and outcome of the algorithm.	130
4.1	Results from the regression analysis performed on the brain and body dataset.	183
4.2	Results from the regression analysis performed on the brain and body dataset using a log-log transformation.	193
6.1	Confusion matrix for an elementary binary classification system to distinguish enemy aircraft from flocks of birds.	259

- 6.2 Diagrammatic confusion matrix indicating the location of True Positives, False Negatives, False Positives and True Negatives. 260
- 6.3 Readings of the sensitivity, specificity and fallout for a thought experiment in a radar receiver to distinguish enemy aircraft from flocks of birds. 263
- 7.1 Dietary habits and number of limbs for some animals. 322
- 7.2 Predicted classes of three hypothetical binary base classifiers and the ensemble generated by majority voting. 342
- 7.3 Predicted classes of three hypothetical binary base classifiers with high correlation in their predictions. 343
- 7.4 Predicted classes of three hypothetical binary base classifiers with low correlation in their predictions. 343
- 8.1 Films considered in building a content-based filtering recommendation system. 393
- 8.2 Scores provided by users regarding the three features used to describe the films in our database. 393
- 8.3 Utility matrix of users v books used for collaborative filtering. We need to estimate the scores marked with question marks. 398

Preface to the Second Edition

IT HAS BEEN SEVEN YEARS since the first edition of this book hit the shelves. Back then, Artificial Intelligence (AI) and Machine Learning (ML) were promising fields, gradually gaining prominence. Little did we anticipate the explosion in interest and development that has since occurred. The book has been warmly received, not only by business practitioners but also by universities adopting it as a textbook. This new edition aims to build upon that success, expanding several sections, updating code to reflect the latest advancements in Python libraries and modules, and addressing the ever-growing landscape of Generative AI (Gen AI).

The world of data science has evolved rapidly, and so have the tools we use. The updated edition reflects these changes, ensuring that the examples and exercises remain relevant. I have incorporated the latest features of popular libraries like Scikit-learn, pandas, and NumPy, and introduced new sections on cutting-edge topics such as Generative AI. For completeness, here are the versions used for this revision of the book.

The explosion in AI and ML has led to the book's widespread adoption prompting this new edition.

The world of data science has evolved rapidly, and so have the tools we use.

- Python - 3.10.14
- Pandas - 2.2.2
- NumPy - 1.26.4
- Scikit-learn - 1.5.1
- StatsModels - 0.14.2
- SHAP - 0.45.1

As you can see, pandas and Scikit-learn were not even in version 1.0 when the first edition came out. Today, these are robust, well-known packages that any Jackalope Data Scientist should be familiar with.

Additionally, this edition addresses the crucial issues of explainability, transparency, and fairness in AI. These topics have gained significant attention in recent years, and rightly so. As we integrate AI into more aspects of our lives, understanding and mitigating biases, ensuring fairness, and maintaining transparency are of utmost importance.

As before, the book is designed to be a practical companion for data analysts and budding data scientists. It assumes a working knowledge of programming and statistical modelling but aims to guide readers deeper into the wonders of data analytics and machine learning. Python remains our tool of choice, thanks to its versatility and the robust ecosystem that has flourished around it. This edition continues to leverage the power of the Jupyter Notebook, providing a flexible, interactive environment for exploring data science concepts.

Some packages were not even in version 1.0 when the first edition of the book came out.

This edition addresses the crucial issues of explainability, transparency, and fairness in AI.

Python remains our tool of choice, thanks to its versatility.

I have made sure to maintain the book's structure, allowing each chapter to stand alone as much as possible, so readers can use it as a reference as well as a textbook. Whether you are revisiting fundamental concepts or diving into new, advanced topics, this book has something for you.

As before, each chapter stands alone as much as possible.

Since the first edition, the feedback from readers has been invaluable. The discussions, debates, and dialogues with colleagues across various industries have greatly influenced the updates in this edition. I remain grateful for the insights and suggestions from the community.

Your feedback has been invaluable, thanks!

I owe a debt of gratitude to my colleagues in academia and industry for their continuous input and suggestions. A special thanks to my editor, Randi Slack, and the technical reviewers for their insightful comments. And, of course, I am grateful for my family and friends, whose encouragement and support have been invaluable throughout this journey. Finally, thanks to you, the reader; I hope you find something useful in the following pages.

London, UK

Dr Jesús Rogel-Salazar

January 2025



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Preface

THIS BOOK IS THE RESULT of very interesting discussions, debates and dialogues with a large number of people at various levels of seniority, working at startups as well as long-established businesses, and in a variety of industries, from science to media to finance. The book is intended to be a companion to data analysts and budding data scientists who have some working experience with both programming and statistical modelling, but who have not necessarily delved into the wonders of data analytics and machine learning. The book uses Python¹ as a tool to implement and exploit some of the most common algorithms used in data science and data analytics today.

It is fair to say that there are a number of very useful tools and platforms available to the interested reader such as the excellent open-source R project² or proprietary ones like SPSS[®] or SAS[®]. They are all highly recommended, and they have their strengths (and weaknesses). However, given the experience I have been lucky to have had in implementing and explaining algorithms, I find Python to be a very malleable tool. This reminds me of a conversation with an

¹ Python Software Foundation (1995). Python reference manual. <http://www.python.org>

² R Core Team (2014). R: A language and environment for statistical computing. <http://www.R-project.org>

experienced analyst at a big consultancy firm who mentioned that doing any machine learning or data science related task in Python was impossible. I politely disagreed. It is true though that there may be more suitable tools for certain tasks, but it would be a truly Herculean labour to present them all in one single volume. With that in mind, the choice of using Python throughout this book suggested itself: Python is a popular and versatile scripting and object-oriented language; it is easy to use and has a large active community of developers and enthusiasts, not to mention the richness of the iPython/Jupyter Notebook, as well as the fact that it has been used by both business and academia for some time now.

The main purpose of the book is to present the reader with some of the main concepts used in data science and analytics using tools developed in Python such as Scikit-learn³, pandas⁴, NumPy⁵ and others. The book is intended to be a bridge to the data science and analytics world for programmers and developers, as well as graduates in scientific areas such as mathematics, physics, computational biology and engineering, to name a few. In my experience, the background and skills acquired by the readers I have in mind are a great asset to have. However, in many cases the bigger picture is somewhat blurred due to the sharp specialisms required in their day-to-day activities. This book thus serves as a guide to exploit those skills in the data science and analytics arena. The book focusses on showing the concepts and ideas behind popular algorithms and their use, but it does not get into the details of their

We shall show in this book that doing machine learning or data science with Python is indeed possible.

iPython/Jupyter Notebook is a flexible web-based computational environment that combines code, text, mathematics and plots in a single document. Visit <http://ipython.org/notebook.html>

³ Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*. 12, 2825–2830.

⁴ McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media.

⁵ Scientific Computing Tools for Python (2013). NumPy. <http://www.numpy.org>

implementation in Python. It does, however, use open-source implementations of those algorithms.

The examples contained in this volume have been tested in Python 3.5 under MacOS, Linux and Windows 7, and the code can be run with minimal changes in a Python 2 distribution. For reference, the versions of some of the packages used in the book are as follows:

- Python - 3.5.2
- Pandas - 0.19.1
- NumPy - 1.11.2
- Scikit-learn - 0.18
- StatsModels - 0.6.1

In particular I have chosen to use the Anaconda Python distribution⁶ provided by Continuum Analytics, as it offers installations in all three computer systems mentioned above, plus having the advantage of offering a rich ecosystem of libraries readily available directly from the distribution itself; and most importantly it is available to all. There are a few other ways of obtaining Python as well as other versions of the software: For instance directly from the Python Software Foundation, as well as distributions from Enthought Canopy, or from package managers such as Homebrew. Anaconda offers an easy environment to install and maintain the software, with minimum hassle for the user. I assume that the reader is working with the computer via scripts as well as interactively in a shell.

⁶ Anaconda (2024). Anaconda 3.0. <https://www.anaconda.com/download/>

Python Software Foundation
<https://www.python.org>

Enthought Canopy <https://www.enthought.com/products/epd/>

Homebrew <http://brew.sh>

The book shows the use of computer code by enclosing it in a box as follows:

```
> 1 + 1 # Example of computer code  
  
2
```

We have made use of a dipole (>) to denote the command line terminal prompt shown in the Python shell. Please note that the same commands can be used in the iPython interactive shell or iPython/Jupyter Notebook, although the look and feel may be quite different. As you may have already noticed, the book uses margin notes, such as the one that appears to the right of this paragraph, to highlight certain areas or commands, as well as to provide some useful comments.

This is an example of the margin notes used throughout this book.

The book is organised in a way that individual chapters are sufficiently independent from each other, so that the reader is comfortable using the contents as a reference rather than a textbook. Inevitably, there will be occasions where certain topics make reference to other parts of the book, and I will point out when that may be the case. I would also like to take this opportunity to mention that the implementations presented are by no means the only or best way to do things. Programming is pretty similar to the creative process of writing: The fact that you have a set of words does not imply that we all write reports in a poetic manner. I would be delighted to hear from you all about the implementations and changes you make to the code presented here. Do get in touch!

Programming is a creative process, and as such there is more than one way to do things.

We start in [Chapter 1](#) with a discussion of what data science and analytics are, from the point of view of the process and results obtained. We pay particular attention to the data exploration process as well as the data munging that needs to be carried out prior to the application of algorithms and analysis.

The data science workflow is discussed on [Chapter 1](#).

In [Chapter 2](#) we take the opportunity to remind us of some important features of the Python language. The aim is to revisit some important commands and instructions that provide the base for the rest of the book. This will also give us the opportunity to revise some commands and instructions used in later chapters.

A Python primer is given in [Chapter 2](#).

In [Chapter 3](#) we cover basic elements of machine learning, pattern recognition and artificial intelligence that underpin the algorithms and implementations we will use in the rest of the book.

[Chapter 3](#) covers the basics of machine learning, pattern recognition and artificial intelligence.

By the time [Chapter 4](#) is reached, we will have the necessary foundations to implement regression analysis using Python via both StatsModels and Scikit-learn. The main points in the usage of generalised linear models for regression are covered in this chapter.

[Chapter 4](#) covers various regression algorithms

In [Chapter 5](#) we talk about clustering techniques, whereas [Chapter 6](#) covers classification algorithms. These two chapters are central to the data science workflow: Clustering enables us to assign labels to our data in an unsupervised manner; in turn we can use these labels as targets in a classification algorithm.

[Chapters 5](#) and [6](#) cover clustering and classification techniques, respectively.

In [Chapter 7](#) we introduce the use of hierarchical clustering, decision trees and talk about ensemble techniques such as bagging and boosting. It is worth pointing out that ensemble techniques have become a common tool among data scientists, and you are highly recommended to check this section out.

Dimensionality reduction techniques are discussed in [Chapter 8](#). There we will cover algorithms such as principal component analysis and singular value decomposition. As an application we will talk about recommendation systems.

Last but not least, in [Chapter 9](#) we will cover the support vector machine algorithm and the all-important Kernel trick in applications such as regression and classification.

The book was made possible, as I mentioned before, thanks to discussions, presentations and exchanges with colleagues both in academia as well as in business. I am very grateful for their input and suggestions. I would also like to thank my editor at CRC Press, Randi Cohen, as well as the technical reviewers for their comments and suggestions. Finally, the encouragement that my family and friends have given me to take up yet another writing project has been invaluable. This goes to you all!

London, UK

Dr Jesús Rogel-Salazar

February 2017

[Chapter 7](#) deals with hierarchical clustering decision trees and ensemble techniques.

[Chapter 8](#) talks about dimensionality reduction.

[Chapter 9](#) deals with support vector machines.

Reader's Guide

THIS BOOK IS INTENDED TO be a companion to any Jackalope Data Scientist from beginners to seasoned practitioners. The material covered here has been developed in the course of my interactions with colleagues and students and is presented in a systematic way that builds upon previous material presented.

Read [Chapter 1](#) to understand the Jackalope reference.

I highly recommend reading the book in a linear manner. However, I realise that different readers may have different needs; therefore, here is a guide that may help in reading and/or consulting this book:

- Managers and readers curious about Data Science:
 - Start by reading [Chapter 1](#) where you will learn what Data Science is all about
 - Follow that by reading [Chapter 3](#) where an introduction to machine learning awaits you
 - Make sure you understand those two chapters inside-out; they will help you to understand your Jackalope Data Scientists

- In particular cover the new entries in [Section 3.2](#) which address some areas in terms of trends and challenges as well as an introduction to Generative AI and prompt engineering
- Beginners:
 - If you do not have a background in programming, start with [Chapter 2](#), where a swift introduction to Python is presented
 - Follow that by reading [Chapter 1](#) and [Chapter 3](#) to understand more about what Data Science is and the principles of machine learning.
- Readers familiar with Python:
 - You can safely skip [Chapter 2](#) and go directly to [Chapter 4](#)
- Seasoned readers may find it easier to navigate the book by themes or subjects
 - **Regression** is covered in [Chapter 4](#), including:
 - * Ordinary least squares
 - * Multivariate regression
 - * LASSO and Ridge regression
 - * Support vector machines for regression are covered in [Section 9.1.3](#)
 - **Clustering**:
 - * K-means is covered in [Chapter 5](#)
 - * Hierarchical Clustering is covered in [Section 7.1](#)

- **Classification** is generally covered in [Chapter 6](#) including:
 - * KNN
 - * Logistic regression
 - * Naïve Bayes
 - * Support vector machines for classification are covered in [Section 9.1.4](#)
- **Decision Trees and Ensemble Techniques** are discussed in [Chapter 7](#)
- **Recommendation Systems** are introduced in [Section 8.4](#)
- **Text Manipulation** examples are provided in [Section 6.4.2](#) where tweets are used as the main data source. We also discuss some aspects in [Section 9.2](#)
- **Image Manipulation** examples are provided in [Sections 8.2.1](#) and [8.3.1](#)
- **Generative Artificial Intelligence** is explained in [Section 3.2.2](#), prompt engineering in [Section 3.2.3](#) and further topics are discussed in [Section 8.3.1](#)
- **Explainability and Ethical Aspects** are covered in [Sections 3.2.1, 3.2.2, 7.5, 8.5](#) and [9.2](#)



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

About the Author

DR JESÚS ROGEL-SALAZAR IS a leading technology expert with experience in the field working for companies such as The Ortus Group, TympaHealth, Barclays Bank, AKQA, IBM Data Science Studio, Dow Jones and others. He is a visiting researcher at the Department of Physics, and the Business School at Imperial College London, UK and a member of the School of Physics, Astronomy & Mathematics at the University of Hertfordshire, UK. He obtained his doctorate in Physics at Imperial College London for work on quantum atom optics and ultra-cold matter.

He has held a position as senior lecturer in mathematics as well as a consultant and data scientist in the financial industry since 2006. He is the author of the book *Essential MATLAB and Octave* and the companion book to this volume *Advanced Data Science and Analytics with Python* and *Statistics and Data Visualisation with Python*, also published by CRC Press. His interests include mathematical modelling, data science and optimisation in a wide range of applications including optics, quantum mechanics, data journalism, telematics, fintech and healthtech.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

1

Trials and Tribulations of a Data Scientist

THE EVER-INCREASING AVAILABILITY OF data requires the use of tools that enable businesses and researchers to draw conclusions and make decisions based on the evidence provided by the data itself. From performing a regression analysis to determining the relationship between data features, or improving on recommendation systems used in e-commerce, data science and analytics are used every day by all of us. This book is intended to provide those interested in data science and analytics a perspective into the subject matter using Python, a popular programming language available for various platforms and widely used both in business and academia.

Data science and analytics is used every day by all of us.

Python will be used throughout the book; get well acquainted with it!

In this chapter we will cover what data science is and how it is related to various disciplines from mathematics to business intelligence, machine learning and artificial intelligence, and from programming to design. We will discuss the characteristics that make a good data scientist and the composition of a data science team. We will also

provide an overview of the typical workflow in a data science and analytics project and shall see the trials and tribulations in the work cycle of a data scientist, or a practitioner of machine learning and artificial intelligence techniques.

1.1 *Data? Science? Data Science!*

THE USE OF DATA AS evidence in support for decision-making is nothing new. You only have to take a look at the original meaning of the word *statistics* as the analysis and interpretation of information relating to states such as economic and demographic data. Nowadays, the word statistics is either understood as a branch of mathematics that deals with the collection, analysis, interpretation and presentation of data¹; or more colloquially as a fact or figure obtained from a study based on large quantities of data. Simply take a look at the news on any given day and you will surely get to hear about statistics, proportions and percentages, all in support (or not) of a new initiative, plan or recommendation. The power of data is all around us and we use it all the time.

Now, what about the word “science”? Well, you may remember from your school days that science is a system that enables the organisation of knowledge, based on testable evidence and predictions. Notice the key word *evidence* is mentioned there again.

No surprises here so far, right? From a very simplified point of view, the scientific method makes use of data and

Statistics was originally understood as the analysis and interpretation of information about states.

¹ Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

Science is organised knowledge.

their analysis to acquire, correct and integrate knowledge. Nonetheless, data science is not just simply the direct use of statistics, or the systematisation of data. How shall we understand that much-loved combination of the words *data* and *science*?

However, Data Science $\neq$ Data + Science.

1.1.1 So, What Is Data Science?

DATA SCIENCE AND ANALYTICS HAVE been gaining prominence as some of the more sought after disciplines in academic and professional circles. Furthermore, recent advances in machine learning and artificial intelligence have had a transformative impact, particularly the advent of Large Language Models (LLMs) and generative Artificial Intelligence (Gen AI). These technological strides have significantly elevated the capabilities of data scientists, enabling them to extract nuanced knowledge and insights from an ever-expanding array of data sources.

There is no doubt that the term *data science* is a true neologism of our time. The term has being widely used and, to a certain extent, even abused. As we have mentioned before, data science is rather more than the sum of data on the one hand and science on the other one; although it is inevitably related to both concepts.

In the case of defining data science, the whole is indeed greater than the parts.

Data science has been slowly formalised, and there are now even professional certifications such as those offered by the Alliance for Data Science Professionals² with the aim to define the standards needed to ensure a framework under which data scientists can develop and apply their skills. We

² AfDSP. (2023, Feb). Alliance for Data Science Professionals. alliancefordatascienceprofessionals.co.uk

have come a long way since the first edition of this book, when data science was seen as a budding field.

We will continue to apply the practical definition proposed in the first edition of this book where *data science and analytics* is seen as a portmanteau for a number of overlapping tasks related to data — from collection, provision and preparation, analysis and visualisation, curation and storage — that exploit tools from empirical sciences, mathematics, business intelligence, machine learning and artificial intelligence. The aim of these tasks is to enable effective, pragmatic and most importantly actionable decisions. In a nutshell, data science can be understood as the extraction of knowledge and insight from various sources of data, and the skills required to achieve this range from programming to design, and from mathematics to storytelling.

The integration of machine learning techniques has reshaped the landscape of data science, empowering practitioners to derive actionable intelligence from complex datasets with greater efficiency. Large Language Models (LLMs), exemplified by models like GPT, have demonstrated remarkable language “understanding” and generation capabilities, facilitating more sophisticated analysis and interpretation of textual data. Moreover, Gen AI has emerged as a powerful tool, enabling the creation of synthetic data that can seamlessly augment real-world datasets, thereby enhancing not only the robustness but also the diversity and generalisability of analytical models across a wide range of applications and industries.

In this book we will use a practical definition for *data science* as a combination of overlapping tasks related to data with the aim to derive actionable decisions.

Data science skills range from programming to design, and from mathematics to storytelling.

As data science continues to evolve, the skill set required for proficiency in this field has expanded. Beyond traditional competencies in programming, design, mathematics, and storytelling, there is now a growing emphasis on mastering the intricacies of machine learning algorithms and leveraging the potential of artificial intelligence. The dynamic interplay between data science and these technological advances underscores the need for professionals to stay abreast of the latest developments in machine learning and AI, ensuring their ability to harness the full potential of these innovations in extracting meaningful insights from diverse data sources. This evolving synergy between data science and cutting-edge technologies underscores the ongoing transformation of how knowledge is derived and applied in our increasingly data-driven world.

Data science skills include a wide range of areas, from programming to storytelling, design and artificial intelligence.

The motivation for data science and analytics, as well as machine learning and artificial intelligence, in deriving valuable insights from data continues being great, and it is widely welcomed by businesses. Despite the recent advances in the area, it is still however very challenging task. Companies such as Google, Netflix and Amazon have demonstrated that careful storage and analysis of data delivers a very competitive edge. These days there are easier and cheaper ways to collect large amounts of data than ever before, and mobile is a ubiquitous presence. This has allowed companies, particularly start-ups and scale-up, to develop in-house data science capabilities that attract great talent.

Careful storage and analysis of data delivers a very competitive edge.

Typical examples of data science products are better explained by the questions they aim to answer; these questions are the drivers to the acquisition and selection of the appropriate data to be interrogated in order to provide insight into an area of interest. I am sure you can come up with a few of examples relevant to you, but here are some that come to mind:

• What product will sell better in conjunction with another popular product?	Market basket analysis
• Who will be declared prime minister (or president, or winner; depending on the flavour of the government system of interest) in the next general election?	Predictive analytics
• How can customers be encouraged to spend a longer time in an online portal?	E-commerce
• Are there any discernible patterns that allow us to characterise different groups of sales agents, customers or businesses?	Clustering analysis and market segmentation
• What advertisement should be placed on what site?	Advertising and marketing
• Given the interests of a customer, what other products can be recommended to them?	Recommendation systems
• What are the latest developments and breakout reports in newspapers and social media that may affect the industry of interest?	Social media analysis
• Given someone's interests and hobbies, who may be suitable potential partners?	Online services
• How can we keep potentially sensitive information protected and react proactively to information we store?	Cybersecurity

Some examples of data science outputs make it easier to clarify what the discipline does. This list is by no means exhaustive.

• How can we distinguish valid, relevant documents such as emails (ham), from invalid, irrelevant ones (spam)?	Classification analysis
• How to determine if a retail transaction is valid or not?	Fraud prevention
• What is the demand for a particular service at a particular time or place?	Demand forecasting
• How can we use Gen AI to create compelling marketing content that resonates with our target audience across various channels?	Content generation
• Given a user's historical preferences and behaviour, how can we employ Large Language Models to recommend personalised content such as articles, videos, or products?	Personalised content
• How can we enhance chatbot interactions by employing Gen AI to generate contextually appropriate and empathetic responses to user queries?	Automated chatbot responses
• In healthcare, how can machine learning models assist doctors by analysing patient data and generating insights to support accurate and timely diagnoses?	Healthcare diagnosis

By and large, apart from specific technologies mentioned above, these are not questions that decision-makers, businesses and industries, large and small, have recently started formulating. So, why the resurgence in seeking answers to them? The main answer is the availability of potentially useful data, big or small, together with the impact of technology, computer science and statistics in everyday life. Secondly, the continuous advances in the field such as Large Language Models and Gen AI enable for new

The availability of large volumes of data has enabled data science to flourish.

ways to seek answers. Out of the ingredients mentioned above, accessible datasets may be the most important one, since without them the insight provided by technology alone is rather limited. After all, the plural of anecdote is not data. Having said that, it is important to note that this does not mean that every single data science case to be tackled falls into the category of so-called *big data*, particularly when we take into account that the adjective *big* can be used in a relative manner. We shall expand on this point later on in [Section 1.3.1](#).

The plural of *anecdote* is not *data*.

One important thing to bear in mind about the outputs of data science, machine learning and analytics is that in the vast majority of cases they do not uncover hidden patterns or relationships as if by magic, and in the case of predictive analytics they do not tell us exactly what will happen in the future. Instead, they enable us to forecast what may come. In other words, once we have carried out some modelling, there is still a lot of work to do to make sense out of the results obtained, taking into account the constraints and assumptions in the model, as well as considering what an acceptable level of reliability is in each scenario.

Note that this book is about data science, not necessarily about big data.

Predictive analytics do not tell us the future; instead they allow us to forecast.

Similarly, there is the tacit prerequisite of having accurate, timely data that can be readily utilised to make sense out of the modelling results, and reflect the state-of-the-art in an application. It is therefore imperative that decision-makers as well as innovation, IT and business stakeholders take time to understand the information that will be needed, as well as being prepared to realise that certain data may not be fit for their purpose. It is indeed disheartening to

For data to be useful it should be available and it has to be timely.

come to terms with the fact that some data may not have the necessary features to be used in building a prediction, for example. Nonetheless, it is better to realise that is the case at an early stage, rather than relying on unsuitable results to make important decisions that impact the business.

Even a few years ago, data science may have not yet been considered a well-defined subject. Nonetheless, the number of academic and training programmes that are offered by universities and at various workplaces has seen a steady increase. This is a natural result of the need that exists for well-informed, capable experts that we get to call *data scientists*. So what do data scientists do and what do they look like? It shall all be uncovered.

1.2 *The Data Scientist: A Modern Jackalope*

THE TERM USED TO DESCRIBE the person that deals with the seemingly disparate array of tasks described above may seem to be yet another, more fashionable way to describe a statistician or a business analyst. However, we can certainly agree that there is a gap between the latter two, and that the skills required by a data scientist involve aspects that include both statistics and a strong business acumen, but also foundations in computer science, mathematics, modelling and programming, not to mention good communication skills. A simplified diagram of these skills and their relationship is shown in [Figure 1.1](#).

In that sense a data scientist's role goes beyond the collection and reporting on data; it must involve looking at

Realising that data may not be fit for answering the questions at hand is a difficult but important thing to bear in mind.

The need for capable data scientists in industry has seen a healthy increase in recent times.

A data scientist requires the knowledge of mathematics and computer science, but also a good business background.

The role of a data scientist goes beyond the collection and reporting on data.

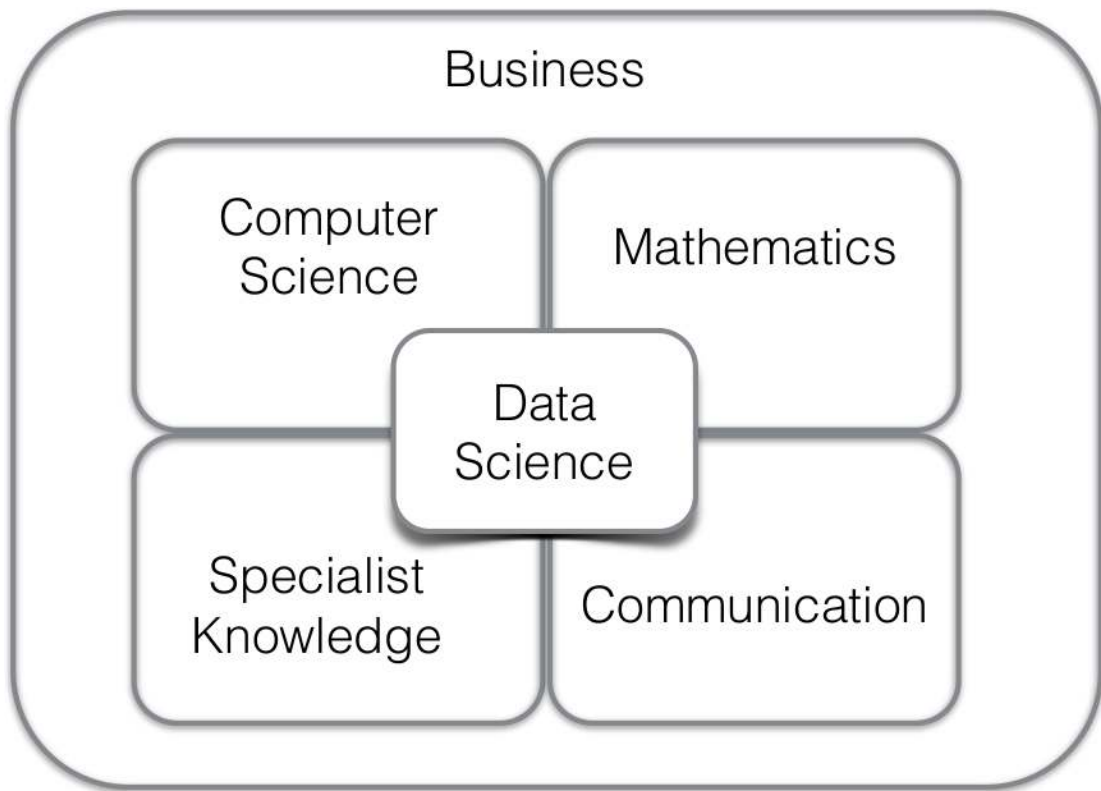


Figure 1.1: Simplified diagram of the skills needed in data science and their relationship.

a business application or process from multiple vantage points and determining what the main questions and follow-ups are, as well as recommending the most appropriate ways to employ the data at hand.

In terms of characteristics, a data scientist has an inquisitive mind and is prepared to explore and ask questions, examine assumptions and analyse processes, test hypotheses and try out solutions and, based on evidence, communicate informed conclusions, recommendations and caveats to stakeholders and decision-makers.

Not only does a data scientist need an inquisitive mind, but also good communication skills.

In other words, a data scientist is a true new Renaissance woman or man. No wonder that despite being branded the sexiest job of the 21st century a few years ago, as well as the increasing demand for these individuals, it is hard to find people with the right skills to fill in these roles, although many carry the title. This has led to branding data scientists as Unicorns.

No wonder data scientists have been dubbed Unicorns.

To a certain extent, the symbolism of a Unicorn as a creature that is beautiful, mysterious and difficult to tame or even capture may be applicable to describe a data scientist. However, in my opinion, it may not be totally appropriate given the fact that, as majestic as Unicorns can be, they are way too common as far as popular culture goes.

Branding data scientists as Unicorns is a result of the quixotic expectations businesses and industries have and thus is not appropriate.

The shortage that businesses experience when trying to attract data scientists is more likely due to the fact that they have created internal expectations for the role and that no single individual can fulfil, thus appealing to the magical nature of a common mythical beast. They have created their idea of the *data scientist Unicorn*, and unfortunately the fascination prevails.

To tackle the prevailing image, I am convinced that the use of a new symbol is needed. And a silly one at that! There is an allegory I usually propose to colleagues and those who talk about the *data science Unicorn*. It is still another mythical creature, less common perhaps than the Unicorn, but more importantly with some faint fact about its actual existence: a *Jackalope*. You can see an artistic rendition of a couple of Jackalopes in [Figure 1.2](#).

A new allegory is needed to tackle the unrealistic image of a data scientist.

The *Jackalope* is the one we propose.



Figure 1.2: Jackalopes are mythical animals resembling a jackrabbit with antlers.

A Jackalope is said to be a strange beast that looks like a jackrabbit with a pair of stag horns. It is described as a shy but clever and cunning animal, and if threatened it can be dangerous. If you are ever in the Mountain West in the United States, you may stumble into Jackalope heads mounted as trophies; but of course that is not the only place where Jackalopes are endemic; there are tales of the *Hasenbock* in Austria³ or you may hear the Huichol stories in Mexico about how the *tátSu* (rabbit) lost its antlers to the *kaukamali* (deer)⁴.

A Jackalope is a mythical being similar to a jackrabbit with a pair of stag horns.

³ Toelken, B. (2013). *The Dynamics of Folklore*. University Press of Colorado.

⁴ Zingg, R., J. Fikes, P. Weigand, and C. de Weigand (2004). *Huichol Mythology*. University of Arizona Press.

No need to explain that a Jackalope is indeed an imaginary, mythical being, much like the Unicorn, but it seems to be a better metaphor for the data scientist. We can argue that it is rather difficult at best, and impossible at worst, getting hold of a single individual who is able to be an all-round ninja programmer, with vast expertise in mathematics, statistics and probability, plus knowledge of computer science and well-versed in business. And who knows all the intricacies of all the latest machine learning techniques and recent Gen AI models. This way of thinking offers no solution to businesses interested in getting the benefit of exploiting the available data they have.

It is indeed difficult to get hold of a Unicorn.

Well, if you cannot get them in the wild, make them up from various parts — in the best style of Dr Frankenstein and his monster — and that is where the image of the Jackalope comes in handy. In 1932, Douglas Herrick did indeed put together his creation when he stuck a pair of deer horns on a dead jackrabbit and mounted it as a trophy⁵. The rest is history, as the Converse County city of Douglas, Wyoming became the Jackalope capital of the United States.

If you cannot get data scientists in the wild, make them up.

⁵ Martin, D. (2003, Jan 19th). Douglas Herrick, 82, Dies; Father of West's Jackalope. *The New York Times*.

Furthermore, you do not have to resort to obtaining a fake hunter's trophy to catch a glimpse of a Jackalope. As I mentioned earlier, there is faint but fascinating kernel of truth to the tales of horned rabbits. That is definitely more than one can say about one-horned horses. The Jackalope legend owes its plausibility to the existence of a virus known as the cottontail rabbit papilloma virus (CRPV). This virus causes the growth of keratinous tumors, which can

Plus, there is a faint but fascinating kernel of truth to the existence of horned rabbits.

sometimes appear as horn-like protrusions on the heads or bodies of infected rabbits.⁶ These unusual growths might have sparked the imaginations of early storytellers, giving rise to the myth of the Jackalope. The virus was discovered in the 1930s by Richard E. Shope and was the first example of a cancer caused by a virus.

The use of this allegory is proposed to show how silly it is to simply employ wishful thinking in the pursuit of exploiting data and hoping that a single individual will come to the rescue. What I am trying to say is that one should think optimistically about the prospect of finding capable data scientists if we are prepared to be realistic about distinguishing mythological aspirations from messy reality.

What I propose is that the best way to tackle the data science needs of a business — a startup or a large conglomerate — is to put together a rangale of Jackalope data scientists, than daydreaming of a bliss of non-existent Unicorns. After all, there are indeed better chances of seeing a Jackalope-like animal than a Unicorn, right?

The next question is thus related to how the rangale of data scientists should be put together, what roles they should have and what resources to provide them with. These points are perhaps not easy to answer, as they depend to a large extent on the area in a business where the insight is being sought, and for what purpose (see [Section 1.4](#)). Nonetheless, there are some general guidelines that can be taken into account when tackling the data scientist conundrum.

⁶ Zimmer, C. (2012). *Rabbits with Horns and Other Astounding Viruses*. Chicago Shorts. University of Chicago Press.

It is possible to find capable data scientists if we are prepared to be realistic about our expectations.

I propose therefore to put together a rangale of Jackalope Data Scientists.

Not only is it important to know what qualities a data scientist should have, but also what role they are expected to play and what tools they will use to do their jobs.

1.2.1 *Characteristics of a Data Scientist and a Data Science Team*

IT SEEMS THAT EVERYONE LOVES, or would love to have, a data scientist, and as we have seen, the wishful list of desired characteristics makes it more difficult for businesses to choose among otherwise capable candidates.

Everyone would like to have their own data scientist, and knowing what is important for the business needs is a major aspect to consider.

The analogy that comes to mind is that of the everlasting dating puzzle where everyone is waiting for Princess or Prince Charming, but is unable to find “the one”. For a data scientist to be considered “the one”, the skills required include those discussed in the previous section and summarised in [Figure 1.1](#).

Let us pause for a moment before we tackle the subject at hand and consider what the purpose of the data science team is or will be. This is a crucial step in building that team, as these objectives will help identify the important traits that the data scientists are expected to have.

Furthermore, having a clear idea of how they will fit in the organisation and what problems they are expected to solve will aid in defining the size of the team and the type of expertise needed. It is not uncommon to hear of organisations that are interested in riding the data science wave, but that do not have a clear goal regarding the purpose of their data science journey.

Having a clear idea of how a potential data scientist will fit in the organisation and what they will work on is important.

With the objective of the data science team in mind, it becomes much easier to decide what is relevant in a

particular case. In general, what makes a good data scientist is a linear combination of some of the following traits:

- Curiosity
- Grasp of machine learning
- Data product building and management
- Effective communication of data insights
- Programming and data visualisation abilities
- Experience in machine learning and artificial intelligence
- Knowledge of statistics and probability (other mathematical areas are welcomed)
- Healthy skepticism, in the scientific tradition: Carry out experiments, test hypotheses, etc.

Some important traits in a data scientist.

The important thing to realise here is that the linear combination of the features mentioned above do not necessarily have to be equally weighted, and that is the main reason for the persistence of the Unicorn fallacy we have been discussing. Should your data scientists lack some more developed branches in their antlers, all you need to do is give them a helping hand and provide them with colleagues who will help in developing those skills, but more importantly cover the gap in those desirable features. In other words, much like Mr Herrick, put together your very own Jackalope team with people who have a broad-range of generalist interests, but a deep expertise in a certain area or two.

The features mentioned do not have to be combined in equal measures.

Apart from offering them training of course.

The sensible thing to do is to start with a solid core and not allow the list above let you get carried away. In other words, setting the foundations of the data science team is similar to having strong foundations in a building; without them the whole tower may collapse in an instant. Furthermore, use this core to your advantage and bank some of the easy wins to start with. The four pillars in this data science quartet I am referring to may include, with variations in the titles, the following main roles:

- Data Science Project Manager
- Lead or Principal Data Scientist
- Data Architect or Data Engineer
- Data Operations Engineer

Start a data science team with a solid core, perhaps made out of more than one person.

The data science quartet.

Having a person who is able and experienced in managing technical teams is an important role to have in the mix. The main idea is to cover the fact that many a data scientist is far more interested in tackling questions and problems head on, rather than dealing with managing a project from end to end. One way to help them deliver is to have a knowledgeable individual who is able, on the one hand, to keep track of how projects are going, attend meetings and manage relationships. On the other hand, they should have a general understanding of techniques, algorithms and technology to be able to liaise with the team effectively. The project manager does not have to be a ninja programmer, but should be able to understand what the rest of the team are working on and the challenges they may be facing.

First, a Data Science Project Manager is needed.

The second figure in the quartet is that of the principal data scientist. Not only is it necessary to have a good project manager, but also have someone with a strong background in a quantitative field: Physics, mathematics, computer science, etc. Ideally the academic credentials this person would speak for themselves. In terms of programming, this person may not be a developer in the full sense of the word; however, they should have a firm background in coding and solving problems with the use of technology. An important ingredient of the role is to be able to act as an advisor or guide to other data scientists and analysts in the team.

Followed by a Lead Data Scientist.

The third pillar in the team is the data architect or data engineer, who will provide expertise in terms of data structures, databases, software engineering and computational capability. It is important for the data architect or engineer to be able to disentangle the data resources that the business may (or may not) have, and be able to use their expertise to assess what data is available, when it is available, and manage the constraints that the business, regulation and security impose on the workflow. Ideally, the data architect/engineer would be interested in quantitative topics, but most importantly their programming skills must be spot on. Note that the data architect will use the same technology that the data scientists employ in their day-to-day activities.

A Data Architect or Data Engineer is the third pillar in the quartet.

The fourth pillar in our quartet is the Data Operations Engineer. They are responsible for managing and optimising the infrastructure and processes related to data that are used by the rest of the team. They oversee tasks

such as setting up and maintaining data storage systems, databases, and data warehouses and data lakes, optimising database performance, and implementing data quality checks. Data Operations Engineers collaborate closely with the rest of the team, providing support in deploying machine learning models and ensuring that the data infrastructure meets scalability requirements. They play a crucial role in automating routine data tasks, enforcing security measures, monitoring system performance, and planning for disaster recovery.

Our quartet is completed by a Data Operations Engineer.

Finally, there are four aspects that are important to remember when considering putting together a data science team. First, consider who the main stakeholders of the data science team are, and clarify the lines of reporting. Remember that everyone wants their own data scientist, and confusing or conflicting messages can lead to undesired results.

Clear reporting lines are also important.

Second, for data scientists to be able to work independently and (more importantly) productively, it is important for them to be able to navigate the stack entirely. This enables extracting relevant data with appropriate tools (see [Section 1.3](#)). A data science team without strong IT skills or engineering support will have a hard time doing the job they do best.

Having appropriate tools to work with is paramount.

Third, once data has been identified for tackling a problem, proper interpretation is not necessarily easy, and misrepresentation of the results can be very damaging. It is not uncommon to see the use of tools such as machine

It is necessary to have appropriate expertise to interpret and rework results.

learning algorithms to be seen as a black box; in practice, knowing the capabilities, limitations and trade-offs requires experience.

Fourth, have the product always in mind: Not only is it important to have the right IT and statistics and/or machine learning skills, but also the team has to have a clear idea of the final product they are helping build with their efforts, as well as their target audience. You may be able to come up with the most amazing models and results, but they may not be of much use if the product is of no interest to stakeholders or if the data scientist fails to communicate the results to them.

Also have a clear idea of the final product and communicate results clearly.

When considering the tools used to present results, it is essential to evaluate their accessibility and usability for the intended audience. A data scientist may leverage cutting-edge technology to create visually stunning and highly interactive presentations, designed to impress and engage stakeholders. However, if the audience lacks the technical infrastructure, expertise, or even the appropriate permissions to access these tools, the effort is in vain. It is critical to align the choice of presentation tools with the audience's technical capabilities and context, ensuring that the insights are not only impactful but also accessible and actionable. Otherwise, you risk alienating the very people you aim to inform and persuade, losing the opportunity to make a meaningful impact before the conversation even begins. Remember that the presentation's effectiveness hinges not just on its sophistication but on its ability to connect with the audience in a meaningful way.

Appropriate technology for presenting and delivering results is also important.

The presentation's success depends not only on sophistication but also on meaningful audience connection.

A point in case, in my experience, is the use of great JavaScript libraries such as D3. I am an advocate for their use, as they can be effective and even great fun to use. However, they only work on “modern” browsers, and unfortunately a large number of institutions only support old browsers unsuitable to render the created assets. This becomes a relevant point when considering the deployment of solutions (dashboards, reports, etc.).

Be mindful of the ways in which solutions are going to be deployed.

1.3 *Data Science Tools*

WITH OUR NEWLY ACQUIRED DATA science team and the individual high-calibre data scientists and analysts who compose it, we are able to keep abreast of the latest developments in the field of analytics and data science, and we are able to extract actionable insights from our data. However, not only do we need to be flexible, agile and experts, we are also required to have the right tools and infrastructure to enable the team to fulfil the objectives agreed with the team sponsors. To that end, there are a number of considerations that we would need to think about in helping the team decide on the tools needed as well as some other points such as:

The tools chosen need to enable us to be flexible, agile and experts.

- Regulatory and security requirements of hosting and manipulating the data
- Locations of data sources – and related subjects such as whether we would need/have immediate access to them, or would get them in batches for upload

Some considerations when choosing appropriate tools.

- Responsiveness requirements for queries – e.g., real-time v fixed reporting
- Volume of queries/searches to be run
- Format of the data source
- Quality of the data

The security consideration above is usually a big question for any business that requires its data to be in a particular jurisdiction and does not plan to create its own cloud service. Some cloud providers may not be able to guarantee that data will stay in Europe, for example.

Security of the data is very important.

Data science and analytics is all about data, statistical analysis and modelling. It is therefore important to have the technology that enables those functions. A data warehouse or a data lake, extract, transform, load (ETL) software, statistical, modelling and data-mining tools are necessary. Similarly, an appropriate hardware and network environment are required and better off if they are in the cloud.

A data warehouse or data lake, ETL software, statistical, modelling and data-mining tools are necessary.

The technologies used in the analytics arena have evolved at a fast pace in the last few years, and a number of open-source projects, with lots of support have emerged, for instance:

- *Data framework:* **BigQuery, Spark, Databricks, Snowflake.** They serve distinct purposes in the realm of data processing and analytics. BigQuery, is a serverless data warehouse by Google Cloud, excels in real-time analytics

Data framework technologies

and ad-hoc querying, offering a scalable solution for business intelligence and data warehousing in a cloud environment. Apache Spark, is an open-source distributed computing system, provides a versatile framework supporting multiple programming languages and is widely used for batch processing, interactive queries, stream processing, and machine learning tasks.

Databricks, built on top of Apache Spark, is a unified analytics platform designed to simplify collaborative data science, machine learning, and analytics. It streamlines Spark cluster deployment and management, offering an integrated workspace for teams to work together seamlessly. Finally, Snowflake, a cloud-based data warehousing platform, distinguishes itself with its separation of storage and compute resources. These frameworks collectively address diverse data processing needs, from real-time analytics and collaborative data science to scalable cloud-based data warehousing.

- *Streaming data collection:* **Kafka, Flink, Amazon Kinesis.** Kafka, Flink, and Amazon Kinesis are notable tools for streaming data collection and processing, each offering unique features and capabilities to meet the demands of real-time data applications. These tools play a crucial role in modern data architectures, enabling organisations to harness the power of continuous data streams for analytics, monitoring, and decision-making.
- *Job scheduling:* **Azkaban and Apache Oozie.** Tools for job scheduling are essential in orchestrating and managing workflows, ensuring that various tasks are executed in

Streaming data collection technologies

Job scheduling technologies

a timely and coordinated manner. Azkaban is an open-source workflow management system developed at LinkedIn, designed to simplify the execution, monitoring, and dependency management of jobs. It allows users to define job dependencies graphically through a web-based user interface, making it user-friendly for workflow creation.

- *NoSQL databases:* **MongoDB** and **Apache Cassandra**.

NoSQL databases

For querying non-relational databases, several tools cater to the unique structures and requirements of these databases. MongoDB offers MongoDB Compass as a graphical user interface tool for visually exploring and querying data stored in MongoDB. Apache Cassandra has tools like DataStax DevCenter and CQL Shell for querying.

- *Workflow management:* **Apache Airflow**, **Luigi**, **Prefect**, **Apache NiFi**. Tools for data science workflows are crucial for orchestrating and automating the various stages of the data science lifecycle. Apache Airflow stands out as a popular open-source platform for workflow automation and scheduling. Airflow allows data scientists to define, schedule, and monitor complex workflows as Directed Acyclic Graphs (DAGs). It integrates seamlessly with different data science tools, databases, and services, enabling the automation of tasks such as data extraction, transformation, model training, and deployment. Other tools in this category include Luigi, Prefect, and Apache NiFi, each with its own strengths and features tailored to streamline and automate different aspects of the data

Workflow management tools

Other workflow tools include Luigi and Prefect.

science workflow. These tools empower data scientists to focus on modelling and analysis while ensuring the systematic and automated execution of tasks throughout the data science pipeline.

1.3.1 *Open-Source Tools*

THE MODEL OF DEVELOPING TOOLS whose source code is made available for contribution has shifted the environment for their deployment both in small and large enterprises. The collaborative nature of the various projects provides a pool of knowledge and quality assurance that is difficult to beat. A rich and wide set of tools in the open-source domain have contributed to the expansion of data science. They include tools that process large datasets as well as data visualisation, together with prototyping tools:

- **Python:** Data manipulation, prototyping, scripting, and the main focus in this book
- **Apache Airflow:** A platform to programmatically author, schedule and monitor workflows.
- **Jupyter Notebook:** A web application that can host code, data, notes, equations, etc. — in other words, an interactive online document.
- **Spark:** Cluster-computing framework for data analytics
- **The R Project for Statistical Computing:** Data manipulation and graphing
- **Julia:** High-performance technical computing

There are many open-source tools that can be readily used in the data science workflow.

We will be using Python in this book.

R is a noteworthy software package widely used by the data science community.

- **GitHub, Subversion:** Software and model management tools
- **Ruby, Perl, OpenRefine:** Prototyping and production scripting languages

A few years ago Hadoop would have been the tool to be mentioned when processing massive datasets. These days, a ubiquitous tool for this task is Apache Spark. It is a distributed computing system that provides a fast and general-purpose cluster computing framework for big data processing. It is designed for scalability and efficiency, offering in-memory processing capabilities that significantly speed up data processing tasks. Spark supports various programming languages, including Scala, Java, Python, and R, making it accessible to a broad range of users.

Apache Spark is a great option for processing massive datasets.

R is widely regarded as the go-to programming language for statistical computing and data analysis. While it may not be celebrated for the elegance or readability of its code, it is highly valued for its powerful capabilities and the quality of its outputs. One of R's greatest strengths lies in its extensive ecosystem of packages available through the Comprehensive R Archive Network (CRAN). With thousands of specialised packages covering a vast range of statistical methods, data visualisation techniques, and machine learning tools, R offers unparalleled flexibility for tackling diverse analytical challenges.

R is widely regarded as the go-to programming language for statistical computing and data analysis.

The use of scripting languages such as **Python** provides a professional platform for application development and

deployment. It is very well suited for prototyping and testing new ideas. Furthermore it supports various data storage and communication formats, such as XML and JSON; plus there is a large number of open-source libraries for scientific computing and machine learning.

Python has become the *lingua franca* for data science.

Python has a number of very useful libraries such as SciPy, NumPy and Scikit-learn. SciPy extends Python into the domain of scientific programming. It supports various functions, including parallel programming tools, integration, ordinary differential equation solvers, and even extensions for including C/C++ code within Python code. Scikit-learn is a Python-based machine learning package including many algorithms for unsupervised learning (clustering algorithms), supervised learning (naïve Bayes, support for vector machines), and other algorithms for dataset manipulation. It is for these reasons that we will use Python in the rest of this book.

Python is a well-supported language with a wide variety of modules and libraries.

See [Chapters 5, 6 and 9](#) for more information.

1.4 From Data to Insight: the Data Science Workflow

AS WE HAVE SEEN ABOVE, the role of a data scientists is an interesting one, and at times a challenging one. Not only do we need the right combination of skills (either in a team or an individual), but also the right tools and business questions. In this section we will address the steps that a data science project may follow. It is important to emphasise that although we may categorise and separate the various steps, the workflow is not necessarily a linear one, as we shall see.

The workflow of the data science expert is worth discussing too.

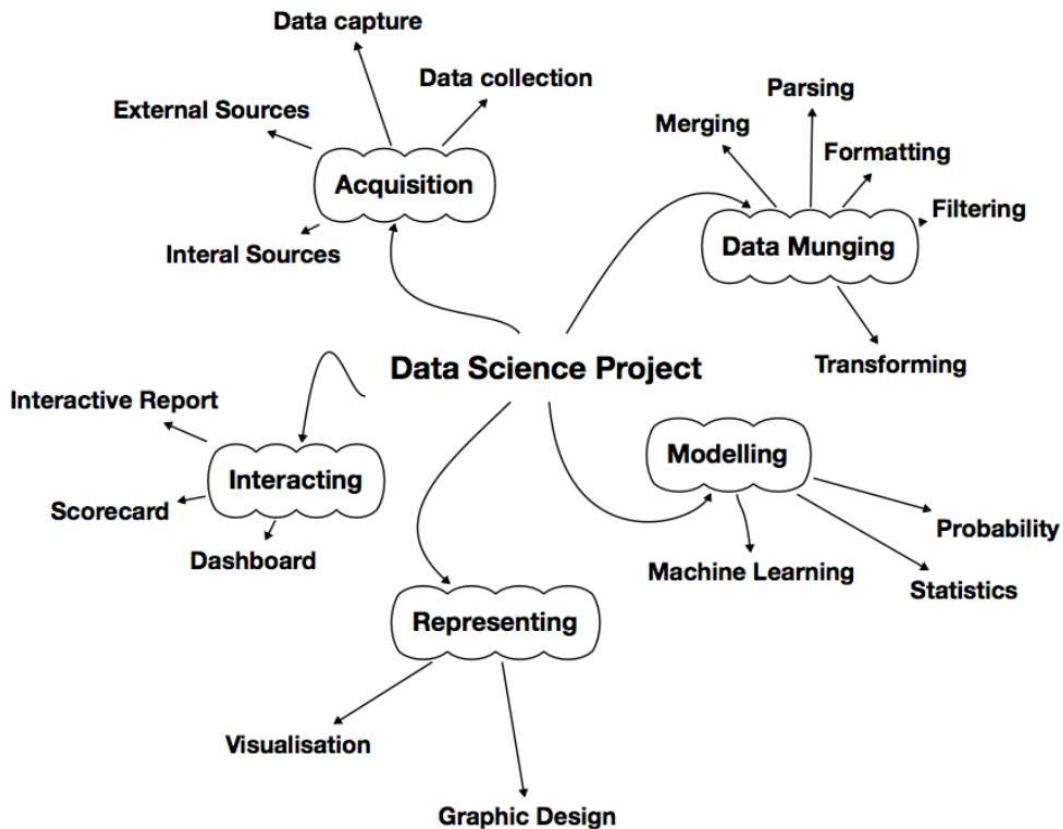


Figure 1.3: The various steps involved in the data science workflow.

With our newly acquired data science team (see [Section 1.2.1](#)) and the right combination of skills, we are ready to tackle our first assignment, and it is now that key measurements of success for the project should be identified. Furthermore, it is important to realise from the start that in the vast majority of cases there will not be a unique, final answer. It is thus better to frame the problem as an iterative process where a better solution is reached on each iteration. The various steps, shown in [Figure 1.3](#), in the data science workflow include:

The data science process is iterative.

- Question identification
- Data acquisition
- Data munging
- Model construction
- Representation
- Interaction

The steps in the data science workflow

The fact that they have been listed in that order does not mean that they have to be followed one after the other. In some cases you may start, for example, with an internal dataset and immediately create some plots even before cleaning the data. Also, once you have started the project, you may move between steps in every iteration.

The workflow outlined above is not necessarily followed in sequence.

1.4.1 *Identify the Question*

THERE IS NO SUCH A thing as insight without a clear and concise question, as well as having a way to measure the success or failure of the answer obtained after running your favourite machine learning algorithms. It is not a matter of getting a dataset and simply massaging it and creating some plots. On the contrary, let the questions point out the potential datasets that may be useful in answering them and to what extent.

Without a clear question, there is no insight.

Another important thing to bear in mind is the fact that although we may encapsulate a problem in a single

question, in many cases it is much easier to break it down into smaller parts that can be tackled in a more straightforward manner. Furthermore, at every iteration there may be more, smaller or larger, follow-up questions that will also require answers. Remember that it is an iterative process!

Breaking down the problem into smaller questions is useful.

1.4.2 *Acquire Data*

ONCE YOU HAVE A PROBLEM to tackle, the first thing is figuring out if you or your organisation has the appropriate data that may be used to answer the question. If the answer is no, you will need to find appropriate sources for suitable data externally – web, social media, government, repositories, vendors, etc. Even in the case where the data is available internally, the data may be in locations that are hard to access due to technology, or even for regulatory and security reasons.

Identify appropriate sources of suitable and useful data.

1.4.3 *Data Munging*

IF THERE IS NO INSIGHT without a question, then there is no meaningful data without the process of data munging. Data munging, also known as data wrangling, is the essential step of cleaning, transforming and organising raw data into a structured format suitable for analysis. This process is foundational in ensuring that the data used for any analytics or modeling is reliable, accurate, and relevant. As Steve Lohr of *The New York Times* aptly noted, data scientists can spend anything between 50% and 80% of their

There is no data without data munging, or wrangling.

time performing what he calls “data janitor work”⁷—a term that underscores the often unglamorous but essential nature of this stage. Based on my own experience, I can confirm the truth of this observation. Without thorough data preparation, the extraction of valuable insights becomes a near-impossible task.

⁷ Lohr, S. (2014, Aug 17th). For Big-Data Scientists, ‘Janitor Work’ Is Key Hurdle to Insights. *The New York Times*.

While many might be tempted to focus solely on the more exciting aspects of data science, such as building sophisticated models or uncovering groundbreaking patterns, the process of munging or wrangling data is where one truly becomes intimately familiar with their dataset. This deep familiarity means that any subsequent or follow-up questions can often be addressed with greater speed and efficiency, making data preparation not just a preliminary step but an ongoing and invaluable part of the analytical workflow.

Munging is where one truly becomes intimately familiar with their dataset.

1.4.4 Modelling and Evaluation

HAVING A CLEAN DATASET TO feed to a machine learning or statistical model is a good start. Nonetheless, the question remains regarding what the most appropriate algorithm to use is. A partial answer to that question is that the best algorithm depends on the type of data you have, as well as its completeness. It also depends on the question you decided to tackle. Once the model has been run through the so-called *training* dataset, the next thing to do is to evaluate how effective and accurate the model is against the *testing* dataset and decide if the model is suitable for deployment.

Every model needs to be evaluated.

For more on training and testing datasets, see [Section 3.12](#).

1.4.5 Representation and Interaction

THEY SAY THAT A PICTURE is worth a thousand words, and it does stand to reason that the vast majority of us get further information from a couple of well-presented plots than by looking at rows and rows of data. Data visualisation is more of an art than a science, and much has been written (or drawn) about by brilliant designers and data journalists. Simply take a look at the great images produced by David McCandless in his book *Information is Beautiful*⁸ or the visualisations produced by Manuel Lima in *Visual Complexity*⁹.

You do not have to produce such artistic beauties, but make sure that the data representation that you decide to use is accurate, simple, and provides clarification to the story you want to communicate. In some cases, there is the possibility of telling the story in a way that the reader/viewer is able to interact with the data representation, for instance in the form of dashboards, reports or interactive plots. These may be fun to use, but the same principles of accuracy, simplicity and clarity do apply¹⁰.

1.4.6 Data Science: an Iterative Process

THE SIMPLE FACT THAT A machine learning model has been run on clean data does not mean that the work of the data scientist is done and dusted. On the contrary, the effectiveness of the model needs to be carefully monitored, as the outcome depends on the data that is fed to it. This is

Data visualisation is more of an art than a science, but an important one nonetheless.

⁸ McCandless, D. (2009). *Information is Beautiful*. Collins.

⁹ Lima, M. (2011). *Visual Complexity: Mapping Patterns of Information*. Princeton Architectural Press.

¹⁰ Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

a simple case of *garbage-in-garbage-out*. Similarly, any new data with a variety of new features may degrade the accuracy of the initial model, and thus it becomes necessary to adjust parameters or acquire new data. Continuous monitoring and adaptation are critical aspects of maintaining model relevance and ensuring it performs optimally in dynamic, real-world scenarios where the data landscape is ever-evolving. This iterative process underscores the importance of an ongoing commitment to refining models and adapting to changing data patterns to achieve sustained success in machine learning applications.

Furthermore, even if there were such a thing as a never-changing model, the fact is that new and follow-up questions arise naturally from the data science process. This means that the workflow is better to be treated as an upward spiral where constant iterations provide improvement and new insights.

Think of this process as an upward spiral.

A few questions that the data scientist team and their stakeholders should always have in mind:

- What data was used and why?
- Where was the data acquired and who owns it?
- Was the entire dataset used? Is a sample representative of the entire population?
- Were there any outliers? Have they been considered in the analysis?
- What assumptions were made when applying the model/algorithm? Are they easily relaxed/strengthened?

Some questions to always bear in mind during any data science project.

- What does the result of the model mean to the process/business/product?

1.5 *Summary*

IN THIS CHAPTER WE ADDRESSED some crucial aspects that will enable us to implement and acquire that elusive mythical being called a data scientist. We have provided a working definition for the term *data science* and have described how it is a rapidly evolving multi-disciplinary field requiring expertise in mathematics, computer science, statistics and business.

We mentioned a few examples of data science products and have seen that the main motivation for data science and analytics is deriving valuable and actionable insights from data. Our discussion pointed out that in order to fulfil that motivation, a data scientist with the appropriate skills is needed. Unfortunately, understanding about this subject has created expectations that point at the data scientist role as one akin to the mythical Unicorn. We argued that a better understanding of the role indicates that we should perhaps use a different allegory: A Jackalope. It is still a mythical being, but the fact that you can put one together out of different parts (as in a team), or the hint that there may be a scientific explanation for potential existence provides a more hopeful panorama for many businesses interested in exploiting their data.

We continued our discussion with the three main pillars that would sustain a productive data science team, namely, a

data science project manager, a lead or principal data scientist and a data architect. We saw that not only is the team composition important, but also the tooling provided to carry out their tasks in a productive manner. We concluded this chapter by outlining the steps in a simplified data science workflow and explained their importance as part of what should be seen as an iterative process.

In the next chapter we will provide a brief refresher of some important concepts of using Python. This will enable us to have a point of reference for the rest of the book where Python will be used to implement a variety of algorithms that form part of the Jackalope's bag of tricks.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Python: For Something Completely Different

THERE IS NO SHORTAGE OF programming languages and paradigms. With that in mind, it may seem surprising that what started up as a “hobby programming project” by Guido van Rossum in 1989 has taken on a life of its own, with a very active community, used in a wide variety of applications. Its success is perhaps due to the compactness of its code, or the fact that it is open-source, or even the variety of its toolsets. Whatever the reason, Python is a great tool to have under your Jackalope Data Scientist belt.

Python started up as a hobby programming project.

Python is not named so after the non-venomous snakes whose family includes some of the largest in the world. Instead it is named after the famous British comedy troupe *Monty Python*, and it was created to be appealing to Unix/C coders. Today, as ever, Python’s emphasis on code readability and expressive syntax has made it into a general-purpose, high-level, object-oriented programming language with a plethora of supporting packages and modules.

We have to thank Monty Python for the name of this flexible programming language.

```

> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break
the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation
to guess.
There should be one -- and preferably only one --
obvious way to do it.
Although that way may not be obvious at first
unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain,
it's a bad idea.
If the implementation is easy to explain,
it may be a good idea.
Namespaces are one honking great idea
-- let's do more of those!

```

“Pythonic” coding style guidelines
are followed by so-called
Pythonistas.

Perhaps some of the most defining features of the language include the use of indentation for grouping statements, together with what has become to be known as the “Pythonic” coding style, i.e., style guidelines and idioms used by veteran Python programmers, aka, Pythonistas. These guiding principles can be read in any Python installation by importing the `this` module. These aphorisms, compiled by the long-time Pythonista Tim Peters, are shown in the code listing printed above.

The use of indentation is an important feature of Python.

This book assumes some knowledge of general programming principles and a level of familiarity with Python. Nonetheless, in this chapter we will briefly review some of the concepts and idiosyncrasies of the language that will be used throughout the book. If you are a seasoned Python programmer, you may want to skip this chapter and move on to the next one. If, however, you are interested in a refresher, then go ahead and read the following pages, you might find something completely different. For those of you who are beginning their Python journey, this chapter may whet your appetite to learn more. There are plenty of resources to quench your thirst^{1,2}.

We will assume a certain familiarity with programming principles and with Python in particular.

2.1 *Why Python? Why not?!*

WE MENTIONED ABOVE THAT PYTHON emphasises code readability, and this in turn has an impact on productivity. Not only is a data scientist able to create scripts to be executed as a batch, but also is able to start an interactive

¹ Downey, A. (2012). *Think Python*. O'Reilly Media.

² Langtangen, H. (2014). *A Primer on Scientific Programming with Python*. Texts in Computational Science and Engineering. Springer Berlin Heidelberg.

console (iPython shell, for instance) or notebook (Jupyter Notebook - see [Section 2.1.2](#)). Furthermore, the Python ecosystem is supported by various packages that extend and enhance the language. For example, the NumPy module provides functions that enable the manipulation of numeric arrays and matrices. The SciPy module enables functionality usually expected in scientific computing such as optimisation, linear algebra routines, Fourier transformation, etc.

The support that Python has for hierarchical modularity makes it possible for programmers and developers to build further functionality. A good example is the pandas package, which extends the NumPy arrays into dataframes for ease of data manipulation and analysis (see [Section 2.5](#)). We will be using pandas in the rest of the book. Similarly, in this book we will extensively use packages such as Matplotlib, statsmodels and Scikit-learn which implement plotting, statistical models and machine learning algorithms, respectively.

It is true that Python is an interpreted language, and as such code may be often slower than compiled code tailored to a particular machine architecture. In that respect, although the source code in Python is interpreted on the fly, the main advantage is flexibility. This is an important point in the data science workflow as we are interested in the balance between implementation time versus execution: In many cases we are more interested in getting to grips with the raw data rather than in fine-tuning the execution time for a particular machine.

Interactive sessions are available through the iPython shell and the Jupyter Notebook.

NumPy provides numerical functions to be used with Python.

The pandas package enables data manipulation and analysis.

Other popular packages are Matplotlib, statsmodels and Scikit-learn.

Python is an interpreted language, in other words, code is read and executed line-by-line.

That brings us to another advantage: Since Python code is not compiled, we have portability. Scripts developed in one environment can be executed in any other one with the appropriate installation. We will cover the use of environments in [Section 2.7](#). Python is fast enough for the vast majority of the computational tasks in a data science workflow: It is important to get the logic right in the first place and, if needed, make the execution faster later.

Python code is flexible and portable.

It is fast enough for the typical data science workflow.

For the purposes of this book, I assume that a suitable Python installation is available to you. Furthermore, I will also assume that the installation is for version Python 3.x rather than Python 2.x. Python 2 was a great version, but it needed evolving, and in January 2020 the Python Software Foundation announced their decision to sunset it. Python 3 has become the default Python interpreter for many Linux distributions and if you, or your organisation, are still using version 2, please consider transitioning away from it.

We will be working with version 3 of the Python distribution.

In particular, I find that the Anaconda distribution³, built by Anaconda (formerly known as Continuum Analytics), is robust and complete enough for our purposes. Furthermore, they have suitable distributions for Windows, MacOS and Linux. You can obtain an installation package from <https://www.anaconda.com> and follow the steps provided. Please note that any other Python 3.x installation with the appropriate packages is equally suitable to follow the discussions in the rest of the book. I will explicitly mention any modules or packages that are required, so that the more seasoned user is able to use pip, easy-install, Homebrew or any other installation method they are comfortable with.

³ Anaconda (2024). Anaconda 3.0. <https://www.anaconda.com/download/>

2.1.1 *To Shell or Not To Shell*

WE HAVE SEEN ABOVE THAT Python is an interpreted language, and as such it is possible to interact with the different commands that have to be executed during the course of a session. Similarly, it is also possible to write all the commands first and execute them in a more conventional development workflow. Both approaches have their advantages and disadvantages, and fortunately you do not have to choose one over the other, as Python is flexible enough to enable you to do both.

If you are interested in interacting with the code as you are writing it, then starting, for instance, an iPython shell will allow you to type a command and immediately send it to the interpreter for execution. You can then take the output and continue your work or analysis. This way of working lets us see the results of the commands as we are working. It is an excellent way of prototyping code to be implemented in longer projects, or get a sense of the data you are dealing with.

Unfortunately, working in this way makes the code somewhat ephemeral unless you save the commands that you are issuing to the interpreter. If you are interested in keeping track of your development and executing a series of commands repeatedly, then you can save those commands in a text file to create a Python script which by convention has a “.py” extension. The execution of these scripts does not require a Python shell and instead can be run directly from the command line using the following syntax:

Python shell enables interactivity. You can also develop scripts to be executed without user intervention.

The interactive shell lets you assess the results of each command you send to the interpreter.

Python scripts let you save the series of commands that form a program. They are saved with the .py extension.

```
python myscript.py
```

where we are assuming that the script called `myscript.py` is saved in the local path. The use of Python scripts makes it easier to launch longer jobs that do not need input from a user to be executed.

The command above is launched directly from the terminal; no need for the Python/iPython shell.

In this book we will take the approach of using an interactive shell (code is compatible with Python/iPython shells), so that we are able to use the output given by the interpreter to explain the different steps we are taking. These individual commands can then be saved in a suitable script that can be run later. We will thus present code as follows:

```
> 42 + 24

66
```

In this book we will present code as used from the Python shell for ease of presentation.

Notice that the diple, `>`, represents the Python shell prompt where the next line of code is expected. Furthermore, if the command sent to the interpreter expects a printed result, the shell will automatically display it. For ease of explanation, in cases where we do not expect a result, or the discussion requires us to break down the code, we will show it in a script style. In other words, no shell prompt will be shown:

```
a = 42 + 24
```

In cases where we do not expect a result, we will not show the shell prompt.

Comments in Python are indicated with the use of the hash symbol, #. The interpreter will ignore any commands that follow until the end of the line. In the example below we enter a comment after carrying out a division operation

A comment in Python is entered with the hash symbol, #.

```
> 2/3 # Python 3
```

```
0.6666666666666666
```

Python 3

Note that the behaviour in Python 2 was different, as the operation above would have resulted in an integer division:

```
> 2/3 # This integer division returns 0
```

```
0
```

Python 2

Since we have passed two integers to the interpreter, Python 2 carries out an integer division, returning only the integer part. If we want a real division, we can do the following:

```
> 2/3.0
```

```
> 2/float(3)
```

You can avoid integer division in Python 2 by casting integers into floats.

2.1.2 Jupyter Notebook

WE HAVE SEEN HOW THE interactive shell in Python allows us to assess the results of the code we are executing. That may be good enough for a number of tasks, but sometimes it may be desirable to present the code in a way that is

easier to interact with, not just for the original programmer, but also with other members of a team, stakeholders, or the general audience.

The Jupyter Notebook is a great way to do this. Not only does it let you run your code in the same way as the interactive shell and add comments to your code, but also enables you to document your code, calculations and processing all within a web-based interface. In this book, we have made a conscious decision not to use the Jupyter Notebook for printing purposes; however, I would encourage you to make use of it whenever you can.

The Jupyter Notebook lets us interact with the code and also add rich media, text and maths.

The Jupyter Notebook supports the inclusion of text, mathematical expression and inline graphics as well as other rich media such as websites, images, video, maths, etc. At its core, a notebook is a JSON document with the extension `.ipynb`, which makes the files very light and highly portable. The web interface is very handy, and if required, the notebook can be exported to a number of formats such as HTML, $\text{\LaTeX}$, PDF, Markdown or even raw Python. Furthermore, the Jupyter project aims to exploit the parts of the code that are not specific to Python and extend their use to other programming languages.

Originally Jupyter Notebooks were called iPython Notebooks.

A notebook is a JSON document, keeping with the general portability principle of Python.

2.2 *Firsts Slithers with Python*

WE HAVE ALREADY HAD THE opportunity to interact with the iPython shell and have seen some simple operations such as addition (+) and division (/). We can continue our

exploration of the programming language as an interactive calculator. As we would expect, the rest of the arithmetic operations are supported by Python as shown in [Table 2.1](#). Notice that the exponentiation in Python is represented with two stars, `**`. So far we have used integers and floating point numbers in the examples presented. It is therefore natural to ask what other types are supported by Python.

Exponentiation is denoted with `**` in Python.

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

Table 2.1: Arithmetic operators in Python.

2.2.1 Basic Types

AN IMPORTANT FACT TO REMEMBER about Python is that it is a dynamically typed language. In other words, we do not need to declare variables before we use them, and it is not necessary to specify their type. Furthermore, each and every variable that we create is automatically a Python object.

Python is a dynamically typed language: We do not need to specify the variable type in advance.

2.2.2 Numbers

AS WE SAW IN THE previous section, Python supports two types of numbers: Integers and floating point numbers. So we can assign the value of an integer to a variable as follows:

Python supports integers and floating point numbers.

```
> Universe = 42
```

Notice that assignment does not require Python to print anything as a response. We can check the type of an object with the command `type`:

```
> type(Universe)

int
```

The command `type` lets us see the type of an object.

Python will let us know what type of object we are dealing with; in this case, the object `Universe` is of type integer. Let us see an example for a floating point number:

```
> Universe2 = 42.0
> type(Universe2)

float
```

In this case, we can see that the object is a float.

2.2.3 *Strings*

A `STRING` IS EFFECTIVELY A sequence of characters. In Python, strings can be defined with the use of either single (`' '`) or double quotes (`' ' ' '`) as follows:

```
> string1 = 'String with single quotes'
> string2 = '“String with double quotes”'
> type(string1)

str
```

Strings in Python can be defined with single or double quotes.

In the example above we have requested the type of the variable `string1` and, as expected, Python tells us that indeed it is a string.

We can ask Python to print a string as follows:

```
> print(string2)

String with double quotes
```

In Python 2 the print statement does not require the brackets.

The `+` operator is overloaded for strings, and it results in the concatenation of these objects:

```
> dead, parrot = 'Norwegian', 'Blue'
> print(dead + ' ' + parrot)

Norwegian Blue
```

Concatenation of strings can be achieved with the `+` symbol.

In the example above we have also demonstrated the way in which Python is able to deal with simultaneous assignation in a single line. In other words, the string “Norwegian” is assigned to the variable `dead` and the string “blue” to the variable `parrot`. Note that mixing operators between strings and numbers is not allowed, and an error will be thrown if they are. Instead you will have to convert a number to a string using the `str` function.

Python is able to carry out multiple assignation in a single line. This is part of a Pythonic programming style!

Strings are immutable objects in Python and this means that we cannot change individual elements of a string. We shall discuss more about immutable objects in the context of tuples in [Section 2.2.6](#).

Strings in Python are immutable.

2.2.4 Complex Numbers

PYTHON ALSO SUPPORTS COMPLEX NUMBERS, and it denotes the imaginary number $i = \sqrt{-1}$ as `j`, and so for a number n , `nj` is interpreted as a complex number.

In Python, the imaginary number i is denoted with the letter `j`.

Let us see an example: If we want to define the complex number $z = 2 + 3i$, we simply tell Python the following:

```
> z = 2 + 3j
> print('The real part is {0}, \
      the imaginary part is {1}' \
      .format(z.real, z.imag) )

The real part is 2.0, the imaginary part is 3.0
```

Please note that although the numbers used in the example above are integers, Python recasts them as floating point numbers to suit the complex number object. In the piece of code shown above, we have also demonstrated the fact that we can use the backslash (`\`) to break a line for code readability.

The backslash allows us to break a line.

Remember that each and every entity in Python is an object. Each object has a number of possible actions it is able to perform, i.e., methods. In the example above we have called the `real` and `imag` methods associated with a complex number object to obtain the real and imaginary parts, respectively. Another use of a method is shown in the example above for a string, in that case the `format` method to tell Python how to format the printing of a string.

The method of an object can be invoked by following the name of object with a dot (`.`) and the name of the method.

2.2.5 Lists

A LIST IS PRETTY MUCH self-explanatory: It is a sequence of objects, and these objects can be either of the same type or not. We denote a list with square brackets, `[ ]`. Lists are mutable objects, and therefore it is possible to change individual elements in a list:

```
numbers = [ 1, 3.14, 2.78, 1.61]
expect = ['Spanish', 'Inquisition']
mixed = [10, 8.0, 'spam', 0, 'eggs']
```

A list is denoted by square brackets `[ ]`.

It is possible to refer to elements of a list using an index that corresponds to their position in the list:

```
> print(numbers[0])
1

> print(numbers[1:3])
[3.14, 2.78]
```

We can refer to elements in a list with an index.

Indexing in Python starts with the number zero, and thus the first element of the `numbers` list is referred to as `numbers[0]`. Also, we can refer to a sub-sequence of the list using the colon notation as `start:end`, where `start` refers to the first element we want to include in the sub-sequence, and `end` is the last element we want to consider in the slice.

Indexing in Python starts at zero.

Slicing refers to the subsetting of an array-like object such as lists and tuples.

Remember that Python interprets the slicing operation up to, but not including, the last item in the sequence. In the

example above, Python reads from index 1 and up to index 3 but not including 3. That is why only the second and third elements of the numbers list are returned.

Since lists are mutable objects, it is possible for us to change elements in a list:

```
> expect[0] = 'nobody'
> print(expect)

['nobody', 'Inquisition']
```

We are able to change the elements of a list because they are mutable objects.

We can also add elements to a list with the append method:

```
> numbers.append(1.4142)
> print(numbers)

[1, 3.14, 2.78, 1.61, 1.4142]
```

append lets us add elements to a list.

The new element, 1.4142, is added to the numbers list at the end, increasing the length of the list by one element.

Concatenation of lists is easily achieved with the overloaded + operator:

```
> print(numbers + expect)

[1, 3.14, 2.78, 1.61, 1.4142, 'nobody',
 'Inquisition']
```

The + symbol lets us carry out list concatenation.

Notice that if the two lists are numerical, the result using the + operator is again the concatenation of the list elements, not the sum.

Another useful method of a list is `sort`, which does exactly what we would expect: It allows us to sort the list's values. This method will also enable us to see the difference between mutable and immutable objects in our discussion about tuples ([Section 2.2.6](#)).

The `sort` method allows us to sort a list *in place*.

Let us define a list to work with:

```
> List1 = [3, 6, 9, 2, 78, 1, 330, 587, 19]
> print(List1)

[3, 6, 9, 2, 78, 1, 330, 587, 19]
```

We can now invoke the `sort` method as follows:

```
> List1.sort()
> print(List1)

[1, 2, 3, 6, 9, 19, 78, 330, 587]
```

As we can see, using `sort` with a list results in the elements being ordered in ascending order.

There are a couple of things to note here. First, we have called the `sort` method using the dot (`.`) notation. When executing the first line in the code above, the interpreter does not return any values, and that is a good sign: It means that the method executed correctly. In order to see what happened we issue the second command, which lets us print the contents of `List1`. As shown above, the elements of the list are now ordered.

This takes us to the second point to note. Since lists are mutable, we can change them and in this case the `sort` method has changed the elements in `List1` to be in

Since lists are mutable, we are able to change their elements. In this case sorting the elements for instance.

ascending order. We have sorted the list *in place*. There was no need to create a copy of the list and sort it.

Objects in Python also have functions associated with them. Lists are no exception, and in this particular case there is a sorted function too. The difference is that a function will create a new object. Let us take a look:

Lists have a sorted function.

```
> List1 = [3, 6, 9, 2, 78, 1, 330, 587, 19]
> print(sorted(List1))

[3, 6, 9, 2, 78, 1, 330, 587, 19]
```

So far so good, nothing has changed and we end up with a sorted list. However, let us take a look at the List1 object one more time:

```
> print(List1)

[3, 6, 9, 2, 78, 1, 330, 587, 19]
```

As you can see, the object was not changed! Instead, what the sorted function has done is create a new object with the contents of List1 in ascending order.

The sorted function creates a new object with the elements of the original list, but in ascending order.

We could have assigned the result of the function to a new variable and thus create an object that can be referred to at a later stage.

Incidentally, if you require the elements in descending order, all you have to do is pass the reverse parameter to either the method:

```
> sorted(List1, reverse=True)

[587, 330, 78, 19, 9, 6, 3, 2, 1]
```

or the function:

```
> List1.sort(reverse=True)
> print(List1)

[587, 330, 78, 19, 9, 6, 3, 2, 1]
```

We can also request the sorting to be done in reverse order both for the `sorted` method and the `sort` function.

A very useful pythonic way of constructing lists without the need of a full-blown loop is the so-called *list comprehension*. A typical usage is in the creation of lists whose elements are the result of some operations applied to each member of another sequence or iterable. For example, let us create a string with a sentence:

List comprehension is useful when we need to create a list out of operating on elements of another sequence.

```
> sentence = 'List comprehension is useful'
> print(sentence)

Lists comprehension is useful
```

We can use the string above to create a list of lists with each word in the sentence in upper- and lower-case letters, as well as determining the length of the word. And we can do all this in a single line of code:

```
> words = [[word.upper(), word.lower(), \
            len(word)] for word in sentence.split()]

> print(words)

[['LIST', 'list', 4],
 ['COMPREHENSION', 'comprehension', 13],
 ['IS', 'is', 2],
 ['USEFUL', 'useful', 6]]
```

We are using the string methods `split()`, `upper()` and `lower()` to separate the words in the sentence, and convert them to upper- and lower-case.

Since this is a list of lists, we can request the content of the second element in the list called words:

```
> words[1]

['COMPREHENSION', 'comprehension', 13]
```

You can even ask for the third element of the second component of the words list as `words[1][2]`.

which in turn is a list. This can be verified by checking the type of words with the type function. Try it!

2.2.6 Tuples

A TUPLE MAY BE SEEN as a list by another name: They are also sequences of objects, and they may be of mixed type too. They are indeed closely related to lists and apart from the fact that they are defined with round brackets, (), the main difference is that tuples are immutable.

Tuples are defined with round brackets ().

As we have mentioned above, immutable objects cannot be changed. In other words, we cannot add or remove elements

Tuples are immutable objects.

and thus, unlike lists, they cannot be modified in place. Let us take a look at some tuples:

```
> numbers_tuple = (1, 3.14, 2.78, 1.61)
> expect_tuple = ('Spanish', 'Inquisition')
> mixed_tuple = (10, 8.0, 'spam', 0, 'eggs')
```

As you can see, the only change in the definitions above, compared to the lists in [Section 2.2.5](#), is the use of the round brackets. As with lists, the elements of a tuple can be referred to by their index:

```
> mixed_tuple[4]

'eggs'

> mixed_tuple[0:3]

(10, 8.0, 'spam')
```

Tuples can also be sliced with the help of an index.

Let us see what happens when we try to change one of the elements of a tuple:

```
> expect_tuple[0]='nobody'

TypeError: 'tuple' object does not support item
assignment
```

We are not able to change elements of a tuple as they are immutable objects.

This shows that there are manipulations that are not possible to be done with a tuple. What about sorting? Well, the sorted function still works. Let us define a tuple as follows:

```
> Tuple1 = (3, 60, 18, 276, 87, 0, 9, 4500, 67)
> print(Tuple1)

(3, 60, 18, 276, 87, 0, 9, 4500, 67)
```

We can now apply the sorted function to the tuple:

```
> print(sorted(Tuple1))

[0, 3, 9, 18, 60, 67, 87, 276, 4500]
```

The result of the sorted function on a tuple is a list.

Not too bad, right? But have you noticed something odd? Well, it seems that the result is not a tuple anymore, but a list! We can see that thanks to the square brackets, and we can ensure this by using the type command:

```
> type(sorted(Tuple1))

list
```

This is the result of tuples being immutable: The only way to allow for the elements of the tuple to be ordered is by using the mutable nature of a list. Similarly, since the elements of a tuple cannot be changed, there is no point in having a sort method. Let us have a look:

Since tuples are immutable, we cannot change their elements in place.

```
> Tuple1.sort()

AttributeError: 'tuple' object has no attribute
'sort'
```

As stated by the error returned by Python, tuples do not have a sort attribute.

2.2.7 Dictionaries

WE ARE ALL FAMILIAR WITH the concept of a dictionary:

If we are interested in finding the meaning of a new or unknown word, we simply open up a book (e-book, or access a webpage) that lists words in a specified order (alphabetically, for instance). This enables us to search for the word we are interested in. A dictionary in Python serves the same purpose, and it is composed of *keys* and *values*. Let us take a look.

In comparison with actual dictionaries, keys are equivalent to words and values are the definitions.

A Python dictionary is defined with the use of curly brackets, { }. Furthermore, the key-value pairs are separated by a colon (:) as follows:

We define a dictionary with curly brackets { }.

```
> dictio = {'eggs':1, 'sausage':2,\
           'bacon':3, 'spam':4}
> print(dictio)
{'bacon': 3, 'eggs': 1, 'sausage': 2, 'spam': 4}
```

The keywords in a dictionary can be any immutable Python object including numbers, strings and tuples. The value associated with a particular key can be changed by reassigning the new value to the element of the dictionary with the relevant entry. For instance, in our example above we can see that the value to the key spam is 4:

The keys can be any immutable object: numbers, strings or tuples for example.

```
> print(dictio['spam'])
```

```
4
```

We can change the value of this key by simply reassigning any new value. We can for example reassign the value associated to the spam key:

```
> dictio['spam']='Urggh'
> print(dictio['spam'])

'Urggh'
```

The values in a dictionary can be modified.

This can be done repeatedly:

```
> dictio['spam']='Lovely spam'
> print(dictio['spam'])

'Lovely spam'
```

The modification can be done as many times as required.

It is possible to access the keys and values in the form of straight lists with the aid of the `keys()` and `values()` methods:

```
> print(dictio.keys())

dict_keys(['spam', 'bacon', 'eggs', 'sausage'])

> print(dictio.values())

dict_values(['Lovely spam', 3, 1, 2])
```

A list of dictionary keys can be obtained with the `keys()` method. Similarly, `values()` returns a list of values in a dictionary.

We can also obtain the key-value pairs in the form of a list of tuples with the `items()` method:

```
> print(dictio.items())

dict_items([('spam', 'Lovely spam'), ('bacon', 3),
('eggs', 1), ('sausage', 2)])
```

Note that the result of `items()` is a list of tuples.

Finally, it is possible to get rid of key-value pairs with the use of the `del` function:

```
> del dictio['bacon']
> print(dictio)

{'sausage': 2, 'eggs': 1, 'spam': 'Lovely spam'}
```

We can remove entries from a dictionary with `del`.

2.3 *Control Flow*

NOT ONLY IS IT IMPORTANT to understand the types and objects that are available in any programming language, but also as important is to understand how to control the flow of a programme. In this manner we are able to follow the logic behind how the programme itself is organised, in other words, the order in which the individual statements are executed and the results we expect to obtain.

In particular it is important to mention that in Python the whitespace is a meaningful character, as it enables the definition of blocks of code by having the same level of indentation. Let us see some typical structures to control the flow of a programme in Python.

Whitespace is a meaningful character in Python.

2.3.1 Conditional Statements — `if... elif... else`

CONDITIONAL BRANCHING ENABLES US TO perform different actions depending on the result of boolean operations. If a condition is met, then we apply an action; otherwise, a different one is performed. In Python we can do this as follows:

```
if condition1:
    block of code executed
    if condition1 is met
elif condition2:
    block of code executed
    if condition2 is met
...
elif conditionN:
    block of code executed
    if conditionN is met
else:
    block of code executed
    if no conditions are met
```

The `if... elif... else...` lets us test various conditions and create branches for our code.

As you can see, each block of code is indented at the same level. Also, notice that it is possible to nest various conditions with the help of the `elif` reserved word. The conditions are logical expressions that can test for scalar comparison and thus we can use any of the comparison operators listed in [Table 2.2](#). Let us see an example:

The conditions to test are logical expressions that evaluate to `True` or `False`.

Operation	Operator
Equal	==
Different	!=
Greater than	>
Less than	<
Greater than or equal to	>=
Less than or equal to	<=
Object identity	is
Negated object identity	is not

Table 2.2: Comparison operators in Python.

```
> Age = 40
if Age > 50:
    print('A wise person')
else:
    print('Such a youngster')

Such a youngster
```

Here we test if the Age variable is greater than 50, in which case we print one message; otherwise, we print another one.

Finally, remember that the conditions are tested one-by-one in the order they are provided in the code. If a condition is met, the rest of the tests are not executed.

2.3.2 while Loop

A while LOOP IS USED when we need to repeat a block of code until a condition is no longer met. The structure of a while loop in Python is as follows:

```
while logical_test:
    block of code to be executed
    don't forget to update the test variable
```

The structure of a while loop in Python.

An important thing to remember is that at the very beginning of the while loop, the logical test must evaluate to True; otherwise, the block of code is never executed. Also, in order to avoid infinite loops we need to update the control variable inside the block of code.

The while loop requires a logical test at the beginning of the block.

We can see how this works by counting down from 10:

```
> countdown = 10
while countdown >= 0:
    print(countdown)
    countdown -= 1

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

Note that `countdown -= 1` is a shorthand for `countdown = countdown - 1`.

2.3.3 for Loop

TYPICALLY, A while LOOP IS used in cases when we do not know in advance how many times the block of code will need to be executed. If we know how many iterations are needed, we can use a for loop. In Python, a for loop iterates over a sequence: A list, tuple or a string for example.

A for loop is useful when we know how many times the code needs to be repeated.

```
for item in sequence:
    block of code to be
    executed
```

This is the same basic structure used in list comprehension.

The example we used for the while loop in [Section 2.3.2](#) can be written with a for loop as follows:


```
> countdown_list = [10, 9, 8, 7, 6, \
    5, 4, 3, 2, 1, 0]
    for x in countdown_list:
        print(x)

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

Notice the colon and indentation in the `for` block of code.

We could simplify the example above by avoiding the explicit definition of the list and instead define a range:

```
> for x in range(10, -1, -1):
    print(x)

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

`range` enables us to define a sequence of numbers as an object. This means that the values are generated as they are needed.

In the example above we used the `range(start, end, step)` function to generate a sequence of numbers from `start` to `end-1` in steps given by `step` to go from 10 to 0 in steps of `-1`.

2.3.4 *Exception handling* — `try... except`

IT IS NOT UNUSUAL TO have syntactically correct blocks of code with statements that in certain cases may cause an error during execution. These errors are not necessarily fatal in the execution of a programme, and instead they are anomalous or exceptional cases that require special processing.

Syntactically correct code may cause errors during execution.

Instances such as those described above are called *exceptions*, and when they happen, we are interested in catching them

and taking appropriate action, for example by generating an error message. This is what is known as *exception handling*. In Python this can be done with the `try...except` structure as follows::

```
try:
    Block of code that may raise an
    exception
except exception1:
    Block of code to run if exception1
    is raised
except exception2:
    Block of code to run if exception2
    is raised
...
except exceptionN::
    Block of code to run if an exceptionN
    is raised
```

Exception handling in Python can be done with the `try... except` structure.

In the structure above `exception1`, `exception2`, ..., `exceptionN` are either standard exceptions that Python knows about, or exceptions you can define yourself. Standard exceptions are detailed in the appropriate documentation; we list some common ones in [Table 2.3](#).

For further information about standard exceptions see <https://docs.python.org/3.12/library/exceptions.html>

For example, we can try to calculate the reciprocal of the elements of a list and print each of the values. However, if the sequence contains the number zero, we can try to catch the exception with `ZeroDivisionError`:

Standard Exception	Meaning
ArithmeticError	Arithmetic error
FloatingPointError	Floating point operation failure
IOError	Input/Output (I/O) operation error
IndexError	Sequence subscript out of range
KeyError	Dictionary key not found
TabError	Inconsistent use of tabs/spaces
UnicodeError	Unicode-related error
ZeroDivisionError	Division by zero

Table 2.3: Standard exceptions in Python.

```
> class OtherError:
    pass

> for x in range(3,-2,-1):
    try:
        print('The reciprocal of {0} is {1}'.\
            format(x, 1.0/x))
    except ZeroDivisionError:
        print('Divide by zero? \
            You can't do that!!')
    except OtherError:
        print('Other')

The reciprocal of 3 is 0.333333333333
The reciprocal of 2 is 0.5
The reciprocal of 1 is 1.0
Divide by zero? You can't do that!!
The reciprocal of -1 is -1.0
```

We are using the ZeroDivisionError exception to handle this particular exception.

There are several things to unpack in the code shown above. First, we have defined a class called `OtherError` to capture a general error not defined in the standard exceptions. Note that using classes is part of the use of Python as an object-oriented programming language; given the scope of this chapter — and of the book — we will not be able to give much cover on them.

Since Python is an object-oriented programming language, we are able to define classes.

Second, the `for` loop goes over the range from 3 to `-1`; we calculate the reciprocal of each number and print it. However, when we reach the value 0 we will run into a problem. In this case, however, we try to handle this with the `except` entry in a sensible manner and without halting the execution of the programme. In this way, the `for` loop continues until the final value in the range is processed. Finally, notice the use of error class we defined.

Remember that range covers up to end `-1`.

2.3.5 Functions

NOW THAT WE HAVE COVERED some of the elementary control flow structures in Python, we can start combining them into logical blocks to carry out specific tasks. In particular we can construct pieces of code that can be used when necessary and whose outcome depends on the input parameters provided. In other words, we are talking about functions.

A function is a good way to write code that can be repeated, and whose outcome typically depends on inputs provided.

Sometimes Python is described as having “batteries included”, and this is because it has access to a Standard Library which is a collection of built-in functions and modules to make some tasks easier for us. We have been

Read more about modules and the Standard Library in [Section 2.3.6](#).

using some of these built-in functions already, for instance `print()` or `range()`. In Table 2.4 we include some common Python built-in functions.

For further information about built-in functions, see <https://docs.python.org/3/library/functions.html>.

Built-in Function	Description
<code>abs()</code>	Absolute value
<code>chr()</code>	String representing a Unicode integer
<code>help()</code>	The built-in help system
<code>input()</code>	Request input from user
<code>len()</code>	Length (the number of items) of an object.
<code>max()</code>	Largest item in an iterable.
<code>min()</code>	Smallest item in an iterable.
<code>pow()</code>	Exponentiation.
<code>print()</code>	Print objects to the text stream.
<code>range()</code>	Immutable sequence
<code>sum()</code>	Sum of the items of an iterable.
<code>zip()</code>	Parallel iteration over multiple iterables.

Table 2.4: Some built-in functions in Python.

A function in Python has the following syntax:

```
def my_function(arg1, arg2=default2,... \
                argn=defaultn):
    ''' Docstring (optional) '''

    instructions to be executed
    when executing the function

    return result # optional
```

The function definition starts with the word `def`. Remember that code needs to be indented.

Notice that the function definition starts with the reserved word `def`, and the block of code inside the function is indicated with appropriate indentation.

The input parameters for the function are the dummy variables `arg1, arg2, ... , argn`, and as you can see it is possible to define default values for some of these parameters. Parameters with default values must be defined last in the argument list.

The second line in the function definition is called the *documentation string*, and its purpose is to describe the actions that are performed by the function. Finally, notice that it is not necessary for a function to return a result.

A documentation string enables us to provide information about what a function does. Make sure you use it!

Let us define a function to calculate the area of a rectangle sides *a* and *b*:

```
def rect_area(a, b=1.0):
    '''Calculate the area of a rectangle'''
    return a*b
```

We are defining a default value for the parameter *b*.

Notice that the parameter *b* has been given the default value of 1.0. If we were to call this function with only one parameter, the function will know how to handle the calculations and use the default values when needed.

```
> c = rect_area(20, 2)
> print(c)

40
```

We can call the function in the same way we do any other built-in function in Python.

In the first line of code above, we are calling the `rect_area` function with two parameters, such that we assign the value 20 to *a* and override the default value of *b* with 2. As

expected, the area calculated is 40. Let us try providing only one single value to the function:

```
> c2 = rect_area(42.4)
> print(c2)

42.4
```

Here we have only passed the value 42.4 to the function. In this case the value is assigned to a and the default value of $b = 1.0$ is used in the calculation.

We can include control flow structures in our programmes to make them more useful and flexible. Let us for instance implement a simple function to calculate the factorial of a number:

```
def factorial(n):
    '''Return the factorial of n'''
    f = 1
    if n<=1:
        return f
    else:
        while n>0:
            f *= n
            n -= 1
        return f
```

A function can use any of the other control flow structures of the language.

$\ast=$ and $-=$ indicate repeated operations with the left-hand-side value.

When we pass a number smaller than or equal to the one the function expects, it returns the value 1; and when the number is greater than 1, the factorial is calculated with a while loop. Let us use the function:

```
> print(factorial(-3))  
1  
  
> print(factorial(5))  
120
```

There may be times when it is more convenient to define a simple function on-the-fly, without having to resort to a full `def` structure. In these cases we can exploit the use of the so-called `lambda` functions:

A `lambda` function in Python is an anonymous function created at runtime.

```
lambda arg1, arg2, ... : statement
```

where, as before, `arg1, arg2, ...` are the input parameters, and `statement` is the code to be executed with the input provided.

For example, if we needed to calculate the cube of a list of numbers we could try the following code:

```
x = [1, 3, 6]  
g = lambda n: n**3
```

In this case the object `g` is a `lambda` function that can be called as any other function in Python.

So far, nothing too strange: We have initialised a list with the numbers 1, 3 and 6, and then defined a `lambda` function that calculates the cube of the argument `n`. We can now apply this function, for example:


```
> [g(item) for item in x]

[1, 27, 216]
```

Lambda functions may seem very simple, but it is that simplicity that provides their strength, as it is shown above. This can be seen employed for instance in the implementations of PySpark, the Python Application Programming Interface (API) for Spark, an open-source cluster computing framework.

Lambda functions are very useful in frameworks such as Spark.

2.3.6 *Scripts, Modules and the Standard Library*

WITH THE FLEXIBILITY PROVIDED BY the possibility of controlling the flow of a set of instructions, and the repeatability offered by constructing our own functions, it becomes imperative to be able to store programmes in a way that enable us to use and reuse code.

In Python we are able to do this by saving the instructions that make up a programme in a plain text file saved with the extension `.py`. Furthermore, if we use the interactivity provided by the Jupyter Notebook, it is also possible to save our notebooks in a JSON-formatted notebook with the extension `.ipynb`.

Python scripts have the extension `.py`, whereas notebooks have the extension `.ipynb`.

It is then possible to execute a Python script from the command line by calling Python followed by the name of the script to be executed. For instance, we can create a script defining a `main` function and a call to it. We can save the

function in a script called `firstscript.py` with the following contents:

```
def main():
    '''Print the square of a list of
    numbers from 0 to n'''
    n = input('Give me a positive number')
    x = range(int(n)+1)
    y = [pow(item, 2) for item in x]

    print(y)

main()
```

We are defining a main function in this programme and calling simply with the command `main()`.

In this case we are asking the user for a number n with the built-in function `input()`. We then use this number to calculate a sequence given by the square of the numbers from 0 to n and assign it to the variable `y`. Finally we simply print the list stored in `y`.

Notice that we have used `n+1` for range, and used the `pow()` built-in function to calculate the square. Alternatively, we could have used `item**2`.

Remember that we have said that the script above has been created, but we have not executed it. We can do this by typing the following command in a terminal in the appropriate path:

```
> python firstscript.py

Give me a positive number: 4

[0, 1, 4, 9, 16]
```

In this case we have given the value $n = 4$ as an input.

This is perhaps not the most advanced algorithm to implement, but we can surely see the possibilities. In particular, we can see how we can create scripts to add further functionality to our code, and as such the concept of a module becomes natural.

A module is a file or collection of files containing related Python functions and objects to achieve a defined task. A package is a collection of modules, and they allow us to extend the capabilities of the language, and create programmes that enable us to carry out specific tasks. Any user is able to create their own modules and packages and make them available to others. Some of these packages are readily available for us to be used, and once appropriate installation is done all we need to do is import them whenever we need to use them.

For example, we can use the `math` module to access some common mathematical functions. Let us create for instance a script that implements a function to calculate the area of a circle. In this case we will need the mathematical constant π to carry out the calculations:

```
import math
def area_circ(r):
    return math.pi * r**2

r=3
Area = area_circ(r)
print('The area of a circle with ' + \
      'radius {0} is {1}'.format(r, Area))
```

A module is a file containing related Python functions to achieve a specific task. A package is a collection of modules.

The `math` module contains some common mathematical functions.

We can use the value of π with `math.pi`.

Running the programme will result in the following output:

```
> python area_circ.py
```

```
The area of a circle with radius 3 is 28.2743338
```

Notice that we need to tell the Python interpreter that the constant π is part of the `math` module by using the syntax `math.pi`. In the example above, we are importing all the functions of the `math` module. This can be somewhat inefficient in cases where only specific functionality is needed. Instead, we could have imported only the value of π as follows:

```
from math import pi
```

In some cases it may be more efficient to load only the needed functionality from a module.

A large number of modules are available from the Python Standard Library and more information can be found in <https://docs.python.org/3/library/>. Since we all can create our own modules, we are also able to make them available to others, and that is what makes Python a powerful language to use in data science, analytics, machine learning and artificial intelligence, as well as in many other areas. Distributions such as Anaconda come already with a number of useful modules and packages available to us. In the rest of the book we will deal with a few of these packages, and in the rest of this chapter we will introduce a few, namely NumPy, SciPy, pandas, Matplotlib and

See the current Anaconda package list here <https://docs.anaconda.com/free/anaconda/pkg-docs/>.

Scikit-learn. For other packages you may want to check the Python Package Index (<https://pypi.org>).

2.4 *Computation and Data Manipulation*

WITH THE PROGRAMMING STRUCTURES DISCUSSED so far we are ready to take up a large number of tasks, not only in data science, machine learning and artificial intelligence, but also in more general settings. In the next chapter we will cover the differences and overlaps in these areas. As we shall see in the rest of the book, however, computation with data and its manipulation can be managed more effectively and easily with the aid of linear algebra. In this section we will address some basic concepts in both data manipulation and linear algebra with Python.

Data manipulation and computation is a very important step in the data science and analytics workflow.

2.4.1 *Matrix Manipulations and Linear Algebra*

AS WE HAVE MENTIONED ABOVE, linear algebra enables us to carry out computational tasks with data in a very effective way. It also provides a compact notation to express the type of manipulations we need to do to our data, from pre-processing to presenting results. The use of vectors and matrices is therefore a very important area to cover. Vectors and matrices are arrays of numerical objects with a defined set of operations such as addition, subtraction, multiplication, etc.

Linear algebra provides us with an efficient and compact way to carry out complex calculations.

An $m \times n$ matrix is a rectangular array of numbers having m rows and n columns. In particular when $m = 1$ we have

a column vector, and when $n = 1$ we have a row vector. In general, a matrix $\mathbf{A}$ can be represented as follows:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}. \quad (2.1)$$

A matrix can be thought of as a collection of row (or column) vectors.

A Python object that may come to mind when thinking about using arrays is the list. For example, we can create two lists as follows:

See [Section 2.2.5](#) for a discussion about lists.

```
a = [1, 2, 3, 4, 5]
b = [20, 30, 40, 50, 60]
```

However, remember that Python considers these objects as lists, and that each type of object has a defined set of operations. For instance, if we tried to add these two arrays in the mathematical sense, we will find that Python returns an unexpected answer:

```
> a + b

[1, 2, 3, 4, 5, 20, 30, 40, 50, 60]
```

Using the `+` symbol with lists results in concatenation.

Instead of adding the elements of each of the two vectors, Python concatenated the lists. This works because Python has overloaded the `+` symbol, but an error would be returned if we tried to use subtraction or multiplication.

We covered list concatenation in [Section 2.2.5](#).

```
> a - b
```

```
TypeError: unsupported operand type(s)
for -: 'list' and 'list'
```

Using other arithmetic symbols with lists results in an error.

It is clear that a list is a good start for the operations we need to execute, and the use of the programming capabilities of Python would enable us to build functions to define mathematical operations on lists to construct arrays. Although it may be a very good programming practice, rather than building our own functions for this purpose, we can instead exploit the packages that are available to us within Python such as SciPy which provides an ecosystem for mathematics, science, and engineering, and in particular NumPy, a package that supports N -dimensional arrays.

Some useful Python modules for array calculations are NumPy and SciPy.

2.4.2 NumPy Arrays and Matrices

NUMPY EXTENDS THE TYPES SUPPORTED by Python with the definition of arrays to describe a collection of objects of the same type. The dimension of a NumPy array is defined by a tuple of N positive integers called the *shape* of the array. We can think of arrays as an enhancement on lists and as such we can create arrays with the help of lists:

```
import numpy as np

A = np.array([1, 2, 3, 4, 5])
B = np.array([20, 30, 40, 50, 60])
C = A + B
```

NumPy extends the types in Python by including arrays.

We define a NumPy array with `np.array`, where `np` is a convenient alias used for the NumPy package.

In the small piece of code above, we are importing the NumPy package and using the alias `np` to refer to the module. With the aid of the `array` command in NumPy, we transform a list into an array object. If we were to print the content of the array `C`, we would obtain the following:

```
> C

array([21, 32, 43, 54, 65])
```

The use of the `+` symbol with the arrays defined above results in their addition as expected.

Notice that in this case Python has indeed added the arrays element by element as expected. In the example above we could have simply used the list defined in the previous section and written the following:

```
A = np.array(a)

B = np.array(b)
```

As we have mentioned above, NumPy extends the functionality of lists in Python to be able to carry out vector arithmetics such as:

- Vector addition: `+`
- Vector subtraction: `-`
- Element-wise multiplication: `*`
- Scalar product: `dot()`
- Cross-product: `cross()`

These are some of the vector operations that are supported by NumPy arrays.

You may have noticed that we have been referring to vector operations, but what about matrices? NumPy supports matrices too.

```
M1 = np.matrix([[2, 3], [-1, 5]])

M2 = np.matrix([[1, 2], [-10, 5.4]])
```

We can define matrices with the help of `np.matrix`.

In this case we are using the command `matrix` to define the objects, and the result of the multiplication is as expected:

```
> M1 * M2

matrix([[ -28. ,  20.2],
        [ -51. ,  25. ]])
```

We can multiply NumPy matrices with the usual multiplication symbol.

An alternative for defining NumPy matrices is to use the `mat` command to recast NumPy arrays.

A widely used operation in linear algebra is the transposition of a matrix. This can be readily accomplished with the use of the `transpose` command:

```
> M2.transpose()

matrix([[ 1. , -10. ],
        [ 2. ,  5.4]])
```

Finally, with the SciPy package we can use the `linalg` methods that will enable us to do some typical linear algebra computations such as matrix inversion:

Linear algebra methods are included in `linalg` inside SciPy.

```

from numpy import array, dot
from scipy import linalg

x = array([[1, 1], [1, 2], [1, 3], [1, 4]])
y = array([[1], [2], [3], [4]])

n = linalg.inv(dot(x.T, x))
k = dot(x.T, y)

coef = dot(n,k)

```

We can invert a matrix with the `.inv` method.

Matrix multiplication with arrays can be done with the `dot()` function.

In the code above we have defined a couple of arrays, x and y . We have then calculated $n = (x^T x)^{-1}$ with the help of the `.inv` command from the linear algebra module. Note that the command `.T` in the code returns the transpose of a matrix.

Finally, we then calculated $k = x^T y$, and finally $coef = nk = (x^T x)^{-1} x^T y$. Let us take a look at the result:

We shall come back to this calculation in the context of regression in [Chapter 4](#).

```

> print(coef)

[[-4.4408921e-16]
 [ 1.0000000e+00]]

```

Notice that the result is a vector, and we have deliberately called the result `coef`, as we can think of the result of this simple calculation as the coefficients of a linear regression using the arrays x and y . We will come back to this result in [Chapter 4](#) in the context of regression.

2.4.3 *Indexing and Slicing*

AS IT IS THE CASE with lists, it is possible to access the contents of an N -dimensional array by indexing and/or slicing the array. We can do this using the usual notation `start:end:step` which will extract the appropriate elements starting at `start` in steps given by `step` and until `end-1`.

Arrays and matrices can be indexed and sliced with the usual colon notation for lists and tuples.

```
> a = np.arange(10)
> print(a[2:6]); print(a[1:9:3])

[2 3 4 5]
[1 4 7]
```

In the example above, we are selecting first the elements from 2 and up to but not including 6. We then ask for the elements from 1 through to 8 in steps of 3.

The same notation can be used with arrays of more dimensions. Let us see an example:

The same applies to arrays of more than 1 dimension.

```
> b = np.array([np.arange(4), 2*np.arange(4)])
> print(b.shape)

(2,4)
```

With the `shape` command, we can see the dimensions of matrices and arrays.

The array `b` above is a 2×4 array as can be seen from the `shape` command. We can select all the elements in row zero as follows:

```
> print(b[:1, :])

[[0, 1, 2, 3]]
```

We are using the colon notation to slice the array. `:1` refers to the zero-th row, whereas `:` indicates all columns.

So far so good, but until now the arrays, matrices and vectors that we have been dealing with have been numerical. However, in many situations the data that we have to deal with is not necessarily all numbers of a single type. There is therefore a need to find a way to accommodate the manipulation of disparate data types, including categorical and text data. In cases like that, the capabilities of NumPy are restricted; nonetheless, Python can still help as we shall see in the next section.

The arrays we have discussed so far have all been numerical. In many cases we need to deal with different data types, and Python can still help.

2.5 *Pandas to the Rescue*

YOU MAY BE THINKING THAT we have lost the plot and that, in the style of the Monty Python troupe, we are simply listing animal names as part of a sketch. You would be wrong, as pandas is actually a powerful library that enables Python to work with structured datasets using panel data or dataframes. Pandas⁴ started life as a project by Wes McKinney in 2008 with the aim of enabling Python to be a more practical statistical computing environment.

⁴ McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media.

Pandas is a great addition to the Python stack. It allows us to manipulate indexed structured data with many variables, including work with time series, missing values and multiple datasets. In pandas, a 1D array is called a *series*,

whereas dataframes are collections of series. The rich assortment of data types that can be held by a dataframe, together with the manipulations that it enables, makes pandas an indispensable tool for the Jackalope Data Scientist.

In some sense, we can think of a pandas series as an extension of a NumPy array, and indeed we can use them to initialise a series:

```
import numpy as np
import pandas as pd

array1 = np.array([14.1, 15.2, 16.3])

series1 = pd.Series(array1)
```

Pandas is a powerful library that enables us to carry out complex data manipulation in a very straightforward manner.

A typical alias for the pandas library is `pd`.

We could also have used a list or a tuple for initialisation. A very useful feature of pandas is the ability of using indices and column names to refer to data. Let us consider the data shown in [Table 2.5](#) for some animals detailing their number of limbs and dietary habits:

Animal	Limbs	Herbivore
Python	0	No
Iberian Lynx	4	No
Giant Panda	4	Yes
Field Mouse	4	Yes
Octopus	8	No

Table 2.5: Sample tabular data to be loaded into a pandas dataframe.

We can load this data into Python by creating lists with the appropriate information about the two features describing the animals in the table.

```
features = {'limbs':[0,4,4,4,8],\
            'herbivore':['No','No','Yes','Yes','No']}\

animals = ['Python', 'Iberian Lynx',\
            'Giant Panda', 'Field Mouse', 'Octopus']

df = pd.DataFrame(features, index=animals)
```

We can load data into a pandas dataframe with lists, dictionaries, arrays, tuples, etc.

Note that we have defined the features `limbs` and `herbivore` from [Table 2.5](#) as a dictionary, where the keys will be the names of the columns in our pandas dataframe, and the values correspond to the entries in the table. Similarly, we are defining a list called `animals` that will be used as an index to identify each of the rows in the table.

We can have a look at the first three entries in the dataframe `df` with the command `head`:

```
> df.head(3)
```

	limbs	herbivore
Python	0	No
Iberian Lynx	4	No
Giant Panda	4	Yes

The `head` method lets us see the first few rows of a dataframe. Similarly, `tail` will show the last few rows.

As we mentioned above, we can refer to the column data by the name given to the column. For instance, we can retrieve

the data about the number of limbs of rows 2 through to 4 using the following command:

```
> df['limbs'][2:5]
```

Giant Panda	4
Field Mouse	4
Octopus	8

We can view the contents of a dataframe column by name, and the data can be sliced with the usual colon notation.

Notice that we have referred to the name of the column as a string. Furthermore, we have used slicing to select the data required.

Similarly, the information about a single row can be obtained by locating the correct index:

```
> df.loc['Python']
```

limbs	0
herbivore	No
Name: Python, dtype: object	

```
> df.loc['Iberian Lynx']
```

limbs	4
herbivore	No
Name: Iberian Lynx, dtype: object	

The content of a row can be retrieved with the `.loc` method.

There are a number of very useful commands in pandas that facilitate various tasks to understand the contents in a dataframe. For example, we can get a description of

the various columns. If the data is numeric, the `describe` method will give us some basic descriptive statistics such as the count, mean, standard deviation, etc.

```
> df['limbs'].describe()
```

```
count    5.000000
mean     4.000000
std      2.828427
min      0.000000
25%      4.000000
50%      4.000000
75%      4.000000
max      8.000000
```

The `describe` method provides us with descriptive statistics of numerical data.

Whereas if the data is categorical, it provides a count, the number of unique entries, the top category, etc.

```
> df['herbivore'].describe()
```

```
count      5
unique     2
top        No
freq       3
```

We can also obtain useful information of categorical data with `describe`.

It is very easy to add new columns to a dataframe. For example, we can add a class to our data above as follows:

```
df['class']=['reptile','mammal','mammal',\
            'mammal','mollusc']
```

Adding columns to a pandas dataframe is very easy.

Let us look at the result:

```
> df
```

	limbs	herbivore	class
Python	0	No	reptile
Iberian Lynx	4	No	mammal
Giant Panda	4	Yes	mammal
Field Mouse	4	Yes	mammal
Octopus	8	No	mollusc

Pandas also allows us, among other things, to create groups and aggregations. Let us group the dataframe by the herbivore column:

```
> grouped = df.groupby(df['herbivore'])
> grouped.groups

{'No': ['Python', 'Iberian Lynx', 'Octopus'],
 'Yes': ['Giant Panda', 'Field Mouse']}

> grouped.size()

herbivore
No      3
Yes     2
```

pandas allows us to group data and create aggregations. The method `.groups` contains the grouped information, `.size` returns a simple count.

We can also apply aggregation functions. Let us try to calculate the average number of limbs for herbivores and carnivores in our dataset:

```
> from numpy import mean
> limbs = df['limbs'].groupby(df['herbivore'])\
    .aggregate(mean)
> print(limbs)
```

```
herbivore
```

```
No      4
```

```
Yes      4
```

In this case we are applying the mean function from NumPy to calculate the average per group in our dataset.

In the example above, we used Python itself to input data into a pandas dataframe. Although this is possible for a small dataset, in reality you may be interested to ingest data from other sources. Fortunately pandas has a very robust input/output ecosystem and is able to take data from a myriad of sources. [Table 2.6](#) lists some of them:

We can import data into a pandas dataframe from a variety of sources.

Source	Command
Flat file	read_table
	read_csv
	read_fwf
Excel file	read_excel
	ExcelFile.parse
JSON	read_json
	json_normalize
SQL	read_sql_table
	read_sql_query
	read_sql
HTML	read_html

Table 2.6: Some of the input sources available to pandas.

Pandas is a very versatile and rich tool, and we have only touched the surface in this brief discussion. We will be

using pandas extensively in the rest of the book, and whenever possible we will provide explanations to aid the discussion. Nonetheless, we urge you to take a deeper look into this great library.

Pandas is a very versatile library, and we will continue using it in the rest of the book.

2.6 *Plotting and Visualising: Matplotlib*

THEY SAY THAT A PICTURE is worth a thousand words, and data visualisation makes the case quite emphatically. There are a number of tools that enable data visualisation in the context of business intelligence such as Tableau and QlikView or Power BI. In Python, there are some really good modules that support very nice visuals such as Seaborn, or interactivity such as Bokeh. For our purposes we will concentrate on the robustness provided by Matplotlib and its MATLAB-style API called Pyplot.

Python is able to create plots and graphs. Here we will cover some of Matplotlib's functionality.

In a good Pythonic style, Matplotlib is an object-oriented plotting library that can generate a variety of visualisations: Simple plots, histograms, bar charts, scatter plots and more with a few lines of code. If you are familiar with MATLAB or Octave, you will find Pyplot very easy to use. Let us start by importing the modules:

```
import numpy as np
import matplotlib.pyplot as plt
```

Matplotlib is an object oriented plotting library. Pyplot is a MATLAB and Octave inspired API for Matplotlib.

In a Jupyter Notebook you can use the magic command `%matplotlib inline` to load NumPy and Matplotlib.

Let us create a simple figure to plot the following functions:

$$y_1 = x^2, \quad (2.2)$$

$$y_2 = x^3. \quad (2.3)$$

With the aid NumPy we can create a vector with entries for x and calculate y_1 and y_2 :

```
x = np.linspace(-5, 5, 200)
y1 = x**2
y2 = x**3
```

The command `linspace` lets us create an equally spaced vector with a specified number of points.

We can create a plot using the `plot` command as follows:

```
fig, ax = plt.subplots()
ax.plot(x, y1, 'r', \
        label=r'$y_1 = x^2$', linewidth=2)
ax.plot(x, y2, 'k--', \
        label=r'$y_2 = x^3$', linewidth=2)
ax.legend(loc=2) # upper left corner
ax.set_xlabel(r'$x$', fontsize=18)
ax.set_ylabel(r'$y$', fontsize=18)
ax.set_title('My Figure')
plt.show()
```

The commands to create a plot are very similar to those in programming languages such as Octave or MATLAB.

Remember that Matplotlib is an object-oriented library, and thus we are using objects to create our plots. You may want to take a look at resources for MATLAB and Octave,⁵ and you will see that the commands above are very similar to those used in those programming languages. The result of the commands above can be seen in [Figure 2.1](#).

⁵ Rogel-Salazar, J. (2014). *Essential MATLAB and Octave*. Taylor & Francis.

Finally, you may want to create a file with the plot itself to be shared and not just be shown on your computer screen. You can save the plot to a file with a single command. In

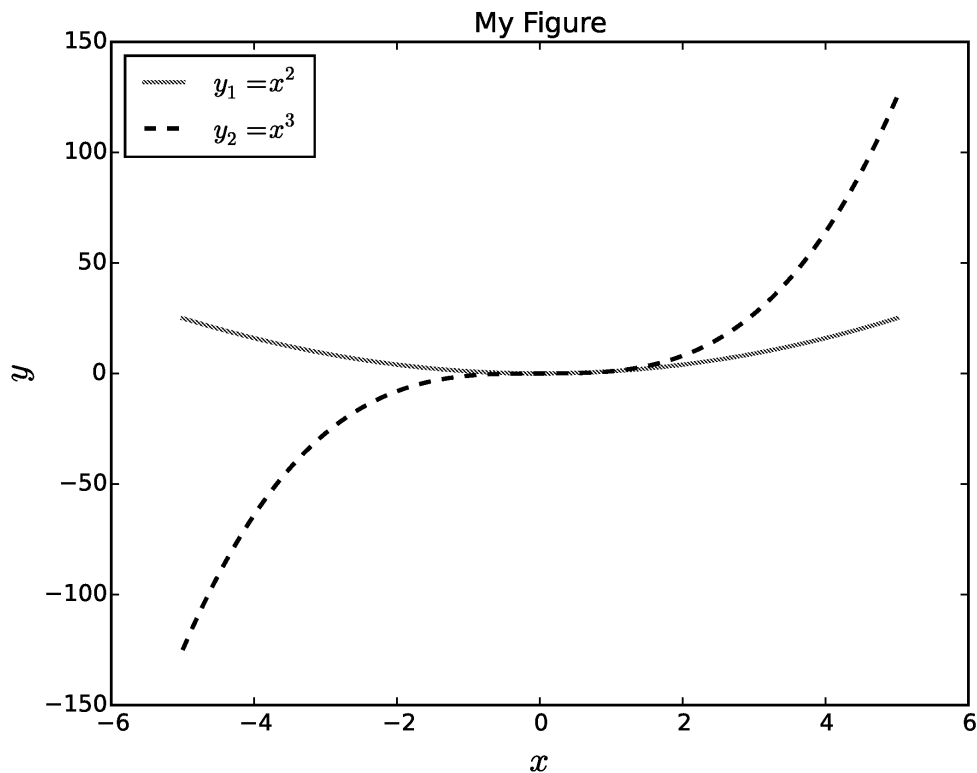


Figure 2.1: A plot generated by Matplotlib.

this case, we can create a PNG file with the following line of code:

```
> fig.savefig('firstplot.png')
```

2.7 Virtual Environments

AS WE HAVE SEEN, PYTHON is a very versatile and flexible programming language. That versatility is enhanced by the

Python is versatile, but we need to be mindful of the versions of the language and packages we use.

number of modules and packages we can use. Although Python scripts are multi-platform and can be executed without change from one machine to another, it is important to consider whether we have the same versions of the packages used during development. If you write a programme with Python 3.8, someone with Python 2.0 may not be able to execute it. Furthermore, if you implement for example a classification algorithm using a particular package using version 1.4, someone with an different version (older or newer) may encounter inconsistencies or incompatibilities.

See [Chapter 6](#) about classification algorithms.

One option may be to run multiple versions of a package at the same time, but sadly that is not possible. Instead, a way to deal with this is to use isolated context to keep dependencies required by the programmes we run. These isolated contexts are called *virtual environments*, and they help us avoid conflicting dependencies. Virtual environments comprise two important components: First, the Python interpreter that the virtual environment runs, and second a unique folder that contains the modules and packages installed in the virtual environment. These libraries, like silent guardians, stand at the ready, prepared to serve the Python interpreter's every beck and call when the virtual environment is activated.

A virtual environment is an isolated context with to keep the dependencies of our programmes.

You may be wondering what sets these virtual environments apart from one another. In essence, it is their isolation: Each virtual environment exists as an independent entity, separate from others and in that way, the packages installed within one virtual environment hold no sway over those

Each virtual environment is independent of any others, including system-wide libraries.

nestled within another, nor do they dare to tamper with the realm of system-wide libraries.

Using a virtual environment is a very smart way to maintain dependencies for a project. As explained above, a virtual environment is like a “special insulated folder” that lets us isolate projects, manage dependencies, and ensure that each project has its own specific Python interpreter and libraries and as a result help us minimise reproducibility issues. Virtual environments and containers, like Docker, serve similar but different purposes in the context of software development and deployment. Let us take a look.

A virtual environment ensures that each project has its own specific Python interpreter and libraries.

Virtual Environments:

Features of virtual environments.

- **Isolation at the Python Level:** Virtual environments provide isolation at the level of Python packages and libraries. They allow you to create separate environments for different projects, each with its own set of dependencies.
- **Lightweight:** Virtual environments are relatively lightweight compared to containers. They don’t include an entire operating system (OS); they rely on the host system’s OS.
- **Python-Centric:** Virtual environments primarily focus on managing Python dependencies. They don’t encapsulate the entire application and its runtime environment, only the Python-related aspects.
- **Sharing with Third Parties:** While you can share a virtual environment by packaging it and sending it to a

third party, it's not as self-contained as a container. It may still rely on the recipient having a compatible Python interpreter installed.

Containers (e.g., Docker):

Features of containers.

- **Isolation at the System Level:** Containers provide full system-level isolation. They encapsulate the entire runtime environment, including the application, its dependencies, libraries, and even the OS!
- **Heavier:** Containers are more heavyweight than virtual environments because they include an entire OS, which can make them less resource-efficient for some use cases.
- **Application-Agnostic:** Containers are not tied to a specific programming language or runtime. You can run applications developed in various languages within the same container.
- **Self-Contained:** Containers are highly portable and self-contained. You can package everything needed to run an application into a container image, making it easy to share with third parties. The recipient doesn't need to worry about the underlying infrastructure as long as they have a container runtime (like Docker) installed.

Virtual environments are a great choice for managing Python-specific dependencies and isolating projects within the Python ecosystem, and they are suitable for many development scenarios. However, containers like Docker offer a broader level of isolation and encapsulation, making

Containers offer a broader level of isolation.

them ideal for packaging and sharing entire applications, including all their dependencies and the runtime environment.

If you want to share a project with a third party and ensure that it runs consistently across different environments, containers are generally a better choice. Virtual environments can be shared, but might require some additional setup on the recipient's end if they don't have the same Python interpreter version or dependencies installed. On the other hand, since virtual environments are lightweight, in many cases all you need is to create a requirements file (or similar), which is nothing more than a plain text file that lists the components to be installed by the virtual environment manager of your choice.

Virtual environment managers of choice include pip, conda, venv or pyenv. Since we are using Anaconda in this book, I will mention the use of conda here, but remember that you can choose the virtual environment manager that you prefer. We can create a virtual environment called myvenv for example as follows:

```
> conda create -n myvenv python=x.x package=y.y  
package=z.z
```

In the example above, `-n` is a shortcut for name and is followed by the name we want to give to our virtual environment, `x.x` will indicate the version of Python you want to install, and you can specify what packages you want to install in the virtual environment and also their version, in this case we have version `y.y` and `z.z`.

A virtual environment can be shared with the help of a requirements file which lists the packages that are 'required' to be installed.

Virtual environment managers include pip, conda, venv or pyenv.

We can specify the version required to be installed in the virtual environment.

After the above command finishes executing, you are ready to use your virtual environment. All you have to do is to activate it as follows:

```
> conda activate myenv
```

To use a virtual environment, you need to activate it.

Upon activation, the PATH in your computer is modified pointing specifically to the isolated Python environment. Once you are done with using the virtual environment, you can deactivate it as follows:

```
> conda deactivate
```

And when you are done, remember to deactivate it.

You can check the virtual environments you have available to you as follows:

```
> conda env list
```

As a Jackalope Data Scientist, you are likely to create and use different projects, and virtual environments can help organise them. Imagine you have three distinct data science projects: project01, project02, and project03. The first step is to create a dedicated folder; let us call it Projects, to house these projects. Within this folder, create sub-folders for each project. For each project, it is a good practice to create a separate virtual environment. We will name them ven01, ven02, and ven03. This ensures that the dependencies for each project are isolated, preventing conflicts and maintaining project-specific configurations.

Virtual environments let us organise our projects.

You may be wondering where to save your script, say script01.py. Well, save it in the project sub-folder. In this

case, for example it will be part of the first project and hence it goes in the `project01` sub-folder. This helps in keeping all project-related files organised within their designated spaces. By doing this, you maintain a clear separation between projects, making it easy to locate and manage project-specific scripts. Avoid saving the scripts inside the virtual environment. The virtual environment is meant for managing dependencies, not for storing your project files. Placing scripts directly in the virtual environment can lead to confusion and potential issues.

```
Projects/
|-- project01/
|   |-- ven01/
|   |-- script01.py
|-- project02/
|   |-- ven02/
|   |-- ...
|-- project03/
|   |-- ven03/
|   |-- ...
```

Adopting a clean and structured approach to organising your data science projects pays dividends in the long run. Save your Python scripts within their respective project sub-folders, keeping them separate from the virtual environment. This not only enhances project organisation but also ensures a hassle-free development experience as you work on multiple projects simultaneously. Remember, the key is to keep things modular, maintain clear project

Avoid saving the scripts inside the virtual environment.

Our projects can be organised in folders as shown in the schematic here.

Remember to keep your scripts separate from the virtual environment.

boundaries, and leverage virtual environments for dependency management.

2.8 Summary

IN THIS CHAPTER WE HAVE covered some of the most important aspects of programming with Python. We started by looking at some of the advantages of using Python in the data science and analytics workflow as well as covering some of the general *Pythonista* programming style used by Python programmers.

We have seen how Python can be used as a scripting language as well interactively in a shell such as iPython. We can also use an enriched ecosystem with the help of the iPython/Jupyter Notebook. We also covered the different types that are supported by the language: Numbers, strings, complex numbers, lists, tuples, and dictionaries. Similarly, we saw how Python deals with mutable and immutable objects.

We can direct the way in which a programme will execute instructions with the help of control flow structures. Furthermore, Python expands its functionality with a number of modules and packages that can be readily imported and used. In this chapter we covered some modules such as NumPy and SciPy. With the help of the powerful pandas library, we can carry out data analysis and manipulation in a very straightforward manner.

Matplotlib is a module that enables us to create plots and visualisations as part of the analysis we perform. These are by no means the only useful modules available to you as a Python programmer, but they are some of the ones we will use throughout the rest of the book. We will use other modules we have not covered here, and we will explicitly mention this in the appropriate sections.

Finally, we covered an important aspect of Python: Virtual environments. These enable us to create and manage separate environments for our Python projects with different versions of package dependencies.

In the next chapter we will cover important concepts from machine learning and pattern recognition that will provide us with the context in which data science and analytics operate. Within that, we will present another useful Python library: Scikit-learn.

3

The Machine That Goes “Ping”: Machine Learning and Pattern Recognition

WE STARTED OUR DISCUSSIONS ABOUT data science and analytics by stating that the use of data as evidence is nothing new. Furthermore, we saw in [Chapter 1](#) how data science is a portmanteau for a number of overlapping tasks, taking tools from empirical sciences, mathematics, business intelligence, pattern recognition, machine learning and artificial intelligence. In this chapter we will focus our attention on the latter three in order to provide context to ideas explored so far and frame the algorithms that we will discuss in the following chapters.

3.1 Recognising Patterns

AN INTEGRAL PART OF BEING human is our ability to identify structure and order in the stimuli we receive, aided by information we have acquired previously. From that

point of view we are firmly in the realm of *cognitive psychology*, where language usage, memory, creativity and thinking are the main interests¹. Needless to say, the innate ability to recognise patterns is not unique to humans, and it is general among animals. Do not worry, we are not about to enter into a discussion of perceptual processes or theories of object recognition. However, it may be useful and illustrative to briefly consider the psychological theory of feature analysis.

In feature analysis, we are said to recognise an object by considering the constituent parts, or features, of the object. We then assemble them together to determine what the object is. For example, we know that a cat is a small furry animal with triangular ears, long whiskers and playful claws. When we see a cat, we recognise it for what it is because it satisfies these (admittedly simplified) rules.

This may seem a far-fetched example, were it not for the fact that actual research work on image recognition has been demonstrated using cat faces (and human bodies)². The field of pattern recognition is thus interested in the systematic detection of regularities in a dataset, based on the use of algorithms. We can then use these patterns to take actions such as classifying objects like cats. This does sound familiar, right?

The dataset in question may not necessarily be constrained to the features of a cat, or the parts of a body. As a matter of fact, exploiting patterns has been the bread and butter of sciences such as mathematics, physics or chemistry. As

¹ Eysenck, M. and M. Keane (2000). *Cognitive Psychology: A Student's Handbook*. Psychology Press.

If we know what a cat looks like, we can recognise other cats.

² Le, Q. V., R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean, and A. Y. Ng (2011). Building high-level features using large scale unsupervised learning. *CoRR. abs/1112.6209*.

a brief example, think of the efforts of Tycho Brahe and Johannes Kepler³: One methodically recording the positions of celestial bodies, and the other unraveling the mysteries behind these measurements and summarising them in what we now know as Kepler's Laws.

³ Gilder, J. and A. Gilder (2005). *Heavenly Intrigue: Johannes Kepler, Tycho Brahe, and the Murder Behind One of History's Greatest Scientific Discoveries*. Knopf Doubleday Publishing Group.

Recognising patterns is indeed a useful thing to do, and as you may imagine, it is helpful in more than one domain of applications. As a field, pattern recognition started up as part of science and engineering, resulting in a long list of applications to very practical problems.

Nonetheless, engineering was not alone in developing techniques: Other areas such as computer science have also developed capabilities to exploit regularities seen in data. It is therefore an interesting exercise to stop looking for a moment at individual spheres of knowledge and gaze at where they may end up converging. We have talked here about psychology, physics, mathematics, engineering and computer science.

Pattern recognition is useful in many areas such as science, engineering, computer science, etc.

The advances of these domains of knowledge, together with the relentless curiosity of the human mind, have posed questions with the aim to understand a variety of areas. In particular, self-reflection has made us turn the attention to ourselves, and has made the brain a hot area of study. It is a topic that still has many mysteries to reveal.

Self-reflection has made us turn attention to ourselves.

An important aim in the efforts to understand ourselves is the explanation of how the brain works and how it is able to be the centre of complex activity. This activity manifests

"How does the brain work'?" is a very important and difficult question.

itself as creativity, cognition, learning or intelligence, for example.

Is it possible to understand this marvellous organ? And if so, can we replicate its functions? Enter the realm of *Artificial Intelligence*!

3.2 *Artificial Intelligence and Machine Learning*

ARTIFICIAL INTELLIGENCE IS A FIELD that carries a number of connotations: From helpful companion androids through to sentient killer robots, and even the singularity. The key is in the allure of that second word in the compound noun: *Intelligence*. What is intelligence and how can we quantify it? The contempt is due to the first word: *Artificial*. Is it possible to recreate intelligence with the aid of a machine in such a way that the behaviour is similar to that of a person demonstrating intelligence?

This is not a new aim: The idea of recreating a human-like being has been the inspiration of stories such as that of the Golem, Pinocchio or Frankenstein's monster. It is therefore hardly surprising that when the genius of Alan Turing⁴ formulated the concept of a Universal Machine in 1936, the tantalising possibility of an intelligent machine was raised.

In the so-called Turing test, Alan Turing considered playing an "imitation game"⁵ where a player would have to decide which of two interlocutors is a human being and which is a machine, based only on their written responses to the player's questions.

The singularity is a hypothetical event where machines are capable of recursive self-improvement leading to an intelligence explosion.

⁴ Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*. 42(2), 230–265.

⁵ Turing, A. M. (1950). Computing machinery and intelligence. *Mind*. 59, 433–460.

If a machine could not be distinguished from the person, then the machine could be said to be “thinking”. Indeed, if there is no way of telling what other human beings are thinking except by a process of comparison with one’s own thinking, then there is no reason to regard machines any differently.

If we cannot tell what other humans are thinking, there is no reason to regard machines any different in that respect.

As a field, the aim of artificial intelligence is to make machines carry out tasks that are associated with the intellectual processing competence of humans. It is a Herculean labour, and one that not only involves advances in computer science, but also in neuroscience, psychology and even philosophy.

Artificial intelligence aims to make machines carry out tasks associated with intellect.

Among other tasks, pattern recognition, as discussed in the previous section, is an integral part of the various functions that an artificial intelligent agent would have to accomplish. The ability to recognise regularities would enable her to continuously adapt to a variety of changing environmental conditions. This adaptation allows a person to act and react to their surroundings and change their behaviour through learning. The same would be thus expected of the artificial intelligent agent.

Or him...

In order to distinguish the different levels of intelligence that can be described, sometimes it is useful to consider the concept of **Artificial General Intelligence (AGI)**⁶ which refers to the intelligence of machines that allows them to comprehend, learn, and perform intellectual tasks much like humans. AGI aims to emulate the human mind and behaviour to solve complex problems.

⁶ Gobble, M. M. (2019). The Road to Artificial General Intelligence. *Research-Technology Management*. 62(3), 55–59.

AGI is often described as “strong AI” and is characterised by machines that possess self-awareness, consciousness, and the ability to adapt to their surroundings. These machines can perform a wide range of tasks and exhibit creative actions that were previously limited to humans. The development of AGI is a real endeavour, and there is ongoing research and debate about its feasibility and potential implications. While progress has been made in the field of AI, AGI remains an aspirational goal rather than a current reality.

AGI systems possess self-awareness, consciousness, and the ability to adapt to their surroundings.

It is important to distinguish AGI from “narrow AI”. Narrow AI, also known as **Artificial Narrow Intelligence** (ANI), is designed to solve specific problems and/or perform specific tasks. Some examples of ANI include chatbots, expert systems, and even self-driving cars. In contrast, AGI aims to create machines that possess intelligence equal to humans and can perform any task that a human being is capable of. AGI systems would have the ability to understand the world, learn, plan for the future, and solve unfamiliar tasks.

ANI systems are designed to solve specific problems or perform specific tasks.

3.2.1 *Challenges and Future Trends*

DEVELOPING AGI POSES SEVERAL CHALLENGES. One of the key challenges is achieving a level of intelligence that is on par with human capabilities, including the ability to comprehend, reason, and communicate in natural language. There are ongoing discussions about the ethical and societal implications of AGI. As AGI systems become more

Part of the challenges of developing AGI include ethical and societal implications.

advanced, questions arise about their impact on employment, privacy, and the potential for misuse.

In terms of future trends, advancements in AGI are expected to focus on areas such as natural language processing⁷, machine learning, cognitive architectures, and knowledge representation. These advancements will contribute to the development of more intelligent and capable AGI systems.

⁷ Rogel-Salazar, J. (2020). *Advanced Data Science and Analytics with Python*. Chapman & Hall/CRC data mining & knowledge discovery series. CRC Press, Taylor & Francis Group.

One area that has been capturing everyone's imagination is Generative Artificial Intelligence (Gen AI). It refers to AI systems that have the ability to generate new content, such as images, text, or music, based on patterns and examples from existing data. These systems use techniques like deep learning and neural networks⁸ to generate creative outputs. Examples of Gen AI include Large Language Models (LLMs) like GPT-4 and image generation models like DALL-E.

Gen AI refers to systems that are able to generate new content.

⁸ Rogel-Salazar, J. (2020). *Advanced Data Science and Analytics with Python*. Chapman & Hall/CRC data mining & knowledge discovery series. CRC Press, Taylor & Francis Group.

While Gen AI focuses on specific tasks like content generation, AGI aims to achieve a broader level of intelligence that encompasses a wide range of tasks and problem-solving abilities. AGI systems would have the capacity to reason, understand natural language, learn from experience, and adapt to new situations. It is worth noting that Gen AI systems such as LLMs have been hailed as impressive advancements in AI, but they are not considered AGI⁹. LLMs are powerful models that can generate text based on prompts, but to-date they lack the comprehensive cognitive abilities and adaptability of AGI.

⁹ Goertzel, B. (2023). Generative AI vs. AGI: The Cognitive Strengths and Weaknesses of Modern LLMs. *arXiv preprint arXiv:2309.10371*.

The advances we are seeing in the field of Gen AI, such as the development of more sophisticated language models and image generation techniques, contribute to the overall progress in AI research. These advancements provide valuable insights and tools that can potentially inform the development of AGI in the future.

However, it is important to recognise that AGI remains a theoretical pursuit, and there are ongoing debates and challenges surrounding its feasibility and implications. While Gen AI showcases the potential of AI systems to generate creative outputs, achieving AGI requires significant advancements in various aspects of AI, including reasoning, perception, and adaptability.

Achieving AGI requires big advancements in various aspects of AI, including reasoning, perception, and adaptability.

3.2.2 *Unleashing the Power: Generative Artificial Intelligence*

TECHNOLOGY IS A RAPIDLY EVOLVING arena, and one term is causing ripples of excitement: Generative Artificial Intelligence (Gen AI). Its usage has become much more widespread than other machine learning tools, as the entry level does not require much technical expertise. However, there are some aspects that any user, experienced or not, should take into account when using these tools.

It is said that tools based on Gen AI are poised to transform the way we conduct business and lead our lives. In this section, we will delve into the profound impact of Gen AI, exploring its capabilities and outlining a some practical steps to adopt these tools.

Gen AI is poised to transform our lives.

Gen AI is not confined to automating routine tasks; it is a catalyst for innovation across diverse sectors, and it could reshape entire industries. While the promises of Gen AI are captivating, navigating its complexities requires a strategic approach. It is tempting to wait for industry giants or emerging startups to pave the way, but there are areas where Jackalope Data Scientists must take the lead. Large Language Models (LLMs) internal to an organisation, specific use cases, and the integration of Data and ML Ops demand proactive engagement.

Navigating the complexities of Gen AI requires a strategic approach.

We discussed the science part of the data science portmanteau, and as good Jackalope Data Scientists, we know that building and experimenting are key to harnessing the potential of our endeavours. It is the same with Gen AI. I would like to suggest here some pragmatic steps that can help us navigate the use of Gen AI tools ranging from policy development, technological adoption, strategic planning, and data optimisation.

A scientific approach to the use of Gen AI, where experimentation is employed, can lead to better results.

1. **Policy:** Develop robust policies to address the risks and challenges associated with Gen AI, including concerns about 'hallucinations', data privacy, and copyright claims. Ensure a comprehensive strategy that aligns with ethical considerations.
2. **Technology:** Embrace the current state of Gen AI technologies while recognising their level of maturity. Although the orchestration layer is evolving, immediate value can be extracted from existing tools. Begin building applications and put together your jackalope teams to capitalise on upcoming advancements.

A good Jackalope Data Scientist would be wise to look at these aspects when using Gen AI tools.

3. **Strategy:** Define a clear vision and roadmap for your Gen AI journey. Identify short-term wins and long-term objectives, conducting a thorough assessment of your company's operations to pinpoint areas where Gen AI can provide maximum value.
4. **Data:** Prepare your data for the Gen AI revolution. Ensure access to clean, relevant, and diverse data, addressing the widespread issue of poor data quality. Establish a foundation that allows your organization to leverage Gen AI effectively.

Developing a robust Gen AI policy may not be the most thrilling aspect of a rollout, but it stands as the linchpin for unlocking value swiftly. An effective Gen AI policy is paramount for unlocking value, aligning teams, and mitigating legal, security, and other risks associated with this groundbreaking technology. While Gen AI holds tremendous potential, it also introduces new risks, making policy development an essential precursor to productive engagement.

An effective Gen AI policy is paramount for unlocking value.

Building an effective Gen AI policy necessitates a comprehensive understanding of the associated risks such as:

- **Hallucinations**

Gen AI, while remarkable in its capabilities, is not immune to the occasional quirk. One such phenomenon is termed “hallucinations” where a Large Language Model (LLM) ventures to guess an answer when faced

A hallucination happens when the model produces responses without a solid foundation, akin to a spontaneous guess.

with uncertainty. In essence, these are instances where the model produces responses without a solid foundation, akin to a spontaneous guess. This unpredictable behavior poses a unique challenge in the realm of AI.

However, with great potential comes a measure of risk. The potential for misleading or even malicious content looms large with hallucinations. The spontaneity of these guesses could lead to outputs that may be inaccurate, misleading, or, in extreme cases, harmful. Such occurrences not only carry the risk of reputational damage but also invite legal implications, making it imperative to address and mitigate this facet of Gen AI.

The risk of hallucinations is inaccurate, misleading, or, in extreme cases, harmful responses.

Mitigating the risk associated with hallucinations is an ongoing process, as the landscape of AI continually evolves. Present solutions involve incorporating human oversight to monitor and intervene when necessary. The human touch adds a layer of discernment, helping to identify and rectify instances where the AI model may veer into uncharted and potentially problematic territory.

Incorporating a human in the process can help mitigate the risks.

- **Bias**

Bias can emerge as an inadvertent byproduct of the learning process. As these models delve into vast datasets for training, they may inadvertently pick up biases present in the data, leading to biased outputs in subsequent interactions. The inherent risk associated with bias lies in its potential to influence decision-making processes and perpetuate stereotypes. When the generated content reflects or amplifies existing biases, it

The risk associated with bias lies in its potential to influence decision-making processes.

can have far-reaching consequences, from reinforcing societal stereotypes to impacting crucial decision-making.

Addressing and mitigating bias in Gen AI involves a multifaceted approach. Developing robust policies to explicitly recognise and counteract biased outputs is a fundamental step. Additionally, a proactive stance involves considering diverse datasets during the training phase, promoting a more inclusive and equitable representation within the model's knowledge base.

The use of diverse datasets during training can help mitigate the bias risk.

- **Quality Control**

Gen AI, by its nature, introduces variability in its outputs. The quality of these outputs is not always guaranteed to meet pre-defined thresholds, and this variability introduces a unique set of challenges.

Output variability may have an impact on the quality of responses.

The risk associated with variable outputs extends beyond the AI realm, potentially impacting customer experiences, product design, and broader business operations. Inconsistencies or deviations from desired quality standards may lead to dissatisfaction among users and hinder the effective integration of AI-generated content into various applications.

Mitigating this risk involves a strategic combination of oversight and policy. Incorporating human oversight allows for real-time assessment and intervention, ensuring that outputs align with established quality standards. Outlining comprehensive quality control measures within organisational policies provides a

Human oversight enables real-time quality monitoring and intervention.

framework for managing and optimising the variability inherent in Gen AI.

- **Data Security**

The security of information is paramount. Generative AI, being a product of extensive data training, raises concerns about the potential compromise of sensitive information and the implications of mishandling data. The risk associated with data security involves the possibility of compromising sensitive information and the potential for breaches of privacy regulations. If not handled securely, the data used in training AI models may become vulnerable, posing a threat to privacy and regulatory compliance.

If not handled securely, the data used in training AI models may become vulnerable.

To mitigate these risks, organisations must establish clear and comprehensive policies for handling data securely. This includes not only protecting sensitive information from unauthorised access, but also ensuring compliance with privacy regulations and guidelines. A proactive approach to data security is foundational for the responsible deployment of AI, including Gen AI.

A proactive approach to data security is foundational for the responsible deployment of AI.

- **Transparency**

The opacity of Gen AI models presents a unique challenge in understanding the decision-making processes that lead to specific outputs. The lack of transparency in comprehending how these models arrive at their conclusions poses inherent risks.

The risk associated with transparency concerns the potential for misunderstanding or misinterpreting

outputs, especially in scenarios requiring accountability or regulatory compliance. Without a clear understanding of the model's internal workings, explaining decisions becomes a complex task.

The risk of transparency lies in potential misunderstandings, especially in accountable or regulated scenarios.

Mitigating this risk involves a thoughtful consideration of how to maximise transparency in AI applications. It is crucial to adopt innovative approaches to enhance transparency, drawing on years of research at the intersection of AI and human-computer interaction (HCI). A human-centred perspective to AI transparency¹⁰ is key to understanding that transparency is about facilitating appropriate human comprehension. It is possible to use approaches such as: model reporting, publishing evaluation results, providing explanations, and communicating uncertainty. By actively addressing and incorporating transparency considerations, organisations can enhance the interpretability of AI outputs.

¹⁰ Liao, Q. Vera and Wortman Vaughan, Jennifer (2024, Feb 29). AI Transparency in the Age of LLMs: A Human-Centered Research Roadmap. *Harvard Data Science Review*. <https://hdsr.mitpress.mit.edu/pub/aelql9qy>

• Intellectual Property Concerns

Gen AI introduces a layer of complexity regarding intellectual property (IP) concerns. The uncertainty surrounding the ownership of content generated by AI systems can lead to disputes over IP rights, posing potential challenges for organisations.

Generative AI adds complexity to intellectual property (IP) concerns.

The risk associated with IP concerns involves the potential for disputes over ownership rights, leading to legal complexities and uncertainties. Organisations may find themselves grappling with questions about the

copyright protection of AI-generated content and the delineation of rights among various stakeholders.

Mitigating IP concerns necessitates a proactive approach within organisational policies. Addressing potential disputes over ownership rights and establishing clear guidelines for IP-related matters are crucial steps. By incorporating well-defined IP protocols into the policy framework, organisations can navigate the intricate landscape of IP in the realm of Gen AI.

Policies can help address IP concerns.

There are other strategic considerations that must be taken into account:

- **Vendor Lock-In:** This poses a significant risk in the realm of implementing Gen AI, as over-reliance on a single vendor may potentially limit the organisation's capabilities. The inherent danger lies in the dependency on a specific provider for solutions, which could hinder flexibility and innovation. To mitigate this risk, you are advised, especially in the early stages, to work collaboratively with various partners. By diversifying collaborations, you can explore different capabilities, reducing the vulnerability associated with being tied exclusively to one vendor.
- **Building v Buying:** The strategic consideration of whether to build Gen AI solutions in-house or rely on third-party vendors requires a nuanced approach. Jackalope Data Scientists need to find a balance that aligns with their specific use cases and goals. This

The danger lies in the dependency on a specific provider for solutions.

In-house building offers customisation and control, while external solutions expedite implementation.

involves acknowledging that both building and buying have their merits. While in-house development allows for customisation and control, leveraging external solutions can expedite implementation. To navigate this decision successfully, you must evaluate your specific needs and priorities, ultimately creating a hybrid strategy that maximises benefits.

- **Training and Fine-Tuning LLMs:** When delving into the complexities of training or fine-tuning Large Language Models (LLMs), Jackalope Data Scientists face the consideration of associated expenses and time investments. The decision-making process involves strategically evaluating whether training or fine-tuning aligns with organisational goals, taking into account the costs and benefits of each approach. By adopting a thoughtful and informed stance, organisations can optimise the use of LLMs in a manner that aligns with their resources and objectives.
- **Unknown Outcomes:** A notable challenge in working with Gen AI is the unpredictable nature of LLM responses. To address this challenge, Jackalope Data Scientists should invest in creating technical proofs of concept. These prototypes serve as essential tools for understanding real-world scenarios and gaining insights into how LLMs interact with specific inputs. By testing and experimenting with concrete examples, organisations can enhance their understanding of the technology's capabilities and limitations.

Jackalope Data Scientists need to balance the associated expense and time involved in training LLMs.

A proof of concept aids in understanding real-world scenarios.

The stochastic application of LLMs introduces a challenge, as it results in different responses given the same set of variables. Mitigating this challenge involves adopting the right approach for predictable and scalable results. Jackalope Data Scientists should focus on refining their methodologies to achieve more consistent outcomes, acknowledging the inherent variability while ensuring that the results align with their expectations and requirements.

Refine methodologies for consistent outcomes while acknowledging variability.

- **Ethical Considerations:** Undoubtedly, ethical considerations play a crucial role in the development and deployment of Gen AI. Prototypes are deemed essential not only for assessing technical feasibility, but also for uncovering and mitigating ethical concerns and biases. Jackalope Data Scientists must prioritise the creation of prototypes that facilitate a comprehensive examination of ethical implications, ensuring that the deployment of Gen AI aligns with ethical standards and societal expectations.
- **Data Considerations:** The importance of well-structured data in Gen AI applications cannot be overstated. To mitigate data-related challenges, Jackalope Data Scientists need to allocate sufficient time for data-related work and testing during the project plan. Ensuring that the data is well-organised and aligned with the requirements of Gen AI models is crucial for achieving accurate and reliable results. A thoughtful approach to data considerations enhances the overall effectiveness and success of Gen AI implementations.

Prototypes can help in uncovering and mitigating ethical concerns.

Well-organised data is crucial for achieving accurate and reliable results.

3.2.3 *The Art of Prompt Engineering*

ONE ASPECT THAT MERITS SOME extra consideration when using Gen AI is the set of instructions that we provide to the models. This has been dubbed as prompt engineering, and it involves crafting precise and effective prompts to guide the behaviour of a LLMs, shaping the output they generate. This is akin to mastering a difficult craft, like writing jokes, where the art lies in crafting well-thought-out cues to guide the model's output and achieve the intended outcomes. In essence, prompt engineering is the art of instructing the AI model through carefully constructed input, enabling users to achieve desired outcomes.

After all, not everyone is Ernest Scribbler... writer of jokes.

Prompt engineering serves as a bridge between human intent and machine understanding. Crafting prompts requires a nuanced understanding of the model's capabilities and tendencies. It is not merely about providing instructions; it is about fine-tuning the language and context to elicit the desired response. Let's explore this artistry with some practical examples as shown in [Table 3.1](#).

Prompt engineering serves as a bridge between human intent and machine understanding.

As we can see in the examples provided, the use of generic prompts can result in broad and less targeted outputs. For instance, a generic prompt like "Write a short story about a journey" leaves the narrative open-ended and may result in a story lacking depth or specific themes. To enhance the output, prompt engineering can be very helpful. An engineered prompt such as "Craft an imaginative narrative detailing a jackalope's transformative journey through a futuristic cityscape, exploring themes of technological

Generic prompts can result in broad and less targeted outputs.

Precision in the prompt ensures a more nuanced response.

Use Case	Prompt	Description
Content Generation	Generic Prompt: "Write an article about artificial intelligence."	By employing a generic prompt, the model is given a broad topic without specific focus. This may result in a generalised and less targeted content generation, covering a wide range of AI aspects.
	Engineered Prompt: "Compose an engaging piece on the ethical implications of artificial intelligence in healthcare, focusing on patient privacy and data security."	Through prompt engineering, the AI is directed towards a specific facet of artificial intelligence, ensuring the generated content delves into the ethical concerns within healthcare. This approach enhances relevance and depth in the content, aligning with the desired focus.
Creative Writing	Generic Prompt: "Write a short story about a journey."	The generic prompt leaves the narrative open-ended, potentially resulting in a story lacking depth or specific themes.
	Engineered Prompt: "Craft an imaginative narrative detailing a jackalope's transformative journey through a futuristic cityscape, exploring themes of technological evolution and resilience."	Through precision in the prompt, the model's capabilities are channeled towards a more nuanced and compelling storytelling experience. The engineered prompt sets a specific context, ensuring the narrative explores defined themes and creates a richer, more immersive story.

Table 3.1: The art of prompt engineering.

evolution and resilience” provides a specific context for the story. This precision in the prompt ensures that the model’s capabilities are channeled towards a more nuanced and compelling storytelling experience, exploring defined themes and creating a richer narrative.

Prompt engineering underscores the importance of context and specificity. It enables users to tap into the vast potential of Gen AI by tailoring queries to suit their objectives. The more precise the prompt, the more targeted and valuable the AI-generated output becomes. However, it is crucial to approach prompt engineering with a clear understanding of its limitations. While it empowers users to guide the AI, there is no guarantee of perfect results. The stochastic nature of LLMs means that different prompts may yield varied responses, emphasising the need for thorough testing and refinement.

Navigating the art of prompt engineering requires a thoughtful approach to maximise the effectiveness of interactions with Gen AI. Here are some best practices to consider when crafting prompts:

- **Role Description:** Start prompts with a clear description of the model’s role. This helps set the context and provides a framework for generating responses. This practice ensures that the model understands its intended role, leading to more targeted and relevant answers. For example, “You are a seasoned journalist covering the latest advancements in AI technology. Explain the key features of the technology in layman’s terms”.

The more precise the prompt, the more likely it is to get targeted output. However, there is no guarantee of perfect results.

Provide a role description to the model role. Much like Mr Anchovy being told to become a chartered accountant.

- **Common Tasks:** Leveraging a model for common tasks increases the likelihood of obtaining accurate and valuable information. Familiar tasks allow the model to draw from its extensive knowledge base, providing insightful responses. For example, “As a tech enthusiast, provide a concise overview of the benefits of AI for everyday users”.

Leverage a model for common tasks.
 - **Contextual Clarity:** Offering context is key to obtaining nuanced and relevant answers. Providing additional information about the scenario helps the model grasp the intricacies of the question and generate more tailored responses. For example, “Imagine you are a customer support representative for a tech company. Clarify the advantages of using AI to a user concerned about data privacy”.

Provide context. Nothing is worse than asking for a generic argument at the Argument Clinic.
 - **Precision Matters:** Being precise in your prompts minimises ambiguity and ensures the model focuses on the specific aspects you are interested in. Clearly defining the input and desired outcome enhances the effectiveness of prompt engineering. For example, “Act as a technical instructor and explain the core architecture of a deep neural network, highlighting its output layer”.

Being precise ensures focus. Much like the Derbyshire Light Infantry and their precision display of bad temper.
 - **Avoid Ambiguity:** Ensure that your prompts are clear and unambiguous. Ambiguous queries may lead to unpredictable or less relevant outputs.

Ensure your prompts are clear and unambiguous.
 - **Test and Iterate:** Continuously test and iterate your prompts. Adjusting and refining your approach based
- Test and iterate your prompts.

on the model's responses improves the overall quality of generated content.

- **Encourage Step-by-Step Responses:** When dealing with complex topics, break down your queries into step-by-step instructions. This approach aids in generating more coherent and structured responses.

Break down your queries into step-by-step instructions.

- **Persistence Pays Off:** Do not hesitate to experiment with different prompts. Trying multiple approaches allows you to explore various perspectives on a topic and encourages the model to generate a diverse range of responses. Persistence is the key to uncovering hidden insights.

Experiment with different prompts.

By incorporating these best practices, you can enhance your prompt engineering skills and unlock the full potential of Gen AI, fostering a more fruitful collaboration between human intent and machine-generated insights. Prompt engineering is a powerful tool in your Jackalope Data Scientist toolkit. By mastering the art of crafting precise and context-rich prompts, you can unlock the full potential of these language models, harnessing their capabilities for a wide array of applications.

Use these best practices to foster a more fruitful collaboration between human intent and machine-generated insights.

Gen AI is no longer just a buzzword; it is the driving force behind the next wave of technological disruption. Users that proactively engage with Gen AI will position themselves as industry leaders, creating a self-reinforcing cycle of innovation. As the Gen AI journey unfolds, strategic considerations and testing methodologies, will be instrumental in harnessing the power of innovation while

mitigating potential pitfalls. The path forward lies in a careful blend of innovation, strategic thinking, and a resilient policy framework.

3.3 *Narrowing it Down: Machine Learning*

LET US NARROW DOWN OUR scope and consider the benefits that Artificial Narrow Intelligence (ANI) can bring to the table. We have seen how pattern recognition is a useful tool for any Jackalope Data Scientist and looked at the ambitious goals of Artificial General Intelligence (AGI). Where does machine learning fit in this story?

Machine learning can be understood as a sub-field of artificial intelligence. In that respect, machine learning has a much humbler aim than artificial intelligence: Instead of aspiring to the ultimate sentient robot, machine learning is interested in studying the methods that can be used to improve the performance of an intelligent agent over time, based on stimuli from the environment.

Notice that although this definition uses evocative language, the stimuli may not necessarily be read in real-time or even directly by the intelligent agent, and for that matter the intelligent agent does not have to be necessarily artificial. Think for example of a business manager who is interested in understanding what marketing materials have the best response in a certain sector of her online customers. The use of data about the browsing habits of her customers could play the role of the stimuli, and this understanding will help improve performance in her business. It is not hard to see

Machine learning is a sub-field of artificial intelligence focussed on improving the performance of an intelligent agent.

Stimuli may not be read in real-time or directly by the agent, and it doesn't have to be artificial.

The use of data generated by businesses can be used to improve their performance.

why machine learning has become an essential part of data science and analytics.

Machine learning has become ubiquitous in modern life, for example, every time that you check your email and identify a spam email in your inbox, you are providing an extra example to an algorithm (quite possibly a naïve Bayes classifier - see [Section 6.4.2](#)) that will adapt and learn in order to catch similar spam in the future. Similarly, online retailers are able to target products to customers based on items that other similar customers have purchased before with the aid of *collaborative filtering*. We mentioned image recognition earlier on, and other examples include fraud detection, advertisement placement, web search, etc.

The use of machine learning has become ubiquitous in modern life.

See [Section 8.4.2](#) for more information on collaborative filtering.

3.4 *Data Is Good, but other Things Are also Needed*

MACHINE LEARNING MAY OFFER US a substantial advantage — and insight — into the problems and questions in our business. In that respect, all we seem to need is an enormous amount of data at our disposal and the rest should follow. Data is indeed an asset in this respect and given current trends, data availability may not be a problem. However, we need to assess whether the data available is indeed relevant to the questions we are seeking to answer.

Data must be treated as an asset.

As you can imagine, it is fairly easy to go down the route that more data is always better. Nonetheless, it may be the case that having access to better relevant data is preferable to having so-called big data. I maintain that any efforts we

Having relevant data is preferable to having so-called big-data, particularly at early stages.

can spend in improving our data are worth investigating and investing in. After all, the patterns we are trying to exploit can only be as good as the data we employ.

With that in mind, it is often the case that having a wide variety of data may be more important than having a lot of it. By the same token, being able to apply a variety of clever algorithms may prove to be much more fruitful than simply having rows and rows of raw data. What is more, if the algorithms employed are scalable, adding more data may be a straightforward task.

We mentioned above the need to have relevant data, and the challenge there is to identify when we do indeed have it and when we do not. If we happen to be well-versed in the business domain where the data is being generated, we may have a good chance to decide if it is relevant or not. However, in cases where we do not have such experience, we should be able to face this challenge by enlisting the help of people with experience in the subject matter area. Having that expertise at our disposal can be as valuable as the data itself!

It is therefore recommended to have discussions and reviews with subject matter experts from an early stage in the process. This is particularly true in cases where the data science team may not be fortunate enough to have such expertise. This would be a good opportunity to add knowledge to the rangale of Jackalope Data Scientists.

Additionally, if we are indeed interested in gaining insight from the data, it is also important to discuss the results

Using a variety of algorithms is preferable to having lots of data that is not being used.

Make sure you have access to relevant subject matter expertise too. It may prove as valuable as the data itself.

Never miss an opportunity to add knowledge to the rangale of data scientists.

Communication skills are as important as technical ones.

of the modelling stages with subject matter experts and decision-makers. These discussions need to be organised with the understanding that not all of the people involved may be able to follow intricate and lengthy explanations about the finer points of a particular machine learning algorithm.

Effective communication serves as a linchpin in ensuring the seamless progression of the data science process. The ability to articulate and discuss the key issues inherent in the process in an inclusive manner is paramount. Inclusivity ensures that all stakeholders, regardless of their technical expertise or background, can engage meaningfully in the discourse surrounding the data science workflow. By fostering an environment where diverse perspectives are acknowledged and valued, the data science team can harness collective intelligence to refine and optimise their methodologies.

Only through open and comprehensive communication can the true efficacy of the data science process be fully realised, leading to more informed decision-making, innovative problem-solving, and ultimately, the achievement of desired outcomes. Not only does it facilitate the exchange of ideas and insights but also cultivates a collaborative environment where diverse perspectives are valued and integrated. Therefore, the emphasis on effective and inclusive communication stands as a crucial cornerstone in unlocking the full potential of data science endeavours.

Communicating your process and results, from the early stages is an important component to becoming a successful Jackalope Data Scientist.

Effective communication unlocks data science's full potential, fostering informed decision-making, innovation, and desired outcomes.

3.5 *Learning, Predicting and Classifying*

THE IMPLEMENTATION OF MACHINE LEARNING algorithms involves the analysis of data that could be employed in the improvement (learning) of the agent (model) and subsequently using the results to make predictions about quantities of interest or making decisions in the face of uncertainty.

It is important to bear in mind that machine learning is interested in the regularities or patterns of the data in order to provide predictive and/or classifying power. This is not necessarily the same as causality. We would need a more thorough examination to claim causes and effects given the data we observe.

Machine learning is interested in regularities and patterns in data.

Machine learning tasks are traditionally divided into two camps: Predictive or supervised learning, and descriptive or unsupervised learning. Let us start with **supervised learning**: A good example of this type of task is that of a traditional teacher-pupil situation where the teacher presents the pupil with a number of known examples to learn from.

We talk about two types of tasks: Supervised and unsupervised.

Let us return to the classification of cat faces: A teacher who knows what a cat looks like will present the pupil with several training images of cats and other animals, and the pupil is expected to use the *features* or *attributes* of the images presented to learn what a cat looks like. The teacher will have provided a *label* to each of the images as being of cats or not. In the testing part, the teacher will

Supervised learning makes use of labelled data.

present images of various kinds of animals, and the pupil is expected to classify which ones show a friendly feline face.

In machine learning parlance we talk about supervised learning when we are interested in learning a mapping from the input to output with the help of labelled sets of input/output pairs. Supervised learning lets us make predictions based on the data that we see and thus apply **generalisations**.

Supervised learning lets us make predictions.

Each input has a number of features that can be represented in terms of an N -dimensional vector that will help in the task of learning the label of each of the training examples. Think of a supervised learning task as providing an annotated map to a mountaineer who is signing up to our Kilimanjaro expedition and asking her to identify similar landscape features to those marked on the map as she walks along.

Or, where there two mountaineers? Oh well...

The other type of machine learning task is **unsupervised learning**. In this case, following our example of the teacher-pupil situation, the teacher takes a Montessori-style approach and lets the pupil develop, on her own, a rule about what a cat (or any other animal of the pupil's preference) looks like, without providing any hints or labels to the learner.

In this case, from a machine learning point of view, there are no input/output pairs. Instead, we only have the unlabelled inputs and their associated N -dimensional feature vectors, without being told the kind of pattern that

In contrast, unsupervised learning does not make use of labelled data.

we must look for. In that respect, an unsupervised learning task is less well-defined than a supervised one.

That does not mean that it is less useful; on the contrary, we can use unsupervised learning to gain a better understanding of the data we have acquired, and it can provide us with a description or classification of the dataset as well as discovering interesting patterns in the data. In other words, unsupervised learning lets us represent our data better by extracting **structure** from it.

Unsupervised learning can help us understand the structure of our data, providing us with ways to describe or classify data points.

Under unsupervised learning, in the case of our Kilimanjaro mountaineers, we would ask them to go on their journey without an annotated map, and instead identify interesting areas in the landscape they are able to see from the summit. One important thing to note is that an unsupervised learning task may enable us to assign labels to those inputs and thus open the door to the use of predictive or supervised learning.

Unsupervised learning may provide us with labels to be used in a supervised learning task.

We have touched upon data being labelled or not, and that has given us some clues as to the sort of problems we can tackle with each of them. Let us now turn our attention to the features and labels (if they exist). In [Section 3.1](#) we gave an example of some of the features that would enable us to recognise a cat. Some of those features can be quantified; for example we mentioned that it must be a small furry animal: How small? We can associate a number to this measurement and then we would be talking about a **numerical** or **continuous** variable. Continuous variables are typically associated with measurement units and we can represent

Numerical features are associated with measurement units and represented by numbers.

them with real numbers. For instance, in a scientific experiment measuring the growth of plants over time, for instance iris flowers, the height of the plants would be considered a continuous variable. This variable is associated with measurement units (e.g., centimetres) and can be represented with real numbers, such as 10.5 or 15.2 cm.

We may also have attributes that cannot be represented by a number, and instead they provide a description of the type of attribute we are referring to. In the example of the cat, we mentioned triangular ears, instead of round or floppy ones. Other examples include colour (black, white, etc.), gender (male/female), etc. We refer to these attributes as **categorical** or **nominal** variables, and are typically related to a class or category.

Type of learning	Categorical	Continuous
Supervised	Regression	Classification
Unsupervised	Dimensionality reduction	Clustering

This categorisation among the type of features and labels in a dataset may seem superfluous. However, a further bit of scrutiny will let us see that this innocuous grouping enables us to identify the type of machine learning algorithms that may be more suitable to the problems we would like to tackle. Take a look at [Table 3.2](#), where we provide some quintessential machine learning algorithms that fall into each category. We will cover these and other algorithms in the rest of the book.

We will talk about the iris dataset in [Section 3.11](#).

Categorical features provide a description of the attributes of the data.

Table 3.2: Machine learning algorithms can be classified by the type of learning and outcome of the algorithm.

This distinction between feature types will make it easier for us to understand the sort of machine learning task we want to employ.

3.6 Machine Learning and Data Science

MANY PROBLEMS THAT WE WOULD like to tackle using machine learning tend to have a high complexity. We have to bear this in mind when trying to apply algorithms, as it is not very likely to find a perfect practical solution.

Nonetheless, if the machine can learn, so can we. Machine learning algorithms are suitable to the solution of problems encountered in the data science and analytics workflow where we are interested in deriving valuable insights from data.

Let us take for instance the case of a supervised learning task where our ultimate aim is to find a function $h(\mathbf{x})$, called the hypothesis. This is related to the formal procedure in statistics¹¹ called hypothesis testing and is used to probe our ideas about the world, and help us test specific predictions. The function $h(\mathbf{x})$ enables us to predict values for the problem of interest based on the given input data $\mathbf{x}$. In a practical case, the inputs in the feature vector $\mathbf{x}$ are varied and some may be more useful than others. We would have to decide what the important features to take into account are, and then include them in our model.

The optimisation of the predictor $h(\mathbf{x})$ is done using training data points so that for each one of them we have input values $\mathbf{x}_{train}$ that correspond to an output y which is known in advance. Learning, in this sense, is thus the effective use of data in the task of training a model in order to accomplish the job it was set to complete.

Remember that if the machine can learn, so can we.

¹¹ Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

We need to decide what are the important features to include in our models.

In supervised learning we know the output y in advance.

From this point of view, we can relate the tasks that involve that training to the data science workflow steps we listed in [Section 1.4](#): Once we have identified the task at hand, we need to acquire relevant data, extract pertinent features and build our model. In addition to those steps, we also have to consider three important parts that will enable us to decide what sort of machine learning algorithm to choose for our problem.

The steps to follow are all framed by the data science workflow discussed in [Section 1.4](#).

With each prediction that we make, we can find the difference between that prediction and the true output value. We do this in order to assess how well our predictor is performing. An important part of the process is to be able to obtain a model that can perform well in a general setting, rather than memorising the intricacies of the data provided.

Generalisation is an important outcome of the model.

For example, if we are interested in building an algorithm that is able to recognise cat faces, we would like it to perform well with new, previously unseen cats. If the algorithm is only able to recognise *Bowman*, the Iberian lynx, but not *Mittens*, the kitty, then it is not a great algorithm to be deployed. If, however, the algorithm is able to recognise that a cat is a cat even if it is a drawing, a photograph or an actual real-life cat, then the algorithm is great. We shall come back to this point when we discuss the evaluation of algorithms in [Section 3.9](#).

Model evaluation must be part of the entire process.

At this point, it is pertinent to make clear that there is no such thing as a perfect model, just good-enough ones. The improvement in learning comes from generalising regular patterns in the training data to be able to say something

There is no such thing as a perfect model.

about unobserved data points. We should therefore be careful not to obtain a model that “memorises” the data, also known as **overfitting**. We can avoid this by employing techniques such as regularisation and cross-validation as we shall see later on in this chapter.

We shall discuss more about avoiding this memorisation in [Sections 3.8](#) and [3.13](#).

3.7 Feature Selection

MACHINE LEARNING CAN BE A powerful tool to have under our Jackalope Data Scientist belt. Not only can it be used together with computer science, mathematics and statistics to help us filter and prepare our data, but also to extract value out of it. It is therefore important to be able to separate the valuable relationships and patterns from any random, confounding ones. In any real application, it is inevitable to have a mix of distracting noise together with the signals we want to exploit.

We have to discern between actual patterns and random noise.

Unprocessed data can thus be thought of as the raw material that can be filtered and prepared to obtain the insights desired. However, as it is the case with cooking, the quality of the ingredients is as important as the steps specified in the recipe. With that analogy in mind, we need to be able to think through the available independent variables or features (ingredients) that will be included in the model (recipe).

As in cooking, the quality of the ingredients is as important as the steps of the recipe.

In some cases using the unprocessed, raw data may be suitable. However, in many cases it is preferable to create new features that synthesise important signals spread out in

the raw data. This process is known as **feature selection** where not only should we consider the features readily available, but also the creation and extraction of new features and even the elimination of some variables too.

Feature selection considers both existing features, and the creation of new ones.

The careful selection of the features to be used in the modelling helps with the understanding of the model outcomes. It also has a large effect on the predictions obtained from the application of machine learning algorithms. A common way to create new features is via mathematical transformations that make the variables suitable for exploitation by a particular algorithm. For instance, many algorithms rely on features having a linear relationship, and finding a transformation that renders nonlinear features to be represented as being linear in a different feature space is definitely worth considering. We will see some examples of this in the next chapter and also in [Section 9.1](#).

Mathematical transformations are a typical way to create new features.

It is true that knowing, a priori, the appropriate transformations and aggregations we should make is a hard task in and of itself. In many cases, experience with similar datasets and comparable applications is invaluable. Nonetheless, if you are starting up, not all is lost. Fortunately another common way to extract features is to use machine learning itself.

Experience with data transformations is an invaluable asset to be used.

In this case, unsupervised learning may provide a way to find useful clusters (see [Section 5.1](#)) in the data that may point us out in the right direction. Similarly, dimensionality reduction (see [Section 8.1](#)) can help us to determine

Unsupervised learning can also be useful in the feature selection process.

combinations of features that explain the variance shown in our dataset. We shall have an opportunity to talk about these types of algorithms later on in the book.

3.8 *Bias, Variance and Regularisation: A Balancing Act*

AS WE HAVE MENTIONED in the previous section, machine learning algorithms enable us to exploit the regularities in the data. Our task is therefore to generalise those regularities and apply them to new data points that have not been observed. This is called **generalisation**, and we are interested in minimising the so-called *generalisation error*, i.e., a measure of how well our model performs against unseen data.

Generalisation refers to the performance of a model against unseen data.

If we were able to create an algorithm that is able to recall the exact noise in the training data, then we would be able to bring our training error down to zero. That sounds great, and we would be very happy until we receive a new batch of data to test our model. It is quite likely that the performance of the model is not as good as a zero generalisation error would have us believe. We have ended up with an **overfit** model: We would be able to describe the noise in our data instead of uncovering a relationship, given the variance in our data.

In principle we can bring our training error down to zero. Unfortunately, this translates into a larger generalisation error.

The key is to maintain a balance between the propensity of our model to learn the wrong thing, i.e., the **bias**, and the sensitivity to small fluctuations in the data, i.e., the **variance**.

The key is to maintain a balance between bias and variance.

In the ideal case scenario, we are interested in obtaining a model that encapsulates patterns in the training data, and that at the same time generalises well to data not yet observed. As you can imagine, the tension between both tasks means that we cannot do both equally well and a trade-off must be found in order to represent the training data well (high variance) without risking overfitting (high bias).

High-bias models typically produce simpler models that do not overfit, and in those cases the danger is that of underfitting. Low-bias models are typically more complex and that complexity enables us to represent the training data in a more accurate way. The danger here is that the flexibility provided by higher complexity may end up representing not only a relationship in the data but also the noise. Another way of portraying the bias-variance trade-off is in terms of complexity v simplicity.

Models with a high bias tend to generate simpler representations, which prevent overfitting but may lead to underfitting. For instance, a linear regression model (see [Chapter 4](#)) with few parameters could exhibit high bias. On the other hand, models with low bias are often more complex, allowing for a more accurate representation of training data. An example is a deep neural network¹² with numerous layers and parameters. The intricate architecture enables the model to capture intricate patterns and dependencies within the training data. For instance, in image recognition tasks, a convolutional neural network (CNN) with low bias and high complexity can learn

We aim for a model that captures training data patterns and generalises to unseen data.

Another way to look at this is in terms of complexity versus simplicity of a model.

¹² Rogel-Salazar, J. (2020). *Advanced Data Science and Analytics with Python*. Chapman & Hall/CRC data mining & knowledge discovery series. CRC Press, Taylor & Francis Group.

hierarchical features like edges, textures, and shapes at various levels.

The tension between bias and variance, simplicity and complexity, or underfitting and overfitting is an area in the data science and analytics process that can be closer to a craft than a fixed rule. The main challenge is that not only is each dataset different, but also there are data points that we have not yet seen at the moment of constructing the model. Instead, we are interested in building a strategy that enables us to tell something about data from the sample used in building the model.

Keeping that balance is more an art than a science.

In order to prevent overfitting, it is possible to introduce ways to penalise our models for complexity. This can be done by adding extra constraints such as smoothness, or requiring bounds in the norm of the vector space we are working on — more on this later. This process is known as **regularisation**, and the effects of the penalty introduced can be adjusted with the use of the so-called regularisation hyperparameter, λ .

Regularisation is the process of introducing to our model a penalty for complexity.

Regularisation can then be employed to fine-tune the complexity of the model in question. In a sense it is a way to introduce Occam's razor principle to our model. Some typical penalty methods that are introduced for regularisation are the $L1$ and $L2$ norms that we will discuss in the following section. In [Section 3.13](#) we will touch upon how the hyperparameter λ can be tuned with the use of **cross-validation**.

Occam's razor tells us that when we have two competing theories that make the same predictions, the simpler one is preferred.

3.9 *Some Useful Measures: Distance and Similarity*

ONCE WE HAVE BUILT A set of models based on the training data we have, it is important to distinguish a good performing model against a less good one. So, how do we ascertain that a model is good enough for our purposes? The answer is that we need to evaluate the models with the aid of a scoring or objective function.

Remember that we are working with the principle that models are good enough.

Various machine learning algorithms have appropriate ways to let us evaluate how much we can trust what had been learned and how predictive the model obtained is. The performance of a model will therefore depend on various factors such as the distribution of classes, the cost of misclassification, the size of the dataset, the sampling methods used to obtain the data, or even the range of values in the selected features. It is important to note that evaluation measures are usually specialised to the type of problem and algorithm used, and the score provided will be meaningful to the problem domain. For instance, in the case of classification problems, the classification accuracy may provide a more meaningful score than other measures.

Evaluation measures are usually specialised to the type of algorithm used.

In general, model evaluation can be posed as a constrained optimisation problem given an objective function. The aim can then be presented as the problem of finding a set of parameters that minimises that objective function. This is a very useful way to tackle the problem, as the evaluation measure can be included as part of the objective function itself. For example, consider the case where we are

Model evaluation can be posed as a constrained optimisation problem.

interested in finding the best line of fit given a number of data points: A perfect fit would be found in the case where the data points align flawlessly in a straight line. As you can imagine, that is very rarely the case.

We will discuss regression in [Chapter 4](#).

Instead of expecting the unexpected, we can evaluate how well a line fits the data when we take into account the difference between the location of a point and its corresponding prediction as obtained from the model. If we minimise that distance, then we can evaluate and compare various calculated predictions. This particular evaluation measure used in regression analysis is known as the sum of squared residuals (SSR), and we will discuss it in more detail in [Chapter 4](#).

In regression, the minimisation of the sum of squares error is a typical evaluation measure.

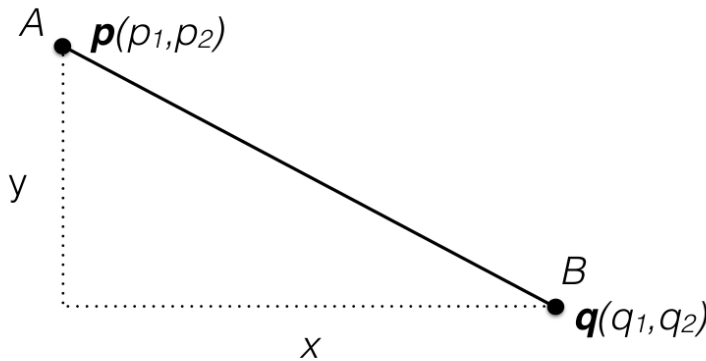


Figure 3.1: Measuring the distance between points A and B.

As we can see, the concept of distance arises naturally as a way to express the evaluation problem, and indeed a number of conventional evaluation procedures rely on measures of distance. Consider the points A and B in a two-dimensional space shown in [Figure 3.1](#). Point A has

Various evaluation measures rely on measures of distance.

coordinates $\mathbf{p}(p_1, p_2)$, and point B has coordinates $\mathbf{q}(q_1, q_2)$. We are interested in calculating the distance between these two points. This can be achieved in different ways, and we are familiar with some, such as the Euclidean and the Manhattan distances.

- **Euclidean distance:** This corresponds to the ordinary distance calculated using the straight line that joins points A and B ; in two dimensions, it corresponds to the distance given by the Pythagorean theorem. Given the coordinates of each of the two points in question, we can obtain the distance between A and B as:

Remember the Pythagorean theorem?

$$d_E = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} = \sqrt{x^2 + y^2}, \quad (3.1)$$

where the distances x and y are shown in [Figure 3.1](#). It is possible to extend this definition to n dimensions:

$$\begin{aligned} d_E &= \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}, \\ &= \sqrt{\sum_{i=1}^n x_i^2}, \end{aligned} \quad (3.2)$$

This is the well-known Euclidean distance.

where x_i is the distance along the i -th dimension. The Euclidean distance is also known as $L2$ -norm.

We also call it the $L2$ -norm.

- **Manhattan distance:** It is easy to see why this distance measure gets this name if we think of the distance that a New York yellow cab would cover while travelling along the streets in Manhattan. Apart from Broadway, the cab cannot move diagonally in the street-avenue grid. Instead, it can only move north-south and east-west. In the case of points A and B in [Figure 3.1](#), the Manhattan distance is

To measure the Manhattan distance, think of a yellow New York taxi cab and its journey through the island.

given by

$$d_M = |(q_1 - p_1) + (q_2 - p_2)| = |x + y|. \quad (3.3)$$

where $|\cdot|$ is the absolute value. For an n -dimensional space we can extend the above definition as:

$$\begin{aligned} d_M &= |(q_1 - p_1) + (q_2 - p_2) + \cdots + (q_n - p_n)|, \\ &= \left| \sum_{i=1}^n x_i \right|, \end{aligned} \quad (3.4)$$

The Manhattan distance is also known as the $L1$ -norm.

The Manhattan distance is also known as the $L1$ -norm.

From a geometrical point of view, the idea of measuring the distance between two points makes intuitive sense.

Furthermore, if the distance is zero, we can argue that the two points are effectively the same one, or at the very least *similar* to one another. This idea of similarity is therefore another useful tool in the development of evaluation measures, particularly in the case where features are not inherently amenable to being placed in a geometric space.

Another important concept, related to that of distance, is similarity.

Given two points A and B , the similarity measure d must satisfy a certain number of general conditions:

1. Must be positive: $d(A, B) \geq 0$
2. If the measure is zero, the points A and B are equal and vice versa: $d(A, B) = 0 \iff A = B$
3. Must be symmetrical: $d(A, B) = d(B, A)$
4. Must satisfy the triangle inequality: $d(A, B) + d(B, C) \geq d(A, C)$

The two distance measures we discussed above can be used to gauge similarity; however, there are a number of other useful ways to do this, for example the cosine and Jaccard similarities:

- **Cosine similarity:** This similarity measure is commonly used in text mining tasks, for example. In these cases, the words in the documents that comprise the corpora to be mined correspond to our data features. The features can be arranged into vectors, and our task is to determine if any two documents are similar or not. Cosine similarity is based on the calculation of the dot product of the feature vectors. It is effectively a measure of the angle θ between the vectors: If $\theta = 0$, then $\cos \theta$ is 1, and the two vectors are said to be similar. For any other value of θ the cosine similarity will be less than 1. The cosine similarity of vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ is given by:

$$s_c(\mathbf{v}_1, \mathbf{v}_2) = \cos(\theta) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{|\mathbf{v}_1||\mathbf{v}_2|}, \quad (3.5)$$

where $|\mathbf{v}_i|$ corresponds to the usual Euclidean norm to measure the magnitude of the vector $\mathbf{v}_i$.

- **Jaccard similarity:** The Jaccard similarity measure provides us with a way to compare unordered collections of objects, i.e., sets. We define the Jaccard similarity in terms of the elements that are common to the sets in question. Consider two sets A and B with cardinalities $|A|$ and $|B|$. The common elements of both sets are given by the intersection $A \cap B$. In order to give us an idea of the relative size of the intersection compared to the sets, we

Although distance is useful, similarity can be measured in other ways too.

Please note that the cosine similarity is a measure of orientation and not magnitude.

With Jaccard similarity we can compare unordered collections of objects.

divide the former by the union of the sets. This can be expressed as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}. \quad (3.6)$$

In the case of document similarity, for example, two identical documents will have a Jaccard similarity of 1 and those completely dissimilar a value of 0. Intermediate values correspond to various degrees of similarity.

The comparison of documents is a good candidate for the use of Jaccard similarity.

There are other distance and similarity measures that can be used. The choice will depend to a great extent on the type of problem to tackle as well as the algorithms and techniques to be used to solve the problem. In the following chapters we will address specific algorithms and evaluation measures that are appropriate to each of them.

There are other ways to measure distance and similarity, these are some of the most useful/common ones.

3.10 Beware the Curse of Dimensionality

WE HAVE BEEN REFERRING TO data features as an integral part of the ingredients we will use with our machine learning algorithms. Once we identified the features to be included in our model, we can consider them as the different dimensions along which our data instances can be placed: For a single feature we have a one-dimensional space, two features can be represented in two dimensions, etc.

The features selected to be included in our model can be considered as the different dimensions our data inhabit.

It follows that as we increase the number of features, the number of dimensions that our model must include is

increased too. Not only that, but we will also increase the amount of information required to describe the data instances, and therefore the model.

As the number of dimensions increases, we need to consider the fact that more data instances are required, particularly if we are to avoid overfitting. The realisation that the number of data points required to sample a space grows exponentially with the dimensionality of the space is usually called the **curse of dimensionality**¹³, a term used by Richard Bellman in the context of dynamic programming, and a great way to describe this issue!

The curse of dimensionality becomes more apparent in instances where we have datasets with a number of features much larger than the number of data points. We can see why this is the case when we consider the calculation of the distance between data points in spaces with increasing dimensionality.

Let us consider, without loss of generality, that we have a set of $M = 10$ data instances belonging to three different classes. We are interested in finding the closest neighbour to each of the data points and, in this case, assess if they belong to the same class or not. This is a very simple classification task. We can simplify the discussion by considering the use of a unit length measurement and counting the number of data points that fall within it. We are depicting this situation in [Figure 3.2](#) where we show the 10 data instances, represented with a triangle, an open circle and a plus sign, in one, two and three dimensions.

As the number of dimensions increases, we need to use more data points to avoid overfitting.

¹³ Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. Rand Corporation. Research studies. Princeton U.P.

The problem is more apparent with data points in higher dimensions

Let us count the number of data points that fall within one unit length measurement.

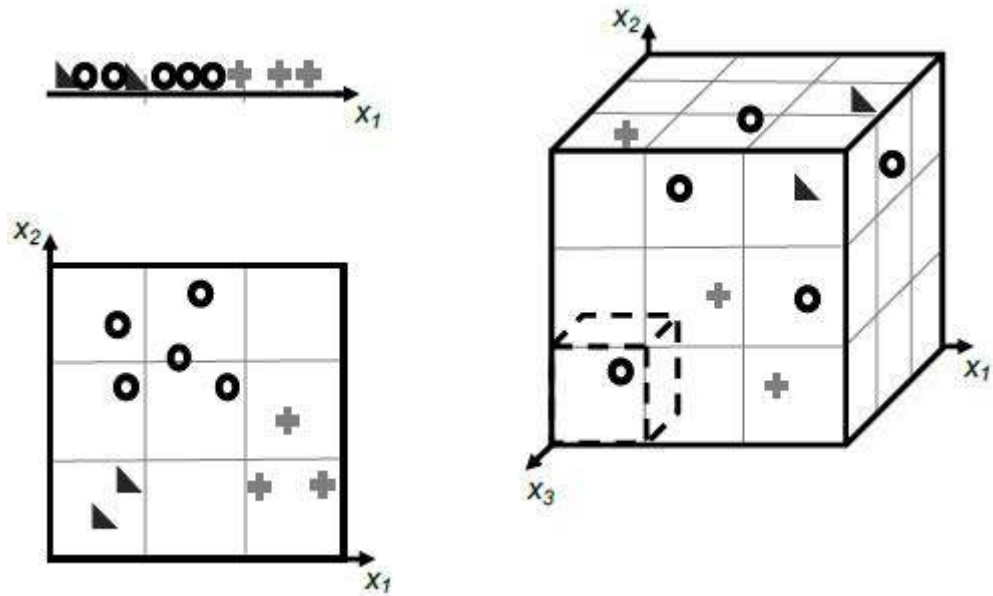


Figure 3.2: The curse of dimensionality. Ten data instances placed in spaces of increased dimensionality, from one to three. Sparsity increases with the number of dimensions.

In a one-dimensional space, the unit measurement is given by a line. In the example shown in Figure 3.2 we can see a space with three unit intervals, so the sample density is $\frac{10}{3}$. In other words, we have about 3.333... data points per interval and thus finding a near neighbour and assessing its class is certainly possible.

In the two-dimensional case, we would have to search an area of $3 \times 3 = 9$ unit squares and here the sample density is $\frac{10}{9}$, or an average of 1.111... data points per square. In this case it is less likely to find a neighbour of a given data instance within the unit square where the data point of interest is located. Finally, in the 3D case, we have a feature space of $3 \times 3 \times 3 = 27$ unit cubes and the sample density is $\frac{10}{27}$ or 0.111... on average. In this case our search for a

As the number of dimensions increases, the distance between data points becomes larger and larger. We end up with very sparse spaces.

neighbour within a cubic square becomes more difficult, as most of the feature space is effectively empty. We say that the space is *sparse*.

It is easy to see that as we keep adding features (dimensions), our space becomes sparser. It is due to this sparsity that we require a larger and larger number of data instances when dealing with higher-dimensions. For example, if we were interested in carrying out our classification task using one feature with a range of values between 0 and 1, and we wanted our dataset to cover 25% of this range, we would need 25% of the complete population. By adding another feature, we would require 50% of the population ($0.5^2 = 0.25$), and with three features we would need 63% of the population ($0.63^3 \simeq 0.25$).

It is due to this sparsity that we require a larger and larger number of data instances.

We may think that simply adding more data is the appropriate solution to dispel the curse of dimensionality. However, as we saw above, it is important to remember that the number of data instances needed grows exponentially with the number of dimensions. In practice we very rarely have access to an infinite amount of data. Furthermore, using too many features actually results in overfitting.

The number of data instances needed grows exponentially with the number of dimensions.

Following our classification example above with three categories (triangle, open circle and plus sign), it is much easier to find various ways to classify the data instances into separate classes when considering a higher number of features. This is a great thing to start with, but we must be careful to avoid overfitting, or even getting carried away with false patterns. Furthermore, the sparsity in a higher

Sparsity in higher dimensional space is not homogeneous.

dimensional space is not homogeneous, and it turns out that the space around the origin is much more sparse than in the corners of the hyperspace.

In order to understand this issue, let us consider a 2D space to start with. The mean of the feature space is at the centre of a unit square. If we wanted to search the space within one unit distance from the centre, we would be searching in the area given by a circle of unit radius (circumscribed by the square). Any data instances that fall outside the area of this circle turn out to be closer to the edges of the square and become more difficult to classify as their feature values are more distant to those in the centre (the mean). Let us now consider this situation in N dimensions:

- The volume of a unit hypercube of N dimensions is $1^N = 1$
- The volume of a unit hypersphere of N dimensions¹⁴ is:

$$V(N) = \frac{\pi^{N/2}}{\Gamma\left(\frac{N}{2} + 1\right)} r^N, \quad (3.7)$$

where the radius $r = 1$ for the unit hypersphere and $\Gamma(\cdot)$ is the gamma function.

In Figure 3.3 we show how the volume of the hypersphere tends to zero as the dimensionality N increases.

Nonetheless, the volume of the hypercube remains fixed.

This means that in higher-dimensional spaces, most of the data is actually in the corners of the hypercube that defines the feature space, making the classification task more difficult to achieve.

Data instances further away from the centre of a unit circle are more difficult to classify. This is aggravated in higher dimensions.

¹⁴ DLMF (2015). NIST Digital Library of Mathematical Functions. <http://dlmf.nist.gov/>, Release 1.0.10 of 2015-08-07

In higher-dimensional spaces, most of the data is in the corners of the hypercube.

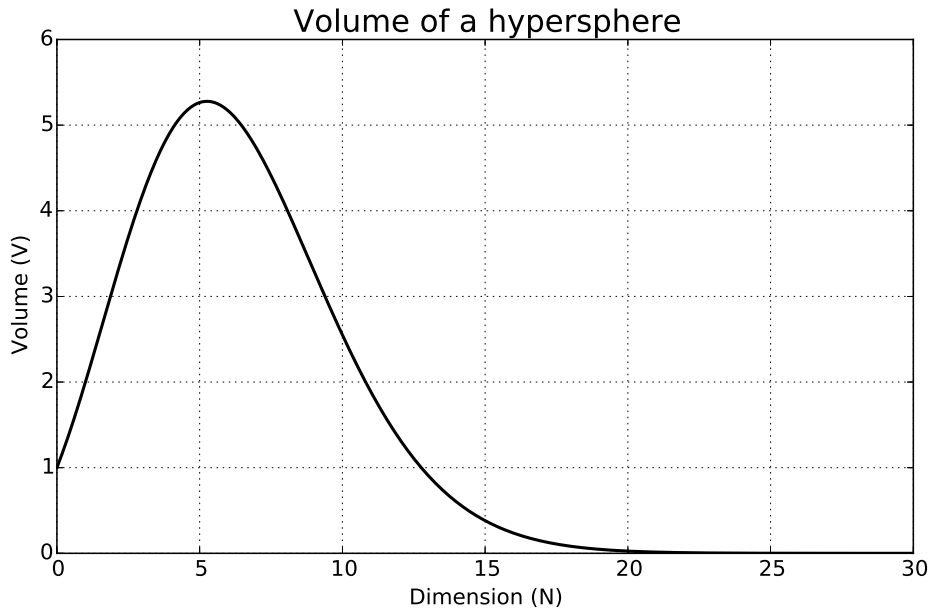


Figure 3.3: Volume of a hypersphere as a function of the dimensionality N . As the number of dimensions increases, the volume of the hypersphere tends to zero.

The curse of dimensionality is a very real thing, and there is not much that can be done to eliminate it completely. However, it is possible to minimise it, for instance by carefully checking that low-dimensional methods are effective in higher dimensions. Avoiding the curse of dimensionality can be done by increasing the amount of data, but even before going down that route it is worth considering if the features used are indeed a suitable collection.

Eliminating the curse of dimensionality completely is not an easy task.

In that respect, apart from a careful feature selection process, we can also reduce the dimensionality of the problem by transforming the data from a higher-dimensional space into a space with fewer dimensions as it is the case with

Dimensionality reduction aims to transform data from a higher-dimensional space into one with fewer dimensions.

Principal Component Analysis (PCA). We will discuss this technique in [Chapter 8](#). As for avoiding overfitting, in [Section 3.13](#) we will discuss the ideas behind **cross-validation**. But first we need to make a stop to talk about Scikit-learn.

See [Section 8.2](#) for a discussion on Principal Component Analysis (PCA).

3.11 *Scikit-Learn is our Friend*

THE BROAD AIM OF MACHINE learning, as we have seen, is the development and application of algorithms that can learn from data. This goal can be accomplished in a variety of ways and in recent times, with the advancement of computer power, together with the availability of useful software tools this is a goal that can be within the reach of many of us.

The advancement of computer power and the availability of useful software tools put machine learning within our reach.

Among the many programming languages and tools that are at hand, in this book we have chosen to use Python. Within the libraries and packages at our disposal, we concentrate mainly on Scikit-learn, as it contains a wide-range of machine learning algorithms. Scikit-learn builds upon libraries that we have already seen in [Chapter 2](#) such as NumPy, SciPy, pandas and Matplotlib. Scikit-learn is able to interact with pandas dataframes and other objects in Python. It is worth mentioning that the focus of Scikit-learn is the modelling part of the data science workflow, rather than the manipulation of data.

Scikit-learn contains a wide-range of machine learning algorithms and builds upon libraries that we have discussed.

Scikit-learn enables us to run popular models and techniques such as:

- Regression
- Clustering
- Feature selection
- Dimensionality reduction
- Classification
- Cross-validation

Some popular models and techniques available in Scikit-learn.

In the following chapters we shall have the opportunity to explore some of the implementations of these models. Scikit-learn also comes packaged with some test datasets that can be used for investigating the usage of the various models.

Scikit-learn comes packaged with some test datasets ready to play with.

Given that we will be using this library extensively, it is worth mentioning the typical data representation expected by the models in Scikit-learn. As explained in [Section 2.4](#), matrices and vectors are a favourable data representation, in particular for mathematical calculations and manipulations. Scikit-learn bears that in mind, and it expects data to be represented in the form of two-dimensional numeric matrices with M data instances (rows) and N distinct features (columns).

Scikit-learn expects data in a matrix representation.

The matrices can be NumPy or SciPy arrays for instance.

A canonical example in data science and analytics is the *Iris dataset*, and as you can imagine it is included with Scikit-learn. The set was first used by Ronald Fisher¹⁵, and it has a total of $M = 150$ samples of three species of iris flowers: Setosa (50), Virginica (50) and Versicolor (50). For each specimen we are provided with $N = 4$ feature measures (in centimetres): Sepal length, sepal width, petal length and petal width.

¹⁵ Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7(2), 179–188.

We can load this dataset by importing it directly from Scikit-learn as follows:

```
from sklearn.datasets import load_iris

iris = load_iris()
```

With the two lines of code above, we have imported the dataset and loaded it into an object named `iris`. We can now inspect the matrix that contains the feature data. For instance the size of the matrix must be $M = 150$ and $N = 4$, and we can verify this with the `shape` method as follows:

The Iris dataset is represented by a 150×4 matrix.

```
> iris.data.shape
(150, 4)
```

Let us see the first six data instances:

```
> iris.data[0:6,0:4]

array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4]])
```

We can use slicing and dicing to see the contents of the dataset.

The dataset also contains the class to which each data instance belongs, i.e., Setosa, Versicolor or Virginica. The information can be obtained by looking at the `target_names` of the `iris` object:


```
> print(iris.target_names)

['setosa' 'versicolor' 'virginica']
```

The class names of the iris flowers.

Remember that Scikit-learn expects data to be in numeric format, so using strings to represent the classes is not suitable. Instead, each of the three categories has been encoded with numbers corresponding to the position of the names in the list above:

```
> iris.target[0:151:50]

array([0, 1, 2])
```

We will come back to this dataset every so often in the rest of the book.

3.12 *Training and Testing*

THE MODELS WE USE FOR gaining insight into our business or research questions require data. With data as a resource, we need to be mindful of how, when and where it is used. Let us imagine that we are tasked with running a classification model based on the Iris dataset we saw in the previous section. We can consider using all 150 records for this purpose and base our model on the four features provided.

We need to be mindful of how, when and where data is used.

We carry out our modelling, and the result can be used to classify any new iris flower we encounter based on the four

measurements (features) used. However, how do we know how well (or how badly) our model performs? We would have to wait until we get new data not seen by the model. This may be an issue, as we do not necessarily have at hand a Mark Watney¹⁶ – Martian botanist extraordinaire – or any other (fictional or real) botanist to obtain new iris specimens, whether they are grown on Earth, Mars or elsewhere.

We are interested in the performance against unseen data.

¹⁶ Weir, A. (2014). *The Martian: A Novel*. Crown/Archetype.

What is more, we must remember that we build a model because we are interested in using it effectively. This means that we should care about its performance with new unseen data and therefore a way to assess this is with error rates. If we use our entire dataset to train the model, determining our training error becomes impossible. Not to mention the fact that the model will be built to account for the training data only and may thus overfit it, i.e., it will not generalise to new data instances.

We have discussed some causes of overfitting earlier in this chapter.

One way to tackle this problem is to prepare two independent datasets from the original one:

- **Training set:** This is the data that the model will see, and it is used to determine the parameters of the model
- **Testing set:** We can think of this as “new” data. The model has not encountered it yet, and it will enable us to measure the performance of the model built with the training set

The testing set is sometimes also called the holding set.

In some cases instead of partitioning the data into two sets, it is divided into three. The third component is called the **validation set**, and it is used for tuning the model. All three

Sometimes a third partition is used to validate the model.

parts need to be representative of the data that will be used with the model. It is important to clarify that the testing data must not be used for training the model, and that the validation set must not be applied for testing.

The testing set must never be used for training the model.

We can see a schematic representation of this situation in [Figure 3.4](#). Notice that the use of the training set in the modelling provides us with a measure of the training error. In turn, when applying the testing dataset we can get a measure of how well the model performs, i.e., we obtain a measure of the generalisation error. Finally, when using unseen new data, we are getting a measure of the so-called out-of-sample error.

A measure of generalisation error is obtained from the testing set.

From the discussion we had about the curse of dimensionality, we know that the more data instances we have for modelling, the better. On the other hand, the more test data we have, the more accurate will be our error estimate. A common way to split the data set into training and testing is 80/20% , and typically between one-third and one-tenth of the data is held out for testing. Other combinations are possible such as 70/30% split for example, but ensure you have enough data in your training set.

In cases when a straightforward splitting leaves us with datasets that may not be representative of the data, we can consider using a stratification method, for instance in situations where a particular class in the data is not represented in the training set. Stratification will aim at splitting the dataset so that each class is represented in both the testing and training sets.

There are a number of considerations that must be made when splitting a given dataset.

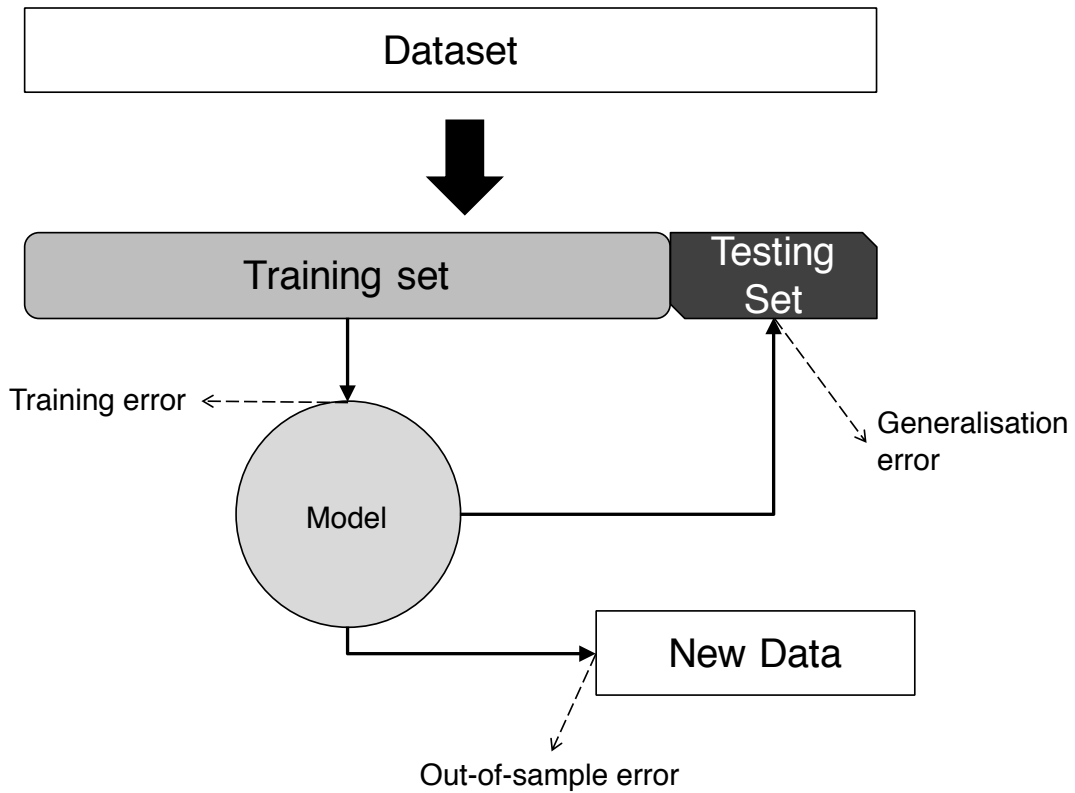


Figure 3.4: A dataset is split into training and testing sets. The training set is used in the modelling phase and the testing set is held for validating the model.

Splitting the data can be done by partitioning the set into two (or three) sections without altering the order of the data. However, in cases where the dataset has been ordered, this naïve procedure may lead to unbalanced training and testing sets. A better approach is to randomly select the data instances that will form each of the two partitions. It makes sense thus to carry out the randomisation as part of your workflow, regardless of the ordering of the data.

Splitting the dataset into training and testing is better done with random selection.

Scikit-learn is able to assist with the splitting of the dataset into random train and test subsets with the

For versions of Scikit-learn earlier than 0.18, this functionality was in the `cross_validation` module

`model_selection` module and its `train_test_split` function following this syntax:

```
model_selection.train_test_split( \
    *arrays, \
    test_size=None, \
    train_size=None, \
    random_state=None, \
    shuffle=True, \
    stratify=None)
```

Training and testing datasets can be obtained with the `train_test_split` function.

where `*arrays` are the datasets that will be split, `test_size` accepts a value between 0.0 and 1.0 representing the proportion of the dataset to be included in the testing set, `train_size` can be left out, and if so its value is automatically set to be the complement of the test size. The `random_state` parameter initialises the random number generator for sampling, `shuffle` is a boolean to determine whether shuffling is done or not before splitting and `stratify` can be used to split the data in a stratified manner using the array passed to this parameter as the class label.

The train size parameter is optional.

We can see how this can be applied to the case of the Iris dataset. We would like to store the features in arrays called `X_train` and `X_test` and the corresponding targets in `Y_train` and `Y_test`. We would like to hold 20% of the data for testing:

```
from sklearn import model_selection

X_train, X_test,\
```

Let us split the Iris dataset with Scikit-learn.

```
Y_train, Y_test =\
    model_selection.train_test_split(\
        iris.data, iris.target,\
        test_size=0.2, random_state=0)
```

We are using the Pythonic style of multiple assignment.

We can check the sizes of the newly created sets:

```
> print(X_train.shape, Y_train.shape)

((120, 4) (120,))

> print(X_test.shape, Y_test.shape)

((30, 4) (30,))
```

We can verify the size of the matrices we have after the split.

The modelling task will then be done with the training dataset `X_train` and `Y_train`. We can assess how well our model works by measuring the error against the testing dataset `X_test` and `Y_test`.

3.13 *Cross-Validation*

SINCE WE ARE INTERESTED IN making accurate and useful predictions, we need to ensure that any models we create generalise well to unseen data. We have discussed how a training and testing dataset split can help us with this goal. Nonetheless, the parameters that we obtain with the use of a single training dataset may end up reflecting the particular way in which the data split was performed.

In other words, we avoid overfitting.

The solution to this problem is straightforward: We can use statistical sampling to get more accurate measurements. This process is usually referred to as **cross-validation**. Cross-validation improves statistical efficiency by performing repeated splitting of data into training and validation sets, and re-performing model training and evaluation every time. The aim of cross-validation is to use a dataset to validate the model during the training phase.

Cross-validation improves statistical efficiency by performing repeated splitting of data.

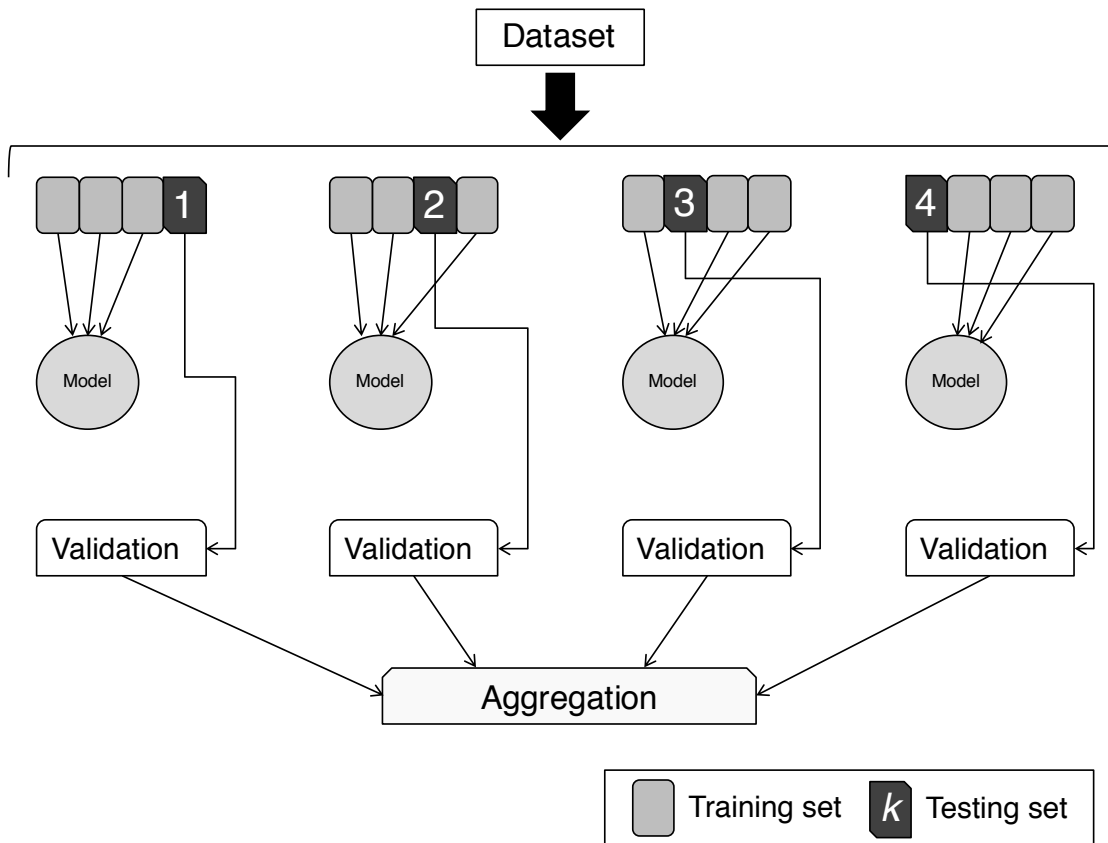
Let us see why this helps by considering the following scenario: We have carried out an initial training/testing split. The training set is used for modelling, and we perform the evaluation with the testing set. Imagine now that a different random state had been used to split the data. We would expect the model to see different data points during training. The generalisation error obtained with the second split would be different from the first. We can reduce variability by repeating this scenario over and over again, using different partitions and averaging the validation results over the rounds. Moreover, cross-validation is a great tool when we have a large, but limited amount of data points. Let us see how this can be done with k -fold cross-validation.

Cross-validation can help reduce variability by repeated use of training and testing splits.

3.13.1 k -fold Cross-Validation

A COMMON CROSS-VALIDATION TECHNIQUE IS the k -fold procedure: The original data is divided into k equal sets. From the k subsets, a single partition is kept for validating the model, and $k - 1$ subsets are used for training.

We need to split the dataset into k partitions.



The process is then repeated k times, using one-by-one each of the k subsets for validation. We will therefore have a total of k trained models. The results of each of the folds can be combined, for instance by averaging, to obtain a single estimation of the out-of-sample error. We can see a schematic representation of the k -fold cross-validation procedure for the case $k = 4$ in [Figure 3.5](#).

There are other procedures of cross-validation such as Leave-One-Out (LOO) where one single data sample is

Figure 3.5: For $k = 4$, we split the original dataset into four and use each of the partitions in turn as the testing set. The result of each fold is aggregated (averaged) in the final stage.

Other cross-validation procedures are also available.

taken for validation and the rest of the $M - 1$ data points are used for training. If we were to remove p samples from the complete set, we would be implementing the so-called Leave-P-Out (LPO) procedure.

Scikit-learn enables us to carry out cross-validation splits with the aid of functions such as `KFold`, `LeaveOneOut` and `LeavePOut`. The idea behind these functions is to generate k lists of indices that can be used to select the appropriate data points for each fold.

Cross-validation with the k -fold method is implemented with the `KFold` function.

Let us look at the `KFold` function which provides training and testing indices to split data into k consecutive folds (without shuffling by default):

```
model_selection.KFold(n_splits=5, \
    shuffle=False, \
    random_state=None)
```

`KFold` splits dataset into k consecutive folds.

where `n_splits` is the number of folds, `shuffle` is a boolean to shuffle the data before splitting and `random_state` initialises the random number generator for sampling.

Let us see an example. We can create 10 folds for the Iris dataset as follows:

```
kfindex = model_selection.KFold(n_splits=10, \
    shuffle=True, \
    random_state=0)
```

We can now use this cross-validator to split our data with the help of its `split` method as follows:

```

for train_ix, test_ix in kfold.split(iris.data):
    X_train, X_test = \
        iris.data[train_ix], iris.data[test_ix]
    Y_train, Y_test = \
        iris.target[train_ix], iris.target[test_ix]

```

KFold effectively maintains an index that keeps track of the data instances that go into each of the training and testing sets.

Each fold can then be used once as a validation, while the $k - 1$ remaining folds form the training set. We can look at the folds as follows:

```

> for i, (train_ix, test_ix) in \
    enumerate(kfold.split(iris.data)):
    print(f'Fold i:')
    print(f'  Train: index=train_ix')
    print(f'  Test:  index=test_ix')

```

Fold 0:

```

Train: index=[ 0  1  2  3 ...
Test:  index=[ 7 33 40 51 ...

```

...

Fold 9:

```

Train: index=[ 0  1  2  3 ...
Test:  index=[ 9 21 36 39 ...

```

Remember that these correspond to the indices of the data instances. Since we used shuffling, these may be different in your computer.

Cross-validation is a useful and straightforward way to get a more accurate estimate of the out-of-sample error, and at the same time a more efficient use of data than a single training/testing split. This is because each record in the dataset is used in both training and validating.

Cross-validation provides a more efficient use of our data.

Cross-validation can also be useful in feature and model selection procedures. For example, it can be used for tuning the regularisation parameter λ introduced in [Section 3.8](#): We split out training data and train a model for a fixed value of λ . We can then test it on the remaining subsets and repeat this procedure while varying λ . Finally, we select the best λ that minimises our measure of error.

Cross-validation can be very useful in tuning the hyperparameter in regularisation.

Despite these advantages, we must bear in mind that cross-validation increases the computational work that needs to be done, and if overused, can lead to overfitting. I urge you to use cross-validation given the advantages mentioned above. In a true case of “do as I say, not as I do”, there will be examples in the book where I will not perform the cross-validation step as part of the explanations of the various models we will tackle. However, it is worth remembering that cross-validation is an integral piece of the modelling part of the data science workflow.

Cross-validation increased the computational work that needs to be done.

3.14 *Summary*

I WOULD LIKE TO FINISH this chapter with a few thoughts that we must always keep in mind during our work in data science and analytics:

- If the machine can learn, so can we!
- Remember that machine learning and data science are not focussed on causality, but on prediction, insight and knowledge.

- Take a pragmatic approach to the use of Gen AI tools, ranging from policy development, technological adoption, strategic planning, and data optimisation.
- Prompt engineering serves as a bridge between human intent and machine understanding. Crafting prompts is an art that is all about fine-tuning the language and context to elicit the desired response.
- All models are wrong: There is no such thing as a perfect model, just good-enough ones.
- The data science and analytics workflow is a balancing act:
 - Bias v variance
 - Complexity v simplicity
 - Overfitting v regularisation
 - More data v cunning algorithm and resources
 - Accuracy v insight
 - Effort saved v computational cost
 - Jackalopes v unicorns
- Having a lot of data (even big data) is good, and being able to construct models is a great skill. Nonetheless, they are not magic wands.
- Beware the curse of dimensionality.
- Splitting our data into training and testing not only is good practice but a must.

- An important part of the modelling phase in data science is the use of cross-validation. Remember that the testing set must never be used for training.
- Embrace the use of Gen AI, it is after all a catalyst for innovation across diverse sectors, and it could reshape entire industries.

4

The Relationship Conundrum: Regression

REGRESSION ANALYSIS IS ONE OF the most widely used tools in statistical analysis. Most of us may have come across it at some point either by employing it or interpreting it. It is a powerful technique due to both its ease of calculation and simplicity of assumptions. However, it is due to these attributes that sometimes regression is misapplied or misinterpreted.

Regression is a well-known and widely used machine learning tool.

In this chapter we will cover the main aspects of regression analysis starting up with a motivation to the problem and covering both linear and polynomial regression techniques. Similarly, in [Section 3.8](#), we will see how feature selection can be done with the help of appropriate regularisation techniques.

We will cover here some of its most important aspects.

See [Section 3.8](#) about regularisation.

4.1 Relationships between Variables: Regression

CONSIDER A SITUATION WHERE YOU are interested to determine the association between two (or more) pieces of

information; say for example, the relation of the height of a child compared to that of her parents, or ice cream sales and temperature, or even the body mass of an animal and the mass of its brain. We can collect data for these events and use it for constructing a model that enables us to explore the relationship between the variables in question. Ultimately, our goal is to use our model to predict the outcome of the variable of interest given the values of the other variable(s).

All these are actual, well-documented examples of regression usage.

We usually call the quantity of interest the **response** or **dependent** variable and denote it with the variable y . The other quantities are called **predictors**, **explanatory** or **independent** variables and denote them as x . Intuitively, we know that two quantities are correlated if there is a relationship between the two variables, i.e., the value of one tells us something about the value of the other one.

The dependent variable is the quantity we want to predict.

The independent variables are also called regressors.

In a correlation analysis we estimate a value bounded between -1 and 1 , and we call it the correlation coefficient. This coefficient tells us the strength of the linear association between the two variables. If the two quantities vary in tandem (if one increases/decreases, the other one does too) the correlation coefficient is positive, whereas it is negative when the two quantities vary out of sync (if one decreases, the other one increases).

The correlation coefficient measures the degree of *linear* relationship among variables.

It is important to remember that the correlation coefficient measures the strength of linear relationship between the variables, and therefore a value of zero does not mean that there is no relationship at all. It simply indicates that there is no *linear* relation between the variables in question.

A zero correlation coefficient simply indicates no linear relationship, but other types are available!

Determining the strength of the relationship provides us with some clues towards answering our original question. Although we can tell whether the (linear) relationship is strong (± 1) or not (0), the correlation coefficient does not tell us how. A regression analysis does allow us to start seeing how.

Regression analysis lets us explore the relationships among variables.

Before we continue, a word of caution: Just because we measure a correlation between two variables, it does not mean that there is a causal relationship between them. In other words, the fact that people use umbrellas when it rains does not mean that umbrellas cause rain to fall. We are better off avoiding Sir Bedevere's type of reasoning: "If you weigh the same as a duck, then, you're made of wood and must be a witch".

You may have heard the age old aphorism: "*Correlation does not imply causation*".

Similarly, we must be careful when considering relationships between variables, as they may be related to a third, confounding, variable. Take for example the relationship between ice cream sales and temperature we mentioned earlier on: As summer approaches, the ice cream van is busy selling more ice cones. A similar trend has been noted for the murder rates: As the heat rises, the number of killings do too¹. In a simplistic analysis, one may risk looking at the relationship between ice cream sales and murder, and concluding that one causes the other, without taking into account the weather. Always be on the lookout for confounding variables.

¹ Lehren, A. W. and Baker, A. (2009, Jun 18th). In New York, Number of Killings Rises With Heat. *The New York Times*.

Nonetheless, trying to figure out the existence of these relationships and explaining them is by no means

something new. As a matter of fact even the name of the technique carries some historical connotations: Sir Francis Galton, a 19th century polymath and first cousin of Charles Darwin, effectively coined the term. Galton was interested in a variety of subjects, from psychology to astronomy as well as statistics. The acceptance of fingerprints as evidence in court was advanced thanks to Galton's studies², including estimating the probability that two people have the same fingerprints.

Back to our subject of interest, Galton pioneered the application of statistical methods to many of his scientific interests. For instance, he indeed was interested in the relative size/height of children and their parents³ (both in animals and plants). Among his observations he noticed that a tall parent is likely to have a child that is taller than average. However, the child is likely to be less tall than the parent. Similarly, a parent who is shorter than average would have children taller than the parent, but still below the average. In other words, the difference in height between parent and offspring is proportional to the parent's deviation from the typical population. He described this by saying that the height of the offspring *regresses* towards a *mediocre* point.

Regression towards the mean is a purely statistical phenomenon and can be seen as a fact of life, if you will. This phenomenon, often observed in scientific studies, social phenomena, and everyday experiences, entails the tendency for extreme values to move closer to the average upon subsequent measurements or observations. The key to the

² Cole, S. (2004). History of fingerprint pattern recognition. In N. Ratha and R. Bolle (Eds.), *Automatic Fingerprint Recognition Systems*, pp. 1–25. Springer New York.

³ Galton, F. (1886). Regression Towards Mediocrity in Hereditary Stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*. 15, 246–263.

The height difference in proportional to the parent's deviation from the typical population.

We would call this the mean in modern, politically-correct terms.

matter is the expectation value of the measured mean: A sprinter who breaks the world record in a race is expected to run to her average time in the next one, or the score in a mid-term exam can be expected to be less bad than the score in the final.

Regression to the mean is an inescapable fact of life.

All in all, regression is thus the mean value of a response variable as a function of one or more explanatory variables. A regression model is an approximate to it. As a first attempt to determining the dependence among the variables, the simplest thing we can do is check if the relationship follows a straight line.

In that sense, a linear regression model assumes (among other things) that the response can be described by a linear function. Even if it is not, we can at least approximate linearly over a range of values or carry out transformations to linearise relationships.

Linear regression assumes:

- Linear relationship.
- Multivariate normality.
- No or little multicollinearity.
- No auto-correlation.
- Homoskedasticity.

The veracity of a linear model may or may not reflect the actual relationship among the variables in question, and we should remind ourselves that there is no such a thing as a perfect model!

The linear regression model has therefore the following form:

$$\begin{aligned} \mathbf{y} &= f(\mathbf{x}) + \varepsilon, \\ &= \beta_0 + \beta_1 \mathbf{x} + \varepsilon, \end{aligned} \quad (4.1)$$

where β_0 is the intercept of the line, β_1 is the slope of the line, and ε denotes a vector of random deviations or

The intercept is the point where the line crosses the y -axis.

residuals assumed to be independent and identically normally distributed. We refer to β_0 and β_1 as the regression coefficients. In the next section we will extend the model to more than one independent variable and will see how to implement the model using matrix notation.

The β_i coefficients are called the regression coefficients.

4.2 Multivariate Linear Regression

IN THE PREVIOUS DISCUSSION we have only taken into account the relationship between the dependent variable and a single independent one. We can extend the model to include many more variables; for example let us consider N observations on the response y_i with $i = 1, 2, 3, \dots, N$; and with M regressors $\mathbf{x}_j$ with $j = 1, 2, 3, \dots, M$. The multivariate linear regression model is written as:

Remember that $\mathbf{x}_j$ are vectors that contain the various data points we will use in our model.

$$y_i = \beta_0 + \sum_{j=1}^M \beta_j \mathbf{x}_j + \varepsilon_i. \quad (4.2)$$

We would like to express the linear regression model in terms of matrices. As such, we can write the independent variable as an $N \times M$ matrix:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1M} \\ 1 & x_{21} & x_{22} & \cdots & x_{2M} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{NM} \end{pmatrix}, \quad (4.3)$$

The matrix $\mathbf{X}$ provides us with a compact representation of the collection of the different M features for each of our N data points.

whereas the independent variable is a column vector with N

elements:

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}. \quad (4.4)$$

Similarly, the vector $\mathbf{Y}$ allows us to collate all the various responses y_i .

Finally, the regression coefficients and the residuals are given by:

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_M \end{pmatrix}, \quad (4.5)$$

The intercept β_0 has been included in the regression coefficient vector $\boldsymbol{\beta}$.

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_M \end{pmatrix}. \quad (4.6)$$

Note that we have included the intercept β_0 in the expression of the regression coefficients. This is why we have a column of 1s in the matrix shown in expression (4.3) on the previous page.

In that manner, we end up with the following form for the regression model:

$$\mathbf{Y} = \boldsymbol{\beta}\mathbf{X} + \boldsymbol{\varepsilon}. \quad (4.7)$$

This is the expression for a multivariate linear regression model.

Our task is therefore to find the regression coefficients in the vector $\boldsymbol{\beta}$. The simplicity of Equation (4.7) is provided

by the use of matrices. Furthermore, it is also their use that will make all the calculations and manipulations to find the regression coefficients much easier as we shall see in the next section.

4.3 Ordinary Least Squares

WE ARE TASKED WITH FINDING the regression coefficients β in the multivariate regression model given by Equation (4.7). Let us recall that we are interested in predicting the value of the dependent variable given the values of the explanatory variables. If we were able to craft a perfect linear model, the actual value of y would match exactly the prediction $f(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M)$. This implies that the residuals ε are zero.

A perfect prediction would make the residuals equal to zero.

In a more realistic scenario, we would find a pretty good line of best fit to the data by minimising the error. One way to implement a suitable objective function for this purpose is to minimise the sum of squared residuals as follows:

$$SSR = \varepsilon^2, \quad (4.8)$$

$$\begin{aligned} &= |\mathbf{Y} - \mathbf{X}\beta|^2, \\ &= (\mathbf{Y} - \mathbf{X}\beta)^T (\mathbf{Y} - \mathbf{X}\beta), \end{aligned}$$

$$SSR = \mathbf{Y}^T \mathbf{Y} - \beta^T \mathbf{X}^T \mathbf{Y} - \mathbf{Y}^T \mathbf{X} \beta + \beta^T \mathbf{X}^T \mathbf{X} \beta. \quad (4.9)$$

In linear regression we are thus interested in minimising the sum of the square residuals.

Notice that the third term in the last expression is actually a scalar: $\mathbf{Y}^T \mathbf{X} \beta = (\beta^T \mathbf{X}^T \mathbf{Y})^T$.

Since we require the minimum of the SSR quantity in Equation (4.9), we take its derivative with respect to each of

the β_i parameters. This leads us to the following expression:

$$\begin{aligned}\frac{\partial(SSR)}{\partial\beta_i} &= \frac{\partial}{\partial\beta_i} \left(\mathbf{Y}^T \mathbf{Y} - \boldsymbol{\beta}^T \mathbf{X}^T \mathbf{Y} - \mathbf{Y}^T \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\beta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\beta} \right), \\ &= -\mathbf{X}^T \mathbf{Y} + \left(\mathbf{X}^T \mathbf{X} \right) \boldsymbol{\beta}.\end{aligned}\quad (4.10)$$

We need to take the first derivative to calculate the minimum.

We can now equate the above expression to zero, leading us to the solution of the matrix equation as:

$$\boldsymbol{\beta} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{Y}.\quad (4.11)$$

This is the solution to the linear model given in Equation (4.7).

We refer to Equation (4.11) as the normal equation associated with the regression model.

We have already encountered this calculation in [Section 2.4.1](#) where we used Python to demonstrate the use of linear algebra operations such as transposition, inversion and multiplication. Let us go through the calculations one step at a time.

We have actually already implemented this calculation in [Section 2.4.1](#).

4.3.1 The Maths Way

WE CAN USE THE NORMAL equation given in expression (4.11) to solve the linear system given by the linear model in Equation (4.7). Let us see how this is done for the same data used in [Section 2.4.1](#). For the independent variable we have:

We have a single feature and four records.

$$\mathbf{X} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix},\quad (4.12)$$

and for the dependent variable:

$$\mathbf{Y} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}. \quad (4.13)$$

We have a very succinct dataset with only four observations and one single feature. In other words, we have an $N = 4$ by $M = 1$ system.

In this case we have a 4×1 linear system.

Let us express Equation (4.7) as $\beta = \mathbf{M}_1 \mathbf{M}_2$. We can now start by calculating $\mathbf{M}_1$ as follows:

$$\mathbf{M}_1 = (\mathbf{X}^T \mathbf{X})^{-1}, \quad (4.14)$$

We calculate the first part of the solution as $\mathbf{M}_1 = (\mathbf{X}^T \mathbf{X})^{-1}$.

$$\begin{aligned} &= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \right)^{-1}, \\ &= \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}^{-1}, \\ &= \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}. \end{aligned} \quad (4.15)$$

Whereas the second part, $\mathbf{M}_2$, is given by:

$$\mathbf{M}_2 = \mathbf{X}^T \mathbf{Y}, \quad (4.16) \quad \text{The second part is given by } \mathbf{M}_2 = \mathbf{X}^T \mathbf{Y}.$$

$$\begin{aligned} &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \\ &= \begin{bmatrix} 10 \\ 30 \end{bmatrix}. \end{aligned} \quad (4.17)$$

Finally, the regression coefficients are given by:

$$\begin{aligned} \beta &= \mathbf{M}_1 \mathbf{M}_2, \\ &= \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix} \begin{bmatrix} 10 \\ 30 \end{bmatrix}, \\ \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \end{aligned} \quad (4.18)$$

Finally, the multiplication $\mathbf{M}_1 \mathbf{M}_2$ gives us the coefficients of the linear regression.

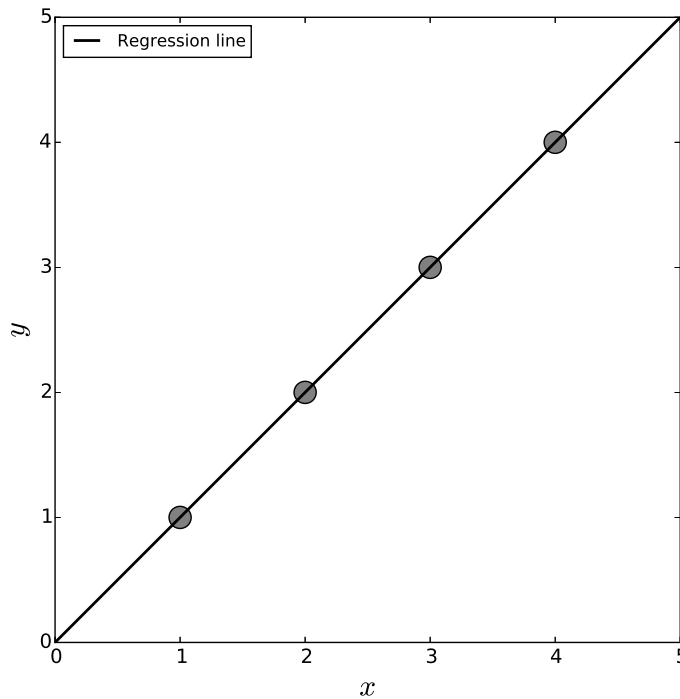
As we can see from the results above, the intercept of the model is 0 and the slope of the line is 1. In other words, the model can be expressed by the following equation:

$$\mathbf{Y} = \mathbf{x}, \quad (4.19) \quad \text{Remember that the model is } \mathbf{Y} = \beta_0 + \beta_1 \mathbf{x}.$$

and therefore the line of best fit is given by a line at 45° crossing the origin, as can be seen in [Figure 4.1](#). The grey circles correspond to the data points used for the regression, and the line is given by Equation (4.19).

Not only is it important to be able to carry out the operations that enable us to determine the regression coefficients, but also we must be able to interpret them. In case of the intercept β_0 , we can consider this to be the expected mean value of the predicted variable when the independent variable is not present. By the same token, a “unit” increase in the independent variable is associated with a β_1 “unit” increase in the predicted variable.

Not present in this case means that the dependent variable is zero, $x = 0$.



Please note that in the example used in this section, since all the points considered are perfectly aligned, the line of best fit does indeed pass through every point. However, in

Figure 4.1: The regression procedure for a very well-behaved dataset where all data points are perfectly aligned. The residuals in this case are all zero.

a more realistic situation, the presence of noise cannot be ignored. This is why it is important to get an estimate of the sum of squared residuals. A schematic representation of this situation is shown in [Figure 4.2](#), where the distance from each of the data points to the line of best fit is shown as a dot-dashed line.

Noise cannot be ignored and that is why we estimate the sum of squared residuals.

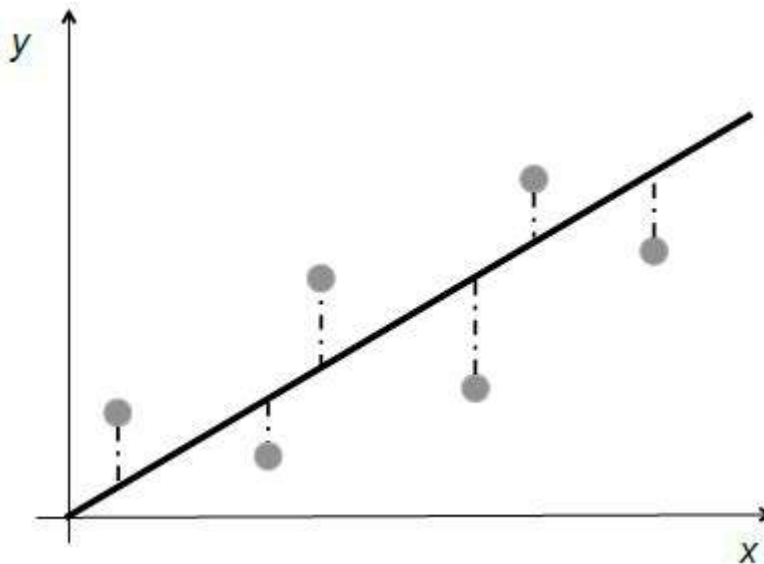


Figure 4.2: The regression procedure for a very well-behaved dataset where all data points are perfectly aligned. The residuals in this case are all zero.

Although the calculations we have covered above are straightforward, they can be somewhat laborious. This is particularly true in the case where there are more than one or two independent variables at play. Furthermore, the matrix inversion that must be carried out is not a trivial calculation, and we should be careful as there is no guarantee that a given matrix is invertible. In those cases, we say that the matrix is singular or degenerate, and methods to approximate the inverse are needed.

The calculations shown are straightforward, but can get somewhat laborious.

We can implement a function in Python with the normal equation given by expression (4.11). A naïve approach may be well suited for simple problems. In a nutshell, we do not really want to solve the problem “by hand” as done in this section, but perhaps we are better off using Python libraries that are readily available for this task such as those in statsmodels and Scikit-learn. We will do that in the following sections.

Instead of doing the computations “by hand”, we are better off using a computer.

4.4 *Brain and Body: Regression with One Variable*

LET US NOW TAKE A look at running a linear regression with a slightly larger dataset. In this case, we are going to follow one of the examples that we mentioned at the beginning of this chapter. The dataset that we will use looks at the relationship of the body mass of an animal and the mass of its brain⁴. The data is available at <http://dx.doi.org/10.6084/m9.figshare.1565651> as well as at <http://www.statsci.org/data/general/sleep.html>.

⁴ Allison, T. and D. V. Cicchetti (1976, Nov 12). Sleep in mammals: ecological and constitutional correlates. *Science*. 194, 732–734.

We will assume that the data has been downloaded into a comma-separated-value (CSV) file with the name `mammals.csv` and saved in a sub-folder called `Data`. We can use pandas to manipulate the file. Let us start by looking at a scatter plot of the data. Before we can do this we need to upload the necessary modules:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

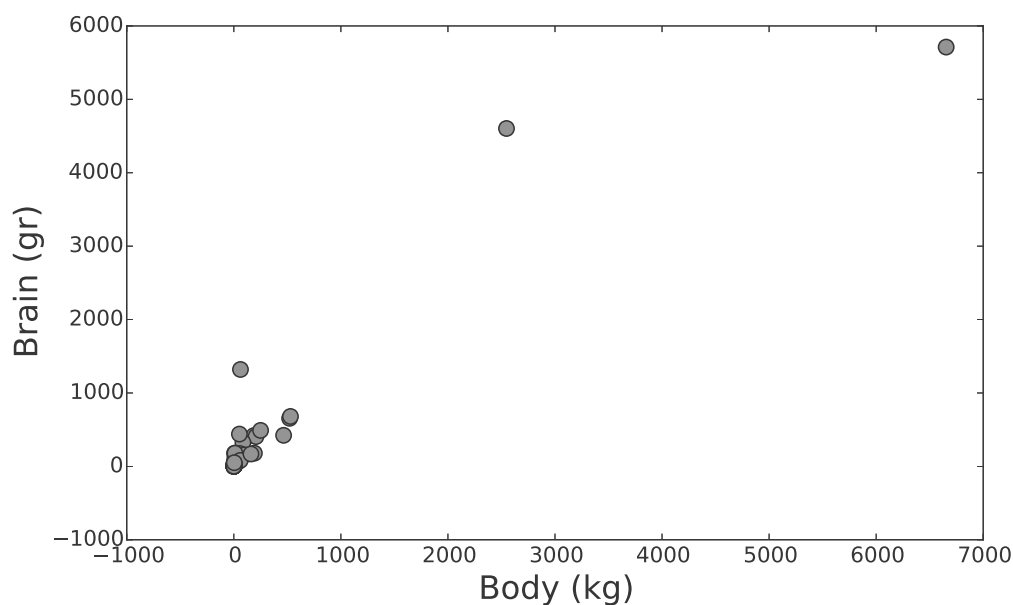
We start by importing the Python libraries that will help us manipulate the data.

We are importing NumPy and Matplotlib as well as pandas to start our analysis. Notice the aliases that we are giving to each of these libraries; they have become canonical.

Let us load the data into a pandas dataframe. We shall call this `mammals`. We can use the dataframe to visualise the data with a scatter plot as shown in [Figure 4.3](#):

```
mammals = pd.read_csv(u'./Data/mammals.csv')  
plt.scatter(mammals['body'], mammals['brain'])
```

Using the `read_csv` method from pandas, we can read our CSV file.



Let us take a look at the number of entries for each of the two variables we are considering. First the variable called `body`:

Figure 4.3: Scatter plot of the brain (gr) versus body mass (kg) for various mammals.

```
> body_data = mammals['body']
> body_data.shape

(62, )
```

The shape command allows us to see the size of the dataset imported into pandas.

We can apply the same command to the variable `brain`, and the result is as follows:

```
> brain_data = mammals['brain']
> brain_data.shape

(62, )
```

As we can see, there are 62 records in the dataset. We have assigned the values of each of the columns of the dataset to new variables for ease of reference.

Here, we will make use of the `statsmodels` module to perform our regression analysis. We show how to use Scikit-learn in [Section 4.4.1](#). First we have to add a column of 1s to the `body` variable so that we can accommodate for the calculation of the intercept, β_0 . `Statsmodels` has a handy function for this purpose: `add_constant`.

We make use of `statsmodels` here. You can see an example of using Scikit-learn in [Section 4.4.1](#).

```
import statsmodels.api as sm
body_data = sm.add_constant(body_data)
```

We make use of the `statsmodels` library to carry out the regression.

In the code above we are loading the `statsmodels` package and using the `sm` alias to refer to it. We are then adding a column of 1s to the `body_data` array making it a 62×2 array.

We are ready to run our regression with the ordinary least squares (OLS) method implemented in statsmodels:

```
regression1 = sm.OLS(brain_data, body_data).fit()
```

In particular, we are using the ordinary least squares method implemented in statsmodels.

If you are familiar with R, you are probably familiar with the “formula notation” used to refer to the dependency of a variable on appropriate regressors. So, if you have a function $y = f(x)$, in R you can denote that dependency with the use of a tilde, i.e., $\sim$. This is an easy way to deal with denoting the dependency among variables, and fortunately statsmodels has an API that allows us to make use of it in Python too:

```
import statsmodels.formula.api as smf
regression2 = smf.ols(formula = \
    'brain ~ body', \
    data = mammals).fit()
```

The formula API in statsmodels allows us to simplify the notation in the linear regression. It uses formulas similar to those in R.

Please note that the R-style formula does not require us to add the column of 1s to the independent variable. The regression coefficients obtained with both methods are the same, and do not forget that the `.fit()` command is needed in both cases.

Do not forget to use the `.fit()` command to carry out the regression.

Let us now take a look at the results of fitting the model to the data provided. Statsmodels provides a summary method that renders a nice-looking table with the appropriate information. Unfortunately the formatting of the output is more suitable for showing on a computer

screen than in a book page. Nonetheless, you are encouraged to try the following command in your shell:

```
regression1.summary()
```

You can run the same command for the second model we executed, and you will see the same results:

```
print(regression2.summary())
```

Among the information provided, you will have the following entries:

OLS Regression Results

```
=====
Dep. Variable:          brain
Model:                  OLS
Method:                 Least Squares
No. Observations:       62
R-squared:              0.873
Adj. R-squared:         0.871
```

OLS tells us the name of the dependent variable, the model used and the method as well as the number of observations used. It also tells us the value of the coefficient of determination, R^2 . The values of this number range between 0 and 1, and it tells us how well the data fit the model. A value of 1 is an indication that the regression line obtained fits the data perfectly well, whereas a value of 0 tells us that the linear model is not good. This measure is related to the

statsmodels provides a nice summary of the regression using the `summary()` command. The output is more suitable to be shown on a computer screen: Go ahead and try it in your Python implementation.

Part of the summary provided by OLS gives us information about the goodness of fitness via the R^2 coefficient.

R^2 is called the coefficient of determination.

Pearson correlation coefficient⁵ between the dependent and explanatory variables. We could formulate the linear regression model as a maximisation problem for R^2 .

⁵ Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

Having said that, there are some drawbacks with only looking at the value of R^2 . Namely, that it increases as we add more explanatory variables to the mix. We should therefore be careful when running regression models by adding extra features: An increase in the value of R^2 may not be due to the explanatory power of the input, but to the fact that we added that extra input. That is why OLS also provides information for the adjusted R^2 . It is very similar to R^2 , but it introduces a penalty as extra variables are included in the model. The adjusted R^2 value increases only in cases where the new input actually improves the model more than would be expected by pure chance.

Although the coefficient of determination provides an indication about how well the model fits the data, it should be used with care.

An alternative is the adjusted R^2 value.

For the case of our dataset, an $R^2 = 0.873$ is a pretty good outcome as 87% of the total variance in brain mass is explained by the linear regression model based on the body mass. This means that the regression line obtained must be a good one. And as yet we have not even mentioned what this line is and how it can be obtained from OLS.

	coef	std err	t	P> t	95% Conf. Interval	
Intercept	91.0044	43.553	2.090	0.041	3.886	178.123
body	0.9665	0.048	20.278	0.000	0.871	1.062

Table 4.1: Results from the regression analysis performed on the brain and body dataset.

In [Table 4.1](#) we can see the regression parameters obtained from running the ordinary least squares method on the brain and body dataset. The column named “coef” presents the estimated values of the coefficient for the model. Notice that since we are using data with named columns, the table clearly shows the intercept of our linear equation and the name of the variable (“body”). In case the data does not have column names, the variables will be presented with generic names. OLS lists the rest of the coefficients using the names of the variables included in the training data.

OLS lists the names of the variables as they appear in our data.

The “std err” column corresponds to the basic standard error of the estimate of the coefficient; “t” is the so-called t-statistic and it tells us how statistically significant the coefficient is⁶. The P-value is listed in the “P > |t|” column, and it helps us determine the significance of the results considering the null-hypothesis that the coefficient being equal to zero is true. A small P-value (typically ≤ 0.05) indicates strong evidence against the null hypothesis, and you should go with the value obtained for the coefficient. Finally “95% Conf. Interval” gives us the lower and upper values of the 95% confidence interval.

⁶ Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

A small P-value indicates evidence against the null hypothesis, and so it is rejected.

The results shown in [Table 4.1](#) indicate that the intercept for the model is $\beta_0 = 91.0044$, and the slope of the line is $\beta_1 = 0.9665$, leaving us with the following model:

$$\text{Brain} = 0.9665(\text{Body}) + 91.0044, \quad (4.20)$$

This is our equation for a line of best fit.

and the P-values obtained indicate a rejection of the null-hypothesis. We can get the parameters using the `params` method:

```
> regression2.params
```

```
Intercept      91.004396
mammals.body    0.966496
```

The regression parameters can be obtained with the `params` method of the fitted model object.

We can use this equation to predict the brain mass of a mammal given its body mass, and this can easily be done with the `predict` method in OLS. Let us consider new body mass measurements that will be used to predict the brain mass using the model obtained above. We need to prepare the new data in a way that is compatible with the model. We can therefore create an array of ten new data inputs as follows:

```
new_body = np.linspace(0, 7000, 10)
```

If we used the formula API to train our model, the `predict` method does not require us to add a column of 1s to our data, and instead we simply indicate that the new data points are going to be treated as a dictionary to replace the independent variable (i.e., `exog` in statsmodels parlance) in the fitted model. In other words, you can type the following commands:

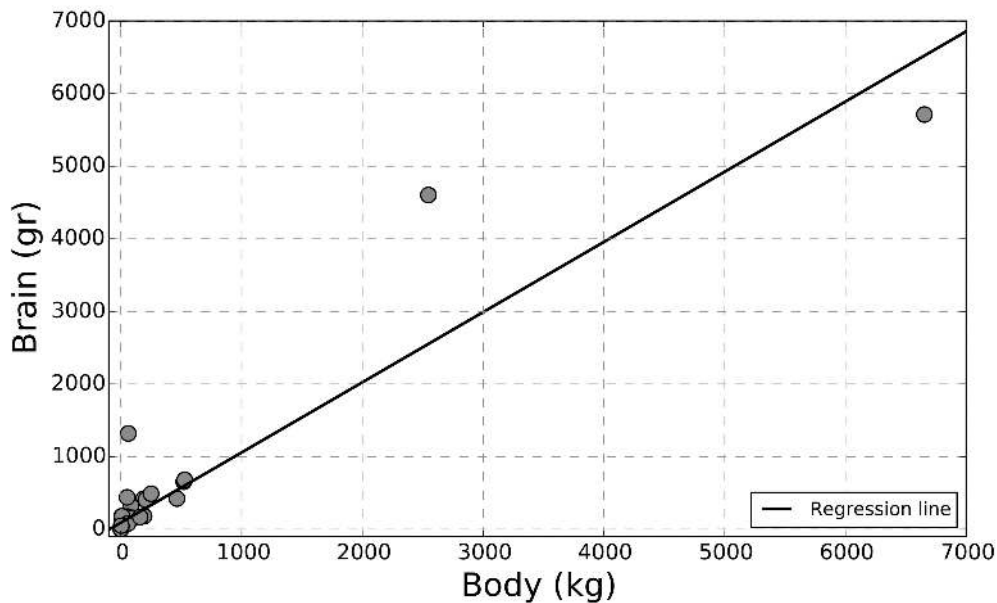
```
brain_pred=regression2.predict(exog=\
    dict(body=new_body))
print(brain_pred)
```

The `predict` method of the StatsModel formula API does not need the addition of a column of 1s.

`exog` refers to the independent variable.

This will generate the following output:

```
array([91.00439621, 842.72379329,
      1594.44319036, 2346.16258744,
      3097.88198452, 3849.6013816 ,
      4601.32077868, 5353.04017576,
      6104.75957284, 6856.47896992])
```



The numbers shown correspond to the brain mass predictions for the artificial body mass measurements used as input. In [Figure 4.4](#) we can see the regression line given by Equation (4.20) in comparison to the data points in the set. Please note that if you are not using the formula API, the input data will require the addition of a column of 1s to obtain the intercept.

Figure 4.4: Scatter plot and the regression line calculated for the brain (gr) versus body mass (kg) for various mammals.

We can use the values above to construct the line of best fit given by the model.

So far so good, but can we do better than that? For example, look at the clustering that happens in the region below 1,000 kilogram mark for body mass and compare it with the ones that take place after the 2,000 or 6,000 kilogram marks. Are the latter outliers, or can we come up with a better model that encompasses these differences? Let us take a look at a typical transformation carried out in a variety of analyses. But before that, let us use Scikit-learn.

4.4.1 *Regression with Scikit-learn*

WE HAVE SEEN HOW TO use statsmodels to perform our linear regression. One of the reasons to use this module is the user-friendly output it generates. Nonetheless, this is not the only way available to us to perform this analysis. In particular, Scikit-learn is another very useful module, and one that we use extensively throughout the book. For completeness, in this section we will see how to perform linear regression with Scikit-learn.

We can implement a regression model using Scikit-learn.

Let us import the modules to read the data. We will use pandas to load the data into a dataframe called mammals:

```
import numpy as np
import pandas as pd

mammals = pd.read_csv(u'./Data/mammals.csv')
```

This is exactly the as same we did in the previous section.

As we discussed in [Section 3.11](#), we know that Scikit-learn expects data to be represented by two-dimensional numeric matrices with M data instances (rows) and N distinct

features (columns). In this case, we have 62 instances and one feature. Let us arrange the data as expected by creating appropriate arrays for the dependent and independent variables:

```
body_data = mammals[['body']]
brain_data = mammals[['brain']]
```

Notice the use of double brackets to get the right shape for the arrays.

We are now ready to create our model. First, we need to create an instance of a linear regression model from `linear_model` in Scikit-learn. This can be done as follows:

```
from sklearn import linear_model
sk_regr = linear_model.LinearRegression()
```

We carry out linear regression modelling with `linear_model` in Scikit-learn.

With that in place, we simply use the `fit` method for the model and we are done:

```
sk_regr.fit(body_data, brain_data)
```

We can now check that the intercept and coefficient obtained are the same as the ones we calculated with statsmodels. We can also check the value of the R^2 coefficient:

```
> print(sk_regr.coef_)
[[ 0.96649637]]

> print(sk_regr.intercept_)
[ 91.00439621]

> print(sk_regr.score(body_data, brain_data))
0.872662
```

The coefficients and intercept are obtained with the `.coef_` and `.intercept_` methods.

The R^2 coefficient is calculated with the `.score` method.

Finally, the predictions can be calculated with the predict method:

```
new_body = np.linspace(0, 7000, 10)
new_body = new_body[:, np.newaxis]
brain_pred = sk_regr.predict(new_body)
```

Note the use of `np.newaxis` to render the Numpy array into the format expected by Scikit-learn.

4.5 Logarithmic Transformation

ONE OF THE PRINCIPAL TENETS of the linear regression model is the idea that the relationship between the variables at play is linear. In cases when that is not necessarily true, we can apply manipulations or transformations to the data that result in having a linear relationship. Once the linear model is obtained, we can then undo the transformation to obtain our final model.

Hence the name!

A typical transformation that is often used is applying a logarithm to either one or both of the predictive and response variables.

Applying the logarithmic function is a typical way to transform our data.

Let us see what happens to the scatter plot of the body and brain data we have been analysing when we apply the logarithmic transformation to both variables. We will create a couple of new columns in our pandas dataframe to keep track of the transformations performed:

```
from numpy import log
mammals['log_body'] = log(mammals['body'])
mammals['log_brain'] = log(mammals['brain'])
```

Remember that `log` in Python refers to the base-*e* logarithm.

We can plot the transformed data, and as we can see from [Figure 4.5](#) the data points are aligned in a way that indicates a linear relationship in the transformed space.

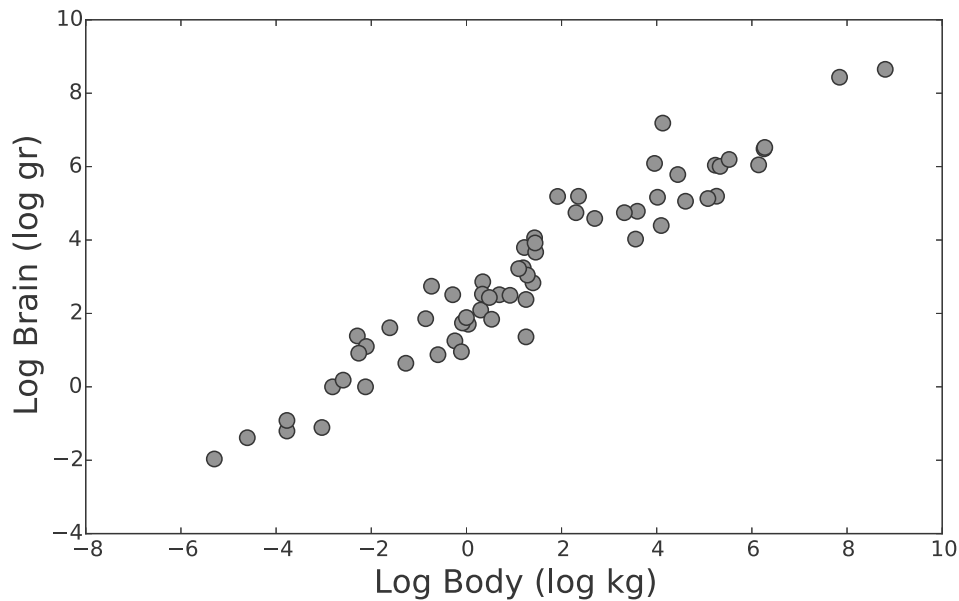


Figure 4.5: Scatter plot in a log-log scale for the brain (gr) versus body mass (kg) for various mammals.

Why has this happened? Well, remember that we are trying to use models (simple and less simple ones) that enable us to exploit the patterns in the data. In this case, the relationship we see in this data may be modelled as a *power law*; e.g., $y = x^b$.

The transformation has helped us convert our problem into a simpler one.

The log-log transformation applied to the data maps this nonlinear relationship to a linear one, effectively transforming the complicated problem into a simpler one:

$$\log(y) = \log(x^b), \quad (4.21)$$

$$\begin{aligned} \log(y) &= b \log(x), \\ y' &= bx', \end{aligned} \quad (4.22)$$

where we are using the notation $\log$ for the inverse of the exponential function e . As we can see, we have transformed our power law model, a nonlinear model in the regressor, into a form that looks linear as shown in Equation (4.22).

Now that we have this information at hand, we can train a new model using the transformed features attached to the `mammals` dataframe:

```
log_lm=smf.ols(formula = 'log_brain ~ log_body',\
               data = mammals).fit()
```

If we print the `log_lm.summary()`, we will see some of the following information:

```
OLS Regression Results
=====
Dep. Variable:          log_brain
Model:                  OLS
Method:                 Least Squares
No. Observations:      62
R-squared:              0.921
Adj. R-squared:        0.919
```

The logarithmic transformation has increased the value of R^2 from 0.873 to 0.921. We can see the value of the sum of

In engineering and other disciplines, they use the notation $\ln$ for this function.

The only difference here is that we are using the transformed variables in the OLS function.

However, the results returned by OLS are remarkably different.

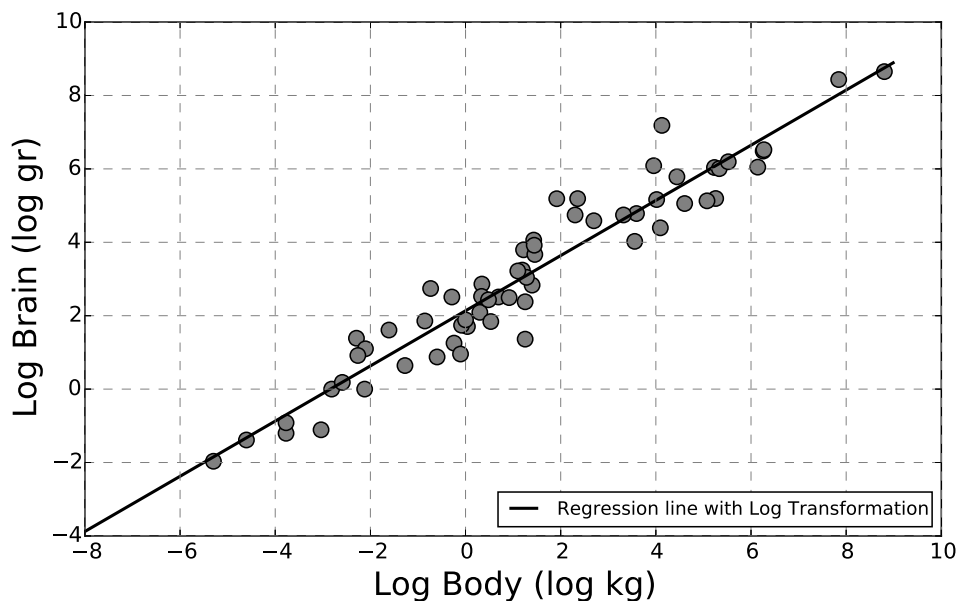
squared residuals with the following command:

```
> log_lm.ssr
```

```
28.9227104215
```

We can obtain the sum of the square residuals with the `ssr` method.

Let us now take a look at the statistics for the model as well as the all-important coefficients. As we can see from [Table 4.2](#), the new model has an intercept of $\beta_0 = 2.1348$ and a slope of $\beta_1 = 0.7517$. The regression line for this model can be seen in [Figure 4.6](#).



Remember that the coefficients obtained above are for the transformed data. If we wanted to relate this to the original features, we would need to undo the transformation. In this

Figure 4.6: Log-log scale figure with a scatter plot and the regression line calculated for the brain (gr) versus body mass (kg) for various mammals.

	coef	std err	t	P> t	95% Conf. Interval	
Intercept	2.1348	0.096	27.227	0.000	1.943	2.327
log_body	0.7517	0.028	26.409	0.000	0.695	0.809

Table 4.2: Results from the regression analysis performed on the brain and body dataset using a log-log transformation.

case we have a model given by

$$Brain = A(Body)^{0.7517}, \tag{4.23}$$

where $A = e^{2.1348}$.

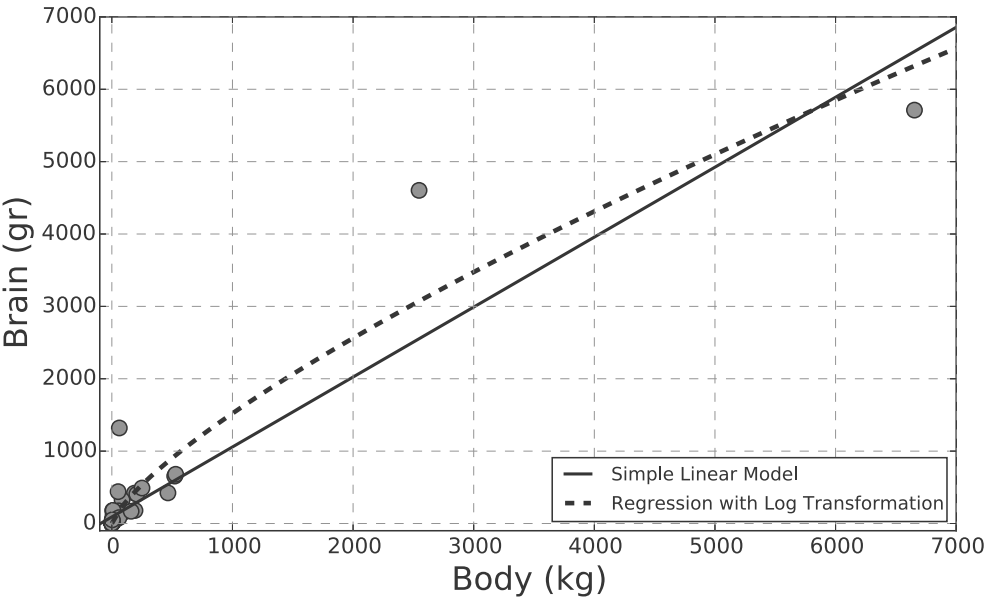


Figure 4.7: Comparison of the simple linear regression model and the model with logarithmic transformation for the brain (gr) versus body mass (kg) for various mammals.

In [Figure 4.7](#) we can see the original scatter plot and a comparison of the two models. It is easy to see how the logarithmic transformation allows for greater flexibility, capturing those data points which are a struggle for the simple linear model. This comparison demonstrates that it is possible to build a variety of models to explain the behaviour we observe in the data. In [Section 4.7](#) we will see how we can fit a polynomial regression to the same dataset. Nonetheless, please remember that carrying out appropriate splitting of training and testing sets, together with cross-validation, is an unrivalled way to decide which model, among those tried, is the most suitable to use with unseen data.

The logarithmic transformation enabled us to capture those data points which are a struggle for the linear model.

4.6 *Making the Task Easier: Standardisation and Scaling*

GIVEN THAT THE MAIN UNDERLYING concept behind linear regression is the assumption of a linear relationship, transformations such as the one covered in the previous section make the task easier for both the learning algorithm and for us. As you can imagine, there may be many more tricks up our jackalope sleeves to transform and pre-process the data in order to facilitate our modelling. In this section we are going to present a couple of the most widely used techniques to transform our data and provide us with anchors to interpret our results.

Remember that the main assumption of linear regression is the existence of a linear relationship.

Data pre-processing is nothing new in the data science workflow.

One of those techniques consists on centring the predictors such that their mean is zero, and it is often used in

regression analysis. Among other things, it leads to interpreting the intercept term as the expected value of the target variable when the predictors are set to zero. Another useful transformation is the scaling of our variables. This is convenient in cases where we have features that have very different scales, where some variables have large values and others have very small ones.

Centring the data around their mean leads us to interpret the intercept as the expected value of the target variable when the predictors are set to zero.

As mentioned above, standardisation and scaling may help us interpret our results: They allow us to transform the features into a comparable metric with a known range, mean, units and/or standard deviation. It is important to note that the transformations to be used depend on the dataset and the domain where the data is sourced from and applied to. It also depends on the type of algorithm and answer sought. For example, in a comprehensive study of standardisation for cluster analysis, Milligan and Cooper⁷ report that standardisation approaches that use division by the range of the feature provide a consistent recovery of clusters. We shall talk about clustering in the next chapter. Let us go through the two techniques mentioned above in a bit more detail.

The transformations to be applied depend on the dataset and the domain of application.

⁷ Milligan, Glenn W. and Cooper, Martha C. (1988). A study of standardization of variables in cluster analysis. *Journal of Classification* 5(2), 181–204.

4.6.1 Normalisation or Unit Scaling

THE AIM OF THIS TRANSFORMATION is to convert the range of a given variable into a scale that goes from 0 to 1. Given a feature f with a range between f_{min} and f_{max} the

transformation is given by:

$$f_{scaled} = \frac{f - f_{min}}{f_{max} - f_{min}}. \quad (4.24)$$

Unit scaling transforms our data into a scale between 0 and 1.

Notice that this method of scaling will cast our features into equal ranges, but their means and standard deviations will be different.

Unit scaling leaves the means and standard deviations unchanged.

An alternative formulation divides each feature by its range without subtracting the minimum value. We can apply this unit scaling to our data with the preprocessing method in Scikit-learn that includes the `MinMaxScaler` function to implement unit scaling.

```
from sklearn import preprocessing
```

Scikit-learn comes with a preprocessing module.

Once we have loaded the appropriate function, we can apply the scaling as follows:

```
scaler = preprocessing.MinMaxScaler()

mammals_minmax = pd.DataFrame(\
    scaler.fit_transform(\
        mammals[['body', 'brain']]),\
    columns = ['body', 'brain'])
```

Scikit-learn includes `MinMaxScaler` to implement unit scaling.

Let us see the minimum and maximum values of the transformed data:

```
> mammals_minmax.groupby(lambda idx: 0).\
  agg(['min', 'max'])
```

	body		brain	
	min	max	min	max
0	0.0	1.0	0.0	1.0

Here we are using pandas to manipulate our dataset.

4.6.2 z-Score Scaling

AN ALTERNATIVE METHOD FOR SCALING our features consists of taking into account how far away data points are from the mean. In order to provide a comparable measure, the distance from the mean is calculated in units of the standard deviation of the feature data.

The standard deviation is a measure of dispersion.

In this case a positive score tells us that a given data point is above the mean, whereas a negative one is below the mean. The standard score explained above is called the z-score as it is related to the normal distribution. The transformation that we need to carry out for a feature f with mean μ_f and standard deviation σ_f is given by:

z-score scaling is related to the normal distribution.

$$f_{z\text{-score}} = \frac{f - \mu_f}{\sigma_f}. \quad (4.25)$$

Strictly speaking, the z-score must be calculated with the mean and standard deviation of the population; otherwise, we are making use of Student's t statistic⁸.

Scikit-learn's pre-processing method allows us to standardise our features in a very straightforward manner:

⁸ Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

```

scaler2 = preprocessing.StandardScaler()

mammals_std = pd.DataFrame(\
    scaler2.fit_transform(mammals[['body', 'brain']]), \
    columns = ['body', 'brain'])

```

Scikit-learn includes
StandardScaler to implement
z-score scaling.

After the transformation we should have features with 0 mean and standard deviation 1; let us check that this is the case:

```

> mammals_std.groupby(lambda idx: 0).\
    agg(['mean', 'std'])

```

	body		brain	
	mean	std	mean	std
0	1.790682e-18	1.008163	-3.223228e-17	1.008163

Once again, the aggregation
shown here uses pandas.

4.7 Polynomial Regression

IN THE PREVIOUS SECTION WE have seen how a simple transformation in the input and output variables makes a complex model into a simpler one. In fact, we can try fitting different models using more and more complex functions. One important point to note is that a model is said to be linear when it is linear *in the parameters*. With that in mind, the model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon, \quad (4.26)$$

A model is said to be linear when
it is linear in the parameters.

and the model

$$y = \beta_0 + \beta_1 x + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 + \varepsilon \quad (4.27)$$

With this kind of models we can then talk about polynomial regression.

are both linear, as the parameters β_i are linear. In the case of the examples above, the models are given by second-order polynomials in one and two variables, respectively. When using such models to fit our data, we talk therefore about *polynomial regression* and in general the k -th order polynomial model in one variable is given by:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_k x^k + \varepsilon. \quad (4.28)$$

This is general polynomial model.

The techniques for fitting a linear regression model can be used for the models above too.

Polynomial models can be very useful in cases where we know that nonlinear effects are present in the target variable. A polynomial model is effectively the Taylor series expansion of an unknown function and thus can be used to approximate it. Furthermore, it is possible to use different orthogonal functions to define the model. For instance, if we decide to use trigonometric functions, we end up effectively in the realm of harmonic analysis, and the regression would give us the coefficients we would obtain via a Fourier transform.

A polynomial model is effectively the Taylor series expansion of an unknown function.

Let us fit a quadratic model to the brain and body dataset we have been using in the previous sections. We can start by adding a feature that corresponds to the square of the body mass:


```
mammals['body_squared']=mammals['body']**2
```

We are trying a quadratic model with our test dataset.

We can now fit the quadratic model given by Equation (4.26), and using statsmodels is a straightforward task:

```
poly_reg=smf.ols(formula=\
    'brain~body+body_squared',\
    data=mammals).fit()
```

The application of OLS remains the same.

We can take a look at the parameters obtained:

```
> print(poly_reg.params)

Intercept      19.115299
body           2.123929
body_squared   -0.000189
```

The parameter for the quadratic term seems to be small.

In other words, we have a model given by

$$\text{Brain} = 19.115 + 2.124(\text{Body}) - 1.89 \times 10^{-4}(\text{Body})^2. \quad (4.29)$$

It may seem that the coefficient of the quadratic term is rather small, but it does make a substantial difference to the predictions. Let us take a look by calculating the predicted values and plot them against the other two models:

```
poly_brain_pred=poly_reg.predict(exog=\
    dict(body=new_body,\
    body_squared=new_body**2))
```

However, it makes a substantial difference to the predictions made.

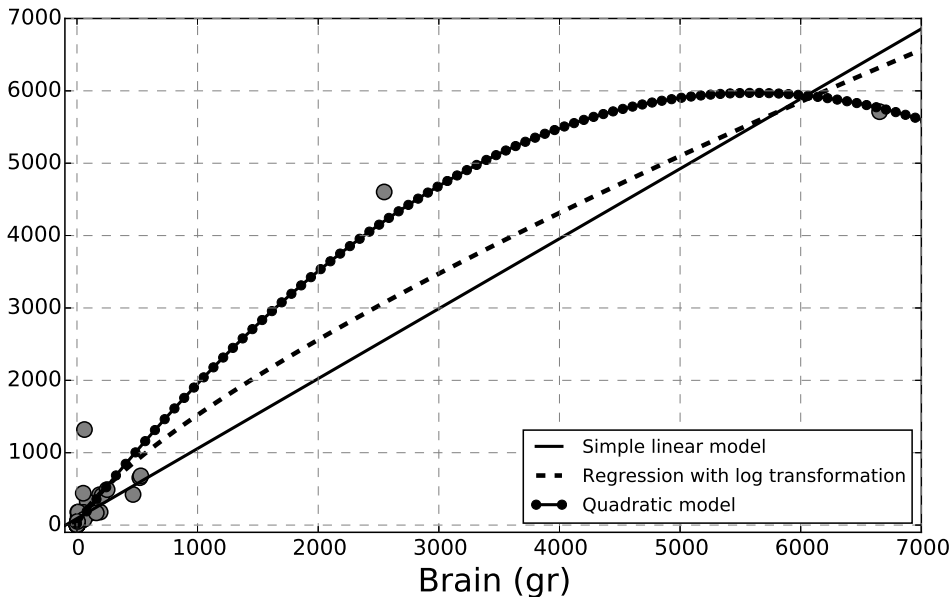


Figure 4.8: Comparison of a quadratic model, a simple linear regression model and a model with logarithmic transformation fitted to the brain (gr) versus body mass (kg) for various mammals.

As we can see in [Figure 4.8](#), the polynomial regression captures the data points much closer than the other two models. However, by increasing the complexity of our model, we are running the risk of overfitting the data. We know that cross-validation is a way to avoid this problem, and other techniques are at our disposal, such as performing some feature selection by adding features one at a time (forward selection) or discarding non-significant ones (backward elimination). In [Section 4.9](#) we shall see how feature selection can be included in the modelling stage by applying regularisation techniques.

We have to bear in mind that increasing the complexity of our model, increases the chances of overfitting.

When using polynomial regressions, there are a number of things that should be taken into account. For instance, the

order of the polynomial model should be kept as low as possible; remember that we are trying to generalise and not run an interpolation.

Once a model is obtained, take particular care not to overuse the model; extrapolating with the aid of a polynomial model is a perilous task. There are other technical issues to be aware of. For instance, as the order of the polynomial increases, matrix inversion calculations become inaccurate; this is a form of ill-conditioning, and it introduces errors in the estimation of the parameters.

Extrapolating with a polynomial model is a perilous task.

Another aspect to consider is that if the values of the independent variables are limited to a narrow range, there can also be significant ill-conditioning of the problem, or multicollinearity in the features used to train the model.

Multicollinearity is the name we give to the situation where two or more features in our model are highly (or even moderately) correlated with each other. This becomes aggravated in a polynomial regression as higher powers of a feature are highly correlated with each other. Let us look at this empirically by taking the 9th and 10th powers of an array and calculating their correlation:

Multicollinearity arises when two or more features are highly correlated with each other.

```
x = np.random.random_sample(500)

x1, x2 = x**9, x**10

cor_mat = np.corrcoef(x1,x2)
```

We obtain an array of 500 random samples and calculate their 9th and 10th powers.

Let us take a look at the correlation matrix:

```
> print(cor_mat)

[[ 1.          0.99877083]
 [ 0.99877083  1.          ]]
```

As we can see, the correlation between x^9 and x^{10} is high.

As we can see for the random numbers generated by my computer, the correlation coefficient between x_1 and x_2 is quite close to 1. Multicollinearity results in having wide swings in the values of the parameter estimations when small changes in the data are included. Also, the coefficients obtained may be such that their standard errors are quite high with low significance levels, although they are actually jointly significant and the R^2 is high.

The numbers may be different in your computer.

It is important to note that multicollinearity is not exclusive of the use of polynomial models. In fact, it is quite possible that two seemingly independent features included in our data are highly correlated among themselves, having a confounding effect in our model, and thus we should avoid using these features together in our model.

Remember that multicollinearity is not exclusive of polynomial models.

4.7.1 Multivariate Regression

IN THE EXAMPLES SO FAR, we have concentrated mainly on regression models that have one single independent variable to explain the target we are interested in. As mentioned earlier in this chapter, in the case where we have more than one input variable we are entering the realm of multivariate regression.

Multivariate regression refers to having more than one input variable in our model.

In a sense we have already – indirectly – seen an example of a multivariate regression in [Section 4.7](#) when we addressed the ideas behind polynomial regression. In that case, the added features were powers of a single input variable. For the more general case of a multivariate regression, the features are given by different independent variables.

Polynomial regression is effectively a multivariate regression problem.

If we consider a set of M predictors $x_1, x_2, x_3, \dots, x_M$ that are hypothesised to be related to a response variable y , the multivariate regression model can be expressed as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_r x_M + \varepsilon, \quad (4.30)$$

A general multivariate linear model.

and the best part is that the parameter estimation for this model can be achieved with the same techniques discussed in [Section 4.3](#).

We can continue using statsmodels for polynomial regression.

This means that we can continue using the same statsmodels libraries described in the previous sections. Also, as before, multicollinearity should be avoided when considering the various independent features to be included in our model.

4.8 Variance-Bias Trade-Off

NOW THAT WE HAVE EXPLORED the ideas behind describing the relationship among variables with a model of the type given the expression:

$$y = f(x) + \varepsilon, \quad (4.31)$$

we can take a look at the expected prediction error obtained when estimating a model $\hat{f}(x)$. This is given by the expectation of the squared error

$$E \left[\left(y - \hat{f}(x) \right)^2 \right]. \quad (4.32)$$

The expectation of the squared error can be decomposed into bias, variance and noise.

This expectation value can be decomposed into portions that correspond to bias, variance and noise, respectively.

In order to facilitate the decomposition, let us first consider a random variable Z with a probability distribution given by $P(Z)$. We denote the expectation value of Z as $E[Z]$. Let us calculate the expectation value of $(Z - E[Z])^2$:

$$\begin{aligned} E \left[(Z - E[Z])^2 \right] &= E \left[Z^2 \right] - 2E[Z]E[Z] + E^2[Z], \\ &= E \left[Z^2 \right] - 2E^2[Z] + E^2[Z], \\ &= E \left[Z^2 \right] - E^2[Z] \end{aligned}$$

We will use the result of this calculation later on.

and thus

$$E[Z^2] = E[(Z - E[Z])^2] + E^2[Z]. \quad (4.33)$$

Using the expression above, we can now take a look at the decomposition of the expectation of the squared error:

$$\begin{aligned}
E \left[\left(y - \hat{f}(x) \right)^2 \right] &= E \left[y^2 - 2\hat{f}(x) + \hat{f}^2(x) \right], \\
&= E \left[y^2 \right] - 2E[y]E \left[\hat{f}(x) \right] + E \left[\hat{f}^2(x) \right], \\
&= E \left[(y - E[y])^2 \right] + E^2[y] \\
&\quad - 2E[y]E \left[\hat{f}(x) \right] \\
&\quad + E \left[\left(\hat{f}(x) - E \left[\hat{f}(x) \right] \right)^2 \right] \\
&\quad + E^2 \left[\hat{f}(x) \right] \\
&= E \left[\left(\hat{f}(x) - E \left[\hat{f}(x) \right] \right)^2 \right] + \quad (4.34) \\
&\quad \left(E[y] - E \left[\hat{f}(x) \right] \right)^2 + \quad (4.35) \\
&\quad E \left[(y - E[y])^2 \right] \quad (4.36)
\end{aligned}$$

where the first term (4.34) corresponds to **variance**, the second (4.35) to the square of the **bias** and finally the third (4.36) is the **noise**.

This decomposition shows that apart from the noise, there are two sources of error in our model. Our task is the minimisation of these two error sources that preclude our algorithm from generalising.

We need to find a balance between the variance and bias.

On the one hand, variance tells us how sensitive the model is to small fluctuations in the training set; on the other hand bias is related to the difference between the expected value of our estimator and its true value. High variance results in overfitting, whereas high bias results in underfitting. Finding a good model is therefore a matter of balancing the

High variance gives us more complex models, whereas high bias yields simpler ones.

bias and the variance. This tradeoff applies to algorithms used in supervised learning.

4.9 Shrinkage: LASSO and Ridge

THE DECOMPOSITION OF OUR PREDICTION error into its variance and bias components makes it clear that a balance between the two is required for any regression problem we may encounter. In general, linear regression exhibits high variance and low bias, and it should therefore stand to reason that lowering the variance at the expense of the bias is the way to go.

Linear regression exhibits high variance and low bias.

Furthermore, we have also seen that our ability to interpret the outcome when adding more and more features is diminished. It would be therefore preferable to identify those features that are deemed to be the most important ones. Unfortunately, our linear regression model, as it stands at the moment, does not allow us to do this automatically.

Feature selection is not possible within the straightforward linear model.

Let us recall that given our model

$$y_i = \beta_0 + \sum_{j=1}^M \beta_j x_j + \varepsilon_i, \quad (4.37)$$

This is the linear model which we have been using all along.

we are interested in choosing the coefficients β_0 and β_j in order to minimise the Ordinary Least Squares (OLS) criterion given by Equation (4.8), which is just the sum of squared errors. The coefficients are in effect a way of determining whether a particular feature is important or

The model coefficients are a way to determine if a feature is important or not.

not. In particular, the closer the coefficient is to zero the less significant the feature is.

Let us then consider replacing the estimates with a smaller value such that

$$\tilde{\beta}_k = \frac{1}{1 + \lambda} \beta_k. \quad (4.38)$$

The parameter λ lets us tune the value of the coefficients.

When $\lambda = 0$ our coefficients are unchanged, and as λ gets larger and larger, the coefficients start shrinking down to zero. In this manner, with the right choice of λ we can get an estimator with an improved error. The estimate is biased, but remember that we were happy to sacrifice some of that to make up for the variance.

The new estimates are biased though.

Shrinkage of the coefficients is therefore a form of regularisation as we penalise the model for increased complexity as given by the size of the coefficients. In [Section 3.9](#) we introduced the L_2 - and L_1 -norms, and it is thus natural to consider these measures for the size of the coefficients.

Shrinkage is a form of regularisation.

The use of the L_2 -norm results in the so-called **ridge regression**⁹:

$$\hat{\beta}^{ridge} = \min \left\{ |\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}|_2^2 + \lambda |\boldsymbol{\beta}|_2^2 \right\}, \quad (4.39)$$

⁹ Hoerl, A. E. and R. W. Kennard (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*. 12(3), 55–67.

whereas the application of the L_1 -norm leads to the Least Absolute Shrinkage and Selection Operator (**LASSO**)¹⁰

$$\hat{\beta}^{lasso} = \min \left\{ |\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}|_2^2 + \lambda |\boldsymbol{\beta}|_1 \right\}. \quad (4.40)$$

¹⁰ Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *J. R. Statist. Soc. B*. 58(1), 267–288.

In both cases, as the value of λ is increased, the bias increases, whereas the variance decreases. It controls the amount of penalty imposed on the model and therefore it is important to find a good value for this parameter. **Model selection** is the process of finding the appropriate value for the hyperparameter, and cross-validation is a good way to tackle this problem.

The tuning parameter λ is sometimes called the hyperparameter.

We discussed cross-validation in [Section 3.13](#).

For instance, using k -fold cross-validation and a set of possible hyperparameter values $\lambda \in \{\lambda_1, \dots, \lambda_m\}$, we partition our data in K folds: $F_1, F_2, \dots, F_K$. For each value of $k = 1, \dots, K$, we train on the feature values in the training set F_i with $i \neq k$ and validate on the feature values in F_k .

A quick recipe to find a suitable value for the hyperparameter λ , using cross-validation.

For each value in the set λ , we compute our estimate on the training set as well as the error on the validation set and compute the average error over all folds. The latter provides us with a curve that corresponds to the cross-validation error. The value of the hyperparameter to choose is such that it minimises the cross-validation error itself which in turns corresponds to the best score for the model.

Do not be deceived by the similar look between the ridge and LASSO regressions; the solutions do have significant differences. Although the ridge regression works well in cases where there are coefficients whose values are actually close to zero, the algorithm never explicitly sets them to this value.

Ridge and LASSO differ on the type of penalty imposed.

Unless $\lambda = \infty$.

This means that in some cases, with ridge regression the feature selection wished for is not possible, particularly when there are a large number of features involved. In the

case of LASSO, the usage of the $L1$ -norm as the penalty means that it is possible for some of the coefficients to be shrunk down to zero, making feature selection possible.

Feature selection is possible with LASSO.

It is important to remember that if the features included in our model are not on the same scale, the estimates obtained with both ridge and LASSO are not fair. In those cases it is recommended to use some scaling as described in [Section 4.6](#). Let us now see how we can run ridge and LASSO regressions in Python. In this case we are going to use Scikit-learn to carry out the modelling.

We discussed scaling in [Section 4.6](#).

We will continue working with the body and brain dataset we have been using all along, and in order to make things more interesting, we will use a feature that corresponds to the cube of the body size. We can do this as follows:

Assuming that we have already added the square of the body size.

```
mammals['body_cubed']=mammals['body']**3
```

A contrived feature, but it will serve our purposes for a demo.

Let us first start by scaling our data using z-score scaling:

```
from sklearn import preprocessing
X = mammals[['body', 'body_squared', 'body_cubed']]
Y = mammals[['brain']]

Xscaled = preprocessing.\
    StandardScaler().fit_transform(X)
Yscaled = preprocessing.\
    StandardScaler().fit_transform(Y)
```

As we know, this scaling can be done with `StandardScaler`.

Not only are we interested in finding the coefficients that describe each of the ridge and LASSO models, but we

also want to find a good value for λ in each case. In order to do that, we will need to carry out the cross-validation procedure as described earlier in this section.

Fortunately, Scikit-learn provides us with `GridSearchCV` which is a helpful function that lets us perform an exhaustive search over specified parameter values by implementing a fit and a score methods. The latter will let us choose the value for our hyperparameter λ . First, we need to create our test and training sets:

```
from sklearn import model_selection as ms

XTrain, XTest, yTrain, yTest = \
    ms.train_test_split(Xscaled, Yscaled, \
        test_size= 0.2, random_state=42)
```

Scikit-learn lets us carry out an exhaustive search of parameter combinations with the `GridSearchCV` method.

We split our data into training and testing sets.

We can now define a set of parameters to be used in our search:

```
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge, Lasso

lambda_range = np.linspace(0.001, 0.2, 25)
lambda_grid = [{'alpha': lambda_range}]
```

We need to define a dictionary that holds the set of values to be searched. Note that Scikit-learn refers to the hyperparameter as α .

Our search will use each of the values in the `lambda_grid` dictionary with values ranging between 0.001 and 0.2, and carry our cross-validation with the number of folds desired. Let us try with 20 folds as follows:

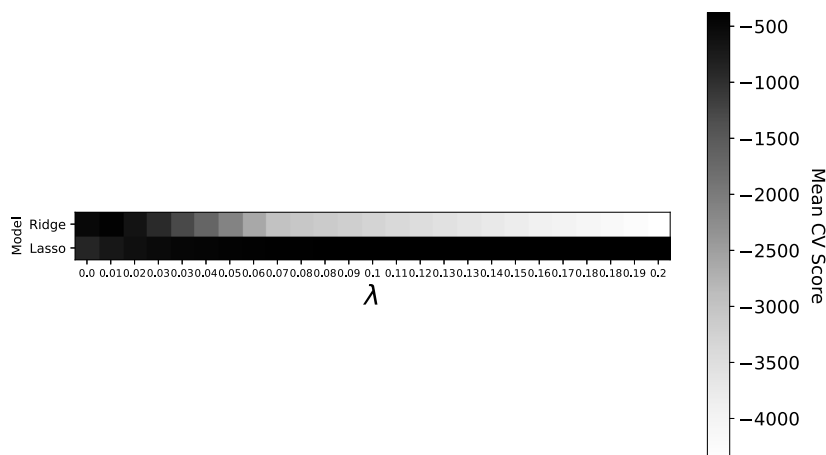
```

model1 = Ridge(max_iter=10000)
cv_ridge = GridSearchCV(estimator=model1,\
    param_grid=lambda_grid,\
    cv=ms.KFold(n_splits=20))
cv_ridge.fit(XTrain, yTrain)

model2 = Lasso(max_iter=10000)
cv_lasso = GridSearchCV(estimator=model2,\
    param_grid=lambda_grid,\
    cv=ms.KFold(n_splits=20))
cv_lasso.fit(XTrain, yTrain)

```

GridSearchCV takes care of the cross-validation step using the set of parameters to be searched. In this case we are using k -fold cross-validation.



In [Figure 4.9](#) we show a heatmap with the values for the hyperparameters used in our search and their method, as well as their corresponding cross-validation mean scores. We can obtain the actual values selected with the help of the `best_params_` method as follows:

Figure 4.9: Using GridSearchCV, we can scan a set of parameters to be used in conjunction with cross-validation. In this case we show the values of λ used to fit ridge and LASSO models, together with the mean scores obtained during modelling.

```

> bestLambda_ridge = cv_ridge.best_params_['alpha']
> bestLambda_lasso = cv_lasso.best_params_['alpha']
> print('Best lambda for ridge model is', \
      bestLambda_ridge)
> print('Best lambda for LASSO model is', \
      bestLambda_lasso)

Best lambda for ridge model is 0.13366666666666668
Best lambda for LASSO model is 0.009291666666666666

```

Finding the best hyperparameter λ for our models.

We can now use these parameters with the corresponding models to extract the coefficients:

```

> Brain_Lasso = Lasso(alpha=bestLambda_lasso,\
  max_iter=10000)
> Brain_Lasso.fit(XTrain,yTrain)
> print(Brain_Lasso.coef_)
[ 1.65028091 -0.          -0.76712502]

```

Similar code can be written for the ridge model.

As we can see, the second coefficient has been shrunk down to zero with the application of the LASSO regression.

Finally, let us take a look at the residual sum of squares obtained with the test dataset:

```

> lasso_prediction = Brain_Lasso.predict(XTest)
> print('Residual sum of squares: {0:4f}'.format(\
  np.mean((lasso_prediction - yTest) ** 2)))

Residual sum of squares: 0.0114

```

Once again, similar code can be written for the ridge model.

We can also look at the explained variance score:

```
> print('Variance score: {0:.4f}'.format(\
    Brain_Lasso.score(XTest, yTest)))

Variance score: -0.3388
```

4.10 *One of a Kind: Regression with Categorical Features*

IN THIS CHAPTER WE HAVE discussed how to run different kinds of regression analyses, and by and large we have used continuous features in them. However, it is possible to include categorical features too, but we need to be mindful of the way in which we interpret the results.

Here we see how to use categorical variables in a regression analysis.

In this case we are going to use a dataset that captures information about the weight, eating and exercise habits for some participants in a survey¹¹. The data is available at <https://doi.org/10.6084/m9.figshare.25820893.v1>. We will assume that the data is stored in a CSV file called `weight.csv`, and it contains the following features:

¹¹ Rogel-Salazar, J. (2024, May). Weight Data. <https://doi.org/10.6084/m9.figshare.25820893.v1>

- `gender` - Gender of the participant (1, 2)
- `exercise` - Exercise habits (A - always, B - sometimes, C - never)
- `breakfast` - Do you eat breakfast? (Y/N)
- `weight` - Participant weight in kilograms

The features in this dataset are mainly categorical.

Let us upload the data into pandas and take a look at the contents:

```
> weight = pd.read_csv(u'wight.csv')
> weight.head()
```

	gender	exercise	breakfast	weight
0	1	C	Y	74.8
1	2	B	Y	77.1
2	2	A	Y	49.9
3	2	A	N	56.7
4	2	B	Y	72.6

We have four categorical features in this dataset: Gender, exercise and breakfast.

The dataset contains four categorical features: Gender, exercise, breakfast, and one continuous variable: Weight. To use the information in the dataset, we will need to ensure that the data is in numerical format. As such we need to be careful to transform the data and continue treating as categorical the features that encode categories. We will not use the gender variable in this section, but since it is codified in numerical values, in principle we can use the information as is. We will concentrate on the other two features and create two new columns where the exercise levels will be mapped into 1, 2 and 3 and the breakfast responses into 0 (N) and 1 (Y).

We need to ensure that our categorical variables are correctly encoded. This will make our analysis robust and our interpretations easier.

```
weight['c_exercise'] = weight['exercise']
    .map({'A': 0, 'B': 1, 'C': 2})
weight['c_breakfast'] = weight['breakfast']
    .map({'Y': 1, 'N': 0})
```

We use the map method to transform our features.

Pandas is also able to ensure that columns are treated as categorical with the use of the `astype` method as follows:

```
weight['gender'] = weight['gender']
    .astype('category')
weight['c_breakfast'] = weight['c_breakfast']
    .astype('category')
weight['c_exercise'] = weight['c_exercise']
    .astype('category')
```

We ensure categorical variables are correctly treated with the `astype('category')` method.

We saw in [Section 2.5](#) that the `describe` method can provide us with some useful descriptive statistics. However, out of the box it can only be used with continuous variables:

```
> weight.describe()

           weight
count  108.000000
mean    72.262037
std     13.145986
min     49.900000
25%     61.200000
50%     70.550000
75%     81.600000
max     119.700000
```

The `describe()` method works well with continuous variables.

We can pass an argument to the method to specify the type of data that we would like to include; in this case let us ask pandas to provide information about the categorical variable with `include='category'`:

```
> weight.describe(include='category')
```

	gender	c_exercise	c_breakfast
count	108	108	108
unique	2	3	2
top	2	0	1
freq	62	57	95

We use the `include='category'` parameter to describe categorical features.

As we can see, it provides us the number of unique categories as well as the top category and its frequency. In this case the breakfast feature for example has two unique values, and the Y value has 95 instances.

We can also request a value count and do that in groups. In this case we use the `value_counts` method:

```
> weight[['c_breakfast', 'c_exercise']]
.value_counts()
```

c_breakfast	c_exercise	
1	0	51
	1	36
	2	8
0	0	6
	1	5
	2	2

The `value_counts` method can be used to look at the frequency in our categorical features.

We have 51 participants who have breakfast and always exercise, 36 who have breakfast and sometimes exercise, 6 who do not have breakfast and always exercise, whereas 2 participants never exercise and have no breakfast, and so on.

It is time to run our regression analysis. We will do this here using statsmodels and will create a model where we assume that the breakfast eating habits of the participants is a predictor of the weight such that

$$\text{weight} = \beta_0 + \beta_1 \text{breakfast}. \quad (4.41)$$

The breakfast feature has different levels, In this case 0 or 1. When the level is 0, the equation reduces to $\text{weight} = \beta_0$, and when the level is 1 we have that $\text{weight} = \beta_0 + \beta_1$.

In statsmodels we can use the API we employed in [Section 4.4](#) to use formula notation in our linear regression as follows:

```
model1 = smf.ols(formula= 'weight ~ c_breakfast',
                  data=weight).fit()
print(model1.summary())
```

Part of the summary printed out with the last command above includes the following:

```
OLS Regression Results
=====
Dep. Variable:                weight
No. Observations:              108
R-squared:                     0.006

                                coef    std err
-----
Intercept                    74.9077     3.653
c_breakfast[T.1]             -3.0077     3.895
```

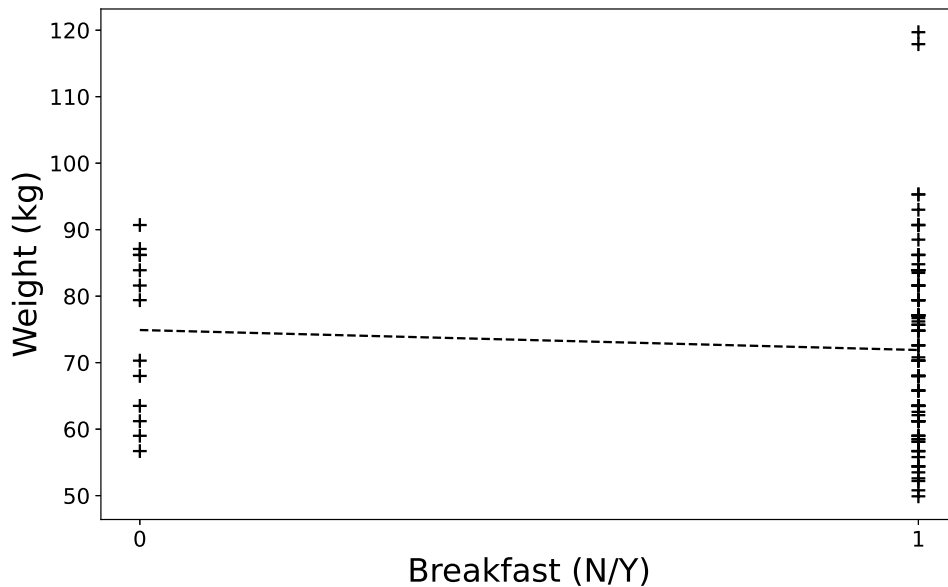
Note that in this case the breakfast variable has different levels, in this case 0 or 1.

The use of the regression model implementation is the same as shown in [Section 4.4](#).

The summary method works in the same way too.

Let us concentrate on the parameters shown above. To understand what these mean, we need to refer to our proposed model given in Equation (4.41) and notice that the Intercept corresponds to our β_0 coefficient and `c_breakfast[T.1]` is β_1 . As discussed above, since our breakfast feature is categorical, it can only take certain values. If the participants do not have breakfast, this means that `breakfast=0` and our regression equation simplifies into `weight=74.9077`, whereas if they do eat breakfast (`breakfast=1`) our equation is given by `weight= 74.9077 + (1)(-3.0077) = 71.9`.

Our coefficients are obtained directly from our analysis. It is their interpretation that may need a bit of consideration.



We can plot the observation and the predicted values as shown in [Figure 4.10](#). Although breakfast only has two

Figure 4.10: Linear regression using a categorical feature. The interpretation of the regression line is different from those obtained with continuous features.

values, we can still draw a regression line showing the relationship it has with weight. We know that the predicted value for participants who do not eat breakfast is 74.9077 and for those that do is 71.95; so the slope of the line is slightly negative, which makes sense since β_1 is negative in this example.

We can compare these predicted values to the mean weight scores for the participants that do and do not eat breakfast. We will do this by grouping the data:

```
> weight_group = weight.groupby('breakfast')
> weight_group.weight.agg(['mean', 'std'])
```

	mean	std
breakfast		
N	74.907692	12.146568
Y	71.900000	13.295936

Note that a regression analysis with a single categorical variable is the same as doing a *t*-test¹² comparing the mean of the dependent variable for each of the levels in the categorical variable. As we can see, the regression equation predicts that for a participant, their weight will be the mean value of the group determined by their breakfast eating habits! Let us take this one step at a time: For participants who do not eat breakfast, their mean weight is the same as the intercept, i.e., 74.9077. The coefficient for breakfast, β_1 , is the amount we need to add to get the mean for participants who do eat breakfast; in other words we need to add -3.0077 to get 71.9, the mean for this group.

We can still draw a line of best fit.

Let us obtain the mean weight for each group and compare with our regression results.

¹² Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

The predicted value is given the mean for each level in our categorical feature.

Note that in the analysis above we have used a feature that has been codified as categorical. The API enables us to use the variables in their raw form, but we need to indicate which variables are categorical explicitly. We do this by wrapping the variable name in a `C()` function as follows:

```
model2 = smf.ols(formula='weight ~ C(breakfast)',
                 data=weight).fit()
```

We can also indicate that a variable is categorical with the `C()`.

Note that the parameters show the level of the categorical variable as follows:

```
> print(model2.params)

Intercept          74.907692
C(breakfast)[T.Y]  -3.007692
```

The summary shows the original level for the categorical variable in question.

In this case `T.Y` indicates the value of the parameter of a `Y` value in `breakfast`.

Let us finish this section by looking at a regression with a feature with three levels, i.e., `exercise`. In this case the feature has levels 0, 1 and 2 corresponding respectively to “always”, “sometimes” and “never”.

We can use features with more levels too.

```
> model3 = smf.ols(formula='weight ~ c_exercise',
                   data=weight).fit()
```

Let us take a look at part of the summary:

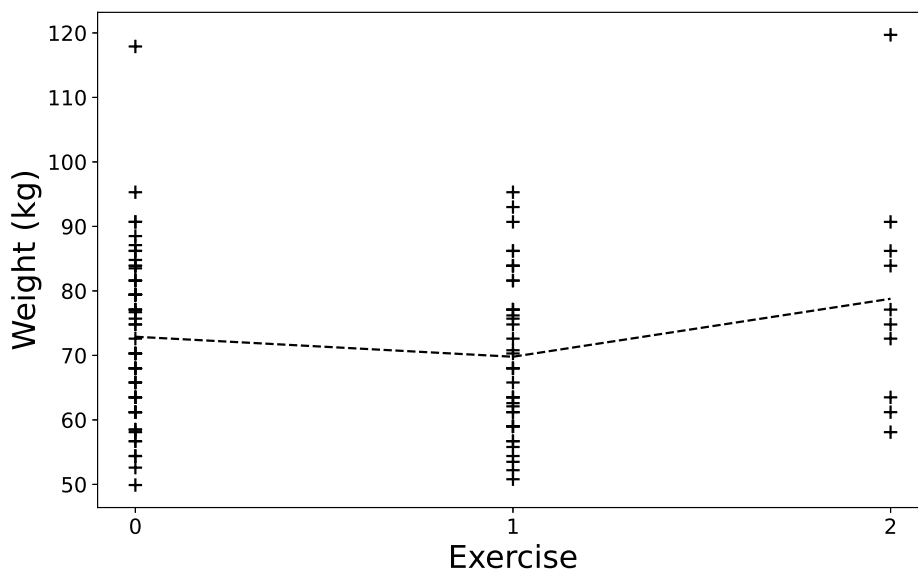


Figure 4.11: Linear regression using a categorical feature with three levels.

```
> print(model3.summary())
```

OLS Regression Results

```
=====
Dep. Variable:          weight
No. Observations:         108
R-squared:                0.038
```

```
Intercept          72.894737
c_exercise[T.1]    -3.102054
c_exercise[T.2]     5.885263
```

The summary shows the coefficient for each level.

The interpretation of the coefficients is every similar to what we did for the binary feature above. Let us take a look at the regression line for this model as shown in [Figure 4.11](#).

Level 0 is the omitted group, and so the intercept is the mean for group 0, i.e., the participants who always exercise. The coefficient for exercise=1 the mean for group 1, minus the mean of the omitted group. This help us determine the predicted value for those participants who sometimes exercise. And the coefficient for exercise=2 is the mean of group 3 minus the mean of the omitted group, i.e., those participants who never exercise.

We can check this by looking at the mean for each group. We can do this with the help of the groupbby method in pandas as follows:

```
> weight.groupby('c_exercise',
    observed=False).weight.mean()
```

```
c_exercise
0    72.894737
1    69.792683
2    78.780000
```

Once again, the regression equation requires us to consider the levels of the categorical value in question.

We can compare our results with the mean values for each of the levels in our categorical feature.

4.11 Summary

IN THIS CHAPTER WE HAVE addressed the topic of regression analysis. It is a natural first step in our data science and analytics journey, as it is one of the most widely used techniques out there. Mastering regression is a must for a Jackalope Data Scientist.

We have seen how regression allows us to describe relationships between input features and the target variable,

bearing in mind that correlation does not imply causation. We have also seen how to interpret our regression coefficients when using categorical variables.

Using the language of linear algebra, we implemented the Ordinary Least Squares (OLS) model to solve the linear regression problem and extended this to a multivariate situation. Furthermore, we have also seen how the models can be used to carry out polynomial regression.

The use of appropriate transformations to our input (and output) data was shown to be beneficial to our modelling task, and the interplay between bias and variance in our models is an important concept to take into account at the modelling stage.

The continuous tension between bias and variance can be used to our advantage in the form of regularisation techniques such as ridge and LASSO, allowing us to fine-tune our models in a very flexible manner. Regression is indeed a useful tool that every Jackalope Data Scientist should have in the toolbox.

5

Jackalopes and Hares: Clustering

HAVE YOU EVER CONSIDERED HOW we can distinguish a rabbit from a stag, or a jackalope from a hare? Since we are able to do this instantaneously, this seems to be a very silly question. A jackalope is effectively the same as a hare except for the antlers. In the case of the rabbit and the stag, if the animal is small and has long ears then it is a rabbit, whereas if it has prominent antlers then it is a stag. We use distinguishing features that allow us to create groups based on similarities and differences.

Of course, if we come across a jackalope we may have to think twice.

We talk about **clustering** when the groups made out of similar data points do not have a pre-defined name or label. When the label does exist, we talk about **classification** and will cover it in [Chapter 6](#). Clustering analysis is an unsupervised machine learning task, whereas classification is a supervised one. In this chapter we will present some important algorithms that enable us to cluster hares and jackalopes, as well as rabbits and stags.

Clustering is an unsupervised learning task, whereas classification is a supervised one.

5.1 *Clustering*

A CLUSTER CAN BE THOUGHT of as a group of similar data points, and therefore the concept of similarity is at the heart of the definition of a cluster. Bigger the similarity among points leads to better clustering and thus to better results. We have mentioned that clustering analysis is an unsupervised learning task. This means that its goal is to provide us with a better understanding of our dataset by dividing the data points into relevant groups. Once the clusters are defined, we can assign them a label and use them as the starting point for classification of unseen data.

Let us consider a couple of examples: Imagine an alien life-form arriving to Earth and being presented with the animals listed in the dataset used in [Chapter 4](#). It does not know what a cat, rabbit, horse and roe deer, are and it does not even have a name for these “strange-looking” animals.

The alien is shown examples of various specimens, and starts to make notes: Cats have pointy triangular ears, rabbits have long oval ones; horses have manes, and deer have antlers. In other words, our alien friend is looking at the similarities (and differences) among the specimens presented for examination. This enables it to group the animals even if it does not know what humans call them. With the help of its clusters, our friendly alien can create its own labels for the groupings. The next time the alien sees a small mammal with a round head, whiskers and triangular ears, it will be able to classify this animal with other cats.

A cluster is a group of similar data points, and the concept of similarity is therefore important.

Imagine an alien life-form that does not know the animals on Earth.

Our alien friend when presented with a variety of animals can cluster them by their similarities, even if it does not know the names we give to these animals.

The application of clustering in the simple example above shows how it helps us enhance our knowledge of the dataset we are working with. Clustering provides us with a layer of abstraction from individual data points to collections of them that share similar characteristics. It is important to clarify that the enhancement is made by extracting information from the inherent structure of the data itself, rather than imposing an arbitrary external one.

Clustering enables us to enhance our knowledge of our datasets.

In that sense, a cluster can be conceived to be a *potential* class, and the solution to a clustering problem is determining these classes, as per the data at hand. Clustering is therefore a data exploration technique that contributes to our familiarity with the dataset. Once we have created suitable clusters and provided them with a label, these clusters can be the starting point to use supervised machine learning techniques. This is true not only about clustering, but also about other unsupervised learning algorithms we will encounter in the rest of the book.

A cluster is a potential class.

Clustering is a data exploration technique.

5.2 *Clustering with k-means*

ONE OF THE SIMPLEST ALGORITHMS that one can implement to solve a clustering problem is the so-called *k-means* algorithm¹. Its goal is to partition an N -dimensional dataset into k different sets, whose number is fixed at the start of the process. The algorithm performs a complete clustering of the dataset; in other words, each data point considered will belong to exactly one of the k clusters.

¹ MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press.

The k -means procedure is said to be a *greedy* algorithm, as it employs a heuristic based on local optimal choices, and therefore the solution found depends on the initial conditions given. The most important part of the process is determining the partitions that form the k sets. Each partition is defined with reference to a centroid, which is the centre of mass, inertia, or gravity of a body or system of bodies. Our clustering algorithm requires us to define k centroids, and assigning each data point to the cluster with the nearest centroid. The centroid is then updated by taking the mean of the data points in the cluster.

A heuristic helps us solve a problem in a quick fashion by finding an approximate solution.

k -means requires the definition of k partitions to which data instances are assigned.

From the brief description above, it is clear that the data required needs to have features in a vector-like form. We also have to bear in mind that the process is iterative in nature. Two other important things to consider are the following:

- The partitions are not scale-invariant, and therefore the same dataset may lead to very different results depending on the scale and units used. In [Figure 5.1](#) we show two representations of the same dataset using two scales
- The initial k centroids are set at the beginning of the process, and different locations may lead to different results

The partitions are not scale-invariant.

Different k centroids lead to different results.

Remember that the number of clusters is set at the beginning of the modelling process. The general idea behind k -means can be summarised in the following four steps:

The value k is an input to the algorithm.

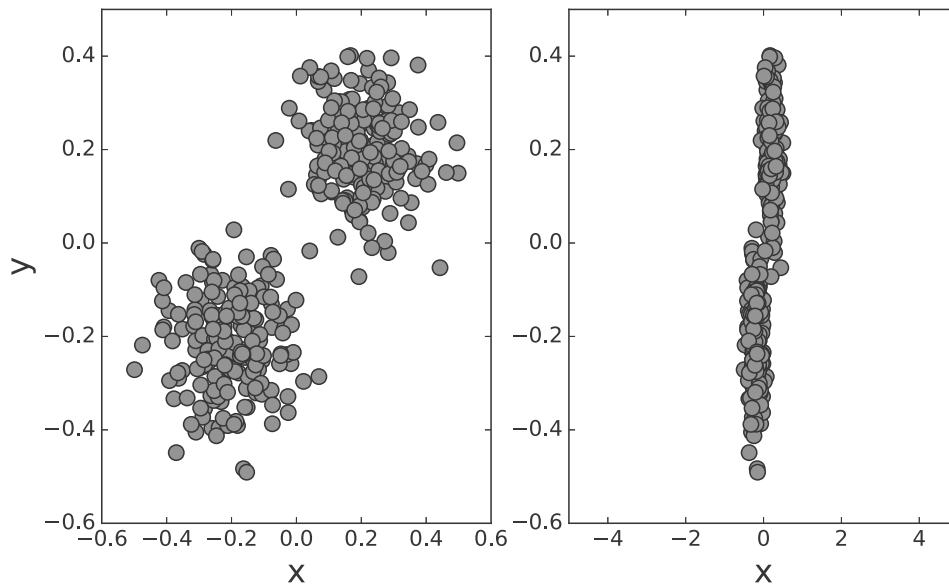


Figure 5.1: The plots show the exact same dataset but in different scales. The panel on the left shows two potential clusters, whereas in the panel on the right the data may be grouped into one.

1. Choose the location of the initial k centroids
2. For each data point, find the distance to each of the k centroids and assign the point to the nearest one
3. Once all data points have been assigned to a cluster, recalculate the centroid positions
4. Repeat steps 2 and 3 until convergence is met

Step 1 above requires us to choose the initial centroids, and there are several possibilities for this. One can choose the positions at random. We can also start with a global centroid and choose points at a maximum distance from it. A good alternative is using multiple random initial conditions on various clustering trials.

The centroids will no longer change position.

Note that a random choice may lead to a divergent behaviour.

A cluster is fundamentally characterised by the similarity among its data points. This similarity implies that data points within a cluster share common attributes and are more closely related to each other than to those in different clusters. In the context of k -means clustering, this similarity is represented by the proximity of a data point to a designated centroid. That closeness is determined by the distance measure or similarity measure chosen for the task. We know the conditions that should be met by a good similarity measure as discussed in [Section 3.9](#). An intuitive way to measure similarity is the Euclidean distance calculated from the N features that describe the data points giving us a clear way to quantify how similar or different the data points are based on their feature values.

Similarity among points is measured by their closeness to the centroids.

As is the case with many algorithms, k -means aims at minimising an objective function. The optimisation of this objective function lets us know how well our clustering task is performing. The recalculation of the centroids at each iteration has to be carried out with this in mind. Using the Euclidean distance $d(x, c_i)$ from point x to the centroid c_i , a typical objective function is a squared error function that is given by:

As with other algorithms, k -means aims at minimising an objective function too.

$$SSE_{k\text{-means}} = \sum_{i=1}^k \sum_j^N d\left(x_j^{(i)}, c_i\right)^2. \quad (5.1)$$

By using the Euclidean distance, k -means effectively partitions the data into clusters where each point is assigned to the cluster with the nearest centroid, thereby minimising the overall distance within each cluster.

As such, given two clustering results, we will prefer the one with the lower sum of squared errors. This is an indication that the centroids have converged to better locations and therefore to a better local optimum for the objective function above.

5.2.1 Cluster Validation

IT IS IMPORTANT TO NOTE that even in cases where no partition exists, k -means will return a partition of the dataset into k subsets. It is therefore useful to validate the clusters obtained. Cluster validation can be further used to identify clusters that should be split or merged, or to identify individual points with disproportionate effect on the overall clustering.

Cluster validation is an important part of the process to determine the effectiveness of the algorithm.

This can be done with the help of two measures: Cohesion and separation. **Cohesion** is a measure of how closely related data points within a cluster are, and is given by the within-cluster sum of squared errors (SSE):

Cohesion tells us how closely related the data points are in a given cluster.

$$C(c_i) = \sum_{x \in c_i} d(x, c_i)^2. \quad (5.2)$$

Separation is a measure of how well clusters are segregated from each other:

Separation indicates if clusters are self-contained.

$$S(c_i, c_j) = d(c_i, c_j)^2. \quad (5.3)$$

Figure 5.2 shows a diagrammatic representation of cluster cohesion and separation. The overall cohesion and separation measures are given by the sum over clusters; in

the case of separation, it is not unusual to weight each of the terms in the sum.

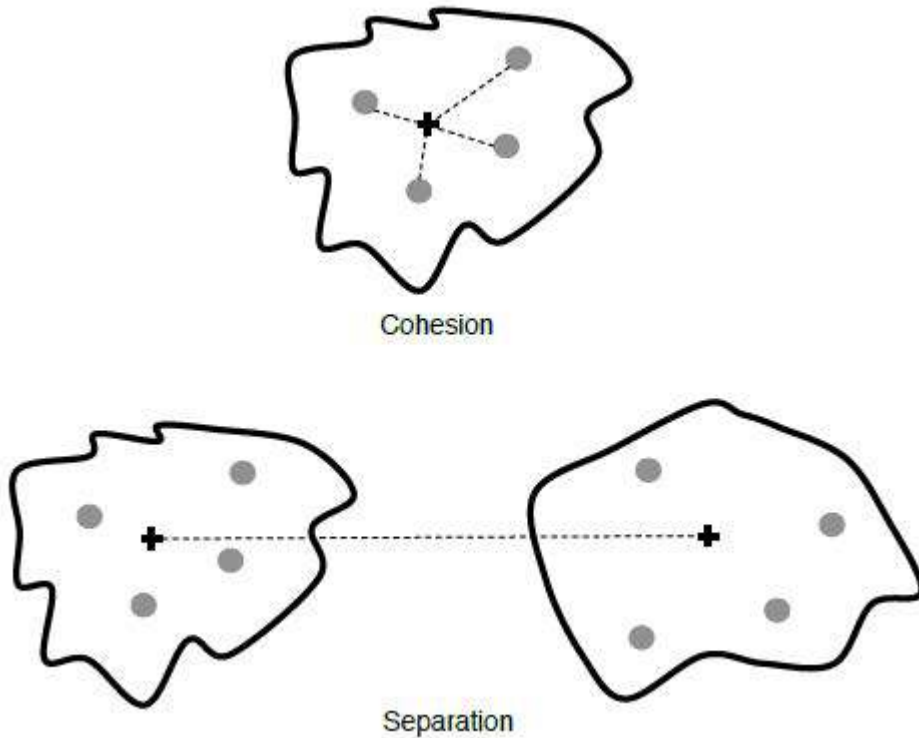


Figure 5.2: Diagrammatic representation of cluster cohesion and separation.

We can use these two definitions to provide us with an overall measure of clustering validity $V_{overall}$ by taking a weighted sum over clusters such that

$$V_{overall} = \sum_{i=1}^k w_i V(C_i), \quad (5.4)$$

The overall clustering validity measure combines both cohesion and separation.

where V can be cohesion, separation or a combination of them.

An alternative measure of validity that provides us with a combination of the ideas behind cohesion and separation in a single coefficient is given by the **silhouette**² score.

For a point x_i the silhouette is defined by the average in-cluster distance to x_i , denoted by a_i , and the average between-cluster distance to x_i , denoted by b_{ij} . Out of these measures it is possible to obtain $b_i = \min_j(b_{ij})$:

$$s(x_i) = \frac{b_i - a_i}{\max(a_i, b_i)}, \quad (5.5)$$

and it has values between -1 and 1 . The value of a_i tells us how dissimilar x_i is in the cluster, and thus we would prefer small values for this quantity. Having a large b_i implies that x_i has been badly matched to a cluster nearby. We are interested in having high separation and low cohesion, and this situation corresponds to values close to 1 for the silhouette coefficient. The average silhouette over the entire dataset tells us how well the clustering algorithm has performed and can be used to determine the best number of clusters for the dataset at hand.

All in all, k -means is pretty efficient both in time and complexity; however, it does not perform very well with non-convex clusters, or with data having varying shapes and densities. One possible way to deal with some of these issues is by increasing the value of k , and later recombining the sub-clusters obtained. Also, remember that k -means requires a carefully chosen distance measure that captures the properties of the dataset.

² Rousseeuw, P. J. (1987). Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Comp. and App. Mathematics*. 20, 53–65.

The silhouette coefficient combines the ideas behind cohesion and separation.

We are interested in having high separation and low cohesion.

Complexity is linear in the number of records.

5.2.2 *k-means in Action*

Let us take a look at an example of clustering with *k-means* using Scikit-learn. We will use data that record the results of chemical analysis of Italian wines grown in the same region and from three different cultivars. The dataset can be found in the UCI Machine Learning Repository under “Wine Dataset”³ and is available at <https://archive.ics.uci.edu/ml/datasets/Wine>.

The 13 attributes determined by the chemical analysis are: Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavonoids, Nonflavonoid phenols, Proanthocyanins, Colour intensity, Hue, OD₂₈₀/OD₃₁₅ of diluted wines, and Proline. The data also has information about the cultivar where the wine is from. In this case we will not use this information, as *k-means* is an unsupervised machine learning algorithm.

We have pre-processed the data into a CSV file with column names, and we can now use pandas to read the file:

```
wine = pd.read_csv(u'./Data/wine.csv')
```

We can see the names of the columns in our dataset:

```
> wine.columns

Index([u'Cultivar', u'Alcohol', u'Malic_Acid',
       u'Ash', u'Ash_Alcalinity', u'Magnesium',
       u'Total_Phenols', u'Flavonoids',
```

³ Lichman, M. (2013a). UCI Machine Learning Repository, Wine Data. <https://archive.ics.uci.edu/ml/datasets/Wine>. University of California, Irvine, School of Information and Computer Sciences.

We have pre-processed the data to add appropriate column names.

These are the features included in the Wine dataset.

```
u'NonFlavonoid_Phenols',
u'Proanthocyanins', u'Colour_Intensity',
u'Hue', u'0D280_0D315_DilutedWines',
u'Proline'],
dtype='object')
```

In order to simplify the example, we are going to concentrate on a couple of features out of the 13 mentioned above. In this situation, we will use the Alcohol and Colour Intensity features, as they are some of the more obvious descriptors that a wine tester would note. We will therefore create an array that contains the values of these columns, and while we are at it we will also extract the cultivar for each of the wines:

Perhaps a wine chemist or a proper sommelier would start with other features.

```
X1=wine[['Alcohol','Colour_Intensity']].values
Y=wine['Cultivar'].values
```

We will use $k = 3$, as we know that there are three different cultivars in the dataset. In general, we would not necessarily have this information to start with. In that case a search of the optimal k can be carried out with the help of the silhouette score. Here we will use the KMeans method in Scikit-learn's cluster:

```
from sklearn import cluster
cls_wine = cluster.KMeans(n_clusters = 3)
cls_wine.fit(X1)
```

The method KMeans takes the parameter `n_clusters` to indicate the number of clusters k .



The shaded areas shown in [Figure 5.3](#) illustrate the clusters obtained for the wines in the dataset using the alcohol and colour intensity features. The data points are shown as filled circles, and their colours correspond to their actual cultivar as per the dataset itself. Finally, we mark the position of the final centroids with the help of filled stars.

We can take a look at the clusters to which each of the data points have been assigned with the aid of the `labels_` method:

```
> print(cls_wine.labels_)

[2 2 2 0 2 2 2 2 2 0 2 2 2 2 0 0 2 2 0 2 2 2]
```

Figure 5.3: *k*-means clustering of the Wine dataset based on Alcohol and Colour Intensity. The shaded areas correspond to the clusters obtained. The stars indicate the position of the final centroids.

We have used $k = 3$ in the example above and only used a couple of the features available.

```

1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
2 2 2 0 0 2 0 2 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1
1 1 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1
1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 2 2 0 1 2
2 2 2 2 2 2 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 0 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0]

```

The predicted labels are obtained with the `.labels_` method.

The values correspond to the cluster (0, 1, or 2) in which the records have been placed. The algorithm does not know anything about the cultivars, so these numbers make no reference to the values that appear in the dataset.

The output of fitting `KMeans` also provides information about the location of the final centroids, and it is saved in `cluster_centers_`:

```

> print(cls_wine.cluster_centers_)

[[ 13.38472222   8.74611108]
 [ 12.25353846   2.854      ]
 [ 13.45168831   5.19441558]]

```

The coordinates of the final centroids are obtained with the method `.cluster_centers_`.

Finally, we can see the silhouette score obtained with the use of the features and value of k chosen:

```

> from sklearn.metrics import silhouette_score
> print(silhouette_score(X1, cls_wine.labels_))

0.509726787258

```

The silhouette score is calculated with `silhouette_score()`.

Remember that we only used two features. Perhaps the score can be improved either by using more relevant features or, in the case where we do not know k , by varying the number of clusters. We leave this as an exercise for the Jackalope Data Scientist in you!

Give it a go!

5.3 *Optimising the Number of Clusters: The Elbow Method*

IN THE PREVIOUS SECTION WE have seen how to implement k -means, and we mentioned that one of the parameters that need to be provided at the start is the chosen number of clusters k . In many cases k is not known and we need to search for the most appropriate number, and one way to do this is the so-called elbow method.

In many cases the number of clusters is not known.

The elbow method is a heuristic approach commonly employed to determine the optimal number of centroids (k) to use in a k -means clustering algorithm. This method involves systematically evaluating different values of k , typically ranging from 1 to a chosen upper limit n (where n is a hyperparameter set based on specific requirements). In principle the maximum of n is equal to the number of datapoints in the dataset. In that way, each data point will be its own cluster. That of course will not be very informative.

We need to evaluate different values of k systematically.

Our search for the optimal k requires us to calculate the within-cluster sum of squares (WCSS) for each value of k . This is a measure of the total variance within each cluster.

By plotting the WCSS against the number of clusters, the method seeks to identify a point where the rate of decrease sharply slows down, resembling an “elbow” and hence the name. This “elbow point” is considered to be the most appropriate number of clusters, as it balances the trade-off between having too many clusters (which can lead to overfitting) and too few (which can lead to underfitting).

Sometimes also referred to as a “hockey stick”.

Let us re-do our wine analysis from the previous section, but this time let us search for the value of k . In the Scikit-learn implementation for k -means, the model has an attribute called `inertia_` which is the sum of squared distances of samples to their closest cluster centre, weighted by the sample weights if provided and thus is our within-cluster sum of squares (WCSS).

The `inertia_` attribute captures the sum of squared distances.

We will store the inertia values in a list called `wcss` and will fit a k -means model for our data using different values, in this case between 1 and 7. Each model is initialised with a random state as follows:

```
wcss = []
for k in range(1, 8):
    clustering = cluster.KMeans(n_clusters=k,
                                random_state=10)
    clustering.fit(X1)
    wcss.append(clustering.inertia_)
```

We are fitting several models, one for each value of k .

The WCSS measures the total variance within each cluster, providing an indication of how tightly grouped the data points are within each cluster. As the number of clusters

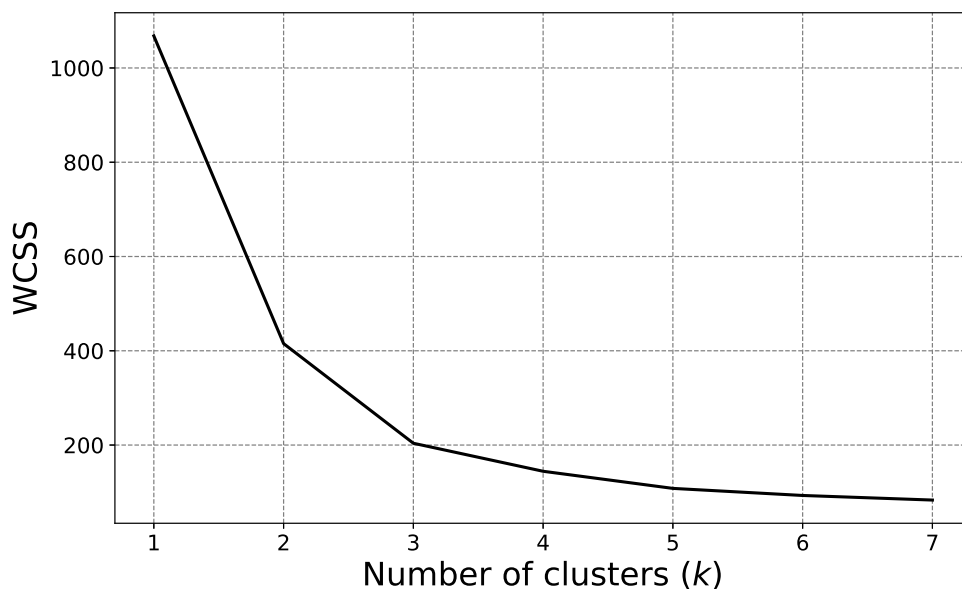


Figure 5.4: k -means clustering with the elbow method. The plot enables us to find a suitable number of clusters in our data.

increases, the WCSS decreases, indicating better clustering performance. However, the rate of decrease will start to slow down as more clusters are added. The optimal number of clusters is identified at the point where the elbow or bend in the plot occurs. The result of our analysis is shown in [Figure 5.4](#), and we can see that $k = 3$ is where the elbow occurs.

The inflection point, or elbow point, indicates a significant reduction in WCSS with the addition of a new cluster, but after this point, the marginal gain of adding more clusters diminishes. This “elbow point” strikes a balance between having too few clusters, which can lead to underfitting, and having too many clusters, which can lead to overfitting.

The optimal number of clusters is identified at the “elbow” or bend in a scree plot.

By identifying this elbow, we can make a more informed decision about the most appropriate number of clusters for our data analysis.

5.4 *Density's Delight: DBSCAN Clustering*

IN THE PREVIOUS SECTIONS WE delved into the realms of k -means clustering and explored how the elbow method can help us partition data into distinct groups based on centroids and optimal cluster counts. While k -means offers simplicity and efficiency, it often struggles with irregular cluster shapes and noise. In this section we aim to go beyond these limitations.

Welcome to the world of Density-Based Spatial Clustering of Applications with Noise, or as it's more commonly known, DBSCAN. Here we will see how DBSCAN, with its unique approach to density and neighbourhood concepts, excels in identifying clusters of arbitrary shapes and effectively handling outliers. Let us get started with the intricacies of density-based clustering and discover a robust alternative to clustering.

DBSCAN was introduced⁴ in 1996 as a way to identify clusters in large spatial datasets and handling noise (outliers). In density-based clustering, clusters are defined as regions where data points are densely packed compared to the rest of the dataset. Objects in less dense areas, which typically separate these clusters, are often treated as noise and border points. DBSCAN relies on the concept of

Sometimes we need to find clusters with irregular shapes and with noise.

DBSCAN stands for Density-Based Spatial Clustering Applications with Noise.

⁴ Ester, M., H.-P. Kriegel, J. Sander, and X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pp. 226–231. AAAI Press.

“density-reachability” which aims to connect data points based on distance thresholds and a density criterion. This requires a minimum number of points within a given vicinity. As a result, clusters obtained with DBSCAN can have arbitrary shapes, comprising all density-connected points and those within their vicinity.

Density-reachability aims to connect data points based on distance and density.

A significant advantage of DBSCAN is its relatively low computational complexity, as it requires a linear number of range queries. Moreover, it consistently produces the same results for core and noise points, though border points may vary slightly. This deterministic nature eliminates the need for multiple runs. Let us take a look at some concepts that will make it easier to understand the algorithm.

The algorithm is computationally efficient.

Let us consider a dataset that needs clustering. In DBSCAN we use a parameter ε that specifies the radius of a neighbourhood around each point. The datapoints are then classified as follows:

The parameter ε is an important component to define clusters.

- **Minimum Points** (*minPts*): The minimum number of points required to form a dense region (cluster)
- **Core Point**: A point that has at least *minPts* within its ε -neighbourhood
- **Border Point**: A point that has fewer than *minPts* within its ε -neighbourhood but is in the neighbourhood of a core point
- **Noise Point**: A point that is neither a core point nor a border point. It is considered an outlier

Some important definitions in DBSCAN.

A point p_i is said to be *directly reachable* from a core point p_c if they are located within ϵ from each other. A point p_i is said to be *reachable* from a core point p_c if there is a path $p_c, \dots, p_i$ where each p_{c+1} directly reachable from p_c . This means that all the intermediate points in the sequence must be core points. Finally, any non-reachable points from any other point are outliers or noise points.

As mentioned before, reachability is a core part of the algorithm.

We define a cluster in DBSCAN by all points (both core and non-core) that are reachable from a core point p_c . This implies that each cluster must contain at least one core point. Non-core points can be part of a cluster but are considered its “edge” since they cannot reach other points. Reachability is not a symmetric relation; core points can reach non-core points, but non-core points cannot reach others. More formally, clusters in DBSCAN are defined by the concept of **density-connectedness** which stated that two points p_i and p_j are density-connected if there is a point q such that both p_i and p_j are reachable from q . In this way, density-connectedness is symmetric.

Each cluster in DBSCAN must contain at least one core point.

We are now ready to look at the steps in the DBSCAN algorithm:

The steps in the DBSCAN algorithm.

1. Neighbourhood Query: For each point in the dataset, find all points within a distance ϵ (ϵ -neighbourhood)
2. Identify Core Points: Mark all points as core points if they have at least *minPts* within their ϵ -neighbourhood
3. Form Clusters: Starting from an unvisited core point, retrieve all points in its ϵ -neighbourhood. If a point in this neighbourhood is a core point, add its neighbours to

the cluster. This process continues iteratively to form a cluster

4. Label Border Points: Points that are not core points, but that fall within the ε -neighbourhood of a core point, are labeled as border points and added to the cluster
5. Identify Noise: Points that do not belong to any cluster (i.e., are not reachable from any core points) are labeled as noise

Noise points are considered outliers.

To determine the minimum number of points, *minPts*, for DBSCAN clustering, a common heuristic to use is that the value should be at least equal to the number of dimensions in the dataset plus one. This can be expressed as:

$$\text{minPts} \geq \text{data dimensions} + 1. \quad (5.6)$$

A common heuristic to find *minPts*.

Suppose that our dataset has three dimension (i.e., columns), the minimum number of points should be at least $3 + 1 = 4$. For this example, let us set the minimum points to 6 since:

$$6 (\text{minPts}) \geq 3 (\text{data dimensions}) + 1.$$

To choose the value for ε , we use a method involving finding the closest data points to a given one. Let us take a look.

5.4.1 Finding Friendly Neighbours: Nearest Neighbours

WE ARE GOING TO TAKE a small detour to consider the problem of finding the nearest neighbours to a given data

point in a dataset. This is an important aspects in many applications but should not be confused with the *k*-means algorithm we covered in [Section 5.2](#). In this case we are interested in finding the nearest neighbours, and this can be done for example by brute-force by calculating the pair-wise distances between points in our dataset. This may work well for small data samples, but as the number of points n grows, the approach becomes very inefficient.

Do not confuse the discussion here with the *k*-means algorithm!

Scikit-learn implements an algorithm in `NearestNeighbors` where we have a choice to select the approach for our search with the `algorithm` parameter using any of the following options: `brut` for brute-force, `kd_tree` to use a tree in *k*-dimensions where we use the knowledge of relative distances between points such that if points p and q are far apart, but points q and r are close, then we can infer that p and r are far apart too. There is also `ball_tree` where data is partitioned in hyperspheres, making it more costly than the KD tree but can be beneficial in highly structured data. Finally, there is the `auto` option which lets the implementation decide the best approach for the given data.

Scikit-learn provides us with a way to try different approaches to finding neighbours.

Consider the following four-dimensional points:

Let us look at an example.

$(0, 0, 3, 0), (2, 0, 0, 1), (0, 0, 2, 1), (1, 2, 1, 0), (1, 2, 2, 0), (1, 1, 1, 1)$

and we want to find the closest three neighbours to any one point. We will start by creating a list with our data points:

```
> s = [[0, 0, 3, 0], [2, 0, 0, 1],
        [0, 0, 2, 1], [1, 2, 1, 0],
        [1, 2, 2, 0], [1, 1, 1, 1]]
```

Let us now load the NearestNeighbors using the auto option, and let us fit the model to the data points above:

```
from sklearn.neighbors import NearestNeighbors
kn = NearestNeighbors(n_neighbors=3,
                     algorithm='auto')
kn.fit(s)
```

We are looking for three neighbours using the auto approach.

We can now find the nearest neighbours to a point using the kneighbors method which returns the distances and the indices to the neighbours of each point:

```
> dist, idx = kn.kneighbors(s)
> print(dist)

[[0.          1.41421356  2.44948974]
 [0.          1.73205081  2.64575131]
 [0.          1.41421356  1.73205081]
 [0.          1.          1.41421356]
 [0.          1.          1.73205081]
 [0.          1.41421356  1.73205081]]
```

The kneighbors method lets us find the nearest neighbours of a given point.

As we can see, the first column is all zeroes. This is because it contains the distance from each point to itself, the second column is the closest neighbour and the third is the second closest neighbour. Should we have asked for more than three neighbours, we would have more columns. We are actually interested in the closest neighbour, and we would like to sort the distances in ascending order:

Note that the size array returned will depend on the number of neighbours searched.

```

> import numpy as np
> dist = np.sort(dist[:,1], axis=0)
> print(dist)

[1.          1.          1.41421356
 1.41421356 1.41421356 1.73205081]

```

We are interested in the closest neighbours, and we are ordering the results.

We will use this information in our DBSCAN analysis by looking at k -neighbours such that:

$$k(\text{neighbours}) = \text{minPts} - 1. \quad (5.7)$$

For the example we have been using, we have that $\text{minPts} - 1 = 6 - 1 = 5$. We will then use the elbow method described in [Section 5.3](#) to find an optimal ϵ value for clustering our data. We simply order the distances of our nearest neighbours in ascending order and look for the bending in the plot. We are now ready to take DBSCAN for a spin.

The elbow method will help us hone in on the value of ϵ .

5.4.2 DBSCAN in Action

We are going to have a look at the application of DBSCAN using Scikit-learn. The dataset we will use can be found in the UCI Machine Learning Repository under “Wholesale customers”⁵ and is available at <https://archive.ics.uci.edu/dataset/292/wholesale+customers>. It contains information about clients of a wholesale distributor including the annual spending in monetary units (m.u.) on diverse product categories.

⁵ Cardoso, M. (2013). UCI Machine Learning Repository, Wholesale Customers. <https://archive.ics.uci.edu/dataset/292/wholesale+customers>. University of California, Irvine, School of Information and Computer Sciences.

The dataset contains eight attributes including product categories:

- Fresh: Annual spending (m.u.) on fresh products
- Milk: Annual spending (m.u.) on milk products
- Grocery: Annual spending (m.u.) on grocery products
- Frozen: Annual spending (m.u.) on frozen products
- Detergents_Paper: Annual spending (m.u.) on detergents and paper products
- Delicatessen: Annual spending (m.u.) on delicatessen products
- Channel: Customers' channel
- Region: Customers' Region

A description of the features in our dataset.

In this example we have categorical variables given by channel and region and will instead concentrate on three variables: Grocery, Detergents_Paper, and Milk. Let us read the data into a pandas dataframe and select the columns of interest:

```
customers = pd.read_csv(u'wholesale_customers.csv')
cols = ['Grocery', 'Detergents_Paper', 'Milk']
df = customers[cols]
```

We will concentrate on a subset of the features in our dataset.

We are going to employ some scaling to our data similar to what we did in [Section 4.6](#) with z-score scaling:

```

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
scaled_data = ss.fit_transform(df)
scaled_customer_data = pd.DataFrame(columns=cols,
                                     data=scaled_data)

```

Remember that clustering is susceptible to scaling.

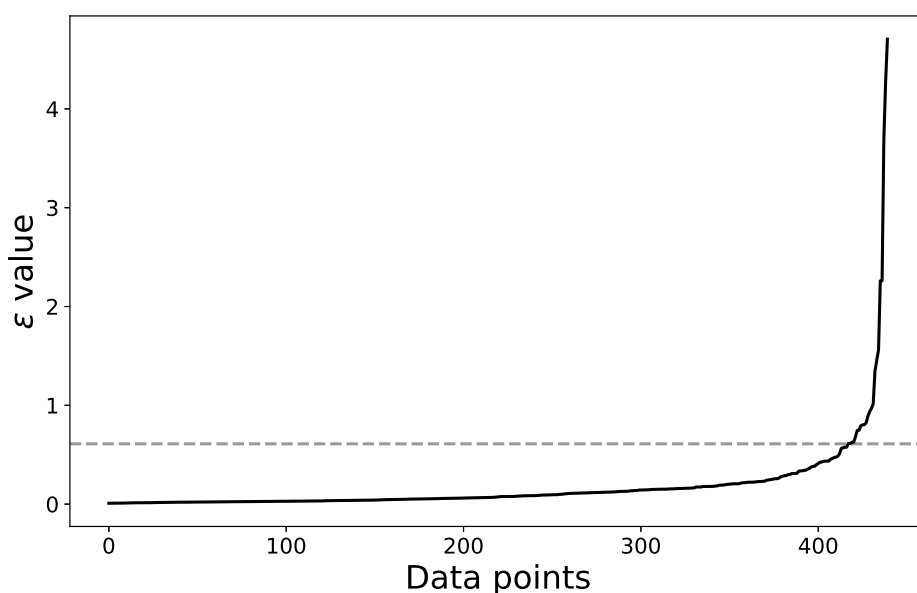


Figure 5.5: Using the elbow method to determine the optimal value of the ϵ parameter to be used in our DBSCAN analysis.

As discussed in [Section 5.4](#) we need to set a value for *minPts*, and given that the number of features is 3 we need a value greater than $3 + 1$. Let us settle for *minPts* = 5. Let us now use the elbow method with the help of the nearest neighbours algorithm described in the previous subsection:

```
from sklearn.neighbors import NearestNeighbors
nn = NearestNeighbors(n_neighbors=5)
nbrs = nn.fit(scaled_customer_data)
dist, inx = nbrs.kneighbors(scaled_customer_data)
dist = np.sort(dist[:,1], axis=0)
```

We are looking for five closest neighbours in this case.

In [Figure 5.5](#) we can see a plot of the distances obtained in the code above, showing a dotted line where the chosen $\epsilon = 0.61$ value is located given the “elbow” in the plot.

We obtain a value of ϵ with the elbow method.

Let us now use the Scikit-learn implementation of DBSCAN on our data with $\epsilon = 0.61$ and *minPts* = 5:

```
from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=0.61, min_samples=5)
dbs.fit(scaled_customer_data)
```

We can finally train our DBSCAN algorithm.

The model has a method called `labels_` that provides the cluster labels for each point in the dataset used to train the model. Note that noise points are given the label `-1`.

The cluster membership can be obtained with the `labels_` method.

```
labels = dbs.labels_
```

Let us see how many clusters were found by the algorithm as well as how many noisy points were identified:

```
n_clusters = len(set(labels)) - (1 if
    -1 in labels else 0)
n_noise_ = list(labels).count(-1)
```

We can easily find the number of clusters and noise points.

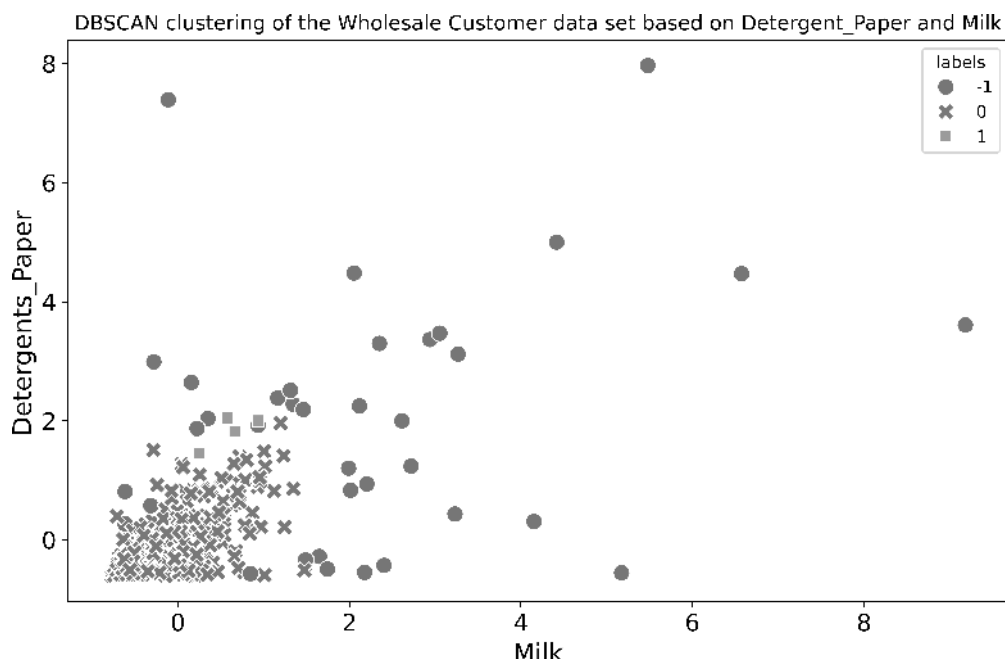


Figure 5.6: DBSCAN clustering of the Wholesale customer dataset based on Detergent_Paper and Milk.

Let us see what the result is:

```
> info = f'We have {n_clusters} clusters and
    {n_noise_} noise points'
> print(info)
```

```
We have 2 clusters and 39 noise points
```

As it was the case for *k*-means, we can use the silhouette score described in [Section 5.2.1](#) to evaluate our algorithm:

```
from sklearn.metrics import silhouette_score
s_score = silhouette_score(scaled_customer_data,
    labels)
```

We can evaluate our model with the help of the silhouette score.

Let us see how our DBSCAN implementation performed::

```
> print(f'Silhouette coefficient: {s_score:.3f}')  
  
Silhouette coefficient: 0.542
```

We can see that DBSCAN was able to capture about 54% of the data.

5.5 *Summary*

CLUSTERING IS A VERY IMPORTANT application in the data science workflow, as it provides ways to explore our data. In this chapter we discussed the difference between clustering and classification. The former is an unsupervised learning task that allows us to establish meaningful groupings within our data. The latter is a supervised learning task that exploits information about the classes contained in our data in order to make predictions about unseen data instances. We will cover classification in the following chapter.

We discussed the k -means algorithm for clustering data, where data instances are assigned to their closest centroid among the k pre-defined ones. We saw how the cohesion, separation and silhouette measures let us validate the clusters we obtain with the algorithm.

Remember that we require prior knowledge of k or alternatively a search for an optimal value. It is worth pointing out that a typical k -means algorithm is started with a random choice for the initial centroids. This means that it

is possible to obtain different results on different runs of the algorithm with the same data. This is an important point, as the clusters we obtain will need to be evaluated not just using cohesion, separation and silhouette scores, but also for fitness to our end goal within the business or research context where we are employing the algorithm.

We also discussed how a heuristic rule can help us determine a value for k in k -means. This is known as the elbow method, and it relies on assessing the results of training various models with different values of k . The optimal value is found at the point of inflection, or elbow, of a scree plot.

In cases where clusters are expected to have different shapes or there is noise (outliers) in the data, we can use DBSCAN to determine our clusters. DBSCAN relies on concepts related to density reachability and connectedness and offers a computationally efficient way to find clusters. We also discussed a way to find nearest neighbours to a given data point based on distance and we saw how this can support us in finding optimal values to be used in our DBSCAN implementation.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

6

Unicorns and Horses: Classification

DISTINGUISHING UNICORNS FROM HORSES OR jackalopes from hares by using their differences and similarities can be very useful. We saw this in the previous chapter with the help of wines and customer sales. The groupings we can create based on the chosen features can tell us something about the members in each. Furthermore, depending on the context in which we create those groups or clusters we can exploit research or business knowledge and assign a label to each of the clusters.

Once we are in possession of labelled data, we can take a step further and use those labels in a supervised learning task, where the labels become our targets. In this chapter we will discuss how **classification** algorithms are used and scored. In particular, we will cover some important algorithms such as k Nearest Neighbours, Logistic Regression and the famous Naïve Bayes classifier, letting us classify unicorns and horses, jackalopes and hares and, why not, rabbits and stags too.

We discussed supervised learning in [Section 3.5](#).

6.1 *Classification*

CLASSIFICATION IS A TASK THAT involves arranging objects systematically into appropriate groups or categories depending on the characteristics that define such groupings. It is important to emphasise that the groups are pre-defined according to established criteria. In our case, the use of classification is to determine the category to which an unseen observation belongs, depending on the information of a training dataset with appropriate labels. Classification is therefore a supervised learning task.

Classification is a supervised learning task.

In [Section 5.1](#) we encountered our alien friend trying to make sense out of the fluffy earthlings it encountered. Let us see another example: Imagine that we are given a Whizzo Quality Assortment chocolate box. We have never tried this brand, but we know that in a box of assorted chocolates there are some delicious ones and others that are not so good. We try a few of the sweets and notice that among the traditional pralines there are some extremely nasty confections: One of them has a raw frog inside, others have cockroaches.

You may want to avoid some of the chocolates in the Whizzo Quality Assortment box.

We notice the shape, aroma and general look of these two nasty sweets and make a note about them for future reference. We even go as far as giving them a name based on the defining attributes we have noted. In other words, we use the features of the confections to cluster them appropriately and provide us with a better understanding of the contents of the box, and even label them. In that way,

In particular, the “crunchy frog” and the “cockroach cluster” may not be very palatable.

the next time we get another one of these chocolate boxes, we know we are better off avoiding the “crunchy frog” and the “cockroach cluster” sweeties, as we are able to classify them as unpalatable confections, and devour the rest of the box.

We can see the clear relationship between classification and clustering as discussed in [Section 5.1](#). Whereas in clustering the aim is to determine the groups from the features in the dataset, classification uses the labelled groups to predict the best category for unseen data. In the context of the examples we presented earlier on, the alien life-form is able to determine whether the next animal it sees is classified to be a cat or a rabbit depending on the pre-determined groups it established at an earlier step. Similarly, we will be able to avoid eating “crunchy frogs” from our box of chocolates without the need of tasting these exotic delicacies again.

Our friendly alien life-form will be able to classify animals after having applied clustering to the specimens it was presented with.

A typical real-life example of a classification problem is determining if a given email that arrives to an email box is *spam* or *ham*. We all know how annoying it is to get unsolicited messages that advertise all sort of things and have no relevance to us. It would be preferable to have an email box without spam, and as such many modern email clients have implementations of classifiers whose task is to predict if an email is spam or not, and filter those messages that are deemed to be spam.

Ham of course being not spam.

And without “Uuuuuuugggh!”

These classifiers may or may not be all that great. However, they certainly try to get better at their task if they encounter

more and more examples of labelled data. In the case of spam, every time a spam message makes it through to our inbox, we may get the option of marking or labelling the offending message as spam. In that way we are adding data to the training of the classifier and helping its improvement.

We help improving a spam classifier by marking misclassified emails as spam, or a legitimate messages as ham.

With that in mind, it becomes obvious that there is a need to quantify how well, or how badly, a classifier is performing. If we carried out the categorisation task at random, sometimes we would classify some observations into the right category and sometimes not. We would prefer a classifier that performs better than this random classifier. At the other end of this spectrum we have a perfect classifier, which categorises all data points into the correct class every time. Since we have labelled data to train a classifier, we can — at the very least — use this information to tell us something about the performance of the classifier. There are a few common ways to present this information, and we discuss them below.

As you can imagine, getting a perfect classifier may be a hard thing to achieve.

6.1.1 *Confusion Matrices*

A VERY CONVENIENT WAY TO evaluate the accuracy of a classifier is the use of a table that summarises the performance of our algorithm against the data provided. Karl Pearson used the name *contingency table*¹.

These days the machine learning community tends to call it a **confusion matrix**, as it lets us determine if the classifier is confusing two classes by assigning observations of one class

¹ Pearson, K (1904). On the theory of contingency and its relation to association and normal correlation. In *Mathematical contributions to the theory of evolution*. London, UK: Dulau and Co.

to the other. One advantage of a confusion matrix is that it can be extended to cases with more than two categories.

In any case, the contingency table or confusion matrix is organised in such a way that its columns are related to the instances in a predicted category, whereas its rows refer to actual classes. To explain the use of the confusion matrix, let us consider a binary classification system implemented for example by a World War II air reconnaissance troop. Their task consists of distinguishing enemy aircraft from flocks of birds. Let us imagine that they have taken 100 measurements and created [Table 6.1](#). The troop is of course interested in detecting correctly when an enemy plane is flying above them.

We can also transpose the table, by the way.

		Predicted Class	
		Enemy Aircraft	Flock of Birds
Actual Class	Enemy Aircraft	20	4
	Flock of Birds	6	70

Table 6.1: Confusion matrix for an elementary binary classification system to distinguish enemy aircraft from flocks of birds.

From [Table 6.1](#), we can see that the troop has correctly predicted 20 enemy planes. These are called the *True Positives*. Similarly, they have correctly predicted 70 flocks, and these are called the *True Negatives*.

Please note that the word “positive” relates to the class we are interested in classifying.

A *False Positive* is a case where we have incorrectly made a prediction for a positive detection. From the table we can see that the troop has predicted six cases as aircraft, but they turned out to be flocks. Finally, a *False Negative*

In that sense, a false positive is a case where we have incorrectly classified an observation as belonging to the class of interest.

is a case where we have incorrectly made a prediction for a negative detection. Again, the table lets us see that the troop has predicted four cases to be birds, but unfortunately they were actual enemy planes. In total, we have 90 correct classifications and 10 instances of misclassification. In [Table 6.2](#) we show a diagram of the location of True Positives, False Negatives, False Positives and True Negatives in a confusion matrix.

		Predicted Class	
		Class 1	Class 2
Actual Class	Class 1	True Positives (<i>TP</i>)	False Negatives (<i>FN</i>)
	Class 2	False Positives (<i>FP</i>)	True Negatives (<i>TN</i>)

Table 6.2: Diagrammatic confusion matrix indicating the location of True Positives, False Negatives, False Positives and True Negatives.

The use of a confusion matrix enables us to see how well the troop has performed the classification of enemy aircraft or flocks of birds. There are also a few quantities that help us with determining the performance of our classifier. Let us define a few of them.

A confusion matrix lets us see how well our classifier performs in the identification of the class of interest.

Recall or True Positive Rate: It is also known as *sensitivity* or *hit rate*. It corresponds to the proportion of positive data points that are correctly classified as positive versus the total number of positive points:

The true positive rate is also known as recall or sensitivity.

$$TPR = \frac{TP}{TP + FN}. \quad (6.1)$$

The higher the True Positive Rate, the fewer negatives will be missed. In our example $TPR = 20/24 = 0.833$.

True Negative Rate or Specificity: It is the counterpart of the True Positive Rate, as it measures the proportion of negatives that have been correctly identified. It is given by:

The true negative rate is also known as specificity.

$$TNR = \frac{TN}{TN + FP}. \quad (6.2)$$

In the example, $TNR = 70/76 = 0.921$.

Fallout or False Positive Rate: It corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points:

The false positive rate is also known as fallout.

$$FPR = \frac{FP}{FP + TN} = 1 - TNR. \quad (6.3)$$

In other words, the higher the false positive rate, the more negative data points we will have misclassified. In our example $FPR = 6/76 = 1 - 0.921 = 0.079$.

Precision or Positive Predictive Value: It is the proportion of positive results that are true positive results. It is given by:

The precision is also known as positive predictive value.

$$PPV = \frac{TP}{TP + FP}. \quad (6.4)$$

In our example $PPV = 20/26 = 0.769$.

Finally, the **Accuracy** is given by the ratio of the points that have been correctly classified and the total number of data points:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} \quad , \quad (6.5)$$

The accuracy of a classifier is the ratio of correctly classified instances and the total number of data points.

and in our example the accuracy is $ACC = 90/100 = 0.9$.

6.1.2 ROC and AUC

THE RECEIVER OPERATOR CHARACTERISTIC (ROC) IS a quantitative analysis technique used in binary classification. It has its origins in the British efforts during World War II to differentiate enemy aircraft from noise, using systems such as “Chain Home”². Different operators had a wide range of skills, and changes in the radar receiver gain levels could influence signal-to-noise ratios. As such, flocks of birds could be mistaken for enemy aircraft. Each receiver operator was said to have their own characteristic and thus the name.

Hence the example in the previous section.

² Galati, G. (2015). *100 Years of Radar*. Springer International Publishing.

As we saw in [Section 6.1.1](#), we can obtain a lot of useful information from true positives, false negatives, etc. Nonetheless, it is sometimes easier to compare one single metric rather than several. That is where ROC³ becomes very handy: It lets us construct a curve in terms of the true positive rate against the false positive rate. Unfortunately ROC curves are suitable for binary classification problems

³ Fawcett, T. (2006). An introduction to ROC analysis. *Pattern. Recogn. Lett.* 27, 861–874.

only. In a ROC curve, the **True Positive Rate** is plotted as a function of the **False Positive Rate** for different cut-off points or *thresholds* for the classifier. Think of these thresholds as settings in the receiver used by the radar operators.

The receiver operator characteristic (ROC) curve shows the true positive rate as a function of the false positive rate.

If our classifier is able to distinguish the two classes without overlap, then the ROC would have a point at the 100% sensitivity and 0% fallout, i.e., the upper left corner of the curve. This means that the closer the ROC curve is to that corner, the better the accuracy of the classifier. A different way of representing this information is in terms of the Area Under the ROC Curve, or AUC. This means that for a perfect classifier the area under the ROC curve will be equal to 1. In that sense the AUC is the one single metric we were looking for.

A perfect classifier would have a 100% sensitivity and 0% fallout.

A perfect classifier would have an Area Under the ROC Curve (AUC) equal to 1.

Setting	Enemy Aircraft Detected (%)	Flocks Detected (%)	Flocks Incorrectly Detected (%)
	Sensitivity	Specificity	Fallout
Off	0	100	0
S ₁	18	96	4
S ₂	34	95	5
S ₃	58	84	16
S ₄	70	78	22
S ₅	88	62	38
S ₆	97	25	75
Full blast	100	0	100

Table 6.3: Readings of the sensitivity, specificity and fallout for a thought experiment in a radar receiver to distinguish enemy aircraft from flocks of birds.

The other side of the coin is a classifier that by chance distinguishes the two classes. It would be no better than tossing a coin to assign observations to each of the two categories. This would be represented by a diagonal line from the origin and going to the $(1,1)$ point in the plot. In that case, the AUC would then be 0.5.

A classifier that is as good as guessing would have a ROC curve described by a diagonal line.

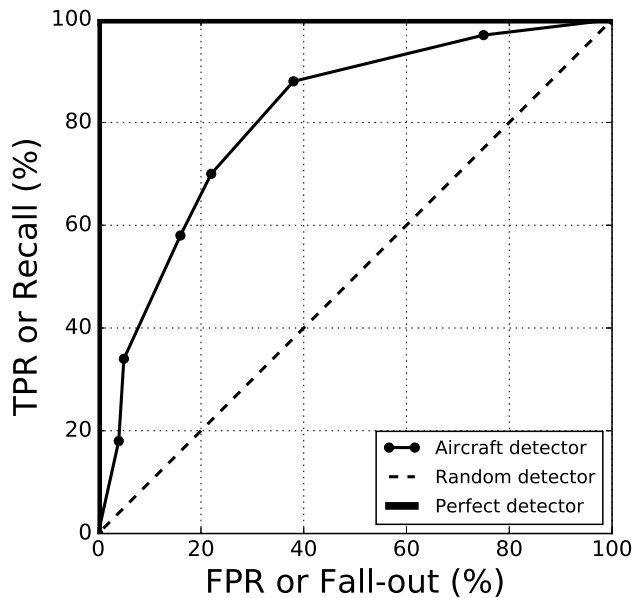


Figure 6.1: ROC for our hypothetical aircraft detector. We contrast this with the result of a random detector given by the dashed line, and a perfect detector shown with the thick solid line.

Let us see an example by considering a receiver that can be placed in eight different settings from off to full detection. The readings from this thought experiment are as shown in Table 6.3. In Figure 6.1 we can see the ROC curve for the data in Table 6.3. We are also showing the diagonal line that represents the 0.5 AUC classifier, as well as the curve given by a perfect classifier whose AUC is equal to 1.

Figure 6.1 shows a set of typical ROCs.

We now know what a perfect classifier looks like in terms of a ROC curve. Similarly, we also know that a diagonal line is equivalent to a random guess. With this in mind, it is clear that we would prefer classifiers that are better than guessing, in other words those whose ROC curves lie above the diagonal.

We prefer classifiers that are better than guessing.

We would also prefer those classifiers whose ROC curves are closer to the curve given by the perfect classifier. If you end up with a ROC curve that lies below the diagonal, your classifier is worse than guessing, and it should be immediately discarded. In the following sections we will discuss some of the most popular classification algorithms and show a few examples.

6.2 *Classification with KNN*

NOW THAT WE HAVE CLARIFIED the goal of classification, we are in a position to discuss our first classifier: The k Nearest Neighbours algorithm⁴ or KNN, for short. We will use the labels provided by the training data to help us decide the class to which a new unseen observation belongs.

⁴ Cover, T. M. (1969, Jan). Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory*, IT-13, 21–27.

Recall that we consider our data points to exist in an N -dimensional space given by the N features in our dataset. In the KNN classifier, similarity is given by the distance between points. We classify new observations taking into account the class of the k nearest labelled data points. This means that we need a distance measure between points, and we can start with the well-known Euclidean distance we discussed in [Section 3.9](#).

In KNN similarity is given by the distance between data instances.

As it was the case in k -means for clustering, the value of k in KNN is a parameter that is given as an input to the algorithm. For a new unseen observation, we measure the distance to the rest of the points in the dataset and pick the k nearest points. We then simply take the most common class among these to be the class of the new observation. In terms of steps, we have the following:

1. Choose a value for k as an input
2. Select the k nearest data points to the new observation
3. Find the most common class among the k points chosen
4. Assign this class to the new observation

Please note that step 3 may pose some issues in cases where we choose an even k . For example, if $k = 4$ we may encounter a situation where two points belong to class A and the other two to class B. In this case it is not possible for us to decide whether to assign our new sample to either A or B.

As you can see, KNN is a very straightforward algorithm and it is very easy to explain. Furthermore, it is able to learn nonlinear boundaries among classes effectively, providing flexibility in handling complex datasets. However, it may be easy to overfit a dataset, and the question about choosing the value of k is an important one. We also need to consider the best distance measure for our dataset. Finally, it is worth mentioning that KNN has low bias and high variance.

The number of neighbours k to take into account for classification is a parameter in this model.

An even number of neighbours may pose a decision problem, so it is better to concentrate on odd values of k .

These issues can be dealt with using cross-validation.

6.2.1 KNN in Action

FOR THIS SECTION WE WILL make use of the Iris dataset we introduced in [Section 3.11](#), containing 150 samples of three species of iris flowers: Setosa, Virginica and Versicolor; with four features: Sepal length, sepal width, petal length and petal width.

Let us start by creating our training and testing datasets. Remember that not only are we interested in the features, but also in the labels. We will load the features in object *X* and the labels in *Y*:

```
X = iris.data
Y = iris.target
```

We introduced the Iris dataset in [Section 3.11](#).

In supervised learning, we are also interested in the labels.

Remember to load the Iris dataset first.

With the aid of `train_test_split` we can create our testing and training datasets. In this case we are using 70% of the data for training and 30% for testing:

```
import sklearn.model_selection as ms

XTrain, XTest, YTrain, YTest =\
    ms.train_test_split(X, Y,\
        test_size= 0.3, random_state=7)
```

We can construct training and testing datasets with `train_test_split`. The parameter `random_state` initialises the pseudo-random number generator used for sampling.

We need to find the appropriate value of *k* for our problem, and we can indeed fit various models for different values of *k*. We can facilitate this search with the help of

The value of *k*, number of neighbours, is a parameter that we need to determine.

GridSearchCV as we did for the hyperparameter λ in [Section 4.9](#).

Let us load the relevant libraries:

```
from sklearn import neighbors

from sklearn.model_selection import GridSearchCV
```

We will search odd values between 1 and 20 and find the best value of k with cross-validation:

```
k_neighbours = list(range(1,21,2))

n_grid = [{'n_neighbors': k_neighbours}]
```

We will be searching the optimal value of k in the odd values between 1 and 20.

We will apply this to the `KNeighborsClassifier()` function in the module `neighbors`:

```
model = neighbors.KNeighborsClassifier()

cv_knn = GridSearchCV(estimator=model,\
    param_grid=n_grid,\
    cv=ms.KFold(n_splits=10))

cv_knn.fit(XTrain, YTrain)
```

We are performing cross-validation with 10 folds in our training data.

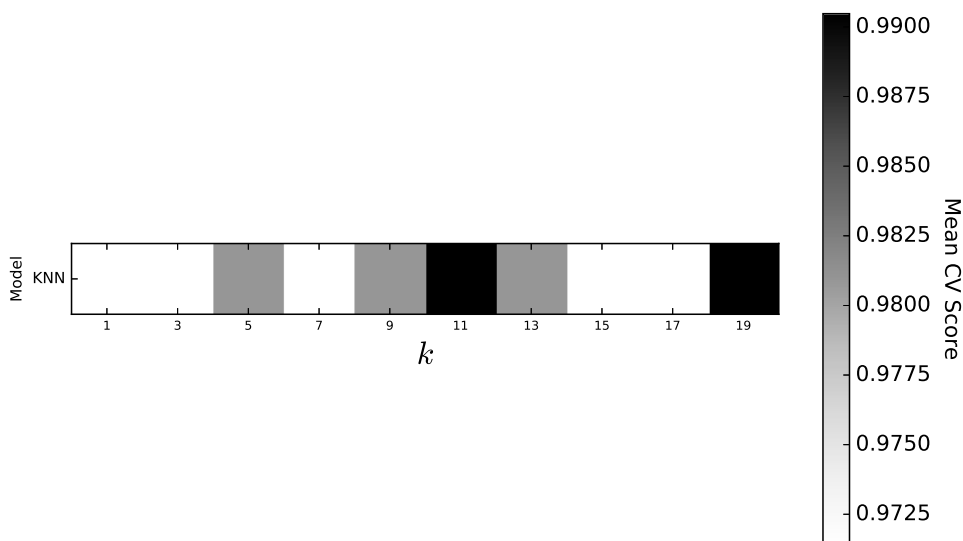
The result of the search can be seen as follows:

```
> best_k = cv_knn.best_params_['n_neighbors']
> print('The best parameter is k={0}'.\
      format(best_k))
```

The best parameter is k=11

In this case we see that the best number of neighbours is 11, and in [Figure 6.2](#) we show a heatmap of the different values of k considered and their corresponding score.

The score evaluated in by the model is the accuracy of the classification.



Finally, let us see how our model performs on the testing dataset using $k = 11$ for the number of neighbours. In this case we will only use two of the features of the dataset for visualisation purposes. We will concentrate on the sepal

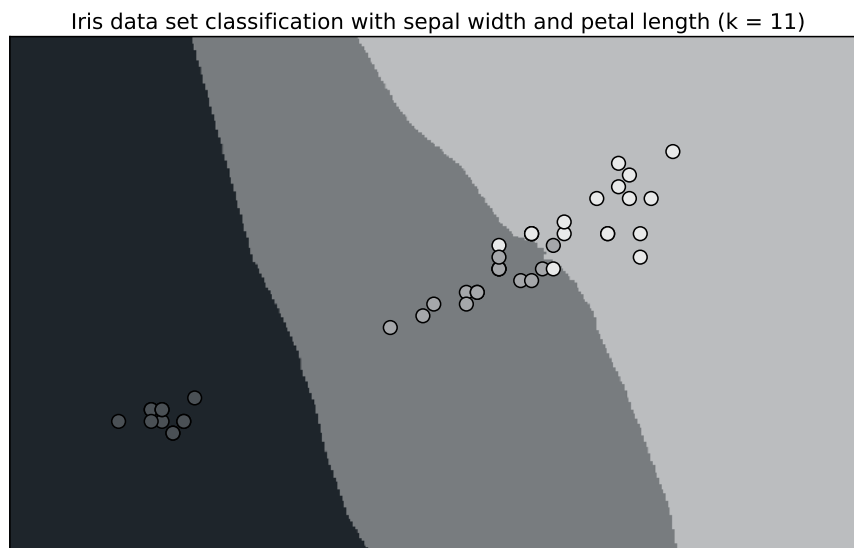
Figure 6.2: Accuracy scores for the KNN classification of the Iris dataset with different values of k . We can see that the best parameter found is 11 neighbours.

width and the petal length:

```
knnclf = neighbors.KNeighborsClassifier\
    (n_neighbors=best_k)

knnclf.fit(XTrain[:, 2:4], YTrain)
```

We will use $k = 11$ to fit a model to our training dataset.



The prediction can be obtained using the `predict` method of our model:

```
y_pred = knnclf.predict(XTest[:, 2:4])
```

In [Figure 6.3](#) we show a contour map of the three Iris classes obtained using KNN based on sepal width and petal length.

Figure 6.3: KNN classification of the Iris dataset based on sepal width and petal length for $k = 11$. The shaded areas correspond to the classification mapping obtained by the algorithm. We can see some misclassifications in the upper-right-hand corner of the plot.

The predictions of our model can be obtained with the `predict` method.

The data points correspond to the flowers in the testing set, coloured according to their class. We can see some instances where the flowers have been misclassified.

Let us construct a confusion matrix for our classifier:

```
> from sklearn.metrics import confusion_matrix

> confusion_matrix(YTest,y_pred)

array([[12,  0,  0],
       [ 0, 14,  2],
       [ 0,  2, 15]])
```

A confusion matrix can be obtained with the `confusion_matrix` function from the module `metrics`.

As we can see, four instances for classes 2 and 3 (two each) have been misclassified. We can obtain a classification report as follows:

```
> from sklearn.metrics \
import classification_report
> print(classification_report(YTest, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	12
1	0.88	0.88	0.88	16
2	0.88	0.88	0.88	17
avg / total	0.91	0.91	0.91	45

The `classification_report` function provides information about the precision and recall of the classification model trained.

The report gives us information about the precision and recall for our classifier as well as the F_1 score which

provides a measure of accuracy based on the precision and recall as follows:

$$F_1 = 2 \frac{(\text{precision})(\text{recall})}{(\text{precision} + \text{recall})}, \quad (6.6)$$

The F_1 score provides a measure of accuracy based on the precision and recall.

and ranges between 1 (best) and 0 (worst).

6.3 *Classification with Logistic Regression*

WE ARE FAMILIAR WITH THE concept of regression as discussed in [Chapter 4](#). We saw that regression is a supervised machine learning task that enables us to obtain predictions for continuous variables. In contrast, logistic regression is used in the prediction of a discrete outcome and therefore best suited for classification purposes. Logistic regression is in effect another generalised linear model that uses the same basic background as linear regression. However, instead of a continuous dependent variable, the model is regressing for the probability of a (binary) categorical outcome. We can then use these probabilities to obtain class labels for our data observations.

Regression is used for continuous variables. For logistic regression, think of categorical variables.

Logistic regression predicts the probability of a categorical outcome.

Let us recall that a linear regression model is the conditional mean of the outcome variable $\mathbf{Y}$ given the value of the covariate $\mathbf{X}$ noted as $E(\mathbf{Y}|\mathbf{X})$ and is given by:

$$E(\mathbf{Y}|\mathbf{X}) = \beta\mathbf{X}, \quad (6.7)$$

For a univariate case $\beta\mathbf{X} = \beta_0 + \beta_1\mathbf{x}$.

where we assume that this conditional mean is a linear function taking values between $-\infty$ and ∞ .

In logistic regression, we are interested in determining the probability that an observation belongs to a category (or not) and therefore the conditional mean of the outcome variable must lie in the interval $[0, 1]$. We need to extend the linear regression model to map the outcome variable into that unit interval. We can do this with the help of the sigmoid function defined as:

$$g(z) = \frac{e^z}{1 + e^z}. \quad (6.8) \quad \text{The sigmoid or logistic function.}$$

We need to map our outcomes to values between 0 and 1.

We show a plot of this function in [Figure 6.4](#) where we can see that the domain of the function is $(-\infty, \infty)$ and its range is $[0, 1]$, as required. This function is also known as the **logistic function**.

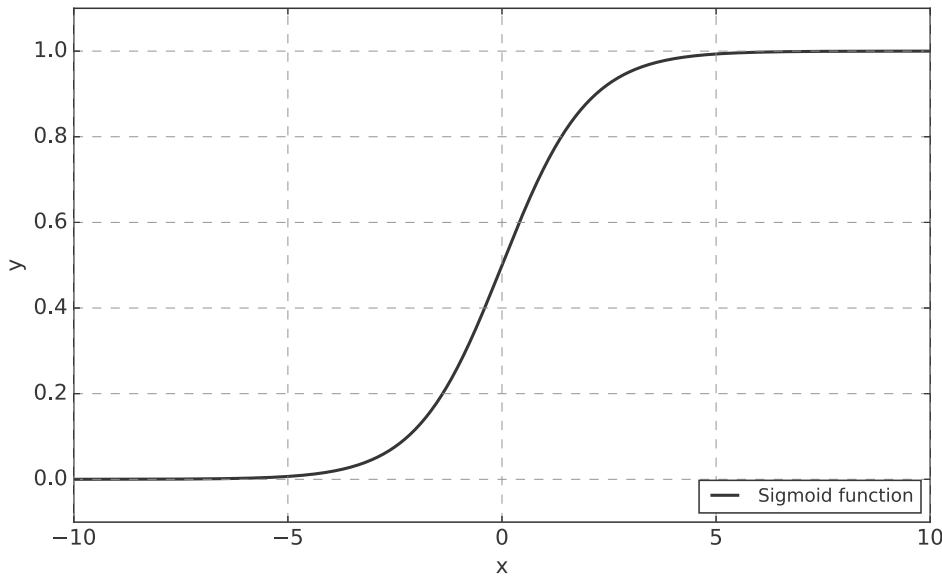


Figure 6.4: Plot of the logistic function $g(z) = \frac{e^z}{1 + e^z}$.

Our transformed regression model (for a univariate case) is therefore given by:

$$E(\mathbf{Y}|\mathbf{X}) = P(\mathbf{X}) = \frac{\exp(\beta\mathbf{x})}{1 + \exp(\beta\mathbf{x})}. \quad (6.9) \quad \text{Our transformed model.}$$

It is also relevant to mention an important transformation related to the logistic function, namely the **logit function** or **log-odds function**. The logit of a probability given by $p \in [0, 1]$ is:

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right). \quad (6.10) \quad \text{The logit or log-odds function.}$$

The interpretation of the quantity above is such that if p represents a probability, the logit of the probability is the logarithm of the odds. Conveniently, the difference between the logits of two probabilities is the logarithm of the odds ratio, which can be expressed in terms of simple addition and subtraction.

The difference between the logits of two probabilities is the logarithm of the odds ratio.

We can show that we can recover our linear model by applying the logit function to Equation (6.9):

$$\begin{aligned} \ln\left(\frac{P(\mathbf{X})}{1-P(\mathbf{X})}\right) &= \ln P(\mathbf{X}) - \ln(1-P(\mathbf{X})) \\ &= \ln\left(\frac{e^{\beta\mathbf{x}}}{1+e^{\beta\mathbf{x}}}\right) - \ln\left(1 - \frac{e^{\beta\mathbf{x}}}{1+e^{\beta\mathbf{x}}}\right) \\ &= \ln(e^{\beta\mathbf{x}}) - \ln(1+e^{\beta\mathbf{x}}) + \ln(1+e^{\beta\mathbf{x}}) \\ &= \beta\mathbf{x}. \end{aligned} \quad (6.11)$$

We recover a linear model when applying the logit function to our transformed model.

The logit function can therefore be a very useful tool in interpreting the results obtained from the logistic regression algorithm.

Another important difference between the linear and the logistic regressions is the error term. In the case of the linear regression, the error term follows an independent normal distribution with zero mean and constant variance.

Normal distributions are also called Gaussian

In logistic regression, however, the outcome variable can take only two values: 0 or 1. This means that instead of following a Gaussian distribution, it follows a Bernoulli one. The Bernoulli distribution corresponds to a random variable that takes the value 1 with probability p and 0 with probability $q = 1 - p$. For more information on these distributions, see Rogel-Salazar (2023)⁵.

Think of a coin-tossing experiment.

Our logistic regression model given by Equation (6.9) will provide us the probability that an observation belongs to a binary class or not. We need a threshold that helps us carry out the classification task, and a typical example is as follows:

$$y_i = \begin{cases} 1, & \text{if } P(\mathbf{X}) \geq 0.5. \\ 0, & \text{otherwise.} \end{cases} \quad (6.12)$$

These are typical thresholds applied alongside a logistic regression.

Since the logistic regression model is a generalised linear model for outcomes in $(0, 1)$, the solution can be implemented using Ordinary Least Squares and still use measures such as R^2 . In cases where the outcome is exactly 0 or 1, other methods, such as maximum likelihood estimation, are needed.

See [Chapter 4](#) for more information.

⁵ Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

Let $\mathbf{x}$ be a set of training data and $\mathbf{y}$ correspond to the labels for the dataset. We are interested in maximising the log-likelihood of the data, and so for i samples we have that the coefficients are given by:

$$\beta \leftarrow \max_{\beta} \sum_i \ln P(\mathbf{X}_i, \beta). \quad (6.13)$$

The objective function for logistic regression.

We can also apply regularisation to the logistic regression model and thus the optimisation problem can be written as:

$$\beta \leftarrow \max_{\beta} \sum_i \ln P(\mathbf{X}_i, \beta) - \lambda \|\beta\|_n, \quad (6.14)$$

The objective function for regularised logistic regression.

where $\|\cdot\|_n$ is the L - n norm. In that way we can achieve the penalisation for high coefficients as discussed for the linear regression in [Section 4.9](#).

6.3.1 Logistic Regression Interpretation

AS WE HAVE MENTIONED ABOVE, the interpretation of the results from logistic regression is better understood in terms of the odds and the odds ratio. We know that the odds of an event with probability p is given by:

$$\text{odds} = \frac{p}{1-p}, \quad (6.15)$$

The odds of an event with probability p .

which is effectively the ratio of the probability that the event will take place (p) to the probability that it will not ($1-p$).

In a univariate case, from Equation (6.11) we have that:

$$\frac{P(\mathbf{x})}{1-P(\mathbf{x})} = \exp(\beta_0 + \beta_1 x). \quad (6.16)$$

The odds for the univariate case.

Remember that in linear regression, we interpret the parameter β_1 as the change in the target variable given a unit change in the covariate x . Let us see how we can use this to interpret our coefficients in the logistic regression. We consider a univariate case and a change of one unit in the variable x . The odds would be given by

$$\text{odds}(x+1) = \exp(\beta_0 + \beta_1(x+1)). \quad (6.17)$$

Let us now take the odds ratio of Equations (6.16) and (6.17):

$$OR = \frac{\exp(\beta_0 + \beta_1 x)}{\exp(\beta_0 + \beta_1(x+1))} = e^{\beta_1}. \quad (6.18)$$

If we now take the logarithm of the odds ratio, we have that it is equal to the coefficient β_1 :

$$\ln(OR) = \beta_1, \quad (6.19)$$

and this means that the coefficients obtained from logistic regression can be interpreted as the change in the logit function for a unit change in the covariate. In other words, the odds ratio of a binary event gives the increase in likelihood of an outcome if the event occurs. Similarly, the odds ratio in logistic regression represents how the odds change with a unit increase in a given feature, holding all other features constant.

For example, imagine that you estimate a logistic regression model with features x_1 and x_2 such that the model can be expressed as $10.0145 + 0.25x_1 + 0.04x_2$. The effect of the odds of a unit increase in x_1 is given by $\exp(0.25)=1.284$, and this

Let us consider a unit change in x .

We then calculate the odds ratio.

The coefficients correspond to the change in the logit function for a unit change in x .

An example of the interpretation of the logistic regression coefficients.

means that the odds increase by about 28%, regardless of the value of x_2 .

Before we move on to see an example of the application of logistic regression, let us address an important point regarding the number of classes in our problem. Our discussion has concentrated on a problem with two classes (A and B, 1 and 0, etc.). Logistic regression can be used in a multiclass setting too, and a typical strategy is the so-called *one-versus-the-rest* strategy. In this case the modelling is done by using one classifier per class so that for each of our classifiers, the chosen class is modelled against the rest of the classes. In other words, if we have a problem with three classes 0, 1 and 2, we will need three classifiers:

1. Classifier one for class 0 versus 1 and 2
2. Classifier two for class 1 versus 0 and 2
3. Classifier three for class 2 versus 0 and 1

It is possible to use logistic regression in cases involving more than two classes.

One-versus-the-rest strategy for three classes.

6.3.2 *Logistic Regression in Action*

FOR OUR EXPLORATION OF LOGISTIC regression, we are going to use a breast cancer dataset that contains real-world cases from a study by at the University of Wisconsin Hospital, which was led by Wolberg and Mangasarian⁶. The dataset includes valuable features that can be utilised for classification tasks, such as distinguishing between malignant and benign tumors; as you can imagine, it has been widely used in machine learning research and applications due to its relevance and accessibility.

⁶ Mangasarian, O. L. and W. H. Wolberg (1990, Sep.). Cancer diagnosis via linear programming. *SIAM News*. 25(5), 1 & 18.

The dataset is hosted in the UCI Machine Learning Repository under the tile “Breast Cancer Wisconsin (Original) Dataset” and is available at

[https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Original\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original))⁷. We are interested in

classifying breast tissue samples into benign or malignant cases. The dataset contains 699 instances with an ID column and ten features as follows:

1. Sample Code Number (ID number)
2. Clump Thickness (1- 10)
3. Uniformity of Cell Size (1- 10)
4. Uniformity of Cell Shape (1- 10)
5. Marginal Adhesion (1- 10)
6. Single Epithelial Cell Size (1- 10)
7. Bare Nuclei (1- 10)
8. Bland Chromatin (1- 10)
9. Normal Nucleoli (1- 10)
10. Mitoses (1- 10)
11. Class (2 for benign, 4 for malignant)

⁷ Lichman, M. (2013b). UCI Machine Learning Repository, Wisconsin Breast Cancer Database. [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Original\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)). University of California, Irvine, School of Information and Computer Sciences.

The features in our dataset.

We assume that the data has been pre-processed into a CSV file with appropriate names for the columns detailed above. We can then load it into Python as follows:


```
bc = pd.read_csv(u'./breast-cancer-wisconsin.csv')
bc = bc.dropna()
```

The dropna method drops instances with missing information.

We have dropped any instances where no information is available, leaving us with 683 instances; we can see how they are distributed with the help of describe:

```
> bc['Class'] = bc['Class'].astype('category')
> bc['Class'].describe()
```

```
count      683
unique        2
top          2
freq        444
Name: Class, dtype: int64
```

We cast the column Class as a categorical variable.

As we can see, the most frequent class is '2' (benign) with 444 instances. Let us prepare our data by separating the labels from the rest of the dataset. We will also drop the ID column:

```
X = bc.drop(['ID', 'Class'], axis=1)
X = X.values
Y_raw = bc['Class'].values
```

We separate the labels from the rest of the dataset by dropping the appropriate column.

It would make our task easier to use '0' and '1' as the labels for our classes instead of the labels used by in the original dataset. This can easily be done with LabelEncoder as follows:

```
from sklearn import preprocessing
label_enc = preprocessing.LabelEncoder()
label_enc.fit(Y_raw)
Y = label_enc.transform(Y_raw)
```

The `LabelEncoder` method lets us reassign the encodings used for our classes.

It is possible to see the classes with `label_enc.classes_` and very importantly we can invert the encoding with `label_enc.inverse_transform()`.

Let us split our data into training and testing as we have been doing so far:

```
import sklearn.model_selection as ms

XTrain, XTest, YTrain, YTest =\
    ms.train_test_split(X, Y,\
        test_size=0.3, random_state=1)
```

We are splitting our data into 70% for training and 30% for testing.

Scikit-learn implements logistic regression in the `linear_model` module and is called `LogisticRegression`. We are going to use regularisation in our model, and we can choose between different types of penalties such as *L1*, *L2* or a combination of them referred to as *elasticnet*. The hyperparameter in this case is implemented as *C*, and it corresponds to the inverse of the regularisation strength. This means that the smaller the value of *C*, the stronger the penalty.

The logistic regression model is implemented in the `linear_model` module. The hyperparameter is denoted by *C*.

Please note that some penalties may not work with some solvers implemented in the Scikit-learn. For example, the

Limited-memory Broyden–Fletcher–Goldfarb–Shanno⁸ algorithm, better known as LBFGS, is the default solver for the model implementation, and it only accepts the L_2 penalty. The default solver prior to version 0.22 of Scikit-learn was liblinear based on the LIBLINEAR Library for Large Linear Classification⁹.

We will use the LIBLINEAR solver and use GridSearchCV to determine the best values for these penalty and the hyperparameter:

```
from sklearn.linear_model \
    import LogisticRegression

pen_val = ['l1', 'l2']
C_val = 2. ** np.arange(-5, 10, step=2)
grid_s = [{'C': C_val, 'penalty': pen_val}]
model = LogisticRegression(solver='liblinear')

from sklearn.model_selection\
    import GridSearchCV

cv_logr = GridSearchCV(estimator=model,\
    param_grid=grid_s,\
    cv=ms.KFold(n_splits=10))
```

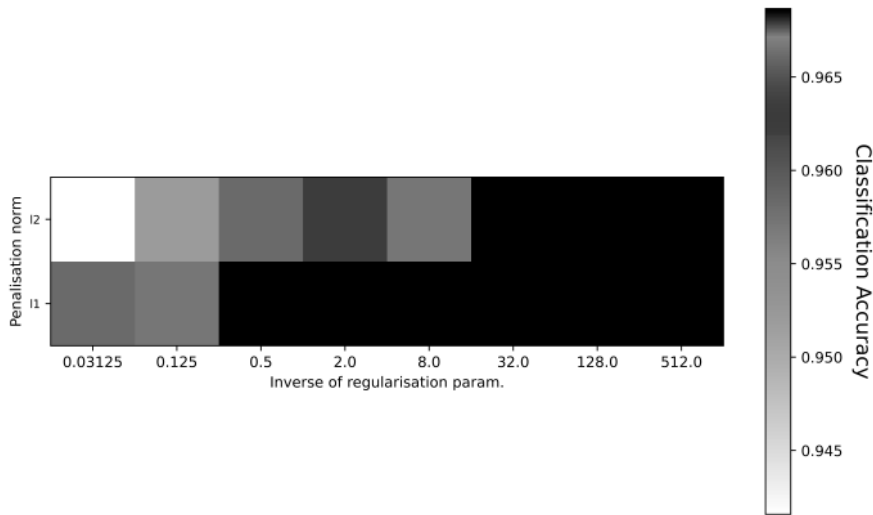
The fitting can now be done as follows:

```
cv_logr.fit(XTrain, YTrain)
best_c = cv_logr.best_params_['C']
best_penalty = cv_logr.best_params_['penalty']
```

⁸ Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*. 45(1), 503–528.

⁹ Fan, R. E., K. W. Chang, C. J. Hsieh, X. R. Wang, and C. Lin (2008). LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*. 9, 1871–1874.

As with other examples we have seen, we can search for optimal values of the hyperparameter with GridSearchCV.



In this case the grid search was carried out with 10 folds, over the hyperparameter values defined, as well as $L1$ and $L2$ penalties, returns the following parameters:

```
> print('The best parameters are:\n
cost={0} and penalty={1}').\n
format(best_c, best_penalty))

The best parameters are: cost=0.5 and penalty=l1
```

Figure 6.5: Heatmap of mean cross-validation scores for the Logistic Regression classification of the Wisconsin Breast Cancer dataset for different values of C with $L1$ and $L2$ penalties.

Figure 6.5 shows a heatmap of the parameter search we have performed. It is now possible for us to use the parameters above to create a classifier for our problem and train it to be used on unseen data. All we have to do is create an instance of the logistic regression model using the parameters obtained:

```
b_clf = LogisticRegression(solver='liblinear',
                           C=best_c, penalty=best_penalty)

b_clf.fit(XTrain, YTrain)
```

We are instantiating a model with the parameters found above.

The classifier can now be used to predict the class in our testing dataset with the predict method. We obtain the probabilities assigned to each instance with the predict_proba method:

```
predict = b_clf.predict(XTest)
y_proba = b_clf.predict_proba(XTest)
```

predict gives us the predicted class, where as predict_proba gives us the probabilities.

Please note the y_proba is an array with two columns, each column indicates the probability that the instance belongs to either of the two classes.

Let us take a look at the classification accuracy score:

```
> print(b_clf.score(XTest, YTest))
0.9560975609756097
```

The accuracy of the model can be seen with the score method.

The coefficients for the model can be obtained with the .coef_ method:

```
> print(b_clf.coef_)
[[ 0.33019135  0.25262708
    0.31634685  0.05621455
   -0.05283308  0.36893348
    0.18713312  0.2545694
    0.13853633]]
```

The coefficients are stored in the coef_ method.

Remember that the odds ratios are obtained by taking the exponential of the coefficients. They tell us how a unit increase or decrease in a variable affects the odds of having a malignant mass:

```
> print(np.exp(b_clf.coef_))

[[1.39123432  1.28740309
   1.37210609  1.05782462
   0.94853832  1.4461914
   1.20578779  1.28990607
   1.14859141]]
```

Taking the exponential of the coefficients lets us interpret the results.

For instance we can expect the odds of having a malignant mass to increase by about 28% if the measure for “Uniformity of Cell Shape” increases by one unit.

Finally, let us take a look at obtaining the ROC and AUC measures. In the case of ROC, we can use `roc_curve` whose inputs are the true labels of the instances and the target scores such as probability estimates of the positive class or confidence values. Let us use the `y_proba` estimates we calculated above:

```
from sklearn.metrics import roc_curve, auc

fpr, tpr, threshold=roc_curve(YTest, y_proba[:,1])

plt.plot(fpr, tpr)
```

The next step is to calculate the ROC and AUC.

and the AUC can be calculated with the true positive and false positive rates obtained above:

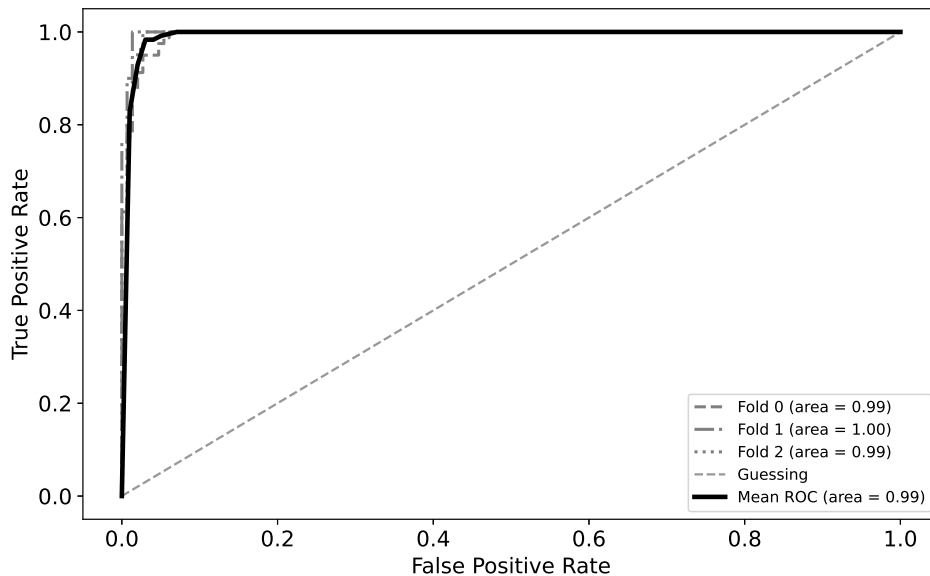


Figure 6.6: ROC curves obtained by cross-validation with $k = 3$ on the Wisconsin Breast Cancer dataset.

```
> print(auc(fpr, tpr))
```

```
0.997519671570
```

In [Figure 6.6](#) we present the ROC curves obtained by applying cross-validation on the breast cancer dataset, using three folds for simplicity in this demonstration. Each ROC curve represents the performance of the classifier on a different fold of the dataset, allowing us to observe variations in its ability to distinguish between classes. Additionally, we calculate and display the mean ROC curve, which is averaged across all the folds to provide an overall measure of classifier performance. For comparison, we

It is possible to obtain various ROC plots with the aid of cross-validation.

include the random classifier's performance, represented by the diagonal line, to highlight the classifier's effectiveness relative to a baseline model. This visual comparison underscores the value of cross-validation in assessing the robustness and reliability of classification models.

6.4 Classification with Naïve Bayes

PROBABILITY IS AT THE CENTRE of many everyday life applications: From weather to sports, passing by finance and science. As a theory, we owe much to the French mathematicians Pierre de Fermat and Blaise Pascal who used to discuss problems related to games of chance with each other¹⁰. As such, in the early 17th century, it was widely believed that it was not possible to calculate — i.e., predict — the outcome of rolling dice for example. We now know that is not really the case.

¹⁰ Devlin, K. (2010). *The Unfinished Game: Pascal, Fermat, and the Seventeenth-Century Letter That Made the World Modern*. Basic ideas. Basic Books.

In the previous section we used logistic regression to estimate the probability that a data instance belongs to a particular class and, based on that, decide on the label that should be assigned to that instance. We assumed that we knew what was meant by probability, in other words, a number between 0 and 1 that indicates how likely it is for an event A to occur. We denote the probability of event A as $P(A)$.

We denote the probability of an event A as $P(A)$.

From a traditional standpoint, probability is presented in terms of a **frequentist** view where data instances are drawn from a repeatable random sample with parameters

There are two probability schools of thought: *frequentist* and *Bayesian*.

that remain constant during the repeatable process. These assumptions enable us to determine a frequency with which an event occurs. In contrast, the **Bayesian** view takes the approach that data instances are observed from a realised sample and that the parameters are unknown. Since the repeatability of the data sample does not hold under this view, the Bayesian probability is not given in terms of frequencies, but instead it represents a state of knowledge or a state of “belief”.

The Bayesian approach is named after the 18th century English statistician and Presbyterian minister Thomas Bayes who is known for formulating a theorem that bears his name: *Bayes' theorem*. Bayes' stardom in the scientific world is grounded on a posthumous publication¹¹ presented to the Royal Society by his friend Richard Price.

Bayes' theorem states that the probability of a particular hypothesis is given by both current information (data) and prior knowledge. The prior information may be the outcome of earlier experiments or trials, or even educated guesses drawn from experience. This is the reason why many frequentist practitioners have shunned the Bayesian approach for centuries. Nonetheless, Bayesian statistics has stood the test of time¹² and demonstrated its usefulness in many applications. Let us consider the set of all possible events, Ω , which is our *sample space*. Event A is a member of the sample space, as is every other event. The probability of the sample space is $P(\Omega) = 1$. The probability $P(A)$ is given by:

In the Bayesian view, probability reflects a state of knowledge.

¹¹ Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions*. 53, 370–418.

¹² McGrayne, S. (2011). *The Theory that Would Not Die: How Bayes' Rule Cracked the Enigma Code, Hunted Down Russian Submarines, & Emerged Triumphant from Two Centuries of Controversy*. Yale University Press.

$$P(A) = \frac{|A|}{|\Omega|}, \quad (6.20)$$

where $|A|$ denotes the cardinality of A . We show a Venn diagram of this situation in [Figure 6.7.a](#)). If $|A|$ were to have equal cardinality to Ω , then the probability of A would be at most 1.

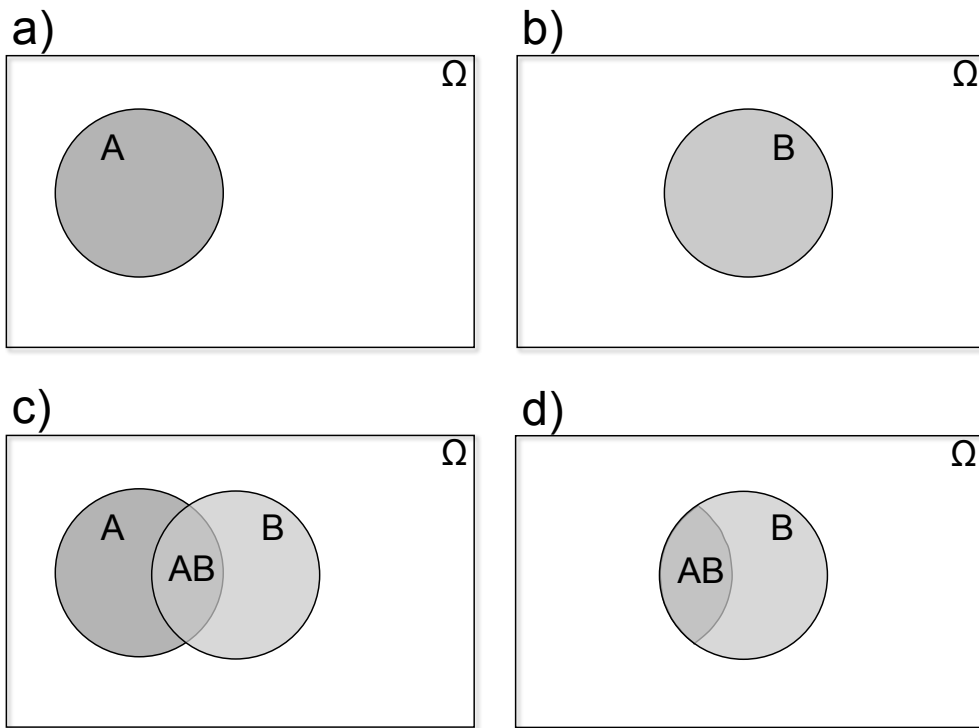


Figure 6.7: Venn diagrams to visualise Bayes' theorem.

For a different event B , we have a similar situation (shown in [Figure 6.7.b](#)) and the probability $P(B)$ is:

$$P(B) = \frac{|B|}{|\Omega|}. \quad (6.21)$$

We are using the frequentist view in this case.

Thinking about the cancer dataset we analysed the [Section 6.3.2](#), let us consider A to be the set of women with breast cancer, and B the set of women whose test for cancer was positive. Any woman in either of the sets above would be interested to know if her test was positive *and* if she actually has cancer. This would be given by the intersection of A and B , namely $A \cap B$, or AB for short. This would be the situation depicted in [Figure 6.7.c](#)). So, how do we calculate that? Actually, in the same fashion as before:

The probability of events A and B happening together is related to the intersection $A \cap B$.

$$P(AB) = \frac{|AB|}{|\Omega|}. \quad (6.22)$$

We are using AB as a shorthand for $A \cap B$.

Notice that the calculation above aims at excluding the following events: 1) women with cancer and with a negative test result ($A - AB$), and 2) women with a positive result but without cancer ($B - AB$).

All in all, for this example, the question of real importance is whether for a randomly selected woman, *given that the test is positive, what is the probability that she has cancer?* In terms of our Venn diagrams (see [Figure 6.7.d](#))) the question above is equivalent to: *given that we are in region B , what is the probability that we in fact are in region AB ?*

The next question to ask is the probability of event A , given event B , i.e., $P(A|B)$.

We denote the conditional probability of A given B as $P(A|B)$, and it can be calculated as follows:

$$P(A|B) = \frac{|AB|}{|B|} = \frac{\frac{|AB|}{|\Omega|}}{\frac{|B|}{|\Omega|}} = \frac{P(AB)}{P(B)}. \quad (6.23)$$

It is also possible to calculate $P(B|A)$:

$$P(B|A) = \frac{P(BA)}{P(A)}, \quad (6.24)$$

and for our example, it will give us the probability of a randomly selected woman to test positively given that she has the condition. Incidentally, if events A and B are independent from each other, information about one does not affect the other and thus $P(A|B) = P(A)$. This means that $P(AB) = P(A)P(B)$.

If events A and B are independent from each other, information from one does not affect the other.

The numerators of Equations (6.23) and (6.24) are actually the same. This means that we can write the following:
 $P(A|B)P(B) = P(B|A)P(A)$, which in turn can be rearranged as

$$P(AB) = P(BA).$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \quad (6.25)$$

This is Bayes' theorem.

This result is what we know as **Bayes' theorem**. We call $P(A|B)$ the *posterior* probability, $P(A)$ the *prior* probability and $P(B|A)$ the likelihood. Bayes' theorem can be thought of as a rule that enables us to update our belief about a hypothesis A in light of new evidence B , and so our posterior belief $P(A|B)$ is updated by multiplying our prior

$P(A|B)$ is the posterior probability and $P(A)$ is the prior.

belief $P(A)$ by the likelihood $P(B|A)$ that B will occur if A is actually true. This rule has had a number of very successful applications and perhaps one of my favourite ones is a very early one by Pierre Simon Laplace to determine the mass of Saturn¹³.

In Bayesian statistics we tend to work along the following steps:

1. We first set out a probability model for an unknown parameter of interest, and include any prior knowledge about the parameter, if available at all
2. Using the conditional probability of the parameter on observed data, we update our knowledge of this parameter
3. We then evaluate the fitness of the model to the data and check the conclusions reached based on our assumptions
4. We may then decide to start over again using the new information gained as our starting prior

The priors we use in the first step above reflect the best approximation to what we are interested in modelling. This information may come from expert information, researcher intuition, previous studies, other data sources, etc. In the past, Bayesian analysis would require a lot of manual computation. These days, the help of computers has lowered this barrier considerably as we shall see in the next section.

¹³ Laplace, P. and A. Dale (2012). *Pierre-Simon Laplace Philosophical Essay on Probabilities: Translated from the fifth French edition of 1825 With Notes by the Translator*. Sources in the History of Mathematics and Physical Sciences. Springer New York.

These are some typical steps followed in Bayesian statistics.

The priors reflect the best approximation to what we are interested in modelling.

6.4.1 Naïve Bayes Classifier

LET US NOW SEE HOW Bayes' theorem can be used in a classification task. A typical example, and one that most of us are familiar with, is the task of determining if a new email that arrives to our inbox is a good email ("ham") or a bad email ("spam"). Let us imagine that we have an existing corpus of email data that can be decomposed in such a way that we end up with a dataset containing $\{x_i\} = x_1, x_2, \dots, x_n$ features and we are interested in classifying new emails as belonging to the class S of spam emails.

A collection of text documents is called a *corpus*.

We can use Bayes' theorem to calculate the conditional probability of an email belonging to class S given the observation of features $\{x_i\}$ as:

$$P(S|\{x_i\}) = \frac{P(\{x_i\}|S)P(S)}{P(\{x_i\})}. \quad (6.26)$$

We are calculating the probability of an email to be spam given the features we observe in the corpus.

The value of the likelihood $P(\{x_i\}|S)P(S)$ can be obtained directly from the training data: We know that we have spam emails correctly labelled as such. We can therefore obtain the probability of observing the set of features $\{x_i\}$ given the spam email sample we have at our disposal. Similarly, the value of the prior $P(S)$ can also be obtained from the data, as the training dataset includes both spam and ham. Finally, $P(\{x_i\})$ is a constant that does not depend on the class S and can be incorporated at the end of all our computations.

The majority of the quantities needed can be readily calculated.

On the whole, the most difficult part in the calculation outlined above is the estimation of the likelihood function $P(x_1, x_2, \dots, x_n | S)$. To get this value, we need to have a lot of data in order to cover every possible combination of features $\{x_i\}$ and obtain a good estimation. This may be achievable, but remember that a model is just a reasonable approximation to reality. As such, we can make a suitable assumption to simplify the computation of this quantity.

The hardest part is the likelihood function $P(\{x_i\} | S)$.

Naïvely, we can assume for example that the features $\{x_i\}$ are conditionally independent from each other. If that is the case we can rewrite the likelihood function as:

$$P(x_1, x_2, \dots, x_n | S) \simeq P(x_1 | S) \cdot P(x_2 | S) \cdot \dots \cdot P(x_n | S). \quad (6.27)$$

We can make the calculation tractable by assuming that the features are conditionally independent from each other.

It is this naïve assumption that makes the calculation above tractable and gives the name to this classification algorithm.

6.4.2 *Naïve Bayes in Action*

THE EXAMPLE WE ARE GOING to follow for the naïve Bayes classifier is based on text classification. Instead of a spam detector, we will work with X (formerly known as Twitter) data coming from my own streams @quantum_tunnel, and @rogueloop. Our task is to determine if a given tweet is about “data science” or not. The dataset is available¹⁴ at <https://dx.doi.org/10.6084/m9.figshare.2062551.v1>.

You can follow me on X (formerly known as Twitter): @quantum_tunnel and @rogueloop.

¹⁴ Rogel-Salazar, J. (2016a, Jan). Data Science Tweets. <https://dx.doi.org/10.6084/m9.figshare.2062551.v1>

The corpus to be used in this task is already split into training and testing datasets. The training data contains 324 labelled tweets related to “data science” in a file called

Train_QuantumTunnel_Tweets.csv with three columns: A “data science” label, the date when the tweet was published and the actual published text of the tweet. The testing dataset is not labelled and contains 163 tweets. Let us load the training data into a pandas dataframe:

```
import pandas as pd

train = pd.\
    read_csv('Train_QuantumTunnel_Tweets.csv',\
             encoding='utf-8')
```

The corpus of tweets is already split into training and testing sets.

In Python 3 it is important to specify the encoding of a text file. In this case UTF-8.

We can inspect the data by slicing the dataframe:

```
print(train[62:64])
```

Tweets 62 and 63 actually read:

Tweet 62: *And that is [Chapter 3](#) of “Data Science and Analytics with Python” done... and moving on to the rest! Super chuffed! #BookWriting*

Tweet 63: *See sklearn trees with #D3 <https://t.co/UYSioXbcbu>*

The content of the tweets has not been pre-processed. We will clean the input in the next steps.

We would like to pre-process the text of the tweets to get rid of URLs and hashtags. We can do so by writing a function as follows:

```
import re

def tw_preprocess(tw):
    ptw = re.sub(r'\"http\S+', '', tw)
    ptw = re.sub(r'\"#', '', ptw)
    return ptw
```

We will use the regular expressions package re.

We can now apply the function to the relevant column in the Pandas dataframe:

```
train['Tweet'] = train['Tweet'].\
    apply(tw_preprocess)
```

The apply method of a Pandas dataframe allows us to pre-process our corpus.

We are interested in characterising the tweets by the words that appear in each of them, and so these words will be our features. To that end, we need to tokenise our tweets and generate a term-document matrix. In other words, a matrix whose rows correspond to documents (in this case tweets) and its columns correspond to terms (words). We can get this done with the help of CountVectorizer in Scikit-learn:

```
from sklearn.feature_extraction.text \
    import CountVectorizer

vectoriser = CountVectorizer(lowercase=True,\
    stop_words='english',\
    binary=True)
```

A term-document matrix is a matrix whose rows correspond to documents and its columns to words.

CountVectorizer enables us to create our term-document matrix.

We can now apply our vectoriser to the training tweets as follows:

```
X_train = vectoriser.\
    fit_transform(train['Tweet'])
```

We use our vectoriser to transform our corpus.

The result is a large sparse matrix, so printing it is not a good idea. Nonetheless, we can still see the vocabulary that has been gathered from the training set by fit_transform. This can be done with get_feature_names_out:

```
> vectoriser.get_feature_names_out()[1005:1011]

['putting', 'python',
 'quantitative', 'quantum',
 'quantum_tunnel', 'question']
```

We can list the vocabulary created with the help of `get_feature_names`.

We are now in a position to create our model using the sparse matrix generated by our vectoriser and the labels provided with the training dataset:

```
from sklearn import naive_bayes

model = naive_bayes.MultinomialNB().\
    fit(X_train, list(train['Data_Science']))
```

We are using the `MultinomialNB` algorithm from `naive_bayes`.

We have not employed cross-validation as yet, but we can certainly take a look at the scores obtained with cross-validation as follows:

```
> import sklearn.model_selection as ms
> ms.cross_val_score(naive_bayes.\
    MultinomialNB(), X_train, train['Data_Science'],\
    cv=3)
> print(scores)

[0.71296296 0.77777778 0.72222222]
```

In this case we use cross-validation with three sets, enabling us to see the scores attained.

We can see the confusion matrix that we obtain for the training dataset with the model we just created:

```
> from sklearn.metrics import confusion_matrix
> confusion_matrix(train['Data_Science'],\
    model.predict(X_train))

array([[195,  1],
       [ 0, 128]])
```

We discussed confusion matrices in [Section 6.1.1](#).

Finally, we can now apply our model to the testing dataset provided. We need to load the data and let us apply the same pre-processing we used for the training dataset:

```
test = pd.\
    read_csv('Test_QuantumTunnel_Tweets.csv',\
    encoding='utf-8')

test['Tweet'] = test['Tweet'].\
    apply(tw_preprocess)
```

We need to load the training dataset and apply the same transformation used for training.

We now have to apply the vectoriser to the testing dataset via the transform method and to finally apply the predict method of the model:

```
> X_test = vectoriser.transform(test['Tweet'])
> pred = model.predict(X_test)
> print(pred[:10])

[1 0 0 0 1 1 0 0 1 0]
```

We have to use the transform method of our vectoriser before running our prediction.

We can also see the probability assigned to each tweet as follows:

```
> pred_probs = model.predict_proba(X_test)[: ,1]
> print(pred_probs[:10])
```

```
[0.65535414 0.05017025 0.2080205
0.01821897 0.50935507 0.6324575
0.39506173 0.02644639 0.89253171
0.39506173]
```

The `predict_proba` method lets us see the probabilities assigned to each of the testing tweets.

Let us check for example the probability assigned to the tweet with id 103:

```
> pred_probs[102]
```

```
0.9996193373887406
```

Remember that Python starts counting from 0.

Let us see the text of this tweet:

Finished writing [Chapter 4](#) for my DataScience and analytics with Python book. Moving on to discussing some classification analysis

The rest of the data can be checked in a similar fashion. Notice that we have used a very small corpus for this demo, but you can see how powerful Bayes' theorem is, even when making the naïve assumption we described earlier in this chapter. Furthermore, in this example we were not particularly careful when cleaning the data, for instance we left numbers and punctuation in the corpus, also we did not apply any stemming on the text either.

More sophisticated pre-processing of the corpus is left to the reader as an exercise: Get rid of punctuation, numbers, etc.

6.5 *Being Zen with Data: Dealing with Imbalanced Classes*

AS WE HAVE SEEN, CLASSIFICATION can be a very useful task to complete. However, as any Jackalope Data Scientist knows, datasets rarely spread out evenly across attributes of interest, and therefore the number of items that belong to each class will likely not be evenly distributed. Picture the following situation: You are on a mission to predict equipment malfunctions in a power plant. Thankfully, serious malfunctions should be a small fraction of all equipment operations. Or consider testing individuals for cancer or a virus. Thankfully, positive results should be a small fraction of those tested. Here are a few more scenarios to tickle your analytical fancy:

- A financial institution detecting fraudulent credit card transactions
- A company predicting which customers are at risk of leaving (churn)
- An online market predicting which users will buy certain items in their portal
- A company scanning network traffic to detect malware or unusual port activity, etc.

In these and many other similar situations, data points from the adverse class are often few and far between. On the one hand, this is great as the adverse situation is not a common occurrence. On the other hand, however, this is not great

Datasets rarely have evenly distributions among classes.

COVID-19, anyone?

Some examples where imbalance is a normal occurrence.

I am sure you can think of some others yourself.

news for us as data scientists, as we are actually interested in avoiding misclassifying these critical elements. As you can imagine, having perfectly balanced data to be used in classification tasks is inherently rare and this imbalance can pose significant challenges, especially when the minority or “rare” class carries the most critical information. The detection of the “rare” class is tricky because standard algorithms often become biased toward the majority class, leading to poor performance in identifying the minority class. In this section we will cover some things that you can do to improve your classification models when facing unbalanced datasets.

Having perfectly balanced datasets is not common.

So, what does *imbalanced data* refers to? Well, it refers to datasets where the distribution of observations in the target class is lopsided. In simpler terms, for a binary situation, one class label has a lot more observations, while the other has a sparse handful. When one class vastly outnumber the others in a classification task, we end up with imbalanced data. This imbalance can cause machine learning models to skew their predictions, favouring the majority class.

Imbalance can cause our models to skew their outcomes.

Consider the following example: Imagine that I have a bag full of Skittles. Let us say that there are 100 Skittles and 98 of them are green, whereas the remaining 2 are red. Without using a clever algorithm like the ones covered in this chapter, I can create a naïve classifier that predicts selecting a Skittle from the bag to be green based on numbers alone. This will make my classifier 98% accurate. Perhaps for my purposes, this classifier will be fit for purpose, but in many other cases, more data, and for that matter, more diverse

Other brands of similar looking sweets are available: Smarties, Lunetas, etc.

data would be a great solution for this problem. When that is not possible, we may want to consider using techniques like those described in this section.

Some of the issues that we may encounter when creating classification models in the face of imbalanced datasets include:

- **Algorithm Bias:** Models may develop a bias towards the majority class, often predicting it as the default output.
- **Minority Class Noise:** Observations from the minority class might be mistaken for noise and overlooked by the model.
- **Misleading Accuracy Scores:** An imbalanced dataset can lead to deceptive accuracy scores, painting an overly rosy picture of model performance.

This is the case of the Skittles example above.

By tackling these challenges head-on with the right strategies, we can ensure our models are fair and accurate, even when the data is playing favourites.

Tackling these issues can ensure the creation of fairer and more accurate models.

We have established that imbalanced datasets present some unique challenges compared to ideal datasets. The type of strategy to use to tackle the issues mentioned above can be better implemented by understanding different types of imbalances that we may encounter. Let us take a look.

- **Between-Class Imbalance:** This occurs when the number of data points differs significantly between classes. This is the same situation as our bags of Skittles where we have a large number of green Skittles but only a few red ones. Another example is given by data from mammograms to

Again, this was the case of our Skittles example above.

detect breast cancer, similar to the data used in [Section 6.3.2](#). Imagine a mammography dataset using images (mammograms) to predict breast cancer. Imagine that you have a set where over 90% of cases are negative for cancer, and less than 10% are positive for the disease.

With enough data samples in both classes, accuracy improves because the sampling distribution better reflects the actual data distribution. However, due to the law of large numbers, the majority class will always have better representation than the minority class.

The majority class will tend to have better representation, skewing our results.

- **Within-Class Imbalance:** This happens when there is balance between classes overall, but one class is not well-represented within a certain feature of the dataset. Let us take a closer look. Imagine a dataset for binary classification with plenty of data points for two classes (green and red) of Skittles of different sizes (large and small). Imagine now that the green class is sparsely populated in terms of the large size of Skittle. This can cause prediction issues similar to those seen with between-class imbalances in those specific features.

The imbalance may be due to an under- or over-representation of a portion of data within a class.

- **Intrinsic and Extrinsic Imbalance:** Intrinsic imbalance arises from the nature of the dataset itself, while extrinsic imbalance is related to external factors like time, storage, and data analysis constraints. Intrinsic characteristics are straightforward and commonly seen, but extrinsic imbalances can add complexity and exacerbate issues.

Imbalances can be due to the nature of the dataset, but also they come from external factors.

For instance, consider the detection is spam email in a mail server. The imbalance can be influenced by whether all email recipient addresses are analysed or just a few.

Furthermore, if logs of suspected spam emails are stored, while logs for normal emails are not (or only selectively stored), this can further skew the dataset.

Let us now look at some techniques that can be used to restore balance.

Let us go zen with your dataset.

6.5.1 *Resampling: Oversampling or Undersampling*

RESAMPLING IS A VERY POPULAR and easy to understand technique. It comes in two flavours, oversampling and undersampling. Let us take a look at each of them in turn.

Oversampling involves creating new data points for the minority class to balance out the dataset. Essentially, more instances are added to the under-represented class. In practice, using the example of fraudulent transactions, our dataset will have a large number of legitimate transactions, but only a small number of fraudulent ones. To balance this, we generate additional fraudulent transaction data points until both classes have similar representation.

Oversampling adds synthetic data points to the minority class

To generate these new samples, the most common approach is to create new data points close to existing minority samples or between two samples, for example. This method helps ensure the new data points are realistic and relevant. However, there are some drawbacks to oversampling, namely:

We create new datapoints between two existing ones.

- **Overfitting:** Adding synthetic data carries the risk of the model learning from noise, especially if the new points are generated from noisy data.

- **Added Noise:** Randomly adding values can introduce more noise into the model.

Alternatively, we can use **undersampling**, which involves removing data points from the majority class to balance the dataset. This means taking fewer samples from the majority class until both classes have an equal number of data points. In our fraud detection example, since most transactions are not fraudulent, undersampling implies the removal of some legitimate transaction data points to reduce the imbalance. We can do this by removing samples that are in close proximity to each other.

Undersampling removes data points from the majority class.

Undersampling works best when there are enough data points in the majority class. This method does not distort the data distribution since artificial data points are not added. However, it can make the statistical properties of the distributions 'looser,' as the majority class now has fewer data points.

Undersampling works best when there are enough data points in the majority class.

In either case, our end goal is to balance the dataset, ensuring both majority and minority classes are represented equally. This balanced approach allows the classifier to assign equal importance to each class during training, leading to more reliable and fair predictions.

Remember that our goal is to balance our dataset.

Let us see how to implement resampling with Scikit-learn. First let us create a dataset with a 90%-10% imbalance for a binary classification task. We will use a function called `make_classification` available in Scikit-learn. The function creates normally distributed clusters around the vertices of

We can create datasets for demo purposes with `make_classification`.

an `n_informative`-dimensional hypercube, where `n_informative` is the number of informative features.

The hypercube has sides of length `2*class_sep`. Larger sides in the hypercube spread out the classes, making the classification task easier. We can specify the number of classes with `n_classes` and the distribution among the classes with `weights`, the number of samples with `n_samples` with `n_features` being the total number of features of which we have `n_redundant` features which are generated as random linear combinations of the informative features.

These are the different parameters we can use with `make_classification`.

In this case we are creating a dataset with 1000 binary samples using ten features of which four are informative and two are redundant:

```
X, y = make_classification(n_classes=2,
                           class_sep=3,
                           weights=[0.1, 0.9],
                           n_informative=4,
                           n_redundant=2,
                           n_features=10,
                           n_clusters_per_class=1,
                           n_samples=1000,
                           random_state=48)
```

`make_classification` enables us to create diverse datasets with a variety of parameters.

Let us use pandas dataframes to make our example easier to manage:

```
> X = pd.DataFrame(X)
> y = pd.DataFrame(y, columns=['label'])
> df = pd.concat([X, y], axis=1)
> df['label'].value_counts()
```

```
label
1    897
0    103
```

Here we have a dataset with 90% of data points labelled 1.

As we can see, we have many more data points for the 1 class than the 0 class. Let us create two new dataframes with each of the classes as follows:

```
df_majority = df[df.label == 1]
df_minority = df[df.label == 0]
```

We create separate dataframes with each of the classes in our dataset.

We can first try to undersample the majority class. We are going to use the `resample` method available in `sklearn.utils`. The resampling will be done without replacement, as we have more examples to choose from. We are going to generate the same number of data points as in the minority dataset.

```
from sklearn.utils import resample
df_majority_under = resample(df_majority,
                             replace=False, n_samples=len(df_minority),
                             random_state=7)
```

We can resample our dataset with the help of `resample`.

Let us combine the new undersampled majority dataset with the original minority one:

```
> df_majority_under = pd.concat([df_majority_under,
    df_minority])
> df_majority_under.label.value_counts()

label
1      103
0      103
```

Alternatively we can oversample the minority class as follows:

```
df_minority_over = resample(df_minority,
    replace=True,
    n_samples=len(df_majority),
    random_state=7)
```

Resampling can be done with replacement.

In this case we are resampling with replacement, as we will need to create more samples than are available to match the number in the majority class. Let us see the combined oversampled minority with the original majority dataset:

```
> df_minority_over = pd.concat([df_majority,
    df_minority_over])
> df_minority_over.label.value_counts()

label
1      897
0      897
```

We obtain a balanced dataset with the number of samples matching the majority class.

6.5.2 Synthetic Minority Oversampling Technique (SMOTE)

ANOTHER TECHNIQUE THAT WE CAN use to address imbalances in our datasets is the Synthetic Minority Oversampling Technique or SMOTE for short. It generates new instances for the minority class synthetically and unlike simply duplicating records, SMOTE enhances diversity by creating artificial instances. In other words, SMOTE examines instances in the minority class, selects a random nearest neighbour using k -nearest neighbours (KNN), and generates a synthetic instance randomly within the feature space.

SMOTE¹⁵ generates new samples in between existing data points based on their local density and their borders with the other class; not only does it perform oversampling. The steps in SMOTE for each minority sample are as follows:

1. Find its k -nearest minority neighbours
2. Randomly select j of these neighbours
3. Randomly generate synthetic samples between the j selected neighbours

One of the things to bear in mind when using SMOTE is overgeneralisation. Since SMOTE generalises the minority class, we may inadvertently disregard the majority one. This may be an issue in datasets with highly skewed class distributions. It is also important to remember that the number of synthetic samples generated is fixed in advance.

SMOTE uses the KNN algorithm we covered in [Section 6.2](#) to help us balance our dataset.

¹⁵ Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*. 16, 321–357.

We should be mindful of overgeneralising our dataset.

In this case we are going to use the implementation in the imbalanced-learn library. First let us create our dataset as done in the previous section with the help of `make_classification`:

```
from sklearn.datasets import make_classification

X, y = make_classification(n_classes=2,
                           class_sep=3, weights=[0.1, 0.9],
                           n_informative=4, n_redundant=2,
                           n_features=10, n_clusters_per_class=1,
                           n_samples=1000, random_state=48)
```

Make sure you install the imbalanced-learn library
<https://imbalanced-learn.org>

As before, `make_classification` is used to create our demo dataset.

As before, let us create pandas dataframes for our dataset and see the distribution of samples:

```
> X = pd.DataFrame(X)
> y = pd.DataFrame(y, columns=['label'])
> y.value_counts()

label
1      897
0      103
```

We are creating a dataset with a 90/10 distribution of classes.

We can now import SMOTE from the imbalanced-learn library and instantiate a model to resample our dataset:

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='auto',
               k_neighbors=5, random_state=14)
X_resampled, y_resampled = smote.fit_resample(X,y)
```

We are selecting $j = 5$ as the number of neighbours to consider.

In the example above, we have chosen $j = 5$ as the number of k -nearest neighbours to consider for our sampling.

Let us see the final distribution of our resampled dataset:

```
> y_resampled.value_counts()
```

label	
0	897
1	897

Our dataset is balanced as expected.

6.6 Summary

CLASSIFICATION IS ANOTHER VERY IMPORTANT tool to have under our Jackalope Data Scientist belt. Together with clustering, these techniques let us understand our data and obtain insights which in turn can be used to get actionable predictions that drive decision-making. We saw how classification is a supervised learning task that exploits information about the classes contained in our data in order to make predictions about unseen data instances, enabling us to tackle a range of problems.

We discussed three well-known techniques, namely: k -Nearest-Neighbours (KNN), Logistic Regression and Naïve Bayes. In KNN we use the similarity of a given data instance and its closest data points. Logistic regression extends the linear model we discussed in [Chapter 4](#) using a sigmoid function to estimate the probability that a given data instance belongs to a class. Finally, naïve Bayes uses Bayes' theorem to update the posterior probability that a

data instance belongs to a class, given that certain features are observed in the data.

We saw how confusion matrices, true and false positives and negatives, as well as the recall and fallout measures, let us evaluate how well our classifiers are performing. The same applies to the Receiver Operator Characteristic (ROC) curve and the Area Under this Curve (AUC). We saw the algorithms described above applied to a variety of problems, from cancer to X data (formerly, Twitter) as well as wine and flower datasets.

Finally, we talked about imbalanced dataset, where one class significantly outnumbers another, and how this imbalance can skew machine learning models. We addressed various challenges and solutions associated with such datasets with techniques like oversampling, where synthetic data points are added to the minority class, and undersampling, where data points from the majority class are removed. We also covered the use of Synthetic Minority Oversampling Technique (SMOTE) and its role in generating synthetic instances to enhance minority class representation. These techniques ensure fair and accurate predictive modelling in the face of imbalanced data, which is a more common occurrence than we may think.

7

Decisions, Decisions: Hierarchical Clustering, Decision Trees and Ensemble Techniques

THE USE OF DIAGRAMS AND illustrations in science, technology, news, business and elsewhere is nothing new. Along with tables and inscriptions, they provide us with useful representations of concepts and data we need to communicate. They make it easier for us to organise our knowledge and information. An image that has been widely used for this purpose for centuries and across cultural and disciplinary landscapes is that of the tree¹. It is easy to see the appeal: Trees enable us to represent information in a hierarchical manner that is easy to follow.

In this chapter we will address a few techniques that, in one way or another, have been inspired by the metaphorical representation of a tree: From roots to branches and leaves. First we will talk about a clustering algorithm that organises groups of data in a hierarchical manner. Then we will talk about decision trees, a tool that is widely used in decision

¹ Lima, M. and B. Shneiderman (2014). *The Book of Trees: Visualizing Branches of Knowledge*. Princeton Architectural Press.

The techniques we will discuss in this chapter have been inspired by trees.

analysis applications and operations research. Towards the end of the chapter we will combine trees in ensembles giving rise to random forests.

7.1 Hierarchical Clustering

THE WORD HIERARCHY EVOKES THE idea of a system where information is ranked according to a relative status, giving rise to different levels. Hierarchical clustering is an unsupervised learning task whose goal is to build a hierarchy of data groups. The hierarchy can be built from the “bottom-up”. In this case each data instance starts in its own cluster, and we successively merge these clusters as we go up in the levels of the hierarchy. This is known as **agglomerative clustering**. In contrast, **divisive clustering** takes the opposite approach, starting with all data instances and performing splits as we go down the levels of the hierarchy. The results of hierarchical clustering are presented in a tree-like structure called a **dendrogram**.

As we know, clustering relies on the existence of a similarity measure among data instances. In a dendrogram, data points are joined together from the most similar (closest) to the most different (further apart). Let us recall our discussion from [Section 5.2](#) about the k -means algorithm. We need to determine k different clusters with three things: The number of clusters k , an initial assignation of data points to the clusters, and a distance or similarity measure $d(x_i, x_j)$. In hierarchical clustering, we only need one thing: A similarity measure among groups of data points.

Hierarchical clustering is an unsupervised learning task.

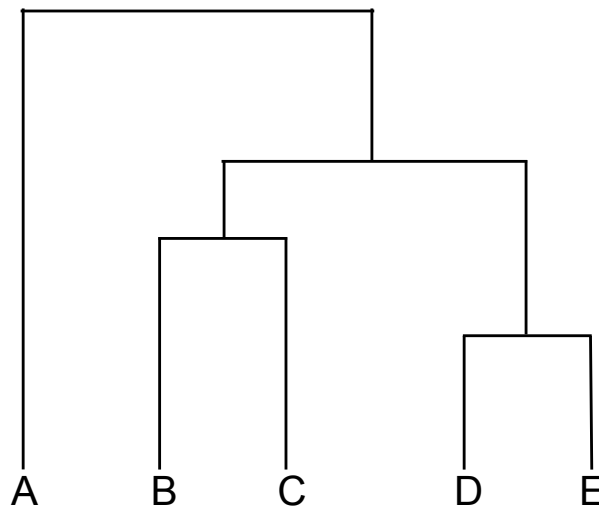
The results of hierarchical clustering are represented by a dendrogram.

In a dendrogram, data points are joined together from the most similar to the most different.

Let us take a look at the agglomerative clustering approach, where our starting point is a situation in which each of our N data instances is on its own cluster. Given the similarity measure $d(x_i, x_j)$, we iteratively merge the two closest groups, repeating until all data instances end up in a single cluster. The point at which two clusters are joined up is called a *node*. At each of the nodes of the resulting tree, we are effectively segmenting our data in a sequential manner.

In hierarchical clustering we, only need to define a similarity measure.

A node is a point where two clusters are joined.



It is easy to see that the similarity among merged groupings decreases monotonically at every node created by agglomerative clustering. The more dissimilar two clusters are, the more distant they are. This is actually depicted in a dendrogram: The height of each branch, or *clade*, tells us how similar or different the groupings are from each other, the greater the height, the greater their dissimilarity. We show a sample dendrogram in [Figure 7.1](#). We see that clusters D and E are more similar to each other than B and

Figure 7.1: A dendrogram is a tree-like structure that enables us to visualise the clusters obtained with hierarchical clustering. The height of the clades or branches tells us how similar the clusters are.

The height of the branches tells us how similar the clusters are.

C. Also, cluster A is significantly different from the rest of the other clusters.

At the start of the agglomerative process, finding similar data points is straightforward, as we are comparing individual data points. However, once we have more than one instance per cluster, we need to define what we mean by similarity among groups or clusters. Given two clusters F and G , we have a few choices:

1. **Single linkage:** We define the similarity by the closest instance pair: $d_{SL}(F, G) = \min d(x_i, x_j)$ with $x_i \in F$ and $x_j \in G$
2. **Complete linkage:** The similarity is given by the farthest pair: $d_{CL}(F, G) = \max d(x_i, x_j)$ with $x_i \in F$ and $x_j \in G$
3. **Group average:** The similarity is given by the mean similarity between F and G :

$$d_{GA}(F, G) = \frac{1}{|F||G|} \sum_i \sum_j d(x_i, x_j) \text{ with } x_i \in F \text{ and } x_j \in G$$

Be aware that single linkage may lead to a situation where a sequence of close data points gives rise to a chain. This is because clusters tend to merge very early on in the process. By the same token, complete linkage may not merge close groups in cases where there are far-apart outliers in the clusters. Finally, group average depends on the scale of the similarities taken into account, and overall is a good compromise between the three choices detailed above.

Another alternative for cluster merging is the so-called Ward method based on an analysis of variance (ANOVA)

Defining what we mean by similarity is a very important step in the process.

Some similarity measures used in hierarchical clustering.

As usual, there are advantages and disadvantages with each of these choices.

An alternative is the use of the Ward method.

approach². In this method, at each stage, two clusters would merge when they provide the smallest increase in the combined error sum of squares from one-way univariate ANOVAs that may be performed for each feature.

² Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.

Finally, remember that hierarchical clustering is an unsupervised algorithm, and in cases where groupings do not actually exist in the data, the algorithm will still impose a hierarchy. Cross-validation may provide some clues as to whether the clusters we obtain may actually make sense.

Remember that hierarchical clustering is an unsupervised task.

7.1.1 Hierarchical Clustering in Action

LET US USE THE IRIS dataset to demonstrate the use of hierarchical clustering with Python. Although Scikit-learn provides hierarchical clustering implementations such as `AgglomerativeClustering` in this case we are going to use the hierarchy module in SciPy.

We will use SciPy with the Iris dataset.

Let us retrieve the Iris dataset and load the data to a suitable variable:

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
```

By now, we are very well acquainted with loading the Iris dataset.

From SciPy, we will be using the `linkage` and `dendrogram` functions from `hierarchy`, and from `distance` we will call the function `pdist`:

```
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.spatial.distance import pdist
```

We will use the functions `linkage`, `dendrogram` and `pdist` from SciPy.

The `pdist` function lets us calculate the pairwise distances between the features in our data, whereas `linkage` is used to run hierarchical clustering on a distance matrix:

```
X_dist = pdist(X)

X_link = linkage(X, method='ward')
```

We calculate pairwise distances among our data points and run the hierarchical clustering algorithm.

We are using the Ward method in our agglomerative clustering task.

We can obtain a measure of how the pairwise distances in our data compare to those implied by the hierarchical clustering. This can be done with the *cophenetic coefficient*³. The better the clustering preserves the original distances, the closer this coefficient is to 1:

```
> from scipy.cluster.hierarchy import cophenet
> coph_cor, coph_dist = cophenet(X_link, X_dist)
> print(coph_cor)

0.872601525064
```

³ Farris, J. S. (1969). On the cophenetic correlation coefficient. *Systematic Biology* 18(3), 279–285.

A cophenetic coefficient closer to 1 indicates that the clustering preserves the original pairwise distances of the data points.

It is sometimes important to know what clusters have been merged at each iteration. This information is actually contained in the matrix returned by the `linkage` function. The *i*-th entry of this matrix tells us what clusters have

The `linkage` function provides information about what clusters have been merged together.

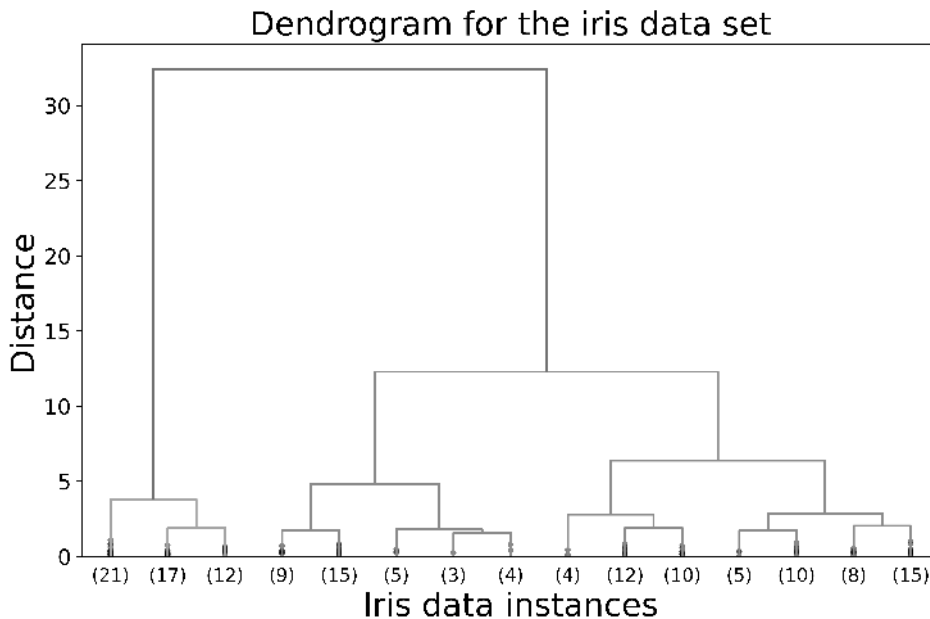
been merged at iteration i (first two entries), as well as their distance (third entry) and the number of samples contained (fourth entry). For example, we can see what data points were merged at the first iteration as follows:

```
> print(X_link[0])

[101., 142.,  0.,  2.]
```

The data instances indexed 100 and 142 were merged first.

which indicates that the data instances with indices 100 and 142 were merged at the first iteration.



We are showing the dendrogram generated for the Iris data set in [Figure 7.2](#). The numbers in brackets along the x -axis

Figure 7.2: Dendrogram generated by applying hierarchical clustering to the Iris dataset. We can see how three clusters can be determined from the dendrogram by cutting at an appropriate distance.

correspond to the number of data instances that are under each clade. Remember that we are starting from a point where each data instance is a cluster (i.e., the final leaves in our tree). In this case we have a total of 150 entries, and showing all those branches may not be that practical. The dendrogram function provides us with a way to truncate the diagram. In this case we are showing the last $p = 15$ merged clusters:

```
import matplotlib.pyplot as plt
dendrogram(X_link, truncate_mode = 'lastp',\
           p=15, show_contracted = True)
plt.show()
```

The dendrogram function lets us plot the result of our hierarchical clustering.

Finally, we can obtain the labels that the hierarchical clustering generates. This can be done with `fcluster`, which takes a threshold corresponding in this case to the distance where we want to truncate our dendrogram. From [Figure 7.2](#) we can see that at a height lower than 10 we have three distinct clusters. Let us then pick a threshold of 9 and use that within `fcluster`:

```
from scipy.cluster.hierarchy import fcluster
max_d = 9
clusters = fcluster(X_link, max_d,\
                   criterion='distance')
```

The `fcluster` function lets us truncate the dendrogram and assign labels to our data points.

The array `clusters` contains the labels 1, 2, or 3 assigned to our data points.

An alternative way of choosing the threshold can be done with the *inconsistency* measure. It compares the height h of each cluster merge to the average, and normalises the difference to the standard deviation:

```
from scipy.cluster.hierarchy import inconsistent
depth = 6
incons_measure = inconsistent(X_link, depth)
```

An alternative way to choose the threshold is with the use of the inconsistency measure.

The matrix returned by `inconsistent` contains the average, standard deviation, count and inconsistency for each merge. We can use this measure to obtain our clusters. However, this method has a high reliance on choosing the right parameters for threshold and depth:

```
clusters_incons = fcluster(X_link, t=8,\
criterion='inconsistent', depth=15)
```

The inconsistency measure can also be used to obtain our labels.

7.2 *Decision Trees*

CONTINUING WITH THE INSPIRATION PROVIDED by trees, in this section we are going to explore one of the most well-known classification algorithms, and one that is actually named after its botanical counterpart. A decision tree is an algorithm described as non-parametrical because it does not require us to make any assumptions about parameters or distributions before starting our classification task. It is also a hierarchical technique: The model is built in such a way that a sequence of ordered decisions about the values of the data features results in assigning a class label to any given data instance.

Decision trees are among the most well-known classification algorithms.

Think of these decisions as questions regarding the data.

You may be asking yourself how to recognise a decision tree, even from a very long way away. Well, you will be happy to know that a decision tree is probably better known by its

You will surely recognise a tree, even from a very long way away.

diagrammatic representation, where rules guide us through the chart to decide a final outcome. The tree structure consists of nodes and edges, where the nodes represent the test conditions or questions we need to consider for classifying our data. The edges represent the answers or outcomes to the questions we are asking. Furthermore, it is easy to understand and indeed explain how a decision is made.

A good tree must have a solid root, and a decision tree is no exception. We have a **root node** which is the node that has no incoming edges, and two or more outgoing edges. An **internal node** has one incoming edge and two or more outgoing ones: Internal nodes represent test conditions at every given level. Finally a **leaf node** has one incoming edge and no outgoing ones; leaf nodes correspond to class labels, as there are no further outcomes.

In [Chapter 2](#) we created a pandas dataframe for some animals, we detailed the number of limbs and dietary habits in [Table 2.5](#). Let us expand the information by adding a column that labels the data. We show this in [Table 7.1](#), and we will use this information to construct our first decision tree.

Animal	Limbs	Herbivore	Class
Python	0	No	Reptile
Iberian Lynx	4	No	Mammal
Giant Panda	4	Yes	Mammal
Field Mouse	4	Yes	Mammal
Octopus	8	No	Mollusc

A decision tree is a type of a directed acyclic graph.

We will talk more about explainability in [Section 7.5](#).

Even a larch has a root.

And as such, our decision trees have roots, as well as branches and even leaves.

Let us revisit our first pandas dataframe from [Chapter 2](#).

Table 7.1: Dietary habits and number of limbs for some animals.

It is possible to evaluate all possible combinations of the test conditions for a given dataset and construct all possible decision trees from there. For example, we can start by considering as our root node the test of whether or not the animal is a herbivore. We can continue by asking how many limbs the animal has, and so on. The result is the decision tree shown in [Figure 7.3](#). Note that we could have started by asking first if the animal has four limbs or not. This would have resulted in having immediately a leaf node where all the mammals in our dataset get classified in one go.

We start here by asking whether the animal in question is a herbivore or not. Then we ask about the number of limbs, and so on.

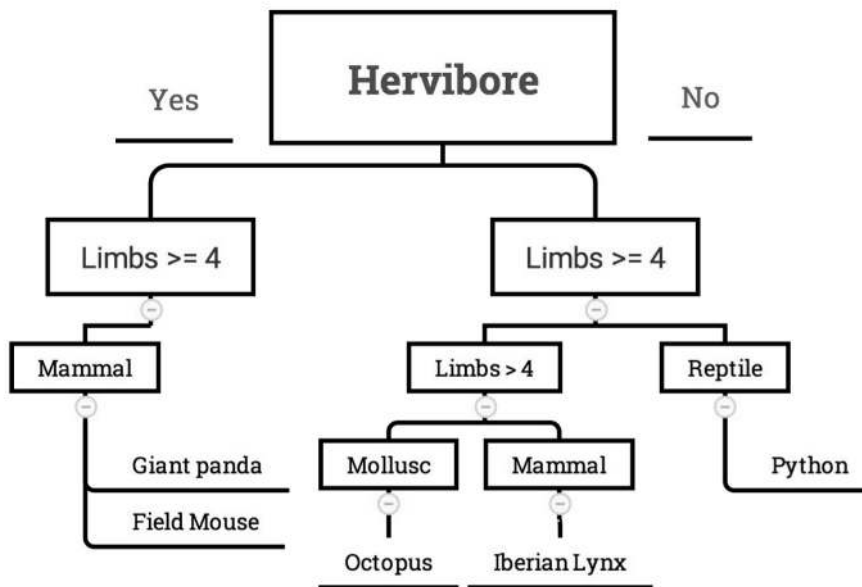


Figure 7.3: Simple decision tree built with information from [Table 7.1](#).

It stands to reason that constructing all possible decision trees is not a very practical thing to do. Instead, we are interested in applying a greedy algorithm that lets us find a

local optimum to solve our classification problem in a much faster way.

One possibility is the application of Hunt's algorithm⁴, which grows a decision tree in a top-down fashion by recursively splitting the dataset into smaller and smaller data subsets. The aim is to minimise the cost of the classification task: On the one hand taking into account the misclassification of a data instance, and on the other one determining the value of a feature exhibited by the data instance.

Selecting smaller subsets of data is an easy thing to do. However, we require smaller subsets that actually lead to leaf nodes that let us classify data instances accurately. With that in mind, we therefore need a measure to partition our dataset at every node. That measure is called *purity*: A partition is said to be 100% pure if all of the data instances in the subset belong to the same class and this means that it is not necessary to split the data any further. This would have been the case in our example above, using [Table 7.1](#), if we had started splitting the data by the number of limbs: Animals described in that table as having four limbs are all mammals.

For a binary classification problem with classes A and B , and a dataset with N_n data instances at node n , Hunt's algorithm proceeds as follows:

1. If node n is pure, i.e., all its data instances N_n belong to class A , we are done as we effectively have a leaf node for class A

⁴ Hunt, E. B., J. Marin, and P. J. Stone (1966). *Experiments in Induction*. New York: Academic Press.

Not only is it necessary to select smaller subsets, but also they need to lead to a more accurate classification.

A partition is 100% pure if all its data points belong to the same class.

2. If node n is not pure, we need to split the dataset further. We need to create a test condition that enables a partition in the dataset. This means that the node becomes an internal node
3. We run the test condition and assign each of the instances N_n to one of the two child nodes created from node n
4. These steps are applied recursively to each and every child node

Hunt's algorithm in a few simple steps.

Step 2 above indicates that we need to create a test condition. We prefer a condition that gives us the best split in terms of purity for each of the child nodes. In other words, the larger the purity, the better our classification. We need to compare the impurity of the parent node with the impurity of each of the child nodes generated after splitting our data.

The test condition we need should give us the best split in terms of purity.

With a total of c classes, let us denote the fraction of records that belong to class i at a given node n as $p(i|n)$. Some popular measures of impurity include the following:

- Entropy:

$$H(n) = - \sum_{i=1}^c p(i|n) \log_2 p(i|n). \quad (7.1)$$

A typical measure used is the entropy.

The entropy of a pure node is zero as $\log_2(1) = 0$. It reaches its maximum when all classes have equal proportions and in the particular case of a binary classifier the maximum is 1.

- Gini impurity:

$$G(n) = 1 - \sum_{i=1}^c p^2(i|n). \quad (7.2)$$

The Gini impurity is also widely used.

The Gini impurity of a pure node is zero. As it is the case for the entropy measure, the Gini impurity has a maximum value of 1 when all classes have equal proportions. The values of the Gini impurity range between 0 and 1 irrespective of the number of classes involved.

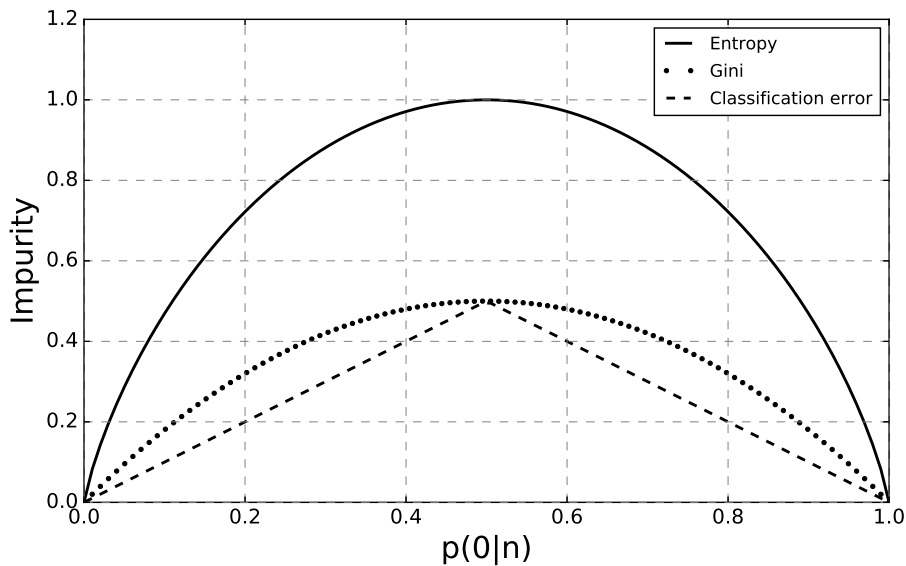


Figure 7.4: Comparison of impurity measures we can use for a binary classification problem.

- Classification error:

$$E(n) = 1 - \max_i p(i|n). \quad (7.3)$$

Classification error.

Once again, the classification error for a pure node is zero. Its values range between 0 and 1.

Figure 7.4 shows the behaviour of the impurity measures described above.

With any of the impurity functions $I(n)$ mentioned above, we are now in a position to compare the purity of the parent and children nodes with the **gain**:

$$\Delta = I(\text{parent}) - \sum_{\text{children}} \frac{N_j}{N} I(j\text{-th child}). \quad (7.4)$$

The gain enables us to compare the purity of the parent and children nodes.

When the impurity measure employed is the entropy, the gain is actually called the **information gain**.

Decision tree algorithms are not exempt from overfitting, and the worst case scenario would be one where we end up with a number of leaves equal to the number of data instances. We can restrict our tree growth to binary splits exclusively (CART algorithm) or by imposing a penalty on the number of outcomes (C4.5 algorithm). A popular regularisation employed is the *gain ratio*, which takes into account the entropy of the distribution of data instances into the children nodes:

Decision trees are not exempt from overfitting.

$$GR = \frac{\Delta_{info}}{-\sum \frac{N_j}{N} \log_2 \left(\frac{N_j}{N} \right)}. \quad (7.5)$$

The gain ratio is a popular regularisation technique employed with decision trees.

The gain ratio reduces the bias towards multi-valued features, as it considers the number and size of nodes when choosing a feature for splitting our data.

Overfitting can also be avoided by determining criteria that stop us from continuing splitting of our data. A trivial example is the case when all the data instances in a node

belong to the same class. We can set a minimum threshold on the gain and stop when the information on a branch becomes unreliable, as we are no longer achieving a gain above the threshold imposed. In this case we are *pre-pruning* our tree, and although it prevents overfitting, its calibration is not straightforward and may also stop growth prematurely.

Pre-pruning our tree consists of setting a minimum threshold on the gain.

An alternative is to work on a fully grown tree and apply *post-pruning*. We effectively examine the tree bottom-up and simplify subtrees either by replacing them with a single node (subtree replacement) or with a simpler subtree (subtree raising).

Post-pruning our tree is another alternative.

7.2.1 *Decision Trees in Action*

THE TITANIC AND ITS FATEFUL maiden voyage, from Southampton, England to New York City, US, is well-known to all of us. Famous even before sailing, the vessel was effectively a floating city, and her passengers came from all backgrounds in society: From the wealthy elite to hopeful immigrants looking to start afresh in America. On the night of April 14, 1912 the Titanic struck an iceberg, ultimately sinking the vessel. The Titanic sent distress signals by telegraph and, although there were other ships nearby, no assistance was promptly provided.

We are all familiar with the fateful maiden voyage of the Titanic.

Using information about passengers of the Titanic, we are interested in building a model based on a decision tree to say something about the chances of surviving the disaster. The data we are going to use can be obtained from

the Kaggle competition “*Titanic: Machine Learning from Disaster*”⁵ at the following URL: <https://www.kaggle.com/c/titanic>. The training data provided contains 891 records with the following attributes:

- Survived: 0= No; 1 = Yes
- Pclass: Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- Name: Passenger Name
- Sex: (female; male)
- Age: Passenger Age
- SibSp: Number of Siblings/Spouses Aboard
- Parch: Number of Parents/Children Aboard
- Ticket: Ticket Number
- Fare: Passenger Fare
- Cabin: Cabin
- Embarked: Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

⁵ Kaggle (2012). Titanic: Machine Learning from Disaster. <https://www.kaggle.com/c/titanic>

The attributes in the Titanic dataset.

Let us load the data into a pandas dataframe and get rid of some information that is incomplete. In this case, we will ignore the Ticket and Cabin columns, and drop instances without values in the rest of the dataframe:

```
titanic = pd.read_csv(u'./train.csv')
titanic = titanic.drop(['Ticket', 'Cabin'], axis=1)
titanic = titanic.dropna()
```

There is some missing data in the dataset.

In this case we are dropping the Ticket and Cabin columns, as well as records with missing data.

These transformations leave us with 712 data instances to work with. There is some extra information that may help us navigate the use of the various attributes in the dataset. Famously, the ship did not have enough lifeboats for all the crew and passengers. Furthermore, the lifeboats used were not used at full capacity. Passengers in the upper classes were the first to be helped out, leaving the rest to fend for themselves.

Let us now ensure that some of our features are actually encoded as categorical variables. In this case, we will do this for the following three features: Pclass, Survived and Sex.

```
titanic['Pclass']=titanic['Pclass'].\
    astype('category')
titanic['Survived'] = titanic['Survived'].\
    astype('category')
titanic['Sex'] = titanic['Sex'].\
    astype('category')
```

No pun intended...

Let us perform some basic exploratory analysis.

We ensure that some variables are treated as categorical.

We can assume that the old adage of *“Women and children first”* may have applied. With that in mind, let us perform some data exploration and see the percentage of passengers who survived the disaster, categorised by travelling class and gender:

```
Pclas_pct = pd.crosstab(titanic['Pclass'],
    titanic['Survived'], margins=True)

Pclas_pct['Percent'] =\
    Pclas_pct[1]/(Pclas_pct[0]+Pclas_pct[1])
```

We first construct a cross-tabulation of the travel class v survival with `crosstab`.

We can use pandas to calculate the percentage of survivors per class.

```
Sex_pct = pd.crosstab(titanic['Sex'],
                      titanic['Survived'], margins=True)

Sex_pct['Percent'] = \
    Sex_pct[1]/(Sex_pct[0]+Sex_pct[1])
```

What about the percentage of survivors per gender?

Let us now look at the numbers:

```
> print(Pclas_pct['Percent'], Sex_pct['Percent'])

Pclass
1.0    0.652174
2.0    0.479769
3.0    0.239437
All    0.404494
Name: Percent, dtype: float64

Sex
female    0.752896
male      0.205298
All       0.404494
Name: Percent, dtype: float64
```

We can now take a look at the percentages calculated above.

We can see from the data provided that about 65% of passengers travelling in 1st class survived, compared to only 23% of those travelling in 3rd class. In terms of gender, 75% of female passengers survived, compared to only 20% of males. In other words, you had a better chance of

A more thorough data exploration analysis can be done. Go ahead, give it a go!

surviving if you were a female passenger travelling in 1st class accommodation.

For our modelling we will use a subset of the features provided and concentrate on three things: Class, gender and age. Remember that Scikit-learn only accepts numerical values as data, and in this case the Sex feature is given as text, i.e., “female” and “male”. Let us pre-process our data with pandas to obtain dummy variables to encode the information in numerical labels:

```
titanic = pd.concat([titanic,\n    pd.get_dummies(titanic['Sex'])], axis=1)
```

This appends two columns to our dataframe, one called female, and the other one called male, indicating the values with False= 0 or True= 1.

We are now ready to start our modelling. Scikit-learn provides a decision tree model in `DecisionTreeClassifier`, accepting various parameters such as the impurity criterion taking values such as entropy and gini. We can also affect the pruning with parameters such as `max_depth` and `min_samples_leaf`, the former determines the maximum depth (levels) of the tree, whereas the latter determines the minimum number of data instances required to split an internal node.

Let us use `GridSearchCV` to determine the best values of the maximum depth and minimum number of samples. We start by splitting our data into training and testing:

Remember that pandas only takes numerical values. We need to encode categorical values appropriately.

The `get_dummies` function lets us do this very easily.

We are making use of the `DecisionTreeClassifier` algorithm implemented in Scikit-learn.

```
X = titanic[['Pclass', 'Age', 'female']]
Y = titanic['Survived']

import sklearn.model_selection as ms

XTrain, XTest, YTrain, YTest =\
    ms.train_test_split(X, Y,\
        test_size= 0.3, random_state=1)
```

Let us split our data into training and testing. Notice that we are only including the column female as a False here implies a True in the male column.

We can now define the values over which our search will be performed. In this case we are going to look at the tree at different depths and a minimum number of data instances in each leaf node:

```
depth_val = np.arange(2,11)
leaf_val = np.arange(1,31, step=9)

grid_s = [{'max_depth': depth_val,\
    'min_samples_leaf': leaf_val}]
```

We will perform our parameter search over the depth of the tree and the threshold of data for node splitting.

We can now instantiate our decision tree model to use entropy as the impurity measure:

```
from sklearn import tree
model = tree.DecisionTreeClassifier(\
    criterion='entropy')
```

Finally, we instantiate our model using entropy for the impurity criterion.

It is possible now to run our search and use the best parameters found:

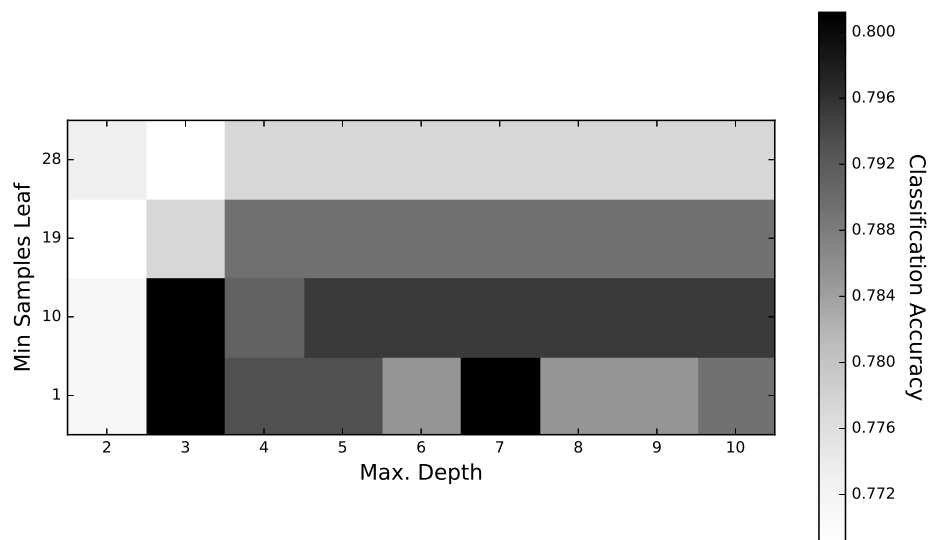


Figure 7.5: Heatmap of mean cross-validation scores for the decision tree classification of the Titanic passengers for different values of maximum depth and minimum sample leaf.

```
from sklearn.model_selection import GridSearchCV

cv_tree = GridSearchCV(estimator=model,\
    param_grid=grid_s,\
    cv=ms.KFold(n_splits=10))

cv_tree.fit(XTrain, YTrain)

best_depth = cv_tree.best_params_['max_depth']

best_min_samples = cv_tree.\
    best_params_['min_samples_leaf']
```

We are using cross-validation to find the best decision tree model for our data.

In this case the parameters turned out to be 3 for the maximum depth and 10 for the minimum number of data instances. We show a heatmap of the classification score in [Figure 7.5](#).

Let us now apply our model to the testing subset we created above:

```
model = tree.DecisionTreeClassifier(\
    criterion='entropy',\
    max_depth=best_depth,\
    min_samples_leaf=best_min_samples)

TitanicTree = model.fit(XTrain, YTrain)
survive_pred = TitanicTree.predict(XTest)
survive_proba = TitanicTree.predict_proba(XTest)
```

This is the moment of truth, we will apply our model to the testing subset and see how it performs.

The confusion matrix we obtain from our model above is as follows:

```
> from sklearn import metrics
> metrics.confusion_matrix(YTest, survive_pred)

array([[104, 22],
       [ 23, 65]])
```

We can take a look at the confusion matrix for this classifier.

The score obtained is:

```
> print(TitanicTree.score(XTest, YTest))

0.789719626168
```

As well as the overall score on the testing dataset.

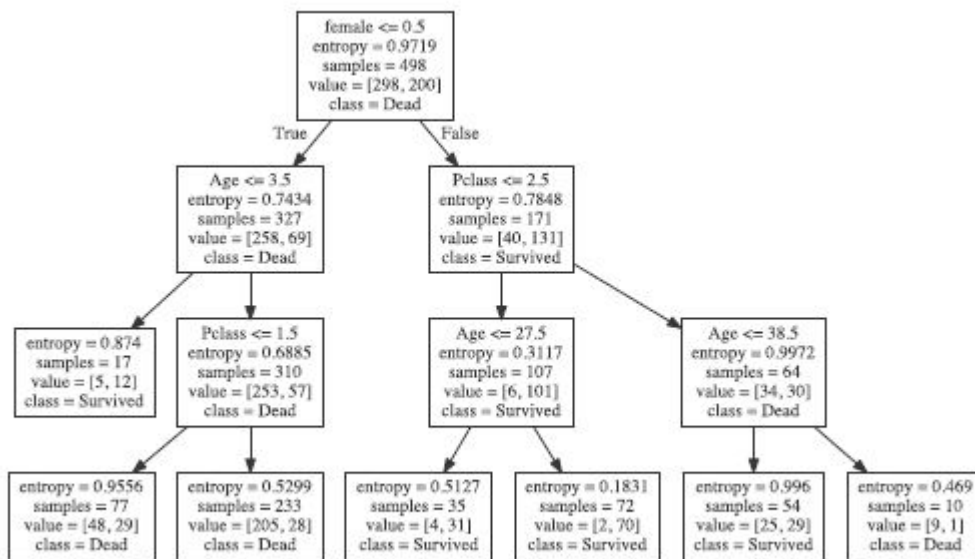


Figure 7.6: Decision tree for the Titanic passengers dataset.

Scikit-learn enables us to generate a Graphviz visualisation of the tree we have generated with the `export_graphviz` method in `tree`. We show our tree in [Figure 7.6](#).

```
tree.export_graphviz(TitanicTree,\n
    out_file='TitanicTree.dot',\n
    max_depth=3, feature_names=X.columns,\n
    class_names=['Dead', 'Survived'])
```

Graphviz is an open-source graph visualisation software. You need a separate installation to visualise the tree.

This code will generate a `.dot` file that can be visualised with Graphviz.

Finally, we can apply our model to the holdout data provided by the test dataset from the Kaggle competition. We need to pre-process our data first as we did for the training dataset:

In this case we are using the Kaggle dataset as a holdout. Our model has not seen this dataset at all yet.

```

titanic_test = pd.read_csv(u'./Data/test.csv')
titanic_test = titanic_test.drop(['Ticket',\
    'Cabin'], axis=1)
titanic_test =titanic_test.dropna()
titanic_test['Pclass'] = titanic_test['Pclass'].\
    astype('category')
titanic_test['Sex'] = titanic_test['Sex'].\
    astype('category')
titanic_test = pd.concat([titanic_test,\
    pd.get_dummies(titanic_test['Sex'])], axis=1)

```

We need to apply the same transformations we performed on the training set.

Now we finish with obtaining our predictions:

```

X_holdout = titanic_test[['Pclass', 'Age',\
    'female']]

survive_holdout = TitanicTree.predict(X_holdout)

```

Finally, we run our model on the holdout dataset.

7.3 Ensemble Techniques

MANY OF US MAY HAVE encountered in our travels the enticing competition of guessing the number of jelly beans, candy or coins in a jar. It is an easy one to grasp: All you have to do is provide your estimate of the number, leave your details and, if you get it right, you are awarded a prize together with a lifetime of newsletters to your inbox. This sort of competition has been running for quite a long time, but not many of us stop to consider the statistical implications of the estimates provided. Not unless you are Francis Galton.

Yes, the same Francis Galton of regression fame we discussed in [Chapter 4](#).

In a county fair held at Plymouth, England, Galton encountered a version of the competition above, but instead of jelly beans, the contestants had to guess the weight of a fat ox⁶. He reports that the average guess of 1,207 lbs among 800 contestants is actually rather close to the actual weight of said ox (1,198 lbs). He remarked that the middlemost estimate expresses the *vox populi*. These days we refer to it as the “wisdom of crowds”.

One single estimate of the weight of the ox (or the number of coins in the jar) may not be accurate or even close to the real value. However, when taking into account a diversity of independent opinions and aggregating them, the estimate becomes more accurate. Of course the wisdom of crowds is not infallible, but it is worth considering. In fact we do this when seeking a second or even third opinion on medical diagnostics. Furthermore, citizen science projects such as Zooniverse⁷ and Fold-it⁸ have demonstrated that the collaboration of the general public and non-professional amateurs provides an invaluable contribution to the scientific endeavour.

A similar effect has been successfully shown when combining classifiers in machine learning. We call these combinations **ensemble techniques**. In that respect, an ensemble is a set of individually trained base classifiers, such as decision trees, whose predictions are combined to determine the class of unseen data. As in the example of the ox investigated by Galton, the results obtained from an ensemble are generally more accurate than those of the individual classifiers.

⁶ Galton, F. (1907). *Vox populi*. *Nature*. 75(1949), 450–451.

Simply take a look at panicked investors in a bursting bubble market.

⁷ Zooniverse. Projects. <https://www.zooniverse.org/projects>

⁸ Fold-it. Solve puzzles for science. <https://fold.it/>

An ensemble is a set of individually trained classifiers whose predictions are combined.

The base classifiers can be any algorithm that provides us with a result that is slightly better than random guessing and hence is referred to as a **weak learner**. For a binary classifier, a weak learner will provide us with a function that correctly classifies a record with at least a probability $\frac{1}{2} + \epsilon$, with a small and positive value for ϵ . In contrast, a **strong learner** provides us with a very successful algorithm to classify our records.

Classifiers can be categorised into weak and strong learners.

Our task is to find a way to aggregate the results provided by our weak learners and obtain a strong one. Nonetheless, in order for the ensemble classifier to be strong, and therefore outperform a weak base classifier, we need the base classifiers to be accurate and show diversity in their misclassifications, in other words, their classification errors must occur on different training records. The accuracy requirement indicates the low bias of the model, whereas diversity suggests that the weak learners are uncorrelated. For these reasons, a decision tree is the epitome of a base classifier: The flexibility provided by the number of labels that we can take into account when growing our tree, for example, enables us to generate various base classifiers.

We aim to combine weak learners and get a strong one.

Base classifiers must outperform random guessing.

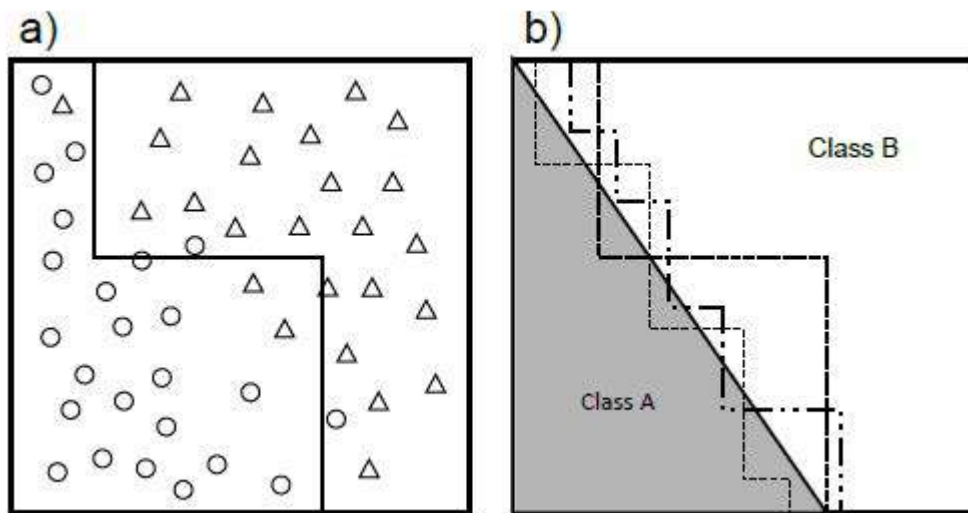
A decision tree is a classic example of a weak learner.

Consider the advantages of an ensemble classifier as a supervised learning task: We require accurate predictions of the true class of new data by learning the classifier h . If our training data is small, the base classifier on its own will find it difficult to converge to h . An ensemble classifier can help by “averaging out” the predictions of the individual base classifiers, improving convergence.

An ensemble classifier can help by averaging out the predictions of the individual base learners.

There are also advantages in terms of computation: An exhaustive search over all possible classifiers is a hard problem. In this case, even with enough data at our disposal, finding the best h is difficult. The use of an ensemble made out of several base classifiers with different starting conditions can help with our search.

This is why we used a heuristic in Section 7.2 to grow our decision trees.



Finally, a single base classifier may provide a classification function that does not adapt very well to the actual h . A good example is the canonical decision tree, whose decision boundaries are given by rectilinear partitions of the feature space, as shown in Figure 7.7.a). In this case, if the true boundary is given by a diagonal line, we would need our base classifier to provide an infinitely large number of segments. However, the ensemble classifier can provide a good approximation to the boundary, as can be seen from Figure 7.7.b).

Figure 7.7: Decision boundaries provided by a) a single decision tree, and b) several decision trees. The combination of the boundaries in b) can provide a better approximation to the true diagonal boundary.

A single base classifier may not provide a good approximation to the actual decision boundary. Instead, an ensemble can provide a better approximation.

The idea of aggregating base classifiers is quite straightforward. Given an initial training dataset, we need to build a set of base classifiers. Each of these n classifiers needs to be trained with appropriate data. A way to do this is shown diagrammatically in Figure 7.8, where the initial training data is used to create multiple datasets according to a given sampling distribution. This distribution determines how likely we are to select a particular record for training. Each of these datasets is used to train weak learners to predict the class of unseen data, and finally the models are aggregated.

Our training data is used to create multiple datasets to train the base classifiers.

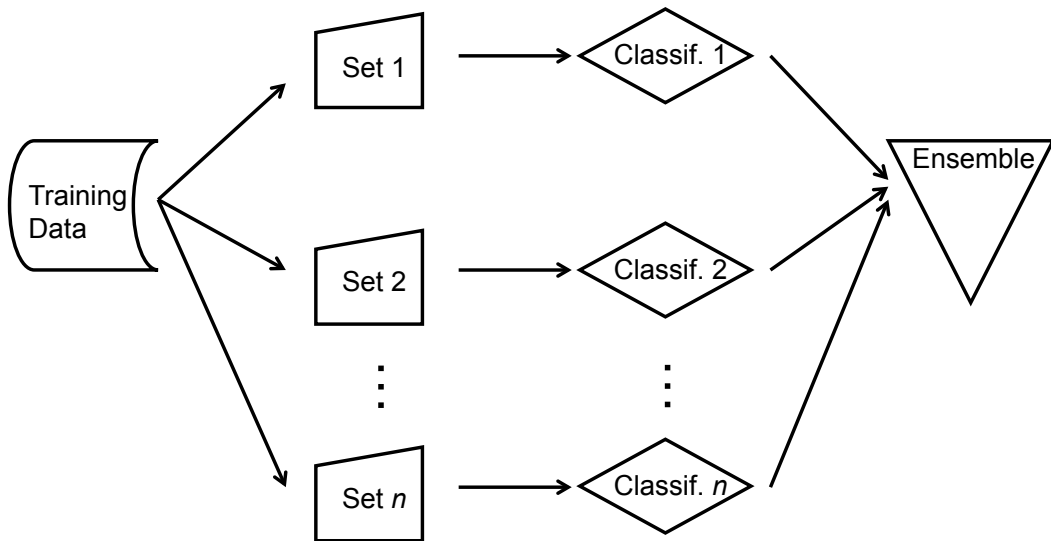


Figure 7.8: Diagrammatic view of the idea of constructing an ensemble classifier.

The ensemble classifier can be constructed by applying different mechanisms. For instance, we can manipulate a) the training set, b) the class labels, or c) the learning algorithm. We shall talk about these mechanisms in more detail. First, let us get a clearer picture of how ensemble methods achieve better results than a single base classifier.

There are different ways to construct our ensemble.

Let us consider a binary classification task with labels A and B . We have a total of 10 new data records whose true label is A for all of them. We build three classifiers whose accuracy is 60%. We obtain the predicted labels returned by each of the three classifiers and take a majority vote of the labels to construct our ensemble. An instance of results is given in [Table 7.2](#).

Classifier	Predicted Class									
1	A	B	A	A	B	B	A	B	A	A
2	B	A	A	A	A	A	B	A	B	B
3	A	B	A	B	A	B	A	B	A	A
Ensemble	A	B	A	A	A	B	A	B	A	A

Table 7.2: Predicted classes of three hypothetical binary base classifiers and the ensemble generated by majority voting.

In the particular example we show in [Table 7.2](#), the majority vote has boosted the accuracy to 70%. All in all, for a majority vote with the three classifiers above we expect the following outcomes:

- All three classifiers are wrong: $(1 - 0.6)^3 = 0.4^3 = 0.064$
- Two classifiers are wrong: $3(0.6)(0.4)^2 = 0.288$
- Two classifiers are correct $3(0.6)^2(0.4) = 0.432$
- All three classifiers are correct: $0.6^3 = 0.216$

Expected outcomes of the ensemble considered in [Table 7.2](#).

As we can see, in about 43% of the cases the majority vote provides the correct label. From the figures above, the ensemble will be correct about 64% of the time ($43\% + 21\%$), performing better than the 60% of a single base classifier. Using a larger number of classifiers will provide a better classification.

Using a larger number of base classifiers provides a better result.

We mentioned that correlation among the classifiers has an impact on the ensemble. Let us show some empirical examples: If all of our three classifiers have correlated outcomes we may see no improvement.

Classifiers with correlated outcomes, may show no improvement.

Consider the outcomes shown in [Table 7.3](#) where classifiers 2 and 3 have an accuracy of 70% and are highly correlated in their predictions. The ensemble shows no real improvement when taking the majority vote (70%).

Classifier	Predicted Class									
1 (Accuracy 80%)	A	A	A	A	A	B	A	B	A	A
2 (Accuracy 70%)	B	A	A	A	A	A	B	A	A	B
3 (Accuracy 70%)	B	A	A	A	A	A	B	A	A	B
Ensemble	B	A	A	A	A	A	B	A	A	B

Table 7.3: Predicted classes of three hypothetical binary base classifiers with high correlation in their predictions.

Let us now compare the result above with that of three classifiers that have very different performances and with uncorrelated results as shown in [Table 7.4](#). As we can see, the ensemble with a majority vote achieves in this case an accuracy of 90%.

In contrast, uncorrelated outcomes in the classifiers may show an improved accuracy.

Classifier	Predicted Class									
1 (Accuracy 80%)	A	A	A	A	A	B	A	B	A	A
2 (Accuracy 60%)	A	B	A	B	B	A	B	A	A	A
3 (Accuracy 70%)	B	A	A	A	A	A	B	A	A	B
Ensemble	A	A	A	A	A	A	B	A	A	A

Table 7.4: Predicted classes of three hypothetical binary base classifiers with low correlation in their predictions.

In the examples above we have used a simple majority voting to decide the outcome for the ensemble. In the

following sections we discuss some of the different ways in which we can construct our ensemble classifier.

7.3.1 *Bagging*

BAGGING STANDS FOR BOOTSTRAP AGGREGATION, and it is an ensemble technique that involves the manipulation of the training dataset by resampling. Given an initial training dataset with N records, in bagging we create multiple training datasets of size N , by sampling uniformly with replacement. This means that some records may be picked more than one time during the process, and some may not even appear at all.

We build classifiers on each bootstrap sample and take a majority vote across the classifiers. Since some of the newly created datasets may contain repeated instances of the data, as well as missing some records altogether, we end up with a situation where some classifiers will have an error rate higher than a classifier that uses all the raw data.

Nonetheless, as we have seen in the example shown in [Table 7.4](#), when combining these classifiers, we may have a result with an accuracy that is better than that of a single classifier on its own. Breiman has shown⁹ that bagging is an effective methodology on learning algorithms where small changes in the training set have large impacts in the resulting predictions. These algorithms are called “unstable”, and some typical examples include decision trees or neural networks.

Bagging stands for bootstrap aggregation.

Note that in bagging, some records may appear more than once in the engineered datasets.

Similarly, some data points may be missing altogether, giving rise to a collection of weak learners.

⁹ Breiman, L. (1996). Bagging predictors. *Machine Learning*. 24(2), 123–140.

The increased accuracy from bagging comes from the reduction in the variance of the individual classifier, improving therefore our generalisation error. If the classifiers are stable, the error incurred comes mainly from their bias, and bagging may not be effective. Since we are resampling the data with replacement, bagging does not focus on particular data instances of the training data. This means that we are less prone to overfitting.

The increased accuracy from bagging comes from a reduction in the variance of the single classifier.

7.3.2 *Boosting*

FROM THE METHODOLOGY USED IN bagging we now know that specific data records are not given any preferential treatment. However, it is worth considering changes that we can implement if we wanted to focus our attention on specific particular records in the training dataset. A possibility is to change the sampling distribution of the training records. Boosting is an iterative methodology that adapts the sampling of the data in order to concentrate on records that have been misclassified in previous iterations.

Boosting is an iterative method that adapts data sampling to focus on misclassified records.

In the initial iteration we start with a uniform distribution assigning equal weights to our N records. At the end of the first round, we change the weights to emphasise those records that were misclassified. In that way, boosting produces a series of classifiers whose inputs are chosen based on the performance of the previous family of classifiers in the series. The final prediction of the series is computed by a weighted vote, depending on the training errors of the individual base classifiers.

We start in the same way we did for bagging.

Boosting produces a series of classifiers with inputs based on the outcome of the previous family of classifiers.

Boosting is aimed at building, with each iteration, classifiers that are better at predicting the label of a data instance than the classifiers in the previous iteration. The sampling is done with replacement, and we may encounter situations where some particular records do not show in a given training subset. This is not a problem, as these overlooked records are more likely to be misclassified and as a result they will be given a higher weight in later iterations, forcing the ensemble to correct for these mistakes. We can see that on each iteration, the base classifiers will concentrate on records that are harder and harder to classify, facing more difficult learning tasks as the task progresses.

We use sampling with replacement.

Overlooked records are more likely to be misclassified. This is corrected in later iterations, as the weight of these instances will be higher.

A popular implementation of boosting is the Ada-Boost algorithm¹⁰, or adaptive boosting. It uses either the approach of selecting a set of data points based on the probability of the instances themselves, or by using all the data instances and weighting the error of each instance by its own probability. Ada-Boost is a fast algorithm with no parameters to tune, except for the number of iterations. Nonetheless, if the base classifiers are complex, Ada-Boost may lead to overfitting. Remember that Ada-Boost may be susceptible to noise.

¹⁰ Freund, Y. and R. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comp. and Sys. Sciences.* 55(1), 119–139.

7.3.3 *Random Forests*

A SINGLE TREE PROVIDES A good shade, but the canopy of a group of trees is difficult to beat for shade. If we grow a group of various decision trees as our base classifiers and enable their growth to use a random effect, we end up with

Difficult to beat a group of trees for shade.

a *random forest*¹¹. We have seen that bagging produces better results than a single base classifier. If the base classifier is for instance a decision tree, the various trees we generate with bagging have different predictions because they use different training sets.

If we were to introduce changes in the way our trees are grown, for example by randomly selecting not only the data instances that are included in each subset, but also by randomly selecting the features to use, we can get base classifiers that are uncorrelated with each other. In effect, random forests add a layer of randomness to bagging with decision trees: On top of growing each tree using each bootstrap sample, in a random forest each node of the tree is split among a subset of randomly chosen features. The result is a classifier that performs very well and is robust to overfitting. A random forest algorithm is very straightforward to understand, as it only has two parameters: 1) the number of features in the random subset at each node, and 2) the number of trees in the forest. The simplest forest we can grow consists of selecting at each node a small number of features to perform the split. We then simply allow the trees to grow to maximum size without pruning. This methodology is known as *Forest-RI* or Random Input selection.

In cases where there is a small number of features, it is possible to create new features by taking random linear combinations of the existing ones. These new features are then used to grow the trees. This process is called *Forest-RC* or Random Combinations. If instead we have a large

¹¹ Breiman, L. (2001). Random forests. *Machine Learning*. 45(1), 5–32.

Not only do we randomly select data instances, but also the features used to grow our trees.

The result is a robust classifier.

We have two parameters in a random forest:

- Number of features at each node
- Number of trees in the forest

We can grow the trees by random input selection or by random feature combinations.

number of predictors, we have to bear in mind that the available feature set will be very different from one node to the next.

We can go one step forward in our inclusion of randomness into the growth of our trees, and instead of simply picking the best split among the chosen features at a node, the thresholds for the split are selected at random for each chosen feature. The best of these randomly generated thresholds becomes the rule for splitting. These types of random forests are known as *extremely randomised trees*¹². As with any of the other ensemble techniques we have outlined, the greater the correlation among the trees we grow, the greater the random forest error rate. We would therefore prefer trees that are uncorrelated among themselves. Running a random forest algorithm is fast, having the advantage of being robust against unbalanced or even missing data. However, with datasets that are particularly noisy, they tend to overfit.

¹² Geurts, P., D. Ernst, and L. Wehenkel (2006). Extremely randomized trees. *Machine Learning*. 63, 3–42.

Remember that we prefer uncorrelated base classifiers.

7.3.4 Stacking and Blending

APART FROM THE ENSEMBLE TECHNIQUES discussed above, there are other methodologies that can be explored. A way to combine multiple base classifiers of different kind, known as *stacked generalisation*, has been proposed by Wolpert.¹³ Some of the steps applied in stacked generalisation are the same as those in cross-validation: For a 2-fold case, we need to split our training data into two disjoint parts. We first train the base classifiers on the first part, and test them on

¹³ Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*. 5(2), 241–259.

See our discussion of cross-validation in [Section 3.13](#).

the second one. We use the predictions from the last step as inputs for training a higher level learner.

An alternative name for stacking is *blending*, and the word has been made popular by the winners of the famous Netflix Prize Competition to improve the accuracy of recommendations provided by Netflix, based on customer film preferences¹⁴ (see [Section 8.4](#) for more information). Sometimes a distinction between the two is introduced: In blending we create a small holdout set out of the training set, and the stacker model is trained on the holdout set. In general, blending is a simpler methodology than stacking, but less data is actually used. Please note that the stacker may overfit to the holdout set.

Blending is another alternative.

¹⁴ Töschner, A. and M. Jahrer (2009). The BigChaos solution to the Netflix grand prize. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf

7.4 Ensemble Techniques in Action

LET US NOW TURN OUR attention to the Scikit-learn implementation of the various ensemble techniques we have covered. We will continue working with the Titanic dataset we used in [Section 7.1.1](#).

We will continue using the Titanic dataset.

Let us apply the same pre-processing as before but add more variables including the port of embarkation. This feature will need to be transformed in order to obtain dummy variables to encode the three categories:

We apply the same pre-processing as before.

- Cherbourg: C
- Queenstown: Q
- Southampton: S

These are the three levels for the port of embarkation.

For completeness, let us detail the steps again:

```
titanic = pd.read_csv(u'./Data/train.csv')
titanic = titanic.drop(['Ticket', 'Cabin'], axis=1)
titanic = titanic.dropna()
catcols = ['Pclass', 'Survived', 'Sex', 'Embarked']
titanic[catcols]=titanic[catcols].\
    astype('category')

titanic = pd.concat([titanic,\
    pd.get_dummies(titanic['Sex'])], axis=1)

titanic = pd.concat([titanic,\
    pd.get_dummies(titanic['Embarked'])], axis=1)
```

These steps are similar to those applied in [Section 7.2.1](#).

We need to encode the port of embarkation.

Let us split our dataset into training and testing. We will use the former to train various models, and the latter for evaluation:

```
import sklearn.model_selection as ms

X = titanic[['Pclass', 'Age', 'female', 'SibSp', \
    'Parch', 'Fare', 'S', 'C', 'Q']]
Y = titanic['Survived']

XTrain, XTest, YTrain, YTest =\
    ms.train_test_split(X, Y,\
        test_size= 0.2, random_state=42)
```

We construct a matrix with the required features.

We then create our training and testing datasets.

We are interested in employing a variety of ensemble methods implemented in Scikit-learn, namely:

- Bagging: `BaggingClassifier()`
- Boosting: `AdaBoostClassifier()`
- Random Forests: `RandomForestClassifier()`
- Extremely Randomised Trees: `ExtraTreesClassifier()`

These ensemble techniques in Scikit-learn accept a parameter called `base_estimator` that determines the base classifier to be used in modelling. By default, the base classifier is taken to be a decision tree, such as `DecisionTreeClassifier`, which we have discussed in [Section 7.2](#). They also take a parameter that determines the total number of base classifiers to be used, namely `n_estimators`.

The parameter `base_estimator` determines the base classifier.

The number of base classifiers to use is given by `n_estimators`.

Let us import the relevant modules we will use:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (AdaBoostClassifier,\
    BaggingClassifier,\
    RandomForestClassifier,\
    ExtraTreesClassifier)
```

We will use a decision tree as our base classifier.

We can write a script that considers each of the ensemble methods above, as well as a standard decision tree to train a variety of models. Let us prepare the ground for the script by building objects to hold relevant information. For example, we define the number of base classifiers as follows:

We will use implementations for Ada-Boost, Bagging, Random Forests and Extremely Randomised Trees.


```
n_est = 100
```

We will use 100 trees in our ensembles.

Let us now define a list with the different models we are going to train:

```
models = [DecisionTreeClassifier(max_depth=3),\
          BaggingClassifier(n_estimators=n_est),\
          RandomForestClassifier(n_est=n_estimators),\
          ExtraTreesClassifier(n_estimators=n_est),\
          AdaBoostClassifier(n_estimators=n_est)]
```

Then we create a list of models to be trained.

Let us also create a list with the names of the models we are going to use:

```
model_title = ['DecisionTree', 'Bagging',\
               'RandomForest', 'ExtraTrees', 'AdaBoost']
```

This list will provide an easy way to identify each of the models.

We also need a suitable structure to keep track of the predictions, probabilities and scores, as well as the true and false positive rates and thresholds:

```
surv_preds, surv_probs, scores,\
fprs, tprs, thres = ([] for i in range(6))
```

We are initialising six empty lists that will get populated next.

We can now traverse the `models` list, and fit each of our models with the `XTrain` and `YTrain` arrays. We can then proceed to obtain predictions and probabilities using `XTest` and `YTest`:

```

from sklearn.metrics import roc_curve
for i, model in enumerate(models):
    print('Fitting {0}'.format(model_title[i]))

    clf = model.fit(XTrain, YTrain)
    surv_preds.append(model.predict(XTest))
    surv_probs.append(model.predict_proba(XTest))
    scores.append(model.score(XTest, YTest))

    fpr, tpr, thresholds = roc_curve(YTest, \
    surv_probs[i][:,1])
    fprs.append(fpr)
    tprs.append(tpr)
    thres.append(thresholds)

```

Let us print the name of the model being trained.

We proceed to train the model and calculate the predictions.

Finally, we calculate the true and false positive rates and thresholds.

In this case, the scores that we have obtained are as follows:

```

> for i, score in enumerate(scores):
    print('{0} with score {1:0.2f}'.\
    format(model_title[i], score))

```

```

DecisionTree with score 0.75
Bagging with score 0.73
RandomForest with score 0.77
ExtraTrees with score 0.78
AdaBoost with score 0.81

```

We can now print the scores obtained when training each of the models in question.

Notice that we have not performed any *k*-folding in our cross-validation, and you are encouraged to do so. For the

time being, we can obtain a wider view of the performance of these models by looking at the ROC.

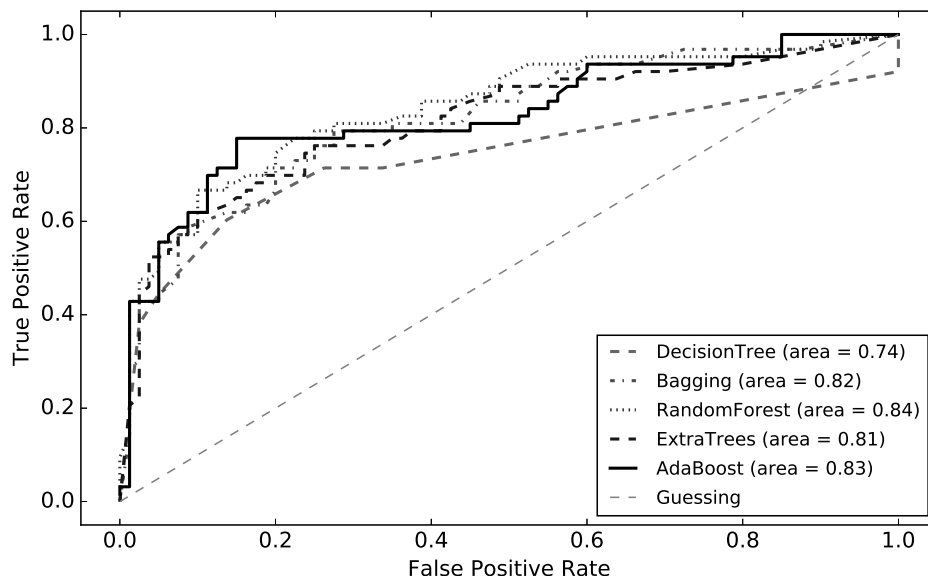


Figure 7.9: ROC curves and their corresponding AUC scores for various ensemble techniques applied to the Titanic training dataset.

In [Figure 7.9](#) we have the ROC curves, and their corresponding AUC scores, obtained with the various ensemble models used. We can see that all of the models used perform much better than random guessing (diagonal dotted line). For the simple splitting we have made, the ROC curves indicate that the ensemble models also perform better than the decision tree. Ada-Boost (AUC = 0.83) and Random Forest (AUC=0.82) are seen to be better at this classification task than the rest.

Remember that the decision tree is our base classifier.

7.5 *Explain Yourself: Explainable AI*

WE MENTIONED IN [SECTION 7.2](#) that the output of a decision tree is easy to understand and can be used to explain how a decision was made. As we know, explainability is an important component in implementing and trusting machine learning models. The mystery shrouding Artificial Intelligence (AI) is one of the first hurdles the industry and regulators are interested in overcoming to mitigate the risks linked with this technology.

Explainability aids in improving trust in AI systems.

Explainable AI is a topic of great interest covering accountability, ethics, and fairness. For example, the European Commission for example has issued the AI Act¹⁵ trying to ensure better conditions for the development and use of AI with transparency, lawfulness, and fairness. As any Jackalope Data Scientist can imagine, explainability is an important component here.

¹⁵ European Commission (2021, Apr). Proposal for a Regulation laying down harmonised rules on Artificial Intelligence. <https://digital-strategy.ec.europa.eu/en/library/proposal-regulation-laying-down-harmonised-rules-artificial-intelligence>

Explainable AI is all about adding a sprinkle of clarity to the black-box world of AI and Machine Learning (ML). It involves a set of processes and methods designed to ensure that the decisions made by these models are as clear and understandable to humans as possible.

This helps opening up the AI black-box.

By weaving an explainability layer into these models, Jackalope Data Scientists can craft more trustworthy and transparent systems, which are invaluable for a wide range of stakeholders, from developers and regulators to everyday users.

In order to build trust in AI and ML models through explainability we should take into account some guiding principles, including:

- **Transparency:** Making sure stakeholders, including end-users, can grasp the decision-making process of the models
- **Fairness:** Ensuring the models' decisions are fair to everyone, including those in protected groups (race, religion, gender, disability, ethnicity)
- **Trust:** Evaluating the confidence level of human users interacting with the AI system
- **Robustness:** Maintaining consistent and reliable performance despite changes in input data or model parameters, and thriving even in uncertain or unexpected scenarios
- **Privacy:** Protecting sensitive user information with utmost care
- **Interpretability:** Offering explanations for predictions and outcomes that humans can easily understand

Some of the guiding principles we should consider when implementing AI and ML models.

Let us see one useful technique to help us with explainability in our models.

7.5.1 SHAP — SHapley Additive exPlanations

IN MANY APPLICATIONS, UNDERSTANDING WHY a model makes a certain prediction can be just as important as the prediction's accuracy. However, achieving top-notch

accuracy for large modern datasets often requires complex models—like ensemble or deep learning models¹⁶—that even experts find hard to decipher. This creates a tug-of-war between accuracy and interpretability.

An important method to support with this task is SHAP (SHapley Additive exPlanations)¹⁷ which can be seen as a unified framework for interpreting predictions. SHAP assigns each feature an importance value for a particular prediction. SHAP values help us quantify how much each feature contributes to a prediction. When a SHAP value is close to zero, it means the feature has little impact on the prediction. Conversely, a SHAP value further from zero indicates that the feature has a more significant influence on the outcome.

7.5.2 SHAP in Action

LET US TAKE A LOOK at using SHAP to explain the decision tree we built in [Section 7.2.1](#) for the Titanic dataset. We built a decision tree model using three variables, namely, Passenger Class (Pclass), Passenger Age (Age) and Passenger Gender (female). The model has a maximum depth of 3, and we required a minimum of 10 data instances per node and the model has the name TitanicTree.

We can call the SHAP library as follows:The

```
import shap
shap.initjs()
```

¹⁶ Rogel-Salazar, J. (2020). *Advanced Data Science and Analytics with Python*. Chapman & Hall/CRC data mining & knowledge discovery series. CRC Press, Taylor & Francis Group.

¹⁷ Lundberg, S. M. and S.-I. Lee (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, Volume 30. Curran Associates, Inc.

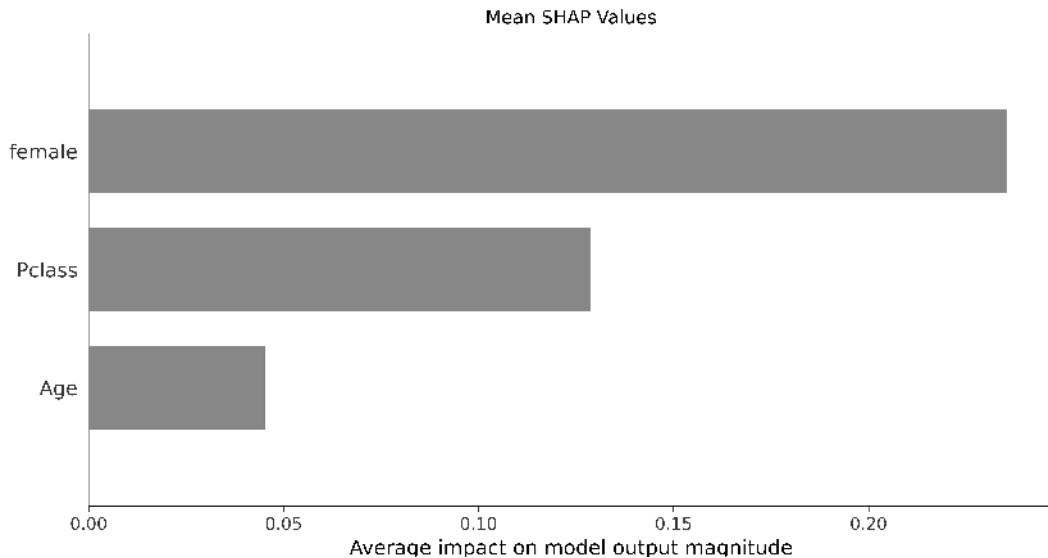
Please refer to [Section 7.2.1](#) for more information on building the decision tree model.

SHAP requires JavaScript.

The last line above initialised the JavaScript code required to display SHAP visualisations. We are going to instantiate a `TreeExplainer` class with the decision tree classifier from [Section 7.2.1](#). This will enable us to compute the SHAP values for each feature of each instance in the test dataset.

```
explainer = shap.TreeExplainer(TitanicTree)
shap_values = explainer.shap_values(XTrain)
```

We instantiate a `TreeExplainer` model and obtain the SHAP values.



SHAP enables us to visualise the values in various ways. Here we will concentrate on two: A summary plot giving us the feature importance, and a summary plot for a specific target. Let us start with the first one. Let us look at the surviving passengers:

Figure 7.10: Mean SHAP values providing us with a view of the average impact of each feature on the model. We can see how the gender of the passenger has a greater impact on the decision, compared to the travelling class and the age.

```
features = X.columns.tolist()
shap.summary_plot(shap_values[:, :, 1],
                  plot_type='bar', features=features)
```

Obtaining a plot for the feature importance for the survivor class.

The output can be seen in [Figure 7.10](#). The features are ordered in descending order of importance. We can see that the gender of the passenger has a greater impact on the decision made by the tree, compared to the class of travel and the age, with the latter having the least impact.

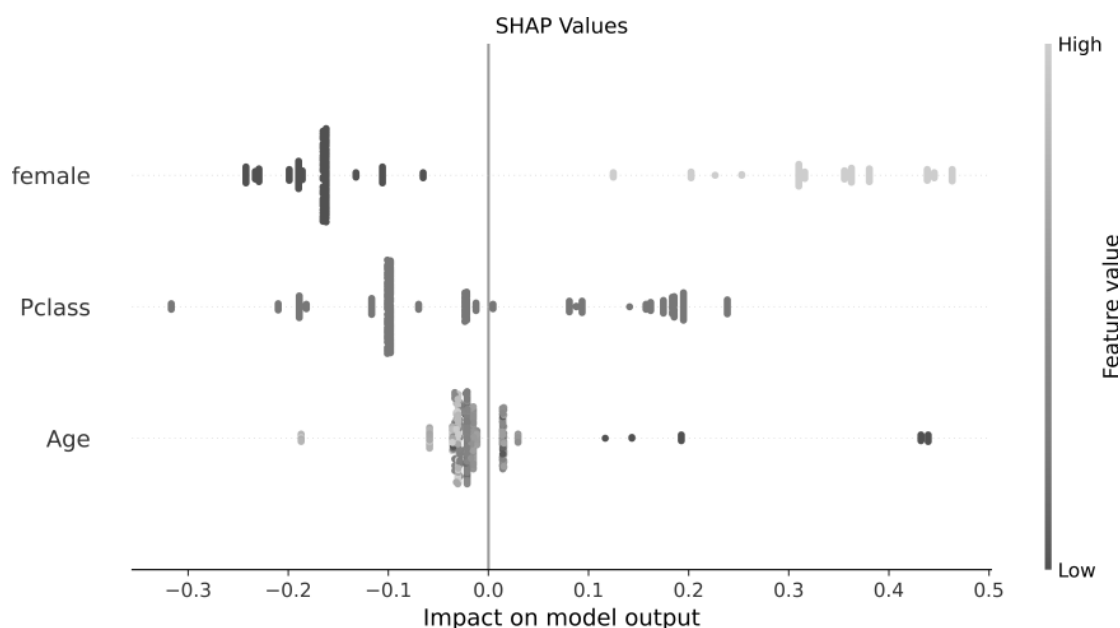


Figure 7.11: Impact on the outcome of the model. Each dot represents a record in the dataset..

We can also look at a more granular view of the impact of each feature for a specific outcome. Let us continue looking at the surviving passengers, i.e., `shap_values[:, :, 1]`:


```
shap.summary_plot(shap_values[:, :, 1], XTrain)
```

The outcome of the code above can be seen in [Figure 7.11](#). As before, the features are ordered by their impact and the plot may require a bit more explanation.

The horizontal axis in [Figure 7.11](#) shows the distribution of the SHAP values for each feature. The plot shows every data instance per feature in the dataset, represented by each dot. For example, we can see that for the female feature the dots are more spread, whereas they cluster closer to 0 for the Age feature. Positive values for a given feature nudge the model's prediction closer to the target label (label= 1). On the other hand, negative values steer the prediction toward the opposite class (label= 0).

The colour of the dots represents the feature values, adding an extra dimension that lets us observe how SHAP values change with varying feature values. This allows us to discern the direction of the relationship. For example, we can see that SHAP values tend to be lower (indicated by the colour bar on the side) when Age is higher.

SHAP values enhance model explainability by providing a clear and consistent measure of each feature's contribution to a prediction. They quantify the impact of individual features, allowing us to see whether a feature pushes the prediction towards a positive or negative outcome and by how much. This granular insight helps in understanding the model's decision-making process, making it more transparent. By visualising these contributions, stakeholders

The summary plot for a specific target shows every data instance in our dataset.

The colour of the dots represents the feature values.

SHAP values enhance model explainability.

These visualisations can help us understand the outcomes of our models.

can better grasp why a model made a certain prediction, identify potential biases, and ensure that the model aligns with ethical standards and fairness, thereby building greater trust and accountability in AI systems.

7.6 *Summary*

THE INSPIRATION THAT TREES HAVE provided in the endeavour of organising and visualising information is undeniable. Their appeal is palpable in different disciplines as relevant communication tools, depicting information about biological species, family relationships, database schemas or computational algorithms. In this chapter we have seen how trees have also influenced the data science arena with algorithms in both supervised and unsupervised learning thanks to the hierarchical organisation that a tree provides.

In regards to unsupervised learning, we considered hierarchical clustering. In particular we concentrated on the description of agglomerative clustering, where we build a hierarchy from the bottom-up. Starting with data instances as their own cluster, we successively merge similar clusters as we go up in the levels of the tree. A way to visualise the result of hierarchical clustering is with the aid of a dendrogram. A dendrogram is therefore a tree-like structure that lets us visualise the clusters obtained with the algorithm. The length of the clades or branches of the dendrogram plot are related to the similarity between clusters.

We also covered the supervised learning task of classification with the help of decision trees. Decision trees are well-known tools in areas such as operations research or decision analysis. Their diagrammatic representation in terms of rules that guide us through the structure is readily understood, even without knowing how the tree itself was constructed. We discussed a heuristic to grow decision trees without having to construct rules for all the different combinations of features in our dataset. The key is the measure of purity before and after data is split by the application of a rule. We considered the application of entropy, Gini impurity and classification error as measures of purity.

We appealed to the wisdom of base classifiers to improve on our predictions. Ensembles of these base classifiers result in more accurate results than those obtained with a base classifier on its own. These base classifiers, or weak learners, are required to be more accurate than random guessing and exhibit diversity in their misclassifications, as well as being uncorrelated with each other. Ensemble classifiers can be constructed by manipulating a) the training dataset, b) the class labels, or c) the learning algorithm. We finished this chapter with details about ensemble techniques such as bagging, boosting and random forests.

Finally, we saw how SHAP values enhance model explainability by quantifying the impact of individual features on predictions, providing clear insights into the model's decision-making process. This transparency helps stakeholders understand, trust, and ensure the ethical

alignment of AI systems. Furthermore, these insights can guide Jackalope Data Scientists in refining the model to address potential biases and improve overall performance.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

8

Less is More: Dimensionality Reduction

THE DRAMATIC INCREASE IN DATA we have seen in recent times not only encompasses an explosion in the number of records, but also in the number of features that describe our datasets. In [Chapter 1](#) we alluded to the fact that the richness of the data available to businesses and researchers provides us with opportunities to make better decisions based on the data itself. Nonetheless, sometimes the number of available features can be overwhelming, to the point that it is difficult to determine what attributes of a dataset are the most important.

We have already discussed the idea behind feature selection as a useful tool under our Jackalope Data Scientist belt, where less is truly more: A careful selection of the features to be included in our models has a large effect on the outcomes of many machine learning algorithms, and in some cases may even help in our understanding of the results obtained. We also saw in [Section 4.9](#) how regularisation techniques, such as LASSO, provide a way to

Learn from Mr Creosote: Even if it is wafer thin, an extra mint can be far too much.

See [Section 3.7](#) for a discussion on feature selection.

Careful selection of features may have a large effect on our results.

perform feature selection in an automatic way, directly as a result of suitable regularisation measures.

Remember that LASSO allows for regression coefficients to be shrunk down to zero.

In this chapter we will discuss Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) as ways of reducing the number of features in our dataset. We do this by generating combinations of the attributes in the dataset with the purpose of projecting them into a lower dimensional space while maintaining as much information as possible.

8.1 Dimensionality Reduction

IN THE ALGORITHMS AND APPLICATIONS we have been discussing, we organise our data in terms of matrices (large and small). This representation enables us to use linear algebra to carry out matrix operations and use compact notations to express our models. For instance, in [Section 3.11](#) we saw how Scikit-learn expects data to be represented by numeric matrices with M instances (row-by-row) and N features in the columns.

We have been representing our data in terms of matrices.

We can think of the features in our data as the different dimensions along which our data instances are specified. We are also aware of the so-called *curse of dimensionality*, where the increase in the number of dimensions requires us to have a much larger number of instances in our datasets. A way to deal effectively with this problem is by considering the features that we are including in our models. A careful selection of these features goes a long way, and sometimes appropriate transformations have the advantage

See [Section 3.10](#) for a discussion about the curse of dimensionality.

Dimensionality reduction is in a way a form of feature extraction.

of reducing the number of features while conveying as much useful information as possible. Dimensionality reduction is thus a form of feature extraction.

In that respect, the process of dimensionality reduction involves taking datasets represented in large matrices and finding “narrower” ones that are close to the original matrix. These narrower matrices have a smaller number of columns and rows, and in a way they are easier to manipulate than the large matrix we started with. Having fewer dimensions in our problem may improve generalisation, we gain some speed in the running of algorithms and use less storage to hold the data. In this respect, a good example to bear in mind is data compression. For instance, in image compression we are interested in reducing the size of the data while still being able to tell what is depicted. We need to balance file size with the resolution we want to keep.

The process of finding the “narrower” matrix we are interested in involves the decomposition of our original large matrix into simpler meaningful pieces. In many cases, the process involves the calculation of the eigenvalues and eigenvectors of the original matrix, i.e., eigenvalue decomposition. It is worth exploring the importance of the process of eigenvalue decomposition and its meaning.

The mathematician David Hilbert¹ was the first person to use the word *eigen* in this context. The word comes from the German language and is a prefix that can be translated as “proper”, “distinct”, “own” or “particular”. Given an $n \times n$ square matrix **A**, and a column vector **x** with n non-zero

Narrower in the sense that having less features leads to less columns in our matrices.

Improved generalisation leads to less chance of overfitting.

Image compression is a good example of dimensionality reduction.

Matrix decomposition is an important step in dimensionality reduction.

¹ Hilbert, D. (1904). Grundzüge einer allgemeinen Theorie der linearen Integralrechnungen. (Erste Mitteilung). *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse.*, 49–91.

elements we can carry out the matrix multiplication $\mathbf{Ax}$. We can ask the following question: Is there a number λ such that the multiplication $\lambda\mathbf{x}$ gives us the same result as $\mathbf{Ax}$? In other words:

$$\mathbf{Ax} = \lambda\mathbf{x}. \quad (8.1)$$

If such a number λ exists, then we say that it is an eigenvalue of the matrix $\mathbf{A}$, and $\mathbf{x}$ is one of its eigenvectors. The interest in these quantities comes into place when we start pondering the application of linear transformations of the matrix $\mathbf{A}$.

In this context λ is the eigenvalue of the matrix $\mathbf{A}$.

A linear transformation is a function between two vector spaces that preserves the operations of addition and scalar multiplication. In simpler terms, a linear transformation takes, for example, straight lines into straight lines or to a single point, and they can be used to elucidate how to stretch or rotate an object, for example.

We can think of linear transformation as ways to stretch or rotate an object.

The use of eigenvalues and eigenvectors comes into place as they make linear transformations easier to understand. Eigenvectors can be seen as the “directions” along which a linear transformation stretches (or compresses), or flips an object, whereas eigenvalues are effectively the factors by which such changes occur. In that way, eigenvalues characterise important properties of linear transformations, for example whether a system of linear equations has a unique solution, and they can also describe the properties of a mathematical model.

Eigenvectors can be seen as the direction along which stretching or flipping is applied.

Notice that Equation (8.1) implies that we can recover the result of the matrix multiplication by multiplying the

eigenvalues with the appropriate eigenvectors. Once we have decomposed our matrix, the pieces can then be used in the modelling steps of other algorithms. In the case of dimensionality reduction, the decomposition is usually applied as a pre-processing step to get a better idea of the data we are dealing with, and typical examples are in effect unsupervised learning tasks.

The learning objective of dimensionality reduction is the use of data in the most meaningful basis possible. This can be achieved by choosing a subset of features and create new ones out of them. In an unsupervised manner, we consider only the data points themselves and not their labels (if they exist). We look therefore to remove data that is not informative, enabling us to get better generalisation and model performance on new data.

For a dataset represented by a matrix $\mathbf{B}$ with M records and N features, we are interested in finding a d -dimensional representation of $\mathbf{B}$, with ($d \ll N$) that encapsulates the information in the original dataset based on particular criteria. Linear dimensionality reduction is based on the concept of performing a linear projection of the data as:

$$\mathbf{b} = \mathbf{U}^T \mathbf{B}, \quad (8.2)$$

where $\mathbf{b}$ has dimensionality d and is a projection of the original matrix $\mathbf{B}$. The projection is obtained with the use of the $N \times d$ matrix $\mathbf{U}$ that defines a d -dimensional linear subspace. We end up with different dimensionality reduction methods depending on the definition of $\mathbf{U}$.

This is a much easier operation as eigenvalues are scalars, i.e., numbers.

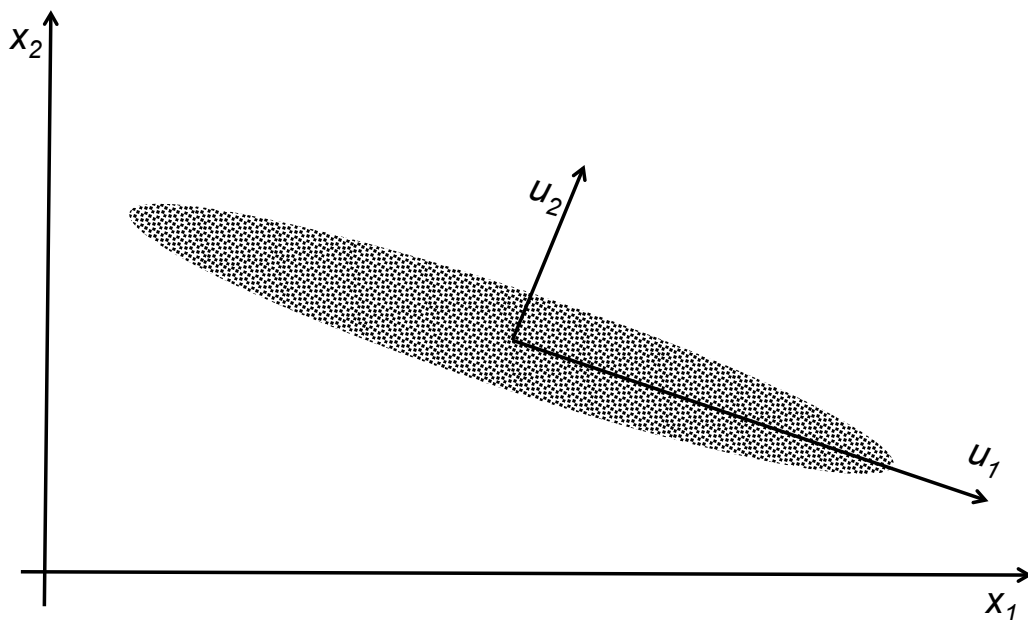
Dimensionality reduction is a pre-processing step.

Think about this as the more meaningful coordinates if you will.

Dimensionality reduction is an unsupervised task.

We are interested in finding a smaller representation of a matrix while keeping as much information as possible.

Dimensionality reduction can be seen as a projection of the original matrix.



Before we enter into further details, let us provide an illustration of the usage of dimensionality reduction. For simplicity, let us consider data in a two-dimensional space with features $\{x_1, x_2\}$ as shown in [Figure 8.1](#). If we were to carry out feature selection and concentrate only on feature x_1 , we naïvely would have indeed reduced the dimensionality of the dataset. However, we would need to ask ourselves whether we have lost meaningful information by disregarding x_2 .

In view of the representation given in [Figure 8.1](#), we can see that the data has substantial variance along both features x_1 and x_2 . Concentrating only on x_1 has the effect of a large loss of information.

Figure 8.1: Simple illustration of data dimensionality reduction. Extracting features $\{u_1, u_2\}$ from the original set $\{x_1, x_2\}$ enables us to represent our data more efficiently.

Here we are naïvely reducing the dimensionality from 2D to 1D.

Let us now consider a coordinate transformation into the features $\{u_1, u_2\}$ shown in [Figure 8.1](#). These new features are given as a linear combination of the original set $\{x_1, x_2\}$ such that:

$$u_1 = a_1x_1 + a_2x_2, \quad (8.3)$$

$$u_2 = b_1x_1 + b_2x_2. \quad (8.4)$$

The new features $\{u_1, u_2\}$ are given as a linear combination of $\{x_1, x_2\}$.

In this new representation, we can concentrate our attention in the extracted feature u_1 . In that case, we are indeed reducing the dimensionality (from 2D to 1D), and most importantly the information loss incurred by ignoring u_2 is much less than in the previous case. This is because the spread of the data is larger along u_1 than along u_2 . In other words, there is less variance along the latter feature.

The information loss by concentrating only on u_1 is much less than in the previous case.

We can now use the new representation of our data and concentrate only on feature u_1 . We can also apply other algorithms: Linear regression, clustering, decision trees, etc. It is worth emphasising that the new feature u_1 is now a combination of the original features x_1 and x_2 as shown in Equation (8.4). Depending on the nature of the data, the interpretation of the extracted feature u_1 may or may not be straightforward, but there is no doubt that the modelling part will turn out to be much more efficient.

Remember that the extracted feature is a combination of the original features.

8.2 Principal Component Analysis

PRINCIPAL COMPONENT ANALYSIS, OR PCA, is a favoured technique for dimensionality reduction. It is widely used

among other things for exploratory analysis, data compression and feature extraction. Its application ranges from brain topology² to seasonality analysis³. It makes use of an orthogonal transformation to take the original N coordinates of our dataset into a new coordinate system known as the **principal components**. The aim of the technique is to use a reduced subspace, provided by the principal components, that seeks to maintain most of the variability of the data.

The principal components are ranked, with the first principal component accounting for the largest contribution to the variance. In succession, each of the rest of the principal components provide its contribution to the variability of the dataset. Since the transformation is orthogonal, each of the components is uncorrelated with each other. The orthogonality constraint comes from the fact that the principal components are the eigenvectors of the covariance matrix. The dimensionality reduction comes into place when representing our data using only those principal components that provide the highest contributions to the variance of the dataset.

As we mentioned above, PCA of a matrix $\mathbf{A}$ is equivalent to performing an eigenvalue decomposition of the covariance matrix of $\mathbf{A}$. A covariance matrix provides a summary of the extent to which corresponding elements from two sets of ordered data move in the same direction. Principal component analysis can be applied to a matrix of any dimensions, but remember that the covariance matrix is a symmetric matrix.

² Duffy, F. H. et al. Unrestricted principal components analysis of brain electrical activity: Issues of data dimensionality, artifact, and utility. *Brain Topography*. 4(4), 291–307.

³ Rogel-Salazar, J. and N. Sapsford (2014). Seasonal effects in natural gas prices and the impact of the economic recession. *Wilmott*. 2014(74), 74–81.

The principal components are ranked in decreasing order of contribution to the variance.

We reduce the dimensionality by considering only those principal components with the highest contribution.

PCA is equivalent to an eigenvalue decomposition of the covariance matrix.

A symmetric matrix is a square matrix that is equal to its transpose.

For a matrix $\mathbf{A}$, it is possible to calculate the mean value of each of its columns and create the vector $\boldsymbol{\mu}$. We define the covariance matrix as:

$$\mathbf{V} = E \left[(\mathbf{A} - \boldsymbol{\mu}) (\mathbf{A} - \boldsymbol{\mu})^T \right], \quad (8.5)$$

The covariance matrix can be expressed in this way.

the V_{ij} element of the matrix corresponds to the covariance of the i and j columns of $\mathbf{A}$:

$$V_{ij} = E \left[(A_i - \mu_i) (A_j - \mu_j)^T \right] = \sigma_{ij}. \quad (8.6)$$

This is the expression for the covariance.

In particular, the diagonal elements of the matrix $\mathbf{V}$ are the variances of the components of $\mathbf{A}$:

$$V_{ii} = E \left[(A_i - \mu_i)^2 \right] = \sigma_i^2. \quad (8.7)$$

This is the expression for the variance.

The eigenvalues of the covariance matrix $\mathbf{V}$ are all real and positive, and the eigenvectors that correspond to distinct eigenvalues are orthogonal. This means that we can decompose the matrix $\mathbf{V}$ as:

$$\mathbf{V} = \mathbf{Q} \boldsymbol{\Lambda} \mathbf{Q}^T = \sum_{i=1}^n \lambda_i \vec{q}_i \vec{q}_i^T, \quad (8.8)$$

The covariance matrix can be decomposed in this way.

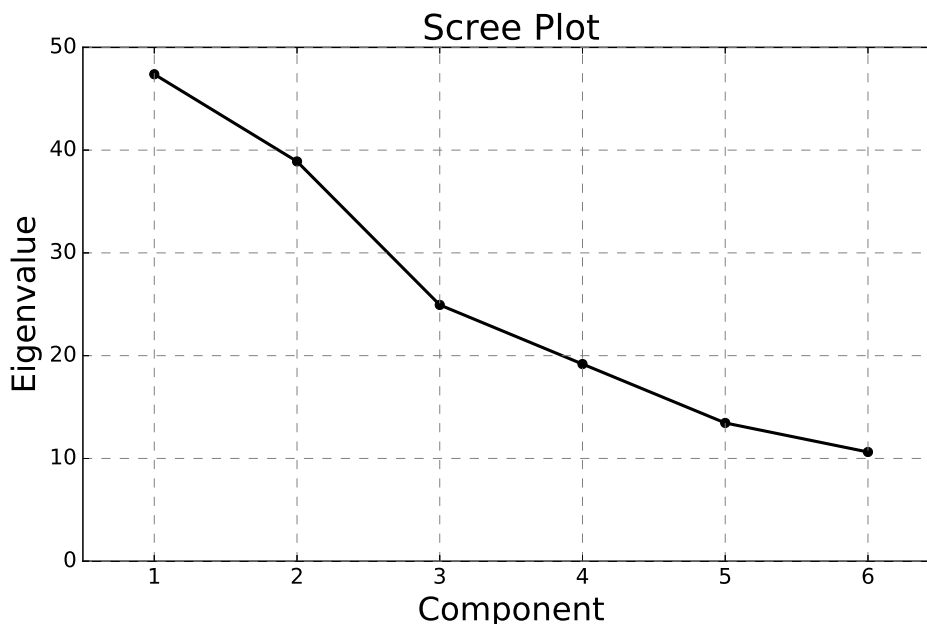
where $\mathbf{Q}$ are the eigenvectors of $\mathbf{A}$, and the values in the matrix $\boldsymbol{\Lambda}$ are the eigenvalues associated with $\mathbf{A}$.

The covariance matrix provides us with information about the spread (variance) and orientation of our dataset. If we are interested in the representation of our data that provides that largest variability, we simply have to find the direction of the largest spread of data. The largest eigenvector of the

The covariance matrix gives us information about the variance of our dataset.

covariance matrix will point into the desired direction, and its eigenvalue is related to the amount of variance explained by this component. In turn, the second largest eigenvector is orthogonal to the first one, and points in the direction of the second largest variability in the data, and so on.

The first eigenvector points in the direction of the largest variability.



It is usual to represent the eigenvalue associated with a component in a scree plot, similar to the one we have seen in [Section 5.3](#). We show a diagrammatic example in [Figure 8.2](#) for six components and their corresponding eigenvalues. The scree plot may provide an indication of how many principal components to keep. Ideally the scree plot shows a steep downward curve: The number of components to keep are those in the steepest part of the plot, defined by the

Figure 8.2: Diagrammatic scree plot showing the eigenvalues corresponding to each of six different principal components.

A scree plot is used to represent the variance contribution of each of the components.

point where the plot bends. As mentioned in [Section 5.3](#) this is sometimes referred to as the *elbow test*.

8.2.1 PCA in Action

WE WILL APPLY PCA TO the image of a jackalope⁴ shown in [Figure 8.3](#). It is available at <https://dx.doi.org/10.6084/m9.figshare.2067186.v1>. We will see how PCA can be used in image compression: We will take an increasing number of components to reconstruct the image. We can read the PNG file into Python with the help of Matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import mean, size
A = imread(r'JRogel_Jackalope.png')
```

⁴ Rogel-Salazar, J. (2016b, Jan). Jackalope Image. <https://dx.doi.org/10.6084/m9.figshare.2067186.v1>

We use `imread` to read an image from a file into an array.

The size is given by the shape of the matrix A:

```
> print(A.shape)
(1880, 1860, 4)
```

The image has 1860×1880 pixels.

As we can see, the image is made out of 1860×1880 pixels arranged in four stacked arrays. Each of the first three arrays corresponds to the red (R), green (G) and blue (B) channels of the image, and the fourth is the so-called alpha, or transparency. In this case, since the image is black and white, the RGB values would be similar, and for the sake of simplicity we will work with a single array made out of the mean value of the four arrays provided:

The stacked arrays that make the image correspond to RGB and alpha values.


```
A1 = np.mean(A,2)
```

We take the mean value of the layers.



We know that the total number of components is given by the original dataset:

```
> full_pc = np.size(A1, axis=1)
> print(full_pc)
1860
```

Figure 8.3: A jackalope silhouette to be used for image processing.

Here we are taking the size of the matrix along the columns, i.e., `axis=1`.

We are interested to see if we can reconstruct the image with a smaller number of components obtained with PCA. We will write a script that uses a number of components less than `full_pc` (in this case 1860) and visually compare the reconstructed image with the original one.

We want to reconstruct the image using a smaller number of components.

The PCA decomposition can be carried out with the implementation in the `decomposition` module in Scikit-learn:

```
from sklearn import decomposition
```

The `decomposition` module is required.

The `.PCA()` function takes a parameter called `n_components` that tells the algorithm the number of components to keep. If the parameter is not provided, all components are kept. In this case, we are going to see the effects that the image reconstruction has when keeping up to 500 components.

We can tell PCA how many components to keep with the `n_components` parameter.

First, we will fit a PCA model with a determined number of components to keep using the `fit` method. Then we will reconstruct the image using the `inverse_transform` method. Let us take a look:

```
components = range(0,600,100)

fig=plt.figure()
for i, num_pc in enumerate(components):
    i+=1
    pca = decomposition.PCA(n_components=num_pc)
    pca.fit(A1)
```

We will programmatically take a number of components to reconstruct the image.

The decomposition is carried out by the `fit` method.

```

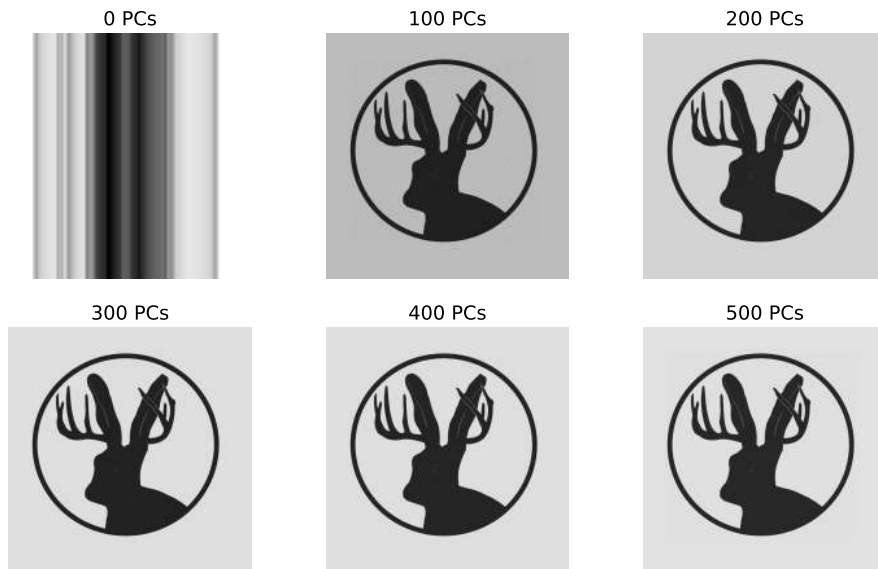
Rec = pca.inverse_transform(pca.transform(A1))
ax = fig.add_subplot(2,3,i,frame_on=False)

# removing ticks
ax.xaxis.set_major_locator(plt.NullLocator())
ax.yaxis.set_major_locator(plt.NullLocator())
plt.imshow(Rec)
plt.title(str(num_pc) + ' PCs')
plt.gray()

```

The reconstruction is done with the `inverse_transform` method.

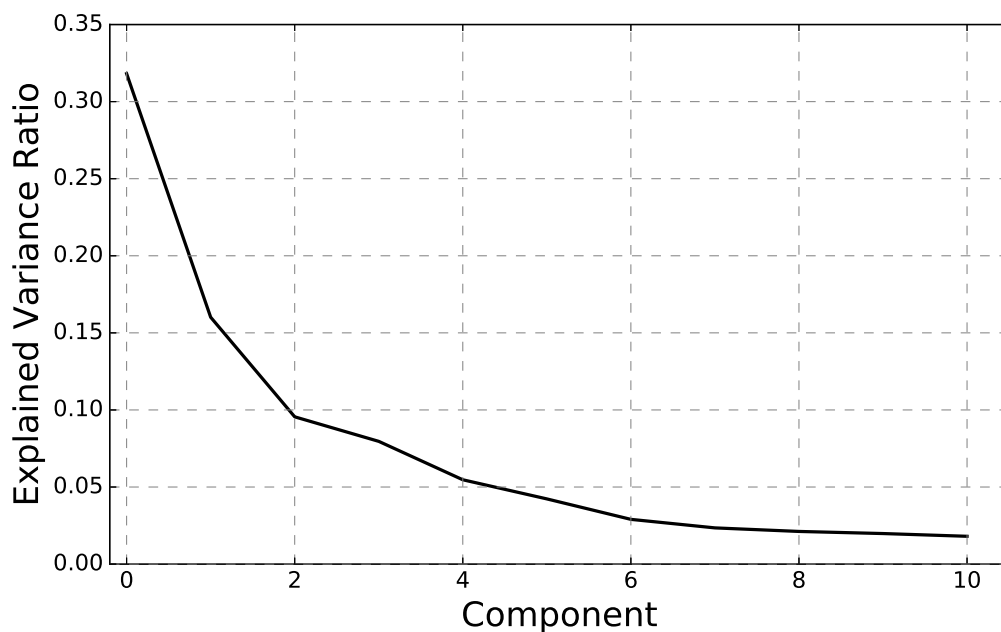
We get rid of the ticks in the axes and show the result.



The results can be seen in [Figure 8.4](#). It is clear that using even 100 components, the image is definitely recognisable,

Figure 8.4: Principal component analysis applied to the jackalope image shown in [Figure 8.3](#). We can see how retaining more principal components increases the resolution of the image.

and by the time we are keeping 500 components, we can clearly distinguish some of the finer features in the image, such as the white streaks in the ears and the eye of the jackalope.



We can take a look at the explained variance ratio we would obtain by allowing all the components in the model and looking at the percentage of variance explained by each of the components:

```
pca1 = decomposition.PCA()
pca1.fit(A1)

var_ratio = pca1.explained_variance_ratio_
```

Figure 8.5: Scree plot of the explained variance ratio (for 10 components) obtained by applying principal component analysis to the jackalope image shown in [Figure 8.3](#).

Remember that by default PCA takes all the components.

We show a scree plot of these values in [Figure 8.5](#) where we can see that the elbow test would indicate that eight components are enough to explain the variability in the image.

8.2.2 PCA in the Iris Dataset

LET US NOW APPLY PCA to the Iris dataset and use the dimensionality-reduced data in a logistic regression classifier. As usual, let us load the dataset into appropriate arrays:

```
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
Y = iris.target
```

This should be second nature to us by now.

We start by splitting our data into training and testing datasets:

```
import sklearn.model_selection as ms

XTrain, XTest, YTrain, YTest =\
    ms.train_test_split(X, Y,\
        test_size= 0.3, random_state=7)
```

We definitely know how to do this and, more importantly, why!

In this case we only have four features: Sepal length, sepal width, petal length and petal width. Let us perform PCA in the training set to give us an idea of the amount of variance explained by the principal components. The result is shown

PCA will let us see how many principal components explain most of the variance.

in the scree plot in [Figure 8.6](#). As we can see, perhaps one or two components are good enough to explain the variance in this dataset.

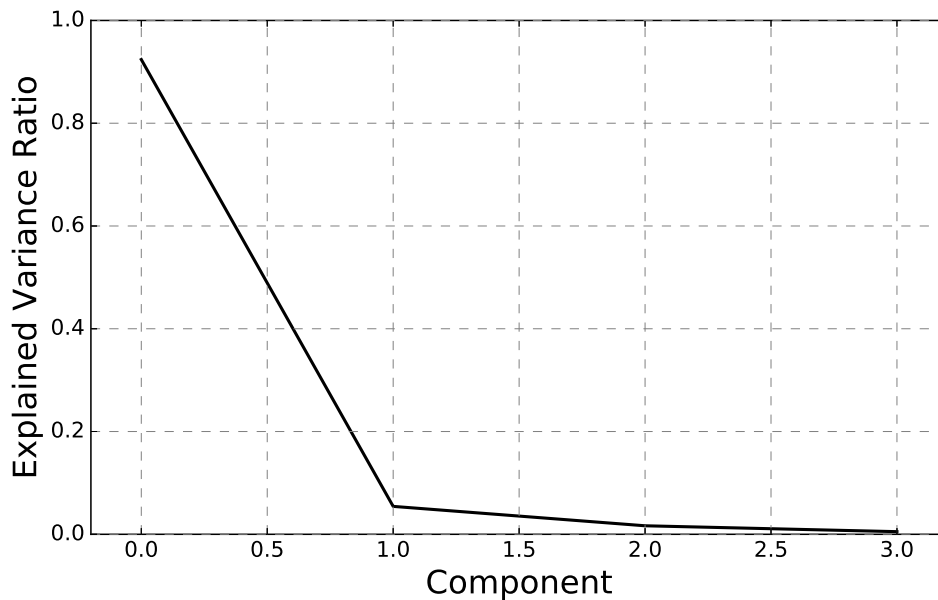


Figure 8.6: Scree plot of the explained variance ratio obtained by applying principal component analysis to the four features in the Iris dataset.

```
from sklearn import decomposition

IrisPCA = decomposition.PCA()
Iris-Decomp = IrisPCA.fit(XTrain)
X-Decomp = Iris-Decomp.transform(XTrain)
var_ratio = IrisPCA.explained_variance_ratio_
```

We simply apply the PCA decomposition to the data. Remember to fit and transform the model.

It is possible to use PCA first to extract the features we are interested in and use them in our classification task later;

in this case, we are employing logistic regression. Let us explore how we can use Scikit-learn to chain the feature extraction to the logistic regression estimator, and at the same time run a search for the best parameters to use. We can do all this with the help of Pipeline, which sequentially applies a list of transformations and estimators. For our purposes we need to chain PCA and LogisticRegression:

```
from sklearn import linear_model
from sklearn.pipeline import Pipeline

logistic = linear_model.LogisticRegression()
pca = decomposition.PCA()

pipe = Pipeline(steps=[('pca', pca),\
                        ('logistic', logistic)])
```

The Pipeline command enables us to chain models. In this case, we chain principal component analysis with a logistic regression.

The steps in the pipeline indicate the transformations that we are expected to carry out. We are also providing a name (included as a string) to be used to refer to the various steps in the pipeline. With that in place, we will conduct a search for both the number of components to keep (from PCA), and the inverse of the regularisation strength used in logistic regression.

The steps in the pipeline indicate the models to be executed.

See [Section 6.3.2](#) for a discussion of logistic regression.

```
from sklearn.model_selection import GridSearchCV
n_components = list(range(1,3))
Cs = np.logspace(-2,4,100)
```

We will search for the best parameters for PCA and logistic regression.

We can now carry out the search employing *k*-fold cross-validation with 10 folds:

```
Iris_cls = GridSearchCV(pipe,\
    dict(pca__n_components=n_components,\
    logistic__C=Cs), cv=ms.KFold(n_splits=10))
```

We need to pass the pipeline to the GridSearchCV command.

We are ready to use our pipe to fit the model:

```
> Iris_cls.fit(XTrain, YTrain)
> print(Iris_cls.best_params_)

{'logistic__C': np.float64(0.21544346900318845),
 'pca__n_components': 1}
```

Training the piped model works in the same way as before.

As we can see, the search indicates that 1 component with a value of C equal to 0.2154 provides the best estimator.

Let us use the result to obtain our prediction:

```
y_pred = Iris_cls.predict(XTest)
```

And the predictions are extracted in the same manner.

Finally, we can obtain a confusion matrix for the prediction of our classifier against the testing set:

```
> from sklearn.metrics import confusion_matrix
> confusion_matrix(YTest,y_pred)

array([[12,  0,  0],
       [ 0, 13,  3],
       [ 0,  3, 14]])
```

In this case we have a three-way confusion matrix.

8.3 Singular Value Decomposition

FINDING THE PRINCIPAL COMPONENTS THAT characterise our data is a powerful tool to have in our Jackalope Data Science toolbox. However, PCA may not be suitable when the initial dimensionality of our data is very large. In the example of the image processing we provided in [Section 8.2.1](#), the resolution of the image could have been very large: For images in megapixels we have $N \geq 10^6$, and the covariance matrix will be $\geq 10^{12}$. Not an impossible task per se, but perhaps other methods may be more suitable and efficient.

One such method is the *Singular Value Decomposition* (SVD) for short. It has the advantage of offering an exact representation of any matrix and, most importantly for us, enables the elimination of parts of the data that are deemed to be less important. It therefore creates an approximate representation with the number of dimensions we choose. In other words, it is a suitable alternative for dimensionality reduction, and for other applications as we shall discuss later in this chapter.

Let us consider an $M \times N$ matrix $\mathbf{A}$. We define the rank r of the matrix $\mathbf{A}$ as the maximum number of linearly independent rows (or columns). We can decompose the matrix $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \quad (8.9)$$

where $\mathbf{U}$ is an $M \times r$ column-orthonormal matrix, $\mathbf{\Sigma}$ is an $r \times r$ diagonal matrix whose elements are called the **singular**

In cases where the dimensionality of our data is very large, PCA may not be suitable.

Singular Value Decomposition (SVD) is a good alternative.

A set of vectors are linearly independent if no vector in the set is a linear combination of the other vectors.

Column-orthonormal: each column is a unit vector and the dot product of any two columns is zero.

values of $\mathbf{A}$, and finally $\mathbf{V}$ is an $N \times r$ column-orthonormal matrix. The columns of $\mathbf{U}$ and $\mathbf{V}$ are called the **singular vectors** of the matrix $\mathbf{A}$. In Figure 8.7 we show a diagrammatic representation of the singular value decomposition.

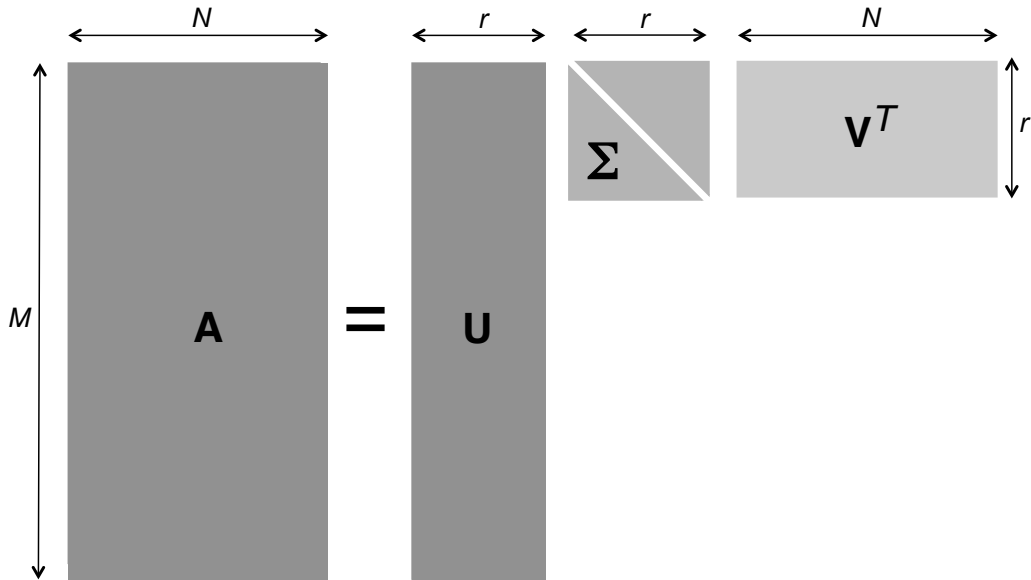


Figure 8.7: Illustration of the singular value decomposition.

The singular vectors of $\mathbf{A}$ provide orthonormal bases for $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$. The SVD process consists of calculating the eigenvalues and eigenvectors of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$. The singular values in Σ are then equal to the square roots of the eigenvalues of $\mathbf{U}$ (or $\mathbf{V}$) in descending order. As we can see from the diagram in Figure 8.7, the number of singular values is equal to the rank r of $\mathbf{A}$.

The singular values are the square roots of the eigenvalues of $\mathbf{U}$ (or $\mathbf{V}$).

SVD results in a representation of the original data in which the covariance matrix is diagonal, and much easier to

In SVD, the covariance matrix has a diagonal representation.

handle than a full matrix, as it is the case in PCA. So, where is the dimensionality reduction? We can think of this problem as finding an approximation to $\mathbf{A}$ with a matrix $\tilde{\mathbf{A}}$ whose rank is r . This problem can be solved by setting the smallest of the singular values of $\mathbf{A}$ to zero. This translates into the elimination of the corresponding rows from both $\mathbf{U}$ and $\mathbf{V}$. It can be shown⁵ that besides achieving a reduction in dimensionality, this process minimises the root-mean-square error between $\mathbf{A}$ and its approximation.

The dimensionality reduction obtained with SVD underlies some techniques used in document analysis such as latent semantic analysis (LSA), where a term-document matrix is used as the basis to obtain linearly independent components. These components can be thought of as hidden or latent concepts in the original data, and hence the name. After the application of SVD, the words are represented by the rows of $\mathbf{U}$, whereas the documents by the rows of $\mathbf{V}$. Document similarity is extracted by comparing the rows of $\mathbf{V}\Sigma$. A similar application to this can be found in the design of recommendation systems, as we will discuss in [Section 8.4](#).

8.3.1 SVD in Action

AS A DEMONSTRATION OF THE power of singular value decomposition, we shall see an application in terms of data compression and noise reduction. Consider the image of a (pixelated) letter *J* as shown in [Figure 8.8](#). As we can see, it is possible to construct the image with four different types

⁵ Golub, G. and C. Van Loan (2013). *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press.

In [Section 6.4.1](#) we obtained a term-document matrix to be used with the Naïve Bayes Classifier.

We will see an example using SVD for image compression and noise reduction.

of pixel columns. It is conceivable that we can represent the same data in a more efficient way.

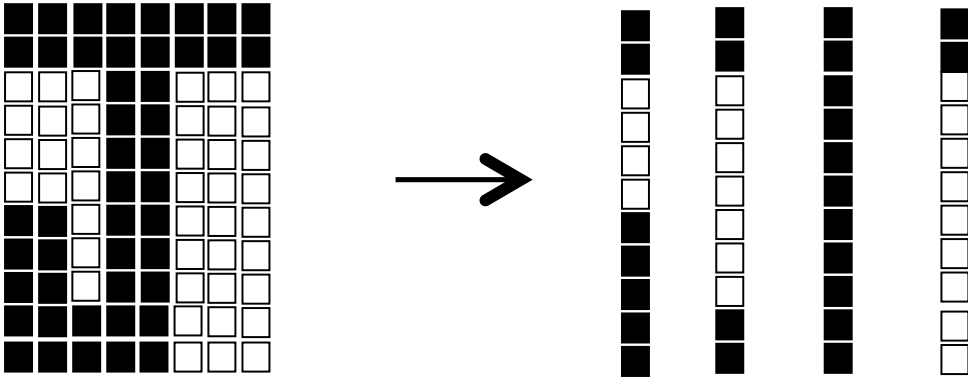


Figure 8.8: Image of a letter *J* (on the left) and its column components (on the right).

Let us construct a noisy picture of the letter *J* shown in [Figure 8.8](#) using 250×150 pixels, i.e., a total of 37,500 pixels to represent this letter:

```
import matplotlib.pyplot as plt
import numpy as np
M = np.zeros((250,150))
M[:31,:]=1
M[:,60:91]=1
M[-31:,:60]=1
M[150:,:31]=1
```

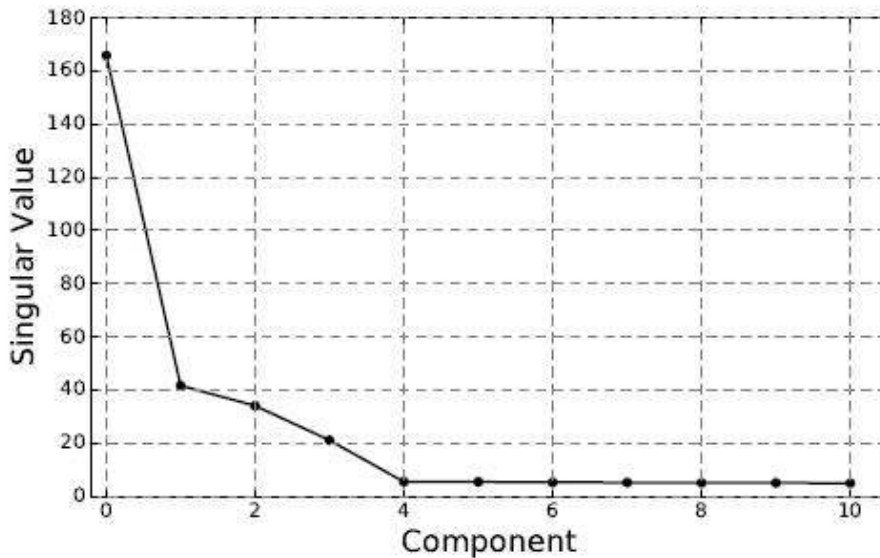
We are constructing the pixelated letter *J* with the help of arrays in Python.

We can even add some random noise to the image. The noise can be generated with the help of the `random.uniform` command:

```
M_noisy = np.asmatrix(np.random.uniform(low=0,\
    high=0.7, size=(250,150)))
```

```
M_noisy = M + M_noisy
```

We are creating a matrix with random entries between 0 and 0.7.



The linear algebra module in NumPy has an implementation of SVD. With it we can readily apply the method to our noisy matrix:

```
U, s, V = np.linalg.svd(M_noisy)
```

Figure 8.9: Singular values obtained from applying SVD in a image of a letter *J* constructed in Python.

NumPy has an implementation of SVD in the linear algebra module.

We can see the first ten singular values obtained from the operation above in [Figure 8.9](#). It is clear that after the fourth component, the curve flattens out.

Let us now put together a script that takes an increasing number of singular components from 1 to 4, and see if we can reconstruct our image using fewer pixels and even reduce the noise:

```
for S in range(1,5):

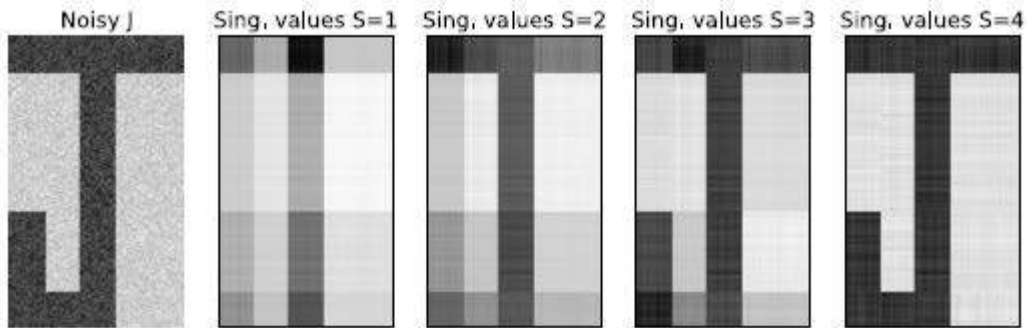
    Sig = mat(np.eye(S)*s[:S])

    U_reduced = U[:, :S]
    V_reduced = V[:, S, :]

    M_rec = U_reduced*Sig*V_reduced
```

The `mat` command interprets the result as a matrix and `eye` corresponds to array representing an identity matrix.

We get rid of the columns we do not need.



The result of the reconstruction process presented above can be seen in [Figure 8.10](#). Each of the panels represents the reconstruction using a different number of components. As we can see, the use of four singular values renders a very good result.

Figure 8.10: Reconstruction of the original noisy letter J (left-most panel), using one to four singular values obtained from SVD.

In terms of the number of elements used in the reconstruction of the image, we needed only 1,600: With four singular values, we have 1,000 from the 250×4 U_{reduced} matrix and 600 from the 4×150 V_{reduced} matrix. That is a good reduction from the original 37,500 elements in the original, with the added benefit of having lessened the noise in the image, as is clearly seen when comparing the first and last panels in [Figure 8.10](#).

We have reduced the number of pixels needed, as well as the noise.

The example above is a simplified application of SVD to more complex images and indeed other data in general. We can consider for instance its use in regression where a clear linear relationship exists between a feature and the target variables, where small noisy errors come into play. Singular value decomposition may enable us to determine the direction in which the data aligns better and safely ignore the rest of the singular values. SVD can be used to detect groupings in the data. This can be useful in determining similarities among data instances. This pattern detection is exploited for example in building recommendation systems, and we shall explore some of the most important concepts in the following section.

We can apply SVD as a pre-processing step.

Think of the four types of columns that make up our letter J.

8.4 Recommendation Systems

WE ARE ALL FAMILIAR WITH online outlets offering us a variety of products from household items, books, music, films, recipes, friends and partners. Some are more specialised than others, and they all have one thing in common: They are interested in continuing a relationship

A variety of outlets offer us a wide range of services, and recommendations are a good way to engage with us.

with us, the customers. One way to continue our engagement with these services is by providing us with meaningful recommendations.

An example of the importance of this type of application is the now famous 2006 Netflix competition⁶ offering a \$1 million USD prize to anyone who could improve their recommendation engine. The 2009 winning entry improved the system by 10%. However, the solution was never implemented due to the engineering efforts that were required.

⁶ Töscher, A. and M. Jahrer (2009). The BigChaos solution to the Netflix grand prize. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf

Picture then the following situation: You are interested in a streaming service that lets you play videos in your favourite device. You create your account and are presented with some of the most popular offerings the service provides as a form of recommendation. The service provider is interested in predicting the rating you are likely to give to an item you have not rated yet. If the rating is high, then the item can be a good product to put in the recommended list and keep you happy and watching.

The aim of a recommendation engine is to predict the likely rating we would give to an unseen item.

Sounds good, but how is that rating produced? The service provider will use information from the items on offer, as well as from other users and their characteristics to try to predict the rating. In some cases maybe even exploiting social network analysis can be done. There are two main camps for this:

Information about the items and from other users is exploited.

1. **Content-based filtering**, where items are mapped into a feature space using their attributes. The

Content-based filtering uses the attributes of the items.

recommendations depend on the characteristics of the items.

2. **Collaborative filtering** considers data from the ratings that other users have provided for specific items. The recommendations depend on the preferences expressed by the user.

Collaborative filtering also uses information from other users.

8.4.1 *Content-Based Filtering in Action*

CONTENT-BASED FILTERING REQUIRES US TO specify the attributes or features that describe the items in our database. We also need to obtain the scores that our users give to each of these features. We can then represent users and items in terms of vectors in the feature space. The item vectors provide a measure of the degree to which items are described by each of the features in question. Similarly, user vectors measure the preferences that the users have for each of the features. We can assume that users will prefer items that are *similar* to the preferences they have expressed.

Content-based filtering is based on the use of vectors describing items based on a series of attributes.

User vectors tell us the preference that users have for the given attributes.

From the discussion about similarity measures in [Section 3.9](#), we know that a convenient way to define similarity is provided by the *cosine similarity*. In this case, the ratings can be obtained by taking the dot product of the user and item vectors and dividing by the product of their norms. Let us go through an example that, although is a little bit oversimplified, will let us see the way in which content-based recommendations work.

For a discussion on cosine similarity, see [Section 3.9](#).

Imagine that we have set up a streaming service and our catalogue contains a total of fiveO films: *Cronos* (1993), *Life*

of *Brian* (1979), *The Never-Ending Story* (1984), *Pinocchio* (1940), and *Titanic* (1997).

Our streaming service has a rather limited catalogue, but it will help us see how content-based filtering works.

The feature space we will consider is given by revenue at the box office, film suitability for children, and Oscar-winning film. The scores (1-5) for these features are shown in [Table 8.1](#).

ID	Film	Box Office	For Children	Oscar
1	<i>Cronos</i>	2	1	1
2	<i>Life of Brian</i>	3	3	1
3	<i>The Never-Ending Story</i>	2	5	1
4	<i>Pinocchio</i>	3	5	5
5	<i>Titanic</i>	5	2	5

Table 8.1: Films considered in building a content-based filtering recommendation system.

ID	Film	Box Office	For Children	Oscar
a	Graham	4	2	5
b	Terry G.	1	2	2
c	Eric	5	4	4
d	John	4	3	1
e	Michael	3	2	5
f	Terry J.	1	1	4

Table 8.2: Scores provided by users regarding the three features used to describe the films in our database.

We can now look at the preferences that our (Pythonic) users have expressed for the three features describing the films in our database. The user scores are shown in [Table 8.2](#). Let us calculate the rating that *Graham* is predicted to give to *Cronos* by obtaining the cosine similarity of the user

and film score vectors:

$$\begin{aligned}
 Sim_{Graham-Cronos} &= \frac{(4)(2) + (2)(1) + (5)(1)}{\sqrt{4^2 + 2^2 + 5^2} \sqrt{2^2 + 1^2 + 1^2}} \\
 &= \frac{15}{\sqrt{(45)(6)}} \\
 &= 0.9128.
 \end{aligned} \tag{8.10}$$

We calculate the cosine similarity between the user and item vectors.

It seems that *Cronos* is indeed a good match for *Graham*. We show below a function that receives as input information from films and users stored in pandas dataframes.

```

import numpy as np, sys
def content_recomm(user, user_df, film_df):
    try:
        u = user_df.loc[user][1:].values
    except:
        print('Error: User does not exist ', \
              sys.exc_info()[0])
        sys.exit(1)
    u_norm = np.linalg.norm(u)
    film_recom = []
    for row in range(film_df.shape[0]):
        f_name = film_df.index[row]
        f = film_df.iloc[row].values
        f_norm = np.linalg.norm(f)
        s = np.dot(u, f)/(u_norm*f_norm)
        if s>0.8:
            film_recom.append((f_name, s))
    film_recom = sorted(film_recom, \
                        key=lambda x: x[1], reverse=True)
    return film_recom

```

We check that the user exists; otherwise, we terminate execution.

We take the norm of the user vector.

In turn we take each of the item vectors and use them to calculate the cosine similarity.

After checking for the threshold, we sort the recommendations and report them.

In the code above we pass on the name of a user (an ID would be preferable in general, though) and calculate the cosine similarity of the user against the films in the database. We then report only those films for which the cosine similarity is greater than 0.8.

We can decide to use a more stringent threshold.

We apply the function to our dataset as follows:

```
import pandas as pd

films = pd.read_csv(u'./Data/FilmCB.csv',\
                    index_col=1)
users = pd.read_csv(u'./Data/UsersCB.csv',\
                    index_col=1)

r1 = content_recomm('Graham', users, films)
r4 = content_recomm('John', users, films)
r6 = content_recomm('Terry J.', users, films)
```

We are assuming that the data is stored in appropriate CSV files.

The `content_recomm` function can be applied to each individual user.

Let us see the results for *Graham*:

```
print(r1)

[('Titanic', 0.99401501176863483),
 ('Cronos', 0.9128709291752769),
 ('Pinocchio', 0.91214859859201181)]
```

We have three recommendations for *Graham*.

As we can see, the top three films for Graham in descending order of cosine similarity are *Titanic*, *Cronos* and *Pinocchio*.

We can do the same for *John* and *Terry J.* We would recommend to *John* all of the five films starting with *Life of*

Recommendations for other users are obtained in a similar fashion.

Brian (0.9898) followed by *Cronos* (0.9607), whereas for *Terry J.* we only have two recommendations: *Titanic* (0.866) and *Pinocchio* (0.8592).

Content-based filtering works well in cases where there are clear attributes that describe the items to be recommended and for which users have provided clear preferences.

However, it has some important drawbacks:

- We need to map each item into a feature space. In general, the feature space may be rather large, and the mapping process is very resource-intensive.
- The recommendations obtained are very limited in scope, as items must be similar to each other.
- User preferences have to be obtained in order for a recommendation to be processed. This is known as the **cold start problem**. It is therefore difficult to provide recommendations to new users who typically have not provided rating information.
- The nature of the filtering makes it difficult to obtain cross-content recommendations. This is because it requires a comparison of items from potentially different feature spaces.

Mapping items into the feature space is resource-intensive.

The cold start problem is a significant drawback.

Cross-content recommendation is difficult.

8.4.2 Collaborative Filtering in Action

THE MAIN ASSUMPTION OF COLLABORATIVE filtering is that users get value from recommendations based on other users with similar preferences and tastes. While in content-based filtering we have used the similarity of items

Collaborative filtering uses a utility matrix to recommend items based on other users' preferences and tastes.

to determine the recommendations, in collaborative filtering instead we make use of a utility matrix whose elements are the preferences reported by users about the items on offer.

There are several approaches that we can take to build our recommendation. For instance, we could take a look at the ratings in our utility matrix to create a matrix detailing the similarity among items. This approach is known as **item-based collaborative filtering** or **memory-based collaborative filtering** and the recommendations that a user receives are based on other items that the user has rated highly in the past. This approach can be thought of in terms of clustering items based on ratings provided by the user. Notice that this item-based filtering is different from content-based filtering, as we are not mapping the attributes of the item to a feature space.

Another alternative is provided by the so-called **model-based collaborative filtering** where instead of looking for similarity among items, we consider the utility matrix to be the result of the product involving two thinner matrices **U** and **V**, encapsulating *latent* concepts in our data. The utility matrix is understood as the ratings that individual users give to specific items available.

Let us consider the utility matrix shown in [Table 8.3](#) detailing the entries (1-10) that users have provided for a set of books: *I, Robot* (I. Asimov, 1950), *The Martian* (A. Weir, 2011), *Do Androids Dream of Electric Sheep?* (P. K. Dick, 1968), *2001 Space Odyssey* (A. C. Clarke, 1968) and *Solaris* (S. Lem, 1961). The question marks in the table indicate items for

There are several approaches to content-based filtering.

In item-based collaborative filtering, users receive recommendations based on items they have rated in the past.

Model-based collaborative filtering is based, the rating that individual users give to specific items available.

Here we will exemplify collaborative filtering for a set of books.

User	<i>I, Robot</i>	<i>The Martian</i>	<i>Do Androids Dream of Electric Sheep?</i>	<i>2001 Space Odyssey</i>	<i>Solaris</i>
Alice	8	2	10	5	1
Bob	4	?	2	10	9
Carl	3	8	4	9	10
Daniel	5	10	4	9	10
Eve	7	2	9	6	?

Table 8.3: Utility matrix of users v books used for collaborative filtering. We need to estimate the scores marked with question marks.

which users have not provided a rating. The missing values occur because there may be many more users than items, plus users only provide scores for a small portion of the items on offer.

Missing values in the utility matrix are denoted with a question mark.

In this case we are not so much interested in reconstructing the utility matrix as it was the case for the example discussed in [Section 8.3.1](#). Instead, we are interested in estimating the missing values in the matrix. Typical applications make use of very sparse matrices.

Our task is to estimate the missing values in the utility matrix.

The idea is that if a user has not rated an item it is likely to be because they have not had a chance to “experience” it. If the estimate is high, then the item is a good candidate to be recommended to that particular user. The use of SVD makes it possible to find those missing values without necessarily having to determine all the missing scores as we shall see.

SVD makes this possible.

Remember that SVD decomposes a matrix $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \quad (8.11)$$

We can think of $\mathbf{U}$ as a matrix where users are represented as row vectors containing linearly independent components. Similarly, the matrix $\mathbf{V}$ corresponds to the items represented as linearly independent row vectors. For the i -th user we can define the row vector problem

$$\mathbf{p}_i = U_i \sqrt{\Sigma}^T. \quad (8.12)$$

Similarly, for the j -th item we have

$$\mathbf{q}_j = \sqrt{\Sigma} V_j^T; \quad (8.13)$$

and the score that user i will give to item j is given by:

$$r_{ij} = \mathbf{p}_i \mathbf{q}_j^T. \quad (8.14)$$

This works for full matrices, but in this case we are dealing with sparse ones. If we impute an initial value of zero, we are effectively indicating that the user will definitely not prefer the item. Some potential solutions include normalising the ratings by subtracting the mean rating by user for example or, as we will do in this case, taking the mean rating of the item in question.

The objective function that we are trying to minimise can then be expressed as follows:

$$\min_{\mathbf{p}^* \mathbf{q}^*} \sum_{(i,j)} \left(r_{ij} - \mu - \mathbf{p}_i \mathbf{q}_j^T \right)^2 + \lambda \left(|\mathbf{p}_i|^2 + |\mathbf{q}_j|^2 \right), \quad (8.15)$$

where μ is the mean value we referred to above, and λ is the hyperparameter that controls the amount of regularisation.

The matrices $\mathbf{U}$ and $\mathbf{V}$ are related to the users and items.

This matrix decomposition enables us to formulate the problem in terms of an objective function to be optimised, as we shall see below.

We have sparse matrices due to the cold start problem.

In this case we will use the mean rating for an item to avoid this issue.

The objective function we are trying to minimise is given by the matrix decomposition defined above.

Please note that this optimisation problem has two unknowns, i.e., $\mathbf{p}$ and $\mathbf{q}$ and convexity is not guaranteed. A further discussion goes beyond the scope of this book, and you are recommended to read more about methods such as *alternating least squares*^{7,8} to tackle this problem.

In this case, we will be tackling this problem with a naïve approach (i.e., use with care!) to find the likely ratings that Bob will give to *The Martian*, and Eve to *Solaris* (see [Table 8.3](#)). Starting with a comma-separated-value (CSV) file holding the data from [Table 8.3](#) we can load the information into a pandas dataframe. We still have to deal with the cold start problem as the SVD method will not be able to deal with missing data. In this case we will initialise the missing values with the mean score provided by other users:

```
import pandas as pd
import numpy as np

A = pd.read_csv(u'./Data/CF_Table.csv',\
               index_col=0, na_values=['?'])

A.fillna(A.mean(), inplace=True)
```

In the code above we are assuming that the missing data in the CSV file is denoted with a question mark, as shown in [Table 8.3](#). We load the data into a pandas dataframe and specify that any question marks encountered are replaced by NaN so that the software handles the missing data appropriately. In this case we are replacing the missing values by the mean score given to each item with the help of

⁷ Takács, G. and D. Tikk (2012). Alternating least squares for personalized ranking. In *Proceedings of the Sixth ACM Conference on Recommender Systems, RecSys '12*, New York, NY, USA, pp. 83–90. ACM.

⁸ Hu, Y., Y. Koren, and C. Volinsky (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, ICDM '08*, Washington, DC, USA, pp. 263–272. IEEE Computer Society.

We can tell pandas what characters represent missing values with `na_values`.

We replace missing values with `fillna`.

The various methods in pandas let us manipulate the missing values as required.

the `.fillna` method. Please note that the replacement is requested to be done in place. In this case, Bob's initial rating for *The Martian* is calculated to be 5.5, and Eve's initial rating for *Solaris* is estimated to be 7.25.

Let us now write a function that uses SVD to improve on the predicted scores:

```
import sys

def cf_recomm(user, book, dat, S=2):
    try:
        uind = dat.index.get_loc(user)
        bind = dat.columns.get_loc(book)
    except:
        print('Error: User/item doesn't exist', \
              sys.exc_info()[0])
        sys.exit(1)
    else:
        uind = dat.index.get_loc(user)
        bind = dat.columns.get_loc(book)

    U, s, V = np.linalg.svd(dat)
    Reduced_Sig = np.asmatrix(np.eye(S)*s[:S])
    U_reduced = U[:, :S]
    V_reduced = V[:, :S]
    recom = U_reduced[uind, :]*Reduced_Sig*\
            np.asmatrix(V_reduced[:, bind]).T

    return recom.item(0)
```

We check that the user (or item) exists; otherwise, we terminate execution.

The scores can be obtained using only the relevant parts of the utility matrix. There is no need to compute the entire multiplication.

SVD is applied to the data, and the dimensionality is reduced. We are using a minimum of $S = 2$ components.

We use the reduced matrices to obtain our estimate and report it.

We can now use the function to determine the scores as follows:

```
> bob_score = cf_recomm('Bob', 'The Martian', A)
> eve_score = cf_recomm('Eve', 'Solaris', A)
> print(bob_score)

7.233127706402881

> print(eve_score)

5.323332988443551
```

We only need to provide the function with the relevant row and column from matrix **A** to calculate the rating estimate.

This results in Bob's score for *The Martian* being 7.23, while Eve is predicted to score *Solaris* with 5.32. We can then recommend Bob to consider reading *The Martian*, whereas Eve may want to consider other items instead of *Solaris*. SVD has enabled us to decompose the utility matrix shown in Table 8.3 into latent features that users rate highly or poorly. In this case for instance, these latent features could be themes such as books with *sentient robots* or *action taking place in Space*.

The singular value decomposition can be understood in terms of latent features in our data.

Please note that in the example above, other than using SVD to find the missing values in our utility matrix, we have not employed any further optimisation or even considered the use of cross-validation. There are a number of recommendation system implementations for Python, and we recommend taking a look at some such as surprise, TensorFlow Recommenders, and recmetrics.

Surprise: <https://github.com/NicolasHug/Surprise>
 TF Recommenders: <https://github.com/tensorflow/recommenders>
 Recmetrics: <https://github.com/statisticianinstilettos/recmetrics>

Recommendation systems that use model-based collaborative filtering are very flexible, and there are a few things that need to be taken into account when using them, including:

- Large datasets are needed to construct meaningful utility matrices
- The utility matrices obtained are very sparse
- The possibility of *shilling attacks* must be considered.
A shilling attack consists of malicious users providing a large number of very positive ratings to their own products and very negative ones to their competitors', skewing the recommendations obtained
- Initial values for the preferences of a new user are potentially non-existent, and therefore a lot of data is needed to tackle this cold start problem. The use of implicit feedback is a way to deal with this issue

Utility matrices can be large and sparse.

Shilling attacks consist of malicious users providing a large number of positive reviews to their own products, for example.

Implicit feedback includes information from other sources such as search patterns, browsing history, etc.

8.5 *Dealing Effectively with Dimensionality Reduction*

AS WE HAVE SEEN IN this chapter, dimensionality reduction is a very useful tool under our Jackalope Data Scientist belt. We can think of it as trying to pack for a week-long holiday with just a carry-on: You want to bring the essentials and leave the fluff behind. While this process can make your life a whole lot easier, it also introduces ethical conundrums that would make even the most seasoned ethicist break into a philosophical sweat.

While dimensionality reduction is a useful tool in our Jackalope Data Scientist belt, there are things we need to consider when using it.

Before diving into the ethical deep end, let us remind ourselves that dimensionality reduction actually is all about transforming data from a high-dimensional space to a lower-dimensional one, while preserving as much valuable information as possible. Techniques like Principal Component Analysis (PCA) discussed in [Section 8.2](#) are tools that Marie Kondo, would be happy to apply in decluttering datasets and keeping only the features that “spark joy”—or in this case, usefulness. .

Remember that dimensionality reduction transforms data from a high- to a lower-dimensional space.

One of the primary ethical concerns with dimensionality reduction is the potential loss of crucial information. Imagine you are Dixon Hill facing a thick dossier of evidence. Dimensionality reduction will enable you to summarise that dossier to fit on a single index card. Sure, it is more manageable, but what if you inadvertently leave out the fingerprint on the murder weapon before you are called to the bridge and have to ask the computer to freeze the programme.

We need to be mindful of the potential loss of crucial information.

Let us consider a more real example. In medical diagnoses using machine learning, reducing dimensions might discard subtle but vital symptoms that differentiate between similar diseases. This could lead to misdiagnoses and, consequently, inappropriate treatments. The ethical stakes here are sky-high—human lives could be at risk.

Here is an example in the medical field.

Then there is the issue of bias. Dimensionality reduction can inadvertently amplify biases present in the original data. It is like using a funhouse mirror to reflect societal prejudices: The distortions might highlight biases even more

Be mindful of bias!

grotesquely. For example, if a dataset has inherent gender or racial biases, reducing its dimensions might obscure these biases from detection but not from influence. This can lead to biased AI models that perpetuate and even exacerbate existing inequalities. A good example of this is the bias inherent in one of the most frequently used datasets in machine learning: The Boston Housing dataset.

Biased AI models can perpetuate or exacerbate existing inequalities.

A former darling of regression analysis, the Boston Housing dataset captures various housing attributes from Boston's 1970s suburbs and is more than just a statistical staple—it harbours some controversial features that warrant closer scrutiny. The dataset was originally published by Harrison and Rubinfeld⁹. In the first edition of this book, we used it to exemplify building pipelines in Python in an appendix. Given the discussion we will have here, I have decided to change the dataset for Appendix 9.3.

⁹ Harrison, D. and D. Rubinfeld (1978). Hedonic prices and the demand for clean air. *J. Environ. Economics and Management*, 81–102.

Let us concentrate on two variables that appear in the dataset:

- $B = 1000(Bk - 0.63)^2$ where Bk is the proportion of black population by town
- LSTAT – percentage lower status of the population

The 'B' feature calculates the squared difference between a town's proportion of black residents and a fixed value (0.63 or 63%). The further a town's black population strays from 63%, the higher its 'B' value. This presents a problem in the form of racial bias as the inclusion of a feature that penalises deviations from a specific racial makeup is deeply

This presents a problem in the form of racial bias.

troubling, as it perpetuates stereotypes and biases. Furthermore, the arbitrary choice of 63% lacks a clear rationale, skewing results and interpretations.

‘LSTAT’ denotes the percentage of the population deemed of “lower status”, a term fraught with socioeconomic implications and ambiguity. What exactly is “lower status”? This vague term is open to biased interpretations and enhances socioeconomic stereotyping: Linking housing prices to the proportion of “lower status” residents can reinforce harmful stereotypes.

Vague terms can enhance bias; in this case socioeconomic stereotyping.

Going back to our discussion on dimensionality reduction, we should take care in using it, as it can also obfuscate the interpretability of models. Picture a Swedish chef who whips up a complex dish with a secret ingredient. Dimensionality reduction is that secret ingredient—it makes the dish (or model) better, but good luck figuring out how it got there. For regulatory compliance and ethical transparency, being able to explain how a model arrives at a decision is crucial. Techniques like PCA, while mathematically elegant, can turn the model into a black box, making it challenging to provide clear explanations to stakeholders or affected individuals.

As discussed in [Section 7.5](#).

Here are some tips and best practices to bear in mind when using dimensionality reduction:

- **Choose Your Weapon Wisely:** Picking a dimensionality reduction method is like choosing the right tool from a vast toolbox. The method should suit your data’s nature—whether it is continuous, categorical, or count

Yes, our Jackalope Data Scientist toolbox!

data. Linear methods like PCA excel at capturing global structures at the expense of local ones.

- **Pre-process Like a Pro:** Before diving into dimensionality reduction, prep your data. Centring and scaling ensure variables contribute equally, crucial for datasets with diverse features. Specific transformations like log-transform for multiplicative changes or variance stabilisation for sequencing data are essential. Proper pre-processing is like priming a canvas before painting.
- **Embed with Care:** When working with similarity or dissimilarity data, choose the right metric—Manhattan distance for binary data or Jaccard for sparse features.
- **Dimension Decisions:** Deciding the number of dimensions to retain is crucial. Use scree plots and the elbow rule to guide your choice. Higher-order components might reveal hidden signals, especially when confounding factors dominate the strongest signal. Remember, the number of dimensions cannot exceed the smaller of the number of observations or variables.
- **Visualise with Proper Ratios:** Aspect ratios in dimensionality reduction plots are like the frame for a masterpiece; they affect perception. Ensure plots reflect the variance explained by output dimensions. For PCA, maintain consistent ratios.
- **Uncover Hidden Patterns:** Dimensionality reduction aims to expose latent variables and structures. Look for discrete clusters or continuous gradients. For clustering, use PCA as a noise reduction step. Continuous gradients

Here are some tips and best practices to bear in mind when using dimensionality reduction.

often appear as horseshoes in PCA plots, indicating linear gradients.

- **Test for Robustness:** Stability is key. Check your results against parameter changes and outliers. Use visualisations and ensure that your application of dimensionality reduction reflects true data patterns, not artefacts.

By following these tips, you can better navigate the ethical and practical complexities of dimensionality reduction, ensuring your data analysis is both insightful and responsible. Dimensionality reduction is a powerful tool, often indispensable for taming high-dimensional data. Despite its utility, it is frequently misused or misinterpreted, leaving Jackalope Data Scientists in a sea of methods, metrics, and parameters.

Dimensionality reduction is a powerful tool. Use it with care!

But wait, there is more! One final nugget of wisdom: Document every decision you make—method selections, distance metrics, parameter values. Keeping a meticulous record is crucial for reproducible research. Tools like Jupyter Notebooks are perfect for this task, allowing you to create comprehensive analysis reports with text, code, and outputs all in one place. This not only helps others replicate your work but also streamlines future analyses.

Remember: Documentation, documentation, documentation!

8.6 *Summary*

THE CURSE OF DIMENSIONALITY INTRODUCED in [Section 3.10](#) is an important issue that has to be considered in any

data science modelling. This is particularly true in those cases where we have a large number of dimensions, imposing the need for larger datasets. We started this chapter by introducing a couple of techniques for dimensionality reduction that no Jackalope Data Scientist should forgo.

It is clear that using the most important features that describe our data science problem is the best way to proceed in the modelling part of our workflow. Therefore, careful feature selection can be of great help in cases where we have a large number of features. An alternative for this is extracting features. This may help reduce the number of dimensions, while conveying as much useful information as possible.

In Principal Component Analysis (PCA), given a feature matrix $\mathbf{A}$, we are interested in performing an eigenvalue decomposition of the covariance matrix. This is because we want to obtain a representation (orientation) of our data that provides the largest variability. The eigenvalues obtained from PCA are related to the amount of variance explained by each of the associated eigenvectors, or components. This enables us to represent our dataset with a reduced number of features and even tackle issues with noisy data as demonstrated here.

Another approach we saw in this chapter is Singular Value Decomposition (SVD) of the matrix $\mathbf{A}$ directly. The method offers an exact representation of any matrix and enables us to eliminate parts of the data that are deemed to be less

important. This is done by finding an approximation $\tilde{\mathbf{A}}$ to the original matrix by setting the smallest singular values of $\mathbf{A}$ to zero. This process eliminates the corresponding rows of both $\mathbf{U}$ and $\mathbf{V}$, the component matrices. We also discussed how SVD can be used in building recommendation systems.

In general, dimensionality reduction helps with:

- Reducing computational expense
- Reducing noise in the dataset
- Increasing the generalisation of our predictions
- Reducing overfitting
- Enhancing our understanding of the data and the outcomes of our models.

Finally, we discussed how dimensionality reduction is crucial for managing high-dimensional data, but it carries risks such as significant information loss and the introduction of biases, which can distort model outcomes and perpetuate stereotypes. To mitigate these risks, it is essential to choose the appropriate method based on the data type, pre-process data correctly, handle categorical data appropriately, and carefully decide on the number of dimensions to retain. Documenting every decision ensures reproducibility and transparency, making dimensionality reduction a more reliable and ethical tool in data analysis.

9

Kernel Tricks up the Sleeve: Support Vector Machines

THE DREADED CURSE OF DIMENSIONALITY can be a tough adversary for any Jackalope Data Scientist out there.

Fortunately, we now know that there are a number of tools that can be used to defend ourselves against it: From careful feature selection to dimensionality reduction. In the last chapter we saw how Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) can be used to reduce the number of features in our problem. This reduction is achieved thanks to combinations of the original attributes that conserve as much information as possible.

Another way to tackle our machine learning problems is with the application of suitable transformations to our data. Take for instance the logarithmic transformation we applied in [Section 4.5](#), where we successfully managed to represent our data in terms of a linear relationship in the transformed space. We can now take a step forward and combine the

Probably as effective as defending yourself against a man armed with a banana...

Remember that the number of features corresponds to the number of dimensions in our problem.

We would like to combine the task of selecting features with the application of useful transformations.

idea of selecting relevant features by applying suitable transformations.

In this chapter we will consider the use of kernel functions as a powerful technique to manipulate datasets in their own original feature space, but as though they were projected into a higher dimensional space. This transformation lets us carry out manipulations in a more straightforward manner.

We then explain the Support Vector Machine (SVM) algorithm.

9.1 *Support Vector Machines and Kernel Methods*

WE HAVE CONCENTRATED SO FAR on extracting features from our existing data to obtain a representation using fewer dimensions while retaining as much information as possible. A different approach is to transform the data in such a way that our learning tasks look easier to be carried out than in the original feature space.

Instead of extracting new features, we can perform transformations to simplify the learning tasks.

In a sense, this is what we have done when applying a logarithmic transformation to the mammals dataset in [Section 4.5](#): In the transformed feature space, the relationship between a mammal's body and brain is better approximated by a straight line than when using the untransformed features. Once we have carried out the learning task, we need to invert the transformation to the original feature space.

We have done this with a logarithmic transformation in [Section 4.5](#).

Continuing with that train of thought, let us consider the datasets shown in [Figure 9.1](#) using a space spanned by features X_1 and X_2 . The data points shown in panel a) can

be separated into two groups by using a diagonal line from the upper-left-hand corner to the lower-right-hand one. In other words, we say that the dataset is linearly separable. However, the squares in panel b) cannot be separated in such an easy manner from the circles in this representation. It may be possible that the data points are actually linearly separable in a different feature space, perhaps even in a higher-dimensional one. If that is the case, we can try to find the transformation as we did for the mammals dataset and carry out our classification task in the new space.

Linear separability refers to the possibility of discriminating data classes with a single decision surface.

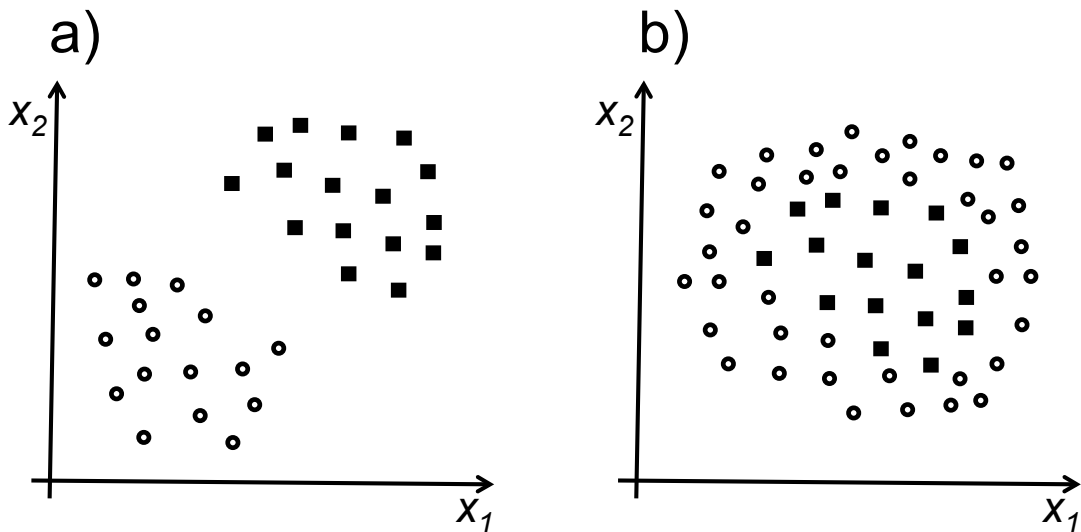


Figure 9.1: The dataset shown in panel a) is linearly separable in the $X_1 - X_2$ feature space, whereas the one in panel b) is not.

Although it is possible to find a new feature space where linear separability can be achieved, there are a couple of things we need to consider. First of all, we may not know the transformation that needs to be carried out, and we will need to search for it. Second, if indeed the data is separable

We may not know the transformation that needs to be applied, plus we are adding complexity to the model.

in a higher dimensional space, we are adding complexity to our model. Could it be possible to manipulate the dataset in its original feature space, but as if it were projected into a higher-dimensional one without the need to explicitly carry out the transformation? It sounds far-fetched, but it turns out that the answer to that question is *YES*, and we will discuss this point in further detail in [Section 9.1.2](#). However, before we get there, let us consider the linearly separable case.

As Jorge Luis Borges would remind us¹, a line is made out of an infinite number of points, a two-dimensional space is made out of an infinite number of lines, a three-dimensional one is made out of an infinite number of planes, etc. As such, a two-dimensional linear classifier will correspond to a decision boundary given by a line (as discussed above). In three dimensions it will be a plane, and in higher dimensions we can talk about a hyperplane. In general we can express a linear classifier as:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b, \quad (9.1)$$

where the weight vector $\mathbf{w}$ is normal to the hyperplane and b is the bias. In our classification task, the training data is used to learn $\mathbf{w}$ and is not used again since we can directly use the weight vector for classifying new unseen data.

If the data points are linearly separable, we can end up with a situation like the one shown in [Figure 9.2](#), where we have more than one separation boundary. Not only is it our task to find the classification boundary, but also to find the best one.

It is possible though to manipulate the dataset without the need to explicitly carry out the transformation.

¹ Borges, J. L. (1984). *El Libro de Arena*. El Ave Fénix. Plaza & Janés.

In general we talk about a hyperplane as the decision boundary for a linearly separable dataset.

The weight $\mathbf{w}$ is normal to the hyperplane and b is the bias.

Our task is to find the best classification boundary.

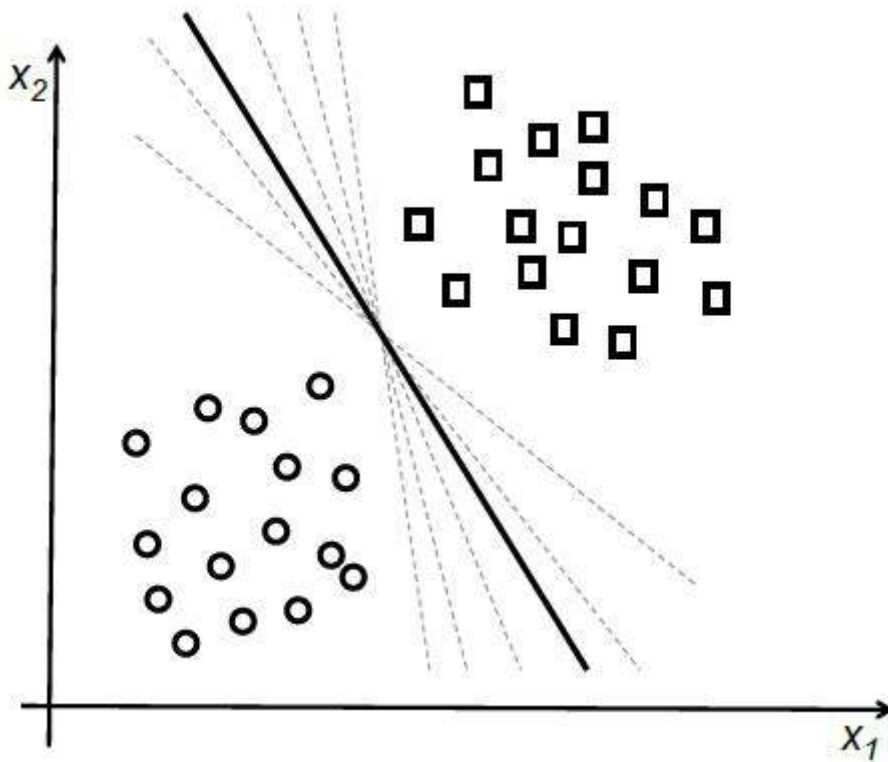


Figure 9.2: Linearly separable dataset may have a large number of separation boundaries. Which one is the best?

9.1.1 Support Vector Machines

A SUPPORT VECTOR MACHINE (SVM) is a binary linear classifier where the classification boundary is built in such a manner as to minimise the generalisation error in our task. Unlike other classifiers we have discussed, the SVM boundary is obtained using geometrical reasoning instead of algebraic. With that in mind, the generalisation error is associated with the geometrical notion of a **margin**, which can be defined as the region along the classification boundary that is free of data points.

A support vector machine minimises the generalisation error associated with the geometrical notion of a margin.

In that manner, an SVM has the goal of discriminating among classes using a linear decision boundary that has the largest margin, giving rise to the so-called **maximum margin hyperplane** or (MMH). Having the maximum margin is equivalent to minimising the generalisation error. This is because using the MMH as the classification boundary minimises the probability that a small perturbation in the position of a data point results in a classification error. Intuitively, it is easy to see that a wider margin results in having better-defined and separate classes.

Given the discriminant function in Equation (9.1), we can determine the class label of a new record by considering the sign of the function $f(\mathbf{x})$:

$$f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i + b \begin{cases} \geq 0 \rightarrow y_i = +1, \\ < 0 \rightarrow y_i = -1. \end{cases} \quad (9.2)$$

The weight vector $\mathbf{w}$ defines the orientation of the classification boundary, whereas the bias b specifies a translation from the origin. In [Figure 9.3](#) we depict the main components of a support vector machine: The maximum margin is defined by the **support vectors**, i.e., the data points that lie closest to the classification hyperplane. These points are marked in solid black in [Figure 9.3](#). The task of finding the MMH boils down to the optimisation of a convex objective function, and this means that we are guaranteed to obtain a global optimum.

The support vectors are in fact the data points that are the most difficult to classify, as they are closest to the

The maximum margin hyperplane is the linear decision boundary with the largest margin.

A wider margin results in having better-defined and separate classes.

The class label of a new record is given by the sign of the discriminant function.

The support vectors are the data points closest to the classification hyperplane.

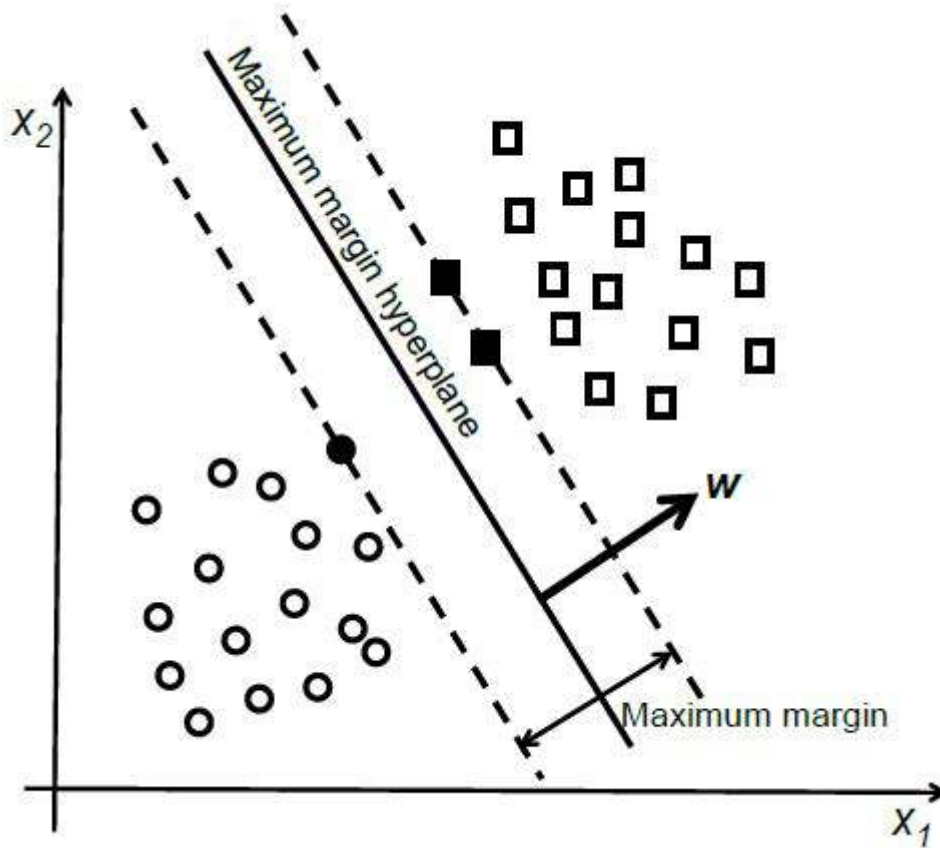


Figure 9.3: A support vector machine finds the optimal boundary by determining the maximum margin hyperplane. The weight vector w determines the orientation of the boundary and the support vectors (marked in solid black) define the maximum margin.

classification boundary. Other points, particularly those that lie farther away, do not contribute to defining the boundary. In that respect, only the support vectors have a direct effect on the location of the classification boundary. We can get a further understanding of the support vectors by considering an analogy with static equilibrium in mechanics: The forces F_n exerted on a static still slab (the decision boundary) defined by the normal vector w must satisfy the following equilibrium conditions:

The support vectors are the data points that are the most difficult to classify.

$$\sum_n \mathbf{F}_n = 0, \quad (9.3)$$

These are the equations of mechanical equilibrium.

$$\sum_n \mathbf{s}_n \times \mathbf{F}_n = 0, \quad (9.4)$$

where $\mathbf{s}_n$ are the support vectors in our analogy. Equation (9.4) implies that the total torque exerted on the boundary is zero, and we can think of these vectors as providing support for the hyperplane to be in “static equilibrium”. The SVM algorithm was proposed by Vapnik and collaborators² in 1992, and its variations and applications are diverse.

² Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. In *5th Annual ACM Workshop on COLT*, Pittsburgh, PA, pp. 144–152. ACM Press.

Remember that we are interested in obtaining a linear classifier where the margin is as large as possible.

Furthermore, let us recall that the distance from a point with coordinates (x_0, y_0) to a line defined by $Ax + By + C = 0$ is given by:

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}, \quad (9.5)$$

and hence the distance from one of the dotted lines in [Figure 9.3](#) and the classification boundary is given by:

$$\frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}, \quad (9.6)$$

This is the distance from the support vector to the classification boundary.

and the margin is therefore twice this amount. It is clear that in order to maximise the margin, we need to minimise $\|\mathbf{w}\|$ with the condition that there are no other points

inside the margin. It is possible to express this optimisation problem as:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2, \quad (9.7)$$

$$\text{s.t. : } y_i (\mathbf{w}^T \mathbf{x} + b) \geq 1, \quad i = 1, \dots, n.$$

The SVM algorithm as an optimisation problem.

Up to this point we have considered a situation where the data points are linearly separable as depicted in panel a) of [Figure 9.1](#). Nonetheless, in general the data may not necessarily be so nicely separated. In those cases we can consider using a larger margin than the one originally obtained at the expense of incurring some training error. This can be achieved by introducing *slack variables* $\xi_i \geq 0$ and replacing the constraint of the optimisation problem in Equation (9.7) with the following one:

$$y_i (\mathbf{w}^T \mathbf{x} + b) \geq 1 - \xi_i, \quad i = 1, \dots, n. \quad (9.8)$$

In some cases we can tolerate a larger margin at the expense of incurring some training error. This can be done with the help of slack variables.

The slack variables generalise our optimisation problem allowing for some misclassification in the training records at a cost C . A data point is misclassified if the value of its slack variable is greater than 1. A bound on the number of misclassified examples is given by the sum of the slack variables. Our optimisation problem can therefore be expressed as:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum \xi_i, \quad (9.9)$$

$$\text{s.t. : } y_i (\mathbf{w}^T \mathbf{x} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

The slack variables provide us with a way to regularise our optimisation problem.

The soft-margin SVM formulation.

This formulation of the support vector machine algorithm is called the **soft-margin SVM**³. This is a form of the variance-bias trade-off we know and love. Every constraint imposed can be satisfied for a sufficiently large ξ_i . The constant C is a regularisation parameter: For small values of C we ignore the constraint and we obtain a large margin, whereas for large values of C the constraint is imposed, and we end up with a narrow margin. If $C \rightarrow \infty$, then all constraints are imposed and we get a “hard” margin.

Let us start by solving the quadratic programming problem given in Equation (9.7). Since it is a constrained optimisation problem, we can start by using the well-known method of Lagrange multipliers,⁴ where we are interested in optimising a function $f(x, y)$ subject to the constraint $g(x, y) = 0$. We construct a new function called the Lagrangian such that:

$$L(x, y, \alpha) = f(x, y) - \alpha g(x, y), \quad (9.10)$$

where the new variable α is called the Lagrange multiplier.

The method can be extended to be applied to n dimensions and m constraints. This means that taking the appropriate derivatives to calculate the gradient of the Lagrangian results in $n + m$ equations all set to zero. This optimisation problem can be tackled with the so-called steepest descent algorithm.

For the case of our interest, we have that $f(\cdot) = \frac{1}{2} \|\mathbf{w}\|^2$ and $g(\cdot) = y_i (\mathbf{w}^T \mathbf{x} + b) - 1 = 0$. Our new unconstrained

³ Cortes, C. and V. Vapnik (1995). Support vector networks. *Machine Learning*. 20, 273–297.

⁴ Bertsekas, D. (1996). *Constrained Optimization and Lagrange Multiplier Methods*. Athena scientific series in optimization and neural computation. Athena Scientific.

The Lagrange multipliers method can be used to solve our optimisation problem.

problem in terms of the Lagrangian is expressed as:

$$\min_{\mathbf{w}, b} L = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^l \alpha_i y_i (\mathbf{w}^T \mathbf{x}_i + b) + \sum_{i=1}^l \alpha_i, \quad (9.11)$$

The constrained form of the SVM optimisation problem.

where l is the number of training points. We know that the partial derivatives with respect to $\mathbf{w}$ and b should be zero at the minimum of the Lagrangian function. We can then write the following:

$$\partial_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^l \alpha_i y_i \mathbf{x}_i = 0, \quad (9.12)$$

$$\partial_b L = \sum_{i=1}^l \alpha_i y_i = 0, \quad (9.13)$$

The partial derivatives of the Lagrangian formulation with respect to $\mathbf{w}$ and b .

From the equations above, we obtain the following conditions that enable us to find the margin for our SVM problem:

$$\mathbf{w} = \sum_{i=1}^l \alpha_i y_i \mathbf{x}_i, \quad (9.14)$$

$$\sum_{i=1}^l \alpha_i y_i = 0. \quad (9.15)$$

These are the conditions that enable us to find the margin for our SVM problem.

The first condition above already tells us something about what the weights $\mathbf{w}$ are: They turn out to be linear combinations of the training inputs and outputs, as well as the values α_i . In turn, we expect that most of the α_i parameters are zero, and those that are not will correspond to the actual support vectors.

The weights $\mathbf{w}$ are linear combinations of the training inputs and outputs.

The problem cast above is known as the **primal** form, and we can solve it directly in that representation. However, there are advantages in actually solving the **dual** representation of the problem. In that manner, instead of finding the minimum over $\mathbf{w}$ and b , with constraints over the parameters α_i , we can maximise over α subject to conditions (9.14) and (9.15). By substituting these conditions into expression (9.11), we free ourselves from dependencies on $\mathbf{w}$ and b . The Lagrangian in the dual form is given by:

$$\max_{\alpha_i} L_D = -\frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^l \alpha_i, \quad (9.16)$$

subject to $\sum_{i=0}^l \alpha_i y_i = 0$, and $\alpha_i \geq 0$.

Transforming the problem into its dual form may seem a bit over the top, after all we could find the minimum in the primal representation. Nonetheless, there is a substantial gain by using the dual, namely we can solve our problem by performing simple inner products of two vectors and mapping them into the real line $\mathbb{R}$. This is a very important result, and we will address it more generally in [Section 9.1.2](#).

Solving the optimisation problem in Equation (9.16) provides us with the values for the parameters α_i . We can then find the weights $\mathbf{w}$ with the aid of condition (9.14), whereas b can be obtained from a support vector such that $y_i = 1$, giving us the maximal margin hyperplane we were looking for. Finally, we can classify an unseen data point with features $\mathbf{x}$ by looking at the sign of $f(\mathbf{x})$.

Instead of solving the primal problem, we would like to solve the dual one.

In this context α is called the dual variable.

The Lagrangian in the dual representation.

The inner product maps two vectors in feature space K into the real line $\mathbb{R}$.

Given the dual variables α_i we can calculate $\mathbf{w}$ and b directly.

It is important to note that the vast majority of the weights will have α_i values equal to zero and that only the actual support vectors will survive. This is effectively a way of reducing the dimensionality of our problem. This reduction has been achieved in a more straightforward manner thanks to the application of the inner products in the dual formulation of our problem. In order to provide some intuition about the role of the inner product, we can appeal to the discussions we had in [Section 3.9](#).

The vast majority of the weights will have α_i values equal to zero, leaving only the support vectors.

The inner product can be used as a measure of the similarity between two vectors defined over an N -dimensional feature space. In a 2D space for example, if two vectors are parallel, then their inner product is 1, and we say that the vectors are *completely similar*. If the vectors are perpendicular, then their inner product is zero, and we say that they are *completely dissimilar*. This notion of similarity can be extended to higher-dimensional spaces without loss of generality.

This is the idea behind cosine similarity discussed in [Section 3.9](#).

With the above discussion in mind, in the case of Equation (9.16), if two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are dissimilar, they will not contribute to the value of L_D . However, if the two vectors are similar, we have two possible results. On the one hand, they can predict the same target value $y_i = \pm 1$. In this case, the value given by Equation (9.16) will be positive. Remember that we are trying to maximise L_D , and the minus sign attached to the first term of Equation (9.16) implies that the situation described above will decrease the value of the overall expression for L_D . This means that the algorithm reduces the importance of similar vectors that make the same prediction.

If two vectors are completely dissimilar, they do not contribute to the value of L_D .

The first term of Equation (9.16) contains the inner product.

On the other hand, if the vectors in question make opposite predictions about the target value y_i , but they are nonetheless similar, then the term containing the inner product is negative. This means that its contribution will increase the value of L_D . This situation helps maximising our objective function. Vectors like these are actually the examples we are interested in, as they are the ones of utmost importance to be able to discriminate our two classes.

One predicts $+1$, whereas the other one -1 .

These are the types of vectors we are interested in, as they help maximise the value of L_D .

9.1.2 *The Kernel Trick*

OUR DISCUSSION OF THE SUPPORT vector machine (SVM) algorithm has relied on the fact that the classes in our problem can be distinguished thanks to a linear boundary. We have seen how the constraint can be relaxed by including slack variables; however, the linear separability limitation remains. Although linear separability is a case of interest, we cannot leave aside the fact that there are many cases where a nonlinear boundary exists.

The application of the SVM algorithm is not confined to linearly separable classes.

In those cases we need the implementation of nonlinear support vector machines. The main idea is to obtain a linear boundary by mapping the data into a higher-dimensional space. That is indeed possible but we can come across a few issues. We may not be certain about the type of transformation that needs to be done to obtain linear separability. Even if we did know, the transformation might turn out to be computationally difficult and time-consuming.

The main aim is to obtain a linear boundary by mapping data into a higher-dimensional space.

To a certain extent, that is the issue we have already circumvented in the previous section by avoiding the optimisation problem in the primal form shown in Equation (9.11). Instead we used the dual formulation where the inner product enabled us to remap the vectors $\mathbf{x}_i$ into a representation where we did not have to carry out any calculations in the original feature space.

A suitable kernel enables us to carry out the required transformation letting us operate implicitly in the original space.

This is possible thanks to the application of a suitable kernel to carry out a transformation that lets us operate implicitly in the original feature space. This is what is known as **the kernel trick**.

A kernel is a function $K(x, y)$ whose arguments x and y can be real numbers, vectors, functions, etc. It is effectively a map between these arguments and a real value. The operation is independent of the order of the arguments. We are familiar with at least one such kernel: The well-known vector product. With $\mathbf{x}$ and $\mathbf{y}$ being two N -dimensional vectors, the inner product is given by:

This means that $K(x, y) = K(y, x)$.

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y} = \sum_{i=1}^N x_i y_i. \quad (9.17)$$

The inner product is a well-known kernel.

The kernel trick is a direct implementation of the Mercer theorem: With K being a mapping as defined above and a non-negative definite, symmetric continuous function, there exists a set of functions $\{\phi_i\}$ and a set of positive real numbers $\{\lambda_i\}$ with $i \in \mathbb{N}$ such that

$$K(u, v) = \sum_{i=1}^{\infty} \lambda_i \phi_i(u) \phi_i(v). \quad (9.18)$$

The Mercer theorem is the mathematics behind the kernel trick.

The Mercer theorem can be seen as an analogue of the singular value decomposition we discussed in [Section 8.3](#). In this case the kernel lives in an infinite-dimensional space. So, for a positive-definite symmetric matrix $\mathbf{A}$, we can define a linear operation such that when applied to a vector $\mathbf{x}$, it generates another vector $\mathbf{y}$:

$$\mathbf{Ax} = \mathbf{y}, \quad \text{or} \quad \sum_{m=1}^N a[m, n]x[m] = y[n]. \quad (9.19)$$

The Mercer theorem can be seen as an analogue of the singular value decomposition.

This is the usual matrix multiplication that we know and love.

The eigenvalues and corresponding eigenfunctions are defined as:

$$\sum_{m=1}^N a[m, n]\phi_i[m] = \lambda_i\phi_i[n]. \quad (9.20)$$

The eigenvalues are non-negative, and the eigenfunctions are orthonormal.

The eigenvalues are non-negative and the eigenfunctions are orthonormal. As such, the eigenfunctions that correspond to non-zero eigenvalues form a basis for the matrix $\mathbf{A}$, and we can therefore decompose it as:

$$a[m, n] = \sum_{i=1}^N \lambda_i \phi_i[m] \phi_i[n]. \quad (9.21)$$

We can decompose the matrix in this way.

The kernel trick implies that we do not need to compute, or even know, the functions ϕ_i . Instead, the kernel defines the appropriate inner products in the transformed space. In that way, it is possible to define nonlinear SVMs where the objective function can be written as:

The kernel trick implies that we do not need to compute, or even know, the functions ϕ_i .

$$L_D = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j K(x_i, x_j). \quad (9.22)$$

We have a choice of kernels to use. Some of the more popular ones include:

- Linear: $K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}$
- Polynomial: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$
- Gaussian: $K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)$
- Sigmoid: $K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x} \cdot \mathbf{y} - \delta)$

Here is a selection of some popular kernels that we can use.

We can certainly take our pick, but remember that more complicated models may not lead to good generalisation. Beware of overfitting.

9.1.3 SVM in Action: Regression

THE SUPPORT VECTOR MACHINE ALGORITHM can be applied in regression problems as a way to optimise the generalisation boundaries for the regression line. In this case the feature variables are first mapped onto the higher-dimensional space, and then a linear model is used.

SVM can be used in regression problems.

The generalisation performance of the SVM depends on the kernel used, but as with other regularised models, it also depends on the hyperparameter C introduced in Equation (9.9). This hyperparameter gives us the chance to fine-tune the model complexity: If C tends to infinity, then we are effectively optimising the model given only the data observed, and disregarding completely the complexity of the model.

Remember, it is all about the bias and variance trade-off.

Let us see how we can run SVM for regression using the mammals data we introduced in [Chapter 4](#). Let us recall

that the dataset looks at the relationship of the body mass of an animal and the mass of its brain. As before, let us start by importing some useful Python libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

We will now load the dataset into a pandas dataframe and sort it. We do this so that at a later stage our plot looks as we expect it and not like a bad piece of pseudo-modern art:

```
mammals = pd.read_csv(u'./Data/mammals.csv')\
    .sort_values('body')
```

Let us load the values of the required columns into appropriate variables for ease of manipulation:

```
body = mammals[ ['body'] ].values
brain = mammals['brain'].values
```

Please note that we have also created objects to hold the values of the quantities of interest. We are interested in comparing the result obtained using the well-known linear regression models we discussed in [Chapter 4](#) with the SVM algorithm. Let us load the relevant models from Scikit-learn:

```
from sklearn.linear_model import LinearRegression
from sklearn import svm
```

SVM contains methods for Support Vector Regression including SVR and LinearSVR. The former accepts as input

The data is available at <http://dx.doi.org/10.6084/m9.figshare.1565651>, as well as, <http://www.statsci.org/data/general/sleep.html>.

We sort the data with the method `.sort_values()`.

We will compare the results of the SVM with the simple linear regression model.

different kernels and the latter is similar to SVR with `kernel='linear'`. `LinearSVR` has more flexibility in the choice of penalties and loss functions. It is also worth mentioning that the default kernel for SVR is a Gaussian kernel, also known as a radial basis function (RBF) kernel. Let us instantiate a couple of SVR models with a fixed value for the hyperparameter `C`:

```
svm_lm = svm.SVR(kernel='linear', C=1e1)
svm_rbf = svm.SVR(kernel='rbf', C=1e1)
```

Scikit-learn has the following implementations for Support Vector Regression: `SVR` and `LinearSVR`.

We instantiate a model with a linear kernel and one with a Gaussian one.

The first model above is an SVM for regression with a linear kernel, whereas the second one has a Gaussian kernel. We can now train our models:

```
svm_lm.fit(np.log(body), np.log(brain))
svm_rbf.fit(np.log(body), np.log(brain))
```

We train our models with the log of the available variables.

For comparison, we will now instantiate and fit a regression model with logarithmic transformation:

```
logfit = LinearRegression().fit(np.log(body), \
    np.log(brain))
```

Please note that we can chain the methods.

The predictions of the models above can be easily obtained with the `predict` method for each of the models. In this case we are directly attaching them to the Pandas dataframe we started with:

We can obtain the predictions for all the trained models.

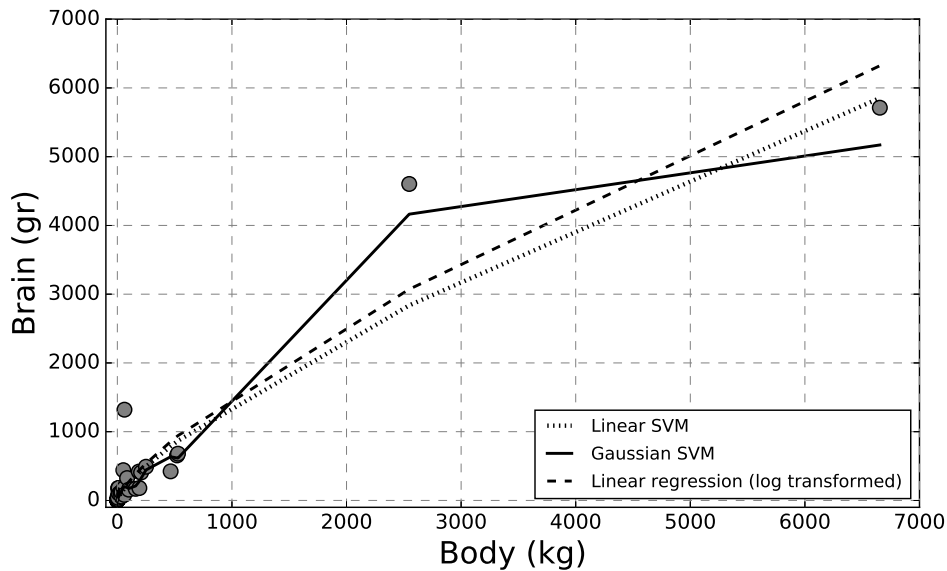


Figure 9.4: Comparison of the regression curves obtained using a linear model, and two SVM algorithms: one with a linear kernel and the other one with a Gaussian one.

```
mammals['log_regr'] = np.exp(logfit.\
    predict(np.log(body)))

mammals['linear_svm'] = np.exp(svm_lm.\
    predict(np.log(body)))

mammals['rbf_svm'] = np.exp(svm_rbf.\
    predict(np.log(body)))
```

The result of the three regression procedures applied to the mammals dataset can be seen in [Figure 9.5](#). Note that in this case the linear kernel performs no better and even worse than the simple linear model we saw in [Chapter 4](#). The Gaussian kernel did not require us to make any explicit transformations and, in this case, gets closer to the values

The Gaussian kernel seems to give better results than the other two models.

observed in the original dataset. Also note that close to the origin the Gaussian kernel produced some wiggles in the regression curve. Finally, please remember that overfitting is still an adversary that needs to be considered. We leave the tuning of the hyperparameter C as well as the implementation of cross-validation as an exercise for the reader.

9.1.4 SVM in Action: Classification

THE SECOND APPLICATION OF SVMs we will see is that of classification. As we saw in [Figure 9.3](#), the maximum margin hyperplane serves as a boundary between different classes. See Equation (9.2) for the mathematical expression of this statement.

Let us implement the SVM for classification using the wine data we encountered in [Chapter 5](#). The dataset can be found in the UCI Machine Learning Repository under “Wine Dataset”⁵ and is available at <http://archive.ics.uci.edu/ml/datasets/Wine>. Recall that the data records the results of chemical analysis of Italian wines grown in the same region from three different cultivars.

Let us start by loading the appropriate libraries into Python so that we can read the CSV file where the data is contained:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

⁵ Lichman, M. (2013a). UCI Machine Learning Repository, Wine Data. <https://archive.ics.uci.edu/ml/datasets/Wine>. University of California, Irvine, School of Information and Computer Sciences.

Using pandas, we will load the data into a dataframe called wine. The data is such that the target variable is contained in the same table and thus we separate the Cultivar feature into a target variable Y, leaving the rest of the features in the variable X:

```
wine = pd.read_csv(u'./Data/wine.csv')

X = wine.drop(['Cultivar'], axis=1).values
Y = wine['Cultivar'].values
```

The data is loaded into Pandas, and the target variable is separated from the rest of the dataset.

In the same way that we dealt with this dataset for clustering in [Section 5.2.2](#), we will only use the Alcohol and Colour Intensity features in our classification task. This will make our example simpler and easier to understand:

```
X1=wine[['Alcohol','Colour_Intensity']].values
```

We will only use the Alcohol and Colour Intensity features.

We can now split our data set into training and testing for cross-validation purposes:

```
import sklearn.model_selection as ms

XTrain, XTest, YTrain, YTest =\
    ms.train_test_split(X1, Y,\
        test_size= 0.3, random_state=7)
```

As good Jackalope Data Scientists, we are well-acquainted with creating training and testing datasets.

Scikit-learn provides implementations of Support Vector Classification including SVC and LinearSVC. As with the Support Vector Regression case, SVC takes as input different kernels, whereas LinearSVC is similar to SVC with

For classification, Scikit-learn has SVC and LinearSVC implementations.

kernel='linear'. Once again, as in the regression case, LinearSVC has more flexibility in the choice of penalties and loss functions. Finally, remember that the default kernel for SVC is a Gaussian one or RBF.

Remember that RBF stands for radial basis function.

Let us create an instance of the SVC classifier with a Gaussian kernel as follows:

```
from sklearn import svm
SVMclassifier = svm.SVC()
```

Remember that the default kernel for SVC is a Gaussian one.

We can use GridSearchCV in conjunction with the training set created above to find a suitable value for the hyperparameter C in our model. In this case we set up a dictionary for our search with values ranging between 0.5 and 2:

```
Cval = 2. ** np.arange(-1, 1.2, step=0.2)
n_grid = [{'C':Cval}]
```

We will search for the best hyperparameter C .

We can now set up our cross-validated grid search with an SVM for classification:

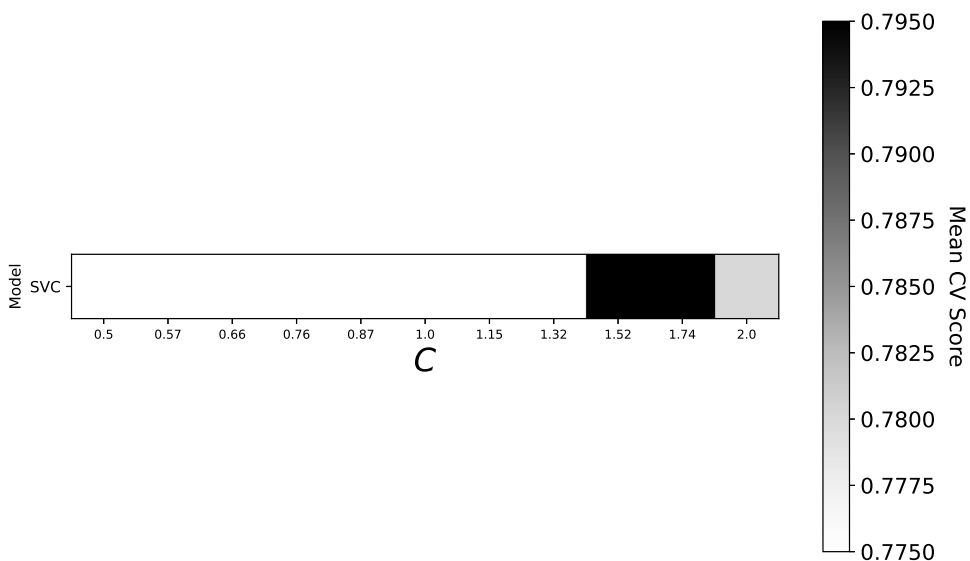
```
from sklearn.model_selection import GridSearchCV
cv_svc = GridSearchCV(estimator=SVMclassifier,\
    param_grid=n_grid,\
    cv=ms.KFold(n_splits=100))
```

We use GridSearchCV to find the best value for the hyperparameter. In this case we are using 100 folds.

Let us now apply the search to data coming from the training set we constructed above. We can also store the value of the best value for the parameter C for future use:

```
cv_svc.fit(XTrain, YTrain)

best_c = cv_svc.best_params_['C']
```



Let us see what the best parameter found is in this case:

```
> print('The best parameter is: C =', best_c)
```

```
The best parameter is: C= 1.5157165665103978
```

Figure 9.5: Heatmap of the mean cross-validation scores for the support vector machine algorithm with a Gaussian kernel for different values of the parameter C .

In [Figure 9.5](#) we can see a heatmap of the search performed with the different values for C . As we can see, for $C =$

1.5157 we find the best mean cross-validation score for the model. With this value it is now possible to construct a model to be used for training and testing:

```
svc_clf = svm.SVC(C=best_c)
svc_clf.fit(XTrain, YTrain)
```

We now create a model using the best parameter found and train it.

It is now possible to obtain the predictions provided by the model for the data points in the testing set. Let us take a look:

```
y_p = svc_clf.predict(XTest)
```

We obtain the predictions for the testing dataset.

For completeness, let us take a look at the classification report obtained for the testing dataset:

```
> from sklearn import metrics
> print(metrics.classification_report(y_p, YTest))
```

	precision	recall	f1-score	support
1	0.77	0.53	0.62	19
2	0.96	0.88	0.92	26
3	0.53	1.00	0.69	9
accuracy			0.78	54
macro avg	0.75	0.80	0.75	54
weighted avg	0.82	0.78	0.78	54

The classification report gives us information about the precision and recall among other things.

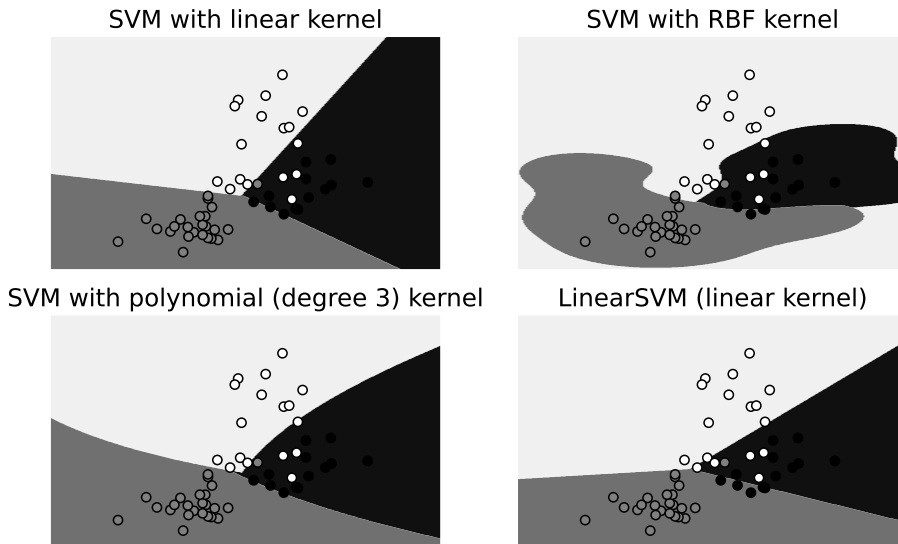
Using the same value for C obtained above, let us compare the classification boundaries obtained using different kernels:

```

C = best_c
svc = svm.SVC(kernel='linear', C=C).\
    fit(XTrain, YTrain)
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7,\
    C=C).fit(XTrain, YTrain)
poly_svc = svm.SVC(kernel='poly', degree=3,\
    C=C).fit(XTrain, YTrain)
lin_svc = svm.LinearSVC(C=C).fit(XTrain, YTrain)

```

We are creating SVM models with linear, Gaussian and polynomial kernels for comparison.



First, we are building a support vector machine (SVM) with a linear kernel with the SVC implementation. A second model has a Gaussian kernel with $\gamma=0.7$ and a third one with a cubic polynomial kernel ($\text{degree}=3$). Finally, with

Figure 9.6: Comparison of the classification boundaries obtained using support vector machine algorithms with different implementations: SVC with a linear, Gaussian and degree-3 polynomial kernels, and LinearSVC.

LinearSVC we create a model using an alternative linear kernel implementation. A comparison of these models is shown [Figure 9.6](#). We can see how the boundaries created by the linear kernels are given by straight lines, whereas the nonlinear ones provide us with more intricate boundaries.

Note that the boundaries for linear kernels are lines, whereas the ones for nonlinear kernels are more intricate.

The plots in [Figure 9.6](#) also show the data points in the testing set coloured by the class to which they belong. We can see the distribution of misclassified points given by each of the four models used. It is clear that we can use more features than the two chosen in this example. This will make the visualisation step more challenging than the 2D case shown here.

9.1.5 Support Vector Machines — Use Cases

SUPPORT VECTOR MACHINES (SVMs) ARE like the Swiss Army knives of machine learning—versatile, reliable, and robust. While they shine in various domains, their application in Natural Language Processing (NLP) is particularly fascinating. They can be used for sentiment analysis, text classification, and named entity recognition (NER), along with the considerations for handling textual data.

SVMs are like the Swiss Army knives of machine learning—versatile, reliable, and robust.

Sentiment Analysis: The Mood Detector

Sentiment analysis is akin to teaching machines to understand human emotions. By analysing text, we can determine whether the sentiment expressed is positive, negative, or neutral. Here is where SVMs come into play.

Sentiment analysis is like teaching machines to understand human emotions.

SVMs excel at classification tasks, and sentiment analysis is fundamentally a classification problem. By training an SVM model on labelled data (e.g., movie reviews marked as positive or negative), the model learns to distinguish between different sentiments. Feature extraction is crucial here. Common techniques include⁶:

- Bag of Words (BoW): Representing text by the frequency of words.
- TF-IDF (Term Frequency-Inverse Document Frequency): Weighing words based on their importance.
- Word Embeddings: Capturing semantic meaning by representing words in continuous vector spaces.

Text Classification: The Organiser

Text classification is like sorting a vast library into neat categories. It involves assigning pre-defined labels to text documents based on their content. SVMs, with their prowess in handling high-dimensional data, are well-suited for this task.

Consider an email spam filter. By training an SVM on a dataset of labelled emails (spam or ham), it learns to classify new emails accurately. The features extracted from the text (such as specific keywords, frequency of certain terms, etc.) serve as the input for the SVM.

Another example is news categorisation. News articles can be classified into categories like sports, politics, entertainment, and technology using SVMs. Here, the

Sentiment analysis is fundamentally a classification problem.

⁶ Rogel-Salazar, J. (2020). *Advanced Data Science and Analytics with Python*. Chapman & Hall/CRC data mining & knowledge discovery series. CRC Press, Taylor & Francis Group.

Text classification involves assigning pre-defined labels to text documents based on their content.

This is similar to the task covered in [Section 6.4.2](#).

challenge lies in feature selection and ensuring that the model captures the nuances of each category.

Named Entity Recognition: The Information Extractor

Named Entity Recognition (NER) involves identifying and classifying entities (such as names of people, organisations, locations) in text. It is like finding the hidden treasures in a sea of words. SVMs can be effectively used for NER by treating it as a sequence labeling problem.

In name entity recognition, the text is systematically broken down into smaller units known as tokens (e.g., individual words)⁷, and the support vector machine model predicts the label for each token (e.g., whether a word is a person's name, a location, etc.). Features for NER might include the word itself, its part-of-speech (POS) tag, neighbouring words, contextual or lexical information, and more.

For example, in a sentence like "Apple Inc. is launching a new product in Cupertino", the model identifies "Apple Inc." as an organisation and "Cupertino" as a location. Training such models involves using labelled datasets where the entities are annotated.

Here are some considerations for using SVMs with textual data:

- **Feature Extraction:** Unlike numerical data, text data requires careful feature extraction. Techniques like BoW, TF-IDF, and word embeddings (e.g., Word2Vec, GloVe) are essential. The choice of features can significantly impact the model's performance.

Named entity recognition involves identifying and classifying entities in text.

⁷ Rogel-Salazar, J. (2020). *Advanced Data Science and Analytics with Python*. Chapman & Hall/CRC data mining & knowledge discovery series. CRC Press, Taylor & Francis Group.

Annotation is an important part of the text pre-processing.

The choice of features can significantly impact the model's performance.

- **High Dimensionality:** Text data often leads to high-dimensional feature spaces, especially with techniques like BoW and TF-IDF. SVMs handle high dimensions well, but it is crucial to manage computational complexity and avoid overfitting.
- **Data Pre-processing:** Text data requires extensive pre-processing – tokenisation, stemming, lemmatisation, removing stop words, and handling punctuation. Proper pre-processing ensures that the features fed into the SVM are meaningful.
- **Imbalanced Data:** In tasks like sentiment analysis, the dataset might be imbalanced (e.g., more positive reviews than negative). Techniques such as oversampling, undersampling, or using different class weights can help balance the data.
- **Kernel Selection:** SVMs offer different kernels (linear, polynomial, RBF) to map data into higher dimensions where it becomes separable. Selecting the appropriate kernel based on the data and problem is crucial for performance.
- **Interpretability:** While SVMs are generally more interpretable than deep learning models, the decision boundary in high-dimensional space can still be complex. Tools and techniques to explain model decisions (like SHAP) are valuable for transparency.

Text data often leads to high-dimensional feature spaces.

Text data requires extensive pre-processing.

See [Section 6.5](#) for more information on dealing with imbalanced datasets.

See [Section 9.1.2](#) for more information on kernels.

See [Section 7.5](#) about explainability.

- Scalability: SVMs can be computationally intensive, especially with large textual datasets. Ensuring scalability through efficient data handling and optimisation techniques is important.

Be mindful of the computational complexity of your models.

SVMs are potent tools in the NLP toolkit, adept at tasks ranging from sentiment analysis to text classification. However, their effective application requires meticulous feature extraction, thoughtful handling of high-dimensional data, and attention to imbalanced datasets. By embracing these considerations, SVMs can unlock the rich potential of textual data, transforming it into actionable insights and knowledge. So, whether you are a Jackalope Data Scientist deciphering sentiments, categorising content, or unearthing named entities, SVMs stand ready as your reliable companions in the ever-evolving landscape of NLP.

Embracing these considerations can help you harness the rich potential of textual data.

9.2 *Transparency, Fairness and Beyond*

YOU HAVE BEEN LUCKY ENOUGH to be invited to a performance by The Great Howdini. In the first act, Howdini pulls a rabbit out of a hat, and everyone, including yourself, claps in amazement. But what if you wanted to know how the trick was done? Howdini will never let you know. That is not the situation we want out of the usage of our models. Just like wanting to understand the magician's sleight of hand, transparency in machine learning means understanding how decisions are made by these sophisticated algorithms.

Mind you, Sheldon may spoil the fun by telling you.

SVMs, with their geometric elegance, draw a hyperplane that separates data points in a high-dimensional space. This hyperplane is the magical dividing line between different classes. While the math is exquisite, the process can often be a black box to the layperson, and hence the need for transparency .

SVMs are elegant, but may be seen as magic boxes.

Transparency is about making the decision-making process understandable. For instance, if an SVM model is used in a hiring process, candidates have the right to know how their CVs were assessed. Did the algorithm give extra weight to certain skills? Did it penalise gaps in employment? The goal here is not to reveal trade secrets but to ensure that the process is fair and understandable.

Transparency makes the decision-making process understandable.

However, achieving transparency is complex, but it is necessary. We discussed SHAP in [Section 7.5.1](#) as a way for people to understand the decisions made in specific instances. This brings us to our next ethical topic: Fairness. Imagine now the Great Howdini attempting a tightrope act, balancing delicately over the abyss. In the realm of SVMs, and other machine learning models, fairness means ensuring that the model does not favour one group over another unjustly. It is about creating a level playing field, where everyone has an equal shot at the prize.

Fairness is about creating a level playing field.

Consider the use of SVMs in lending decisions. If an SVM model is used to determine creditworthiness, fairness dictates that it should not discriminate based on race, gender, or socioeconomic status. But here is the thing: Data itself can be biased. If historical lending data reflects societal

Fairness dictates that a model should not discriminate.

prejudices, an SVM trained on this data may perpetuate those biases. The model might learn that certain postcodes correlate with higher default rates without considering the socioeconomic factors at play.

To tackle this, ethical SVM deployment involves techniques like data pre-processing to remove bias, fairness constraints during model training, and post-processing adjustments to ensure fair outcomes. This is akin to our tightrope walker using a balancing pole – it helps maintain equilibrium. Consider the following example: In healthcare, SVMs can assist in diagnosing diseases. But if the model is trained predominantly on data from one demographic, it might not perform well on others, leading to misdiagnoses. Ensuring diverse and representative training data is crucial here. It is like having a diverse team of doctors to cater to different patient needs.

Ensuring diverse and representative training data is crucial.

It is clear that transparency and fairness are not just nice to have; they are essential. They guide us like a compass, ensuring that our technological advancements benefit society as a whole and do not entrench existing inequities. Achieving transparency is about demystifying the magic, making the decision-making process understandable and accessible. Fairness, on the other hand, ensures that everyone gets a fair shake, preventing the perpetuation of historical biases.

Transparency and fairness are not just nice to have; they are essential.

I encourage us all to embrace these ethical considerations with a spirit of curiosity and responsibility. After all, the use of artificial intelligence and machine learning is not about

pulling rabbits out of hats, but ensuring that the magic serves everyone equally.

9.2.1 *Getting Support — Generative AI*

ADVANCES IN GENERATIVE AI, AS discussed in [Section 3.2.2](#), have captured widespread attention, but Gen AI alone is not sufficient. Support Vector Machines (SVMs) are highly versatile models that, despite having distinct purposes and different underlying principles from Gen AI, can complement each other. Their capabilities can intersect in practical applications, enhancing the overall effectiveness of AI systems.

On the one hand, as we know from this chapter, SVMs are supervised learning models primarily used for classification and regression tasks. The main idea behind SVMs is to find the optimal hyperplane that separates data points of different classes in a high-dimensional space. SVMs are powerful for their ability to handle high-dimensional data and their effectiveness in scenarios with clear margins of separation between classes.

Gen AI on the other hand refers to models that can generate new data instances similar to the training data. These models learn the underlying distribution of the input data and can create new content, such as images, text, or audio. Examples of generative models include Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and autoregressive models like Generative Pre-trained Transformer (GPT).

Gen AI in combination with other models can enhance and complement each other.

SVMs are powerful for their ability to handle high-dimensional data.

Gen AI can generate new data instances similar to the training data.

These models, like many others, can be used in a complementary manner. For example as discussed in [Section 8.5](#), sometimes it is advantageous to create synthetic data for training models. Gen AI can be used for this purpose. By generating additional labelled data that closely resembles the real data, Gen AI can help enhance the performance of classifiers. Another collaborative effort of these two approaches is in feature extraction: Gen AI models, such as autoencoders, can be used to learn compact and meaningful representations of data. These representations (features) can then be fed into SVMs for classification or regression tasks. For example, an autoencoder could reduce the dimensionality of image data, and the resulting feature vectors could be classified using an SVM. There are a number of practical applications of these two approaches, including:

- **Anomaly Detection:** In anomaly detection, generative models can be trained to capture the normal behaviour of data. Anomalies can be detected by evaluating how well new data points fit into this learned distribution. SVMs can be used alongside to classify data points as normal or anomalous based on the features extracted by generative models.
- **Natural Language Processing (NLP):** In NLP, generative models like GPT can be used to generate text data, which can then be classified using SVMs. For example, GPT can generate various text samples for different sentiment classes, and an SVM can be trained to classify sentiments based on these samples.

Gen AI can complement SVMs by generating synthetic training data.

Gen AI can help extracting meaningful features, thereby enhancing SVM performance.

Generative models capture normal data behaviour, and SVMs classify new data points as normal or anomalous based on extracted features.

Generative models create text data for different sentiment classes, which SVMs then classify by sentiment.

- **Image Recognition and Generation:** Gen AI can be used to generate realistic images, which can be used to augment training datasets for image recognition tasks. SVMs can then be used to classify these images. This is particularly useful in medical imaging, where obtaining labeled data can be challenging and expensive.

Gen AI generates realistic images to augment training datasets, and SVMs classify these images.

As discussed in [Section 7.5](#), there are challenges that we need to take into account including:

- **Quality of Generated Data:** The effectiveness of using generative models to augment SVM training depends on the quality of the generated data. Poorly generated data can lead to overfitting or misclassification. Ensuring that generative models produce high-quality, realistic data is crucial.
- **Computational Resources:** Training generative models, especially deep learning-based models like GANs, requires significant computational resources. Combining these models with SVMs can be computationally intensive, necessitating efficient resource management.
- **Integration Complexity:** Integrating generative models with SVMs can add complexity to the model pipeline. It requires expertise in both generative and discriminative modelling, as well as a thorough understanding of how to combine these models effectively.

Poorly generated data can lead to overfitting or misclassification.

Training generative models requires significant computational resources.

Be mindful of adding complexity to your pipeline.

The relationship between Gen AI and other machine learning models, including SVMs is a symbiotic one, where each can enhance the capabilities of the other. While SVMs

Think of a lichen, or the Trill.

excel at classification tasks with clear decision boundaries, Gen AI models bring the ability to generate and model complex data distributions. By leveraging the strengths of both, more robust and effective machine learning systems can be built. Whether it is through data augmentation or feature extraction, the intersection of SVMs and Gen AI holds promise for advancing various fields of artificial intelligence.

We can build more robust and effective machine learning models with this symbiosis.

9.3 Summary

THE KERNEL TRICK IS A useful tool to consider alongside other techniques such as feature selection and dimensionality reduction to tackle any machine learning problem that we encounter.

In this chapter we saw how appropriate mappings into higher-dimensional feature spaces can be used to make regression or classification possible without actually having to carry out explicitly any calculations in the high-dimensional space. This is possible thanks to the so-called *kernel trick*. We applied this trick in the implementation of the Support Vector Machine (SVM) algorithm.

A support vector machine is a binary linear classifier where the classification boundary is constructed so as to minimise the generalisation error in our learning task. The main idea behind SVM is that data may be linearly separable in the high-dimensional space, but not linearly separable in the

original feature space. The use of different kernels provides us with the flexibility to find nonlinear boundaries that separate the different classes in our problem as demonstrated in this chapter.

SVMs are versatile, reliable models. We discussed some use cases in Natural Language Processing (NLP) tasks like sentiment analysis, text classification, and named entity recognition (NER). In sentiment analysis, SVMs classify text as positive, negative, or neutral by analysing labelled data using techniques like Bag of Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), and word embeddings. For text classification, SVMs sort documents into categories, such as spam detection in emails or news categorisation, by extracting relevant features. In NER, SVMs identify and classify entities within text, facilitating the extraction of meaningful information from sentences.

Transparency in machine learning involves understanding how algorithms make decisions, ensuring fairness by clarifying model assessments. For instance, in hiring processes, candidates should know how their CVs were evaluated. Fairness ensures models do not favour one group over another.

Generative AI (Gen AI) models create new data similar to training data, complementing SVMs in practical applications. Gen AI can generate additional labeled data to enhance SVM classifiers and extract meaningful features for SVM classification. Despite challenges like data quality,

computational demands, and integration complexity, the synergy between SVMs and Gen AI promises advancements in artificial intelligence, leveraging their combined strengths for robust machine learning systems.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Pipelines in Scikit-Learn

THE TERM PIPELINE IS USED to describe a series of ordered concatenated data transformations and manipulations. The order of the transformations is important, as each phase of a pipeline feeds from the previous one: The outcome of any given step serves as the input for the next one as data flows through the pipeline from beginning to end.

Data pipelines are very useful, as they are effectively automated transformations used to perform routine data maintenance and analysis tasks, ensuring the validity of the output data to be fed to the next stage of the workflow. For instance in a required workflow it may be necessary to ensure that the data has the correct units and is scaled before imputing any missing values and is ready to be used for training a particular algorithm.

It is possible to implement pipelines in Scikit-learn, helping us improve our code and manage our models. A pipeline can be used to amalgamate all the steps we may need to prepare our data and make appropriate predictions. In [Section 8.2.2](#) we applied a pipeline to reduce the

A pipeline is a series of ordered concatenated data transformations and manipulations.

Data pipelines help us manage our workflow by defining the steps we need to take.

We discussed an example in [Section 8.2.2](#).

dimensionality of the Iris dataset with principal component analysis and then use the result in a logistic regression.

Let us see another example implementing a pipeline for a LASSO regression using the Iris dataset we first encountered in in [Chapter 3](#).

Let us load the dataset:

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris['data']
Y = iris['target']
names = iris['target_names']
```

The Iris dataset can be loaded from `load_iris`.

We will build a pipeline that takes into account two main steps, one to standardise the variables as described in [Section 4.6](#) and then use the result in a LASSO model. We will use the pipeline to search for the appropriate hyperparameter for the model and finally use the parameter found to train the model and score it.

Our pipeline will consist of a standardisation step and a LASSO model.

Let us first load some useful modules: We will use `preprocessing` to standardise our variables, `cv` to create our training and testing partitions, `Lasso` for modelling our data, `GridSearchCV` to search for the optimal hyperparameter and `Pipeline` to construct our pipeline.

```
from sklearn import preprocessing
import sklearn.model_selection as ms
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

These are the modules and methods we will use in this analysis.

We need to create instances for the steps that will be included in our pipeline. In this case standardisation of our variables with Scikit-learn's StandardScaler and a LASSO model.

```
std_scaler = preprocessing.StandardScaler()
lasso1 = Lasso()
```

We need to instantiate the steps to be used in our pipeline.

We can now define our pipeline. We have to provide a label to each of our steps so that we can refer to them later on in the process. Let us call the standardisation step `scaling` and the model `mylasso`:

```
pipe = Pipeline(steps=[('scaling', std_scaler ),\
                        ('mylasso', lasso1)])
```

Note that we are providing a label to each of the steps in our pipeline.

We need to partition our data into training and testing:

```
XTrain, XTest, yTrain, yTest =\
    ms.train_test_split(X, Y,\
                        test_size=0.2, random_state=1)
```

As usual we partition our data into training and testing.

Let us define a set of values to search for the hyperparameter.

```
lambda_range = linspace(0.001, 0.5, 250)
```

We will perform a search over a range of values for the model's hyperparameter.

We can now pass our pipeline to the exhaustive search module and fit the search with the training data as follows:

```

cv_lasso = GridSearchCV(pipe,\
    dict(mylasso__alpha=lambda_range),\
    cv=ms.KFold(n_splits=100))

cv_lasso.fit(XTrain, yTrain)

```

We pass our pipeline as the model to be used by GridSearchCV. We then execute the search over the training dataset.

The grid search will take the raw training data and put it through the pipeline: First, it calls the standardisation process and the result will be fed to the LASSO model for each of the hyperparameter values for cross-validation. We can take a look at the best parameter obtained from this process as follows:

```

bestLambda=cv_lasso.best_params_['mylasso__alpha']
print(bestLambda)

0.001

```

The result of the hyperparameter search is prefixed by the label provided when defining our pipeline.

Please note that we are referring to the parameter from the LASSO model by prefixing it with the label given to the pipeline followed by a double underscore.

Now that we have found the optimal hyperparameter, we can set it as a parameter in the pipeline with the help of the `set_params` method:

```

pipe.set_params(mylasso__alpha=bestLambda)

```

We can pass the new parameter to the pipe with `set_params`.

We are now in a position to train our model:

```
> IrisLassoModel = pipe.fit(XTrain, yTrain)
```

The pipeline can be directly used to train our final model.

Let us take a look at the coefficients we obtained:

```
> IrisLassoModel.named_steps['mylasso'].coef_  
  
array([-0.08441169, -0.02992676,  
       0.42770198,  0.41570303])
```

Note that a few of the attributes have coefficients equal to zero.

Remember that shrinking with LASSO may result in obtaining some coefficients close to zero. You can take a look at the details in [Section 4.9](#).

Finally, let us score the model and create predictions for the testing partition:

```
> IrisLassoModel.score(XTest, yTest)  
  
0.94704726937064513  
  
> Iris_Prediction = IrisLassoModel.predict(XTest)
```

Finally we can score and obtain predictions from our model.
Et voilà!



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Bibliography

AfDSP. (2023, Feb). Alliance for Data Science Professionals.
alliancefordatascienceprofessionals.co.uk.

Allison, T. and D. V. Cicchetti (1976, Nov 12). Sleep in mammals: ecological and constitutional correlates. *Science*. 194, 732–734.

Anaconda (2024). Anaconda 3.0. <https://www.anaconda.com/download/>.

Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions*. 53, 370–418.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. Rand Corporation. Research studies. Princeton U.P.

Bertsekas, D. (1996). *Constrained Optimization and Lagrange Multiplier Methods*. Athena scientific series in optimization and neural computation. Athena Scientific.

Borges, J. L. (1984). *El Libro de Arena*. El Ave Fénix. Plaza & Janés.

Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. In *5th Annual*

ACM Workshop on COLT, Pittsburgh, PA, pp. 144–152.
ACM Press.

Breiman, L. (1996). Bagging predictors. *Machine Learning*. 24(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning*. 45(1), 5–32.

Cardoso, M. (2013). UCI Machine Learning Repository, Wholesale Customers. <https://archive.ics.uci.edu/dataset/292/wholesale+customers>. University of California, Irvine, School of Information and Computer Sciences.

Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*. 16, 321–357.

Cole, S. (2004). History of fingerprint pattern recognition. In N. Ratha and R. Bolle (Eds.), *Automatic Fingerprint Recognition Systems*, pp. 1–25. Springer New York.

Cortes, C. and V. Vapnik (1995). Support vector networks. *Machine Learning*. 20, 273–297.

Cover, T. M. (1969, Jan). Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory*. IT-13, 21–27.

Devlin, K. (2010). *The Unfinished Game: Pascal, Fermat, and the Seventeenth-Century Letter That Made the World Modern*. Basic ideas. Basic Books.

- DLMF (2015). NIST Digital Library of Mathematical Functions. <http://dlmf.nist.gov/>, Release 1.0.10 of 2015-08-07.
- Downey, A. (2012). *Think Python*. O'Reilly Media.
- Duffy, F. H. et al. Unrestricted principal components analysis of brain electrical activity: Issues of data dimensionality, artifact, and utility. *Brain Topography*. 4(4), 291–307.
- Ester, M., H.-P. Kriegel, J. Sander, and X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pp. 226–231. AAAI Press.
- European Commission (2021, Apr). Proposal for a Regulation laying down harmonised rules on Artificial Intelligence. <https://digital-strategy.ec.europa.eu/en/library/proposal-regulation-laying-down-harmonised-rules-artificial-intelligence>.
- Eysenck, M. and M. Keane (2000). *Cognitive Psychology: A Student's Handbook*. Psychology Press.
- Fan, R. E., K. W. Chang, C. J. Hsieh, X. R. Wang, and C. Lin (2008). LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*. 9, 1871–1874.
- Farris, J. S. (1969). On the cophenetic correlation coefficient. *Systematic Biology* 18(3), 279–285.
- Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*. 27, 861–874.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7(2), 179–188.

Fold-it. Solve puzzles for science. <https://fold.it/>.

Freund, Y. and R. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comp. and Sys. Sciences*. 55(1), 119–139.

Galati, G. (2015). *100 Years of Radar*. Springer International Publishing.

Galton, F. (1886). Regression Towards Mediocrity in Hereditary Stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*. 15, 246–263.

Galton, F. (1907). Vox populi. *Nature*. 75(1949), 450–451.

Geurts, P., D. Ernst, and L. Wehenkel (2006). Extremely randomized trees. *Machine Learning*. 63, 3–42.

Gilder, J. and A. Gilder (2005). *Heavenly Intrigue: Johannes Kepler, Tycho Brahe, and the Murder Behind One of History's Greatest Scientific Discoveries*. Knopf Doubleday Publishing Group.

Gobble, M. M. (2019). The Road to Artificial General Intelligence. *Research-Technology Management*. 62(3), 55–59.

Goertzel, B. (2023). Generative AI vs. AGI: The Cognitive Strengths and Weaknesses of Modern LLMs. *arXiv preprint arXiv:2309.10371*.

Golub, G. and C. Van Loan (2013). *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press.

- Harrison, D. and D. Rubinfeld (1978). Hedonic prices and the demand for clean air. *J. Environ. Economics and Management*, 81–102.
- Hilbert, D. (1904). Grundzüge einer allgemeinen Theorie der linearen Integralrechnungen. (Erste Mitteilung). *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse.*, 49–91.
- Hoerl, A. E. and R. W. Kennard (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*. 12(3), 55–67.
- Hu, Y., Y. Koren, and C. Volinsky (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, ICDM '08*, Washington, DC, USA, pp. 263–272. IEEE Computer Society.
- Hunt, E. B., J. Marin, and P. J. Stone (1966). *Experiments in Induction*. New York: Academic Press.
- Kaggle (2012). Titanic: Machine Learning from Disaster. <https://www.kaggle.com/c/titanic>.
- Langtangen, H. (2014). *A Primer on Scientific Programming with Python*. Texts in Computational Science and Engineering. Springer Berlin Heidelberg.
- Laplace, P. and A. Dale (2012). *Pierre-Simon Laplace Philosophical Essay on Probabilities: Translated from the fifth French edition of 1825 With Notes by the Translator*. Sources in the History of Mathematics and Physical Sciences. Springer New York.

Le, Q. V., R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean, and A. Y. Ng (2011). Building high-level features using large scale unsupervised learning.

CoRR. abs/1112.6209.

Lehren, A. W. and Baker, A. (2009, Jun 18th). In New York, Number of Killings Rises With Heat. *The New York Times*..

Liao, Q. Vera and Wortman Vaughan, Jennifer (2024, Feb 29). AI Transparency in the Age of LLMs: A Human-Centered Research Roadmap. *Harvard Data Science Review*..

<https://hdsr.mitpress.mit.edu/pub/aelql9qy>.

Lichman, M. (2013a). UCI Machine Learning Repository, Wine Data. <https://archive.ics.uci.edu/ml/datasets/Wine>. University of California, Irvine, School of Information and Computer Sciences.

Lichman, M. (2013b). UCI Machine Learning Repository, Wisconsin Breast Cancer Database. [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Original\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)). University of California, Irvine, School of Information and Computer Sciences.

Lima, M. (2011). *Visual Complexity: Mapping Patterns of Information*. Princeton Architectural Press.

Lima, M. and B. Shneiderman (2014). *The Book of Trees: Visualizing Branches of Knowledge*. Princeton Architectural Press.

Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*. 45(1), 503–528.

- Lohr, S. (2014, Aug 17th). For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights. *The New York Times*..
- Lundberg, S. M. and S.-I. Lee (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, Volume 30. Curran Associates, Inc.
- MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press.
- Mangasarian, O. L. and W. H. Wolberg (1990, Sep.). Cancer diagnosis via linear programming. *SIAM News*. 25(5), 1 & 18.
- Martin, D. (2003, Jan 19th). Douglas Herrick, 82, Dies; Father of West's Jackalope. *The New York Times*..
- McCandless, D. (2009). *Information is Beautiful*. Collins.
- McGrayne, S. (2011). *The Theory that Would Not Die: How Bayes' Rule Cracked the Enigma Code, Hunted Down Russian Submarines, & Emerged Triumphant from Two Centuries of Controversy*. Yale University Press.
- McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media.
- Milligan, Glenn W. and Cooper, Martha C. (1988). A study of standardization of variables in cluster analysis. *Journal of Classification* 5(2), 181–204.

- Pearson, K (1904). On the theory of contingency and its relation to association and normal correlation. In *Mathematical contributions to the theory of evolution*. London, UK: Dulau and Co.
- Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*. 12, 2825–2830.
- Python Software Foundation (1995). Python reference manual. <http://www.python.org>.
- R Core Team (2014). R: A language and environment for statistical computing. <http://www.R-project.org>.
- Rogel-Salazar, J. (2014). *Essential MATLAB and Octave*. Taylor & Francis.
- Rogel-Salazar, J. (2016a, Jan). Data Science Tweets. <https://dx.doi.org/10.6084/m9.figshare.2062551.v1>.
- Rogel-Salazar, J. (2016b, Jan). Jackalope Image. <https://dx.doi.org/10.6084/m9.figshare.2067186.v1>.
- Rogel-Salazar, J. (2020). *Advanced Data Science and Analytics with Python*. Chapman & Hall/CRC data mining & knowledge discovery series. CRC Press, Taylor & Francis Group.
- Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. Chapman & Hall/CRC The Python Series. CRC Press.
- Rogel-Salazar, J. (2024, May). Weight Data. <https://doi.org/10.6084/m9.figshare.25820893.v1>.

- Rogel-Salazar, J. and N. Sapsford (2014). Seasonal effects in natural gas prices and the impact of the economic recession. *Wilmott*. 2014(74), 74–81.
- Rousseeuw, P. J. (1987). Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Comp. and App. Mathematics*. 20, 53–65.
- Scientific Computing Tools for Python (2013). NumPy. <http://www.numpy.org>.
- Takács, G. and D. Tikk (2012). Alternating least squares for personalized ranking. In *Proceedings of the Sixth ACM Conference on Recommender Systems, RecSys '12*, New York, NY, USA, pp. 83–90. ACM.
- Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *J. R. Statist. Soc. B*. 58(1), 267–288.
- Toelken, B. (2013). *The Dynamics of Folklore*. University Press of Colorado.
- Töscher, A. and M. Jahrer (2009). The BigChaos solution to the Netflix grand prize. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf.
- Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*. 42(2), 230–265.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*. 59, 433–460.
- Weir, A. (2014). *The Martian: A Novel*. Crown/Archetype.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*. 5(2), 241–259.

Zimmer, C. (2012). *Rabbits with Horns and Other Astounding Viruses*. Chicago Shorts. University of Chicago Press.

Zingg, R., J. Fikes, P. Weigand, and C. de Weigand (2004). *Huichol Mythology*. University of Arizona Press.

Zooniverse. Projects. <https://www.zooniverse.org/projects>.

Index

- AdaBoost Classifier, [351](#)
- Agglomerative clustering, [314](#)
- Algorithm bias, [302](#)
- Amazon, [5](#)
- Amazon Kinesis, [23](#)
- Analysis of variance, [317](#)
- Anomaly detection, [445](#)
- ANOVA, [317](#)
- Apache Airflow, [24](#)
- Apache Cassandra, [24](#)
- Apache Oozie, [23](#)
- Apache Spark, [26](#)
- Area-under-the-curve, [262](#)
- Arithmetic operators, [46](#)
- Artificial intelligence, [2–4](#), [101](#), [104](#)
 - General, [105](#)
 - Generative, [107](#), [108](#)
 - Narrow, [106](#)
- AUC, [262](#)
- Azkaban, [23](#)
- Backslash, [49](#)
- Bagging
 - Resampling, [344](#)
- Bagging Classifier, [351](#)
- Bayes' theorem, [288](#), [291](#)
- Bayes, Thomas, [288](#)
- Bayesian analysis, [287](#)
- Bayesian statistics, [292](#)
- Bernoulli distribution, [275](#)
- Bias, [135](#)
- Bias-variance balance, [135](#), [207](#)
- Big data, [8](#)
- Borges, Jorge Luis, [414](#)
- Bowman, [132](#)
- Brahe, Tycho, [103](#)
- Business acumen, [9](#)
- Business intelligence, [90](#)
- Categorical features, [130](#)
- Causation, [167](#)
- Centroids, [228](#)
- Character encoding, [295](#)
- Chemistry, [102](#)
- Citizen science, [338](#)
- Classification, [138](#), [144](#), [255](#), [256](#)
 - F_1 score, [272](#)
- KNN, [265](#)
- Logistic regression, [272](#)
- Clustering, [27](#), [225](#), [226](#)
 - Cohesion, [231](#)
 - DBSCAN, [241](#), [247](#)
 - K-means, [227](#)
 - Separation, [231](#)
 - Silhouette, [233](#)
 - Validation, [231](#)
- Code readability, [49](#)
- Coefficient of determination, [182](#)
- Cognitive psychology, [102](#)
- Cold start problem, [396](#), [403](#)
- Collaborative filtering, [124](#)
 - Item-based, [397](#)
 - Memory-based, [397](#)
 - Model-based, [397](#)
- Communication skills, [9](#), [126](#)
- Complex numbers, [49](#)
- Computer science, [9](#)
- Confounding variable, [167](#)
- Confusion matrix, [258](#)
- Constraints and assumptions, [8](#)
- Control flow

- For loop, 63
- If statement, 61
- While loop, 62
- Convex objective function, 416
- Corpus, 293
- Correlation coefficient, 166
- Cosine similarity, 142, 392, 423
- Cottontail rabbit papilloma virus, 13
- Cross-validation, 133, 137, 149, 157, 209
 - k-fold, 158, 209
 - Leave-one-out, 159
 - Leave-p-out, 160
- Curse of dimensionality, 143, 366
- D₃, 21
- Darwin, Charles, 168
- Data
 - Availability, 8
 - Timeliness, 8
- Data science
 - Academia and Business, 3
- Data compression, 367
- Data pre-processing, 369
- Data science, 2, 3, 131
 - Advertising and marketing, 6
 - Classification analysis, 7
 - Clustering analysis and market segmentation, 6
 - Cybersecurity, 6
 - Definition, 4
 - Demand forecasting, 7
 - E-commerce, 6
 - Examples, 6
 - Fraud prevention, 7
 - Market basket analysis, 6
 - Online services, 6
 - Predictive analytics, 6
 - Python, 40
 - Recommendation systems, 6
 - Security, 22
 - Social media analysis, 6
 - Team, 1
 - Workflow, 27
 - Data acquisition, 30
 - Data munging, 30
 - Modelling and evaluation, 31
 - Question identification, 29
 - Representation and interaction, 32
- Data science tools, 19, 21
 - Data framework, 22
 - Job scheduling, 23
 - NoSQL databases, 24
 - Open-source, 25
 - Streaming data collection, 23
 - Technology, 22
 - Workflow management, 24
- Data science workflow, 132
- Data scientist, 9
 - Characteristics, 15, 16
 - Data architect, 18
 - Data Engineer, 18
 - Definition, 9
 - Jackalope, 9
 - Lead data scientist, 18
 - Project manager, 17
 - Triumvirate, 17
 - Unicorn, 11
- Data stratification, 154
- Data visualisation, 90
 - Bokeh, 90
 - Seaborn, 90
- Dataset
 - Testing, 152
 - Training, 152
- de Fermat, Pierre, 287
- Decision trees, 321
 - Post-pruning, 328
 - Pre-pruning, 328
 - Purity, 324
- Dendrogram, 314
 - Clade, 315
- Dependent variable, 166
- Dictionaries, 58
 - items, 59
 - keys, 58, 59
 - values, 58, 59
- Dimensionality Reduction
 - Effective, 403
- Dimensionality reduction, 134, 366, 423
 - Curse of dimensionality, 366
- Directed Acyclic Graph (DAG), 24
- Distance, 139
- Divisive clustering, 314
- Document analysis, 386
- dot command, 81
- Dot product, 142
- Dual form, 422

- Edge, 322
- Eigenvalue decomposition, 367
- Eigenvalues, 367
- Eigenvectors, 367
- Elbow method, 238
- Elbow test, 375
- Ensemble techniques, 337
 - Bagging, 344
 - Blending, 348
 - Boosting, 345
 - Random forests, 346
 - Stacking, 348
- Error
 - Generalisation, 154
 - Out-of-sample, 154
 - Training, 154
- Euclidean distance, 140, 230
- Evidence-based decision
 - making, 2
- Explainable AI, 355, 356
- Explanatory variable, 166
- Extremely Randomised Trees, 351
- Fairness, 441
- Fallout, 261
- False negative, 260
- False positive, 260
- False positive rate, 261
- Feature analysis, 102
- Feature extraction, 367
- Feature selection, 207
- Feature scaling, 194
 - Normalisation, 195
 - Z-score, 197
- Feature selection, 133, 134, 148, 365
 - LASSO, 365
- Fingerprints, 168
- Flink, 23
- Fourier transform, 199
- Frankenstein's monster, 104
- Function, 67
 - Definition, 68
- Galton, Francis, 168
- Gamma function, 147
- Generalisation, 128, 132, 135
- Generalisation error, 135
- Generalised linear model, 272
- Generative AI, 4, 7, 444
 - Bias, 111
 - Data security, 113
 - Ethics, 117
 - Intellectual Property, 114
 - Prompt engineering, 118
 - Quality Control, 112
 - Transparency, 113
- Golem, 104
- Google, 5
- Graph, 322
- Graphviz, 336
- Greedy algorithm, 228, 323
- GridSearchCV, 211, 282
- Hadoop, 26
- Hallucinations, 109, 110
- Hasenbock, 12
- Hierarchical clustering, 314
 - Complete linkage, 316
- Group average, 316
- Single linkage, 316
- Ward method, 316
- Hit rate, 260
- Huichol mythology, 12
- Hyperparameter, 137
- Hypothesis testing, 131
- Image compression, 367
- Image recognition, 446
- Imbalance
 - Between-class, 302
 - Extrinsic, 303
 - Intrinsic, 303
 - Oversampling, 304
 - Resampling, 304
 - Undersampling, 304
 - Within-class, 303
- Imbalanced datasets, 300
- Imitation game, 104
- Immutable object
 - Strings, 48
 - Tuple, 48, 57
- Impurity measure
 - Classification error, 326
 - Entropy, 325
 - Gini impurity, 326
- Inconsistency measure, 320
- Indentation, 60
- Independent variable, 166
- Indexing, 50, 82
 - Colon notation (:), 50
- Infinite loop, 63
- Information gain, 327
- Iris dataset, 150, 380

- Jaccard similarity, 142
- Jackalope, 9, 11, 376
 - Douglas, Wyoming, 13
- JSON, 72
- Jupyter Notebook, 72
- K nearest neighbours, 265
- K-means, 227
- Kafka, 23
- Kepler, Johannes, 103
- Kernel
 - Gaussian, 427
 - Linear, 427
 - Polynomial, 427
 - RBF, 429
 - Sigmoid, 427
- Kernel methods, 412, 425
- Kernel trick, 425
- KNN, 255, 265
- Kondo, Marie, 404
- L1-norm, 141
- L2-norm, 140
- Label data, 127
- LabelEncoder, 281
- Lagrange multipliers, 420
- Large Language Model, 3, 7, 107, 110
 - GPT, 4
- Latent semantic analysis, 386
- Least squares, 172
- Linear algebra, 76, 173, 224, 366
- Linear independence, 384
- Linear regression
 - Multivariate, 170
 - Normal equation, 173
 - Optimisation, 173
 - Univariate, 169
- Linearly separable dataset, 413
- List, 50
 - append, 51
 - Comprehension, 54
 - Concatenation, 51, 77
 - sort, 52
 - sorted, 53
- LLMs, *see* Large Language Model
- Log-odds function, 274
- Logarithmic transformation, 189
- Logistic function, 273
- Logistic regression, 255, 272, 382
 - Interpretation, 276
 - Regularisation, 276
- Logit function, 274, 277
- Luigi, 24
- Machine learning, 3
- Machine learning, 2, 4, 101, 104, 123, 131
 - Supervised learning, 27, 127
 - Unsupervised learning, 27, 128
- Manhattan distance, 140
- Margin, 415
- math, 74
- Mathematics, 4, 102
- MATLAB, 90
- Matplotlib, 90
 - savefig, 92
- Matrix
 - Inverse, 80
- Matrix rank, 384
- Maximum margin hyperplane, 416
- Mercer theorem, 425
- Minority class noise, 302
- Misleading accuracy scores, 302
- Model selection, 209
- Modelling, 8, 22
 - Interpretation, 19
- Modules, 74
 - math, 75
- MongoDB, 24
- Multiclass classification, 278
- Multicollinearity, 202
- Mutable object
 - List, 50
- Naïve Bayes classifier, 27, 124, 255, 287, 293, 386
- Named entity recognition, 439
- Natural language processing, 445
- Nearest neighbours, 244
- Netflix, 5
- Neural network, 136
 - Convolutional, 136
- Node, 315, 322
- Normal distribution, 197
- Numerical features, 129
- NumPy, 78
 - Arrays, 78
 - Matrix, 78
 - mean, 89
 - SVD, 388

- Transpose, 80
- Objective function, 138
- Odds, 276
- Odds ratio, 277
- OLS, 181
- One-versus-the-rest, 278
- Ordinary least squares, 181
- Overfitting, 133, 135
- pandas, 83
 - crosstab, 330
 - CSV, 89
 - DataFrame, 85
 - describe, 87
 - Excel, 89
 - groupby, 88
 - groups, 88
 - head, 85
 - size, 88
 - tail, 85
- Pascal, Blaise, 287
- Pattern recognition, 101
- PCA, 365, 371, 411
- Pearson correlation, 183
- Physics, 102
- Pinocchio, 104
- Pipeline, 382
- Plotting, 90
- Positive predictive value, 261
- Power BI, 90
- Power law, 190
- Precision, 261
- Predictive analytics, 6
- Predictive learning
 - see Supervised learning, 127
- Prefect, 24
- Primal form, 422
- Principal component analysis,
 - 149, 365, 371
- print, 48
- Probability
 - Bayesian, 288
 - Frequentist, 287
 - Likelihood, 291
 - Posterior, 291
 - Prior, 291
- Programming, 9
- Prompt engineering, 118
- Purity, 324
- Pyplot, 90
- Python, 1, 25, 26, 37
 - Anaconda, 41
 - Commenting code, 44
 - Control flow, 60
 - easy-install, 41
 - Homebrew, 41
 - Indentation, 39
 - Interactive shell, 43
 - iPython Notebook, 40
 - iPython shell, 40
 - Jupyter Notebook, 40, 44
 - Matplotlib, 40
 - Methods, 49
 - Modules, 72
 - Monty Python, 37
 - NumPy, 27, 40
 - Object oriented, 49
 - pandas, 40
 - pip, 41
- Portability, 41
- Pythonic style, 39
- Pythonista, 39
- Scikit-learn, 27, 40
- SciPy, 27, 40
- Scripts, 72
- shell, 42
- Standard Library, 72, 75
- statsmodels, 40
- Strings, 47
- QlikView, 90
- R, 181
 - Formula notation, 181
- R-project, 26
- Random Forest Classifier, 351
- Random_state, 267
- Recall, 260
- Receiver operator characteristic,
 - 262
- Recommendation systems, 390
 - Collaborative filtering, 392, 396
 - Content-based filtering, 391, 392
- Regression, 139, 165, 187
 - Backward elimination, 201
 - Categorical, 214
 - Forward selection, 201
 - Ill conditioning, 202
 - LASSO, 207, 208
 - Linear, 178
 - Logarithmic transformation, 189

- Multivariate, 203
- Polynomial, 198
- Ridge, 207, 208
- Univariate, 178
- Regression analysis, 167
- Regression to the mean, 168
- Regressor, 166
- Regularisation, 133, 135, 137, 276, 420
- Reliability, 8
- Representation, 129
- Resampling, 344
- Response variable, 166
- ROC, 262
- Science, 2
- Scientific method, 2
- Scikit-learn, 149, 178
 - get_dummies, 332
- SciPy, 78
- Scree plot, 374
- Sensitivity, 260
- Sentiment analysis, 437
- SHAP, 356, 357
- shape, 82
- Shilling attack, 403
- Shrinkage, 208
- Sigmoid function, 273
- Simultaneous assignment, 48
- Singular value decomposition, 365, 384, 426
- Singularity, 104
- Slack variables, 419
- Slicing, 50, 82
 - Colon notation (:), 50
- SMOTE, 309
- Social network analysis, 391
- Spark, 72
 - PySpark, 72
- Sparsity
 - Space, 146
- Specificity, 261
- SSR, 139
- Standardisation, 194
- Statistical analysis, 22
- Statistics, 2, 9
- statsmodels, 178
- Steepest descent algorithm, 420
- str, 48
- String
 - Concatenation, 48
 - lower, 55
 - split, 55
 - upper, 55
- Strong learner, 339
- Subject matter expertise, 125
- Subtree raising, 328
- Subtree replacement, 328
- Sum of squared residuals, 139, 172
- Supervised learning
 - Classification, 255, 256
- Support vectors, 416
- Support vector machines, 27, 411, 412, 415
 - Nonlinear, 426
- Support vector regression, 428
- SVD, 365, 384, 390, 411
- SVM, 411, 412
 - Classification, 431
- Regression, 427
- Soft SVM, 420
- Symbiosis, 447
- Synthetic Minority
 - Oversampling Technique, 309
- Tableau, 90
- Taylor series, 199
- Term-document matrix, 386
- Testing dataset, 153
- Text classification, 438
- Text mining, 142
- Titanic, The, 328
- Training dataset, 153
- Transparency, 441
- True negative, 259
- True negative rate, 261
- True positive, 259
- True positive rate, 260
- Tuple, 55
- Turing test, 104
- Turing, Alan M., 104
- type (command), 47
- Types, 46
 - Complex numbers, 46
 - Floats, 46
 - Integers, 46
- Unlabelled data, 128
- Unsupervised learning
 - Clustering, 225, 226
- Utility matrix, 397
- Validation dataset, 153

van Rossum, Guido, [37](#)

Variance, [135](#)

Variance-bias tradeoff, [204](#)

Volume

Hypercube, [147](#)

Hypersphere, [147](#)

Weak learner, [339](#)

Whitespace, [60](#)

Wisdom of crowds, [338](#)

X (formerly, Twitter), [294](#)