Shan Chang · Liangxu Xie

AI-Assisted Drug Design

人民邮电出版社有限公司
POSTS & TELECOM PRESS Co.,LTD

Springer

AI-Assisted Drug Design

Shan Chang · Liangxu Xie

AI-Assisted Drug Design

人民邮电出版社有限公司
POSTS & TELECOM PRESS Co.,LTD

Springer

Shan Chang
Institute of Bioinformatics and Medical Engineering
Jiangsu University of Technology
Changzhou, Jiangsu, China

Liangxu Xie
Institute of Bioinformatics and Medical Engineering
Jiangsu University of Technology
Changzhou, Jiangsu, China

ISBN 978-981-95-8363-8 ISBN 978-981-95-8364-5 (eBook)
https://doi.org/10.1007/978-981-95-8364-5

Jointly published with Posts & Telecom Press
The print edition is not for sale in Mainland China. Customers from Mainland China please order the print book from: Posts & Telecom Press.

Translation from the Chinese Simplified language edition: "人工智能辅助药物设计" by Shan Chang and Liangxu Xie, © Posts & Telecom Press 2025. Published by Posts & Telecom Press. All Rights Reserved.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Drug discovery and development is a data-intensive field. As an emerging technology capable of handling big data and uncovering complex interaction patterns, artificial intelligence has made breakthrough progress in the field of drug discovery and pharmaceutical research. The application of artificial intelligence in drug discovery and development can be traced back to Hansch's quantitative structure-activity relationship proposed in 1964. Today, it is widely used in various new drug research and development stages, including drug target discovery, compound screening, drug target interaction networks, lead compound optimization, drugability analysis, and peptide design. Artificial intelligence-assisted drug discovery (AIDD) has now become a new strategy for pharmaceutical companies. New drug research and development based on artificial intelligence is regarded by the industry as the most transformative research technology. However, the application of artificial intelligence in drug discovery and development requires the integration of expert knowledge from multiple fields such as artificial intelligence, life science, drug discovery and chemistry, which makes it difficult for newcomers to the field to find suitable reference materials.

The authors of this book have been engaged in scientific research in the field of artificial intelligence-assisted drug discovery for a long time. They have accumulated a wealth of technical experience in the research and development of small molecule drugs and peptide vaccines, and has published related papers in authoritative journals, which have been highly praised by domestic and foreign peers. The protein-protein docking method developed by them has achieved the top three positions in the international CASP-CAPRI competition. After years of accumulation, based on some thoughts on the development of this field and the need to cultivate students and share knowledge, the author wrote this book.

Content Summary

This book consists of 18 chapters and covers the following three thematic areas.

The Basic Methods of Artificial Intelligence—Introduces the fundamental principles of machine learning and deep learning methods. This part covers the development overview and basic concepts of artificial intelligence, as well as support vector machines, decision trees, ensemble learning, random forests, feedfoward neural networks, convolutional neural network, and Transformer, aiming to help readers understand the foundational knowledge of artificial intelligence. Machine learning, as the cornerstone of artificial intelligence, remains a key methodological benchmark in the field of AI-assisted drug design. This book not only covers classical machine learning approaches but also incorporates cutting-edge techniques such as deep learning, thereby systematically offering foundational to advanced methodological support to foster innovation and application development in this domain.

Programming and Development Environment—Introduces the Python language and the setup of the programming and development environment. Unlike other books that focus on the Python language, this book does not delve deeply into the programming language itself. Instead, it leans towards directly applying the programming language to the context of drug research and development, that is, using the Python language to set up popular deep learning frameworks such as TensorFlow and PyTorch, to complete the practical applications of artificial intelligence-assisted drug research and development environment.

Fundamentals of Biomedical Sciences—Introduction to commonly used drug databases and protein databases, basic methods of drug screening, deep learning in QSAR models, feature extraction of molecules, prediction of drug molecule properties, de novo generation of drug molecules, prediction of protein structures, and prediction of binding free energies between proteins and ligands. This part will be illustrated with examples to help readers understand the application scenarios of artificial intelligence in the field of drug design.

Changzhou, China

Shan Chang
Liangxu Xie

Acknowledgments

This book was supported by the National Natural Science Foundation of China (Grant No. 62373172, 22003020), Basic Research Program of Jiangsu (Grant No. BK20253050), the Natural Science Foundation of Jiangsu Province (Grant No. BK20191032), the Changzhou Applied Basic Research Project (Grant No. CJ20200045), the Jiangsu Province Double Innovation Doctor Program, the NSFC-Guangdong Joint Fund (Phase II) Special Project on Supercomputing Science Application Research, the Jiangsu Province Double Innovation Program for Vice President of Science and Technology, the Jiangsu Province "Six Talents Peak" Project, and the Changzhou Youth Talent Program.

The successful publication of this book would not have been possible without the help of many people. Special thanks go to Academician Runsheng Chen of the Chinese Academy of Sciences and Prof. Jian Zhang of Shanghai Jiao Tong University for their guidance and recommendations. We also express our gratitude to the team members, including Guoming Bao, Yutao Liu, Shenhuan Ni, Chenghui Yang, Yifan Zhang, and Yuncong Zhang, listed in alphabetical order by given name.

Due to the author's limited proficiency, this book has not been able to incorporate more recent advancements in artificial intelligence-assisted drug discovery and development. Omissions and errors are inevitable, and we sincerely invite readers to offer their criticism and suggestions for future revisions and improvements.

Contents

Chapter 1
Brief Introduction

1.1 The History of Artificial Intelligence

The term "Artificial Intelligence" was first proposed as a research field at the Dartmouth Conference in 1956. Scientists led by John McCarthy (the inventor of LISP and a Turing Award winner), Marvin Minsky (an expert in artificial intelligence and cognitive science), Claude Shannon (the founder of information theory), and Allen Newell (a computer scientist) jointly studied and explored issues related to simulating intelligence with machines. They first put forward the concept of "Artificial Intelligence", marking the official birth of this emerging discipline. However, the dream of "creating machines that can think and act like humans" can be traced back to ancient Greece. For instance, the ancient Greek philosopher Aristotle (384 BC–322 BC) laid down the basic laws of formal logic. But it was not until after the advent of electronic computers that the practical application and development of artificial intelligence began. In 1936, Turing proposed the Turing machine model, which could simulate any mechanical formal algorithm through basic operations such as reading, writing, moving the read–write head left and right, laying the foundation for the theoretical model of modern electronic computers. In 1946, the world's first general-purpose computer, ENIAC, was born. It was initially developed for the US military and could perform 5000 additions and 400 multiplications per second. ENIAC provided the material basis for the research of artificial intelligence. In 1950, Alan Turing proposed the famous "Turing Test" to measure whether a machine has intelligence. If a computer can answer a series of questions posed by a human tester within five minutes and more than 30% of its answers make the tester mistakenly believe they were given by a human, it passes the test. Regarding the ingenious ideas about artificial intelligence, researchers have been making unremitting efforts to explore them. In the following decades, artificial intelligence was first hailed as the key to humanity's bright future, and then dismissed as overly ambitious and unrealistic.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_1

From the 1950s to the 1960s, artificial intelligence (AI) experienced its first golden age. In 1956, the Dartmouth Conference established the name and mission of AI, and the first achievements and the earliest group of researchers emerged. This event is regarded as the birth of AI. In 1959, Arthur Samuel, a pioneer in computer games, developed a checkers program on IBM's first commercial computer, the IBM 701, which successfully defeated the then checkers master Robert Nealey. In 1965, expert systems made their debut. American scientist Edward Albert Feigenbaum and others developed the DENDRAL program, an expert system for chemical analysis, which was used to analyze experimental data to determine the molecular structure of unknown compounds.

In the early 1970s, AI hit a development bottleneck. Even the most outstanding AI programs could only solve the simplest parts of the problems they attempted to address, meaning that at that time, all AI programs were merely "toys". AI researchers encountered fundamental obstacles that were difficult to overcome. Although some limitations were successfully broken through later, many difficult problems remain unsolved to this day.

In the 1980s, AI experienced its second golden age. In 1982, physicist John Hopfield demonstrated that a new type of neural network (now known as the "Hopfield network") could learn and process information in a completely new way. Around the same time, Geoffrey Hinton and David Rumelhart proposed a method for training neural networks—the backpropagation algorithm. These discoveries revitalized connectionism, which had been "abandoned" since the 1970s.

In the mid-1990s, AI finally achieved some of its initial goals and began to be successfully applied throughout the technology industry. Within the field of artificial intelligence, some subfields began to emerge, each focusing on specific problems and methods, such as machine learning, natural language understanding, computer vision, and robotics. In 1997, IBM's supercomputer DeepBlue defeated the world chess champion Garry Kasparov, marking an important milestone in the development of AI. DeepBlue could calculate 200 million moves per second and had a database of 700,000 master games, enabling it to search and estimate the next 12 moves.

In the past few years, AI has witnessed an explosive growth, which is largely attributed to the widespread application of graphics processing units (GPUs)—making parallel processing faster, cheaper and more powerful, as well as the emergence of almost unlimited storage space and massive amounts of data, especially the data that emerged after the "big data movement", such as images, texts, transaction data, map data, etc.

With the development of the Internet and the popularization of computers, the explosive growth of data has laid the foundation for the application of AI. At the same time, the rapid advancement of computer hardware technology, the emergence of GPUs, TPUs and distributed computing, have provided powerful computing power support for artificial intelligence algorithms. In 2016–2017, AlphaGo defeated the Go champion. AlphaGo is an artificial intelligence Go program developed by Google's DeepMind team, which has the ability to self-learn, capable of collecting a large amount of Go game data and famous players' game records, learning and imitating human playing. Later, AlphaGo Zero (the fourth generation of AlphaGo) began to

self-learn Go without any data input and surpassed the second generation AlphaGo (AlphaGo Lee) with a score of 100:0 just 36 h after its birth. On the 40th day after its birth, it defeated AlphaGo Master, the third generation of AlphaGo, which was considered unattainable by human experts.

As shown in Fig. 1.1, artificial intelligence, machine learning and deep learning are in an inclusive relationship in sequence. Machine learning is a subfield of artificial intelligence, and deep learning is a method of machine learning. Machine learning is an interdisciplinary subject involving knowledge from multiple fields such as probability theory, statistics, approximation theory, convex analysis, and algorithm complexity theory. It specifically studies how computers simulate or implement human learning behavior to acquire new knowledge or skills, and reorganize existing knowledge structures to continuously improve their own performance. It is the core of artificial intelligence and the fundamental approach to endowing computers with intelligence.

Deep learning is a type of machine learning, and machine learning is an essential path to achieving artificial intelligence. The concept of deep learning originated from artificial neural networks. For instance, a multi-layer perceptron with multiple hidden layers is a deep learning structure. Deep learning combines low-level features to form more abstract high-level representation attributes or features to discover the distributed feature representation of data. The motivation behind deep learning is to establish neural networks that simulate the human brain for analysis and learning, and to interpret data such as images, sounds and texts by imitating the mechanisms of the human brain.

In recent years, the popularity of artificial intelligence has been driven by the development of big data and faster, more powerful computer hardware. Deep learning

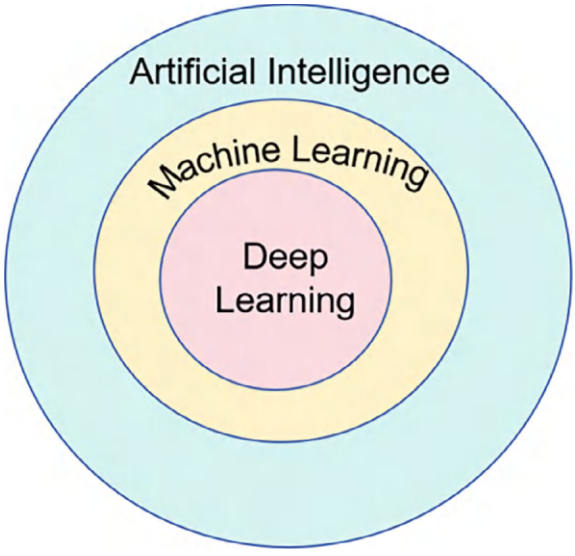


Fig. 1.1 Relationship diagram of artificial intelligence, machine learning and deep learning

methods, represented by deep neural network models, have led to the rise of the third wave of artificial intelligence.

1.2 The History of Traditional Computer-Aided Drug Design

Computer-Aided Drug Design (CADD), based on computer chemistry, is a method for designing and optimizing lead compounds by simulating, calculating and predicting the relationship between drugs and receptor biomolecules through computers. The so-called computer-aided drug design is actually to optimize and design lead compounds by simulating and calculating the interaction between receptors and ligands. There are three directions in which computer-aided drug design is closely combined with artificial intelligence (especially machine learning), namely quantitative structure–activity relationship, molecular docking and ADMET prediction.

In the 1960s and 1970s, the emerging and rapidly developing field of rational drug design came into being, and the study of structure-activity relationships (SAR) was elevated from qualitative speculation to quantitative calculation. In the 1960s, SAR research entered the quantitative era. Hansch analysis, proposed by the medicinal chemist Hansch, linked the overall hydrophobicity, electronic properties, and steric parameters of molecules to the physiological activity of drug molecules, establishing a two-dimensional quantitative SAR method. In 1988, Cramer et al. proposed the comparative molecular field analysis (CoMFA) method based on the spatial structure of molecules. CoMFA compares the physicochemical parameters such as hydrophobicity and electrostatic potential at various points in the vicinity of molecules in the same series and establishes a connection between these parameters and the physiological activity of small molecules, thereby guiding the design of new compounds. Compared with the Hansch method, CoMFA takes into account the internal spatial structure of molecules and is thus referred to as three-dimensional quantitative SAR.

Since the first molecular docking software DOCK was introduced in the 1980s, the molecular docking method has been continuously evolving. Using docking to explore the interactions between small molecules and protein binding pockets has become a very popular simulation method in drug design and is also a commonly used virtual screening strategy. In molecular docking, the training of scoring functions often employs machine learning methods (such as regression methods) to obtain the optimal combination weights.

In the process of drug development, ADMET properties are the key factors to determine whether a compound can be developed into a drug. The ADMET properties of a drug refer to its absorption, distribution, metabolism, excretion and toxicity in the human body, involving the pharmacokinetic and toxicological properties of the drug in vivo. Similar to quantitative structure–activity relationship (QSAR), the prediction of ADMET properties also starts from the physicochemical parameters of

the molecule and establishes the relationship between these parameters and ADMET properties through methods such as neural networks and support vector machines (SVM).

1.3 Overview of Artificial Intelligence-Assisted Drug Development

Artificial intelligence has been applied in drug design since the 1980s. Nowadays, the field of drug research and development is increasingly adopting it. Deep learning methods have become the primary training tool, and the scenarios for research and development have also become more diverse.

In 2015, Wallach et al. introduced the deep learning model AtomNet for predicting the binding affinity of active compounds selected for drug discovery. AtomNet was the first deep learning model to use a convolutional neural network (CNN) to predict the binding affinity of small molecules, employing a novel approach that integrates the structural information of both ligands and targets. However, AtomNet requires the three-dimensional structures of both the ligands and the target proteins, which include the positions of each atom involved in the interaction at the target binding site.

In 2018, Mark Waller from Shanghai University and Marwin Segler from the University of Münster and others published an article in Nature, introducing an artificial intelligence tool that can design synthetic routes for molecules by autonomously learning organic reactions. They combined three different neural networks with Monte Carlo Tree Search (MCTS) to form a new AI algorithm (3N-MCTS), which was trained using automatically extracted rule data. In tests on 435 complex molecular building blocks published after 2015, the 3N-MCTS algorithm was able to complete the design of 80% of the molecular synthetic routes within a single target molecule time limit of 5 s, and when the time limit was extended to 60 s, the completion rate increased to 92%.

In 2020, a team from Stanford University and Merck systematically compared a graph convolution-based multi-task deep learning method with the traditional random forest method based on molecular fingerprints on 31 ADMET datasets. They trained on 31 ADMET datasets and compared the results of random forests and GCNN on the test sets under two different cross-validation strategies. The results showed that the prediction accuracy of the multi-task deep learning method was significantly improved.

The determination of protein structures has been a scientific conundrum plaguing the field of life sciences for over 50 years, especially the structural analysis of important target proteins. To address this fundamental issue, international organizations have held the Critical Assessment of protein Structure Prediction (CASP) competition. In 2020, during CASP, significant breakthroughs were made in monomer

structure prediction. Google's AlphaFold 2 achieved prediction accuracy comparable to experimental results in multiple systems and predicted the structures of 98.5% of human proteins. In contrast, the efforts of scientists over several decades only covered 17% of human protein sequences. Nature reported on this with the headline "It will change everything", and Science magazine listed it as one of the top ten scientific advancements for two consecutive years (2020 and 2021). Yigong Shi, a structural biologist, also stated that the precise prediction of protein structures by AlphaFold is one of the most significant scientific breakthroughs of the twenty-first century. The 2024 Nobel Prize in Chemistry was awarded to Demis Hassabis, John Jumper, and David Baker for their revolutionary work using artificial intelligence to predict protein structures and design new proteins. Alphafold won the Nobel Prize, marking a significant milestone in the field of science. This achievement has attracted widespread attention and recognition from the global scientific community, as it represents a major breakthrough in understanding and predicting protein structures, which holds great potential for advancements in various areas such as medicine, biotechnology, and drug discovery. The breakthrough in protein structure prediction has paved the way for a revolutionary wave of AI applications in the entire drug discovery pipeline. Recent advancements are particularly driven by the emergence of sophisticated AI agents and Large Language Models (LLMs). AI agents are emerging as transformative tools, moving beyond single-task machine learning models to intelligent, multi-step systems. LLMs, the technology behind tools like ChatGPT and AlphaFold 2 (which uses a transformer architecture similar to LLMs), are being adapted to understand the "language" of biology and chemistry.

Chapter 2
Basic Concepts of Machine Learning

This chapter provides a systematic introduction to the fundamental concepts of machine learning. It first clarifies the relationships among artificial intelligence, machine learning, and deep learning, and reviews the historical development and core ideas of artificial intelligence. Key concepts in machine learning, including data partitioning, feature scaling, robustness, generalization, and overfitting, are then introduced to establish a theoretical foundation for model training and evaluation. The chapter further presents the classification of machine learning from the perspectives of task types and learning paradigms, covering regression, classification, clustering, dimensionality reduction, supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Finally, the chapter discusses the application of machine learning in drug development, summarizes common machine learning methods used in drug design, and outlines the general workflow for constructing predictive models, providing a conceptual basis for subsequent chapters.

2.1 Machine Learning, Deep Learning and Artificial Intelligence

In recent years, the presence of artificial intelligence can be seen everywhere in people's lives, both at home and abroad. By applying machine learning technology, we can enable software to describe or predict future scenarios based on vast amounts of data. Nowadays, leading tech giants are making significant investments in the field of machine learning, and many domestic and foreign technology companies are also involved.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_2

2.1.1 Artificial Intelligence

The definition of artificial intelligence can be divided into two parts, namely "artificial" and "intelligence". "Artificial" is relatively easy to understand, but there is much debate about what "intelligence" is. This involves issues such as consciousness, self, and mind, making it difficult to precisely define what "artificially" created "intelligence" is. Research on artificial intelligence usually involves the study of human intelligence, and other studies on the intelligence of animals or artificial systems are also generally regarded as related research topics of artificial intelligence [1].

"Artificial intelligence" is not a concept that emerged and gained popularity rapidly; rather, it has undergone a long and tortuous development process as shown in Fig. 2.1. When the pioneers of artificial intelligence met at Dartmouth, they hoped to create complex machines with intelligence comparable to that of humans through the then-emerging computers. This is what we call the concept of "general artificial intelligence" (General AI), referring to the magical machines (agents) that possess human senses (and even more), reasoning abilities, and human ways of thinking. We have seen many such robots (agents) in science fiction movies, such as the friendly C-3PO and the hostile Terminator. In fact, such robots still only exist in movies and science fiction novels, at least not yet in the real world. What we can achieve is "narrow artificial intelligence" (Narrow AI), which refers to robots (agents) that perform specific tasks at a level comparable to or even surpassing that of humans. There are already many examples of narrow artificial intelligence in reality. These examples are sufficient to demonstrate the characteristics of artificial intelligence, but how is intelligence achieved and how will it develop in the future? This brings us to machine learning, a method for achieving artificial intelligence, whose concept originated from early artificial intelligence researchers. Simply put, machine learning is a process where machines learn from data to improve their performance in specific tasks.

Machine learning involves using algorithms to analyze data, learn from it, and make inferences or predictions [2]. Unlike traditional software development that relies on hand-coding specific instruction sets, in machine learning, we use large amounts of data and algorithms to "train" machines, thereby teaching them how to perform tasks. The machine learning algorithms that have been developed include Decision Tree, inductive logic programming, reinforcement learning, and Bayesian networks, among others.

For many years, despite the need for a large amount of manual coding to complete tasks, computer vision has been one of the most popular application areas of machine learning. Researchers would manually write some classifiers (such as edge detection filters) to help the agent distinguish the boundaries of objects. For example, an image detection classifier can determine whether an object has eight faces. Based on these manually written classifiers, researchers have developed algorithms for understanding images and enabled them to learn how to determine whether there is a stop sign.

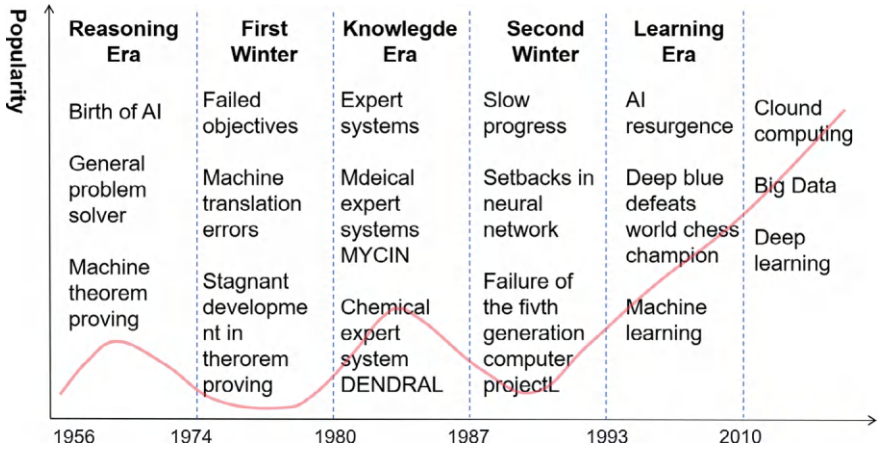


Fig. 2.1 Development process of artificial intelligence

Deep learning is a technique for achieving machine learning. Early machine learning researchers developed neural network algorithms, but they remained obscure for decades. Neural networks are inspired by the interconnection relationships among neurons in the human brain [3]. However, in the human brain, neurons can connect with any other neurons within a certain range, while in artificial neural networks, data transmission occurs through different layers and in different directions. For instance, you can divide an image into several small pieces and input them into the first layer of a neural network, where initial computations are made. Then, the neurons pass the data to the second layer, where the second layer of neurons perform their tasks, and so on, until the final layer outputs the result.

Each neuron assigns a specific weight to its input, and the final output is determined by these weights collectively. Let's take another example: the attributes of a stop sign image are broken down one by one and sent to the neural network for "checking" the shape, color, characters, sign size, and whether it is moving. Eventually, the neural network determines whether it is a stop sign and provides a "probability vector", that is, the prediction result based on the weights. In this example, the neural network has approximately an 86% probability of recognizing the image as a stop sign and a 7% probability of recognizing it as a speed limit sign.

Even basic neural networks consume vast amounts of computing resources, so neural networks were not considered a feasible approach in their early days [4]. However, some researchers led by Professor Geoffrey Hinton from the University of Toronto persisted in using this method.

The method eventually enabled the supercomputer to execute the algorithm in parallel and confirmed the algorithm's effectiveness. In the case of the stop sign, the neural network is very likely to give a wrong answer due to the influence of training. That is to say, to improve the prediction accuracy of the neural network, a large amount of training is still needed, that is, a huge number of pictures are

required for training until the weights of the neural network inputs are adjusted to a very precise level, ultimately enabling the neural network to give the correct answer almost every time. Nowadays, in some cases, machines trained through deep learning perform better than human in image recognition, such as identifying cat images and recognizing cancer signs in blood from medical images [5].

The essence of artificial intelligence lies in "intelligence", and machine learning is the computational method deployed to support artificial intelligence. Simply put, artificial intelligence is a science, while machine learning is the algorithm that makes machines more intelligent. To some extent, machine learning has driven the development of artificial intelligence.

2.1.2 Key Terms in Machine Learning

This section introduces several fundamental concepts commonly used in machine learning, including data partitioning, feature scaling, robustness, generalization, and overfitting. These concepts form the theoretical basis for understanding model training, evaluation, and optimization.

1. Training dataset, validation dataset and test dataset

   (1) Training dataset: A collection of data used for training a model.
   (2) Validation dataset: A collection of samples used for validating, adjusting, and optimizing the model.
   (3) Testing dataset: A collection of samples used to evaluate the accuracy and generalization ability of a predictive model.

In machine learning, after training a model, how can we validate it? Or, how can we measure the performance of the model? There are four ways to validate it.

① Use the entire dataset as the training set. This approach is clearly not feasible. The data in the training set has already been used during model fitting and is then used to test the model, which will undoubtedly lead to overly optimistic results. This is similar to the situation where an exam consists of questions directly from the textbook, which can cause data snooping bias.
② Randomly divide the dataset into a training set and a test set. To verify the performance of the model and avoid misleading results from data snooping, we usually split the data into a training set and a test set. The training set is used for training the model and adjusting parameters, while the test set is used for evaluating the model's performance. The drawback of this approach is that it is not easy to adjust hyperparameters.
③ Randomly divide the dataset into a training set, a validation set, and a test set. This approach involves training the model with the training set, validating it with the validation set, and then continuously adjusting the model based on the results to select the best-performing one. Finally, a final model is trained using the data from both the training and validation sets. Since the validation set is used in the

adjustment of hyperparameters, the test set is used to evaluate the final model. The drawback of this method is that the training set is relatively small, which may affect the quality of training.

④ Cross-validation. Cross-validation refers to dividing the dataset into a training set and a test set, then splitting the training set into K groups. The model is trained K times, with one group used as the validation set each time and the rest as the training set. The performance of the model is evaluated using the test set and the results are recorded. The final prediction accuracy of the model is the average of all the evaluation results from the iterations.

Tips: In practice, fivefold cross-validation as shown in Fig. 2.2 is often used for training and testing. The so-called fivefold cross-validation means that the data is evenly divided into 5 parts, with each part in turn serving as the validation set and the remaining 4 parts as the training set.

The most commonly used method is the fourth one—cross-validation, which can not only avoid data snooping errors but also make the fullest use of the training data.

2. Feature Scaling-Normalization, Standardization

In the field of machine learning, it is common practice to preprocess data using some feature scaling methods before model training. Among the more frequently used feature scaling methods are normalization, standardization, and Centralization:

(1) Min–max normalization: Transform a column of data into a fixed range (typically [0, 1]), and it can also be mapped to other ranges. For instance, image data might be mapped to [0, 255], while in other cases, it could be mapped to [− 1, 1]. The expression for normalization is given in Eq. (2.1).

$$X_{new} = \frac{X_i - X_{\min}}{X_{\max} - X_{\min}} \tag{2.1}$$

(2) Standardization: Transforming data into a distribution with a mean of 0 and a standard deviation of 1, but not necessarily a normal distribution. The expression for standardization is given in Eq. (2.2).

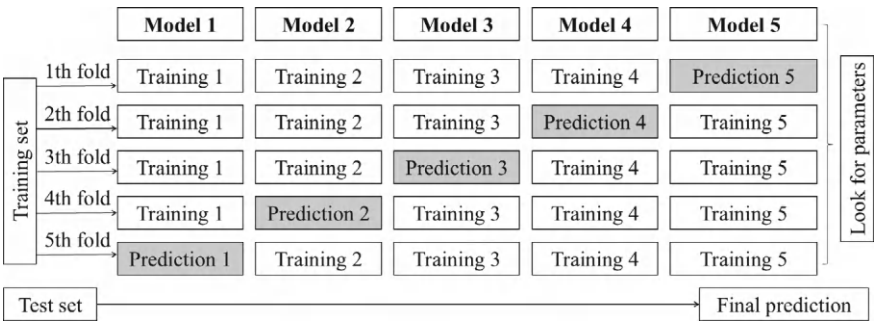


Fig. 2.2 Schematic diagram of fivefold cross-validation

$$X_{new} = \frac{X_i - \mu}{\sigma} \tag{2.2}$$

(3) Centralization (also known as zero-mean processing): Transform the data so that its mean is 0. The expression for centralization is given in Eq. (2.3).

$$X_{new} = X_i - \mu \tag{2.3}$$

Why do we need to normalize, standardize and centralize data? First, in statistical modeling, taking regression models as an example, inconsistent units of measurement for independent variables X can lead to incorrect or misinterpreted regression coefficients. Therefore, it is necessary to process X to a uniform unit of measurement so that they can be compared. Second, in machine learning and statistical tasks, there are many places where "distance" calculations are needed, such as PCA and k-nearest neighbor (kNN). Different units of measurement for different dimensions may cause the distance calculation to be dependent on the features with larger units of measurement, resulting in unreasonable results. Third, when using the gradient descent method for parameter estimation, normalizing and standardizing the data can accelerate the solution speed of gradient descent, that is, improve the convergence speed of the model.

In practical application, how to choose between normalization and standardization? Currently, there is no very clear statement on this. The common practice is to select normalization or standardization based on the boundary characteristics of the data and the specific distribution of the data. There are other methods for feature scaling, such as log transformation, sigmoid transformation, Box-Cox transformation, etc.

3. Robustness and generalization ability

Robustness originally refers to the ability of a control system to maintain stable performance under parameter or structural perturbations. In machine learning, robustness generally describes a model's ability to maintain stable and reliable performance when the input data contain noise or bias.

Generalization ability refers to the predictive power of a learned model on unknown data. That is to say, a well-trained model can maintain the accuracy during training when predicting similar data or data from the test set that was not involved in the training process. Generalization ability has always been the goal of machine learning.

4. Empirical error, generalization error

The fundamental problem of machine learning is to fit a model to the data. The purpose of learning is not to make correct predictions on the limited training set, but to make correct predictions on samples that have never appeared in the training set.

Empirical error refers to the error of a model on the training set data; generalization error, on the other hand, refers to the error of the model on the test set data.

5. Overfitting and Underfitting

Overfitting and underfitting are two common reasons for the poor generalization ability of models, both resulting from the mismatch between the model's learning capacity and the complexity of the data.

Underfitting refers to a situation where the model's learning ability is weak, which occurs when the complexity of the data is high. At this point, the model's learning capacity is insufficient to capture the "general rules" within the dataset, resulting in poor generalization ability. Overfitting, on the other hand, occurs when the model's learning ability is too strong, to the extent that it can capture the unique characteristics of individual samples in the training set and mistake them for "general rules", thereby leading to a decline in the model's generalization ability.

The difference between overfitting and underfitting lies in that underfitting performs poorly on both the training set and the test set, while overfitting can usually learn the properties of the training set data well but performs poorly on the test set. During the training process of neural networks, underfitting is mainly manifested as high bias in the output results, while overfitting is mainly characterized by high variance in the output results. Underfitting, good fitting and overfitting are shown in Fig. 2.3.

The reason for underfitting is usually that the model complexity is too low and the number of features is too small. Underfitting is relatively easy to correct, and the common solutions are as follows.

- Add new features. Consider incorporating feature combinations and higher-order features to expand the hypothesis space.
- Add polynomial features. This method is widely used in machine learning algorithms. For instance, adding quadratic or cubic terms can enhance the generalization ability of linear models.
- Reduce the regularization parameter. Regularization is used to prevent overfitting, but if the model is underfitting, the regularization parameter needs to be reduced.
- Use nonlinear models. For example, kernel Support Vector Machine, Decision Tree, deep learning models, etc.

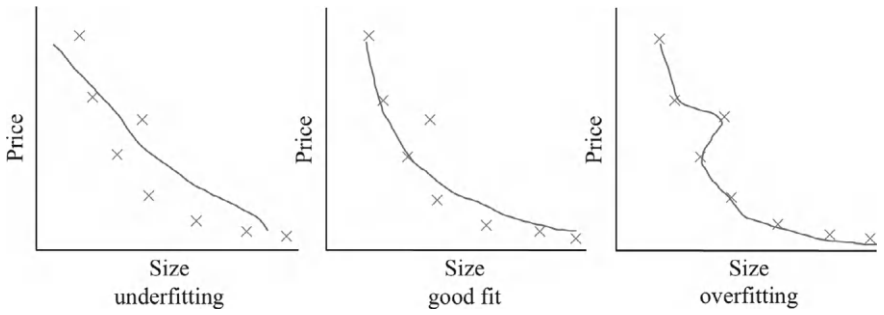


Fig. 2.3 Underfitting, good fit and overfitting

- Adjust the capacity of the model. The capacity of a model refers to its ability to fit various functions. A model with low capacity may have difficulty fitting the training set.
- Use ensemble learning methods. For instance, apply Bagging to multiple weak learners.

The reasons for overfitting are as follows.

- Errors in the selection of modeling samples, such as too few samples, incorrect sampling methods, or incorrect sample labels, can lead to the selected sample data being insufficient to represent the intended classification rules.
- The excessive interference of sample noise causes the machine to mistake some noise for features, thereby disrupting the preset classification rules.
- The hypothetical model cannot reasonably exist, or the conditions for the assumption to hold are actually not met.
- There are too many parameters and the model complexity is too high.

The solutions to overfitting are as follows.

- Regularization (L1 and L2).
- Data augmentation, that is, increasing the number of training data samples.
- Early stopping.

6. Regularization, Data Augmentation, Dropout and Early Stopping

Regularization refers to the process in model training where the error (loss) needs to be reduced to improve accuracy. However, blindly pursuing a reduction in error can easily lead to overfitting. By adding a parameter that represents the complexity of the model structure to the error, a balance can be achieved.

In cases of low error and to prevent overfitting, this method is called regularization. Regularization methods include L0 regularization, L1 regularization, and L2 regularization, which are closely related to norms. In machine learning, L2 regularization is generally used.

Note	Norm: Let X be a linear space over the field K. A function $\|\cdot\|$ is called a norm on X if it is assign a notion of "length" to vectors. Specifically, it maps all vectors in the space to non-negative real numbers, representing their magnitudes. Different norms yield different measures of vector length or size. The common norms and their explanations are shown in Fig. 2.4. Among them, L0 regularization corresponds to L0 norm, L1 regularization corresponds to the L1 norm, and L2 regularization corresponds to the L2 norm. The relationship between regularization and norms lies in the fact that the norm of the model's parameter matrix determines the magnitude of the regularization term

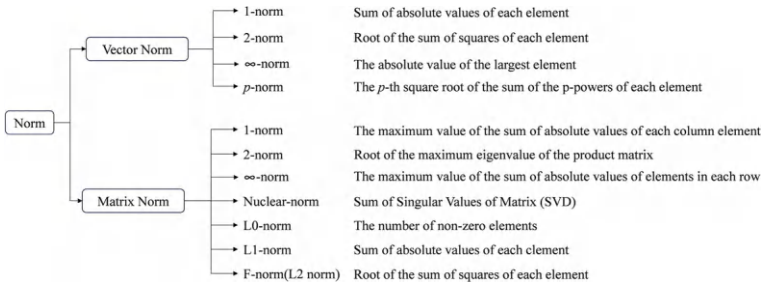


Fig. 2.4 Classification of norms

Data augmentation refers to adding more data to address the overfitting problem caused by insufficient data volume. To obtain more data, the following methods can be adopted.

- Obtain more data from the source.
- Estimate the parameters of the data distribution based on the current dataset and use this distribution to generate more data (not commonly used).
- Data augmentation: Expanding data through certain rules. For instance, in object classification problems, the position, posture, size of the object in the image, as well as the overall brightness of the image, do not affect the classification result. Therefore, we can multiply the database by means of image translation, flipping, scaling, cropping, etc.

Dropout refers to randomly discarding certain nodes in the hidden layer (e.g., with a 50% probability) during training. In neural networks, dropout is often used to prevent overfitting.

Early stopping is a method to prevent overfitting by truncating the number of iterations, that is, to stop the iteration before the model converges on the training dataset to prevent overfitting. Specifically, at the end of each training iteration, the accuracy of the validation set is calculated. When the accuracy no longer improves, the training is stopped.

Root mean squared error (RMSE) is used to evaluate the accuracy of regression predictions, especially to avoid large errors. Since each error is squared, large errors are magnified, making this metric extremely sensitive to outliers.

Note	Mean Squared Error(MSE) is one of the most commonly used methods for computing loss, also referred to as L2 Loss. The specific calculation formula is as follows: $\text{MSE} = \frac{\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$ In addition, the calculation loss includes Ll loss, mean absolute error, mean deviation error Hinge loss, Multi class support vector machine loss, etc. Among others, which will not be exhaustively enumerated here. Choosing an appropriate method for calculating losses can help evaluate the performance of the model

2.2 Classification of Machine Learning

This section introduces the classification of machine learning from two perspectives: task types and learning paradigms.

2.2.1 Classified by Task Type

By task type, machine learning can be classified into regression problems, classification problems, clustering problems, and dimensionality reduction problems, etc.

1. Regression Problems

Regression problems are machine learning problems where the labels are continuous values. For instance, predicting the price of a stock, the duration of rainfall, and the outdoor temperature tomorrow, all have continuous labels. Common regression algorithms include linear regression, support vector regression(SVR) and ridge regression, among others.

Regression example: Predict the house price based on the house area, floor, number of bedrooms, and whether it is in a school district. As shown in Table 2.1, predict the prices of the last two houses using the information of the first six houses.

2. Classification Problems

Classification problems are machine learning problems where the labels are a finite number of discrete values. For example, predicting whether it will rain tomorrow, which team will win a football match, and whether a picture shows a cat or a dog, all have labels with a finite number of discrete values. Common classification algorithms include kNN, Decision Tree, Random Forest, and Support Vector Machine.

For instance, when we were kids, our parents would teach us to recognize objects, telling us that a certain animal was a cat, a dog or a pig. Then, an impression of a cat, a

Table 2.1 Housing price prediction information

No	Area (m^2)	Floor	Bedroom	School district	Price ($\times 10^4$ CNY)
0	88	8	3	Yes	65
1	90	5	3	Yes	71.6
2	65	16	2	No	56
3	120	31	4	Yes	150.3
4	75	5	2	No	88.5
5	99	2	3	Yes	110
6	101	6	3	Yes	?
7	85	7	3	Yes	?

dog or a pig would be formed in our brain (equivalent to model construction). When a "new" puppy came along and you could say "This is a puppy", congratulations, you have successfully classified it! But if you said "This is a little pig", the parent would correct your mistake, saying "Sweetie, no, this is a puppy". Through such repeated training, your cognitive system in the brain is constantly updated. The next time you encounter a new "cat, dog, or pig", you can give the correct "prediction" classification. Both classification problems and regression problems fall under supervised learning.

3. Clustering problem

The clustering problem involves dividing a dataset into several non-overlapping subsets or clusters, with the aim of making the elements within each subset as similar as possible. In simple terms, it is about using a model to categorize a dataset into several groups. The main idea is "birds of a feather flock together", which can be regarded as a special case of classification problems. For instance, market segmentation, community analysis, and the analysis of consumer habits all fall into this category. Common clustering algorithms include K-means, DBSCAN, and EM algorithms.

When we were kids, we might not have known what dogs represented, but because of their similarities, we could group them into one category. We could also classify pictures of dogs we had never seen before as dogs. This kind of thinking is called clustering.

4. Dimensionality Reduction Problem

The problem of dimensionality reduction refers to mapping data from a high-dimensional space to a low-dimensional space through some mapping method. The purpose of data dimensionality reduction: intuitively, the dimension is reduced, which is convenient for calculation and visualization. Its deeper significance lies in the extraction and integration of effective information and the elimination of useless information.

Common dimensionality reduction algorithms include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Laplacian Eigenmaps (LE) and Locally Linear Embedding (LLE), etc.

2.2.2 Classified by Learning Methods

1. Supervised Learning

The input of supervised learning consists of data made up of input features and labels. For instance, a training set includes 10,000 pieces of data, but only 1000 of them are labeled (marked). At this point, we can use supervised learning. First, input the 1000 labeled data to let the machine learning model learn, and then use the learned information to classify the remaining 9000 pieces of data. In other words, supervised learning is to let the model learn based on the questions (features) and answers

(labels) in the training set, and predict the most likely answer for the questions in the test set.

Supervised learning can be divided into regression and classification based on the situation of labels.

In fact, the entire field of machine learning is essentially about one thing: obtaining a certain model through training and learning, and then expecting this model to also perform well on "new samples" (i.e., making predictions). The ability of such a model to apply to new samples is also known as "generalization ability", which is a very important characteristic of machine learning algorithms.

2. Unsupervised Learning

Unlike supervised learning, the input of unsupervised learning consists of data with only feature values but no labels, and it is mainly applied to clustering problems and dimensionality reduction problems.

3. Semi-supervised learning

In semi-supervised learning, the training set includes both labeled sample data and unlabeled sample data. That is to say, during the training process, both labeled and unlabeled data are utilized. Semi-supervised learning involves a large amount of unlabeled data and a small amount of labeled data, mainly leveraging the information in the unlabeled data to assist the labeled data in supervised learning.

4. Reinforcement Learning

Reinforcement learning regards learning as a collection of a series of actions. During the learning process, we do not simply and rigidly tell the machine whether the action it takes is right or wrong. This is similar to training a pet. If we tell a puppy to fetch an apple, but it brings back a bunch of bananas instead, and bananas happen to be my favorite fruit, then I cannot say that the puppy's learning action at time t is wrong. Instead, I can only say that at time t, the puppy's learning action is good and give it a reward. After receiving the reward, the puppy might bring back bananas again the next time we want an apple. We would then criticize it for making the wrong "feedback"—at time $t + 1$, the action of fetching bananas is "not good". Thus, unlike the first three types of learning, the labels in reinforcement learning are not fixed but need to be given a good or bad reward based on the current situation (and possibly historical situations).

2.3 Machine Learning and Drug Design

Traditional drug development is confronted with challenges such as long research and development cycles [6], large financial investment, and low success rates in clinical approval. Meanwhile, drug researchers need to handle and analyze massive amounts of information. With the advancement of computer hardware and software,

the development of artificial intelligence theory, and the accumulation of pharmacological data, machine learning, as a powerful data mining tool [7], has been applied to various subfields of drug development, such as target identification, drug design and structure optimization, drug repurposing, property assessment, and clinical trials. This section will elaborate on important machine learning algorithms, basic theories of drug design, and the application of machine learning in ligand and receptor based virtual screening.

As early as the 1990s, methods such as neural network, Support Vector Machine and Random Forest had already been applied to the screening of anti-cancer drugs, protein sequence design and drug design [8]. Since the twenty-first century, with the rapid development of artificial intelligence in the field of drug research and development and the accumulation of a large amount of pharmacological data, pharmaceutical companies have begun to closely cooperate with artificial intelligence companies, promoting the rapid development of this field [9].

2.3.1 Drug Design Methods

Drug design methods are mainly divided into two categories. One category starts from small-molecule structures and obtains new compounds with improved activity and reduced toxicity through structural modification and optimization, which is referred to as indirect drug design. The other category starts from the structures of biological target biomolecules and aims to identify and design small molecules that can interact with these targets and regulate their functions; this approach is known as direct drug design [10]. Indirect drug design mainly includes quantitative structure–activity relationship (QSAR) methods and three-dimensional pharmacophore modeling, whereas direct drug design primarily consists of molecular docking and de novo drug design methods.

QSAR is an indirect drug design method that predicts molecular activity based on chemical structure information. Its fundamental assumption is that the physicochemical properties and biological activities of molecules depend on their structural characteristics, which can be described using various molecular descriptors. By statistically analyzing experimental data, models relating molecular structure to biological activity can be constructed [11]. For example, in a classification study of 131 skin sensitization compounds using support vector machines, the prediction accuracies for the training set and test set reached 89.77% and 72.09%, respectively. QSAR is therefore an important method for lead compound optimization.

Three-dimensional pharmacophore models describe the spatial arrangement of key atoms or functional groups that are capable of forming interactions such as hydrogen bonding, electrostatic interactions, van der Waals interactions, and hydrophobic interactions with receptor binding sites. Pharmacophore modeling is commonly used for the discovery and preliminary screening of lead compounds [12]. By analyzing and comparing the chemical structures of biologically active compounds, common features are identified and used to construct a pharmacophore

model. This approach is often applied prior to molecular docking to reduce computational workload and improve the efficiency of identifying compounds with high biological activity.

Molecular docking involves aligning each small-molecule ligand from a three-dimensional compound database with a target biomolecule. By optimizing the position, orientation, and conformation of the ligand, molecular docking seeks the most favorable binding mode and estimates the interaction energy between the ligand and the target [13]. By ranking compounds based on docking scores, potential ligands capable of binding to the target can be identified. Parallelized molecular docking, also known as high-throughput virtual screening, enables the screening of hundreds of thousands or even millions of compounds within a relatively short time frame, and has become an important complement to experimental high-throughput screening for lead compound identification.

De novo drug design aims to generate novel drug molecules that match the geometric shape and chemical properties of the active sites (also referred to as binding pockets) of target biomolecules [14]. Two main strategies are commonly employed. The fragment linking method places multiple fragments that interact favorably with different regions of the binding pocket and subsequently connects them into a complete molecule. The fragment growing method starts from an initial fragment positioned within the binding pocket and incrementally extends the molecular structure by adding fragments step by step, with each extension evaluated to select the optimal configuration until the final molecule is constructed.

2.3.2 Common Machine Learning Methods in Drug Design

With the continuous development of artificial intelligence, many machine learning methods have played a crucial role in drug design, propelling it into a new stage of development.

1. Support Vector Machine

Support Vector Machine (SVM) is a statistical learning algorithm proposed by Vapnik and Cortes in 1995 [15]. Initially designed for binary classification problems, it is now also applied to multi-classification and regression problems. The core idea of SVM is to find an optimal hyperplane that separates the binary classification data as much as possible [16]. For the data in the test set, the category of the sample is determined based on its relative position to the optimal hyperplane.

Zhao et al. applied the SVM algorithm to predict the concentration ratios of 126 commonly used drugs in the milk and serum of lactating mothers [17]. Two classification models were constructed using the linear discriminant analysis (LDA) algorithm and the SVM algorithm. The classification accuracy rates of these two models for the test set (30 drugs) were 76.7% and 90%, respectively.

Deeb et al. developed QSAR models using SVM and partial least squares (PLS) methods to predict the inhibitory activity of non-peptide HIV-1 protease inhibitors,

and compared the results obtained by SVM with those by PLS. Eventually, it was found that the SVM model was much better than the PLS model [18]. The root mean square errors of the training set and test set of the SVM model were 0.2027 and 0.2751, respectively, and the determination coefficients (R^2) were 0.9800 and 0.9355, respectively.

Shahid et al. [19] applied the SVM-based recursive feature elimination (SVM-RFE) method. This method can be used to predict the pharmacological properties of drugs for treating complex neurodegenerative diseases (NDDs) and also to solve the binary classification problem of NDD drugs and other drugs. Shahid et al. applied the SVM-RFE model to a group of drugs and successfully classified NDD drugs from non-NDD drugs. They conducted a tenfold cross-validation using the top 40 molecular descriptors selected from 314 descriptors, and the overall accuracy of the model was 80%.

SVM algorithms are often used in drug design. In the above three examples, researchers all employed machine learning methods based on SVMs to address relevant issues in drug design.

2. Decision Tree and Random Forest

Decision Tree (DT) is a non-parametric machine learning algorithm that can establish simple rules based on the input of several simple variables and predict the target value accordingly, thereby solving classification problems.

Random Forest (RF) is an ensemble form of the DT algorithm. As shown in Fig. 2.5, it is an algorithm that predicts classification problems by training multiple DTs. Each DT provides a corresponding prediction result, and the final classification is determined based on the principle of "the minority yields to the majority". This can reasonably reduce the randomness. Meanwhile, to ensure that the outputs of each DT have certain differences, a certain proportion of features are randomly selected from the input features to avoid overfitting and improve generalization ability [20].

DT-based machine learning methods have been adopted to predict the affinity in a specific drug design problem involving approximately 160,000 samples. Ultimately, compared with traditional methods, this approach can extract more protein–ligand binding information and increase the screening efficiency of drug design by 200–1000 times.

3. k-Nearest Neighbor Method

The k-nearest neighbor method (kNN) was initially proposed by Cover and Hart in 1967. The core idea of this method is that if the majority of the k-nearest samples of a sample in the feature space belong to a certain category, then this sample also belongs to this category and has the characteristics of the samples in this category. When making classification decisions, the sample data is only related to a small number of adjacent samples, so it can well solve the problem of sample imbalance [21]. The drawback of the kNN method is that it has a large amount of computation, as it requires calculating the distance between the sample and the remaining samples to obtain the k-nearest neighbor points.

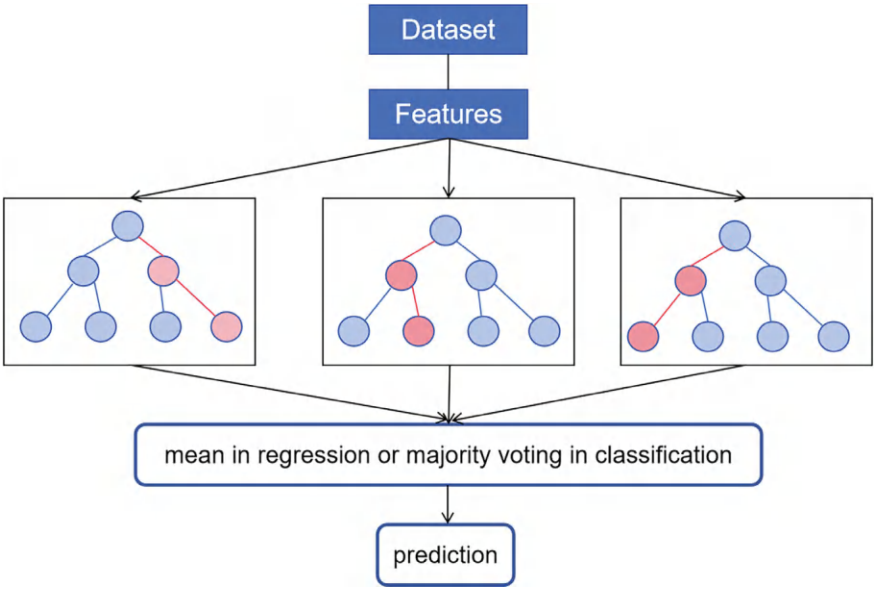


Fig. 2.5 Schematic diagram of Random Forest

4. Naive Bayes Classifier

The Naive Bayes Classifier (NBC) is a simple probabilistic classifier based on the Bayesian theorem with the assumption of independence. It has a solid mathematical foundation and stable classification efficiency, and is one of the widely used classification methods in machine learning. Meanwhile, the NBC model requires few parameters to be estimated, is not very sensitive to missing data, and the algorithm is relatively simple.

2.3.3 Construction of Prediction Models

This section provides an overview of the general workflow for constructing machine learning–based prediction models, including data collection, feature engineering, model selection, and evaluation. Each step plays a crucial role in ensuring the reliability and performance of the final model.

1. Data collection

A comprehensive and accurate dataset is the key to building a predictive model. First, based on the problem, we can search for relevant datasets through various channels (including various databases, related websites, published literature and books, etc.). At the same time, we need to handle abnormal data such as missing, duplicate and

outlier values to improve the quality of the data, and randomly divide the dataset into training set, validation set and test set in a certain proportion.

2. Data description

The collected data is usually not directly applicable for model learning. Therefore, we need to further process the data. In drug design, we often convert data such as molecular structures and protein sequences into features for model learning through molecular descriptors (such as molecular weight, number of atoms, and hydrophilicity/hydrophobicity, etc.) and molecular fingerprints. This is the key point of drug design based on machine learning.

3. Feature selection

During the data collection and data description stages, features are often redundant and irrelevant. To ensure the predictive ability of the model, we need to remove irrelevant and repetitive features before training the model. The purpose of doing this is twofold: first, to reduce the number of features and lower the dimensionality, making the model's generalization ability stronger and avoiding overfitting. Second, to enhance the understanding of the relationship between features and their values. Feature selection is not a necessary process and can be omitted. It is usually applied when there are too many features or when there is too much overlap among features in the data.

4. Model construction

During the model construction phase, we need to determine whether the problem to be solved is a regression problem or a classification problem. We also need to select an appropriate algorithm based on the problem type, data type and quantity, and set reasonable initial values. For regression prediction tasks, logistic regression algorithms and artificial neural networks should be used more often; for classification and discrimination tasks, algorithms such as SVM, DT, RF and artificial neural network should be used more frequently; while for generative tasks, deep learning networks are more applicable.

Following the above steps, we can solve most of the problems in drug design. However, the application of machine learning in the field of drugs is still in its early stage at present. In some areas, there are immature and unexplainable situations, which need to be further verified through experiments to fully prove the reliability of machine learning in the field of drug design. Therefore, we should attach importance to the explainability and reproducibility of the results; otherwise, it may restrict the further development of machine learning in this field.

With the continuous development of artificial intelligence, there have been many examples of using machine learning and deep learning methods to solve problems in drug design (drug design and optimization). Although this approach can overcome the problems of long research and development cycles and high costs in traditional drug research and development, it also faces the challenges of mining massive amounts of data and the need to use different machine learning models for different drug research and development.

References

1. Simon, H.A.: Studying Human Intelligence by creating artificial Intelligence: when considered as a physical symbol system, the human brain can be fruitfully studied by computer simulation of its processes. Am. Sci. **69**, 300–309 (1981)
2. Sarker, I.H.: Machine learning: algorithms, real-world applications and research directions. SN Comput. Sci. **2**, 160 (2021)
3. Shao, F., Shen, Z.: How can artificial neural networks approximate the brain? Front. Psychol. **13**, 970214 (2023)
4. Basheer, I.A., Hajmeer, M.: Artificial neural networks: fundamentals, computing, design, and application. J. Microbiol. Methods **43**, 3–31 (2000)
5. Jiang, X., Hu, Z., Wang, S., Zhang, Y.: Deep learning for medical image-based cancer diagnosis. Cancers **15**, 3608 (2023)
6. Ngo, L.T., Okogun, J.I., Folk, W.R.: 21st century natural product research and drug development and traditional medicines. Nat. Prod. Rep. **30**, 584–592 (2013)
7. Gupta, R., et al.: Artificial intelligence to deep learning: machine intelligence approach for drug discovery. Mol. Diversity **25**, 1315–1360 (2021)
8. Nayarisseri, A., et al.: Artificial intelligence, big data and machine learning approaches in precision medicine and drug discovery. Curr. Drug Targets **22**, 631–655 (2021)
9. Kolluri, S., Lin, J., Liu, R., Zhang, Y., Zhang, W.: Machine learning and artificial intelligence in pharmaceutical research and development: a review. AAPS J. **24**, 19 (2022)
10. Singh, S., Malik, B.K., Sharma, D.K.: Molecular drug targets and structure based drug design: a holistic approach. Bioinformation **1**, 314 (2006)
11. Schuur, J.H., Selzer, P., Gasteiger, J.: The coding of the three-dimensional structure of molecules by molecular transforms and its application to structure-spectra correlations and studies of biological activity. J. Chem. Inf. Comput. Sci. **36**, 334–344 (1996)
12. Qing, X., et al.: Pharmacophore modeling: advances, limitations, and current utility in drug discovery. J. Recept. Ligand Channel Res. **7**, 81–92 (2014)
13. Decherchi, S., Cavalli, A.: Thermodynamics and kinetics of drug-target binding by molecular simulation. Chem. Rev. **120**, 12788–12833 (2020)
14. Honarparvar, B., Govender, T., Maguire, G.E., Soliman, M.E., Kruger, H.G.: Integrated approach to structure-based enzymatic drug design: molecular modeling, spectroscopy, and experimental bioactivity. Chem. Rev. **114**, 493–537 (2014)
15. Liu, X., Gao, C., Li, P.: A comparative analysis of support vector machines and extreme learning machines. Neural Netw. **33**, 58–66 (2012)
16. Mathur, A., Foody, G.M.: Multiclass and binary SVM classification: Implications for training and classification users. IEEE Geosci. Remote Sens. Lett. **5**, 241–245 (2008)
17. Zhao, C., et al.: Prediction of milk/plasma drug concentration (M/P) ratio using support vector machine (SVM) method. Pharm. Res. **23**, 41 (2006)
18. Deeb, O., Goodarzi, M.: Exploring QSARs for inhibitory activity of non-peptide HIV-1 protease inhibitors by GA-PLS and GA-SVM. Chem. Biol. Drug Des. **75**, 506–514 (2010)
19. Shahid, M., Shahzad Cheema, M., Klenner, A., Younesi, E., Hofmann-Apitius, M.: SVM based descriptor selection and classification of neurodegenerative disease drugs for pharmacological modeling. Mol. Inf. **32**, 241–249 (2013)
20. García, S., Fernández, A., Herrera, F.: Enhancing the effectiveness and interpretability of decision tree and rule induction classifiers with evolutionary training set selection over imbalanced problems. Appl. Soft Comput. **9**, 1304–1314 (2009)
21. Rendon, E., Alejo, R., Castorena, C., Isidro-Ortega, F.J., Granda-Gutierrez, E.E.: Data sampling methods to deal with the big data multi-class imbalance problem. Appl. Sci. **10**, 1276 (2020)

Chapter 3
Support Vector Machine

Support vector machine can seek the best compromise between the complexity of the model and its learning ability based on limited sample information, thereby enabling the model to achieve the best generalization ability. Support vector machine exhibits many unique advantages in solving small sample, nonlinear and high-dimensional pattern recognition problems, and are widely applied to other machine learning issues such as function fitting.

3.1 Introduction to Support Vector Machine

Support Vector Machine (SVM) was proposed by Cortes et al. in 1995. Because of its excellent performance in text classification tasks, SVM soon became the mainstream technology of machine learning, and directly set off the climax of "statistical learning" around 2000. But in fact, the concept of SVM appeared as early as the 1960s [1], and statistical learning theory was formed in the 1970s. The research on kernel function is even earlier. Mercer's theorem can be traced back to 1909, while RKHS has been studied since the 1940s. However, after the rise of statistical learning, kernel function technology really became the general basic technology of machine learning.

SVM is a new machine learning method based on statistical learning theory and VC (proposed by Vapnik and Chervonenkis) dimension theory. According to the limited sample information, we can find the best compromise between the complexity of the model and the learning ability in order to obtain the best generalization ability. SVM has a strong theoretical basis, which can ensure that the extremum solution found is a global optimal solution rather than a local minimum, which determines that SVM method has a good generalization ability for unknown samples. Because of these advantages, SVM can be well applied to many fields in pattern recognition,

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_3

such as handwritten number recognition, text classification, image classification and recognition [2].

SVM is a pattern classification method based on analytical statistical theory, and it is also a specific implementation of structural risk minimization (SRM). The most important work of studying SVM is to analyze the essential characteristics of SVM. It was originally developed from the optimal classification problem in linear analysis problems. After solving the problem of pattern recognition, it was then extended to the fields of functional regression and density estimation. The SVM algorithm is widely used. In addition to being used in the field of pattern recognition, the SVM algorithm can also be widely used in natural language processing technology [3]. It can be seen that the role of the SVM algorithm will become increasingly prominent. According to different models, SVMs can be divided into linear SVM and nonlinear SVM [4]. SVM is based on the principle of SRM. One of its core ideas is to introduce kernel function technology, which ingeniously solves the "dimension disaster" and other problems in high-dimensional feature space. Because the kernel function determines the nonlinear processing ability of SVM, the selection of kernel function and its parameters plays an important role in SVM and is also one of the hot research directions of SVM. However, different kernels have different characteristics. Choosing different kernels will lead to different generalization performance of SVM.

3.2 Interval and Support Vector

Given a training dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, $y_i \in \{-1, +1\}$, the most fundamental idea in classification learning is to find a separating hyperplane in the sample space based on the training set. A separating hyperplane refers to a straight line that divides the training set by separating samples of different categories. However, there can be multiple such separating hyperplanes capable of partitioning the training samples, as illustrated in Fig. 3.1.

The hyperplane positioned "centrally" between the two classes of training samples (represented by the bold black line in Fig. 3.1) exhibits the highest tolerance to local perturbations in the training data. For instance, due to limitations or noise in the training set, out-of-sample data points may lie closer to the class separation boundary than the training samples. This proximity would cause errors in many separating hyperplanes, whereas the bold black hyperplane remains minimally affected. In other words, this hyperplane produces the most robust classification results and demonstrates the strongest generalization capability for unseen instances.

For a data point, the farther it is from the hyperplane, the more reliable its final prediction result is. Therefore, it is necessary to find some points that are closest to the hyperplane and ensure that their distances from the hyperplane are as far as possible. The distance from these points to the dividing hyperplane is called the margin. The points that are closest to the dividing hyperplane are called support vectors. Thus, the problem of finding the dividing hyperplane is transformed into the problem of finding the maximum margin.

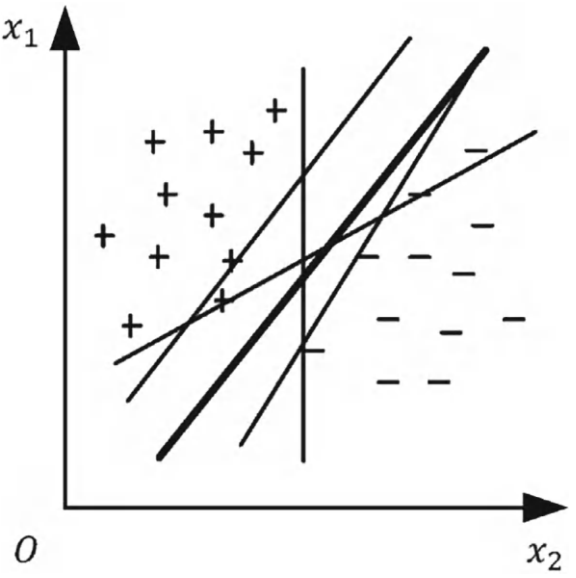


Fig. 3.1 Multiple separating hyperplanes for two-class samples

In the sample space, the partitioning hyperplane can be described by the linear equation in Eq. (3.1).

$$\omega^{\mathrm{T}}\mathrm{x} + \mathrm{b} = 0 \tag{3.1}$$

Let $\omega = (\omega_1, \omega_2, \ldots, \omega_d)$ be the normal vector, which determines the orientation of the hyperplane, and b be the displacement term, which determines the distance between the hyperplane and the origin. Denoting the hyperplane as (ω, b), the distance from any point x in the sample space to the hyperplane (ω, b) can be expressed as:

$$d = \frac{\left|\omega^{\mathrm{T}}\mathrm{x}+\mathrm{b}\right|}{\omega} \tag{3.2}$$

Suppose the hyperplane (ω, b) can correctly classify the training samples. Then, for $(x_i, y_i) \in D$, if $y_i = +1$, we have $\omega^T x_i + b \geq 0$, if $y_i = -1$, we have $\omega^T x_i + b < 0$. As shown in Eq. (3.3).

$$\begin{cases} \omega^{\mathrm{T}}\mathrm{x}_i + b \geq +1, \mathrm{y}_i = +1 \\ \omega^{\mathrm{T}}\mathrm{x}_i + b \leq -1, \mathrm{y}_i = -1 \end{cases} \tag{3.3}$$

As shown in Fig. 3.2, each sample point corresponds to a feature vector. The few training sample points (support vectors) closest to the hyperplane make the equality in Eq. (3.3) hold. The sum of the distances from the two different-class support vectors to the hyperplane is:

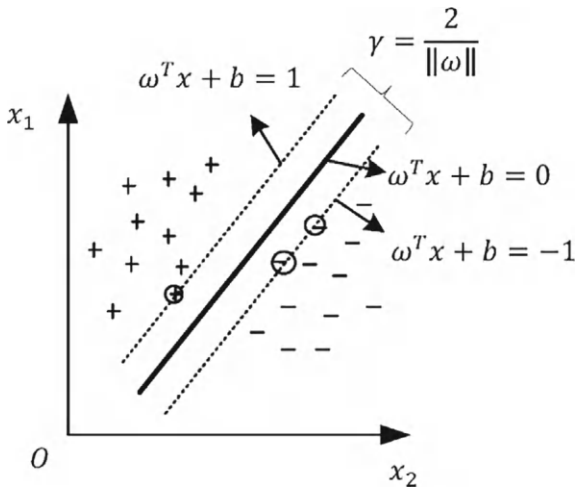


Fig. 3.2 Support vectors and margins

$$\gamma = \frac{2}{\|\omega\|} \tag{3.4}$$

It is called "margin".

The division hyperplane of (maximum margin) is to find one that can satisfy Eq. (3.3). The parameters ω and b under constraints are determined to maximize γ as shown in Eq. (3.4).

$$\begin{aligned} &\min_{\omega,b} x \frac{2}{\|\omega\|} \\ &s.t. y_i\left(\omega^T x_i + b\right) \geq 1, i = 1, 2, \ldots, m \end{aligned} \tag{3.5}$$

Obviously, to maximize the margin, it is only necessary to maximize ω^{-1}, which is equivalent to minimizing ω^2. Thus, Eq. (3.5) can be rewritten as Eq. (3.6), that is.

$$\begin{aligned} &\min_{\omega,b} x \frac{1}{2}\|\omega\|^2 \\ &s.t. y_i\left(\omega^T x_i + b\right) \geq 1, i = 1, 2, \ldots, m \end{aligned} \tag{3.6}$$

This is the basic form of the SVM.

3.3 Kernel Function

The kernel function directly determines the final performance of support vector machines (SVMs) and kernel-based methods [5]. However, the selection of kernel functions remains an unsolved problem. Multiple kernel learning (MKL) addresses

this by employing multiple kernel functions and learning their optimal convex combination as the final kernel function, which essentially leverages ensemble learning mechanisms.

In previous discussions, we assume that the training samples are linearly separable. That is to say, a separating hyperplane exists that can correctly classify the training samples. Yet in real-world scenarios, there might be no such hyperplane within the original sample space that can perfectly separate two classes of samples.

To address such issues, samples can be mapped from the original space to a higher-dimensional feature space, enabling linear separability within this new space. For example, as illustrated in Fig. 3.3, mapping a two-dimensional space to three dimensions allows the identification of a suitable separating hyperplane. Moreover, if the original space is finite-dimensional (i.e., with limited attributes), there always exists a high-dimensional feature space where the samples become separable.

Let $\phi(x)$ denote the feature vector obtained by mapping x. Then, the model corresponding to the hyperplane division in the feature space can be expressed as:

$$f(x) = \omega^T \Phi(x) + b \tag{3.7}$$

Among them, ω and b are model parameters. Similar to Eq. (3.6), there is

$$\begin{aligned} &\min_{\omega,b} x \frac{1}{2}\|\omega\|^2 \\ &s.t. y_i\left(\omega^T \Phi(x_i) + b\right) \geq 1, i = 1, 2, \ldots, m \end{aligned} \tag{3.8}$$

By applying the Lagrange multiplier method to Eq. (3.8), its "dual problem" can be obtained. Specifically, for each constraint in Eq. (3.8), a Lagrange multiplier $\alpha_i \geq 0$ is added. Then, the Lagrangian function of this problem can be written as

$$L(\omega, b, \alpha) = \frac{1}{2}\|\omega\|^2 + \sum_{i}^{m} \alpha_i\left(1 - y_i\left(\omega^T \Phi(x) + b\right)\right) \tag{3.9}$$

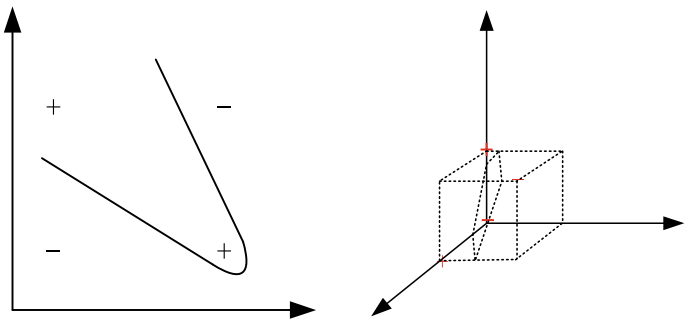


Fig. 3.3 XOR problem and nonlinear mapping

Among them, let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$, and by setting the partial derivatives of $L(\omega, b, \alpha)$ with respect to ω and b to zero, we can obtain

$$\omega = \sum_{i=1}^{m} a_i y_i \Phi(x_i) \tag{3.10}$$

$$0 = \sum_{i=1}^{m} \alpha_i y_i \tag{3.11}$$

Substituting Eq. (3.10) into Eq. (3.9) eliminates the sum therein. Considering the constraint of Eq. (3.11), the dual problem of Eq. (3.6) is thus obtained, that is

$$\begin{aligned} &\min_{\alpha} x \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \Phi\left(x_i^T\right) \Phi\left(x_j\right) \\ &s.t. \sum_{j=1}^{m} \alpha_i y_i = 0, \\ &\alpha_i \geq 0, i = 1, 2, \ldots, m \end{aligned} \tag{3.12}$$

At this point, solving Eq. (3.12) involves calculating $\phi\left(x_i^T\right)\phi\left(x_j\right)$, which is the inner product of samples x_i and x_j after mapping to the feature space. Since the dimension of the feature space may be very high or even infinite, directly calculating $\phi\left(x_i^T\right)\phi\left(x_j\right)$ is extremely difficult. To circumvent this obstacle, we can assume a function as shown in Eq. (3.13), that is

$$k\left(x_i, x_j\right) = \left\langle \Phi(x_i), \Phi\left(x_j\right) \right\rangle = \Phi(x_i)^T \Phi\left(x_j\right) \tag{3.13}$$

The inner product of x_i and x_j in the feature space is equal to the result calculated through the function κ in the original sample space. At this point, we do not need to directly calculate the inner product in the high-dimensional or even infinite-dimensional feature space. Thus, Eq. (3.12) can be rewritten as

$$\begin{aligned} &\min_{\alpha} x \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa\left(x_i, x_j\right) \\ &s.t. \sum_{j=1}^{m} \alpha_i y_i = 0, \\ &\alpha_i \geq 0, i = 1, 2, \ldots, m \end{aligned} \tag{3.14}$$

After solving, Eq. (3.15) can be obtained.

$$f(x) = \omega^T \Phi(x) + b$$

$$= \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i)^T \Phi(x) + b$$
$$= \sum_{i=1}^{m} \alpha_i y_i \kappa\left(x_i, x_j\right) + b \tag{3.15}$$

Here, the function κ is the "kernel function". Equation (3.15) shows that the optimal solution of the model can be expanded through the kernel function of the training samples. This expansion can also be called the "support vector expansion".

Obviously, if the specific form of the appropriate mapping $\phi(\cdot)$ is known, the kernel function κ can be written out. However, in practical tasks, we usually do not know what form $\phi(\cdot)$ takes or what kind of function can serve as a kernel function. Therefore, the following theorem holds.

Theorem 3.1 (Kernel Function) *Let X be the input space and let κ be a symmetric function defined on $X \times X$. Then, κ is a kernel function if and only if for any data set $D = \{x_1, x_2, \ldots, x_m\}$. The kernel matrix κ is always semi-positive definite, such as*

$$\kappa = \begin{Bmatrix} \kappa(x_1, x_1) & \cdots & \kappa\left(x_1, x_j\right) & \cdots & \kappa(x_1, x_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(x_i, x_1) & \cdots & \kappa\left(x_i, x_j\right) & \cdots & \kappa(x_i, x_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(x_m, x_1) & \cdots & \kappa\left(x_m, x_j\right) & \cdots & \kappa(x_m, x_m) \end{Bmatrix} \tag{3.16}$$

Theorem 3.1 indicates that as long as the kernel matrix corresponding to a symmetric function is semi-positive definite, it can be used as a kernel function. In fact, for any semi-positive definite kernel matrix, we can always find a mapping ϕ corresponding to it. In other words, any kernel function implicitly defines a feature space called the "reproducing kernel Hilbert space".

Through the previous discussion, we hope that the samples are linearly separable in the feature space, as the quality of the feature space is crucial to the performance of the SVM. However, we do not know the form of the feature mapping, and thus do not know what kind of kernel function is appropriate. The kernel function only implicitly defines this feature space. For this reason, "kernel function selection" is regarded as the greatest variable of the SVM. If the kernel function is not chosen appropriately, it means that the samples are mapped to an inappropriate feature space, which is very likely to lead to poor model performance.

Table 3.1 lists four commonly used kernel functions. The linear kernel is the simplest kernel function, and kernel algorithms using the linear kernel are usually equivalent to their non-kernel counterparts. For text data processing, the linear kernel is typically adopted, and when the situation is unclear, the Gaussian kernel can be tried first. In the polynomial kernel function, when $d = 1$, it degenerates into a linear kernel. The polynomial kernel function is suitable for all problems where the training data is normalized. The Gaussian kernel, also known as the radial basis function

Table 3.1 Commonly used Kernel functions

Name	Expressions and parameters
Linear Kernel	$\kappa(x_i, x_j) = x_i^T x_j$
Polynomial Kernel	$\kappa(x_i, x_j) = (x_i^T x_j)^d$, $d \geq 1$ is polynomial degree
Gaussian Kernel	$\kappa(x_i, x_j) = \exp\left(-\frac{x_i - x_j^2}{2\sigma^2}\right)$, $\sigma > 0$ is Gaussian Kernel Bandwidth
Laplacian Kernel	$\kappa(x_i, x_j) = \exp\left(-\frac{x_i - x_j}{\sigma}\right)$, $\sigma > 0$

(RBF) kernel, is an example of a radial basis function kernel. The adjustable parameter sigma plays a major role in the performance of the kernel. If it is overestimated, the exponent will be almost linear, and the high-dimensional projection will start to lose its nonlinear power; if it is underestimated, the function will lack regularization, and the decision boundary will be highly sensitive to noise in the training data. The Laplacian kernel function can construct nonlinear decision boundaries and map input sample points to a high-dimensional feature space, making originally linearly inseparable problems become linearly separable in the high-dimensional space.

3.4 Soft Margin and Regularization

In the previous discussion, we have known that it is assumed that the training samples are linearly separable in the sample space or feature space, that is, there exists a hyperplane that can completely separate samples of different classes. However, in real tasks, it is often difficult to determine an appropriate kernel function to make the training samples linearly separable in the feature space [6]. Even if a kernel function is found that makes the training samples linearly separable in the feature space, it is still hard to determine whether this seemingly linearly separable result is caused by overfitting.

One way to alleviate this problem is to allow the SVM to make mistakes on some samples. To this end, we introduce "soft margin". The concept of (soft margin), as shown in Fig. 3.4.

Specifically, the form of the SVM introduced in the previous text requires that all samples satisfy the constraint of Eq. (3.3), that is, all samples must be correctly classified. This is called "hard margin". Soft margin, on the other hand, allows some samples not to satisfy the constraint of Eq. (3.17), that is,

$$y_i(\omega^T x_i + b) \geq 1 \tag{3.17}$$

Of course, while maximizing the margin, the number of samples that do not satisfy the constraints should be as small as possible. Thus, the optimization objective can be written as

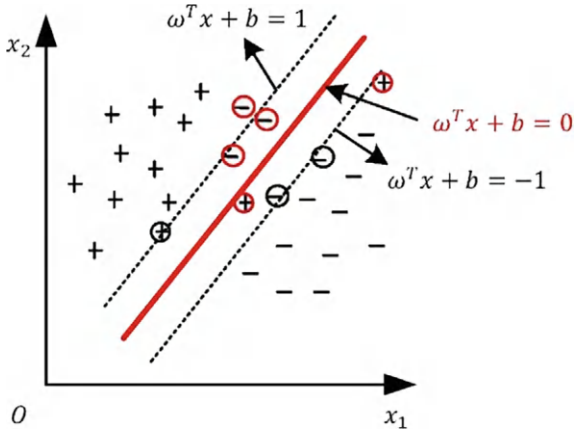


Fig. 3.4 Schematic diagram of soft margin with constraint-violating samples

$$\min_{\omega,b} \frac{1}{2}\omega^2 + C\sum_{i=1}^{m} \ell_{0/1}\left(y_i\left(\omega^T x_i + b\right) - 1\right) \tag{3.18}$$

Here, $C > 0$ is a constant greater than 0, and $\ell_{0/1}$ represents the "0/1 loss function".

$$\ell_{0/1}(z) = \begin{cases} 1, \ if \ z < 0; \\ 0, \ otherwise. \end{cases} \tag{3.19}$$

Obviously, when C is infinite, Eq. (3.18) forces all samples to satisfy the constraint of Eq. (3.17), and thus Eq. (3.18) is equivalent to Eq. (3.6); when C takes a finite value, Eq. (3.18) allows some samples not to satisfy the constraint.

However, the mathematical properties of the $\ell_{0/1}$ loss function are not good as it is non-convex and discontinuous, making it difficult to directly solve Eq. (3.18). Therefore, researchers usually replace the $\ell_{0/1}$ loss function with some other functions, which are called surrogate loss functions. These surrogate loss functions generally have better mathematical properties, such as being convex and continuous functions and being upper bounds of the $\ell_{0/1}$ loss function. The three commonly used surrogate loss functions are the hinge loss function, the exponential loss function, and the logistic loss function, as shown in Eq. (3.20).

$$\begin{aligned} &\ell_{hinge}(z) = \max(0, 1 - z) \\ &\ell(z)\exp(-z)_{exp} \\ &\ell(z)\log(1 + \exp(-z))_{log} \end{aligned} \tag{3.20}$$

If the hinge loss is adopted, then Eq. (3.18) becomes Eq. (3.21).

$$\min_{\omega,b} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m} \max\left(0, 1 - y_i\left(\omega^T x_i + b\right)\right) \tag{3.21}$$

Introducing the "slack variable" $\xi_i \geq 0$, Eq. (3.21) can be rewritten as Eq. (3.22).

$$\min_{\omega,b,\xi_i} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i \tag{3.22}$$

This is the commonly used "soft margin SVM".

Obviously, each sample in Eq. (3.22) has a corresponding slack variable to represent the degree to which the sample does not satisfy the constraint (3.17). However, similar to Eq. (3.6), this is still a quadratic programming (QP) problem. Thus, similar to Eq. (3.9), the Lagrangian function of Eq. (3.22) can be obtained through the Lagrange multiplier method, that is

$$\begin{aligned} L(\omega, b, \alpha, \xi, \mu) = \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i\left(1 - \xi_i - y_i\left(\omega^T x_i + b\right)\right) \\ - \sum_{i=1}^{m}\mu_i\xi_i \end{aligned} \tag{3.23}$$

Among them, $\alpha_i \geq 0$, $\mu_i \geq 0$ are Lagrange multipliers.

Let the partial derivatives of $L(\omega, b, \alpha, \xi, \mu)$ with respect to ω, b, ξ_i be set to 0, then we can obtain

$$\omega = \sum_{i=1}^{m}\alpha_i y_i x_i \tag{3.24}$$

$$0 = \sum_{i=1}^{m}\alpha_i y_i \tag{3.25}$$

$$C = \alpha_i + \mu_i \tag{3.26}$$

Substituting Eq. (3.24) to (3.26) into Eq. (3.23), the dual problem of Eq. (3.22) can be obtained.

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{i=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j \\ &s.t. \sum_{i=1}^{m}\alpha_i y_i = 0, \\ &0 \leq \alpha_i \leq C, i = 1, 2, \ldots, m \end{aligned} \tag{3.27}$$

By comparing the contrastive form of Eq. (3.27) with the dual problem under hard margin (Eq. (3.12)), it can be seen that the only difference lies in the constraints on the dual variables: the former is $0 \leq \alpha_i \leq C$, while the latter is $0 \leq \alpha_i$. Therefore, the same algorithm can be used to solve Eq. (3.27), and after introducing the kernel function, the same support vector expansion as Eq. (3.15) can be obtained.

For the soft margin SVM, the KKT (Karush–Kuhn–Tucker) conditions require that the constraints shown in Eq. (3.28) be satisfied

$$\begin{cases} a_i \geq 0, \mu_i \geq 0 \\ y_i f(x_i) - 1 + \xi_i \geq 0 \\ a_i(y_i f(x_i) - 1 + \xi_i) = 0 \\ \xi_i \geq 0, \mu_i \xi_i = 0 \end{cases} \tag{3.28}$$

Therefore, for any training sample (x_i, y_i), there is always either $\alpha_i = 0$ or $y_i f(x_i) = 1 - \xi_i$. If $\alpha_i = 0$, this sample will have no impact on $f(x)$; if $\alpha_i > 0$, then it must be that $y_i f(x_i) = 1 - \xi_i$, meaning this sample is a support vector. From Eq. (3.26), if $\alpha_i < C$, then $\mu_i > 0$, and thus $\xi_i = 0$, indicating that this sample lies exactly on the maximum margin boundary; if $\alpha_i = C$, then $\mu_i = 0$, and if $\xi_i \leq 1$, this sample is within the maximum margin; if $\xi_i > 1$, this sample is misclassified. From this, it can be seen that the final model of the soft margin SVM is only related to the support vectors, that is, by using the hinge loss function, sparsity is still maintained.

Then, can we use other alternative loss functions for Eq. (3.18). It can be found that if the log loss function ℓ_{log} is used to replace the 0/1 loss function in Eq. (3.18), then almost the logistic regression model is obtained. In fact, the optimization objectives of SVMs and logistic regression are similar, and usually their performance is comparable. The main advantage of logistic regression is that its output has a natural probabilistic meaning, that is, it provides both the predicted label and the probability. However, the output of SVMs does not have a probabilistic meaning, and special processing is required to obtain probability outputs. Logistic regression models can be directly applied to multi-classification tasks, while SVMs need to be extended for this purpose. In addition, the hinge loss has a "flat" section. The zero region of this makes the solution of the SVM sparse, while the logistic loss is a smooth and monotonically decreasing function, which cannot derive a probability similar to that of the support vector. Therefore, the solution of logistic regression depends on more training samples and has a greater prediction cost.

We can also replace the 0/1 loss function in Eq. (3.18) with other alternative loss functions to obtain other learning models. The properties of these models are directly related to the alternative functions used, but they have one thing in common: one of the optimization objectives is used to describe the size of the "interval" that divides the hyperplane, and the other is $\sum_{i=1}^{m} \ell(f(x_i), y_i)$ used to describe the error on the training set, which can be written in a more general form, as shown in Eq. (3.29):

$$\min_f \Omega(f) + C \sum_{i=1}^{m} \ell(f(x_i), y_i) \tag{3.29}$$

Among them, $\Omega(f)$ is referred to as the "structural risk", which is used to describe certain properties of the model f. the second term $\sum_{i=1}^{m} \ell(f(x_i), y_i)$ is called the "empirical risk", which describes the degree of fit between the model and the training

data. C is used to balance the two. From the perspective of minimizing empirical risk, $\Omega(f)$ expresses our desire to obtain a model with certain properties (for example, a model with lower complexity), which provides a way to incorporate domain knowledge and user intent. Additionally, this information helps to reduce the hypothesis space, thereby reducing the risk of overfitting when minimizing the training error. From this perspective, Eq. (3.29) is referred to as a "regularization" problem, where $\Omega(f)$ is the regularization term and C is the regularization constant. L_p norms are commonly used regularization terms, among which the L_2 norm $\|\omega\|_2$ tends to ω make the component values of ω as balanced as possible, that is, to have as many non-zero components as possible, while the L_0 norm $\|\omega\|_0$ and L_1 norm $\|\omega\|_1$ tend to make the components of ω as sparse as possible, that is, to have as few non-zero components as possible.

3.5 Support Vector Regression

Now let's consider the regression problem. Given the training samples $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, $y_i \in R$. we aim to learn a regression model such that $f(x)$ is as close as possible to y. Here, ω and b are the model parameters to be determined.

For a sample (x, y), traditional regression models typically calculate the loss directly based on the difference between the model output $f(x)$ and the true output y. The loss is zero only when $f(x)$ is exactly equal to y. In contrast, Support Vector Regression (SVR) assumes that we can tolerate a maximum deviation of ς between $f(x)$ and y. That is, the loss is calculated only when the absolute difference between $f(x)$ and y is greater than ς. As shown in Fig. 3.5, this is equivalent to constructing an interval band of width 2ς centered on $f(x)$. If the training sample falls within this interval band, it is considered to be predicted correctly.

Thus, the SVR problem can be formalized as Eq. (3.30), that is,

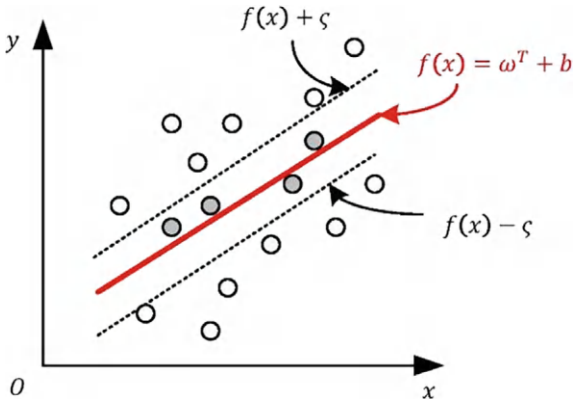


Fig. 3.5 Schematic diagram of support vector regression

$$\min_{\omega,b} \tfrac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m} \ell_\varsigma(f(x_i), y_i) \tag{3.30}$$

Here, C is the regularization constant, and ℓ_ς is the ς -insensitive loss function, as shown in Eq. (3.31):

$$\ell_\varsigma(Z) = \begin{cases} 0, & if\ |z| \le \varsigma \\ |z| - \varsigma, & otherwise \end{cases} \tag{3.31}$$

Introducing the slack variables ξ_i and $\hat{\xi}_i$, Eq. (3.30) can be rewritten as Eq. (3.32).

$$\begin{aligned} &\min_{\omega,b,\xi_i,\hat{\xi}_i} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\left(\xi_i + \hat{\xi}_i\right) \\ &s.t.\quad f(x_i) - y_i \le \varsigma + \xi_i \\ &y_i - f(x_i) \le \varsigma + \xi_i \\ &\xi_i \ge 0, \hat{\xi}_i \ge 0, i = 1, 2, \ldots, m \end{aligned} \tag{3.32}$$

Similar to Eq. (3.23), by introducing the Lagrange multiplier $\mu_i \ge 0$, $\hat{\mu}_i \ge 0$, $\alpha_i \ge 0$, $\hat{\alpha}_i \ge 0$, the Lagrange function of Eq. (3.33) can be obtained through the Lagrange multiplier method, that is

$$\begin{aligned} &L\left(\omega, b, \alpha, \hat{\alpha}, \xi, \hat{\xi}, \mu, \hat{\mu}\right) \\ &= \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\left(\xi_i + \hat{\xi}_i\right) - \sum_{i=1}^{m}\mu_i\xi_i - \sum_{i=1}^{m}\hat{\mu}_i\hat{\xi}_i \\ &+ \sum_{i=1}^{m}\alpha_i(f(x_i) - y_i - \varsigma - \xi_i) + \sum_{i=1}^{m}\hat{\alpha}_i\left(y_i - f(x_i) - \varsigma - \hat{\xi}_i\right) \end{aligned} \tag{3.33}$$

Substituting the dual formula and setting the partial derivatives of $L\left(\omega, b, \alpha, \hat{\alpha}, \xi, \hat{\xi}, \mu, \hat{\mu}\right)$ with respect to ω, b, ξ_i and $\hat{\xi}_i$ to zero, we obtain Eq. (3.34).

$$\begin{aligned} \omega &= \sum_{i=1}^{m}\left(\hat{\alpha}_i - \alpha_i\right)x_i \\ 0 &= \sum_{i=1}^{m}\left(\hat{\alpha}_i - \alpha_i\right) \\ C &= \alpha_i + \mu_i \end{aligned}$$

$$C = \hat{\alpha}_i + \hat{\mu}_i \tag{3.34}$$

Substituting Eq. (3.34) into Eq. (3.33) yields the dual problem of SVR as shown in Eq. (3.35):

$$\begin{aligned}
&\max_{\alpha,\hat{\alpha}} \sum_{i=1}^{m} y_i(\hat{\alpha}_i - \alpha_i) - \varsigma(\hat{\alpha}_i + \alpha_i) \\
&- \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\hat{\alpha}_i - \alpha_i)\left(\hat{\alpha}_j - \alpha_j\right)x_i^T x_j \\
&s.t. \sum_{i=1}^{m}(\hat{\alpha}_i - \alpha_i) = 0 \\
&0 \le \alpha_i, \hat{\alpha}_i \le C
\end{aligned} \tag{3.35}$$

The above process satisfies the Karush–Kuhn–Tucker (KKT) conditions, it is required to meet the constraints of Eq. (3.36):

$$\begin{cases}
\alpha_i(f(x_i) - y_i - \varsigma - \xi_i) = 0 \\
\hat{\alpha}_i\left(y_i - f(x_i) - \varsigma - \hat{\xi}_i\right) = 0 \\
\alpha_i\hat{\alpha}_i = 0, \xi_i\hat{\xi}_i = 0 \\
(C - \alpha_i)\xi_i = 0, (C - \hat{\alpha}_i)\hat{\xi}_i = 0
\end{cases} \tag{3.36}$$

It can be seen that α_i can take a non-zero value if and only if $f(x_i) - y_i - \varsigma - \xi_i = 0$; and $\hat{\alpha}_i$ can take a non-zero value if and only if $y_i - f(x_i) - \varsigma - \hat{\xi} = 0$. In other words, only when the sample (x_i, y_i) does not fall within the ς -insensitive zone can the corresponding α_i and $\hat{\alpha}_i$ take non-zero values. Moreover, the constraints $f(x_i) - y_i - \varsigma - \xi_i = 0$ and $y_i - f(x_i) - \varsigma - \hat{\xi} = 0$ cannot hold simultaneously, thus at least one of α_i and $\hat{\alpha}_i$ must be zero.

Substituting the first Eq. (3.34) into the dual formula, the solution of SVR is given by (3.37) as follows:

$$f(x) = \sum_{i=1}^{m}(\hat{a}_i - a_i)x_i^T x + b \tag{3.37}$$

The samples that make $(\hat{\alpha}_i - \alpha_i) \neq 0$ in Eq. (3.37) are the support vectors of SVR, and they must lie outside the ς -margin band. Obviously, the support vectors of SVR are only a part of the training samples, that is, its solution still has sparsity.

From the KKT conditions shown in Eq. (3.36), it can be seen that for each sample (x_i, y_i), $(C - \alpha_i)\xi_i = 0$ and $\alpha_i(f(x_i) - y_i - \varsigma - \xi_i) = 0$. Thus, after obtaining α_i, if $0 < \alpha_i < C$, it must be that $\xi_i = 0$, and consequently,

$$b = y_i + \varsigma - \sum_{i=1}^{m}(\hat{a}_i - a_i)x_i^T x \tag{3.38}$$

Therefore, after solving Eq. (3.35) to obtain α_i, theoretically, any sample that satisfies $0 < \alpha_i < C$ can be selected to calculate b through Eq. (3.38). In practice, a more robust method is often adopted: multiple (or all) samples that meet the condition $0 < \alpha_i < C$ are selected to solve for b and then the average value is taken.

If the form of the feature mapping as shown in Eq. (3.7) is considered, the first equation in Eq. (3.34) will be as shown in Eq. (3.39), that is

$$\omega = \sum_{i=1}^{m} (\hat{a}_i - a_i)\phi(x_i) \tag{3.39}$$

Substituting Eq. (3.39) into Eq. (3.7), SVR can be expressed as Eq. (3.40).

$$f(x) = \sum_{i=1}^{m} (\hat{a}_i - a_i)\kappa(x, x_i) + b \tag{3.40}$$

Among them, $\kappa(x, x_i) = \phi(x_i)^T \phi(x_i)$ is the kernel function.

3.6 Support Vector Machine Algorithm

3.6.1 Block Selection Algorithm

The chunking algorithm was first proposed by Boser et al. [7]. The main idea is: remove the rows and columns corresponding to the samples with zero Lagrange multipliers in the kernel matrix to obtain the same result as the previous matrix calculation, which converts a large and complex quadratic programming problem into a series of small sub-problems. The key point of the chunking algorithm is to predict which samples in the prediction sample set have zero Lagrange multipliers, discard those with zero, and retain those with non-zero (support vectors). However, the actual support vectors are unknown, so the KKT conditions are introduced for iterative processing to ultimately obtain the global optimal solution. The chunking algorithm reduces the matrix size from the square of the number of training samples to the square of the number of samples with non-zero Lagrange multipliers, significantly reducing the storage capacity requirements during the training process and thus greatly improving the training speed. However, this only solves the problem to a certain extent. The ultimate goal of the block selection algorithm for the quadratic programming problem of the dataset is to identify all support vectors, and thus it is necessary to store the corresponding kernel function matrix. Therefore, the block selection algorithm is still essentially affected by the number of support vectors and has not fundamentally solved the problem of insufficient memory.

3.6.2 Decomposition Algorithm

The decomposition algorithm was first proposed by Osuna et al. [8]. Its main idea is also to decompose large QP problems into a series of small subproblems, but it is different from the block selection algorithm, that is, it is a variant of the active set method. The algorithm process is as follows: in each iteration, the Lagrange multiplier α_i is divided into the working set and the non-working set. The non-working set remains unchanged in this iteration; the working set is the free variable, and these free variables are optimized in this iteration. This process continues until the stopping condition is met. The key to this algorithm lies in choosing the optimal working set to improve the convergence and convergence speed of the algorithm. However, in the actual process, the selection of the working set is random, so its convergence speed is limited.

For the deficiencies in the above-mentioned algorithm, Joachims made a series of improvements to Osuna's method, proposed some strategies and techniques for selecting the working set, and implemented this algorithm, eventually developing the software $\text{SVM}^{\text{light}}$. In terms of selecting the working set, the idea is to pick out q variables from all the variables, which should meet the following conditions: not equal to zero and corresponding to the feasible steepest descent direction of the objective function. This can be solved by solving a simple linear optimization problem.

$\text{SVM}^{\text{light}}$ also introduced shrinking techniques, caching techniques, and gradient increment correction techniques, etc., which have obvious effects on large-scale problems (especially when the proportion of support vectors is small or most support vectors are on the boundary).

3.6.3 Fuzzy Support Vector Machine Algorithm

When conducting data mining, there will be a lot of noise (also known as fuzzy information) in the data set, which can lead to poor performance or even no use at all when using the SVM algorithm for prediction. In view of this situation, Lin et al. proposed the Fuzzy Support Vector Machine (FSVM) algorithm [9]. This algorithm combines fuzzy mathematics with the support vector machine algorithm and is mainly used to handle the noise data in the training samples. The core idea is to use the anomaly data detection method to detect the data in the training data set, identify the abnormal data (noise or outliers), and assign it a very small membership degree, while assigning a larger membership degree to the support vectors - this is done to separate the noise or outliers from the effective samples.

The FSVM algorithm can not only enhance classification accuracy but also address the "over-learning" issue caused by abnormal data. However, in practical applications, it also has some drawbacks. For instance, there might be a large number of abnormal data or these abnormal data may follow a certain distribution. In such

cases, if the algorithm is still used to separate these abnormal data as described above, it will lead to information loss, thereby affecting the generalization ability of the SVM algorithm. Additionally, the FSVM algorithm also has problems such as large computational load of the kernel function, high memory requirement, and long training time. Therefore, how to optimize the training of the FSVM algorithm is a key issue.

In response to the above issues, many scholars have improved the FSVM algorithm. Researchers proposed a method of using the cut set C-means clustering to cluster the training samples and using the cluster centers as new samples for training, in order to address the common problem of excessively long training time in the FSVM algorithm. They conducted numerical experiments, and the results demonstrated that this method effectively enhanced the classification speed and accuracy compared to the general FSVM algorithm. A radius control factor is introduced to address the problems of Lin et al.'s fuzzy method based on the distance between class centers, such as the inability to effectively distinguish noise or outliers and the potential reduction of the membership degree of support vectors. He proposed an improved membership function design method without increasing the time complexity, which effectively enhanced the classification accuracy of the FSVM algorithm. In reference, to address the unreasonable performance of traditional FSVM algorithms on non-spherical distributed data, a new membership function design method based on the distance from samples to hyperplanes was proposed by using intra-class hyperplanes instead of class centers. This method overcame the shortcomings of traditional methods, reduced the dependence of the membership function on the geometric shape of the sample set, and improved the generalization ability of the FSVM algorithm.

The design of the membership function directly affects the classification performance of the FSVM algorithm [9]. The membership degree of a sample decreases as the sample moves further away from the geometric center of the category. This is equivalent to constructing a mapping that can contain all samples within the category with the geometric center of the category as the center of a sphere. Under the influence of this membership function, all samples far from the class center are regarded as outliers. The farther the distance from the class center, the smaller the membership degree. However, samples that are far from the class center but close to the classification surface may also be support vectors. Assigning them a relatively low membership degree will greatly weaken the effect of support vectors on the classification surface and affect the classification performance. Therefore, some scholars have proposed a new strategy, still taking the geometric center of the category as the center of the sphere, and introducing a control factor to control the size of the radius, so that outliers are outside the hypersphere and are assigned a very small membership degree. Support vectors are within the hypersphere and are generally near the edge of the hypersphere. The membership degree is assigned according to the principle that "the farther the distance from the class center, the greater the membership degree", so that samples closer to the edge of the hypersphere within the hypersphere have a greater membership degree.

3.6.4 Sequential Minimal Optimization Algorithm

The basic idea of the decomposition algorithm is to divide the training sample set into two subsets B and N at each iteration step. Only the samples in B are iterated, while the Lagrange multipliers corresponding to the samples in the other set N remain unchanged. Then, a part of the "worst-case sample points" in N are exchanged with a part of the sample points in the working set B. The Sequential Minimal Optimization (SMO) algorithm takes the decomposition algorithm to the extreme by limiting the size of the working set to 2, that is, optimizing the smallest subset of only two sample points each time. This process is repeated multiple times until all Lagrange multipliers satisfy the KKT conditions and the objective function reaches its minimum. At this point, the size of the working set has been reduced to the minimum because the dual problem of the original problem has constraints. As long as one multiplier is changed, at least another multiplier must be adjusted simultaneously to ensure that the constraint is not violated. The execution steps of the Sequential Minimal Optimization algorithm are as follows:

(1) Given the precision ε, let $k = 0$, and take the initial value $\alpha^0 = \left(\alpha_1^0, \ldots, \alpha_l^0\right)^T = 0$

(2) Based on the current approximate solution α^k, select a subset $\{i, j\}$ consisting of two elements from the set $\{1, 2, \ldots, l\}$ as the working set Collection B;

(3) Solve the optimization problem corresponding to the working set B (Eq. (3.41)), Thus, we obtain the solution $\alpha_B^* = \left(\alpha_i^{k+1}, \alpha_j^{k+1}\right)^T$, and based on this, we further have the i-th and j-th components of the new α^k are obtained to get a new feasible approximate solution α^{k+1};

(4) If α^{k+1} satisfies the KKT conditions within the precision range of ε, that is:

$$\begin{aligned} &\sum_{i=1}^{l} y_i \alpha_i = 0, 0 \le \alpha_i \le C \\ &y_j \left| \sum_{i=1}^{l} y_i \alpha_i K\left(x_i, x_j\right) + b \right| \begin{cases} > 1, & \alpha_j = 0 \\ +1, & 0 < \alpha_j < C \\ < 1, & \alpha_j = C \end{cases} \\ &i, j = 1, \ldots, l \end{aligned} \tag{3.41}$$

Then the approximate solution is obtained as $\alpha^* = \alpha^{k+1}$ and the calculation stops; otherwise, let $k = k + 1$ and return to step (2).

(5) Obtain the optimal classification rule function.

The Sequential Minimal Optimization (SMO) algorithm has two specific issues: one is the analytical solution to the optimization problem of two variables; the other is how to select two training samples in the working set B. To analytically solve the optimization problem of two variables, it is assumed that only the corresponding two sample points, α_1 and α_2, are adjusted in each iteration. These two multipliers are defined in the two-dimensional square space of $(0, C)$. At the same time, they lie on the diagonal line determined by $\alpha_1 + s\alpha_2 = \gamma$, where $s = y_1 y_2$, for solving an optimization problem with two variables.

3.7 Example Application

To help readers gain a deeper understanding of the theoretical knowledge of SVMs, we present a classification model for kinase inhibitors based on SVMs and provide the relevant source code.

(1) Dataset

Kinase inhibitors are compounds that inhibit the activity of kinases. Protein kinases are used to regulate multiple functions of cells. Since 2001, 76 kinase inhibitors have been approved for marketing, for instance, Imatinib has been successfully applied in cancer treatment. The dataset used here is a four-class dataset containing 2013 kinase inhibitors, each of which has 5 features. For the sake of presentation, we have replaced the terms as shown in Table 3.2, and the dataset is presented as shown in Fig. 3.6.

The distribution of the dataset in the four-class classification is shown in Table 3.3. There exists a certain problem of data distribution imbalance in the label distribution. There are many solutions to the problem of data distribution imbalance, but for the sake of simplifying the practical code, these methods are not used here.

Table 3.2 Feature conditions

Feature name	Replacement name
r_desc_Average_connectivity_index_chi-2	Feature0
r_desc_Maximal_electrotopological_negative_variation	Feature1
r_desc_Second_Mohar	Feature2
r_desc_Second_Zagreb_index_by_valence_vertex_degrees	Feature3
r_qp_ACxDN^0.5/SA	Feature4

	feature0	feature1	feature2	feature3	feature4
0	0.293245	1.902150	7.867279	611.000000	0.021358
1	0.300698	6.693446	5.678195	347.777778	0.058341
2	0.274048	0.987404	4.915686	401.000000	0.009612
3	0.280079	1.899112	7.295805	650.000000	0.019622
4	0.287003	2.405486	3.201728	481.000000	0.022678
...	...	...	...	...	...
2008	0.283208	5.174158	3.261525	356.000000	0.024340
2009	0.279821	0.861883	3.539995	347.037037	0.008887
2010	0.270120	2.089367	4.400193	509.000000	0.012749
2011	0.269581	1.893247	6.029059	673.000000	0.019620
2012	0.284379	1.861145	4.676438	314.000000	0.008033

2013 rows × 5 columns

Fig. 3.6 Samples of the used dataset

Table 3.3 Distribution of labels

Label	Count
I	1399
I1/2	377
II	190

Prompt

Methods to address class imbalance can be categorized into three groups: simple approaches, data-level resampling, and algorithmic-level strategies. Simple approaches include collecting more data, transforming classification tasks into anomaly detection, adjusting class weights or decision thresholds. Data-level methods involve undersampling (e.g., random undersampling), oversampling (e.g., SMOTE), or hybrid techniques combining both. Algorithmic strategies focus on cost-sensitive learning (assigning higher penalties to minority class errors) and ensemble learning (e.g., Balanced Random Forests).

(2) Import related libraries and create SVM instances

Listing 3.1 Importing Libraries and Creating an Instance of the SVM

```
from sklearn.ensemble import AdaBoostRegressor, RandomFore-
stRegressor, RandomForestClassifier
from sklearn.datasets import load_boston
from numpy import array
from pandas import read_csv
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, Standard-
Scalerfrom sklearn.svm import SVC
estimator = SVC()
```

In the actual code shown in Listing 3.2, we directly call and instantiate various machine learning models from the sklearn library, such as random forests and decision trees, etc.

Listing 3.2 Actual Code

```
from sklearn.ensemble import AdaBoostRegressor, RandomFore-
stRegressor, RandomForestClassifier
from sklearn.datasets import load_boston
from numpy import array
from pandas import read_csv
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, Standard-
Scaler
#Read data file
dataframe = read_csv("kinase_selected.csv", delimiter = ',')
dataset = dataframe.values
X = dataset[:,1:]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print(X_scaled.shape)
#output as
#(2013, 5)
#feature_names = dataframe_cat.columns.values[2:]
feature_names = dataframe.columns.values[1:]
features = array(feature_names)
```

```
print("All features:")
print(features)
print("length of features", len(features))
#output as
#All features:
#['r_desc_Average_connectivity_index_chi-2'
# 'r_desc_Maximal_electrotopological_negative_variation'
# 'r_desc_Second_Mohar'
#'r_desc_Second_Zagreb_index_by_valence_vertex_degrees'
#'r_qp_ACxDN^.5/SA']
#length of features 5
#read label file
label = pd.read_csv('label_kinase_selected.csv', sep=',',)
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder, Standard-
Scaler
le = preprocessing.LabelEncoder()
y=label.iloc[:,-1]
print(y.shape)
#output as
#(2013,)
# lable encoder
Y=le.fit_transform(y)
print(Y)
#output as
#[3 2 1... 2 2 1]
print(label['Binding_Modes'].value_counts())
#output as
#I 1399
#I1/2 377
#II 190
#A 47
# split into input (X) and output (Y) variables
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_
scaled, Y, test_size=0.2, random_state=42, stratify=Y)
print("Feature data dimension: ", X_train.shape)
print("Num of classes: ", Y_train.shape)
#output as
#Name: Binding_Modes, dtype: int64
#Feature data dimension: (1610, 5)
#Num of classes: (1610,)
from sklearn.svm import SVC
estimator = SVC()
estimator.fit(X_train, Y_train)
print("train score: ", estimator.score(X_train, Y_train))
print("test score: ", estimator.score(X_test, Y_test))
#output as
#train score: 0.7521739130434782
#test score: 0.7344913151364765
```

Table 3.4 Score situation

	Train	Test
accuracy_score	0.75217	0.73449

(3) Result Analysis.

After training, the final scores are shown in Table 3.4. The evaluation function used here is accuracy_score, whose calculation formula is shown in Eq. (3.42). The final result is that in the 20% sample test set, 73.449% of the samples were correctly classified.

$$accuracy\_score = \frac{TP}{ALL} \tag{3.42}$$

Here, TP represents the number of correctly classified samples, and ALL represents the total number of samples.

By comparing the results of the training set and the test set, it is found that the score difference is very small, indicating that the SVM algorithm can effectively perform classification tasks without overfitting or underfitting.

References

1. Salcedo-Sanz, S., Rojo-Álvarez, J.L., Martínez-Ramón, M., Camps-Valls, G.: Support vector machines in engineering: an overview. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. **4**, 234–267 (2014)
2. Burges, C.J.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Disc. **2**, 121–167 (1998)
3. Campbell, W.M., Campbell, J.P., Reynolds, D.A., Singer, E., Torres-Carrasquillo, P.A.: Support vector machines for speaker and language recognition. Comput. Speech Lang. **20**, 210–229 (2006)
4. Lin, Y., Lee, Y., Wahba, G.: Support vector machines for classification in nonstandard situations. Mach. Learn. **46**, 191–202 (2002)
5. Ahmad, S., Molla, K.I.: Prediction of protein subcellular localization using support vector machine with the choice of proper kernel. Biotechnologia **98**, 85–96 (2017)
6. Wang, L.: Feature selection with kernel class separability. IEEE Trans. Pattern Anal. Mach. Intell. **30**, 1534–1546 (2008)
7. Gobet, F., et al.: Chunking mechanisms in human learning. Trends Cogn. Sci. **5**, 236–243 (2001)
8. Osuna, E., Freund, R., Girosit, F.: In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 130–136 (IEEE)
9. Lin, C.-T., Yeh, C.-M., Liang, S.-F., Chung, J.-F., Kumar, N.: Support-vector-based fuzzy neural network for pattern classification. IEEE Trans. Fuzzy Syst. **14**, 31–41 (2006)

Chapter 4
Decision Trees

The basic principle of decision trees is similar to the processing mechanism of human beings when making choices, and it has its application scenarios in both machine learning and deep learning. Among them, the widely used random forest is an ensemble learning algorithm constructed based on decision trees. This chapter will first take the binary classification task as an example to explain the basic principle of decision trees, then introduce several core concepts related to the partition selection of decision trees, and finally provide code examples to help readers better understand this algorithm.

4.1 Introduction to Decision Trees

Consider a binary classification task where we seek to develop a model from existing data to categorize new instances. This classification process inherently constitutes a decision-making mechanism. Binary classification is a prevalent problem type encountered in practical scenarios. For example, during triage at hospital receptions, nurses classify patients into emergency and non-emergency cases based on symptom severity. Initial assessments typically involve measuring vital signs, such as body temperature, and detecting fever. If a patient's temperature substantially exceeds normal thresholds (e.g., 41 °C hyperpyrexia), nurses may immediately escalate the case to physicians. For less critical cases, subsequent evaluations are conducted, including inquiries about medical history, physical reactions, and concurrent symptoms requiring urgent attention. Each evaluation step involves urgency-based decisions, culminating in prioritized treatment determinations.

The final classification outcome reflects the required judgment. Decision-making here operates recursively, with intermediate judgments relying exclusively on attribute-specific evaluations. Each test evaluates only its designated attribute's immediate implications. For instance, temperature assessment focuses solely on its

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_4

diagnostic relevance. If results fall within normal ranges, subsequent evaluations proceed. Crucially, current attribute evaluations remain independent of downstream decision factors.

A decision tree is a tree like structure that can be used as a metaphor when considering its structure. A decision tree consists of a root node, several internal nodes, and several leaf nodes. Each internal node represents a test on an attribute, each branch represents a test output, and each leaf node represents a category. A decision tree is a tree like structure used to represent classification and regression models. Starting from the root node, it is divided layer by layer from top to bottom based on the optimal attributes, and finally outputs leaf nodes as the classification result values. The root node contains all the sample sets, and all the samples need to be assigned to internal nodes. Each node corresponds to an attribute test, and the sample set contained in each node is divided into child nodes based on the different attributes. Leaf nodes correspond to the decision results made.

4.2 Decision Tree Partition Selection

The partition selection of decision trees involves concepts such as information entropy, information gain, information gain ratio, and Gini index, which are the contents to be introduced in this section.

4.2.1 Information Entropy

The division of attributes in a decision tree is the key to decision tree learning. Generally, as the division continues, we hope that the samples contained in the branch nodes of the decision tree belong to the same category as much as possible. To obtain the optimal division result, we need to define a division index. The process of establishing a decision tree is an information processing process, and uncertainty often accompanies the information processing process. To eliminate the factors of uncertainty, we can use entropy to measure the size of the uncertainty of the current information. "Information entropy" is a frequently used index to measure the purity of samples. The larger the entropy, the greater the uncertainty. Therefore, we choose information entropy as the selection index in the decision tree algorithm.

Suppose there is currently a set containing D samples, among which the proportion of samples of the kth category is P_k. Then, the information entropy of D is defined as

$$\text{Entropy}(D) = -\sum_{k=1}^{n} P_k \ln P_k \tag{4.1}$$

We use a dataset for determining medication types as an example. During the treatment process, it is crucial to individualize medication administration based on each patient's specific condition. There exist individual variations in a person's response to drug treatment, which can be attributed to factors such as family genetics, age, weight, and the type of illness being treated. In such scenarios, individualized medication administration is necessary. The Kaggle competition features a dataset named Drug200, from which we select a portion of the data for illustration. When considering a patient's medication needs, it is essential to comprehensively evaluate factors such as the severity of their illness, their drug absorption rate, drug tolerance, and others. Table 4.1 lists the "age," "gender," "blood pressure," "cholesterol levels," and "sodium-to-potassium ratio" of patients corresponding to their use of two different drugs, along with the types of drugs administered. This dataset comprises 17 sample instances, where {age, gender, blood pressure, cholesterol levels, sodium-to-potassium ratio} represent the attributes of the samples, and the type of medication is the ultimate decision we seek. By analyzing this dataset, we can determine which drug should be prescribed to patients based on their attributes, and use a decision tree to predict medication types for new patients. In Table 4.1, the proportion of patients using Drug A is 9/17, while the proportion using Drug B is 8/17. Additionally, the information entropy of the corresponding root node is noted.

$$Entropy(D) = -\sum_{k=1}^{2} P_k \ln P_k = -\left(\frac{9}{17}\ln\frac{9}{17} + \frac{8}{17}\ln\frac{8}{17}\right) = 0.6914$$

Assuming that the attribute 'v' has V possible values, using 'v' to partition the sample set D will result in V nodes. Each node contains D^v samples, and the entropy $Entropy(D_v)$ of each node can be calculated. Since the number of samples contained in different branch nodes varies, different weights can be assigned to nodes based on the proportion of their sample sizes.

We need to calculate the information entropy for five attributes: {age, gender, blood pressure, cholesterol, sodium-to-potassium ratio}. Taking "age" as an example, we use the average age as the criterion for partitioning. The subset with ages below 50 is denoted as D1, and the subset with ages above 50 is denoted as D2. D1 contains 7 samples, specifically {1, 3, 5, 6, 7, 8, 9}; D2 contains 10 samples, specifically {2, 4, 10, 11, 12, 13, 14, 15, 16, 17}. Within D1, the proportion of using drug A is 7/7, and the proportion of using drug B is 0/7. In D2, the proportion of using drug A is 2/10, and the proportion of using drug B is 8/10. The information entropy is calculated based on the given formula.

$$Entropy(D(\text{Age})^1) = -\left(\frac{7}{7}\ln\frac{7}{7} + \frac{0}{7}\ln\frac{0}{7}\right) = 0.0$$

$$Entropy(D(\text{Age})^2) = -\left(\frac{2}{10}\ln\frac{2}{10} + \frac{8}{10}\ln\frac{8}{10}\right) = 0.5004$$

Table 4.1 Dataset for determining the type of drug administration

Number	Age	Gender	Blood pressure	Cholesterol	Sodium-to-potassium ratio	Medicine type
1	43	M	High	High	13.972	drugA
2	72	M	High	Normal	9.445	drugA
3	37	F	Low	High	13.091	drugA
4	64	M	High	Normal	9.475	drugA
5	29	M	High	High	12.856	drugA
6	36	F	High	High	11.198	drugA
7	19	F	Normal	High	13.313	drugA
8	38	F	High	Normal	11.326	drugA
9	31	M	High	Normal	11.871	drugA
10	74	M	High	High	9.567	drugB
11	58	F	High	LOW	14.239	drugB
12	68	F	High	Normal	10.189	drugB
13	55	M	High	Normal	11.34	drugB
14	60	F	Normal	High	13.303	drugB
15	70	M	High	High	13.967	drugB
16	60	M	High	High	13.934	drugB
17	59	M	High	High	13.935	drugB

Similarly, we can calculate the information entropy of other attributes separately. The information entropy brought by taking the "gender" attribute as the division basis is

$$Entropy(D(\text{Gender})^1) = -\left(\frac{5}{10}\ln\frac{5}{10} + \frac{5}{10}\ln\frac{5}{10}\right) = 0.6931$$

$$Entropy(D(\text{Gender})^2) = -\left(\frac{4}{7}\ln\frac{4}{7} + \frac{3}{7}\ln\frac{3}{7}\right) = 0.6829$$

The information entropy based on “blood pressure” attribute is

$$Entropy(D(\text{BP})^1) = -\left(\frac{7}{14}\ln\frac{7}{14} + \frac{7}{14}\ln\frac{7}{14}\right) = 0.6931$$

$$Entropy(D(\text{BP})^2) = -\left(\frac{1}{2}\ln\frac{1}{2} + \frac{1}{2}\ln\frac{1}{2}\right) = 0.6931$$

$$Entropy(D(\text{BP})^3) = -\left(\frac{1}{1}\ln\frac{1}{1} + \frac{0}{1}\ln\frac{0}{1}\right) = 0.0$$

The information entropy based on the “cholesterol” attribute is

$$Entropy(D(\text{Chol})^1) = -\left(\frac{5}{10}\ln\frac{5}{10} + \frac{5}{10}\ln\frac{5}{10}\right) = 0.6931$$

$$Entropy(D(\text{Chol})^2) = -\left(\frac{4}{6}\ln\frac{4}{6} + \frac{2}{6}\ln\frac{2}{6}\right) = 0.6365$$

$$Entropy(D(\text{Chol})^3) = -\left(\frac{0}{1}\ln\frac{0}{1} + \frac{1}{1}\ln\frac{1}{1}\right) = 0.0$$

The information entropy based on the attribute of "sodium-to-potassium ratio" is

$$Entropy(D(\text{NaK})^1) = -\left(\frac{2}{2}\ln\frac{2}{2} + \frac{0}{2}\ln\frac{0}{2}\right) = 0.0$$

$$Entropy(D(\text{NaK})^2) = -\left(\frac{7}{15}\ln\frac{7}{15} + \frac{8}{15}\ln\frac{8}{15}\right) = 0.6909$$

Based on the calculations above, the information entropy has been determined for five attributes: “age,” “gender,” “blood pressure,” “cholesterol,” and “sodium-to-potassium ratio.” Notably, the information entropy associated with the attribute of “age” is the lowest.

4.2.2 Information Gain

When assigning attributes, we aim for the samples in each branch node to belong predominantly to the same class. Consequently, we can employ information gain for attribute partitioning in decision trees. Typically, a higher information gain signifies a greater enhancement in purity achieved through the use of that attribute. The information gain represents the change in information entropy resulting from partitioning based on attribute v. The information gain achieved by partitioning the sample set according to attribute v is

$$Gain(D, v) = Entropy(D) - \sum_{v=1}^{V} \frac{D^v}{D} Entropy(D^v) \tag{4.2}$$

where, $\frac{D^v}{D}$ represents the proportion of samples in the branch node, and $Entropy(D^v)$ denotes the information entropy of the branch node.

The information gain derived from the age attribute is as follows:

$$Gain(D, Age) = Entropy(D) - \sum_{v=1}^{2} \frac{D^v}{D} Entropy(D^v)$$

$$= 0.6914 - \left(\frac{7}{17} * 0.0 + \frac{10}{17} * 0.5004\right)$$
$$= 0.3970$$

The information gain derived from other attributes is

$$Gain(D, \text{Gender}) = 0.6914 - \left(\frac{10}{17} * 0.6931 + \frac{7}{17} * 0.6829\right) = 0.0025$$

$$Gain(D, \text{BP}) = 0.6914 - \left(\frac{14}{17} * 0.6931 + \frac{2}{17} * 0.6931 + \frac{1}{17} * 0.0\right) = 0.039$$

$$Gain(D, \text{Chol}) = 0.6914 - \left(\frac{10}{17} * 0.6931 + \frac{6}{17} * 0.6365 + \frac{1}{17} * 0.0\right) = 0.059$$

$$Gain(D, \text{NaK}) = 0.6914 - \left(\frac{12}{17} * 0.0 + \frac{15}{17} * 0.6909\right) = 0.0818$$

It can be observed that the attribute "age" yields the highest information gain, making it the chosen partition attribute for the root node. The decision tree structure diagram for the resulting branch nodes is illustrated in Fig. 4.1.

Then, the decision tree further divides each node in turn. In this example, continue to calculate the information gain when "gender", "blood pressure", "cholesterol" and "sodium-to-potassium ratio" are used as the division attributes. It is found that the information gain obtained by "sodium-to-potassium ratio" is the largest, so "sodium-to-potassium ratio" is used as the division attribute of the branch node. In this way, partition attributes are selected for each branch node in turn. Divide the samples in Table 4.1 into different nodes to obtain the decision tree. The final decision tree constructed in this example is shown in Fig. 4.2. After that, given the "age" and "sodium-to-potassium ratio" of a patient, the corresponding type of medication can be determined.

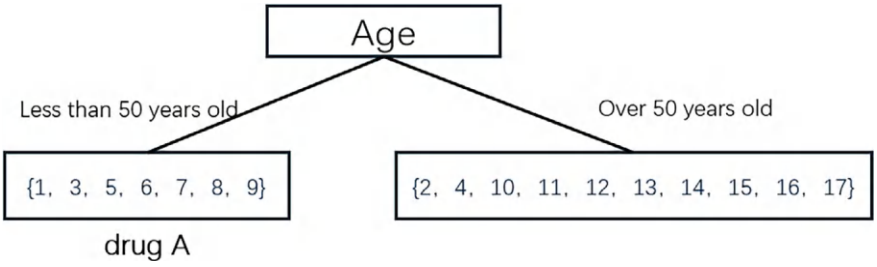


Fig. 4.1 Partitioning of the root node based on the "age" attribute

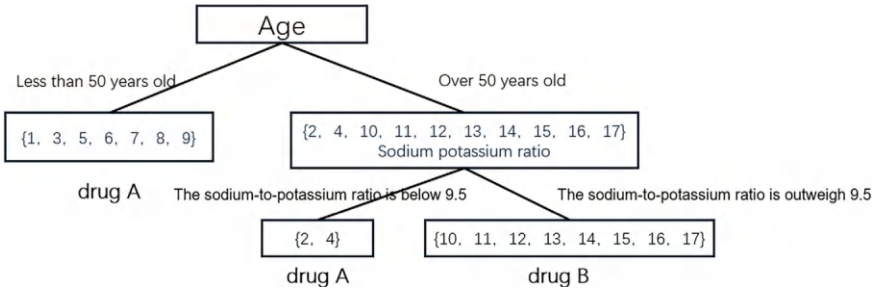


Fig. 4.2 The branch nodes are divided based on the attribute of "sodium-to-potassium ratio", and the decision tree based on information gain is obtained

4.2.3 Information Gain Ratio

Information gain can be used as attribute partition in general. However, it should be noted that the information gain has a preference for attributes with a large number of values. For example, for the samples in Table 4.1, each sample is used as a branch node, and the root node is divided into 17 branches. At this time, the purity of each branch node reaches the maximum. However, such a decision tree has no application value. When a new patient is added to the sample, it is impossible to predict based on the decision tree.

In order to avoid this preference for more attributes, the method of information gain rate can be used to select the optimal attributes. Information gain rate is defined as

$$Gain\_ratio(D, v) = \frac{Gain(D, v)}{IV(v)} \tag{4.3}$$

Among them,

$$IV(v) = -\sum_{v=1}^{V} \frac{D^v}{D} \ln \frac{D^v}{D}$$

IV (v) is called the intrinsic value of attribute v. For example, the intrinsic value for age is

$$\begin{aligned} IV(v) &= -\sum_{v=1}^{V} \frac{D^v}{D} \ln \frac{D^v}{D} \\ &= -\left(\frac{7}{17} \ln \frac{7}{17} + \frac{10}{17} \ln \frac{10}{17}\right) \\ &= 0.6775 \end{aligned}$$

Then the gain rate divided by age as an attribute is

$$\begin{aligned} Gain\_ratio(D, v) &= \frac{Gain(D, Age)}{IV(Age)} \\ &= \frac{0.3970}{0.6775} \\ &= 0.586 \end{aligned}$$

The well-known C4.5 algorithm applies the information gain ratio to select the optimal splitting attribute.

4.2.4 Gini Index

In decision trees, in addition to using information entropy or information gain rate, there is also a Gini index. The Gini index is the probability of two randomly selected samples from dataset D having inconsistent class labels. The definition of Gini index is

$$Gini(D) = 1 - \sum_{i=0}^{m} p_m^2 \tag{4.4}$$

Taking the binary classification task as an example, when we use the model to predict the data, the prediction result must be the dataset of a and B, and the prediction accuracy of the model is less than 100%. Therefore, the dataset marked as a in the prediction will include the samples actually marked as B. At this time, we randomly extract two samples from the data set with the prediction mark of B, in which the proportion of the actual mark of a is p, and the proportion of the actual mark of B is (1 − p), and calculate the probability that the two samples are marked differently as P (1 − p), but at the same time, we also have the data set with the prediction mark of B, extract two samples, and the probability of marking differently is also p (1 − p). At this time, we can get a result of P(1 − p) + (1 − p) P, which represents the prediction effect of our model. Taking the data in Table 4.1 as an example, in sample D, 9 patients took drugA and 8 patients took drugB, then the Gini index of the current sample is

$$\begin{aligned} Gini(D) &= 1 - \frac{9}{17} * \frac{9}{17} - \frac{8}{17} * \frac{8}{17} \\ &= \frac{9}{17} * \left(1 - \frac{9}{17}\right) + \frac{8}{17}\left(1 - \frac{8}{17}\right) \\ &= 0.498 \end{aligned}$$

After calculating the Gini index of the sample, an important application is to calculate the importance of features. In the construction of the decision tree, we will calculate an information entropy for each feature, and the feature importance is the normalized value of the indicator reduction. In scikit-learn, the default feature importance is "Gini importance". The importance of Gini is defined as

$$Gini\_\ \text{importance}(D) = \frac{D^V}{D} * \left(Gini(D) - \sum_{m=1}^{N} \frac{D^{V(m)}}{D^V} Gini\left(D^{V(m)}\right) \right) \tag{4.5}$$

Taking the data in Table 4.1 as an example, combined with Fig. 4.2, we divide the 17 samples into two categories by age. Among them, the group younger than 50 years old used drugA, and the Gini index was 0. The other group uses "sodium-to-potassium ratio" as the division basis, and calculates the Gini index based on "sodium-to-potassium ratio" as

$$\text{Gini(D, NaK)} = 1 - \frac{2}{10} * \frac{2}{10} - \frac{8}{10} * \frac{8}{10} = 0.32$$

Then, we can calculate the absolute value of Gini importance of "age" and "sodium-to-potassium ratio" respectively

$$\text{Gini\_ import(Age)} = \frac{17}{17} * \left(0.498 - \frac{7}{17} * 0 - \frac{7}{17} * 0.32 \right) = 0.310$$

$$\text{Gini\_ import(NaK)} = \frac{10}{17} * \left(0.32 - \frac{2}{10} * 0 - \frac{8}{10} * 0 \right) = 0.188$$

The calculated absolute values of Gini importance of "age" and "sodium-to-potassium ratio" are 0.310 and 0.188 respectively. After normalization, the Gini importance is 0.622 and 0.378 respectively. In this way, we know that an important medication principle when using drugA and drugB is strongly related to "age" and "sodium-to-potassium ratio", but not to the patient's "gender", "blood pressure" and "cholesterol".

The importance of features can provide us with some enlightenment from multiple data features, especially in complex biological systems, it is often necessary to extract the most important determinants from massive data. In the classification problem, the attribute with higher feature importance can be used to explain why this attribute determines different classifications. Of course, we should also note that if the importance of a feature is small, it does not mean that the feature does not provide any information. This only means that the feature is not selected by the decision tree, possibly because another feature contains the same information.

4.3 Code example of Decision Tree

4.3.1 Decision tree Calculation Information Entropy

This chapter introduces the algorithm of the decision tree. The general process of building the decision tree is as follows:

(1) Data collection: generally, it is necessary to collect data containing attributes and target values. Data sources can be literature, existing databases, and data accumulated in scientific research.
(2) Prepare data: data is generally converted to numerical type and standardized as far as possible. Remove outliers.
(3) Analyze Data: analyze which of the data are attributes, which are targets, and whether there is correlation between attributes and targets.
(4) Training algorithm: use the data of the existing training set to train the algorithm.
(5 Test algorithm: verify the accuracy of the algorithm on the test set.
(6) Use algorithm: Apply the algorithm to the new dataset.

The algorithm of the decision tree has been included in the scikit-learn software package, and the construction of the decision tree is realized by coding in Python. The data set used is 17 samples in Table 4.1.

The code is shown in Listing.4.1.

Listing 4.1 Prepare Data

```
#!/usr/bin/python
#encoding:utf-8
# Divide the original data into training data and test data
import numpy as np
from sklearn import tree,metrics
import pydotplus
def gender(s):
    it = {'M':0, 'F':1}
    return it[s.decode("utf-8")]
def BP(s):
    it = {'HIGH':0, 'NORMAL':1, 'LOW':2}
    return it[s.decode("utf-8")]
def Cholesterol(s):
    it = {'HIGH':0, 'NORMAL':1, 'LOW':2}
    return it[s.decode("utf-8")]
def drug_type(s):
    it = {'drugA':0, 'drugB':1}
    return it[s.decode("utf-8")]

drug_feature_E = 'Age', 'Gender', 'BP', 'Cholesterol', 'Na_
to_K'
```

```
drug _class = 'drugA', 'drugB'
# 1、Read in data and convert the data in the original data into
digital form
data = np.loadtxt("drug_decision.csv", delimiter=",",
dtype=str, converters={1:gender, 2:BP, 3:Cholesterol,
5:drug_type})
x, y = np.split(data,(5,),axis=1)
print(x,y)
```

Listing 4.2 Partition Dataset

```
from sklearn.model_selection import train_test_split
# 2、Split training data and test data for cross validation
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3)
```

The whole data set is divided into training set and test set. The purpose is to test the accuracy of the algorithm. The data set can also not be divided, and the whole data can be used to build a decision tree. The calculation result is shown in Fig. 4.3.

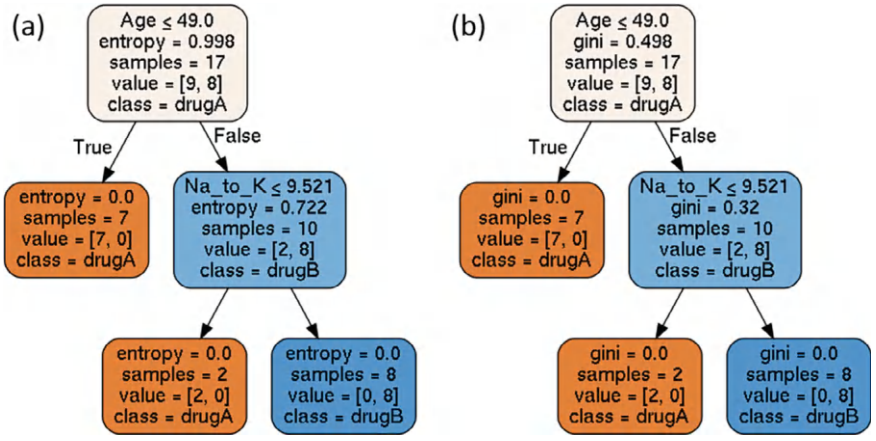


Fig. 4.3 Drug administration type decision tree based on code list. This subfigure a uses information entropy as the division standard, and subfigure b uses Gini coefficient as the division standard

Listing 4.3 Training Algorithm

```
# 3、The decision tree is trained using information entropy as
the division standard
clf = tree.DecisionTreeClassifier(criterion='entropy')
print(clf)
clf.fit(x_train, y_train)
```

Listing 4.4 Test Algorithm

```
# 4、Use training data to predict and verify the results
y_pred = clf.predict(x_train)
y_train = y_train.reshape(-1)
print(y_pred)
print(y_train)
print(np.mean(y_pred == y_train))
```

Listing 4.5 Using Algorithms for Prediction

```
# 5、Predict test data
y_pred = clf.predict(x_test)
y_test = y_test.reshape(-1)
print("accuracy:")
print(metrics.accuracy_score(y_test, y_pred))
print(y_test)
print(np.mean(y_pred == y_test))
```

The calculated feature importance is as follows

```
feature_importances:
('Age', 0.643312969322477)
('Gender', 0.0)
('BP', 0.0)
('Cholesterol', 0.0)
('Na_to_K', 0.35668703067752305)
```

From the importance of characteristics, we can see that "age" and "sodium-to-potassium ratio" have a greater impact on the classification results. The decision tree structure calculated according to the code is shown in Fig. 4.3. This calculation result is also verified in Fig. 4.3. The types of drugs can be completely separated by “age” and “sodium-to-potassium ratio”.

4.3.2 Classification of Inhibitors by Decision Tree

The algorithm of decision tree is a typical machine learning algorithm, which has a wide range of application scenarios. For example, the algorithm can be applied to the classification of kinase inhibitors. We classify the decision tree with the data set of kinase inhibitors. Kinase inhibitors can be divided into four categories, I, I1/2, II and allosteric agent A, according to their binding structures and positions with kinases. Here, the calculated molecular descriptors are used as the characteristics of inhibitors, and the types of kinases are used as classification labels. The decision tree is trained and the model is verified on the test set.

Listing 4.6 Classification of inhibitors using decision trees

```
# from pandas import read_csv
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, Standard-
Scaler
from sklearn import tree,metrics
dataframe = read_csv("kinase_selected.csv", delimiter = ',')
dataset = dataframe.values
X = dataset[:,1:]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print(X_scaled.shape)
#feature_names = dataframe_cat.columns.values[2:]
feature_names = dataframe.columns.values[1:]
features = np.array(feature_names)
print("All features:")
print(features)
print("length of features", len(features))
label = pd.read_csv('label_kinase_selected.csv', sep=',',)
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder, Standard-
Scaler
le = preprocessing.LabelEncoder()
y=label.iloc[:,- 1]
print(y.shape)
```

```
# lable encoder
Y=le.fit_transform(y)
print(label['Binding_Modes'].value_counts())
# split into input (X) and output (Y) variables
X_train, X_test, Y_train, Y_test = train_test_split(X_
scaled, Y, test_size=0.2, random_state=42, stratify=Y)
print("Feature data dimension: ", X_train.shape)
print("Num of classes: ", Y_train.shape)
estimator = tree.DecisionTreeClassifier()
estimator.fit(X_train, Y_train)
print("train score: ", estimator.score(X_train, Y_train))
print("test score: ", estimator.score(X_test, Y_test))
```

The accuracy rate of classification results calculated according to this code is

```
train score: 1.0
test score: 0.6377
```

It can be seen that the decision tree can also be applied to the four classification task of kinase inhibitors. In this task, the accuracy of the training set is 1.0, while the accuracy of the test set is 0.637. It is obvious that our training has the problem of over fitting. In this example, the default value of the decision tree is used. By optimizing parameters or using the integrated learning method, the calculation accuracy can be further improved and the generalization ability of the model can be improved.

4.3.3 Hyperparameter Optimization of the Decision Tree

In machine learning and deep learning models, in order to achieve better prediction performance, it is necessary to optimize the hyperparameters of the model. There is no general calculation method in the parameter optimization problem, and an optimization algorithm needs to be designed according to the specific problem. Wolpert and Macready have put forward the "No Free Lunch Theorem", which proves that there is no one algorithm that is effective for all problems in iterative optimization algorithms. If one algorithm is effective for some problems, it must be inferior to the random search algorithm in other problems. Therefore, the optimization of the model should be based on specific problems.

There are many methods for hyperparameter optimization, such as GridSearchCV, RandomizedSearchCV, Bayesian optimization, etc. Here, we use code Listing 4.7 to show the grid search cross validation method to optimize the parameters of the decision tree.

Listing 4.7 Using grid Search to Cross-Validate and Tune Decision-Tree Hyperparameters

```
    #grid search
from sklearn.model_selection import GridSearchCV

params = {
    'max_depth': range(1, 10),
    'min_samples_leaf': range(1, 20)
}

grid_search = GridSearchCV(
    estimator,
    param_grid = params,
    scoring = 'accuracy',
    cv=5
)

grid_search.fit(X_train, Y_train)
print(grid_search.best_params_)

estimator_opt= tree.DecisionTreeClassifier(max_depth=grid_
search.best_params_['max_depth'],   min_samples_leaf=grid_
search.best_params_['min_samples_leaf'])
estimator_opt.fit(X_train, Y_train)
print("train  score:  ",  estimator_opt.score(X_train,  Y_
train))
print("test score: ", estimator_opt.score(X_test, Y_test))
```

In the case of five-fold cross validation, we optimized the two parameters max_depth and min_samples_leaf from this code. max_depth controls the depth of the decision tree, and min_samples_leaf specifies the minimum number of samples contained by each leaf node. The hyperparameter calculated through grid search cross validation is

```
{'max_depth': 7, 'min_samples_leaf': 12}
```

Using this hyperparameter, we re trained the model, and the calculated result is

```
train score: 0.7652173913043478
test score: 0.6799007444168734
```

It can be seen that our scoring on the training set may be lower than before the optimization of hyperparameters, but we have better results on the test set. The purpose of optimizing parameters is to obtain better performance. However, we

cannot excessively pursue the performance of the model in the training process. This is because we also need to apply the trained model to other problems. If the model trained has the best performance on a certain problem, so that the model is only good at the current problem, it is bound to affect the generalization ability of the model. An example in life is that we can't expect divers to win the championship of table tennis. Therefore, there is a balance contradiction between the prediction ability and generalization ability of the model. This contradiction is similar to the "Occam Razor Principle", whose core idea is that simple models have better generalization ability. Therefore, when we optimize the model, we should consider the generalization ability of the model.

Note

In order to facilitate the explanation of the concepts in this chapter, appropriate modifications have been made to the dataset. The values in the dataset are only used as examples and may differ significantly from real medical data. Therefore, this dataset is only used for educational purposes and cannot be used for real medical diagnosis

Chapter 5
Random Forest

This chapter introduces random forests as a bagging-based ensemble method and summarizes their key ideas, implementation, and typical applications.

5.1 Introduction to Ensemble Learning

In supervised learning algorithms of machine learning, our goal is to learn a stable model that performs well in all aspects. However, the actual situation is often less than ideal, and sometimes we can only obtain multiple biased models (weak supervised models, i.e., models that perform well in certain aspects). Ensemble learning combines these multiple weak supervised models to obtain a better and more comprehensive strong supervised model. The underlying idea of ensemble learning is that even if one weak classifier makes a wrong prediction, other weak classifiers can correct the error. A single learner in an ensemble is typically called a base learner, while the combined ensemble is a strong learner.

Ensemble learning itself is not a standalone machine learning algorithm but completes learning tasks by constructing and combining multiple machine learners. Ensemble learning can be used for classification problem integration, regression problem integration, feature selection integration, outlier detection integration, etc., and can be seen in almost all fields of machine learning.

Ensemble learning is a machine learning method that uses a series of learners for learning and integrates the results of each learner according to certain rules to achieve a better learning effect than a single learner. Its core idea is "Two heads are better than one." and the main process is shown in Fig. 5.1.

There are two options for obtaining several weak learners: they can be of the same type (homogeneous) or of different types (heterogeneous). All weak learners are of the same type or homogeneous. For example, base learners can all be decision trees or neural networks, which are then integrated using frameworks such as Bagging

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_5

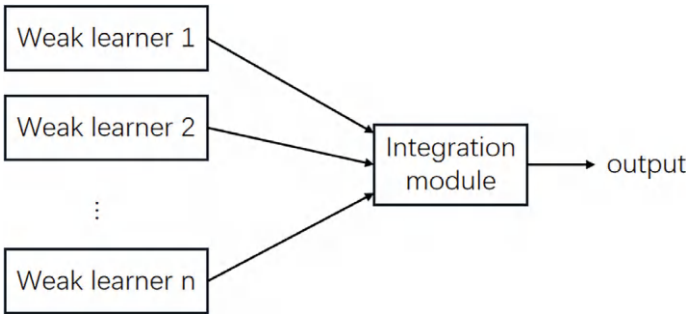


Fig. 5.1 Schematic diagram of the ensemble learning process

or Boosting. Stacking is an ensemble method that uses a meta-learner to combine the predictions of multiple primary learners, which are often (but not necessarily) heterogeneous.

Homogeneous individual learners can be divided into two categories based on the dependency relationship between them:

(1) There is a strong dependency relationship between weak learners, and a series of weak learners basically need to be generated sequentially. The representative algorithm is the Boosting series of algorithms.
(2) There is no strong dependency relationship between weak learners, and a series of weak learners can be generated in parallel. The representative algorithm is the Bagging series of algorithms.

5.1.1 Bagging

The main idea of Bagging is shown in Fig. 5.2. T data sets are sampled with replacement from the original dataset. Based on these T data sets, one base classifier is trained for each, and the base classifiers are combined to make predictions. For classification tasks, Bagging uses a simple voting method for prediction, while for regression tasks, it employs a simple averaging method. In classification prediction, if two classes receive the same number of votes, one class is randomly selected.

As can be seen from Fig. 5.2, Bagging is highly suitable for parallel processing, which is particularly advantageous for tasks involving large volumes of data. When sampling T datasets from the original dataset, it is important to generate T distinct subsets. This is because base classifiers trained on such subsets will exhibit significant differences and a certain degree of diversity, which helps improve the final performance of the ensemble algorithm. However, the base classifiers should not be too poor in performance. If the sampled subsets from the original dataset are completely dissimilar, the resulting base classifiers will perform poorly. Therefore, it is common practice to use sampling with replacement for the original dataset.

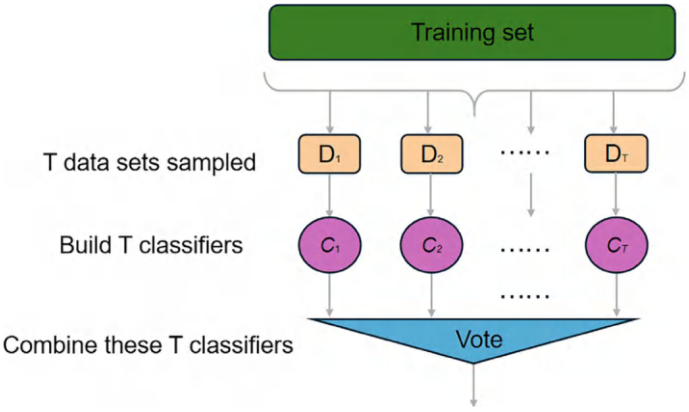


Fig. 5.2 Schematic diagram of bagging

Suppose the initial dataset contains m samples, and e ach time m samples (equal to the size of the original dataset) are sampled with replacement. After sampling, some samples in the initial dataset may appear multiple times in the sampled set, while others may not appear at all.

Prompt

From the perspective of bias-variance decomposition, Bagging primarily focuses on reducing variance.

5.1.2 Boosting

While the generation of base learners in Boosting is logically sequential, modern implementations like XGBoost leverage parallel computing for feature splitting to improve efficiency. The working mechanism of Boosting is as follows: First, a base learner is trained from the initial training set. Then, based on the performance of this base learner, the distribution of training samples is adjusted (e.g., increasing the weights of misclassified samples and decreasing the weights of correctly classified samples), so that the samples misclassified by previous base learners receive more attention in subsequent training. Next, the next base learner is trained on the adjusted sample distribution. This process is repeated until the number of base learners reaches a pre-determined value T. Finally, these T base learners are combined with weights (e.g., base learners with lower error rates are assigned higher weights, and those with higher error rates are assigned lower weights, such that during decision-making, base learners with lower error rates have a greater influence). Typical representatives of

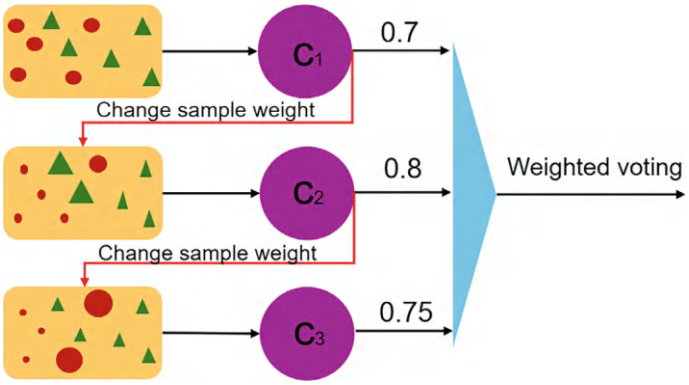


Fig. 5.3 Schematic diagram of boosting

Boosting algorithms include AdaBoost and XGBoost. The Boosting algorithm can be concisely and vividly described in Fig. 5.3.

Prompt
From the perspective of bias variance decomposition, Boosting mainly focuses on reducing bias.

5.2 Integrated Methods

The integration methods are mainly described from two aspects: the combination strategies of ensemble learning and the diversity methods of base learners.

5.2.1 The Combination Strategy of Integrated Learning

Based on the above content, we have gained a general understanding of ensemble learning algorithms. In simple terms, ensemble learning involves training a collection of base learners and then synthesizing the results of these base learners through a specific strategy to obtain the final output of the ensemble learner. Below, we will introduce the commonly used combination strategies.

There are generally three combination strategies in ensemble learning: averaging, voting, and learning. Each of these three strategies has its own advantages and disadvantages, making them suitable for different scenarios in ensemble learning.

(1) Averaging

For numerical outputs, a common combination strategy is simple averaging, as shown in Eq. 5.1.

$$O(x) = \frac{1}{T} \sum_{i=1}^{T} o_i(x) \tag{5.1}$$

In some special cases, weighted averaging is used to assign weights to multiple base learners, typically when there is a significant gap between the base learners, as shown in Eq. 5.2.

$$O(x) = \sum_{i=1}^{T} w_i o_i(x) \tag{5.2}$$

represents the weight of each base learner, and it satisfies Eq. 5.3.

$$\mathrm{w}_i \geq 0, \sum_{i=1}^{T} w_i = 1 \tag{5.3}$$

Prompt

Averaging is generally used for regression problems. Weighted averaging does not necessarily outperform simple averaging; its effectiveness depends on specific scenarios.

(2) Voting

For classification problems, the most commonly used combination strategy is voting, which can be divided into three types: absolute voting, relative voting, and weighted voting.

Absolute voting stipulates that if more than half of the base learners output the same class label, this label becomes the integrated result. However, this method often fails to meet the "more than half" prerequisite in multi-class classification problems, making it generally suitable for binary classification scenarios.

Relative voting determines the integrated result as the class label with the highest number of votes among the base learners' outputs. This approach addresses some limitations of absolute voting and is typically applied to multi-class classification tasks.

Weighted voting combines the outputs of base learners with their respective weights to determine the final result. Weighted voting is a widely used strategy,

especially in algorithms like AdaBoost or when base learners output class probabilities (Soft Voting). It assigns higher influence to learners with better performance or higher confidence.

Prompt

In the event of a tie in voting results, one class is randomly selected as the final outcome.

(3) Learning

When dealing with large datasets, a more powerful combination strategy is the learning method, which involves integrating with another learner. A typical example is Stacking, where base learners are referred to as primary learners, and the learner used for combination is called a secondary learner or meta-learner.

The core idea of Stacking is as follows: First, divide the initial training set into a training subset and a validation subset. Train primary learners using the training subset and validate them on the validation subset. Use the primary learners' prediction results on the validation subset as the training data for the meta-learner, and their predictions on the test subset as the test data for the meta-learner. The labels of the initial dataset serve as the labels for the meta-learner, as illustrated in Fig. 5.4.

Prompt

In this book, Stacking is presented as a combination strategy for ensemble learning.

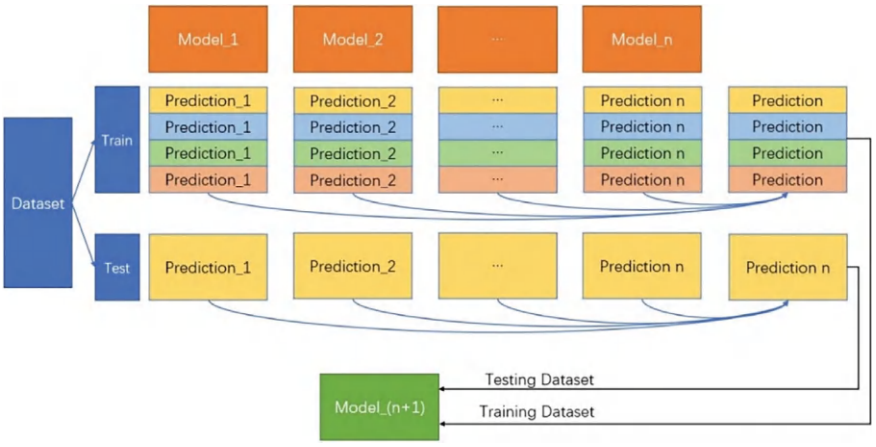


Fig. 5.4 Schematic diagram of the learning method

5.2.2 Methods for Base Learner Diversity

In ensemble learning, to enhance the diversity of base learners, we increase the diversity of models or data to improve the model's generalization ability. Methods for enhancing base learner diversity include data sample perturbation, input attribute perturbation, output representation perturbation, and algorithm parameter perturbation, among others.

(1) Data Sample Perturbation

Data sample perturbation primarily refers to the selective sampling of the dataset. For example, the bootstrap sampling (random sampling with replacement) used in Bagging is highly effective for learning algorithms sensitive to data sample variations, such as decision trees and neural networks. However, it is ineffective for algorithms less sensitive to sample perturbations, such as support vector machines (SVM), naive Bayes, and k-nearest neighbors (KNN). When using these algorithms as base learners in an ensemble, mechanisms like input attribute perturbation are often required.

(2) Input Attribute Perturbation

Input attribute perturbation involves the selective sampling of features from the dataset. By generating different feature subsets from the sample's feature space, the trained base learners will naturally differ. Training base learners on datasets with abundant redundant attributes and feature subsets not only enhances individual diversity but also significantly reduces computational costs due to the reduced number of attributes. Additionally, because of the high redundancy, base learners trained after removing some redundant attributes still maintain good performance. This method is less suitable for datasets with few attributes or low redundancy.

(3) Output Representation Perturbation

The core idea of output representation perturbation is to manipulate the output representation to enhance diversity. For instance, training sample labels can be slightly altered—such as the "flipping method," which randomly changes the labels of some training samples. Alternatively, output representations can be transformed, such as the "output Smearing method", which converts classification outputs into regression outputs to build base learners (though this approach is rare).

(4) Algorithm Parameter Perturbation

Algorithm parameter perturbation refers to constructing multiple similar base learners by adjusting the parameters of the same base learning algorithm. This method is widely used in deep learning, where neural networks have numerous adjustable parameters. Different parameter settings often yield significantly different base learners.

5.3 Random Forest

Random Forest is a variant of Bagging. It constructs a Bagging ensemble with decision trees as base learners and incorporates input attribute perturbation. Specifically, Random Forest first randomly selects attributes or samples from the training set at a certain ratio to form new training subsets. Each subset is then used to train a decision tree for sample partitioning. Finally, the results from multiple decision trees are integrated through specific ensemble methods to obtain the final prediction for samples, as shown in Fig. 5.5. This approach significantly enhances the diversity among base learners, thereby improving the generalization performance of the ensemble.

Random Forest (RF), a representative algorithm of Bagging in ensemble learning, offers advantages including simple code implementation, high computational speed, parallelization capability, low computational cost, excellent ensemble performance, and strong resistance to overfitting. By using Classification and Regression Trees (CART) as base learners, RF can be applied to both classification and regression tasks. For classification tasks, CART classification trees serve as base learners, and the final prediction is generally determined by relative voting (majority voting) for the voting results [1]. For regression tasks, the final prediction of RF is typically the mean of the prediction results from all CART regression trees.

In recent years, RF has been widely used to solve problems in the medical and pharmaceutical fields [2]. For example, Fan et al. established a RF model to predict the efficacy of blackcurrant oil soft capsules in treating hyperlipidemia. The results showed that the area under the ROC curve (AUC) in cross-validation and external validation was 0.827 and 0.857, respectively. Based on a traditional Chinese medicine database, Chen et al. adopted a network pharmacology-based approach to study

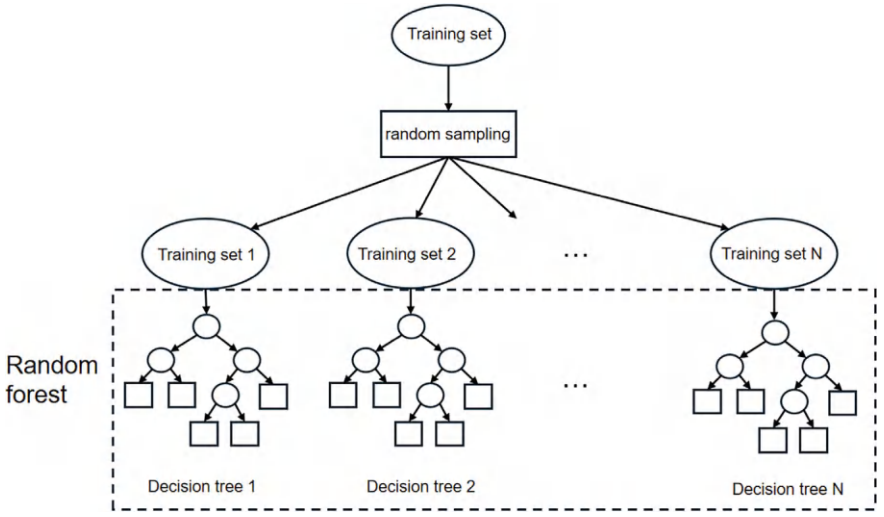


Fig. 5.5 Schematic diagram of random forest

Chinese medicine candidate drugs that can well dock with multiple targets, using deep learning and RF algorithms [3]. The RF algorithm achieved an accuracy of 0.869 on the training set and 0.890 on the test set. Ru et al. constructed a RF algorithm to identify electron transport proteins during cell respiration [4]. In tenfold cross-validation, the sensitivity, specificity, and accuracy exceeded 85%, 80%, and 82%, respectively. In the test set, the F-measure, AUC value, and accuracy exceeded 74%, 95%, and 86%, respectively. RF has contributed to solving many medical and pharmaceutical problems through its powerful modeling capabilities.

5.4 Random Forest Code Example

This section demonstrates how to build a Random Forest (RF) classifier using scikit-learn and introduces its main parameters and commonly used methods.

5.4.1 Introduction to Creating RF Model

Code Listing 5.1 Creating a Random Forest Model

```
# import library
from sklearn.ensemble import RandomForestClassifier
# Instantiate Random Forest Model
RF = RandomForestClassifier(n_estimators=100,
     criterion="squared_error",
     max_depth=None,
     min_samples_split=2,
     min_samples_leaf=1,
     min_weight_fraction_leaf=0.0,
     max_features="auto",
     max_leaf_nodes=None,
     bootstrap=True,
     oob_score=False,
     n_jobs=None,
     verbose=0,
     max_samples=None,
     )
```

The following is a translated and polished version of the text:

(1) Parameters

- **n_estimators**: The number of base learners (decision trees). If n_estimators is set too small, the model is prone to underfitting; if set too large, it may lead to overfitting. This parameter should be adjusted according to actual needs. **Default value**: 100.

- **criterion**: The evaluation criterion used for feature splitting in CART decision trees. Options include{"gini", "entropy", "log_loss"}. The default is " gini ", which is generally suitable for most scenarios.
- **max_depth**: The maximum depth of the tree. **Default**: None. When set to None, nodes will expand until all leaves are pure or contain fewer samples than min_samples_split.
- **min_samples_split**: The minimum number of samples required to split an internal node. If the number of samples in a node is less than this value, the node will not continue to branch. **Default**: 2.
- **min_samples_leaf**: The minimum number of samples required for a leaf node. **Default**: 1. If the number of samples in a leaf node is less than this value, it will be pruned together with its sibling node. For large datasets, increasing this value can help smooth the model, especially in regression tasks.
- **min_weight_fraction_leaf**: The minimum weighted fraction of the total sample weight required at a leaf node. When the leaf weight is below this value, branch weights are assumed to be equal.
- **max_features**: The maximum number of features considered for splitting at each node. Options include auto, sqrt, log2, or an integer. **Default**: auto. If set to auto or None, all features are considered; sqrt means considering up to $\mathrm{sq}rt(n\_features)$ features; $\log_2$ means considering up to $\log_2(n\_features)$.
- **max_leaf_nodes**: The maximum number of leaf nodes. **Default**: None(no limit). This can be restricted to prevent overfitting when dealing with high-dimensional data.
- **bootstrap**: Whether to use partial data (bootstrap sampling) to build decision trees. **Default**: True. If set to False, the full dataset is used.
- **oob_score**: Whether to use out-of-bag (OOB) samples to evaluate the model. Since bootstrap sampling may omit some samples, these OOB samples can serve as a validation set. **Default**: False(not used).
- **n_jobs**: The number of CPU cores to use. **Default**: 1. Setting it to -1uses all available CPU cores.
- **verbose**: Controls the verbosity level during fitting and prediction. **Default**: 0 (silent).

(2) Attributes

- **feature_importances_**: Indicates the importance of each feature, where higher values signify more critical features.
- **estimators_**: A list storing the trained base learners (decision trees).
- **n_features_**: The number of features used after model training.
- **n_outputs_**: The number of output targets after model training.
- **oob_score_**: The score obtained by validating with OOB samples from the training set (available only if oob_score = True).
- **oob_prediction_**: Predictions on OOB samples from the training set (available only if oob_score = True).

(3) Methods

- **apply(X)**: Obtains the leaf node positions of each sample in X across all base learners in the ensemble.
- **fit(X_train, y_train)**: Trains the model on the training set.
- **score(X_test, y_test)**: Returns the prediction accuracy on the test set (for classification tasks).
- **predict(X)**: Uses the trained model to predict labels for input data X.
- **predict_proba(X)**: Returns an array of class probability estimates for input data X (for classification tasks).
- **predict_log_proba(X)**: Returns an array of log probabilities for class membership of input data X (for classification tasks).

5.4.2 The Practical Application of Random Forest in the Classification of Kinase Inhibitors

In order to quickly understand the specific situation of RF, the following introduces the RF classification task for kinase inhibitors. In the code, two models, decision tree and RF, are used to predict and compare the same dataset.

(1) Dataset

Here we use a dataset containing the quality of kinase inhibitors from 2013 samples. The data situation is shown in Fig. 5.6.

The dataset is in CSV format, with 2014 rows and 6 columns. (Excluding the row number and column header, it contains 2013 samples and 5 features.) For convenient display, the features are replaced with nouns as shown in Table 5.1.

(2) Decision tree and random forest code and output results

Code Listing 5.2 Practical Code

```
# Import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as matplot
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import fetch_openml
import joblib
from sklearn.model_selection import train_test_split
from sklearn import tree
import sklearn
# read data
```

	feature0	feature1	feature2	feature3	feature4
0	0.293245	1.902150	7.867279	611.000000	0.021358
1	0.300698	6.693446	5.678195	347.777778	0.058341
2	0.274048	0.987404	4.915686	401.000000	0.009612
3	0.280079	1.899112	7.295805	650.000000	0.019622
4	0.287003	2.405486	3.201728	481.000000	0.022678
...	...	...	...	...	...
2008	0.283208	5.174158	3.261525	356.000000	0.024340
2009	0.279821	0.861883	3.539995	347.037037	0.008887
2010	0.270120	2.089367	4.400193	509.000000	0.012749
2011	0.269581	1.893247	6.029059	673.000000	0.019620
2012	0.284379	1.861145	4.676438	314.000000	0.008033

2013 rows × 5 columns

Fig. 5.6 Introduction to kinase inhibitor dataset

Table 5.1 Characteristic situation

Feature name	Alternative name
r_desc_Average_connectivity_index_chi-2	Feature0
r_desc_Maximal_electrotopological_negative_variation	Feature1
r_desc_Second_Mohar	Feature2
r_desc_Second_Zagreb_index_by_valence_vertex_degrees	Feature3
r_qp_ACxDN^0.5/SA	Feature4

```
df = pd.read_csv("kinase_selected.csv", delimiter = ',',header=0,index_col=0)
label = pd.read_csv('label_kinase_selected.csv', sep=',', )
# Check for missing values
print(df.isnull().any())
#Output results
#Unnamed: 0 False
#r_desc_Average_connectivity_index_chi-2 False
#r_desc_Maximal_electrotopological_negative_variation
False
#r_desc_Second_Mohar False
#r_desc_Second_Zagreb_index_by_valence_vertex_degrees
False
#r_qp_ACxDN^.5/SA False
```

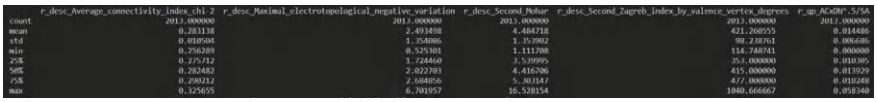
```
feature_list = ['feature'+str(i) for i in range(5)]
feature_list = pd.array(feature_list)
#Show statistics
print(df.describe())
# The results are presented in Fig. 5.7
# Correlation matrix
corr = df.corr()
sns.heatmap(corr,xticklabels=feature_list,yticklabels=
feature_list)
# The results are presented in Fig. 5.8
print(corr)
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder, Standard-
Scaler
le = preprocessing.LabelEncoder()
y=label.iloc[:,-1]
# print(y.shape)
# Create X and y, where X represents feature values and y repre-
sents target values.
dataset = df.values
X = dataset[:,:]
# lable encoder
y=le.fit_transform(y)
# Split the data into training and test datasets.
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=123, stratify=y)
# Instantiation
dtree = tree.DecisionTreeClassifier(
   criterion='entropy',
   max_depth=5, # Setting the tree depth is a common technique
to prevent overfitting.
min_weight_fraction_leaf=0.005   #   Define   the   minimum
percentage of samples required in a leaf node to prevent
overfitting.
# Model training
dtree = dtree.fit(X_train,y_train)
# Output results
print ("Decision Tree:train score:", dtree.score (X_train, y_
train))
print("Decision Tree:test score:", dtree.score (X_test, y_
test))
# Get feature importance
importances = dtree.feature_importances_
# Get feature names
feat_names = df.columns
# Sorting
indices = np.argsort(importances)[::-1]
# plotting
plt.figure(figsize=(10,6))
plt.title("Feature importances by Decision Tree")
```

```
plt.bar(range(len(indices)),          importances[indices],
color='lightblue', align="center")
# plt.step(range(len(indices)), np.cumsum(importances[indi
ces]), where='mid', label='Cumulative')
plt.xticks(range(len(indices)),                    feature_
list[indices],fontsize=14)
plt.xlim([— 1, len(indices)])
# The results are presented in Fig. 5.9
plt.show()
# Instantiated random forest
rf = RandomForestClassifier(
    n_estimators=100,
   criterion='entropy',
    max_depth=7, # Defines the depth of the tree, which can be
used to prevent overfitting
min_samples_split=10, # Continue bifurcation when defining at
least how many samples
#min_weight_fraction_leaf=0.02 # Define how many samples the
leaf node needs to contain at least (expressed in percentage)
to prevent over fitting)
# model training
rf.fit(X_train, y_train)
# Output
print("Random  forest:train  score:",  rf.score(X_train,  y_
train))
print("Random forest:test score:", rf.score(X_test, y_test))
# Feature importance ranking
importances = rf.feature_importances_
# Feature name
feat_names = df.columns
# Sorting
indices = np.argsort(importances)[::-1]
# plotting
plt.figure(figsize=(10,6))
plt.title("Feature importances by RandomForest")
plt.bar(range(len(indices)),importances[indices],
color='lightblue', align="center")
#plt.step(range(len(indices)),  np.cumsum(importances[indi
ces]), where = 'mid', label = 'Cumulative')
plt.xticks(range(len(indices)),                    feature_
list[indices],fontsize=14)
plt.xlim([— 1, len(indices)])
# The results are presented in Fig. 5.10
plt.show()
# Save the model
joblib.dump(dtree, "train_model_dtree.m")
# Model invocation
clf = joblib.load("train_model_dtree.m")
```

```
# Save the model
joblib.dump(rf, "train_model_rf.m")
# Model invocation
clf = joblib.load("train_model_rf.m")
```

By examining Fig. 5.7, we can observe several distribution characteristics of the data, such as maximum and minimum values, median, and mean/average. These insights are highly effective for tasks like identifying and removing outliers and tuning model parameters.

By examining Fig. 5.8, we can observe the degree of correlation between pairwise features. The feature correlation symmetric matrix was generated using the sns.heatmap() function in Seaborn. Notably: The values on the main diagonal represent the correlation of each feature with itself, which are always 1. Among all off-diagonal blocks, the lightest color corresponds to feature2 and feature3, indicating a strong positive linear correlation between them.



	r_desc_Average_connectivity_index_chi-2	r_desc_Maximal_electrotopological_negative_variation	r_desc_Second_Mohar	r_desc_Second_Zagreb_index_by_valence_vertex_degrees	r_qp_ACxDN^.5/SA
count	2013.000000	2013.000000	2013.000000	2013.000000	2013.000000
mean	0.283338	2.493498	4.484718	421.268555	0.014486
std	0.010504	1.354086	1.253902	98.238761	0.006686
min	0.256289	0.525301	1.111708	114.748741	0.000000
25%	0.275712	1.724464	3.539995	353.000000	0.010395
50%	0.282482	2.022703	4.416706	415.000000	0.013929
75%	0.290212	2.684856	5.303147	477.000000	0.018048
max	0.325655	6.701957	16.528154	1040.666667	0.058340

Fig. 5.7 Data distribution of the kinase inhibitor dataset

Fig. 5.8 Feature correlation plot

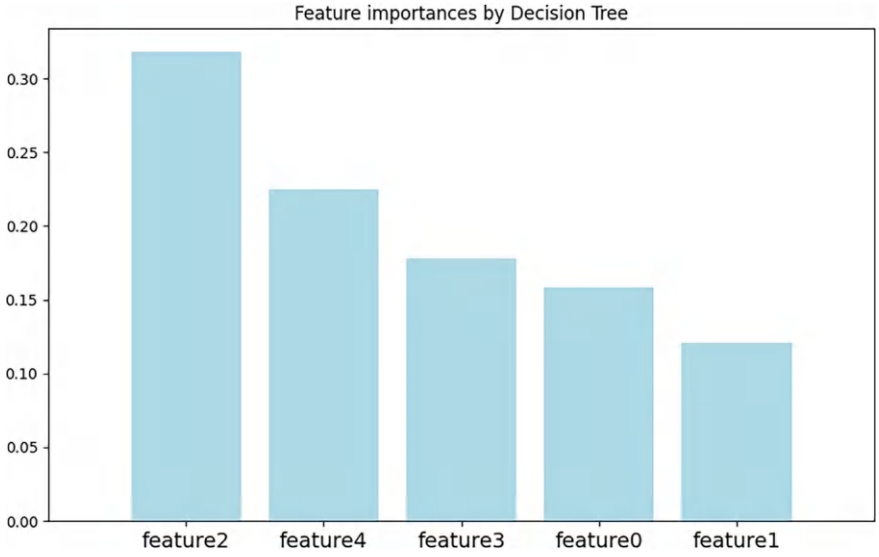


Fig. 5.9 Feature importance calculated by decision tree

From Figs. 5.9 and 5.10, we can obtain the ranking of feature importance. A comparison reveals slight discrepancies in feature importance between the decision tree and RF models, though the differences are minimal. These figures help identify which features are relevant to the classification of kinase inhibitors. Additionally, manually removing features with low importance may potentially improve model predictive performance, though the effectiveness requires empirical validation.

In terms of results, both decision tree and RF can effectively classify kinase inhibitors, and the results of RF are slightly better than those of the decision tree, as shown in Table 5.2. At the same time, it also shows that ensemble learning can effectively improve the prediction and generalization ability of the model, and can train the model to a certain extent.

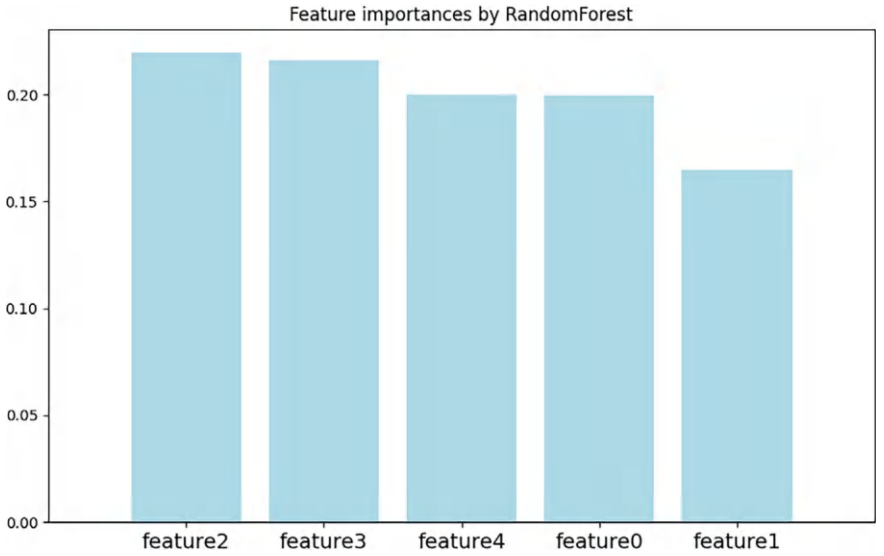


Fig. 5.10 Feature importance calculated by random forest

Table 5.2 Accuracy scores for decision tree and random forest

Model name/loss	Train_score	Test_score
Decision tree	0.743	0.737
Random Forest	0.808	0.769

Note

In Fig. 5.4, the predictions of primary learners on the training set typically use the results of multi-fold cross-validation as features for the meta-learner, rather than direct predictions of the primary learners on the training set. Specifically, Fig. 5.4 illustrates a four-fold cross-validation approach for this process.

References

1. Srivastava, P.K., Petropoulos, G.P., Prasad, R., Triantakonstantis, D.: Random forests with bagging and genetic algorithms coupled with least trimmed squares regression for soil moisture deficit using SMOS satellite soil moisture. ISPRS Int. J. Geo Inf. **10**, 507 (2021)
2. Chen, H.-Y., et al.: Deep learning and random forest approach for finding the optimal traditional chinese medicine formula for treatment of alzheimer's disease. J. Chem. Inf. Model. **59**, 1605–1623 (2019)
3. Chen, X., et al.: Database of traditional Chinese medicine and its application to studies of mechanism and to prescription validation. Br. J. Pharmacol. **149**, 1092–1103 (2006)
4. Ru, X., Li, L., Zou, Q.: Incorporating distance-based top-n-gram and random forest to identify electron transport proteins. J. Proteome Res. **18**, 2931–2939 (2019)

Chapter 6
Neural Networks

Artificial Neural Network (ANN), hereinafter referred to as Neural Network (NN), is an important algorithm in machine learning and also the fundamental algorithm that underpins the development of deep learning.

The evolution of neural networks has been a torturous journey, marked by several pivotal events. In 1943, physiologist W. S. McCulloch and mathematician W. A. Pitts proposed the M-P model [1], a mathematical model of neurons, which was an abstract and simplified version based on the structure and working principle of biological neurons. In 1958, computer scientist Frank Rosenblatt introduced the perceptron, an artificial neural network with a learning algorithm, capable of solving some linearly separable problems [2]. The theoretical model of the perceptron utilized the M-P model and had a simple and feasible learning algorithm. In 1982, John Hopfield proposed a new type of neural network [3], a fully connected network with associative and memory functions, which was easy to implement in hardware. This brought about a revival of neural networks after a period of dormancy. In 1986, Rumelhart and McClelland introduced the BP neural network [4], a multi-layer feedforward network trained by the backpropagation algorithm, which is one of the most widely used neural network models today.

In recent years, with the rapid development of computer hardware and big data, neural networks have achieved remarkable results in fields such as speech recognition, image recognition, pattern recognition, smart healthcare, and drug design [5].

6.1 Inspiration of Biological Neurons for Artificial Neurons

A biological neural network is a complex network composed of billions of interconnected neurons [6]. The human thinking process can roughly be regarded as the biological signals received by human sensory organs being processed through this

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_6

complex network, ultimately leading to the desired answers and responses. Neurons, also known as nerve cells, are highly differentiated cells and one of the basic structural and functional units of the nervous system, possessing the functions of receiving stimuli and transmitting signals. As can be observed from Fig. 6.1, a neuron is composed of a cell body and protrusions. Among them, the protrusions are divided into dendrites and axons. There are more dendrites for receiving information, while there is generally only one axon for transmitting information, with many branches at the end. The integration and processing of information are accomplished by the cell body. The workflow of a neuron is as follows: the biological electrical potential signals transmitted from other neurons are received through dendrites and integrated and calculated in the cell body. When the integrated potential reaches the potential threshold, a signal of a certain intensity is triggered and transmitted out of the neuron through the axon to other related neurons. The more frequently signals are transmitted between neurons, the closer their connection becomes. If the integrated potential does not reach the threshold, no signal will be triggered. If the value is low, no signal is transmitted from that neuron, and the connection between the preceding and following neurons thus weakens.

The main functions of the various parts of a biological neuron are shown in Table 6.1.

Inspired by biological neural networks, researchers in artificial intelligence aim to develop algorithms similar to biological neural networks, with the goal of simulating the human brain's learning process. These algorithms are designed to learn and summarize patterns from certain features, achieving the ability to respond accurately to familiar features. Thus, the concept of artificial neural networks emerged.

From the perspective of information processing, artificial neural networks abstract biological neural networks and establish a certain simple model by using artificial neurons (referred to as neurons) as units and connecting them in a certain way. The

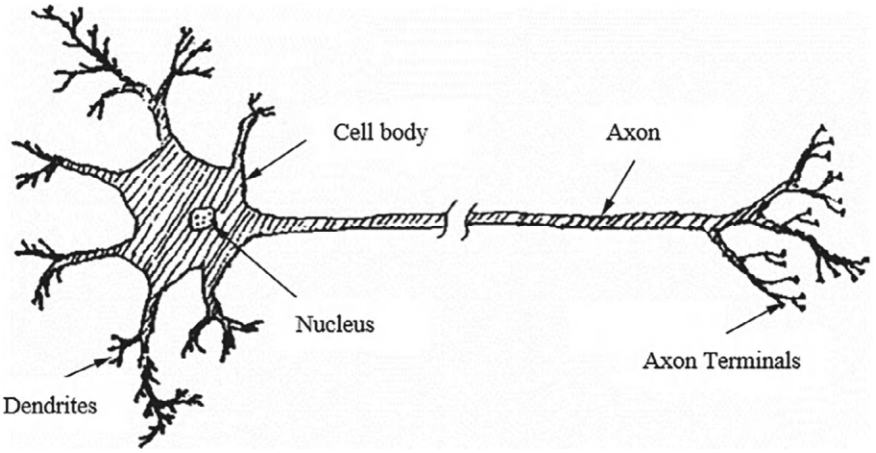


Fig. 6.1 Schematic diagram of neuron structure

Table 6.1 Main functions of different parts of a biological neuron

Name	Key features
Dendrites	Input unit: receives biological signals or other signals transmitted by multiple neurons connected to it
Cell body	Processing unit: Integrates all input dendrite signals. If the signal exceeds the excitation threshold, it outputs an excitation signal; otherwise, it outputs an inhibition signal
Axon	Output unit: transmits an excitatory or inhibitory signal from the cell body to other neurons connected to the axon through the dendrite terminals

M-P neuron model is an abstract and simplified model constructed based on the structure and working principle of biological neurons. It was proposed by Warren McCulloch and Walter Pitts in 1943. Table 6.2 shows the symbols and their meanings involved in the M-P neuron model diagram (Fig. 6.2).

The arrows on both sides respectively simulate the dendrites (input units) and axons (output units) of biological neurons [7], while the ellipse in the middle simulates the cell body (processing unit) of a biological neuron. The workflow of the M-P

Table 6.2 Symbols and meanings in the M-P neuron model

Symbol	Meaning
$x_1, x_2, \ldots, x_n$	The 1st to nth input signals
$w_{1j}, w_{ij}, \ldots, w_{nj}$	The weight value corresponding to the 1st to nth input on the jth neuron simulates the connection strength between biological neurons
Σ	Cumulative summation of weighted signal values
b_j	The upper threshold (bias) of the jth neuron simulates the excitation threshold of biological neurons
$f(\cdot)$	The activation function of the jth neuron adjusts the output signal strength and has a normalizing effect
y_j	The final output signal of the *j*-th neuron

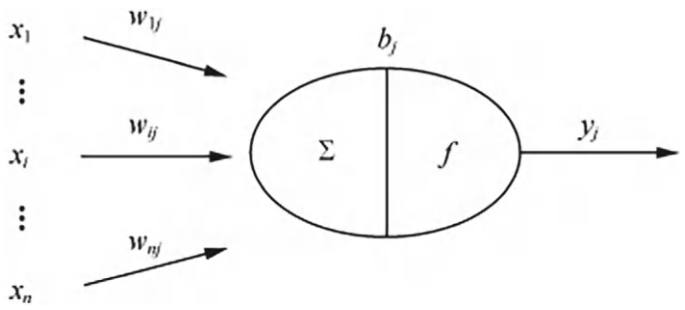


Fig. 6.2 M-P neuron model

model: A neuron receives input signals from n other neurons. These input signals are weighted and summed up. The result is compared with the threshold b, and then processed by the activation function to obtain the output y_j of the neuron, which is expressed as

$$y_j = f\left(\sum_{i=1}^{n}\left(w_{ij}x_i + b_j\right)\right) \tag{6.1}$$

The role of an activation function is to determine whether information is transmitted from a neuron and how much information is transmitted, based on the calculation of the activation function.

6.2 The Main Differences Between Biological Neural Networks and Artificial Neural Networks

So far, no neural network created by humans can match the complexity of the human brain's neural network, not to mention that there are still many unsolved mysteries about biological neural networks. For instance, an artificial neural network can be composed of hundreds of millions of neurons like a biological neural network, but it cannot imitate the irregular and hard-to-explain network structure of the biological neural network because it is difficult to make such a "disorderly" structure work with the existing technical means [8].

For instance, the human brain acquires information through a unified entry point, the sensory organs, which is then processed in the neural network, ultimately leading to consciousness and response. This implies that throughout a person's life, numerous learning tasks, big and small, are accomplished through this complex neural network system, and they can be learned autonomously. In contrast, artificial neural networks typically struggle to handle multiple learning tasks. They often achieve satisfactory results for one type of learning task but find it difficult to perform well in others. Moreover, at the initial stage of constructing an artificial neural network, researchers must meticulously arrange the neurons and configure the algorithms. From this, it is evident that the intelligence of artificial neural networks is significantly lacking in "confidence".

The complexity of biological neural networks is also reflected in their strong fault-tolerant ability. Every day, a large number of neurons in the brain die normally, but this does not affect the normal function of the brain. Moreover, certain functional declines caused by local damage to the brain may gradually recover. Once an artificial neural network is designed, each neuron plays a crucial role in the operation of the entire network under normal circumstances. The failure of a single neuron may affect the training effect of the entire neural network, and it will lead to the collapse of the entire neural network.

In addition, there are also obvious differences between biological neural networks and artificial neural networks in terms of connection structure and learning methods.

Humans acquire knowledge through training. The principle is that relevant learning features are transmitted to the neural network through the sensory organs. At this point, the neurons in the neural network are rather disordered; some have no connection with each other, while others have connections that are either strong or weak. The neural network continuously adjusts the connections between neurons (such as establishing new connections, weakening or strengthening existing connections, or deleting connections) based on the learning features. Eventually, the part of the neural network related to the learning reaches a stable state, meaning that humans have achieved a relatively stable understanding of the skill at this point. During the learning process, it is inevitable for humans to have incorrect perceptions of knowledge. However, the error-correction mechanism of the biological neural network lies in continuing to learn the relevant knowledge and further adjusting the neural network structure to reach a stable state again. In this way, the learning process of a certain knowledge is completed. When encountering problems related to this knowledge, as long as the key features related to the knowledge in the problem are extracted, a quick response can be made through the neural network.

Before training a task with an artificial neural network, researchers must design the architecture and algorithm of the neural network. The architecture of a neural network includes the arrangement and connection structure of neurons, which is significantly different from the "disorderly" arrangement and connection structure of neurons in biological neural networks. In artificial neural networks, neurons are generally arranged in layers, so we call the neurons arranged in one layer as a "neural network layer". People classify these layers into three categories according to their distribution positions: input layer, hidden layer and output layer. Neurons in adjacent neural network layers are interconnected, while neurons in the same layer are generally not connected. Sometimes, neurons in non-adjacent layers are also interconnected. Researchers need to design the algorithm of the neural network in advance, including which activation function to choose for a certain layer of the neural network and which optimization algorithm to configure for the neural network.

The operation process of an artificial neural network is as follows: a set of features enters the neural network through the input neurons, and after being processed layer by layer through the network layers, the result is finally output through the output neurons. This process can only be called operation or forward propagation, and not learning or training, because during the entire process, the parameters within the neural network do not change due to the feature data.

The training of artificial neural networks is achieved through the "error correction mechanism", known as "backpropagation". In simple terms, when the output result of the feature processed by the neural network algorithm differs from the actual result, the algorithm calculates the error value and then feeds the error value back to the hidden layer through backpropagation to adjust the parameters. This process is repeated continuously until a result consistent with expectations is obtained.

6.3 Feedforward Neural Network

Feedforward neural networks are one of the earliest invented and utilized simple neural network types in the field of artificial intelligence, and they are also one of the most widely applied and rapidly developing neural networks at present. In a feedforward neural network, neurons are arranged in a hierarchical structure, with neurons in adjacent layers being interconnected. The output of neurons in the previous layer serves as the input for neurons in the subsequent layer. There are no connections among neurons within the same layer, and the topological structure of the entire neural network contains no loops or cycles [9].

The network structure of the feedforward neural network is shown in Fig. 6.3, which consists of three parts.

Input layer: Located at the first layer, it is used for inputting data.
Hidden layer: Located between the input layer and the output layer, it is used to process data.
Output layer: Located at the last layer, it is used for outputting data.

The symbols and their meanings of the feedforward neural network are shown in Table 6.3.

Let $a^{(0)} = x$, The feedforward neural network propagates information by continuously iterating the formula, that is,

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)} \tag{6.2}$$

$$a^{(l)} = f_l\left(z^{(l)}\right) \tag{6.3}$$

It can also be combined into

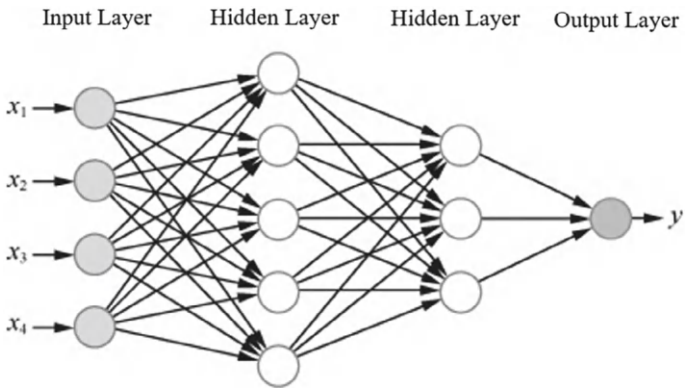


Fig. 6.3 Structure diagram of a feedforward neural network

Table 6.3 Symbols and meanings of feedforward neural network

Symbol	Meaning
L	Number of layers in a neural network
M_l	The number of neurons in layer l
$f_l(\cdot)$	Activation function of neurons in layer l
$W^{(l)} \in R^{M_l \times M_{l-1}}$	The weight matrix from layer $l-1$ to layer l
$b^{(l)} \in R^{M_l}$	Bias from layer $l-1$ to layer l
$z^{(l)} \in R^{M_l}$	The input value of the neuron in the lth layer
$a^{(l)} \in R^{M_l}$	The output value of the neurons in layer l

$$z^{(l)} = W^{(l)} f_{l-1}\left(z^{(l-1)}\right) + b^{(l)} \tag{6.4}$$

Or

$$a^{(l)} = f_l\left(W^{(l)} a^{(l-l)} + b^{(l)}\right) \tag{6.5}$$

In this way, the feedforward neural network can obtain the final output $a^{(L)}$ through the information transmission layer by layer. The entire neural network can be regarded as a composite function $\Phi(x; W, b)$, taking the vector $\boldsymbol{x}$ as the input $a^{(0)}$ of the first layer and the output $a^{(L)}$ of the Lth layer as the output of the entire function.

$$x = \alpha^{(0)} \to z^{(1)} \to \alpha^{(1)} \to z^{(2)} \to ? \to \alpha^{(L-l)} \to z^{(L)} \to \alpha^{(L)} = \Phi(x; W, b) \tag{6.6}$$

Here, W and b represent the connection weights and biases of all layers in the network.

The previous section briefly introduced the forward transmission process of the feedforward neural network. We found that the quality of the output of a neural network is determined by the parameters W and b. Therefore, training a neural network is essentially a process of continuously updating these two parameters to achieve the best results, that is, using backpropagation to adjust the weights and biases of the network layer by layer in reverse based on the output values.

6.4 Backpropagation Algorithm

The backpropagation algorithm can be divided into two parts: forward transmission and backward feedback. Forward transmission is responsible for computing the input features layer by layer through each network layer to obtain the final result, while backward feedback adjusts the weights and biases of the network layer by layer in the reverse direction based on the output values.

Suppose that stochastic gradient descent is adopted for learning the parameters of a neural network. Given a sample (x, y), it is input into the neural network model to obtain the network output y'. Assuming the loss function is $L(y, y')$, to perform parameter learning, it is necessary to calculate the derivative of the loss function with respect to each parameter. The partial derivatives of the parameter weights $W^{(l)}$ and biases $b^{(l)}$ in the l-th layer are calculated.

$$\frac{\partial L(y,y')}{\partial W_{ij}^{(l)}} = \frac{\partial L(y,y')}{\partial z^l}\left(\frac{\partial z^l}{\partial W_{ij}^{(l)}}\right) \tag{6.7}$$

$$\frac{\partial L(y,y')}{\partial b^{(l)}} = \frac{\partial L(y,y')}{\partial z^l}\left(\frac{\partial z^l}{\partial b^{(l)}}\right) \tag{6.8}$$

where, $\frac{\partial L(y,y')}{\partial z^l}$ is the partial derivative of the objective function with respect to the neuron $z^{(l)}$ of layer l, called the error term. Therefore, it is necessary to compute three partial derivatives, namely: $\frac{\partial L(y,y')}{\partial z^l}$, $\frac{\partial z^l}{\partial W_{ij}^{(l)}}$, $\frac{\partial z^l}{\partial b^{(l)}}$ which leads to:

$$\frac{\partial z^l}{\partial W_{ij}^{(l)}} = \frac{\partial\left(W^{(l)}a^{(l-1)}+b^{(l)}\right)}{\partial W_{ij}^{(l)}} \tag{6.9}$$

$$\frac{\partial z^l}{\partial b^{(l)}} = \frac{\partial\left(W^{(l)}a^{(l-1)}+b^{(l)}\right)}{\partial b^{(l)}} \tag{6.10}$$

So the i-th row of the weight matrix $W^{(l)}$ is $a^{(l-1)}$, and the bias is b.

Since $\partial z^{(l+1)} = W^{(l+1)}a^{(l)} + b^{(l+1)}$, and $a^{(l)} = f_l(z^{(l)})$, and $f_l(\cdot)$ is a function calculated element-wise, it follows that

$$\frac{\partial z^{(l+1)}}{\partial a^l} = \left(W^{(l+1)}\right)^{\mathrm{T}} \tag{6.11}$$

$$\frac{\partial a^l}{\partial z^l} = diag\left(f_l\left(z^l\right)\right) = \frac{\partial L(y_l,y_l\prime)}{\partial z^l} = \frac{\partial(y_l-y_l\prime)}{\partial z^l} \tag{6.12}$$

$$= \frac{\partial\left(y_l-W^l a^{(l-1)}+b^l\right)}{\partial z^l} = \frac{\partial\left(y_l-W^l f_{l-1}\left(z^{(l-1)}\right)+b^l\right)}{\partial z^l} \tag{6.13}$$

The partial derivatives with respect to the input are thus obtained.

$$\frac{\partial L\left(y_l,y_l'\right)}{\partial z^l} = \frac{\partial L\left(y,y'\right)}{\partial z^{l+1}}\frac{\partial z^{(l+1)}}{\partial a^l}\frac{\partial a^l}{\partial z^l} \tag{6.14}$$

We define the error term of the *l-th* layer neuron as δ^l, then

$$\delta^l = diag\left(f_1\prime\left(z^l\right)\right)\cdot\left(W^{(l+1)}\right)^{\mathrm{T}}\cdot\delta^{(l+1)} = f_1\prime\left(z^l\right)\Theta\left(\left(W^{(l+1)}\right)^{\mathrm{T}}\cdot\delta^{(l+1)}\right) \tag{6.15}$$

Among them, $diag(x)$ is a diagonal matrix whose diagonal elements are x. Θ is the dot product operator for vectors, indicating the multiplication of corresponding elements. From Eq. (6.15), it can be seen that the error term of the *l-th* layer can be

calculated from the error term of the $l+1$-th layer, which is the backpropagation of errors. After calculating the three partial derivatives, the formula can be written as

$$\frac{\partial L(y_l, y_l')}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} \prod_i (\alpha_j^{(l-1)})^T = \delta_i^{(l)} a_j^{(l-1)} \tag{6.16}$$

Therefore, the gradient of $\partial L(y_l, y_l')$ with respect to the weight $W^{(l)}$ for the l-th layer is

$$\frac{\partial L(y_l, y_l')}{\partial W^{(l)}} = \delta_i^{(l)} \left(a_j^{(l-1)} \right)^{\mathrm{T}} \tag{6.17}$$

Similarly, the gradient of $\partial L(y_l, y_l')$ with respect to the bias $b^{(l)}$ of the l-th layer is

$$\frac{\partial L(y_l, y_l')}{\partial b^{(l)}} = \delta_i^{(l)} \tag{6.18}$$

After calculating the error terms for each layer, we can obtain the gradients of the parameters for each layer and update the parameters accordingly. By following the above formula, the weights can be updated. The process of the backpropagation algorithm is actually to adjust the parameters of the model based on the samples. By repeatedly performing the forward transmission and backward feedback processes, the error will become smaller and smaller, the weights will become more and more stable, and the accuracy of the model will also increase continuously.

The above is the method for calculating the error of each node in a neural network and updating the weights. It can be seen that to calculate the error value of a node, it is first necessary to calculate the error values of each neuron connected to the nodes of the next layer. Therefore, the calculation sequence of the error values must start from the output layer and then calculate the error values of each neuron in the hidden layers in reverse order until the first hidden layer. This is the meaning of the backpropagation algorithm.

With the development of neural networks, the methods for updating weights have become relatively simple and mature. In summary, they involve calculating the gradient and performing gradient descent. The direction of the gradient indicates the direction in which the error expands; thus, when updating the weights, the gradient needs to be negated to reduce the error caused by the weights. In practical engineering applications, the weight update methods will also adjust the weight update strategies according to the actual situation.

6.5 Activation Function

The essence of neural networks can be regarded as "fitting the true functional relationship between features and results through parameters and activation functions". A single-layer neural network can only perform linear classification tasks, but a two-layer neural network can approximate any continuous function infinitely closely.

The function in a neuron consists of two parts: a linear function and a nonlinear function. The linear function part is the sum of the weighted inputs from the upper-layer neurons. The nonlinear function part is the activation function. If a neuron does not contain a nonlinear part, then no matter how many neurons are in the network, the numerical calculation effect produced is nothing more than matrix multiplication, and it cannot fit nonlinear transformation effects. No matter how many layers a neural network has, the effect is no different from that of a single-layer neural network.

In addition, the introduction of activation functions enables data normalization, which not only ensures the stability of neural network training but also significantly reduces the computational pressure on the neural network from data. Common activation functions in neural networks include the Logistic function, Tanh function, ReLU function, Leaky ReLU function, and Swish function.

6.5.1 Logistic Function

The Logistic function, also known as the Sigmoid function, can transform the continuous real-valued input into an output between 0 and 1. If the input is a very large negative number, the output will be 0; if it is a very large positive number, the output will be 1. This characteristic is similar to that of biological neurons, which will be excited (output 1) for some inputs and inhibited (output 0) for others [10]. The Logistic function is defined as

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{6.19}$$

Figure 6.4 shows the curve of the Logistic function.

The Logistic function is rarely used in practical engineering models, mainly due to three reasons. First, when the Logistic function approaches 0 or 1, it becomes flat, meaning the gradient approaches 0. When this activation function is used in neural networks for backpropagation, if the output values are close to 0 or 1, the weights of these neurons will not be updated, and neither will those of the neurons connected to them.

The weights are also updated very slowly. This problem is called vanishing gradient, which can prevent the neural network from performing backpropagation and thus hinder the training of the neural network. First, the training effect is very poor. Second, the output of this function is not centered around zero, which is not

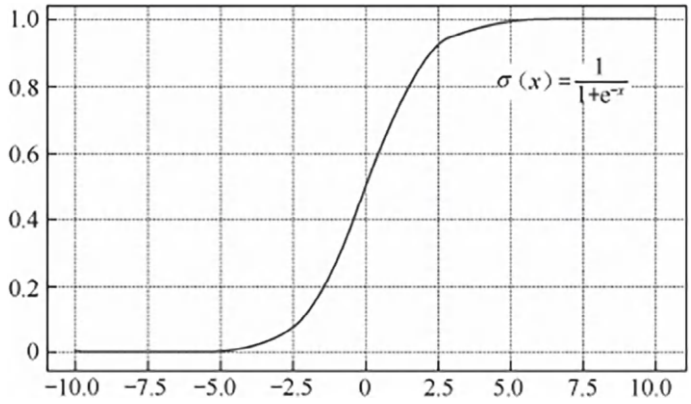


Fig. 6.4 Curve of the logistic function

conducive to the calculation of the next network layer. Third, using this function will lead to a relatively high computational cost for the neural network.

6.5.2 Tanh Function

The Tanh activation function is also known as the hyperbolic tangent activation function. The expression of the Tanh function is

$$\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{6.20}$$

Figure 6.5 shows the curve of the Tanh function.

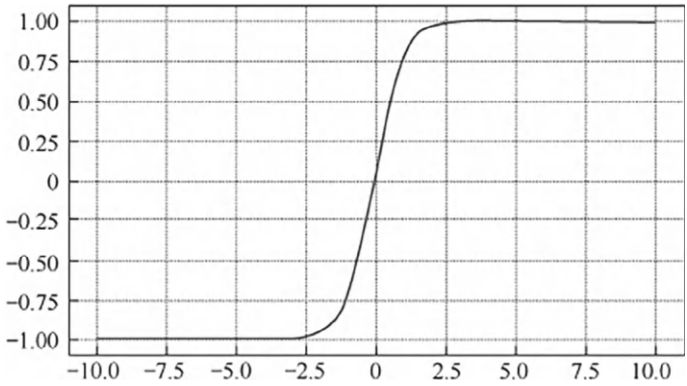


Fig. 6.5 Curve of the tanh function

The difference from the Logistic function is that the Tanh function compresses the calculated data values into the range of (− 1, 1), which is beneficial for further operations in the lower network layers. However, the Tanh function also has the problem of vanishing gradients. Next, we will discuss another nonlinear activation function, the Rectified Linear Unit (ReLU) function [11], which is significantly superior to the previous two functions and is widely used nowadays.

6.5.3 ReLU Function

The ReLU function is a type of function that is semi-corrected from the bottom up. The definition of the ReLU function is

$$\sigma(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases} \tag{6.21}$$

Figure 6.6 shows the curve of the ReLU function.

When the input x is less than 0, the output is 0; when x is greater than or equal to 0, the output is x. This activation function enables the network to converge more rapidly. It does not saturate, meaning it can counter the vanishing gradient problem, at least in the positive region (when x is greater than or equal to 0). Therefore, neurons will not backpropagate all zeros in at least half of the region. Due to the use of simple thresholding,

The ReLU function is highly computationally efficient. However, ReLU neurons also have some drawbacks.

(1) Not centered at 0: Similar to the Logistic activation function, the output of the ReLU function is not centered at 0.

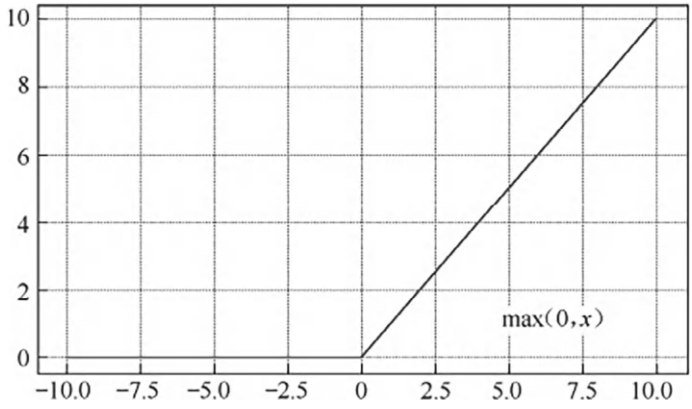


Fig. 6.6 Curve of the ReLU function

(2) During the forward transmission process, if x < 0, the output value of the neuron is 0. In the backward propagation process, the gradient is 0, which means the weights cannot be updated and the network cannot learn.

To address the vanishing gradient problem in the ReLU activation function, researchers have designed the Leaky ReLU function.

6.5.4 *Leaky ReLU Function*

The Leaky ReLU function maintains a small gradient γ when the input $x < 0$. This ensures that there is a non-zero gradient to update parameters even when the neuron is not activated, preventing it from never being activated. The expression of the Leaky ReLU function is

$$\begin{aligned}\sigma(x) &= \begin{cases} x & x > 0 \\ \gamma x & x \leq 0 \end{cases} \\ &= \max(0, x) + \gamma \min(0, x)\end{aligned} \tag{6.22}$$

Among them, γ is a very small constant, for example, 0.1. When $\gamma < 1$, the Leaky ReLU function can also be written as

$$\sigma(x) = \max(0, x) \tag{6.23}$$

As shown in Fig. 6.7, it is the curve of the Leaky ReLU function.

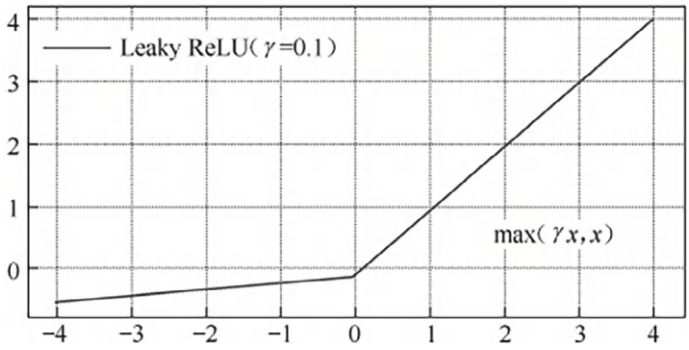


Fig. 6.7 Curve of the leaky ReLU function

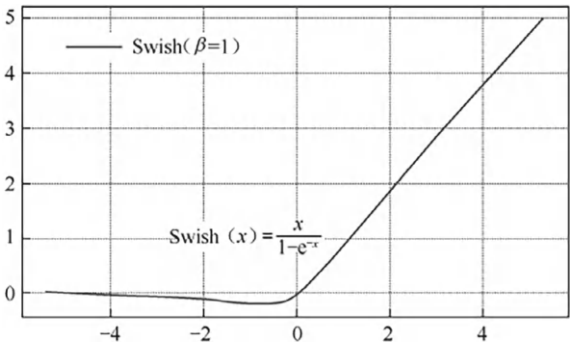


Fig. 6.8 Curve of the swish function (ß = 1)

6.5.5 Swish Function

The Swish function, also known as the self-gated activation function, is defined as

$$Swish(x) = x\sigma(\beta x) \tag{6.24}$$

Here, $\sigma(\cdot)$ is the Logistic function, and β is a learnable parameter or a fixed hyper-parameter. $\sigma(\cdot) \in (0, 1)$ can be regarded as a soft gating mechanism. When $\sigma(\beta x)$ approaches 1, the gate is in the "open" state, and the output of the activation function is approximately x itself; when $\sigma(\beta x)$ approaches 0, the gate is in the "closed" state, and the output of the activation function is approximately 0.

As shown in Fig. 6.8, it is the curve of the Swish function.

6.6 Build a Neural Network with TensorFlow to Classify Kinase Inhibitors

This section will use TensorFlow to build a neural network to achieve the classification function of kinase inhibitors. The implementation process of a neural network is generally divided into four parts: data preprocessing, neural network construction, model training, and model application [12].

6.6.1 Data Preprocessing

Neural networks typically use relatively standardized data as input parameters, as only high-quality data can potentially lead to high-quality processing results. However, in actual engineering practice, data is often incomplete, unstandardized,

inconsistent, or contaminated. Therefore, we need to perform data preprocessing on the dataset according to the task requirements.

In Listing 6.1, the 'read_csv()' function in Pandas is used to read the 'kinase_selected.csv' data table, and the data table is visualized as shown in Fig. 6.9. Observe the data table to facilitate its preprocessing.

Listing 6.1 Reading the Dataset

```
import pandas as pd
import numpy as np
dataframe = pd.read_csv("kinase_selected.csv")
print(dataframe.head())
```

The first column is a serial number and is irrelevant data. The last five columns are feature data. Therefore, the data in the last five columns are extracted as the feature set. Then, standardize each column of data in the feature set so that each number is around 0. This processing not only does not affect the final training effect, but also it is conducive to improving the operational efficiency of neural networks. The standardized processed dataset X_scaled is taken as the final feature dataset. The code implementation is shown in Listing 6.2, and the running result is presented in Fig. 6.10.

	Unnamed: 0	r desc Average connectivity index chi-2	r_desc Maximal electrotopological negative variation	r desc Second Mohar	r desc Second Zagreb index by valence vertex degrees	r qp ACxDN^.5/SA
0	0	0.293245	1.902150	7.867279	611.000000	0.021358
1	1	0.300698	6.693446	5.678195	347.777778	0.058341
2	2	0.274048	0.987404	4.915686	401.000000	0.009612
3	3	0.280079	1.899112	7.295905	650.000000	0.019622
4	4	0.287003	2.405486	3.201728	481.000000	0.022678

Fig. 6.9 Samples of the dataset

Fig. 6.10 The scale of the feature set X_scaled

```
X_scaled.shape: (2013, 5)
```

	cid	SMILES	Binding_Modes
0	748	Cc1ccc(cc1Nc2nccc(n2)c3cccnc3)NC(=O)c4ccc(cc4)...	II
1	ANK	c1nc(c2c(n1)n(cn2)C3C(C(C(O3)COP(=O)(O)OP(=O)(...	I1/2
2	4YV	COC1CCN(CC1)c2nccc(n2)Nc3cc4c(cn3)ncn4C5CCCC5	I
3	P38	Cc1ccc(cc1Nc2c3c(c(cn3ncn2)C(=O)NC(C)c4ccccc4)...	I1/2
4	G93	CCn1c2c(cnc(c2nc1c3c(non3)N)C#CC(C)(C)O)OCC4CC...	I

Fig. 6.11 Original labels of kinase inhibitors

Listing 6.2 Determining the Feature Dataset

```
dataset = dataframe.values
X = dataset[:, 1:]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print(X_scaled.shape)
```

In Listing 6.3, we read the label_kinase_selected.csv file and select the data in the last column as the result label. The visualization of the label_kinase_selected.csv file is shown in Fig. 6.11.

Listing 6.3 Read and Display the label_kinase_selected.csv Data Table

```
label = pd.read_csv('label_kinase_selected.csv')
print(label.head())
```

The data of the neural network result set is in the last column of the label_kinase_selected.csv table, so it is read into the variable. Check the scale of the y result set. The data volume of y is consistent with that of the X_scaled feature set, both being 2013 entries. The last output statement outputs the statistics of the number of values in the label data table, that is, the types of the result set and the corresponding counts. There are a total of 4 labels. There are 1399 I labels, 377 I1/2 labels, 190 II labels, and 47 with A labels. The code implementation is shown in Listing 6.4, and the output result is shown in Fig. 6.12.

Fig. 6.12 Output results

```
y.shape: (2013,)
        Binding_Modes
I                1399
I1/2              377
II                190
A                  47
```

Listing 6.4 Number of Labels in the label_kinase_selected.csv Data Table

```
label = pd.read_csv('label_kinase_selected.csv')
print(label.head())
y = label.iloc[:, -1]
print('y.shape:',y.shape)
print(pd.DataFrame(label['Binding_Modes'].value_counts()))
```

In Listing 6.5, the LabelEncoder is defined to encode the result label set. By using the 'le.inverse_transform()' function to construct the dictionary 'label_encoder_dict', the specific details of the encoding can be viewed. The output 'Y' shows that the original column of string data for the result labels has been converted into a set of numerical encodings. Next, 'keras.utils.to_categorical()' is used. The function performs One-Hot encoding on the List of numbers. Since this neural network is designed for a 4-class classification problem, after One-Hot encoding, the dataset values become a List of arrays each of length 4, which precisely matches the number of output nodes in the neural network designed for 4-class classification. Runing result is shown in Fig. 6.13.

Listing 6.5 Codes of Processing of Result Data Sets

```
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras
le = LabelEncoder()
# label encoder
Y = le.fit_transform(y)
label_encoder_dict = dict(
   zip(le.inverse_transform([0, 1, 2, 3]), [0, 1, 2, 3]))
print('label_encoder_dict:',label_encoder_dict)
```

```
label_encoder_dict: {'A': 0, 'I': 1, 'I1/2': 2, 'II': 3}
label_encoder: [3 2 1 ... 2 2 1]
One-Hot_Encoder:
 [[0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]]
```

Fig. 6.13 Running results

```
print('label_encoder:',Y)
Y = keras.utils.to_categorical(Y)
print('One-Hot_Encoder:\n',Y[:5])
```

6.6.2 Split the Dataset

In the data preprocessing stage, the processed feature set and result set are respectively saved in X_scaled and Y. Next, we use the train_test_split() function to split the dataset. The train_test_split() function has four parameters: the feature set, the result set, the dataset split ratio, and the random state. The dataset split ratio is set to 0.2, meaning that the training set accounts for 80% of the total samples and the test set accounts for 20%. The setting of the random state parameter ensures that the dataset split returned by the program is the same at any time and on any device.

The 'train_test_split()' function returns four parameters, namely the feature training set, feature test set, result training set, and result test set, which are 'X_train', 'X_test', 'Y_train', and 'Y_test', as shown in Listing 6.6.

Listing 6.6 Codes of Dataset Partitioning

```
X_train, X_test, Y_train, Y_test = train_test_split(
   X_scaled, Y, test_size=0.2, random_state=42)
```

6.6.3 Neural Network Construction

Import the Sequential(), Dense(), and Dropout() functions from the keras.layers module. Keras is a high-level neural network API that allows us to easily build the neural networks we need using the pre-packaged functions and models it provides. First, define a Sequential object named model-1. Sequential is a sequential model in Keras. Next, Use the 'add()' method to add neural network layers to it one by one.

Dense is a fully connected layer. In this program, Dense() has three parameters: the first parameter is the number of output nodes; the second parameter, activation, is used to specify the activation function; and the third parameter, input_shape, is used to specify the dimension of the input data. The Dense layer uses bias by default. The input_shape parameter is generally used for the first layer of the neural network because subsequent layers of the network default to receiving all the outputs from the previous layer. Adding a Dropout layer is to prevent overfitting of the model. It randomly retains a certain proportion of neurons and passes the remaining ones to the next layer.

The structure information and parameter status of each layer of the model can be output through 'model.summary()'. The first column shows the neural network layer, the second column shows the size of the output dimension of the neural network layer, and the third column shows the number of parameters. The parameters of each layer of the neural network are the total number of parameters generated by the calculation of the neurons in that layer. In the fully connected layer 'Dense', each neuron node has a weight term and uses a bias term by default. Therefore, the calculation formula for the number of parameters 'Param' is: (input data dimension + 1) × number of neurons. For example, in the first 'Dense' layer, the input data dimension is 5 and there are 128 neurons in this layer, so 'Param = (5 + 1) × 128 = 768'. Because the 'Dropout' layer randomly selects 80% of the data from the output of the upper layer neural network to pass to the lower layer neural network, this layer does not generate any computational data, so 'Param = 0'. The code for building the neural network is shown in Listing 6.7. The structure information of this neural network is shown in Fig. 6.14.

Listing 6.7 Building a Neural Network

```
from keras.layers import Sequential,Dense,Dropout
model = Sequential(name='model-1')
model.add(Dense(128, activation='relu', input_shape=(5,)))
model.add(Dropout(0.2))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))
model.summary()
```

```
Model: "model-1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               768

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 16)                2064

 dropout_1 (Dropout)         (None, 16)                0

 dense_2 (Dense)             (None, 4)                 68

=================================================================
Total params: 2,900
Trainable params: 2,900
Non-trainable params: 0
_________________________________________________________________
```

Fig. 6.14 Structure information of neural network model

6.6.4 Set the Model Optimizer, Loss Function, and Evaluation Metrics

Listing 6.8 configures the optimizer, loss function, and evaluation metrics for this neural network. The 'model.compile()' function is used to configure the learning process of the created network model and compile the model. The 'compile()' step is essential; otherwise, an exception will be thrown when running the model. Generally, three parameters need to be set in the 'compile()' function: optimizer, loss, and metrics.

The parameter 'optimizer' can be given as the name of an optimizer in string form, such as 'sgd', 'adagrad', 'adadelta', 'adam', etc.; it can also be specified as an optimizer function provided by the 'tensorflow.keras.optimizers' module. Using the function form allows for setting the learning rate, momentum, and other hyperparameters of the optimizer, such as 'SGD', 'Adagrad', etc.

The parameter 'loss' can be given as the name of a loss function in string form, such as 'categorical_crossentropy', 'mse', 'sparse_categorical_crossentropy', etc.; or it can be specified as a function provided by the 'tensorflow.keras.losses' module, such as 'MeanSquaredError()', 'SparseCategoricalCrossentropy(from_logits = False)', etc.

The parameters specified in 'metrics' are the evaluation metrics of the model. These metrics are used to assess the performance of the current training model. We

can input the required evaluation metrics in the form of strings or functions into the 'metrics' parameter List. If both the true values and the model's predicted values are scalars and we want to evaluate their accuracy, we can use 'accuracy'; if the true values and predicted values are in the form of one-hot encoding or probability distributions and we want to evaluate their accuracy, we can use 'categorical_accuracy'. Commonly used evaluation metrics also include 'mean_squared_error', 'sparse_top_k_categorical_accuracy', 'mean_absolute_error', 'binary_accuracy', 'mean_squared_logarithmic_error', etc.

Listing 6.8 Configuring the Optimizer, Loss Function, and Evaluation Metrics

```
Model.compile(optimizer='adam',
        loss='categorical_crossentropy',
         metrics=['accuracy'])
```

In Listing 6.9, the model.fit() function is used to execute the neural network training. The parameters of the function are the feature set for training X_train, the result set for training Y_train, the size of each batch, and the number of training iterations. The fit() function returns a History object, and the History.history attribute records the values of the loss function and other metrics as the number of epochs changes.

Listing 6.9 Training the Neural Network

```
history = model.fit(X_train, Y_train,
            batch_size=16,
           epochs=100)
```

The loss function and other evaluation metric attributes are stored in the form of a dictionary in History.history. We can first print its attributes through history.history.keys(), and then select the required attributes to draw images. The specific implementation code is shown in Listing 6.10, and the running result is shown in Fig. 6.15.

Listing 6.10 Drawing Images

```
import matplotlib.pyplot as plt
print(history.history.keys())
plt.plot(history.history['loss'], 'b'
plt.plot(history.history['categorical_accuracy'], 'r')
```

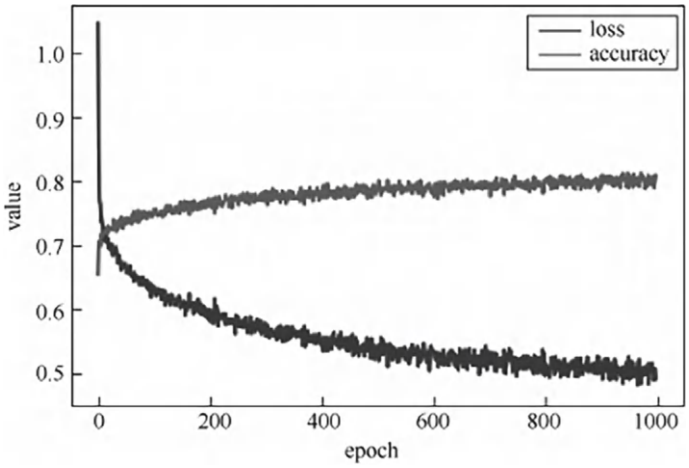


Fig. 6.15 Loss and accuracy changes during the model training process

```
plt.ylabel("value")
plt.xlabel("epoch")
plt.legend(["loss", "accuracy"]).
plt.show()
print('loss----',history.history['loss'][-1])
print('accuracy--',history.history['categorical_
accuracy'][-1])
```

6.6.5 Model Application

From Fig. 6.15, we can clearly observe the changing process of the model's loss value and accuracy value. The training accuracy of this model is approximately 0.8. Next, we can use this model to classify and predict the kinase inhibitors in the test set data. Since the predicted values output by the neural network are in vector form, the coordinate of the maximum value in the vector is the most likely result. Therefore, we traverse the y_predict vector group to obtain the coordinate of the maximum value of each vector and store it in the y_predict_label List. We use the same method to obtain the coordinate of the maximum value of the test set result values in One-Hot form and store it in the y_test_label List. By printing y_predict_label and y_test_label respectively, we can view the details of the result values. The implementation code is shown in Listing 6.11, and the running result is shown in Fig. 6.16.

```
y_predict_label:
 [1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 3, 1, 3, 2, 3, 3, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 2, 3, 1, 1, 1, 1, 1, 1, 1, 1,
2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 3, 1, 1, 1, 2, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 1, 3, 1, 1, 2, 1,
1, 3, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,
1, 1, 3, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1]
y_test_label:
 [1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 2, 1, 1, 1, 1, 2, 3, 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 1,
2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 3, 1, 1, 2, 1, 3, 1, 1, 2, 3, 1, 0, 1, 1, 2, 1, 1, 1, 1,
3, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 3, 1, 1, 1, 2, 1, 1, 2, 1, 1, 3,
3, 1, 1, 2, 1, 2, 1, 1, 3, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 2, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 3, 1, 2, 2, 2, 1, 3, 2, 2, 1, 1, 1, 1, 1, 2, 1,
1, 3, 1, 3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 3, 1, 1, 1, 3, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1,
1, 1, 3, 1, 1, 2, 1, 1, 1, 0, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1]
```

Fig. 6.16 Display of predicted values and actual values

Listing 6.11 Computation of Predicted Values and the Actual Values

```
y_predict = model.predict(X_test)
y_predict_label = [np.argmax(i) for i in y_predict]
y_test_label = [np.argmax(i) for i in Y_test]
print('y_predict_label:\n',y_predict_label)
print('y_test_label:\n',y_test_label)
```

For ease of observation, we can count the frequency of each value in the List and also calculate the accuracy rate of the predictions on the test set. The implementation code is shown in Listing 6.12, and the running results are shown in Fig. 6.17.

Listing 6.12 Statistical Data and Calculation of Prediction Accuracy

```
from collections import Counter
print('predict_Results-',Counter(y_predict_label))
print('y_test_label---',Counter(y_test_label))
```

```
predict_Results-- Counter({1: 166, 3: 17, 2: 17, 0: 2})
y_test_label----- Counter({1: 146, 2: 33, 3: 21, 0: 2})
correct_prediction---- 0.7871287128712872
```

Fig. 6.17 Accuracy of data statistics and model prediction

```
correct_prediction = np.equal(y_predict_label, y_test_
label)
print('correct_prediction--',np.mean(correct_prediction))
```

References

1. Choi, R.Y., Coyner, A.S., Kalpathy-Cramer, J., Chiang, M.F., Campbell, J.P.: Introduction to machine learning, neural networks, and deep learning. Transl. Vis. Sci. Technol. **9**, 14–14 (2020)
2. Popescu, M.-C., Balas, V.E., Perescu-Popescu, L., Mastorakis, N.: Multilayer perceptron and neural networks. WSEAS Trans. Circuits Syst. **8**, 579–588 (2009)
3. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. **79**, 2554–2558 (1982)
4. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)
5. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
6. Kriegeskorte, N.: Deep neural networks: a new framework for modeling biological vision and brain information processing. Annu. Rev. Vis. Sci. **1**, 417–446 (2015)
7. Rall, W.: Electrophysiology of a dendritic neuron model. Biophys. J. **2**, 145 (1962)
8. Zou, J., Han, Y., So, S.-S.: Overview of artificial neural networks. Artif. Neural Netw.: Methods Appl. 14–22 (2009)
9. Bebis, G., Georgiopoulos, M.: Feed-forward neural networks. IEEE Potentials **13**, 27–31 (1994)
10. Berkson, J.: Application of the logistic function to bio-assay. J. Am. Stat. Assoc. **39**, 357–365 (1944)
11. Yarotsky, D.: Error bounds for approximations with deep ReLU networks. Neural Netw. **94**, 103–114 (2017)
12. Hussain, M.A.: Review of the applications of neural networks in chemical process control—simulation and online implementation. Artif. Intell. Eng. **13**, 55–68 (1999)

Chapter 7
Convolutional Neural Networks

Convolutional neural networks (CNNs), also referred to as convolutional networks, are a class of deep feedforward neural networks. In recent years, many advances in deep learning—particularly in computer vision and image processing—have been driven in part by the widespread adoption of CNN-based architectures and the continued development of related training techniques [1].

In this chapter, we introduce CNNs in a structured manner following their architectural components. We also provide several supplementary sections to highlight key concepts and practical considerations. In addition to the theoretical discussion, we include illustrative code examples to help readers better understand and apply convolutional neural networks.

7.1 The Structure of Convolutional Neural Networks

7.1.1 Convolutional Layer

In modern artificial intelligence, CNNs have become indispensable. The core computation in a CNN is the convolution operation, which is performed using a convolution kernel (also called a filter) [2]. The convolution kernel is a small window that translates on the input matrix to extract features by means of convolution operations.

For two-dimentionsl convolution, the convolution kernel can generally be viewed as an N × N matrix. As shown in Fig. 7.1, a 3 × 3 convolution kernel is defined as follows:

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_7

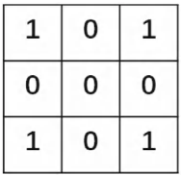


Fig. 7.1 3×3 convolution kernel

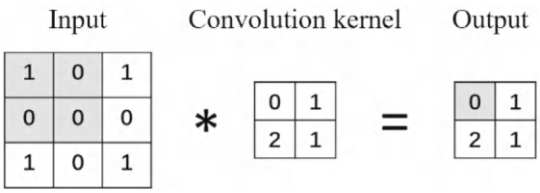


Fig. 7.2 Two-dimensional convolution operation

Note
To facilitate kernel alignment during convolution, the kernel size is typically chosen to be an odd number.

From a spatial perspective, two-dimensional convolution can be regarded as a single-layer (single-channel) convolution. To provide an intuitive understanding of 2D convolution, a concrete example is used to illustrate the operation. As shown in Fig. 7.2. the input is a 3×3 two-dimensional array and the convolution kernel is of size 2×2.

The convolution kernel slides over the input matrix from left to right and from top to bottom, with a stride of 1 at each step, as shown above. At each position, the kernel covers four elements of the input matrix. Each kernel element is multiplied by the corresponding input element, and the products are summed to produce a single value, which is written to the corresponding location in the output matrix.

Note
The stride refers to the step size by which the convolution kernel moves over the input matrix; the strides along the horizontal and vertical directions can be set independently. In convolution, the default stride is 1, meaning the kernel shifts by one unit at each step. In pooling, the default stride is typically set equal to the pooling window size.

The example operation process of two-dimensional convolution is as follows:

$$1 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 1 = 0,$$
$$0 \times 0 + 1 \times 1 + 0 \times 2 + 0 \times 1 = 1,$$

$$0 \times 0 + 0 \times 1 + 1 \times 2 + 0 \times 1 = 2$$
$$0 \times 0 + 0 \times 1 + 0 \times 2 + 1 \times 1 = 1.$$

As shown in the above computation, the four results correspond one-to-one to the four elements of the output matrix. In this example, the convolution operation produces an output matrix of size 2×2.

Note
The convolution operation is denoted by "*", and " $\times$ " is used only to indicate array dimensions.

The code example of two-dimensional convolution operation is shown in Listing 7.1:

Listing 7.1 Code for Two-Dimensional Convolution Operation

```
34. tf.keras.layers.Conv2D(
35.   filters,
36.   kernel_size,
37.   strides=(1, 1),
38.   padding='valid',
39.   data_format = None,
40.   dilation_rate=(1, 1),
41.   activation = None
41.)
```

The main variables in this code segment are explained as follows.

- *Filters*: The number of convolutional kernels.
- Kernel_size: The size of the convolution kernel. If it is a square, it can be represented by a single integer; if it is a rectangle, the height should be indicated by 'h' and the width by 'w', and both can be expressed in the form of a tuple or list (e.g., [h, w]).
- strides: stride. By default, the horizontal and vertical sliding strides are both 1. Other strides can also be set, for example, the vertical or horizontal stride can be set to (1, 1).
- padding: padding. When padding = 'same', zero padding is applied; when padding = 'valid', no padding is needed. The case is not sensitive and will be detailed in Sect. 7.2.3.
- data_format: A string indicating the order of input dimensions. The data format supports two types: channels_last (default) and channels_first. channels_last corresponds to the input format of (batch, height, width, channels), while channels_first corresponds to the input format of (batch, channels, height, width).

- dilation_rate: The dilation coefficient of the convolution kernel. The shape of the convolution kernel is dilated, and the new positions are filled with 0. This will be elaborated in Sect. 7.2.4.
- activation: activation function. It is equivalent to applying an activation function (common ones include ReLU, Softmax, and Sigmoid) after the convolution output. This will be detailed in Sect. 7.1.5.

7.1.2 Multidimensional Convolution Operation

In the previous section, we introduced the convolution operation of two-dimensional arrays. In images, a two-dimensional array with a single channel is called a grayscale image. However, in actual image processing, the dimensions of the array can be higher. For instance, a color image has three color channels: RGB (red, green, and blue). If the height of the color image is h and the width is w, it means that the RGB color image can be represented as a multi-dimensional array of $h \times w \times 3$. This section will introduce the operation of multi-dimensional convolution in three parts: the conventional operation of multi-dimensional convolution, Depth-wise convolution, and Pointwise convolution.

(1) Conventional multi-dimensional convolution.

If the input of a two-dimensional convolution is a single channel, then the output is also a single channel. The input of the convolution operation of multi-dimensional arrays is multi-channel, but the output may be either single-channel or multi-channel. When performing the conventional operation of multi-dimensional convolution, the input array is multi-channel but the output array is single-channel. When performing convolution operations on multi-dimensional arrays, we need a convolution kernel with the same number of channels as the input array to complete the corresponding convolution operation. The shape size of a multi-dimensional array can be expressed as $h \times w \times c$, where h represents height, w represents width, and c represents the number of channels [1].

As shown in Fig. 7.3, suppose the input array is a $3 \times 3 \times 3$ multi-dimensional array, then the size of the convolution kernel can be a $2 \times 2 \times 3$ multi-dimensional array. It has the same number of channels as the input, enabling multi-channel convolution. At each sliding position, the convolution is performed independently within each channel between the input patch and the corresponding kernel slice, following the same procedure as in 2D convolution. The per-channel outputs are then summed element-wise across channels, yielding a single-channel two-dimensional output feature map.

The regular operation process of multi-dimensional convolution is roughly the same as that of two-dimensional convolution, and thus will not be elaborated here. The final output result requires adding the elements at the corresponding positions of the two-dimensional arrays output by each channel one by one. The operation process of the multi-dimensional array output result is as follows.

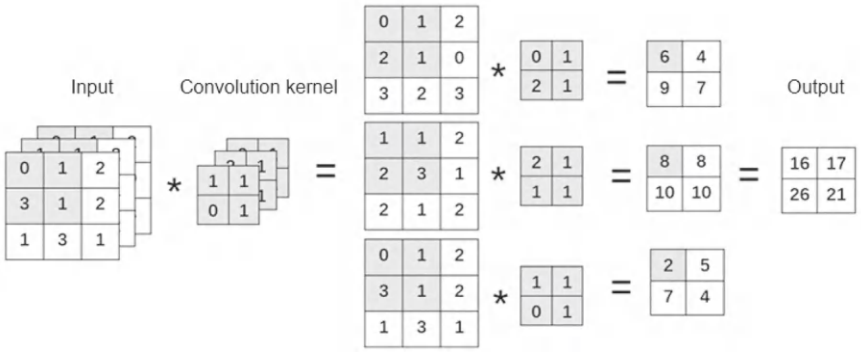


Fig. 7.3 Conventional multi-dimensional convolution

$$\text{Position}(1,1): 6+8+2=16$$
$$\text{Position}(1,2): 4+8+5=17$$
$$\text{Position}(2,1): 9+10+7=26$$
$$\text{Position}(2,2): 7+10+4=21$$

The operation process of multi-dimensional convolution: For a single convolution kernel, there are 3 channels. The convolution kernel of each channel convolves with the input array of the corresponding channel to obtain 3 different convolution results. Then, these 3 convolution results are superimposed at the corresponding positions of the elements to generate a single-channel two-dimensional array. This convolution result then passes through an activation function to obtain the final convolution result. At this point, a 3-channel input array, after convolution operation with a 3-channel convolution kernel, has obtained a single-channel feature map.

Note

1. In the convolution operation of multidimensional arrays, the number of channels of the convolution kernel must be consistent with the input channel.
2. Different convolution kernels implement different functions. You can specify the convolution kernel yourself and perform convolution on the original image. In real convolutional neural networks, there is no need for manual settings, but through machine learning methods, the convolution kernel is found by itself through the back-propagation gradient descent method, which also reflects the power of convolutional neural networks.

The code example of multi-dimensional convolution operation is shown in Listing 7.2:

Listing 7.2 Code for Multidimensional Convolution

```
43. tf.keras.layers.Conv2D(
44.   filters,
45.   kernel_size,
46.   strides=(1, 1),
47.   padding='valid',
48.   data_format = None,
49.   dilation_rate=(1, 1),
50.   activation = None
51. )
```

Note
It should be noted that the convolution operation here is a multi-dimensional (multi-channel) convolution, which should be distinguished from the single-channel convolution described in the previous section. When setting the convolution-kernel parameter kernel_size in Code Listing 8.2, the number of channels involved in the multi-channel convolution should be taken into account.

(2) Depth-wise convolution.

Conventional multi-channel convolution takes a multi-channel input and produces a single-channel output. Depth-wise convolution differs from conventional convolution in that it takes a multi-channel input and produces a multi-channel output.

As shown in Fig. 7.4, suppose a high-dimensional array with a shape of $3 \times 3 \times 3$ and an input of three channels is subjected to a Depth-wise convolution operation with a kernel size of $2 \times 2 \times 3$. Unlike conventional convolution operations, the Depth-wise convolution operation outputs an array with the same number of channels as the input array. It outputs the result of each channel separately, so a three-channel image after a Depth-wise convolution operation generates a three-channel feature map [2].

The number of channels in the output result of the Depth-wise convolution is the same as the number of channels in the input array. It outputs the result of each channel separately without integrating the feature information on each channel. The feature information of the output result is not as good as that of the conventional convolution output result.

(3) Point-wise convolution.

Point-wise convolution. The operation of Point-wise convolution is very similar to the regular operation of multi-dimensional convolution. Its convolution kernel size

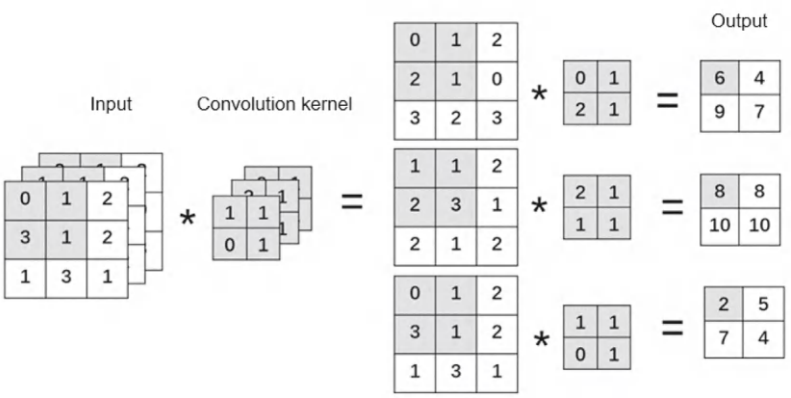


Fig. 7.4 Depth-wise convolution operation

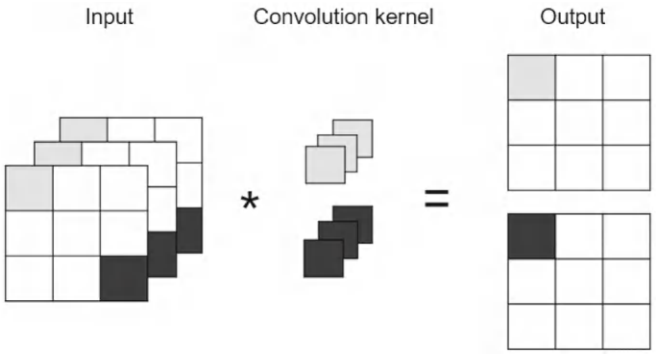


Fig. 7.5 Point-wise convolution operation

is $1 \times 1 \times C$, where C is the number of channels in the input array. The final output array needs to superimpose the results of each channel to generate a new feature map.

Figure 7.5 shows the operation process of Point-wise convolution. The specific operation process is no different from that of the conventional convolution operation. When generating a new feature map, one convolution kernel produces one new feature map.

From the above procedure, it can be seen that Point-wise convolution is similar to standard convolution in form. Point-wise convolution can be applied after Depth-wise convolution to effectively compensate for the limitation that Depth-wise convolution does not mix information across channels at the same spatial location. Specifically, it performs cross-channel feature fusion on the Depth-wise outputs to generate a new feature map [3].

Note

1. During the convolution operation, the number of new feature maps output is equal to the number of convolution kernels (note the difference between the number of convolution kernel channels and the number of convolution kernels).
2. The shape of the convolution kernel is: $1 \times 1 \times$ number of input channels $\times$ number of output feature maps.

The Role of Point-Wise Convolution

(1) Increase nonlinearity: The convolution process of the convolution kernel of point-by-point convolution is equivalent to the calculation process of the fully connected layer, and a nonlinear activation function is also added, which can increase the nonlinearity of the network and enable the network to express more complex features.
(2) Feature dimensionality reduction: The convolution kernel controls the number of channels, thereby reducing the number of parameters in the network and reducing the computational burden.

7.1.3 Pooling Layer

After convolution, the number of parameters is too large. To address this, we need to reduce the dimension of the data and compress the parameters to prevent overfitting. Thus, the concept of "Pooling" is introduced. The Pooling layer, also known as the down-sampling layer or subsampling layer, compresses and extracts the features generated by the convolutional layer. While reducing the number of parameters and the computational burden, it also aims to extract the main features as much as possible.

The Pooling layer can reduce the number of parameters in the network after the pooling operation, prevent overfitting, and bring translation invariance to the convolutional neural network [3]. Translation invariance means that when the input array is slightly translated and then undergoes the pooling operation, most of the data in the output array does not change, and almost the same result can be obtained. In short, the pooling layer can not only reduce the number of parameters but also retain the important feature information of the input and maximize the output of features. Pooling not only improves the computing speed of the neural network but also enhances the robustness of feature extraction [4].

Pooling operations include average pooling, max pooling, and stochastic pooling, etc. [5]. A significant role of the pooling layer is to reduce the size of the feature map, thereby reducing the computational load and the required video memory. Among

them, average pooling and max pooling are the most commonly used pooling operations at present. Next, we will introduce these three pooling operations one by one.

(1) Average pooling.

Pooling is analogous to convolution in that it applies a function to the input using a sliding window. During each pooling step, a pooling window (i.e., a local matrix window) slides over the input feature map, and the pooling function computes an output for each window position. Unlike convolution, pooling has no learnable weights: convolution depends on the specific values of the kernel elements, whereas pooling summarizes the values within the window using a predefined aggregation rule. Based on the aggregation rule, pooling operations are commonly categorized as average pooling, max pooling, and stochastic pooling. These correspond to taking the mean value, the maximum value, or sampling one value from the window according to a probability distribution defined by the activations, respectively.

The definition formula of average pooling is:

$$y_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} x_{kpq} \tag{7.1}$$

Among them, y_{kij} represents the average pooling output value in the rectangular area R_{ij} related to the kth feature map, x_{kpq} represents the element located at (p, q) in the rectangular area R_{ij}, and $|R_{ij}|$ represents the number of elements in the rectangular area R_{ij}.

Figure 7.6 illustrates the computation procedure for average pooling. with a pooling window size of 2×2 and a stride of 2. On the right is the feature map generated after the pooling operation. Each element in the feature map corresponds to the average value of the elements in the pooling window on the left. For example, the "1" in the feature map is the average value of the elements in the first pooling window on the left.

The specific calculation process is as follows:

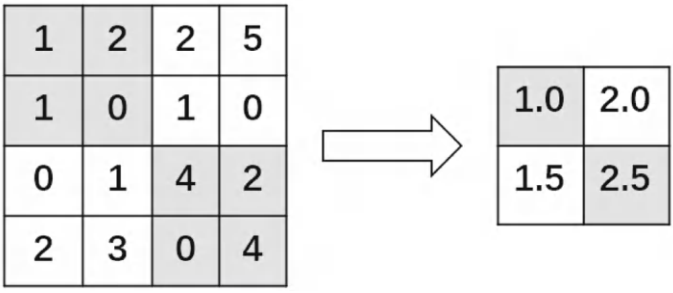


Fig. 7.6 Average pooling

$$(1+2+1+0) \div 4 = 1,$$
$$(2+5+1+0) \div 4 = 2,$$
$$(0+1+2+3) \div 4 = 1.5,$$
$$(4+2+0+4) \div 4 = 2.5.$$

From the above operation process, it can be seen that the elements in the feature map are all the average values of the elements in the pooling window, and they correspond one-to-one.

Features
It can preserve the background well, but it can easily make the picture blurry.

The code example for average pooling is shown in Listing 7.3.

Listing 7.3 Average Pooling Code

```
52. tf.keras.layers.AveragePooling2D (
53.   pool_size = (2, 2),
54.   strides = None,
55.   padding = 'valid',
56.   data_format = (None, **kwargs)
57.)
```

Tips
In Listing 7.3, the assignment of the four parameters in the function for average pooling are all default values, which means that it can run without changing any of the parameters. It should also be noted that when performing pooling operations, the default value of the stride for the pooling window's translation is equal to the size of the pooling window.

(2) Max pooling.

As shown in Fig. 7.7, the calculation process of max pooling is illustrated, with a pooling window size of 2×2 and a stride of 2. The elements in the feature map on the right are the maximum values of the elements within the window each time the window is shifted.

The definition formula of max pooling is:

$$y_{kij} = \max_{(p,q) \in R_{ij}} x_{kpq} \tag{7.2}$$

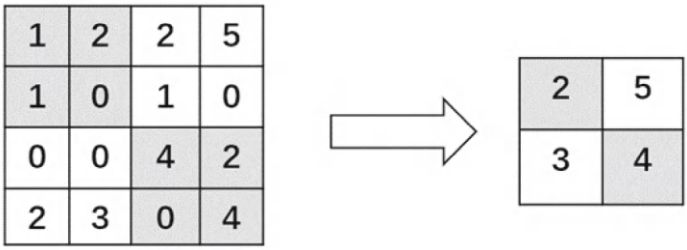


Fig. 7.7 The calculation process of max pooling

Here, y_{kij} represents the maximum pooling output value related to the kth feature map within the rectangular region R_{ij}, and x_{kpq} represents the element located at (p, q) within the rectangular region R_{ij}.

Features
It can preserve texture features very well. Generally, maximum pooling is used and average pooling is rarely used.

The code example of max pooling is shown in Listing 7.4.

Listing 7.4 Example of Max Pooling

```
58. tf.keras.layers.MaxPool2D(
59.   pool_size = (2, 2),
60.   strides = None,
61.   padding = 'valid',
62.  data_format = (None, **kwargs)
63.)
```

(3) Stochastic pooling

It only requires randomly selecting elements in the feature map based on their probability values, that is, elements with larger values have a higher probability of being selected, but other non-zero elements also have the possibility of being selected. Max pooling always selects only the element with the maximum value. The value selection method of stochastic pooling greatly improves the generalization ability.

The operation process of stochastic pooling is shown in Fig. 7.8. First, sum up the elements in the pooling window, then divide each element by the sum of the elements to obtain the probability array; randomly select an element at any position in the pooling window according to the probability; the value of the feature map finally obtained by stochastic pooling is the value at the selected position.

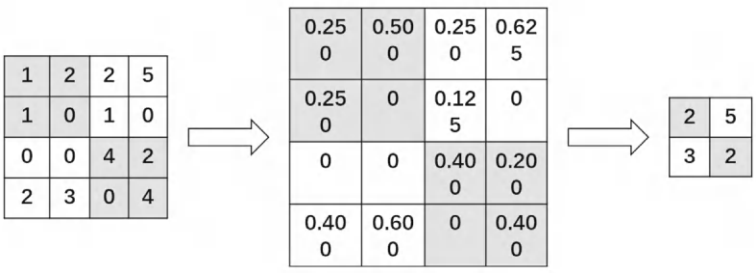


Fig. 7.8 The calculation process of stochastic pooling

Let's take the first operation of the pooling window as an example to introduce the specific calculation process of stochastic pooling.

$$\begin{aligned}&\text{Sum}: 1+2+1+0=4;\\&1 \div 4 = 0.25;\\&2 \div 4 = 0.5;\\&1 \div 4 = 0.25;\\&0 \div 4 = 0;\end{aligned}$$

The definition formula of stochastic pooling is

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k} \tag{7.3}$$

Features
The method is simple and highly random, but it has stronger generalization ability.

7.1.4 Unpooling

In this section, we mainly introduce unpooling, which involves two types: average unpooling and max unpooling.

(1) Average unpooling

Average pooling computes the mean of all elements within each pooling window. Average unpooling [6], performs the reverse mapping by propagating the pooled

output back to the preceding layer and assigning the pooled mean value to every location within the corresponding pooling window. Under this definition, the gradients are distributed uniformly across the window during backpropagation, so that the sum of gradients within each pooling region is preserved. As shown in Fig. 7.9, average unpooling assigns the same value to all positions in the pooling window associated with a given pooled output, making the redistribution process intuitive.

(2) Max unpooling.

The max pooling operation takes the maximum value in the pooling window and discards the other values, passing the maximum value to the next layer. max unpooling is the reverse process of max pooling, where the result of max pooling is backpropagated to the previous layer, and the maximum value obtained by max pooling is passed to the position before pooling, while all other positions are filled with "0". The difference between max unpooling and average unpooling lies in the need to record the position of the maximum element after pooling to ensure the correctness of backpropagation during max unpooling. As shown in Fig. 7.10, the position of the maximum element in the max unpooling array is the same as the initial input position, and "0" is used to fill the other positions.

Tips

(1) Pooling operation does not require parameter learning. There is no gradient calculation in the training of the neural network. The pooling layer only needs to pass the residual term.

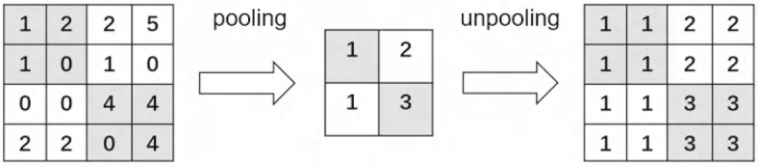


Fig. 7.9 The calculation process of average unpooling

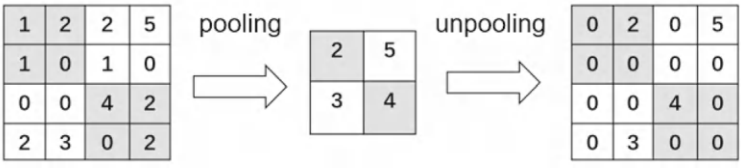


Fig. 7.10 The calculation process of max unpooling

(2) Maximum pooling needs to record the position information of the maximum element value, so that the maximum element is filled in the corresponding position during the back propagation of the maximum pooling, and the positions of the remaining elements are filled with "0"; average pooling, during the back propagation, the average value is filled in the positions of all elements in the pooling window.

7.1.5 Activation Function Layer

The activation function refers to the functional relationship between the output of the upper-layer neurons and the input of the following-layer neurons in a multi-layer neural network [7]. This function is the activation function. The activation function enables the model to output in a non-linear manner rather than in a single linear way. A single linear combination of output and input is no different from a traditional perceptron model. Adding an activation function allows the model to output in a non-linear combination, solving more complex problems and enhancing the model's expressive power.

The commonly used activation functions are the same as those in feedforward neural networks, such as ReLU, Softmax, Sigmoid, etc.

The ReLU function, short for Rectified Linear Unit, is one of the commonly used activation functions in neural networks. The ReLU function is piecewise linear. When the input is positive, the input and output are the same; when the input is negative, the output is always "0".

Features

Advantages:

(1) For the region where the function is positive, the output is linear (not constant) and the gradient is always 1; therefore, it does not suffer from the vanishing-gradient problem

(2) Compared with sigmoid, ReLU is faster in calculation.

Disadvantages:

(1) When the function is less than 0, the gradient is 0, and the model is no longer trained.

The Softmax activation function is used to address multi-class classification problems. It can predict the probabilities of multiple categories and select the category with the highest predicted value when determining the prediction result. The Softmax function normalizes the output values, making them fall within the range of (0, 1).

Features
Disadvantages:

1. The zero point is not differentiable
2. When the input is negative and the gradient is zero, the weights are no longer updated during back propagation.

The Sigmoid function, also known as the Logistic function, returns values within the range (0, 1). When the input value is large, the Sigmoid function returns a value close to 1; when the input value is small, the returned value is close to 0. The Sigmoid function is also used to solve binary classification problems.

Features

Advantages: smooth and easy to differentiate.
Disadvantages:

(1) It works better when the feature differences are complex or not very large
(2) However, when the input approaches positive or negative infinity, the gradient approaches 0, causing gradient dispersion and making it impossible to train the neural network.

7.1.6 Fully Connected Layer

The Fully Connected (FC) layer is located at the very last layer of the entire neural network [8]. After passing through the convolutional kernels and pooling, the neural network reaches this layer. After the convolutional operations, the important features of the input array have been extracted. After being connected to the fully connected neural network, the fully connected layer can classify the extracted features. It can also be said that the fully connected layer is the "classifier" of the convolutional neural network. Before connecting to the fully connected neural network, the features generated by the previous layer need to be flattened into a one-dimensional vector and then fully connected to the input layer of the fully connected neural network. Each layer of the fully connected neural network is connected to all the neurons of the previous layer, eventually forming a dense connection. When outputting at the end of the fully connected network, an activation function should also be added. The input layer receives the features from the previous layer, and the output layer outputs the predicted results.

The Schematic diagram of a fully connected layer is shown in Fig. 7.11. The first layer of a fully connected neural network is the input layer, and the last layer is the

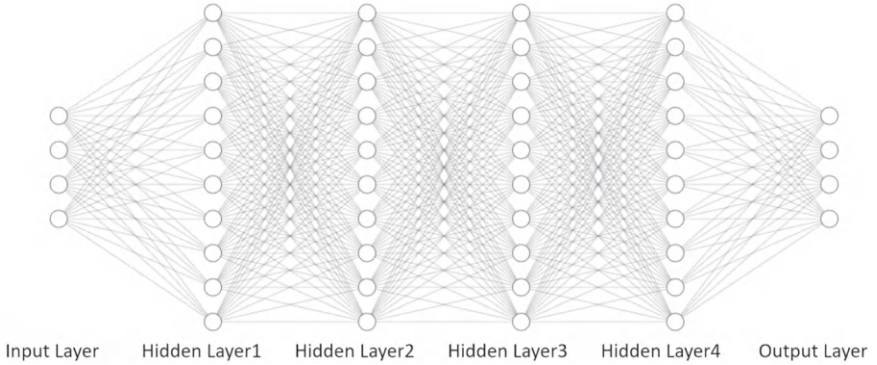


Fig. 7.11 Schematic diagram of fully connected layer (In the figure, there are 6 fully connected layers, including 1 input layer, 1 output layer, and 4 hidden layers. The number of layers can be set according to the model requirements.)

output layer. The layers in between are all hidden layers, among which there can be many hidden layers. The number of layers (length) and the number of neurons in each layer (width) of a fully connected network can be set according to the model's requirements. The choice of activation function, the length and width of the fully connected layer all have a significant impact on the model. Whether it is to increase the length while keeping the width constant or to increase the width while keeping the length constant, it can enhance the complexity and learning ability of the model. However, excessively increasing the length and width of the fully connected layer will lead to overfitting. The increase in the number of neurons means a sharp increase in data volume, which ultimately reduces the operational efficiency.

7.2 Relevant Computations of Convolutional Neural Networks

7.2.1 Feature Map

After the input array undergoes convolution operation, the resulting output array is called a feature map. Assuming the shape of the input array is m $\times$ m $\times$ 1 and the shape of the convolution kernel is $n \times n \times 1$, then the feature map will have a shape of $(m - n + 1) \times (m - n + 1) \times 1$.

Tips

The 1 in $m \times m \times 1$ is the number of layers of the input array, which is also called the number of channels in image processing.

$$\begin{cases} height_{out} = \frac{height_{in} - height_{kernel} + 2*padding}{stride} + 1 \\ width_{out} = \frac{width_{in} - width_{kernel} + 2*padding}{stride} + 1 \end{cases} \tag{7.4}$$

Here, $height_{out}$ and $width_{out}$ respectively represent the height and width of the output feature map, $height_{in}$ and $width_{in}$ respectively represent the height and width of the input array, $height_{kernel}$ and $width_{kernel}$ are the height and width of the convolution kernel, padding is the number of peripheral layers for padding, and stride is the step size.

Tips

(1) The above formula assumes that stride is 1 and padding is 0 (padding will be introduced in detail in Sect. 7.9).
(2) Convolution operations are rounded down, and pooling operations are rounded up.

The general format for the feature image code part is [*batch_size, height, width, channels*], which is represented as [*b, h, w, c*]. Here, "*batch_size*" refers to the batch size, indicating the number of tensors in a batch; height represents the number of rows; width represents the number of columns; and channels represent the number of channels. For instance, [*b, h, w, 1*] represents a single-channel grayscale image; [*b, h, w, 3*] represents an RGB image with three channels; [*b, h, w, c*], where c $\geq$ 4, abstracts the unknown dimension.

7.2.2 Receptive Field

The receptive field refers to the spatial extent of the original image that influences a given element in the feature map produced by a convolutional neural network [9]. CNNs exploit local connectivity: each neuron responds primarily to patterns within a local neighborhood, analogous to how humans perceive the external world through limited local fields of view. As information from local regions is progressively integrated across layers, these local features can be combined to form a more global representation. For example, after repeatedly observing an object, one can recognize it from a particular characteristic; repeated observation corresponds to accumulating multiple local cues, which are then integrated to infer the object's overall identity.

As shown in Fig. 7.12, in the convolutional neural network, the input array size is 5 $\times$ 5, and the size of the convolution kernels is all 3 $\times$ 3, with a stride of 1. Layer1 is the input array of size 5 $\times$ 5, where the gray area represents the size of the convolution kernel; in layer2, the size of the black bold-bordered area is the size of the new feature map generated after the convolution operation on layer1; in layer3, the black area represents the size of the feature map generated after the convolution operation on layer2. Each neuron in layer2 can see a 3 $\times$ 3 area on layer1, and each

neuron in layer3 can see a 3×3 area on layer2, and the area in layer3 can also see a 5×5 area on layer1.

The receptive field size of the output feature map of the first convolutional layer is equal to the size of the filter (when dilation = 1). The formula for calculating the receptive field is:

$$RF_i = (RF_{i+1} - 1) * stride_i + Ksize_i \tag{7.5}$$

Here, RF_i represents the receptive field of the i-th layer, RF_{i+1} represents the receptive field of the $(i+1)$-th layer, stride is the convolutional step size, and $Ksize_i$ is the size of the convolution kernel of this layer.

Tips

(1) Replacing large convolution kernels with smaller ones can reduce the number of parameters, increase network depth, and expand the receptive field. In general, deeper networks tend to have larger receptive fields and can achieve better performance.
(2) For classification tasks, the receptive field of the final-layer feature map should be at least as large as the input image; otherwise, the classification performance may be suboptimal.
(3) For object detection tasks, a mismatch between the receptive field and the target scale—such as a small receptive field for a large object, or an excessively large receptive field for a small object—can make model convergence difficult and severely degrade detection performance.
(4) The calculation of the receptive field size does not take into account the size of padding.

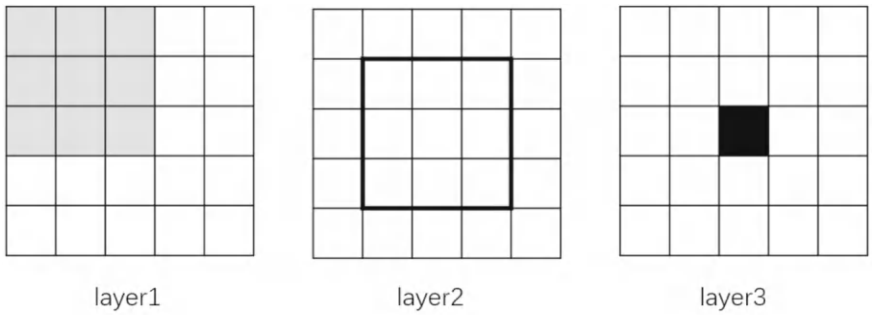


Fig. 7.12 Receptive field

7.2.3 Padding

In the previous chapters, we can observe that after the convolution operation between the input array and the convolution kernel, a phenomenon of numerical loss occurs. That is, only partial elements at the edges of the input array are detected, and a lot of information is lost. This is because the outermost elements of the array can never intersect with the center of the convolution kernel. During the convolution operation, they can only be calculated once, and the convolution kernel cannot extend beyond the matrix.

This process can lead to insufficient feature extraction, which affects the final training results. Therefore, sometimes we hope that the sizes of the input and output remain the same [10]. To solve this problem, before the convolution operation, the edges of the input array are padded with numerical values to ensure that the outermost elements of the array are not marginalized. Usually, the edges are padded with "0" to increase the size of the array, allowing the edge elements to be convolved multiple times and enhancing feature extraction. The size of the padding depends on the specific situation.

Figure 7.13 illustrates an example of edge padding. It can be clearly seen that the original matrix is padded with zeros along its boundary.

As shown in Fig. 7.14, there are three padding modes: Full, Same, and Valid.

(1) Full mode. As shown in Fig. 7.14a, in the first step of the convolution operation, there is only one input element in the convolution kernel, that is, the convolution operation starts when the convolution kernel just intersects with the array. The remaining parts that have not intersected are padded with all "0"s around the input array until the remaining elements in the convolution kernel intersect with "0".

(2) Same mode. As shown in Fig. 7.14b, the convolution operation begins when the center element of the convolution kernel intersects with the first element of the input array. At this point, the periphery of the input array is fully padded with "0" until the remaining elements of the convolution kernel intersect with "0".

(3) Valid mode. As shown in Fig. 7.14c, the convolution operation begins when one element of the convolution kernel intersects with the first element of the input array. At this point, all elements within the convolution kernel intersect with all elements of the input array. There is no space for full "0" padding, and thus no padding is required.

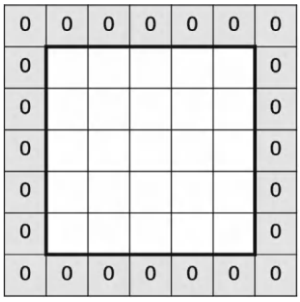


Fig. 7.13 Padding

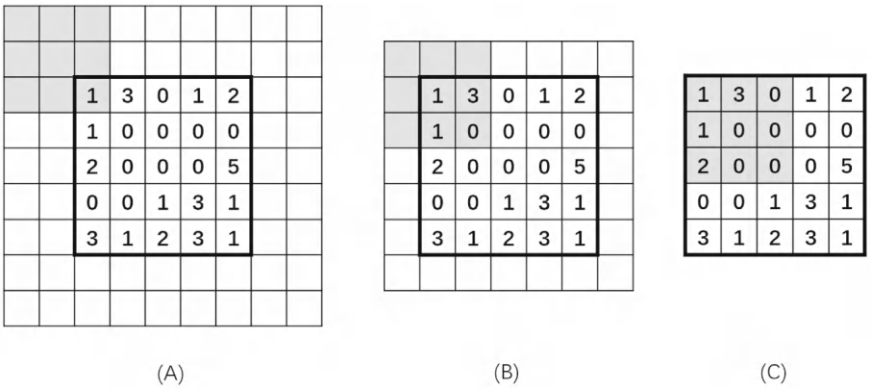


Fig. 7.14 Three padding modes

Tips

While convolution reduces the computational burden, it also produces some defects:

(1) The image will shrink after the convolution operation. After several convolution operations, the image and its features may shrink.
(2) In the convolution operation, the number of pixels covering the edges and corners is less than that of the middle pixels, resulting in the loss of image edge information.

The purpose of full "0" padding is to ensure that the size of the output feature can remain the same as the size of the input feature, mainly to prevent edge features from being ignored.

7.2.4 Dilated Convolution

Dilated convolution involves expanding the convolution kernel, with the additional gaps filled with "0" [11]. This is used to overcome the distortion caused by the stride. The advantage of dilated convolution is that it increases the receptive field without losing information through pooling operations, allowing each convolution to cover a larger area.

Figure 7.15 shows the dilation operation performed on 2×2 and 3×3 convolution kernels respectively.

The calculation formula of dilated convolution is:

$$k_d = (k - 1) \times d + 1 \tag{7.6}$$

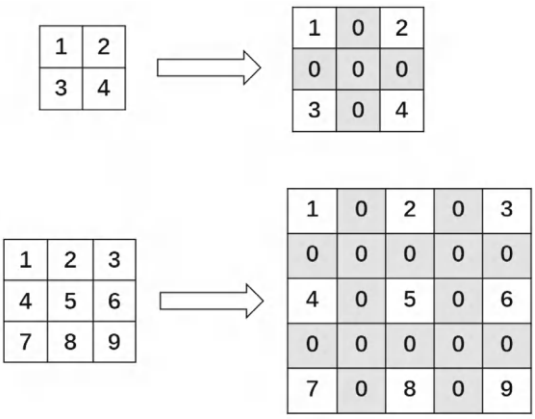


Fig. 7.15 The operation of dilated convolution

Here, k_d represent the dimensions of the dilated convolution, where k is the size of the original convolution kernel and d is the dilation factor. After dilating the original convolution kernel by a factor of d, a new kernel of size k_d is obtained. Generally, k is an odd number and d is an even number, ensuring that k_d is also odd. When using dilated convolution, the original convolution kernel is first dilated by a factor of d, and then the convolution operation is performed to generate the feature map of the next layer.

Tips

When the expansion coefficient is 0, the convolution kernel is a 1×1 matrix, and padding operations cannot be performed at this time. The default value of the expansion coefficient d is 1, and the convolution kernel is an n*n matrix at this time. The size of the expanded convolution kernel can only be less than or equal to the size of the input matrix, otherwise the convolution calculation cannot be performed. Therefore, when setting the expansion coefficient d, you need to calculate the k_d size in advance.

7.3 Example: Predicting the Properties of Drug Molecules Using CNN

The prediction of drug molecule properties is a very active research field in artificial intelligence-assisted drug screening [12]. The diversity of drug molecule structures makes it difficult for us to establish relevant empirical formulas through ordinary mathematical methods to calculate the properties of drug molecules [13]. Neural networks have strong fitting capabilities and can discover the nonlinear relationship

between molecular structure and properties. In this section, we will introduce how to use convolutional neural networks to predict the properties of drug molecules.

Lipophilicity is an important parameter for small-molecule drugs and is commonly quantified by logP. logP is the logarithm of a compound's octanol–water partition coefficient P. The membrane-water partition coefficient characterizes the ratio of the solubility of a drug in the cell membrane and in aqueous solution. Only by passing through the cell membrane can the ingested drug reach the interior of the cell. Therefore, logP is an important factor to be considered in drug design. We will use the Lipophilicity dataset in the MoleculeNet dataset and a convolutional neural network to predict the logP of drug molecules.

First, import the required dataset. The code is shown in Listing 7.5.

Listing 7.5 Importing the Lipophilicity Dataset

```
import pandas as pd
import tensorflow as tf
from keras import backend as K
from keras.models import Sequential

from sklearn.model_selection import train_test_split

from keras.constraints import maxnorm

# Importing the dataset
dataframe = pd.read_csv("/home/yons/xielx/AlogP/
Lipophilicity/ecfp-Lipop.csv", delimiter = ',', header=None)

dataset = dataframe.values
# split into input (X) and output (Y) variables
X = dataset[:,:]
Y = np.loadtxt("/home/yons/xielx/AlogP/Lipophilicity/Lipop_
logP.value")

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
X_train = X_train.reshape(-1, 2048, 1)
X_test = X_test.reshape(-1, 2048, 1)
```

Then, a convolutional neural network is constructed to predict logP, as shown in Listing 7.6.

Listing 7.6 Convolutional Neural Network Predicting logP

```
# Build the model
```

```
model = Sequential()
# Convolutional layer 1
model.add(Conv1D(32,    3,    activation='relu',    input_
shape=(2048, 1), padding="same"))
# Pooling layer 1
model.add(MaxPool1D(pool_size=3, strides=1))
# Convolutional layer 2
model.add(Conv1D(64,   3,   strides=1,   activation='relu',
padding='same'))
# Pooling layer 2
model.add(MaxPool1D(pool_size=3, strides=1))
# Random neuron inactivation
model.add(Dropout(0.25))
# Pull into one-dimensional data
model.add(Flatten())
# Fully connected layer 1
model.add(Dense(1024))
# Activate Layer
model.add(Dense(1))

# View the defined model
model.summary()

model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mean_absolute_error', 'accuracy'])

# train
history = model.fit(X_train, Y_train, epochs=100, batch_
size=64,
verbose=1, validation_data=[X_test, Y_test] )
```

The prediction results are shown in Fig. 7.16. The average absolute errors on the training set and test set are 0.37 and 0.67. The code is provided for illustrative purposes only, and the specific hyperparameters should be further optimized as needed.

We demonstrate the utility of CNNs by applying them to the prediction of molecular properties, with logP serving as a primary example. Furthermore, the applicability of CNNs extends to a wide range of other challenges in computational chemistry and bioinformatics. A prominent application is the prediction of protein–ligand interactions, which is fundamental to understanding biological processes and for in silico drug discovery. Beyond these examples, CNNs are also employed in tasks such as toxicity prediction, protein structure classification, and the analysis of biological imaging data, underscoring their versatility as a powerful tool for pattern recognition within complex, high-dimensional scientific data.

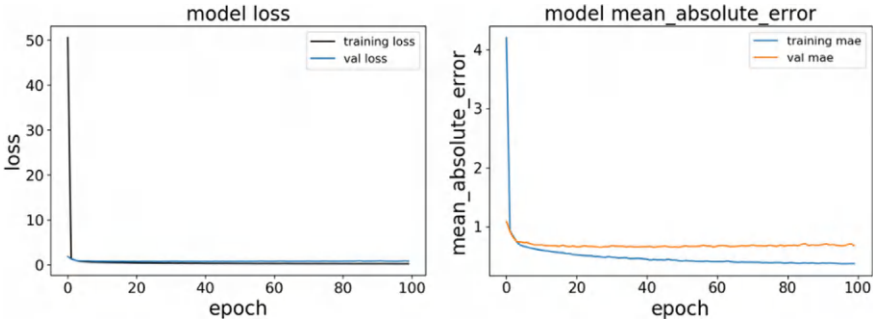


Fig. 7.16 Changes in the loss and mean absolute error during CNN-based logP prediction

References

1. Li, Z., Liu, F., Yang, W., Peng, S., Zhou, J.: A survey of convolutional neural networks: analysis, applications, and prospects. IEEE Trans. Neural Netw. Learn. Syst. **33**, 6999–7019 (2021)
2. Ghiasi-Shirazi, K.: Generalizing the convolution operator in convolutional neural networks. Neural. Process. Lett. **50**, 2627–2646 (2019)
3. Zafar, A., et al.: A comparison of pooling methods for convolutional neural networks. Appl. Sci. **12**, 8643 (2022)
4. Momeny, M., Latif, A.M., Sarram, M.A., Sheikhpour, R., Zhang, Y.D.: A noise robust convolutional neural network for image classification. Results Eng. **10**, 100225 (2021)
5. Gholamalinezhad, H., Khosravi, H.: Pooling methods in deep neural networks, a review. *arXiv preprint* arXiv:2009.07485 (2020)
6. Akhtar, N., Ragavendran, U.: Interpretation of intelligence in CNN-pooling processes: a methodological survey. Neural Comput. Appl. **32**, 879–898 (2020)
7. Karim, M.N., et al.: Global and local neural network models in biotechnology: application to different cultivation processes. J. Ferment. Bioeng. **83**, 1–11 (1997)
8. Basha, S.S., Dubey, S.R., Pulabaigari, V., Mukherjee, S.: Impact of fully connected layers on performance of convolutional neural networks for image classification. Neurocomputing **378**, 112–119 (2020)
9. Araujo, A., Norris, W., Sim, J.: Computing receptive fields of convolutional neural networks. Distill **4**, e21 (2019)
10. Amjad, R.A., Geiger, B.C.: Learning representations for neural network-based classification using the information bottleneck principle. IEEE Trans. Pattern Anal. Mach. Intell. **42**, 2225–2239 (2019)
11. Liu, G., et al.: Partial convolution for padding, inpainting, and image synthesis. IEEE Trans. Pattern Anal. Mach. Intell. **45**, 6096–6110 (2022)
12. Yang, X., Wang, Y., Byrne, R., Schneider, G., Yang, S.: Concepts of artificial intelligence for computer-assisted drug discovery. Chem. Rev. **119**, 10520–10594 (2019)
13. Cohen, N.C., Blaney, J.M., Humblet, C., Gund, P., Barry, D.C.: Molecular modeling software and methods for medicinal chemistry. J. Med. Chem. **33**, 883–894 (1990)

Chapter 8
Generative Deep Learning

This chapter focuses on generative deep learning, with a core emphasis on Generative Adversarial Network (GAN). It first lays the groundwork for deep learning by introducing its three paradigms: unsupervised learning, supervised learning, and reinforcement learning, along with the differences between deep learning and traditional machine learning in terms of data volume, hardware dependence, and other aspects. Subsequently, it introduces GAN, proposed in 2014, elaborating on their dual-module framework consisting of a generator (which produces data resembling real samples) and a discriminator (which distinguishes between real and fake data), as well as the adversarial training mechanism based on Nash equilibrium. Meanwhile, it supplements relevant foundational concepts such as gradient descent and KL divergence, details the iterative training process of GAN—"training the discriminator first, then the generator"—and provides a specific application code example using the MNIST handwritten digit dataset. Finally, it analyzes the advantages of GAN, such as eliminating the need for complex Markov chains and automatically fitting data distributions, as well as their limitations including mode collapse and convergence difficulties; it also introduces derivative models in the "GAN Zoo" and prospects their future expansion into multiple domains and integration with advanced technologies.

8.1 Deep Learning and GAN

This section will first be grounded in the core objectives of artificial intelligence to elucidate the core positioning of deep learning as its key implementation approach. On this basis, it will focus on Generative Adversarial Networks (GANs) and introduce their innovative ideas that break away from traditional generative models.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_8

8.1.1 Deep Learning

Artificial intelligence aims to endow computers with intelligence and enable them to learn independently. Deep learning is one approach to achieving this goal. In many applications of artificial intelligence, we can see the presence of deep learning. Deep learning mainly involves inductive and synthetic work rather than simple deduction [1]. Its most common operation is to use algorithms to analyze data and learn from it to make decisions.

There are three types of deep learning that we are familiar with, namely unsupervised learning, supervised learning, and reinforcement learning [2].

(1) Unsupervised learning. Unsupervised learning refers to the process where, given a specific direction, the algorithm can automatically learn and summarize patterns without the need for us to label the data. While learning and summarizing patterns, the algorithm also categorizes the targets into various classes, which is what we commonly refer to as a "clustering problem".
(2) Supervised learning. Compared with unsupervised learning, in supervised learning, we provide the model with preconceived labels. The model will input corresponding content based on the labels and eventually produce the corresponding output. For instance, to determine whether a potato is still edible based on whether it has sprouted or not, sprouting indicates that the potato is no longer edible, while no sprouting means it can still be eaten. After the model is constructed and trained, by recognizing the shape features of the potato through the model, we can determine whether the potato is edible.
(3) Reinforcement learning. Different from the first two types, reinforcement learning is a learning approach that rewards specific actions to an agent to support their learning, decision-making, and planning. For instance, when educating a child who is reluctant to study, we can offer a reward such as "two hours of study earns half an hour of TV time" to encourage the child to develop the behavior of "studying actively to watch TV". The above example is a process of reinforcement learning, which promotes learning through a feedback mechanism. Reinforcement learning is currently one of the important research directions.

The initial deep learning was a learning process that utilized deep neural networks to address feature representation [3]. In terms of methodology, deep learning mainly adopts multi-layer neural network technology, where the concept of "multi-layer" embodies the "depth" of deep learning. Deep neural networks are a type of neural network and can be understood as a neural network structure with multiple hidden layers. In practical applications, the training effect of the model can be enhanced by adjusting the activation function and the connection mode between neurons. Deep learning mainly explains and analyzes data by creating neural networks that can simulate the human brain for analysis and learning [4].

Although deep learning is a part of machine learning, there are significant differences between the two. Their main application scenarios and goals are different. The main differences between the two lie in six aspects: data volume, hardware

dependence, feature engineering, problem-solving methods, execution time and interpretability.

(1) Data volume. Machine learning can adapt to various data volumes and is particularly adept at scenarios with smaller data volumes. In contrast, deep learning is suitable for scenarios with vast amounts of data.
(2) Hardware dependency. Since deep learning algorithms require a large number of matrix multiplication operations, they need hardware such as Graphics Processing Unit(GPU) for support. Traditional machine learning does not require too high-end hardware facilities and can run on CPUs.
(3) Feature Engineering. Feature engineering is the process of tagging specific domain knowledge with designated features. The purpose of doing this is to reduce the complexity level of data and generate patterns that are easy for learning algorithms to handle. For instance, when dealing with images, machine learning focuses on pixels and other attributes, while deep learning pays attention to other high-level features of the image data.
(4) Problem-solving methods. Traditional machine learning algorithms break down a problem into several parts, solve each part separately, and then combine the results to obtain the desired answer. In contrast, deep learning does not require the problem to be dissected and analyzed; it only conducts centralized processing.
(5) Execution time. Due to the fact that deep learning involves more parameters, it requires a considerable amount of time for training, and thus, the time investment in training is also greater. In contrast, the execution time of machine learning algorithms is relatively shorter.
(6) Interpretability. Interpretability is one of the main differences between machine learning and deep learning algorithms. Many deep learning algorithms lack interpretability.

8.1.2 GAN

In 2014, Ian Goodfellow and others published a paper titled "Generative Adversarial Nets" [5], and since then, GAN have come into the public eye. Since then, people's enthusiasm for researching GANs has been unstoppable, and the number of articles about GANs has been constantly increasing.

GAN is a deep learning model, and its creative generation method is significantly different from that of traditional generative models. The generation results of GAN are produced in a game learning process, while traditional generative models are generated based on artificially set variables. Currently, in addition to being applied to the generation of images, voices, and videos, GAN has also achieved impressive achievements in image translation, image coloring, adversarial sample learning, domain adaptation, and drug molecule generation.

The framework of GAN consists of at least two modules: the generator and the discriminator [6]. The generator is responsible for generating new data that cannot

be distinguished by the discriminator, while the main function of the discriminator is to determine whether the input data comes from real data or from the data generated by the generator. Therefore, the discriminator can be regarded as a binary classifier. The training process of the GAN model belongs to a binary minimax game problem, aiming to make the data distribution generated by the generator as similar as possible to the real data distribution, thereby "confusing" the discriminator. The final result is that the probability of the discriminator identifying the data generated by the generator as fake is 50%.

In the original GAN theory, as long as the corresponding generation and discrimination functions can be fitted, it is fine. Therefore, the generator or discriminator can also be a linear mapping. However, in reality, deep neural networks are generally used as the generator G and the discriminator D.

One notable feature of GAN is that it does not require the manual setting of a desired data distribution. GAN learns the real data distribution through the discriminator D and fits the data range we need. Then, it uses this data range to train our generator. Sometimes, the real data distribution is hard to find, and this step greatly saves the time and labor costs of finding the real data distribution. At the beginning, the discriminator does not fully fit the data distribution we need either, so it needs to be trained and iterated step by step until the best discriminator and generator are obtained.

The following content of this chapter mainly introduces the related concepts, principles and training process of GAN.

8.2 Related Concepts of GAN

Before formally introducing GAN, we will first introduce the concepts related to GAN to help you gain a deeper understanding of it. They are gradient descent, information entropy, KL divergence, Nash equilibrium, Gaussian distribution and Gaussian process.

8.2.1 Gradient Descent Method

We know that machine learning only requires providing a dataset to the model, after which the model will automatically "learn" and continuously optimize its various parameters, eventually obtaining a set of optimal parameters. Under these optimal parameters, the model has the highest matching degree with the task to be learned. This "learning" process is the key to deep learning algorithms. And the gradient descent method is one of the most common methods we use to achieve this "learning" process.

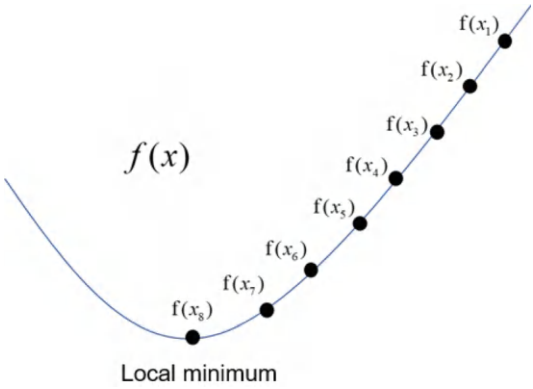


Fig. 8.1 Schematic diagram of the gradient descent method

To better understand the gradient descent method, it is necessary to know its three key elements, which are the initial point, the descent direction, and the descent step size.

Gradient descent is a first-order optimization method and one of the simple approaches we adopt when solving unconstrained optimization problems [7]. In the gradient algorithm, the primary concern is the problem of min $f(x)$, where $f(x)$ is a continuously differentiable function. When it takes the form of x0, x1, x2 ..., f(x) satisfies the following conditions:

$$f(X_{t+1}) < f(X_t), t = 0, 1, 2 \ldots \tag{8.1}$$

At this time, the process can be continuously executed, and eventually it can converge to a local minimum point. The calculation process is shown in Fig. 8.1.

One of the biggest problems in converging to a local minimum is how to find the next point. Suppose there is a function $f(x)$ with a shape as shown in Fig. 8.1. In the first step, we randomly select a point as the initial point, denoted as x_0. The next point x_1 is obtained by taking a small step from x_0 in a certain direction, and this small step is denoted as Δx. For the function $f(x)$, the change in the function value is only related to the change in x. Therefore, generally, x_{t+1} is obtained by moving x_t by Δx.

The gradient in neural networks denotes that of the loss function with respect to the model's weights and biases. The gradient method in neural networks can be expressed by the Eq. (8.2).

$$\begin{pmatrix} \boldsymbol{W}^{(i)} \\ \boldsymbol{B}^{(i)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{W}^{(i)} - \eta \frac{\partial E}{\partial \boldsymbol{W}^{(i)}} \\ \boldsymbol{B}^{(i)} - \eta \frac{\partial E}{\partial \boldsymbol{B}^{(i)}} \end{pmatrix} \tag{8.2}$$

Among them, η is called the learning rate of the neural network. Before we use the gradient method, η needs to be set to a certain value, which determines the update of the weights $\boldsymbol{W}^{(i)}$ and biases $\boldsymbol{B}^{(i)}$ of the i-th layer. $\frac{\partial E}{\partial W^{(i)}}$ and $\frac{\partial E}{\partial B^{(i)}}$ represent the partial

derivatives of the loss function E with respect to the weights and biases of the i-layer, and the gradient of the neural network is the vector summed up by these partial derivatives. An example of the gradient descent method is shown in Listing 8.1.

Listing 8.1 Implementation of Gradient Descent Method

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
  return np.power(x, 2)
def d_f_1(x):
  return 2.0 * x
def d_f_2(f, x, delta=1e-4):
  return (f(x+delta) - f(x-delta)) / (2 * delta)
# Plotting Function.
xs = np.arange(-10, 11)
plt.plot(xs, f(xs))
plt.show()
learning_rate = 0.1
max_loop = 30
x_init = 10.0
x = x_init
lr = 0.1
for i in range(max_loop):
  # d_f_x = d_f_1(x)
  d_f_x = d_f_2(f, x)
  x = x - learning_rate * d_f_x
  print(x)
print('initial x =', x_init)
print('arg min f(x) of x =', x)
print('f(x) =', f(x))
```

8.2.2 *Information Entropy and KL Divergence*

Information entropy is a frequently used metric for measuring the purity of samples. It is expressed by the following equation, which is the expected value of the self-information of the distribution. The unit depends on the base of the logarithm used in the calculation. The continuous expression of information entropy is:

$$H(p) = H(X) = E_{x\sim p(x)}\left[-\log p(x)\right] = -\int p(x)\log p(x)dx \tag{8.3}$$

The discrete expression of information entropy is:

$$H(p) = \sum_x p(x)I(x) = \sum_x p(x)\log\left(\frac{1}{p(x)}\right) = -\sum_x p(x)\log(p(x)) \tag{8.4}$$

From the above formula, we can see that when the probabilities are uniformly distributed, entropy is positively correlated with the uncertainty of the variable; when entropy increases, the amount of information we need to understand the variable also increases. For example, when Team A plays against Team B, if Team A's winning rate is 100%, we can be sure that Team A will win without any additional information. However, if both Team A and Team B have a 50% winning rate, we need a large amount of information to determine which team will win.

The KL divergence, also known as relative entropy, represents the difference between two probability distributions [8]. When the two distributions are the true distribution and the fitted distribution, the KL divergence is equal to the difference between the information entropy of the fitted distribution and that of the true distribution, indicating the information loss when using the theoretical distribution to fit the true distribution. It can be expressed by Eq. (8.5), that is,

$$D_{KL}(p||q) = \sum_{i=1}\left[p(x_i)\log p(x_i) - p(x_i)\log q(x_i)\right] \tag{8.5}$$

Among them, the probability distribution of the real event is denoted as $p(x_i)$, and the fitted probability distribution is denoted as $q(x_i)$ In the KL divergence, a KL divergence of 0 indicates that the fitted distribution is the same as the real distribution. When the fitted distribution is different from the real distribution, the KL divergence is >0. Therefore, we should understand that the KL divergence has no negative values and should be greater than or equal to 0.

8.2.3 Nash Equilibrium

Named after John Nash, the Nash equilibrium also referred to as the non-cooperative game equilibrium is a core concept in game theory. Formally defined, it denotes a strategy profile in a non-cooperative game where no participant can improve their expected payoff by unilaterally changing their strategy, given that all other participants maintain their chosen strategies. In this state, each participant's strategy is optimal relative to the strategies of others: once a participant has finalized their strategy, all remaining participants will persist in selecting the strategy that maximizes their own expected benefits. Such a mutually consistent combination of strategies across all participants constitutes a Nash equilibrium.

Within a Nash equilibrium profile, every participant acts as a self-interested competitor, prioritizing the maximization of individual payoffs when formulating strategies. This self-interested behavior may initially lead to strategic adjustments among participants, but no party can gain from unilaterally altering their choice—eliminating incentives for further modification. Consequently, the system stabilizes at a balanced state where endless strategic confrontation is resolved, and all participants achieve outcomes aligned with their optimal self-interested choices, thereby realizing a Nash equilibrium.

8.2.4 Gaussian Distribution and Gaussian Process

The Gaussian distribution, also known as the normal distribution, is a very important probability distribution in mathematics, physics, engineering, and other fields [9]. Currently, Gaussian processes may not be a major trend in the field of machine learning, but they remain at the forefront of research.

For the distribution of functions, we can use Gaussian processes to represent them. Currently, the general approach in traditional machine learning to represent the distribution of functions is to parameterize the functions and then model the distribution using the parameters. Unlike traditional computational methods, Gaussian processes can directly model and generate non-parametric models. Therefore, Gaussian processes can fit any "black box" function and also fit uncertainties. Because a Gaussian process can quantify uncertainty, we can rely on a Gaussian process to explore data distribution that was previously impossible to achieve efficient training when the amount of data increases. The most well-known application of Gaussian processes is in AlphaGo, the robot that defeated Ke Jie, the top Go player in China at that time, which used Gaussian processes for parameter tuning.

8.3 Theoretical Foundation of GANs

Building on the core concepts such as Nash equilibrium laid out in the preceding sections, this section will systematically dissect the theoretical core and operational mechanisms of GANs.

8.3.1 What is GAN

In the previous section, we briefly introduced the relevant concepts of Nash equilibrium. The GAN in deep learning was developed based on this idea, and the concept is present throughout the model structure and subsequent training.

GAN consists of two parts: one is the generator network, denoted as G network, whose purpose is to generate data that conforms to the distribution of real data; the other is the discriminator network, denoted as D network, whose purpose is to distinguish fake data. The training process of GAN can be briefly described as G and D competing against each other, learning and iterating together until neither can defeat the other.

Let's illustrate GAN with a small story to help readers understand it. Suppose there is a city with a group of counterfeiters, whose counterfeiting skills vary greatly, some being highly skilled and others being very poor. Now, the police are conducting an operation to crack down on counterfeiting. Those with poor counterfeiting skills are quickly caught by the police. Soon, the police find that there are no counterfeit goods on the market, but in fact, the market is still flooded with counterfeit goods from highly skilled counterfeiters, which the police cannot distinguish. Therefore, the police need to learn. After upgrading their learning, the police quickly catch those highly skilled counterfeiters. At this point, the highly skilled counterfeiters, in order to survive, will constantly iterate their techniques to deceive the police's inspection, while the police, in order to catch the counterfeiters, are also constantly learning to identify counterfeits.

As the police and counterfeiters keep learning, the counterfeits produced by the latter become increasingly indistinguishable from the genuine articles, while the police's ability to detect fakes also improves day by day. The police and counterfeiters constantly engage in a battle of wits and upgrade their capabilities. Eventually, the strongest police force and the most skilled counterfeiters will emerge. At this point, the probability that the police will identify the counterfeits as fake is 50%.

We can use this example to explain the training process of GAN, where the generative model is the counterfeiter and the discriminative model is the police. Figure 8.2 shows the birth of the strongest police and the strongest counterfeiter.

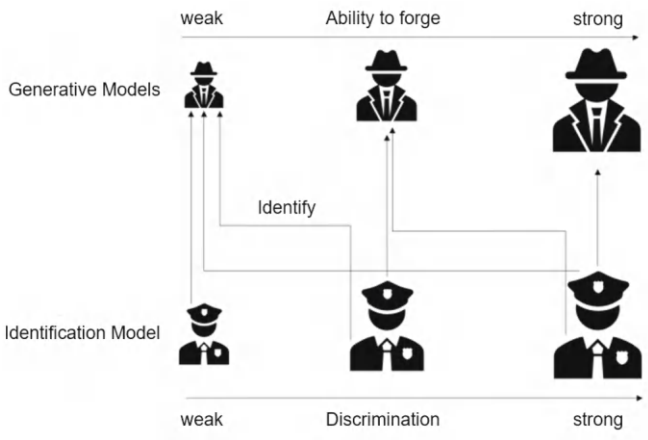


Fig. 8.2 The birth of the strongest police force and the strongest counterfeiter

8.3.2 The Principle of GANs

Let's take a detailed look at the principle of GANs through an example below.

First, define a generator G. At the same time, a random noise Z needs to be defined. The noise Z is a 100-dimensional Gaussian noise with a mean of 0 and a variance of 1. The principle of the generative model generating data is to fit the data distribution through Z. Meanwhile, there is a self-learning parameter θg, which is used to adjust Z to fit the real data distribution.

Listing 8.2 Generator G

```
26. class generator(nn.Module):
27.    def __init__(self):
28.       super(generator, self).__init__()
29.       self.gen = nn.Sequential(
30.          # Use linear transformation to map the input to 256
dimensions
31.          nn.Linear(100, 256),
32.          # relu activation
33.          nn.ReLU(True),
34.          # Linear transformation
35.          nn.Linear(256, 256),
36.          # relu activation
37.          nn.ReLU(True),
38.          # Linear transformation
39.          nn.Linear(256, 784),
40.       # Tanh activation makes the generated data distributed
between [-1,1]
41.          nn.Tanh()
42.       )
43.    def forward(self, x):
44.       x = self.gen(x)
45.       return x
```

Secondly, a discriminator D needs to be defined. D is also an Multilayer Perceptron (MLP) and has its own parameters θd that can be learned independently. Its function is to output a scalar after receiving the input data, which is used to determine whether the input data X is generated by G or is real data. If it is real data, it is assigned a value of 1; if it is generated data, it is assigned a value of 0. When the discriminator is sufficiently well-trained, $D(Z)$ is 0, which means that $1 - D(G(\mathbf{Z}))$ is 1, that is, $\log(1 - D(G(\mathbf{Z})))$ is 0. When the discriminator is imperfect, $\log(1 - D(G(\mathbf{Z})))$ will yield a value <0.

Listing 8.3 Discriminator D

```
class discriminator(nn.Module):
    def __init__(self):
        super(discriminator,self).__init__()
        self.dis = nn.Sequential(
            # The input feature number is 784 and the output
feature number is 256
            nn.Linear(784, 256),
            # Perform nonlinear mapping
            nn.LeakyReLU(0.2),
            # Perform a linear mapping
            nn.Linear(256, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
           # Activation function, in the binary classification
problem, sigmoid can map real numbers to [0,1] as probability
values
            # Multi-classification using softmax function
            nn.Sigmoid()
        )
    def forward(self, x):
```

The objective of the GAN model is to generate data that is indistinguishable from the real data after adversarial training. This means that the data generated by the generator G should gradually approach the real data, and the vector Z should fit the distribution of the real data as closely as possible. Figure 8.3 shows the process by which the data generated by the generator G gradually approaches the real data.

In Fig. 8.3, the orange line represents the probability distribution of the discriminative model, the black dotted line represents the probability distribution of the original data, and the green solid line represents the probability distribution of the data generated by the generative model.

Figure 8.3a shows the situation at the beginning of training, where the orange wavy line indicates that the probability distribution of the discriminative model fluctuates greatly, and the difference between the green line and the black dotted line indicates

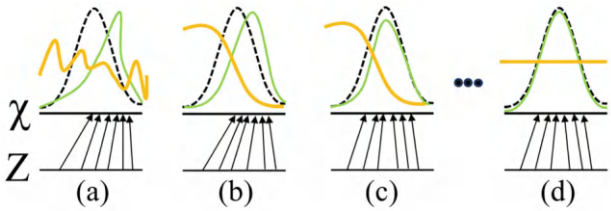


Fig. 8.3 The process by which the data generated by generator G gradually approaches the real data

that the probability distribution of the generated data has a significant difference from that of the real data. Figure 8.3b shows that after training, the discriminator can distinguish between real data and generated data, at which point the probability distribution curve of the discriminator becomes smooth, as shown by the orange line in the figure. The higher the position of the points on the orange line, the higher the probability that the discriminator believes the data generated by the generator is real data. After training for a period of time, the probability distribution of the data generated by the generator becomes increasingly similar to that of the real data, as shown in Fig. 8.3c, where the green line starts to move left towards the black dotted line. After a considerable amount of training, the best result is shown in Fig. 8.3d, where it can be seen that the probability distribution of the data generated by the generator is almost the same as that of the real data, and at this point, the probability that the discriminator determines the generated data to be false is 50%.

We use $P_{data}(x)$ to represent the distribution of the real image set, where x represents the real data, and $P_G(x)$ represents the distribution of the data generated by the generator, which is usually controlled by a parameter θ. Generally, the random noise used by the generator follows a Gaussian distribution, and at this time, θ is the mean and variance of each Gaussian distribution. Therefore, the final probability distribution of the generator is $P_G(x;\theta)$.

The primary objective of the generator G is to make the probability distribution of the generated data identical to that of the real data, i.e., $P_{data} = P_G$, as shown in Fig. 8.3d. This implies that as the probability distribution of the generated data approaches that of the real data, the expectation of $P_G(x;\theta)$ increases. At this point, the issue of having the generator produce data that is judged as real data transforms into the problem of finding the likelihood of the generator $L = \prod_{i=1}^{m} P_G\left(x^i;\theta\right)$ be the maximum likelihood value. We can maximize the likelihood by using a $\theta*$. The specific process is as follows:

$$
\begin{aligned}
\theta^* &= \arg\max_{\theta} \prod_{i=1}^{m} p_G(x^i;\theta) \\
&= \arg\max_{\theta} \log \prod_{i=1}^{m} P_G(x^i;\theta) \\
&= \arg\max_{\theta} \sum_{i=1}^{m} \log P_G(x^i;\theta) \\
&\approx \arg\max_{\theta} E_{x\sim P_{data}}[\log P_G(x^i;\theta)] \\
&= \arg\max_{\theta} \int_x P_{data}(x)\log P_G(x;\theta)dx - \int_x P_{data}(x)\log P_{data}(x)dx \\
&= \arg\max_{\theta} \int_x P_{data}(x)(\log P_G(x;\theta) - \log P_{data}(x))dx
\end{aligned}
$$

$$= \arg \min_{\theta} \int_{x} P_{data}(x) \log \frac{p_{data}(x)}{P_G(x;\theta)}$$
$$= \arg \min_{\theta} KL(p_{data}(x)||P_G(x;\theta))$$

It can be seen that when maximizing the likelihood, the addition of log transforms the result into the expectation of the log-likelihood of x in the probability distribution of the generated data. This not only does not affect the outcome but also simplifies the calculation process.

By calculating the probability integral, the expected value of all x in the true distribution can be roughly obtained. Since $\int_{x} P_{\text{data}} \log P_{\text{data}}(x)dx$ this term is irrelevant to θ, after subtracting this term, the result is equivalent. Then perform the normal extraction of common factors and merge, and you get $\arg \min_{\theta} x \int_{x} P_{\text{data}}(x)\left(\frac{\log P_G(x;\theta)}{\log P_{\text{data}}(x)}\right)dx$, find the maximum value. At this time, swap the numerator and denominator to get $\arg \min_{\theta} x \int_{x} P_{\text{data}}(x)\left(\frac{\log P_{\text{data}}(x)}{\log P_G(x;\theta)}\right)$, that is $\arg \min_{\theta} KL(P_{\text{data}}(x)||P_G(x;\theta))$. The problem of maximizing the likelihood becomes finding the KL divergence, which also indirectly verifies that when $P_{\text{data}} = P_G$ the probability distribution likelihood of the generated data is the largest.

The discriminator mainly determines the authenticity of the received data. The maximum value of the probability distribution of the discriminator is the optimal goal achieved by the discriminator; while the generator hopes that the data it generates will be judged as real data, that is, the value of $V(D, G)$ is the smallest. The generator and the discriminator compete with each other, and we can get the core formula of GAN:

$$\min_{G} \min_{D} xV(\mathrm{D}, \mathrm{G}) = E_{x\sim P_{data}(x)}\left[\log D(\boldsymbol{x})\right] + E_{Z\sim P_z}\left[\log(1 - D(G(z)))\right] \tag{8.6}$$

8.4 The Training Process of GAN

We know that the optimization process of GAN is to first train the discriminator, and then use the trained discriminator to train the generator. When the generator is trained to be able to confuse the discriminator, the discriminator optimizes itself according to the returned loss. The whole process is continuously iterated until the best generator and the best discriminator are obtained.

The reason why we choose to train the discriminator first is that if the generator is trained without a criterion, the data generated in the end will almost all be noise and

not the target data. Therefore, training the discriminator first can ensure the normal operation of the model.

Below, we introduce the training process of GAN in detail.

8.4.1 Train the Discriminator

In the first stage, we train the discriminator first, so we need to fix the generator and train the discriminator. We use random noise to make the generation network continuously generate fake data, and then use the discriminator to determine whether these data are real. At the beginning, the data distribution fitted by the initial vector $\boldsymbol{Z}$ is far from the real data distribution, so the discriminator can easily distinguish the data generated by the generator.

Listing 8.4 Training the Discriminator

```
# Training the Discriminator
    real_img = img.view(num_img, -1).cuda()
    real_label = torch.ones(num_img).cuda()
    fake_label = torch.zeros(num_img).cuda()
    # Feed the real image into the discriminator
    real_out = D(real_img)
    print(real_out.shape[0])
    # Get the loss of the real picture
    d_loss_real = criterion(real_out, real_label)
     # Get the discriminant value of the real picture. The
closer the output value is to 1, the better.
    real_scores = real_out
    # Calculating the loss of fake images
    # Generate some random noise
                    z = torch.normal(0, 0.02,(num_
img,100),dtype=torch.float32).cuda()
     # Random noise is put into the generative network to
generate a fake picture
    fake_img = G(z)
    # Discriminator judges fake pictures
    fake_out = D(fake_img)
    # Get the loss of fake pictures
    d_loss_fake = criterion(fake_out, fake_label)
     # Get the discriminant value of the fake image. For
the discriminator, the closer the loss of the fake image is to
0, the better
    fake_scores = fake_out
    # Loss Function and Optimization
    # Losses include true losses and false losses
    d_loss = d_loss_real + d_loss_fake
    # Before backpropagation, the gradient is reset to 0
```

```
    d_optimizer.zero_grad()
    # Back propagating the error
    d_loss.backward()
    # Update Parameters
    d_optimizer.step()
```

8.4.2 *Training Generator*

The second stage is to train the generator (as shown in Listing 8.5). At this point, the discriminator D has been trained and needs to be fixed for the training of the generator. After the first stage of training, the discriminator has been able to perfectly distinguish the data generated by the generator. Therefore, further training of the discriminator D is meaningless. At this time, the discriminator D needs to be fixed to train the generator until the discriminator cannot distinguish the data generated by the generator as fake data, and one training process is completed (Fig. 8.4).

Listing 8.5 Training the Generator

```
# Training the Generator
            z = torch.normal(0, 0.02, (num_img, 100), dtype=torch.float32).cuda()
                # z = Variable(torch.randn(num_img, z_dimension)).cuda() # Get random noise
    # Random noise is input into the generator to get a fake picture
    fake_img = G(z)
```

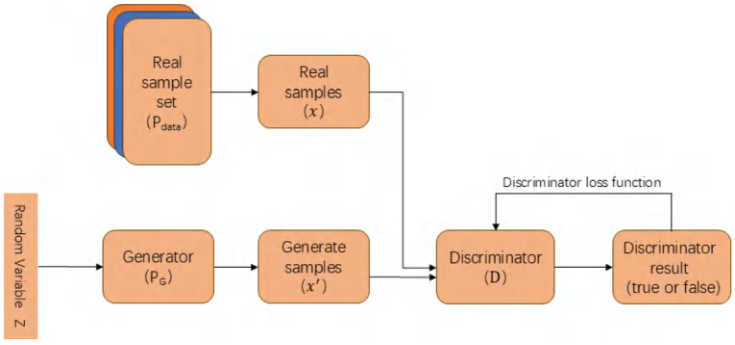


Fig. 8.4 One iteration process of GAN training

```
    # The result obtained by the discriminator
    output = D(fake_img)
      # The loss of the labels of the fake and real images
obtained
    g_loss = criterion(output, real_label)
     # bp and optimize
     # Gradients are reset to zero
     g_optimizer.zero_grad()
     # Perform back propagation
     g_loss.backward()
      # step() is generally used after back propagation to
update the parameters of the generated network
     g_optimizer.step()
```

8.5 Applications and Code Examples of GAN

In this section, we select the handwritten dataset to demonstrate the application of GAN.

First, import the libraries (as shown in Listing 8.6). The libraries used in this case are torch, torchvision, and os.

Listing 8.6 Calling the Library

```
import torch. nn as nn
from torchvision import transforms
from torchvision import datasets
from torchvision.utils import save_image
import torch.autograd
from torch.autograd import Variable
import os
```

Create a folder for saving data and import the data, as shown in Listing 8.7.

Listing 8.7 Creating a Folder for Saving Data and Importing Data

```
# Create a folder.
if not os.path.exists('./img'):
  os.mkdir('./img')
# Confirm the image size and pre-process the image.
def to_img(x):
  out = 0.5*(x+1)
```

```
    # The Clamp function can limit the randomly changing value
to a given interval [min, max]
    out = out.clamp(0, 1)
    # The view() function is used to concatenate a multi-line
Tensor into a single line.
    out = out.view(-1, 1, 28, 28)
    return out
# A batch of 128.
batch_size = 128
# Total 100 batches.
num_epoch = 100
# Noise Dimension.
z_dimension = 100
# Image preprocessing.
img_transform = transforms.Compose([
    transforms.ToTensor(),
    # Single channel #normalization.
    transforms.Normalize([0.5], [0.5])
])

# Load the handwriting dataset to load the data. Here we
store the dataset locally.
# Dataset loading.
# Download the MNIST dataset.
mnist = datasets.MNIST(
    root='./data/mnist/',
    train=True,
    transform=img_transform,
    download=True)
# Data loader data loading (batch reading).
dataloader = torch.utils.data.DataLoader(
    dataset=mnist,
    batch_size=128,
    shuffle=True)
```

Define the discriminator (as shown in Listing 8.8). The discriminator uses a multi-layer network. First, we flatten the 28 × 28 image into 784, then pass it through a multi-layer perceptron, with LeakyReLU() activation function in the middle, and finally through a Sigmoid() activation function to obtain a probability between 0 and 1 for binary classification.

Listing 8.8 Defining the Discriminator 'Discriminator'

```
class discriminator(nn.Module):
    def __init__(self):
        super(discriminator,self).__init__()
        self.dis = nn.Sequential(
```

```
5.       # The input feature number is 784 and the output feature
number is 256.
6.         nn.Linear(784, 256),
7.         # Perform nonlinear mapping.
8.         nn.LeakyReLU(0.2),
9.         # Perform a linear mapping.
10.         nn.Linear(256, 256),
11.         nn.LeakyReLU(0.2),
12.         nn.Linear(256, 1),
13.        # Activation function, in the binary classification
problem, sigmoid can map real numbers to [0,1] as probability
values.
14.         # Multi-classification using softmax function.
15.         nn.Sigmoid()
16.       )
17.    def forward(self, x):
18.       x = self.dis(x)
19.       return x.reshape(-1)
```

Define the generator (as shown in Listing 8.9), which takes a 100-dimensional Gaussian distribution ranging from 0 to 1 as input. It maps this to a 256-dimensional space through a linear transformation, followed by an activation function. Then, it undergoes another linear transformation, followed by another activation function, and finally a linear transformation to 784 dimensions. The last step is to apply the Tanh() activation function, with the aim of generating a distribution of fake image data.

Listing 8.9 Defining a Generator

```
1. class generator(nn.Module):
2.    def __init__(self):
3.       super(generator, self).__init__()
4.       self.gen = nn.Sequential(
5.          # Use linear transformation to map the input to 256
dimensions.
6.         nn.Linear(100, 256),
7.         # relu activation.
8.         nn.ReLU(True),
9.         # Linear transformation.
10.         nn.Linear(256, 256),
11.         # relu activation
12.         nn.ReLU(True),
13.         # Linear transformation
14.         nn.Linear(256, 784),
15.       # Tanh activation makes the generated data distributed
between [-1,1].
16.         nn.Tanh()
```

```
        )
    def forward(self, x):
        x = self.gen(x)
        return x
```

To train the discriminator (as shown in Listing 8.10), we divide its training into two parts: first, to identify real images as real; second, to identify fake images as fake. First, we need to define the metric for the loss, using binary cross-entropy. Next, we define the optimization function, with a learning rate of 0.0003 for the optimization function. During this process, the parameters of the generator are constantly updated.

Listing 8.10 Training the Discriminator

```
for epoch in range(num_epoch):
    for i, (img, _) in enumerate(dataloader):
        # print(img.shape)
        num_img = img.size(0)
        real_img = img.view(num_img, -1).cuda()
        real_label = torch.ones(num_img).cuda()
        fake_label = torch.zeros(num_img).cuda()
        # Calculate the loss of real images.
        # Feed the real image into the discriminator.
        real_out = D(real_img)
        print(real_out.shape[0])
        # Get the loss of the real picture.
        d_loss_real = criterion(real_out, real_label)
        # Get the discriminant value of the real picture. The
closer the output value is to 1, the better.
        real_scores = real_out
        # Calculating the loss of fake images.
        #  z  =  Variable(torch.randn(num_img,  z_
dimension)).cuda()
        z  =  torch.normal(0,  0.02,  (num_img,  100),
dtype=torch.float32).cuda()
        # Random noise is put into the generative network to
generate a fake picture.
        fake_img = G(z)
        # Discriminator judges fake pictures.
        fake_out = D(fake_img)
        # Get the loss of fake pictures.
        d_loss_fake = criterion(fake_out, fake_label)
        # Get the discriminant value of the fake image. For the
discriminator, the closer the loss of the fake image is to 0,
the better.
        fake_scores = fake_out
        # Loss Function and Optimization.
        # Losses include true losses and false losses.
        d_loss = d_loss_real + d_loss_fake
```

```
    # Before backpropagation, the gradient is reset to 0.
    d_optimizer.zero_grad()
    # Back propagating the error.
    d_loss.backward()
    # Update Parameters.
    d_optimizer.step()
```

The fixed discriminator is used to train the generator (as shown in Listing 8.11), with the aim of making the generated images be judged as real by the discriminator. During training, the result of the generated images passed into the discriminator is compared with the real ones, and the parameters in the generator network are updated through backpropagation of the updated parameters, thereby training the network to make the generated images be judged as real by the discriminator.

Listing 8.11 Training the Generator

```
        z = torch.normal(0, 0.02, (num_img, 100),
dtype=torch.float32).cuda()
            # z = Variable(torch.randn(num_img, z_
dimension)).cuda() # Get random noise.

    # Random noise is input into the generator to get a fake
picture.
    fake_img = G(z)
    # The result obtained by the discriminator.
    output = D(fake_img)
     # The loss of the labels of the fake and real images
obtained.
    g_loss = criterion(output, real_label)
     # Gradients are reset to zero.
     g_optimizer.zero_grad()
     # Perform back propagation.
     g_loss.backward()
      # step() is generally used after back propagation to
update the parameters of the generated network.
     g_optimizer.step()
     # Print the intermediate loss.
     if (i + 1) % 100 == 0:
        # The printout is the mean loss of the real image.
        print('Epoch[{}/{}],d_loss:{:.6f},g_loss:{:.6f}
'
            'D real: {:.6f},D fake: {:.6f}'.format(epoch,
num_epoch,    d_loss.item(),    g_loss.item(),    real_
scores.data.mean(), fake_scores.data.mean()))
```

Save the image and the trained model, as shown in Listing 8.12.

Fig. 8.5 Examples of handwritten digit images generated using GAN

Listing 8.12 Saving Images and the Trained Model

```
1.      **if epoch == 0:
2.        *real_images = to_img(real_img.cpu().data)
3.         save_image(real_images, './img/real_images.png')
4.       fake_images = to_img(fake_img.cpu().data)
5.              save_image(fake_images, './img/fake_images-
{}.png'.format(epoch + 1))
6.
7. torch.save(G.state_dict(), './generator.pth')
8. torch.save(D.state_dict(), './discriminator.pth')
9.      g_optimizer.step()
```

Figure 8.5 shows examples of handwritten digit images generated using GAN.

8.6 Advantages and Limitations of GANs

This section will elaborate on the unique advantages of GANs in comparison with traditional generative models, as well as their technical limitations.

8.6.1 Advantages of GAN

Generative Adversarial Networks (GANs), as a landmark architecture in the field of generative models, exhibit multiple core advantages over traditional generative

methods; these advantages primarily manifest in three dimensions: its training mechanism, data distribution fitting approach, and adaptability to learning paradigms:

(1) GAN only uses backpropagation, so it does not require the complex Markov chains that other traditional generative models do.
(2) GAN does not require humans to search for the real data distribution. Instead, it uses the discriminator to fit the real data distribution, saving a significant amount of human and material resources. At the same time, it reduces the pressure on researchers, lowers the difficulty of computation, and enhances its applicability.
(3) The training of GAN is a kind of semi-supervised learning approach, so GAN can be widely applied in the fields of unsupervised learning and semi-supervised learning, thus having a broader prospect.

8.6.2 Disadvantages of GAN

While Generative Adversarial Networks (GANs) demonstrate considerable efficacy in generative modeling tasks, they also exhibit inherent limitations, scenario-specific applicability tendencies, and typical training pathologies—these attributes can be illustrated via the following core aspects:

(1) GAN fits the real data through the discriminator, which is invisible to researchers, thus being a "black box". As a result, some unexpected situations may occur, such as vanishing gradients.
(2) Starting from the algorithmic principle of GAN, GAN is more suitable for continuous expectations, and thus is more applicable to the field of image processing.
(3) If, during training, the data generated by G is not real but is identified as real by D, then G will continuously approach the wrong distribution, while D believes that it is the real data distribution. From then on, the two models will mutually "deceive" each other under the wrong circumstances. This phenomenon is called mode collapse.
(4) GAN aims to reach a Nash equilibrium state, but in practical applications, it is difficult to achieve this state perfectly, and GAN sometimes has difficulty converging.

8.6.3 GAN Zoo

With the development of artificial intelligence, researchers have shown great enthusiasm for the study of GANs. There have been improvements in the GAN model framework and theory, modifications to the GAN model, or combinations with other learning methods to enable it to be applied in more scenarios. As a result, the study of GANs has become a trend in the academic circle. Almost every week, new papers on GANs are published, and over time, this has led to the formation of the GAN Zoo.

The basic model of GANs is composed of three major modules: input and output, generator, and discriminator, combined according to certain principles or ideas [10]. Therefore, the entry points for improving the model structure can be divided into five parts: input and output, generator, discriminator, the module structure of the model, and the combination ideas of the module structure.

A brief introduction to GAN Zoo is presented in Table 8.1.

GAN consists of a generator and a discriminator, and it does not require humans to find the real data distribution, thus having a strong modeling ability and a strong ability to generate data. The image field is currently the main application field of GAN, but its application in text processing, sound, and other fields is also increasing [11]. Although GAN has achieved results in various fields, in fact, its application effect in other fields is not as good as that in the image field. The image field has always been the best application field of GAN models. Therefore, the development direction of GAN can be how to apply it to more fields better and achieve satisfactory

Table 8.1 A brief introduction to GAN Zoo

Improvement direction	Improvement methods	Derivatives of the GAN model
Input	Improvement based on hidden variables Improvements based on latent space Other input improvements	cGAN, BiCoGAN, IcGAN DeLiGAN, NEMGAN FCGAN
Output	Multi-classification output Feature output	SGAN, AC-GAN InfoGAN
Generator	Ensemble methods; Sample module differences Others	AdaGAN MADGAN, MGAN MPGAN
Discriminator	Ensemble methods; Sample module differences Others	PacGAN, GMAN, DropoutGAN D2GAN, StabilizingGAN EBGAN, BEGAN
Multi-module combination	Multi-stage model Auxiliary module model Others	GRAN, StackGAN, ProgressGAN TripleGAN, controlGAN SGAN, MemoryGAN
Model crossover	Neural network structure replacement Composite generative model Others	DCGAN, CapsuleGAN VAEGAN, DEGAN, AAE, BiGAN, Ali SAGAN, KDGAN, IRGAN, LapGAN, QuGAN
Distribution distance measure	f-divergence-based methods IPM-based approach Others	LSGAN, EBGAN WGAN, WG AN-GP, BWGAN, RWGAN, FisherGAN IGAN, MMGAN, OT-GAN
Gradient calculation process		MAGAN, RGAN, LSGAN,

results in these fields. With the development of artificial intelligence, combining advanced technologies such as deep learning to weaken its shortcomings and improve its application effect is also one of its future research directions.

References

1. Goyal, A., Bengio, Y.: Inductive biases for deep learning of higher-level cognition. Proceed. Royal Soc. A **478**, 20210068 (2022)
2. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
3. Liang, H., Sun, X., Sun, Y., Gao, Y.: Text feature extraction based on deep learning: a review. EURASIP J. Wirel. Commun. Netw. **2017**, 211 (2017)
4. Makarenko, A.V.: Deep neural networks: origins development current status. Control Sci. **12**, 3–19 (2020)
5. Goodfellow, I.J., et al.: Generative adversarial nets. Adv. Neural. Inf. Process. Syst. **27**, 1212 (2014)
6. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. IEEE Trans. Evol. Comput. **23**, 921–934 (2019)
7. Battiti, R.: First-and second-order methods for learning: between steepest descent and Newton's method. Neural Comput. **4**, 141–166 (1992)
8. Malinin, A., Gales, M.: Reverse kl-divergence training of prior networks: improved uncertainty and adversarial robustness. Adv. Neural. Inf. Process. Syst. **32**, 79 (2019)
9. Springer, M.D., Thompson, W.E.: The distribution of products of beta, gamma and Gaussian random variables. SIAM J. Appl. Math. **18**, 721–737 (1970)
10. Alqahtani, H., Kavakli-Thorne, M., Kumar, G.: Applications of generative adversarial networks (GANs): an updated review. Arch. Comput. Methods Eng. **28**, 525–552 (2021)
11. Dash, A., Ye, J., Wang, G.: A review of generative adversarial networks (GANs) and its applications in a wide variety of disciplines: from medical to remote sensing. IEEE Access **12**, 18330–18357 (2023)

Chapter 9
Transformer

This chapter introduces the Transformer model, a deep neural network architecture based on Self-Attention, which was first proposed by Google in 2017 in the landmark paper "Attention Is All You Need" [1], and was initially used for natural language processing (NLP) tasks, especially in machine translation, text classification and language modelling. Thanks to its excellent parallel computing characteristics and long-range dependency modelling capability, Transformer has rapidly overtaken traditional Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) to become the dominant model in the field of NLP.

As research continues to advance, the Transformer architecture has also been successfully extended to the field of computer vision. The models represented by Vision Transformer (ViT) have achieved breakthrough results in tasks such as image classification, target detection, and semantic segmentation. Transformer has also demonstrated strong cross-border capabilities in life sciences, especially in molecular science, drug discovery and protein engineering. By modelling the complex dependencies between molecular sequences and 3D structures, Transformer has been applied to molecular property prediction, protein structure modelling, and drug-target interaction analysis, providing strong technical support for the integration of AI and life sciences.

9.1 A First Look at Transformer: From the Dilemma of Sequence Modelling

In the field of NLP, sequence modeling serves as a core task and is widely applied in key scenarios such as machine translation, text generation, and speech recognition. An ideal sequence modeling method should effectively capture complex and long-range dependencies within the input sequence, thereby producing semantically coherent and logically consistent output.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_9

Traditional sequence modeling approaches primarily rely on RNNs and their variants, such as LSTM [2] and GRUs. These models typically process sequences in a step-by-step recursive manner, where each time step updates its state based on the output of the previous one. While this sequential structure performs well on tasks with short-term dependencies, it faces significant challenges when dealing with long sequences. On the one hand, RNNs are prone to issues such as vanishing or exploding gradients, making it difficult to capture long-range dependencies. On the other hand, even with gating mechanisms, models like LSTM and GRU may still lose crucial early information in very long texts, limiting their ability to model context. Furthermore, the inherently sequential nature of RNNs hampers parallelization, resulting in limited training efficiency and difficulty in meeting the demands of today's large-scale data-driven deep learning.

To overcome these bottlenecks, researchers proposed the Attention Mechanism, whose core idea is to enable models to dynamically focus on key information within the input, rather than adhering to fixed orders or local windows. This mechanism computes attention weights by measuring the relevance between queries and keys, and then applies these weights to aggregate values, thereby achieving explicit information focus.

Building upon this idea, the Self-Attention mechanism was introduced, allowing every position in a sequence to interact with all other positions. This significantly enhances the model's ability to capture global dependencies and removes the need for recurrent structures. The Transformer model, which is based entirely on self-attention, redefined the paradigm of sequence modeling with a fully parallelizable architecture. Its structure consists of an encoder and a decoder, each composed of multiple stacked layers. Each layer integrates core components such as Multi-Head Self-Attention and Feed-Forward Networks. The multi-head mechanism captures multi-level semantic features in parallel, while the feed-forward module enhances nonlinear representational capacity.

Compared with traditional models like RNNs, the Transformer not only improves the modeling of long-distance dependencies but also significantly enhances training efficiency. As a result, it quickly became the dominant framework for NLP tasks. In recent years, the application of Transformer has expanded beyond natural language to include computer vision (e.g., image recognition and object detection), life sciences (e.g., molecular property prediction and protein modeling), and other multi-modal tasks, gradually evolving into a foundational architecture for cross-modal generalization in deep learning.

9.2 Building Blocks of Transformer

9.2.1 Self-Attention Mechanism

Self-Attention mechanism is the most representative and innovative structural component of the Transformer model. Its core objective is to capture global dependencies among different positions within an input sequence. Unlike traditional neural networks that rely on fixed receptive fields or stacked structures to model local context, the self-attention mechanism dynamically computes attention weights between any two positions in the sequence, effectively establishing full connectivity. This allows the model to retain global information while enhancing its ability to handle long-range dependencies.

In the self-attention mechanism, the computation of each element involves three key vectors: the Query vector, the Key vector, and the Value vector. The Query vector is used to retrieve relevant information, the Key vector serves as a reference for matching, and the Value vector holds the actual information representation. The similarity between the Query and all Key vectors is calculated via dot-product operations, resulting in attention scores. These scores are then normalized using the Softmax function to convert them into probability weights. Finally, the model performs a weighted sum over the Value vectors using these attention weights to produce the updated representation of each element. The definitions of the Query, Key, and Value vectors are as follows:

$$Q = XW_Q, K = XW_K, V = XW_V \tag{9.1}$$

Here, $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learnable parameter matrices, and d_k is the dimensionality of each attention head. Figure 9.1 illustrates this transformation process:

The similarity between the query vector Q and the key vector K is calculated using a dot-product operation, followed by a scaling step to maintain numerical stability:

$$S = \frac{QK^T}{\sqrt{d_k}} \tag{9.2}$$

Subsequently, Softmax normalisation is performed on S to obtain the attention weight matrix:

$$A = softmax(S) \tag{9.3}$$

The Softmax operation converts the similarity values of each row into probability values, ensuring that the weight of each element sums to 1. Finally, the attention weight matrix A is multiplied by the value matrix V to obtain a weighted representation of the element O:

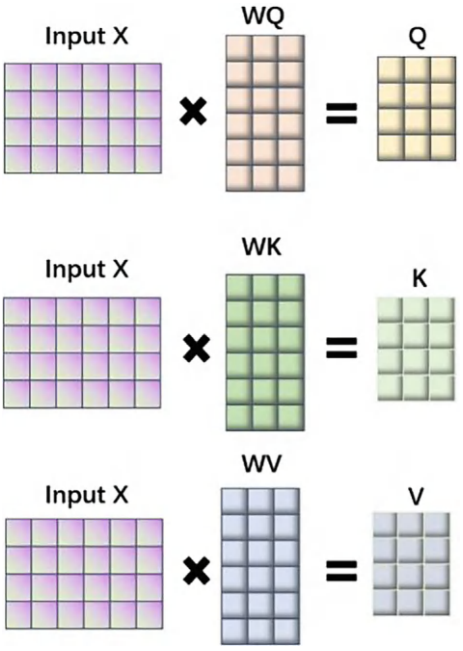


Fig. 9.1 Schematic diagram of the changes in Q, K, and V

$$O = AV \tag{9.4}$$

Figure 9.2 illustrates the structural differences between traditional fully connected models and the self-attention mechanism. In fully connected networks, the connection weights are static parameters that remain fixed after training; whereas in the self-attention mechanism, the weights are dynamically generated similarity matrices based on the input data, offering greater representational flexibility and contextual awareness. Moreover, the self-attention mechanism naturally supports the modeling of variable-length sequences without relying on fixed receptive field windows, making it particularly suitable for various types of sequential or non-sequential inputs such as natural language, image patches, and molecular structures.

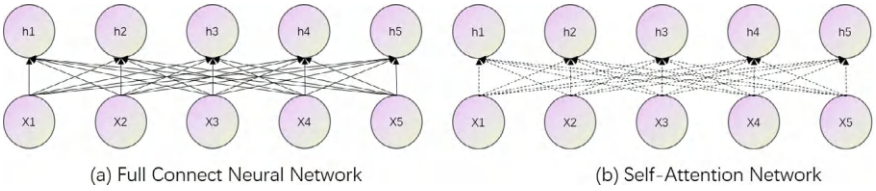


Fig. 9.2 Comparison between fully connected model and self-attention model

9.2.2 Multi-head Self-Attention Mechanism

Although the self-attention mechanism can capture global dependencies between any positions in a sequence and offers superior modeling capabilities compared to convolutional and recurrent structures, a single attention head typically learns features within a specific representation subspace, leading to limitations in information diversity and expressive power. To address this issue, the Transformer introduces the Multi-Head Self-Attention mechanism to enhance the model's perceptual ability across different feature subspaces and increase representational diversity.

The core idea of the multi-head attention mechanism is to project the queries (Q), keys (K), and values (V) into multiple lower-dimensional subspaces, perform self-attention computations independently in each subspace, and then concatenate the outputs of all heads before mapping them back to the original dimension to form a unified sequence representation. In this way, the model can learn diverse contextual patterns in parallel across multiple subspaces, such as local dependencies and global semantics.

Mathematically, the computation of multi-head self-attention is as follows. Suppose there are M attention heads. For the i-th head, the queries, keys, and values are linearly projected into different subspaces:

$$Q_i = XW_i^Q, K_i = XW_i^K, V_i = XW_i^V \tag{9.5}$$

Here, $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d_k}$ are the learnable parameter matrices for the i-th head, and $d_k = d/M$ is the dimensionality of a single attention head. Subsequently, each head computes attention independently:

$$\text{Attention}_i(Q_i, K_i, V_i) = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right)V_i \tag{9.6}$$

Finally, the outputs of all heads are concatenated and projected back to the original dimension through a linear transformation, ensuring that the output of the multi-head self-attention remains compatible with subsequent layers. This process can be expressed by the following formula:

$$O_{\text{final}} = Concat(\text{head}_1, \text{head}_2, \ldots, \text{head}_M)W^O \tag{9.7}$$

Combining the above steps, the complete computation formula for multi-head self-attention is as follows:

$$MultiHead(Q, K, V) = Concat(\text{head}_1, \text{head}_2, \ldots, \text{head}_M)W^O \tag{9.8}$$

The multi-head attention mechanism offers significant advantages over single-head attention in several aspects. First, different attention heads can focus on features that represent different levels or semantic relationships within the input sequence. In

NLP tasks, some attention heads may learn syntactic dependency structures, while others capture long-range semantic associations. In computer vision tasks, certain heads may focus on local edge textures, while others attend to the overall spatial layout. Second, the multi-head mechanism naturally supports parallel computation, fully leveraging GPU resources and significantly improving efficiency during both training and inference. Third, through decomposition and recombination across multiple heads, the Transformer enhances its representational capacity, enabling more fine-grained and robust feature modeling for downstream tasks.

Compared with traditional recurrent models such as RNNs and LSTMs, the multi-head attention mechanism not only removes the constraint of sequential computation but also fully integrates global contextual information. As a result, it performs exceptionally well in long-sequence modeling tasks, providing a unified and powerful modeling paradigm for language understanding, image recognition, protein sequence analysis, and other applications.

Listing 9.1 Multi-head Attention

```
class MultiHeadAttention(nn.Module):
  def __init__(self, input_dim, num_heads):
    super(MultiHeadAttention, self).__init__()

     # Ensure the hidden size is divisible by the number of
heads
     assert input_dim % num_heads == 0, "input_dim must be
divisible by num_heads"

    self.input_dim = input_dim
    self.num_heads = num_heads
     self.head_dim = input_dim // num_heads

     # Linear projections to obtain Q, K, V
     self.query = nn.Linear(input_dim, input_dim)
     self.key = nn.Linear(input_dim, input_dim)
     self.value = nn.Linear(input_dim, input_dim)

     # Final output projection
     self.fc_out = nn.Linear(input_dim, input_dim)

  def forward(self, x):
     batch_size, seq_len, _ = x.size()

     # Project inputs and split into multiple heads
    Q = self.query(x).view(batch_size, seq_len, self.num_
heads, self.head_dim)
     K = self.key(x).view(batch_size, seq_len, self.num_
heads, self.head_dim)
    V = self.value(x).view(batch_size, seq_len, self.num_
heads, self.head_dim)
```

```
27.
28.       # Reorder to [batch_size, num_heads, seq_len, head_
dim] for parallel computation
29.      Q = Q.permute(0, 2, 1, 3)
30.      K = K.permute(0, 2, 1, 3)
31.      V = V.permute(0, 2, 1, 3)
32.
33.      # Scaled dot-product attention
34.         scores = torch.matmul(Q, K.transpose(-2, -1)) /
(self.head_dim ** 0.5)
35.       attn_weights = F.softmax(scores, dim=-1) # [batch_
size, num_heads, seq_len, seq_len]
36.
37.      # Weighted sum of values
38.     head_outputs = torch.matmul(attn_weights, V) # [batch_
size, num_heads, seq_len, head_dim]
39.
40.      # Concatenate heads
41.          concat_output = head_outputs.permute(0, 2, 1,
3).contiguous()
42.      concat_output = concat_output.view(batch_size, seq_
len, self.input_dim) # [batch_size, seq_len, input_dim]
43.
44.      # Final linear layer
45.      output = self.fc_out(concat_output)
46.
47.      return output, attn_weights
```

9.3 Assembling the Transformer: From Scratch to Completion

In the previous chapters, we have provided a detailed explanation of the core components of the Transformer: the self-attention mechanism and the multi-head attention mechanism. Now, we will assemble these components to construct the complete Transformer model. The Transformer consists of two main parts: the encoder and the decoder, each composed of multiple identical stacked layers. Each layer contains three core subcomponents: the Embedding Layer, the Multi-Head Self-Attention mechanism, and the Feed-Forward Network. In addition, the Transformer incorporates Residual Connections and Layer Normalization to enhance training stability and overall model performance.

9.3.1 Embedding

In the Transformer model, embedding is primarily used to map discrete input data (such as words or subwords) into a continuous vector space, enabling subsequent computation by the self-attention mechanism. Since the Transformer itself lacks inherent awareness of sequence order, it is necessary to introduce additional positional encoding to provide information about the position of each token.

Word embedding is the first step in processing inputs in the Transformer model. Its function is to convert discrete tokens (words or subwords) into fixed-dimensional vector representations. The formula is as follows: $E \in \mathbb{R}^{V \times d_{model}}$, V is the vocabulary size. d_{model} is the hidden dimension of the Transformer, which also serves as the dimensionality of the embedding vectors. For each token t_i in the input sequence, its embedding vector is represented as: $E(t_i) \in \mathbb{R}^{d_{model}}$. During training, the values of the embedding matrix are continuously updated to learn the optimal representations of the tokens.

Since the Transformer cannot directly perceive the order of a sequence, additional Positional Encoding (PE) is required to provide positional information. The original Transformer paper proposed a fixed sinusoidal positional encoding, defined by the following formulas:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{9.9}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{9.10}$$

pos represents the position index of the token in the sequence, and i represents the index of the embedding dimension. Positional encoding assigns unique numerical representations to tokens at different positions and uses fixed trigonometric functions to provide relative position information, thereby enhancing the model's ability to generalize across sequences of varying lengths (Fig. 9.3).

Listing 9.2 Implementation of the Positional Encoding Module

```
48. import torch
49. import torch.nn as nn
50. import math
51.
52. class PositionalEncoding(nn.Module):
53.   def __init__(self, d_model, max_len=5000):
54.     super().__init__()
55.     position = torch.arange(max_len).unsqueeze(1)
56.       div_term = torch.exp(torch.arange(0, d_model, 2) *
(-math.log(10000.0) / d_model))
```

Fig. 9.3 Illustration of token embedding and positional encoding

```
    pe = torch.zeros(max_len, d_model)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position * div_term)
    self.register_buffer('pe', pe.unsqueeze(0))

def forward(self, x):
    x = x + self.pe[:,:x.size(1),:]
    return x
```

9.3.2 Encoder–Decoder Architecture

The encoder–decoder architecture is one of the core structures of the Transformer model and is widely used in sequence-to-sequence (Seq2Seq) tasks such as machine translation, text summarization, and speech recognition [3]. This architecture consists of two main components: the encoder and the decoder, both composed of multiple identical stacked layers. The overall structure is shown in Fig. 9.4.

The encoder's task is to process the input sequence and generate a contextual representation that captures the information of the input data. The input to the encoder is typically a sequence (such as words or characters), which it transforms into a fixed-size vector that encodes all relevant information from the input sequence. Each encoder layer consists primarily of two sublayers. The first sublayer is the Multi-Head Self-Attention mechanism, which uses self-attention to assign weights to each element in the input sequence, capturing the relationships among different elements. The self-attention mechanism adjusts the representation of each element based on

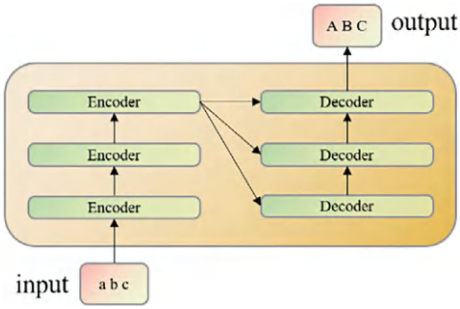


Fig. 9.4 Encoder–decoder architecture of the transformer model

its similarity to other elements in the sequence, allowing the model to understand dependencies within the input.

The second sublayer is a Feed-Forward Neural Network (FFN), composed of two fully connected layers, typically with a ReLU activation function for non-linear transformation. The purpose of the feed-forward network is to further refine each position's representation and produce a new one. The outputs of both sublayers are combined via residual connections and then normalized using Layer Normalization to ensure stable training and efficient gradient flow.

In the multi-layer encoder structure, the input is processed through several stacked encoder layers, ultimately producing a high-dimensional representation that contains the contextual information of the input sequence (commonly referred to as the encoder output or context vector). Each layer updates the representation based on the output from the previous layer, enabling the final output to incorporate various features of the input sequence.

The decoder's task is to generate the output sequence. When producing each element of the output, the decoder relies both on the encoder's output and the elements that have already been generated. The design of the decoder is similar to that of the encoder and consists of multiple layers with the same structure. Each decoder layer also contains three main components:

Multi-Head Self-Attention: Similar to the self-attention mechanism in the encoder, the self-attention in the decoder captures dependencies among different positions within the output sequence. However, since the generation process in the decoder is autoregressive, a masking mechanism is introduced into the self-attention. This ensures that each position can only attend to the current and previous time steps, but not to future outputs, thereby maintaining the causal nature of sequence generation.

Encoder–Decoder Attention: This mechanism allows the decoder to focus on the context vectors generated by the encoder. Specifically, it computes the similarity between the decoder's current state and the encoder's output, and integrates this information with the decoder's internal state to produce the next output. Through this mechanism, the decoder can leverage the global information provided by the encoder to generate a contextually appropriate output sequence.

Feed-Forward Neural Network (FFN): Similar to the feed-forward network in the encoder, the decoder's FFN performs nonlinear transformations on each position's representation within the decoder, generating updated feature representations.

As in the encoder, each layer in the decoder also includes residual connections and layer normalization, which ensure stable and efficient training.

In the Transformer model, the encoder and decoder work closely together to accomplish sequence-to-sequence tasks. First, the input sequence is processed through multiple stacked encoder layers. Each layer uses the self-attention mechanism to capture dependencies among the elements of the input sequence, followed by a feed-forward network to further refine the representations. The encoder ultimately produces a context vector that contains all the critical information from the input sequence.

In the decoding phase, the decoder generates each output element based on the previously generated outputs (captured through self-attention) and the encoder's output (integrated via encoder–decoder attention). In this way, the decoder constructs the output sequence step by step until a predefined end token is reached. Figure 9.5 illustrates the internal structure of the encoder–decoder architecture.

Listing 9.3 Implementation of the Encoder

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class TransformerEncoderBlock(nn.Module):
  def __init__(self, d_model, num_heads, dim_feedforward,
dropout=0.1):
      super().__init__()
      self.mha = MultiHeadAttention(d_model, num_heads)
      self.ffn = nn.Sequential(
        nn.Linear(d_model, dim_feedforward),
        nn.ReLU(),
        nn.Linear(dim_feedforward, d_model)
      )
      self.norm1 = nn.LayerNorm(d_model)
      self.norm2 = nn.LayerNorm(d_model)
```

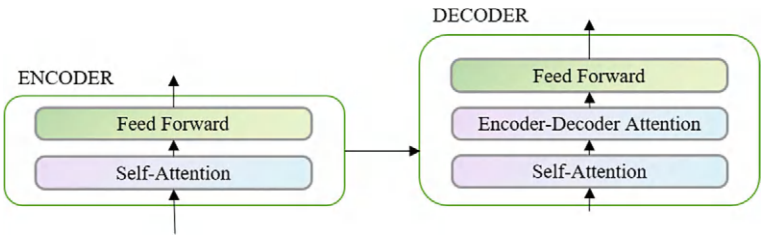


Fig. 9.5 Internal Structure of transformer encoder and decoder

```
79.      self.dropout = nn.Dropout(dropout)
80.
81.    def forward(self, src, mask=None):
82.      attn_output = self.mha(src, src, src, mask)
83.      src = self.norm1(src + self.dropout(attn_output))
84.      ffn_output = self.ffn(src)
85.      src = self.norm2(src + self.dropout(ffn_output))
86.      return src
```

Listing 9.4 Implementation of the Decoder

```
87. import torch
88. import torch.nn as nn
89. import torch.nn.functional as F
90.
91. class TransformerDecoderBlock(nn.Module):
92.   def __init__(self, d_model, num_heads, dim_feedforward,
dropout=0.1):
93.      super().__init__()
94.        self.self_mha = MultiHeadAttention(d_model, num_
heads)
95.      self.enc_dec_mha = MultiHeadAttention(d_model, num_
heads)
96.
97.      self.ffn = nn.Sequential(
98.         nn.Linear(d_model, dim_feedforward),
99.         nn.ReLU(),
100.         nn.Linear(dim_feedforward, d_model)
101.      )
102.
103.      self.norm1 = nn.LayerNorm(d_model)
104.      self.norm2 = nn.LayerNorm(d_model)
105.      self.norm3 = nn.LayerNorm(d_model)
106.      self.dropout = nn.Dropout(dropout)
107.
108.    def forward(self, tgt, memory, tgt_mask=None, memory_
mask=None):
109.      tgt_attn = self.self_mha(tgt, tgt, tgt, tgt_mask)
110.      tgt = self.norm1(tgt + self.dropout(tgt_attn))
111.
112.     enc_dec_attn = self.enc_dec_mha(tgt, memory, memory,
memory_mask)
113.      tgt = self.norm2(tgt + self.dropout(enc_dec_attn))
114.
115.      ffn_output = self.ffn(tgt)
```

```
        tgt = self.norm3(tgt + self.dropout(ffn_output))

        return tgt
```

9.3.3 Residual Connection and Layer Normalization

During the training of deep neural networks, problems such as vanishing gradients and exploding gradients often become more severe as the number of layers increases. These issues can lead to unstable training or even prevent the model parameters from being effectively updated. To address these challenges, the Transformer model incorporates two key techniques into its architecture—Residual Connection and Layer Normalization. These techniques enhance training efficiency and stability while ensuring smooth information flow and effective gradient propagation. Specifically, residual connections alleviate the vanishing gradient problem by allowing gradients to pass directly between layers, while layer normalization accelerates training and improves the model's generalization ability by normalizing inputs at each layer.

Residual Connection, also known as Skip Connection, is a common technique in deep neural networks. It works by directly adding the input of a layer to its output, helping the network transmit gradients more effectively during training and mitigating the issue of vanishing gradients in deep networks. In essence, residual connections allow each layer's output to depend not only on the current layer's computation but also on the original input data. This enables each layer to refine the input rather than fully recomputing it, ensuring uninterrupted information flow and stable gradients. The mathematical form of a residual connection is typically expressed as:

$$y = f(x, \{w_i\}) + x \tag{9.11}$$

Here, x is the input, and $f(x, \{w_i\})$ is the output computed by the current layer. The final output y is obtained by adding xxx and $f(x, \{w_i\})$, which forms the core of the residual connection.

In the Transformer model, residual connections are applied to each sublayer, such as the self-attention layer and the feed-forward network layer. After a sublayer computes its output, the result is added to the original input of that sublayer and then passed on to the next layer. In this way, residual connections ensure the effective flow of information—even as the network becomes deeper, the information can propagate without interruption.

Layer Normalization is a normalization technique primarily used to standardize the inputs of each layer in a neural network. Its core idea is to normalize the activations of each layer so that the outputs have a consistent distribution, leading to a more stable training process. Unlike Batch Normalization, which operates across the batch

dimension, Layer Normalization performs normalization across all features for each individual sample. The computation process is as follows:

$$\hat{x} = \frac{x - \mu}{\sigma} \tag{9.12}$$

Here, x is the input feature vector, μ and σ are the mean and standard deviation of that feature vector, respectively. $\hat{x}$ is the normalized output.

Layer normalization not only accelerates training but also improves the stability of the network, as it eliminates the shift in input distributions across different layers (commonly referred to as "internal covariate shift"). In the Transformer, layer normalization is applied after the output of each sublayer. Specifically, both the output of the self-attention layer and the output of the feed-forward network are followed by a layer normalization step. This ensures that the output of each sublayer maintains a consistent mean and standard deviation, reducing variations in input distribution and making the network easier to train.

Layer normalization is typically used in conjunction with residual connections. The residual connection adds the input to the output of the current layer, and layer normalization then normalizes the result of this addition. The complete operation of each sublayer in the Transformer can be described as:

$$Output = LayerNorm(Input + SubLayerOutput) \tag{9.13}$$

SubLayerOutput refers to the output obtained through the self-attention mechanism or the feed-forward neural network, while *Input* is the input to that sublayer. In the Transformer model, residual connection and layer normalization are two essential components. Residual connections allow the output of each layer to be directly added to its input, ensuring effective information transmission and preventing the vanishing gradient problem. Layer normalization standardizes the input to each layer, eliminating instability caused by variations in input distribution. The combination of these two techniques not only improves the training speed and stability of the Transformer model but also enhances its generalization and performance.

9.3.4 The Complete Transformer Model Architecture

Given an input sequence to the encoder $x_{1:S}$, the output is a vector sequence $H^{enc} = \left[h_1^{enc}, \ldots, h_S^{enc}\right]$. Then, using two matrices, H^{enc} is projected to K^{enc} and V^{enc} as key-value pairs for the decoder, specifically:

$$K^{enc} = W_k' H^{enc}, V^{enc} = W_v' H^{enc} \tag{9.14}$$

Here, W_k' and W_v' are the parameter matrices for linear projection, responsible for mapping the input into appropriate representation spaces.

At decoding step, the previously generated prefix sequence $y_0 : (t-1)$ is first encoded using a self-attention mechanism to obtain $H^{dec} = \left[h_1^{dec}, h_2^{dec}, \ldots, h_t^{dec}\right]$. This encoding process employs a masking mechanism to prevent each position from attending to subsequent tokens, ensuring that the generated sequence remains autoregressive.

The current hidden state h_t^{dec} is then linearly projected to obtain a query vector q_t^{dec}, which is used in a key-value attention mechanism to retrieve relevant information from the encoder outputs K^{enc} and V^{enc}. This mechanism enables the decoder to focus on the most relevant parts of the input sequence.

Each position in the decoder passes through a feed-forward neural network, which applies a nonlinear transformation to integrate all preceding information and generate the final output.

The self-attention module in the decoder is actually a masked self-attention mechanism, designed to ensure the autoregressive property of the generation process. In this setup, the mask prevents each position from attending to future tokens, ensuring that the model can only reason based on already generated content. This mechanism is critical to avoid information leakage from future steps.

The three steps above are repeated multiple times, and the final output probabilities are computed through a fully connected feed-forward network. Figure 9.6 shows

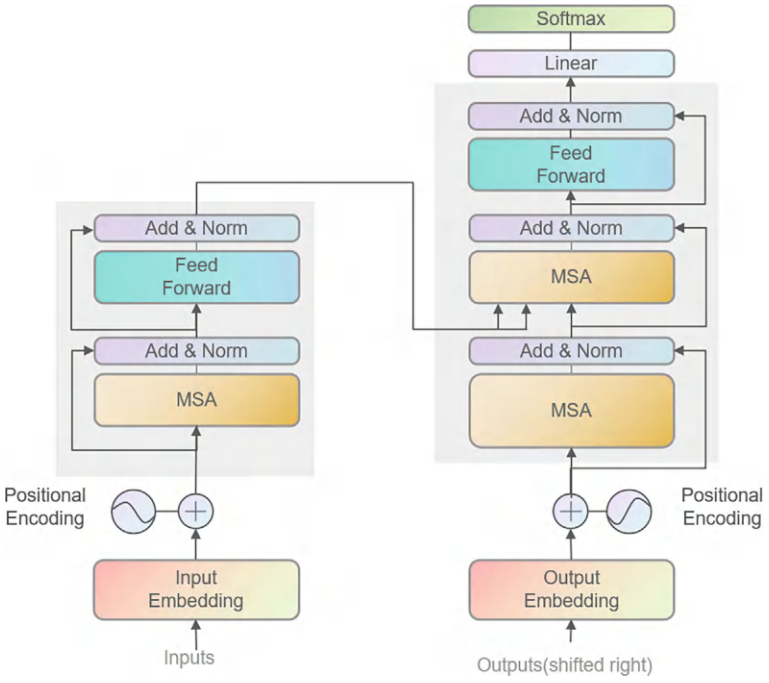


Fig. 9.6 Overall architecture of the transformer model

an example of the Transformer network architecture, “Add and Norm” represents residual connection and layer normalization.

During training, to improve efficiency, we typically feed the right-shifted target sequence $y_0 : (t-1)$ into the decoder, meaning that at step t, the input is from position -1. In this case, a mask is used to block each position from accessing future tokens in the input. This approach is known as masked self-attention.

9.4 Practical Implementation of Transformer

This section demonstrates how to implement a Transformer model based on PyTorch and apply it to a Chinese–English translation task.

Listing 9.5 Importing Required Libraries

```
import re
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
```

Listing 9.6 Building the Vocabulary

```
def build_vocab(lines, is_chinese=True, min_freq=1):
  vocab = {'<pad>': 0, '<sos>': 1, '<eos>': 2, '<unk>': 3}
  word_counts = {}
  for line in lines:
        tokens = list(line.strip()) if is_chinese else
re.findall(r"[\w']+|[^a-zA-Z\s]", line.lower())
     for token in tokens:
       word_counts[token] = word_counts.get(token, 0) +
1
  idx = 4
  for word, count in word_counts.items():
     if count >= min_freq and word not in vocab:
       vocab[word] = idx
       idx += 1
  return vocab
```

Listing 9.7 Custom Dataset Class

```
class TranslationDataset(Dataset):
    def __init__(self, src_lines, tgt_lines, src_vocab, tgt_vocab, max_len=50):
        self.src_lines = src_lines
        self.tgt_lines = tgt_lines
        self.src_vocab = src_vocab
        self.tgt_vocab = tgt_vocab
        self.max_len = max_len

    def __len__(self):
        return len(self.src_lines)

    def __getitem__(self, idx):
                            src_tokens = list(self.src_lines[idx].strip())[:self.max_len]
        tgt_tokens = re.findall(r"[\w']+|[^a-zA-Z\s]", self.tgt_lines[idx].lower())[:self.max_len]
        src_ids = [1] + [self.src_vocab.get(t, 3) for t in src_tokens] + [2]
        tgt_ids = [1] + [self.tgt_vocab.get(t, 3) for t in tgt_tokens] + [2]
        return {
                        'input_ids': torch.tensor(src_ids, dtype=torch.long),
        'labels': torch.tensor(tgt_ids, dtype=torch.long)
        }
```

Listing 9.8 Data Collation Function

```
def collate_fn(batch):
    input_ids = [item['input_ids'] for item in batch]
    labels = [item['labels'] for item in batch]
        input_ids = nn.utils.rnn.pad_sequence(input_ids, batch_first=True, padding_value=0)
    labels = nn.utils.rnn.pad_sequence(labels, batch_first=True, padding_value=0)
    return {'input_ids': input_ids, 'labels': labels}
```

Listing 9.9 Positional Encoding Module

```
class PositionalEncoding(nn.Module):
  def __init__(self, d_model, max_len=500):
    super().__init__()
    pe = torch.zeros(max_len, d_model)
                position = torch.arange(0, max_
len).unsqueeze(1).float()
        div_term = torch.exp(torch.arange(0, d_model,
2).float() * (-torch.log(torch.tensor(10000.0)) / d_model))
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position * div_term)
    self.register_buffer('pe', pe.unsqueeze(0))

  def forward(self, x):
    return x + self.pe[:,:x.size(1)]
```

Listing 9.10 Transformer Encoder–Decoder Model

```
class MyTransformerEncoderDecoder(nn.Module):
    def __init__(self, src_vocab_size, tgt_vocab_size,
embed_dim=128, num_heads=4, num_layers=1, dropout=0.1):
     super().__init__()
    self.src_embed = nn.Embedding(src_vocab_size, embed_
dim)
    self.tgt_embed = nn.Embedding(tgt_vocab_size, embed_
dim)
     self.pos_encoder = PositionalEncoding(embed_dim)
         encoder_layer = nn.TransformerEncoderLayer(d_
model=embed_dim, nhead=num_heads, dropout=dropout, batch_
first=True)
         decoder_layer = nn.TransformerDecoderLayer(d_
model=embed_dim, nhead=num_heads, dropout=dropout, batch_
first=True)
    self.encoder = nn.TransformerEncoder(encoder_layer,
num_layers)
    self.decoder = nn.TransformerDecoder(decoder_layer,
num_layers)
     self.output_proj = nn.Linear(embed_dim, tgt_vocab_
size)

   def forward(self, input_ids, labels=None):
     src = self.pos_encoder(self.src_embed(input_ids))
     memory = self.encoder(src)
     if labels is not None:
```

```
        tgt = self.pos_encoder(self.tgt_embed(labels[:, :-1]))
            tgt_mask = nn.Transformer.generate_square_subsequent_mask(tgt.size(1)).to(tgt.device)
        out = self.decoder(tgt, memory, tgt_mask=tgt_mask)
        logits = self.output_proj(out)
            loss = F.cross_entropy(logits.view(-1, logits.size(-1)), labels[:, 1:].reshape(-1), ignore_index=0)
        return type('Output', (), {'loss': loss, 'logits': logits})
    else:
            raise NotImplementedError("Please perform inference using the generate method.")

def generate(self, input_ids, max_length=50, sos_id=1, eos_id=2):
    self.eval()
    with torch.no_grad():
        memory = self.encoder(self.pos_encoder(self.src_embed(input_ids)))
            ys = torch.ones(input_ids.size(0), 1, dtype=torch.long).fill_(sos_id).to(input_ids.device)
        for _ in range(max_length):
            tgt = self.pos_encoder(self.tgt_embed(ys))
            tgt_mask = nn.Transformer.generate_square_subsequent_mask(tgt.size(1)).to(tgt.device)
            out = self.decoder(tgt, memory, tgt_mask=tgt_mask)
            next_token = self.output_proj(out[:, -1, :]).argmax(dim=-1).unsqueeze(1)
            ys = torch.cat([ys, next_token], dim=1)
            if (next_token == eos_id).all():
                break
        return ys
```

Listing 9.11 Model Training Function

```
def train(model, dataloader, optimizer, device, epochs=10):
    model.train()
    accumulation_steps = 4
    for epoch in range(epochs):
        total_loss = 0
        optimizer.zero_grad()
        for i, batch in enumerate(dataloader):
            input_ids = batch['input_ids'].to(device)
            labels = batch['labels'].to(device)
```

```
        outputs = model(input_ids, labels=labels)
        (outputs.loss / accumulation_steps).backward()
        total_loss += outputs.loss.item()
        if (i + 1) % accumulation_steps == 0 or (i + 1) ==
len(dataloader):
          optimizer.step()
          optimizer.zero_grad()
        print(f"Epoch {epoch + 1} | Loss: {total_loss /
len(dataloader):.4f}")
    torch.cuda.empty_cache()
```

Listing 9.12 Translation Function

```
def translate(model, sentence, src_vocab, tgt_vocab_inv,
device, max_length=50):
  model.eval()
    src_ids = [1] + [src_vocab.get(t, 3) for t in
list(sentence.strip())] + [2]
            input_tensor = torch.tensor(src_ids,
dtype=torch.long).unsqueeze(0).to(device)
     output_ids = model.generate(input_tensor, max_
length=max_length).squeeze().tolist()
  return ' '.join([tgt_vocab_inv.get(i, '<unk>') for i in
output_ids if i not in [0, 1, 2]])
```

Listing 9.13 Main Program Execution

```
if __name__ == "__main__":
     device = torch.device("cuda" if torch.cuda.is_
available() else "cpu")
  print("Current device:", device)

  with open('chinese.zh', 'r', encoding='utf-8') as f:
    ch_lines = f.readlines()
  with open('english.en', 'r', encoding='utf-8') as f:
    en_lines = f.readlines()

  src_vocab = build_vocab(ch_lines, is_chinese=True)
  tgt_vocab = build_vocab(en_lines, is_chinese=False)
  tgt_vocab_inv = {v: k for k, v in tgt_vocab.items()}

  dataset = TranslationDataset(ch_lines, en_lines, src_
vocab, tgt_vocab)
```

```
dataloader = DataLoader(dataset, batch_size=2,
collate_fn=collate_fn, shuffle=True)

model = MyTransformerEncoderDecoder(len(src_vocab),
len(tgt_vocab)).to(device)
optimizer = torch.optim.Adam(model.parameters(),
lr=5e-4)

train(model, dataloader, optimizer, device, epochs=20)

# Test sentence
test_sentences = ["人工智能改变世界"]
for sent in test_sentences:
translation = translate(model, sent, src_vocab, tgt_
vocab_inv, device)
print(f" Chinese: {sent}").
print(f" Translation: {translation}\n").
```

Figure 9.7 shows the loss curve during the training of the model. Figure 9.8 presents the output results generated by the Transformer model.

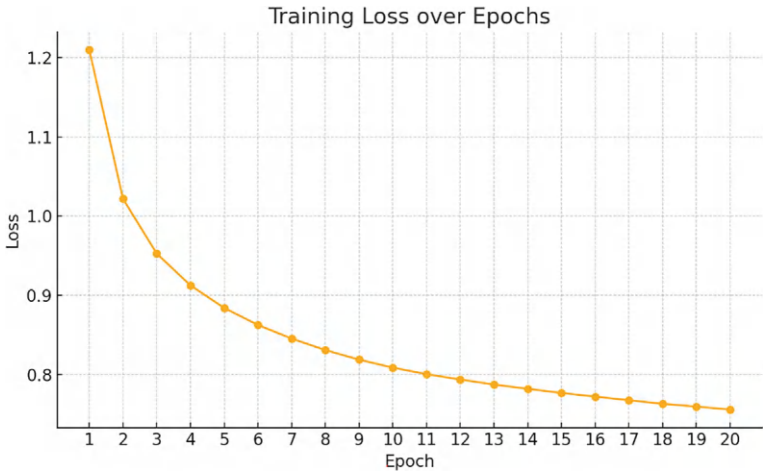


Fig. 9.7 Training loss curve

```
chinese: 人工智能改变世界
Translation: artificial intelligence can change the world
```

Fig. 9.8 Model testing results

9.5 Applications of Transformer in Molecular Science

In recent years, the application of Transformer models in molecular science has rapidly expanded, making them powerful tools for handling molecular data. Originally developed in the field of natural language processing, the self-attention-based architecture of Transformers is now widely used in various subfields such as chemistry, drug design, protein engineering, and bioinformatics. The emergence of Transformers has greatly enhanced the efficiency and intelligence of research in molecular science.

9.5.1 Application of Transformer in Molecular Property Prediction

Molecular property prediction is a fundamental task in fields such as drug discovery and materials science. It involves modeling the physicochemical properties of small molecules (such as solubility, LogP), toxicological attributes (such as toxicity, biocompatibility), and biological activities (such as target binding affinity). Traditional methods rely on molecular descriptors and molecular fingerprints combined with machine learning algorithms. In recent years, deep learning—especially graph neural networks—has significantly improved prediction accuracy. Transformer models have further advanced this field; their self-attention mechanisms can represent molecular data as a "molecular language," effectively capturing complex relationships in sequences (such as SMILES [4], SELFIES [5]) and molecular graph structures.

In practical applications, encoder-based Transformer models such as BERT and its chemistry-specific variants (e.g., Mol-BERT, FP-BERT [6]) are particularly popular. Researchers typically use large-scale unlabeled molecular data for self-supervised pretraining (such as masked language modeling), followed by fine-tuning for specific property predictions like solubility, activity, and toxicity. Common data sources include PubChem [7] and ChEMBL [8], which cover tens of millions to over a hundred million molecules.

Furthermore, the scale of pretraining data has a significant impact on model performance, although the improvement tends to plateau beyond a certain point. The powerful sequence modeling and global information capturing abilities of Transformers are continuously pushing the boundaries of molecular property prediction, providing a solid foundation for molecular screening and virtual drug discovery.

9.5.2 Application of Transformer in Molecular Generation and Optimization

In addition to molecular property prediction, Transformers are also widely applied to de novo molecule generation—that is, the automatic design of compounds with novel structures and desirable properties. Traditional generative methods include rule-based assembly, evolutionary algorithms, variational autoencoders, and generative adversarial networks. Transformers, however, introduce a new paradigm as "molecular language models." By autoregressively learning from large-scale molecular sequence data, they possess stronger abilities in structure generation and modeling chemical validity.

Common approaches include autoregressive and sequence-to-sequence Transformers such as GPT and BART, which generate new molecules character by character on SMILES or SELFIES during training. To ensure chemical validity, many studies adopt SELFIES encoding; for example, the SELF-BART model has significantly improved both the validity and diversity of generated molecules.

Transformers are also used for molecular structure optimization, such as through conditional generation (incorporating target property information) or sequence-to-sequence translation, enabling automatic optimization of lead compounds. Transformer models have achieved target-oriented or property-constrained molecular optimization. Pretrained generative models like Chemformer [9] and MolGPT [10] outperform traditional methods on metrics such as molecular validity, novelty, and drug-likeness, providing powerful tools for intelligent drug design and virtual chemical space exploration.

9.5.3 Application of Transformer in Molecular Docking and Interaction Prediction

Molecular docking and molecular interaction prediction (such as drug-target interactions and protein–protein interactions) are key steps in modern drug discovery. Traditional methods mostly rely on physical scoring, handcrafted features, and machine learning, which struggle to fully capture complex structures and long-range interactions.

Transformer models offer a new approach to these tasks. They can simultaneously encode small molecule and protein sequences, integrating molecular graph and spatial structural information. For example, in MolTrans, the model divides small molecule and protein sequences into fragments, uses Transformers to extract contextual semantics for each, and then employs attention mechanisms to model interactions between fragments, enabling more accurate and interpretable binding

predictions. Models like Graph Transformers are also used to integrate molecular topology and 3D structure, further improving the accuracy of interaction prediction.

With the integration of structural prediction tools like AlphaFold2 [11] and large-scale biomolecular data, Transformers are playing an increasingly important role in virtual screening, drug repurposing, and elucidation of molecular mechanisms, driving molecular science and intelligent drug design to new heights.

9.6 Advantages and Limitations of Transformer

Thanks to its unique self-attention mechanism and parallel computation capabilities, the Transformer model has become a mainstream deep learning architecture across multiple fields, including natural language processing, computer vision, and molecular science. In molecular modeling, Transformer excels at capturing complex long-range dependencies and flexibly adapting to various input formats—such as molecular sequences, graph structures, and 3D coordinates—supporting a wide range of tasks from molecular property prediction and molecule generation to protein–ligand interaction modeling.

Literature has shown that Transformer models exhibit strong generalization ability in molecular language modeling, particularly when pretrained on large-scale data. Their generative variants have also demonstrated impressive capacity to design novel compounds with desirable properties across chemical space.

However, the model still faces several limitations. First, its computational complexity grows quadratically with input sequence length, resulting in resource bottlenecks when processing long molecular sequences or high-resolution structures. Second, compared to graph neural networks, standard Transformers lack an inductive bias toward molecular graph topology, which limits their ability to capture structural relationships. In addition, current positional encoding strategies have limited expressiveness in 3D molecular space and struggle to accurately model spatial relations between atoms.

In generative tasks, Transformer models may also suffer from weak controllability and unstable outputs, necessitating enhancements through conditional generation, reinforcement learning, or other control mechanisms.

In summary, while Transformer models show great potential and broad adaptability in molecular science, their performance still relies on further architectural refinement and task-specific design. Future research may focus on efficient attention mechanisms, structure-aware modeling, and multimodal information integration to continue expanding their impact.

References

1. Vaswani, A., et al.: NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 5998–6008
2. Graves, A.: Long short-term memory. Super. Seq. Label. Recurr. Neural Netw. **385**, 37–45 (2012)
3. Sutskever, I.V., Le, Q.V.: NIPS'14: Proceedings of the 28th International Conference on Neural Information Processing Systems, pp. 3104–3112
4. Weininger, D.: SMILES a chemical language and information system. J. Chem. Inf. Comput. Sci. **28**, 31–36 (1988)
5. Krenn, M., Häse, F., Nigam, A.K., Friederich, P., Aspuru-Guzik, A.: Self-referencing embedded strings (SELFIES): a 100% robust molecular string representation. Mach. Learn. Sci. Technol. **1**, 045024 (2020)
6. Wen, N., et al.: A fingerprints based molecular property prediction method using the BERT model. J. Cheminform. **14**, 71 (2022)
7. Lu, J., Zhang, Y.: Unified deep learning model for multitask reaction predictions with explanation. J. Chem. Inf. Model. **62**, 1376–1387 (2022)
8. Gaulton, A., et al.: ChEMBL: a large-scale bioactivity database for drug discovery. Nucl. Acids Res. **40**, D1100–D1107 (2012)
9. Irwin, R., Dimitriadis, S., He, J., Bjerrum, E.J.: Chemformer: a pre-trained transformer for computational chemistry. Mach. Learn. Sci. Technol. **3**, 015022 (2022)
10. Bagal, V., Aggarwal, R., Vinod, P.K., Priyakumar, U.D.: MolGPT: molecular generation using a transformer-decoder model. J. Chem. Inf. Model. **62**, 2064–2076 (2022)
11. Jumper, J., et al.: Highly accurate protein structure prediction with AlphaFold. Nature **596**, 583–589 (2021)

Chapter 10
Python Programming and Computational Environment Setup

Python is a simple yet efficient programming language that is widely used in artificial intelligence. Well-written Python code reads almost like plain English, allowing you to focus on problem-solving rather than on syntactic details. Python also provides a comprehensive standard library; when specific functionality is required, it can often be implemented by importing relevant modules with minimal code, which simplifies development. With its concise syntax and robust ecosystem, Python accelerates development cycles and integrates well with C/C++. Performance-critical components can be implemented as C/C++ extensions and invoked from Python, a capability that is particularly valued in data science. These advantages underpin our choice of Python as the primary language for artificial intelligence aided drug discovery.

10.1 Introduction to Python

Python was created by Guido van Rossum at the Centrum Wiskunde and Informatica (CWI) in the Netherlands in the early 1990s as a successor to the ABC language. Python offers efficient high-level data structures and supports object-oriented programming. Its clear syntax, dynamic typing, and interpreted execution model make it well suited to scripting and rapid application development across many platforms. With continuous updates and new features introduced in each release, Python has evolved into a widely adopted language for standalone and large-scale projects.

Because of its simplicity, readability, and extensibility, the use of Python in scientific computing has steadily increased worldwide. Many universities now teach introductory programming courses using Python. In addition, many open-source scientific computing packages provide Python bindings, such as OpenCV, VTK, and ITK. A large ecosystem of Python libraries is also available, including the widely used trio of NumPy, SciPy, and Matplotlib, which provide efficient array computing, numerical

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_10

routines, and plotting, respectively. Consequently, the ecosystem built around Python and its extension libraries is well suited for engineers and researchers to process experimental data, generate figures, and develop scientific computing applications.

The Python language has several major features, including:

(1) Easy to learn. Python is extremely easy to get started with because it has very simple documentation.
(2) Object-oriented. Python is a fully object-oriented language. Functions, modules, numbers, and strings are all objects. It fully supports inheritance, overloading, derivation, and multiple inheritance, which is beneficial for enhancing the reusability of source code.
(3) Portability. Due to its open-source nature, Python has been ported to many platforms (modified to run on different systems). These platforms include Linux, Windows, FreeBSD, macOs, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acom RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE, PocketPC, Symbian, and Google's Android platform based on Linux.
(4) Interpreted nature. A program written in a compiled language such as C or C++ can be transformed from a source file into an executable program for the computer. This process is accomplished through a compiler and various flags and options. When the program is run, a loader software copies the program from the hard disk to the memory and executes it. However, a program written in Python does not need to be compiled into binary code; it can be run directly from the source code. Internally in the computer, the Python interpreter converts the source code into an intermediate form called bytecode, and then translates it into machine language used by the computer for execution.
(5) Open source. Python is one of the FLOSS (Free/Open Source Software). Users can freely distribute copies of this software, read its source code, make modifications to it, and use parts of it in new free software.
(6) High-level language. When programming in Python, there is no need to consider many low-level matters, such as how to manage the memory used by the program and other low-level details.
(7) Extensibility. If a critical piece of code needs to run faster or if certain algorithms are not to be disclosed, they can be written in C or C++ and then used within a Python program.
(8) Rich libraries. The Python standard library is extensive and can assist with a wide range of tasks, including regular expressions, document generation, unit testing, threading, database operations, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interface), Tk, and other system-related operations. This is known as Python's "batteries included" philosophy. In addition to the standard library, there are many other high-quality libraries available, such as wxPython, Twisted, and the Python Imaging Library.
(9) Standardized code. Python enforces indentation, which enhances code readability and makes programs clearer and more visually organized.

10.2 Brief Introduction of Python

When running Python programs, we need first to check if your computer has a newer version of Python installed. If not, we need to install it first. Additionally, we also need a Python editor to facilitate writing and running Python programs. When inputting Python code, an integrated development environment can recognize them and highlight different parts, allowing you to easily understand the structure of the code.

Every programming language keeps evolving with the introduction of new concepts and technologies, and Python developers have been constantly striving to do the same to enrich and enhance its functionality. The version of Python used in this book is Python 3. Interested readers may also refer to other books [1].

10.2.1 Introduction to Anaconda

When you start using Python, there are many programs available for writing code. For instance, after downloading Python and completing the installation, you will get an IDE with a graphical user interface (GUI). Additionally, you can download IPython Notebook to write code in a web-based interactive environment. If you are working on a macOS system or have Cygwin installed on a Windows system, you can use built-in text editors such as Nano, Vim or Emacs in the terminal window to write code.

However, in this book, we do not choose to download and install Python directly from the official Python website. Instead, we opt to install Anaconda, which comes with Python. Compared to other versions, it offers many advantages for beginners and is equally suitable for experienced programmers. The main advantage of Anaconda is that it pre-installs some of the most popular Python add-on modules, so you don't have to install these modules and their dependencies one by one.

Anaconda is a packaged collection that includes over 700 open-source packages related to data science, covering various aspects such as data visualization, machine learning, and deep learning. It is an integrated management tool or system for Python, integrating all the packages needed for Python-based data computing and analysis. It can not only be used for data analysis but also in the fields of big data and artificial intelligence. Additionally, after installing Anaconda, Python, IPython, Jupyter Notebook, and the integrated development environment Spyder are installed by default. Installing Anaconda saves a lot of time spent on downloading module packages and makes usage more convenient.

Another advantage of Anaconda is its cross-platform nature—it has three versions for Linux, macOS and Windows. Therefore, if you become familiar with its usage on the Windows system, you can still use the familiar interface when switching to the macOS system.

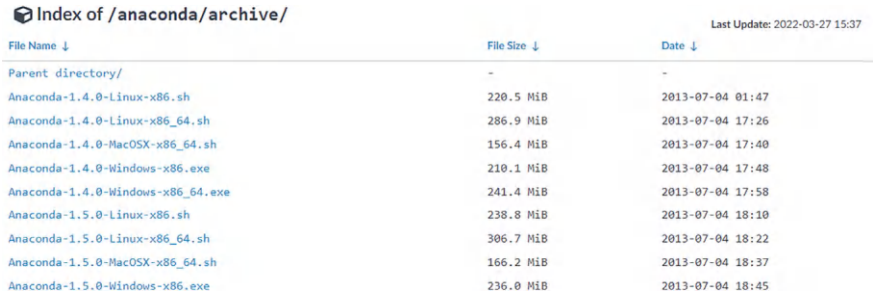
If you are already familiar with Python and all the available additional packages, there is one thing to note when using Anaconda: the syntax for installing additional packages is a bit different. In Anaconda, you need to use the 'conda install' command. For example, to install the additional package 'argparse', you should enter 'conda install argparse'. This syntax is different from the usual 'pip install'. If you download and install Python from the Python official website, you should use 'python—m pip install argparse' to install the 'argparse' package. Anaconda also allows you to use the 'pip install' syntax. In actual use, you can use either method, but when you are learning how to install additional packages, you should be aware of this minor difference.

10.2.2 Installing Anaconda

(1) Download. Since the Anaconda official website is a foreign site, the download speed is very slow. It is recommended to use the Tsinghua University Open Source Software Mirror Station, as shown in Fig. 10.1, to download the corresponding Anaconda3 version.
(2) Installation. Open the downloaded software and follow the steps to install it. Click the Next button in Fig. 10.2; in the Select Installation Type dialog box that pops up, select the All Users radio button and click Next, as shown in Fig. 10.3; in the Choose Install Location dialog box that appears, select the installation path (you can choose a partition other than C:).

Click the Next button, as shown in Fig. 10.4; in the pop-up Advanced Installation Options dialog box, select the Register Anaconda as the system Python 3.7 checkbox, and then click the Install button, as shown in Fig. 10.5.

The successful installation interface is shown in Fig. 10.6. Click the Next button, and in the pop-up Microsoft Visual Studio Code Installation dialog box, click the Skip button, as shown in Fig. 10.7.


Index of /anaconda/archive/

Last Update: 2022-03-27 15:37

File Name ↓	File Size ↓	Date ↓
Parent directory/	-	-
Anaconda-1.4.0-Linux-x86.sh	220.5 MiB	2013-07-04 01:47
Anaconda-1.4.0-Linux-x86_64.sh	286.9 MiB	2013-07-04 17:26
Anaconda-1.4.0-MacOSX-x86_64.sh	156.4 MiB	2013-07-04 17:40
Anaconda-1.4.0-Windows-x86.exe	210.1 MiB	2013-07-04 17:48
Anaconda-1.4.0-Windows-x86_64.exe	241.4 MiB	2013-07-04 17:58
Anaconda-1.5.0-Linux-x86.sh	238.8 MiB	2013-07-04 18:10
Anaconda-1.5.0-Linux-x86_64.sh	306.7 MiB	2013-07-04 18:22
Anaconda-1.5.0-MacOSX-x86_64.sh	166.2 MiB	2013-07-04 18:37
Anaconda-1.5.0-Windows-x86.exe	236.0 MiB	2013-07-04 18:45

Fig. 10.1 Relevant versions of Anaconda3 on the open source software mirror site

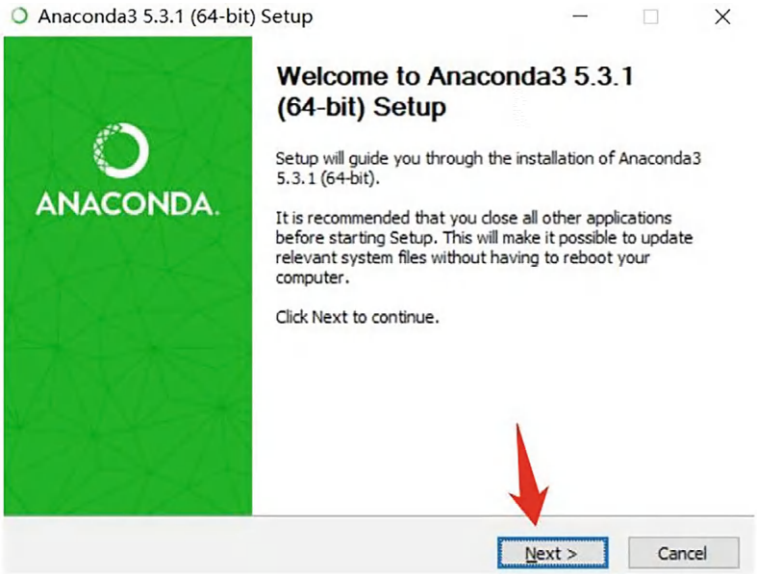


Fig. 10.2 Installation settings of Anaconda3 5.3.1

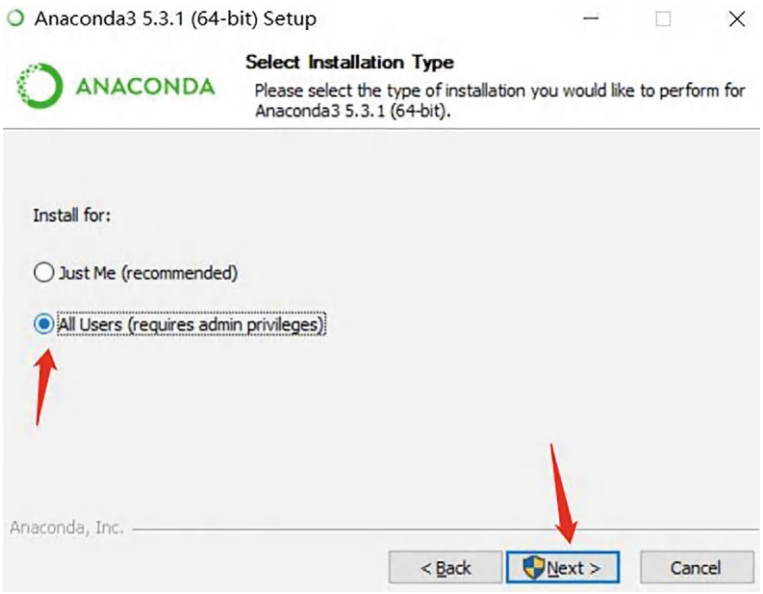


Fig. 10.3 Setting the installation type

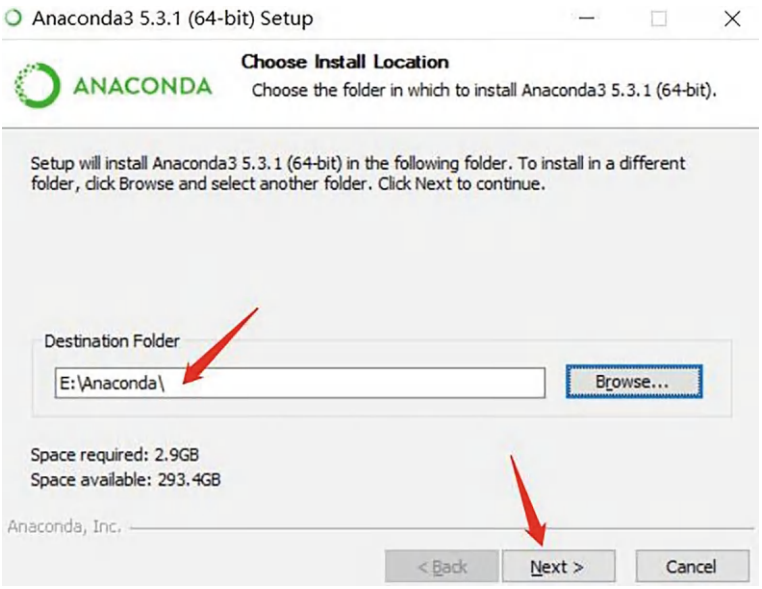


Fig. 10.4 Setting installation path

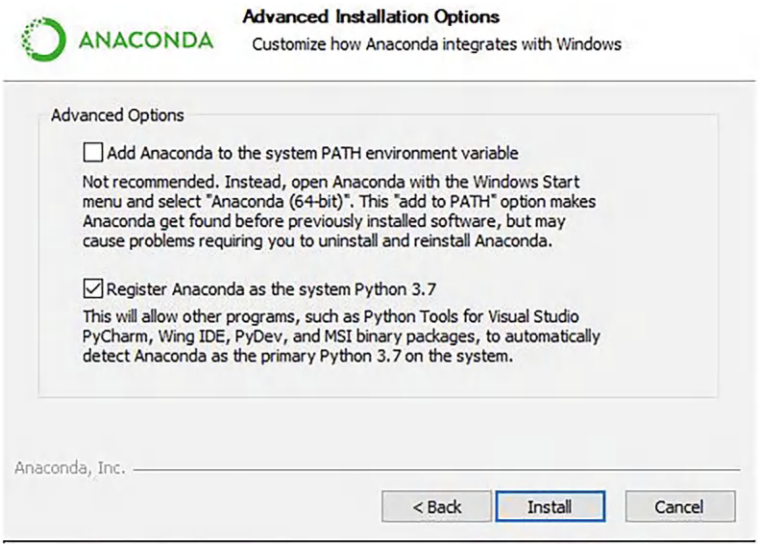


Fig. 10.5 Advanced installation option

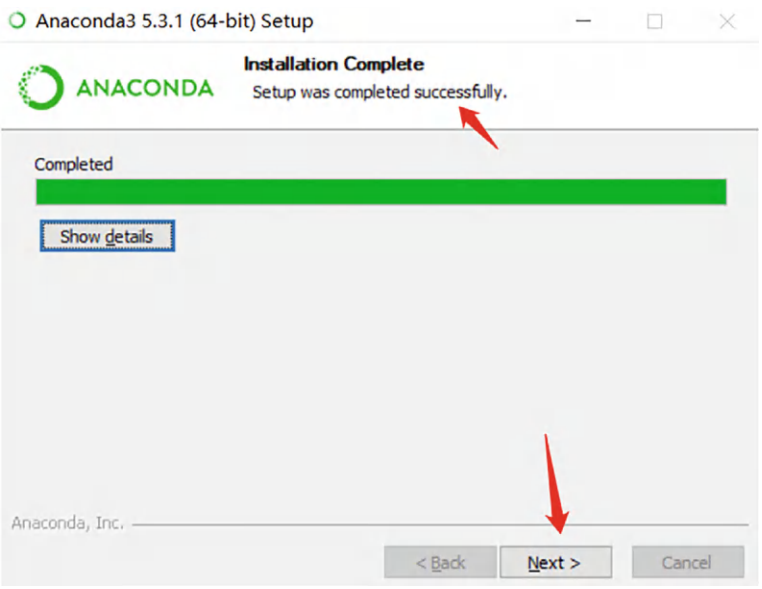


Fig. 10.6 Installation interface

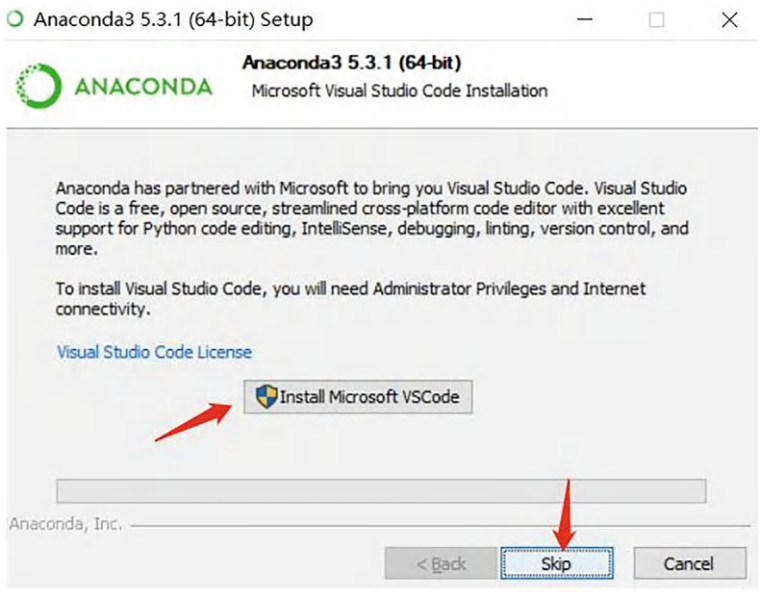


Fig. 10.7 Microsoft Visual Studio Code installation

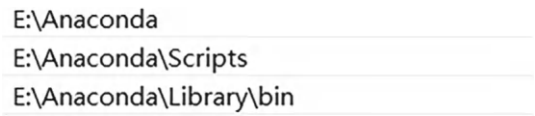


Fig. 10.8 Configuring the environment variable path

(3) Configure environment variables. Right-click the "This PC" icon, select "Properties" from the pop-up shortcut menu, choose "Advanced system settings" in the popped-up window, click the "Environment Variables" button in the "System Properties" dialog box that appears, select "Path" in the "Environment Variables" dialog box that pops up, and click the "Edit" button. In the "Edit Environment Variable" dialog box that opens, click the "New" button and add the path where you installed Anaconda according to the format shown in Fig. 10.8.
(4) Check if the installation was successful. Enter "python" in the command window to see if the Python environment is available, as shown in Fig. 10.9.

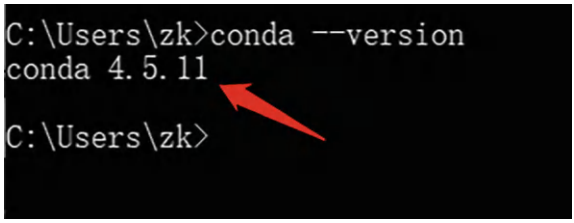
In the command window, enter 'conda—version' to check if there is a conda environment, as shown in Fig. 10.10.

If all the above operations are successful, you can then use Python in Anaconda.

Then, just enter the commands one by one, and Python will execute them in sequence.

Fig. 10.9 Checking the Python environment

Fig. 10.10 Verify the conda environment



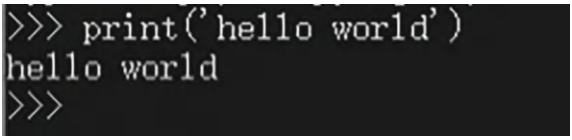


Fig. 10.11 Terminal running Python Code

10.2.3 Running Python Code Snippets in the Terminal Window

Python comes with an interpreter that can be run in a terminal window, allowing you to try out Python code snippets without having to save and run an entire program. Many books and online tutorials about Python demonstrate how to run code in the Python shell. To run Python code in this way, you need to first open a command prompt window (on Windows) or a terminal window (on macOS), type "python", and press Enter. You will then see the Python prompt (">>>").

Then, just enter the commands one by one, and Python will execute them in sequence.

For example, as shown in the code snippet in Fig. 10.11, the prompt indicates that the terminal window is being used, and the text following represents the code that needs to be executed by pressing the Enter key.

This method of running code is simple and interesting, but it becomes less suitable as the number of lines of code increases. When your task requires multiple lines of code to complete, a more convenient way is to write all the code in a text file called a Python script and then run this script. The following introduces the method of creating a Python script.

10.2.4 Introduction to PyCharm

PyCharm is a Python Integrated Development Environment (IDE) [2], equipped with a full set of tools that can enhance users' efficiency when developing with the Python language, such as debugging, syntax highlighting, project management, code navigation, intelligent prompts, auto-completion, unit testing, and version control. Additionally, this IDE offers some advanced features to support professional web development under the Django framework.

PyCharm supports both Google App Engine and IronPython. With the support of advanced code analysis programs, these features make PyCharm a powerful tool for both professional and novice Python developers.

After mastering PyCharm, you can continue to use it to write complex large-scale projects. The Professional Edition requires a paid license, but many developers find the Community Edition to be very useful as well.

PyCharm provides a linter that checks whether the code adheres to generally accepted Python programming conventions and offers suggestions for improvement

when the code does not conform to Python code formatting settings. It integrates a debugger designed to help you efficiently eliminate errors and supports various modes, enabling you to work efficiently with numerous popular Python libraries.

10.2.5 Configuring PyCharm

(1) After installing a newer version of Python and PyCharm, you can start writing and running your first Python program. Before doing so, you need to configure PyCharm to ensure it uses the correct Python version on your system. Once that's done, you can start writing and running your programs.

Open the software and click File → Settings in the menu bar in sequence, as shown in Fig. 10.12. In the Settings window that pops up, select Project Project1 → Python Interpreter, as shown in Fig. 10.13.

(2) Click the small gear icon in the upper right corner of the Project Interpreter interface, then click the "Show All" button in the pop-up options. Next, click the button in the window that pops up, as shown in Fig. 10.14.
(3) In the "Add Python Interpreter" interface, select "New environment" or "Existing environment". It is recommended to select the "Existing environment" radio button, then find the python.exe based on the path where you installed Anaconda. Next, check the "Make available to all projects" checkbox to apply this Python environment to all projects. Click the "OK" button, as shown in Fig. 10.15.

After the settings are completed, you can view the library information included in the current configured Python environment in the interface shown in Fig. 10.16. Click the OK button to complete the configuration of the PyCharm environment.

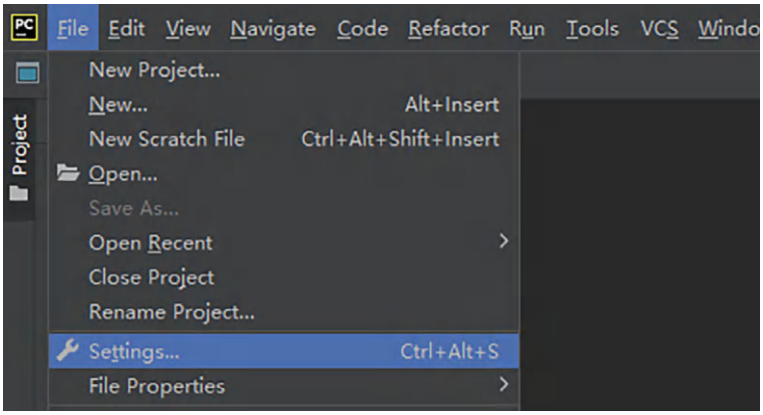


Fig. 10.12 Settings menu of PyCharm

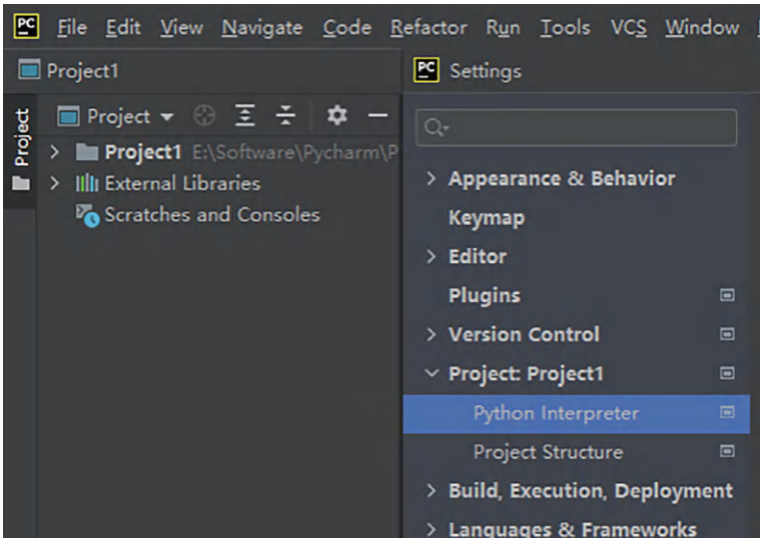


Fig. 10.13 Selecting the Python interpreter

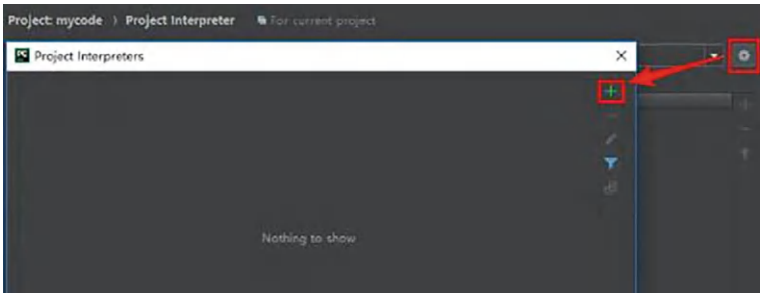


Fig. 10.14 Configuration interface

After successfully configuring the above steps, you can run Python Code in PyCharm.

10.2.6 Writing Python Scripts in PyCharm

Before writing your first program, you need to create a folder named python_work in PyCharm to store the projects you develop. It's best to use lowercase letters for file and folder names and replace spaces with underscores, as this is the naming convention in Python [3].

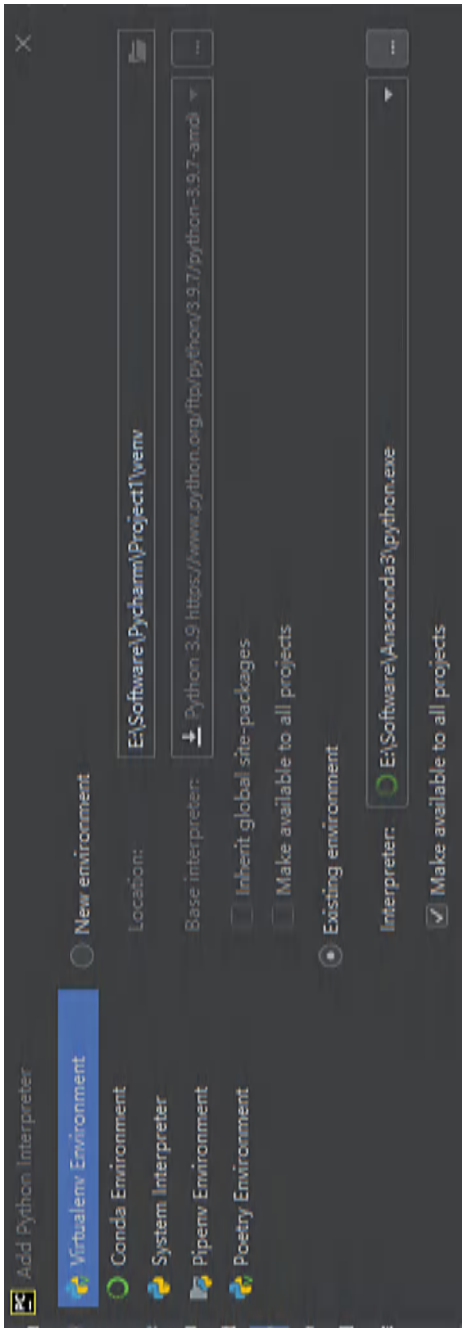


Fig. 10.15 Selecting the Python environment

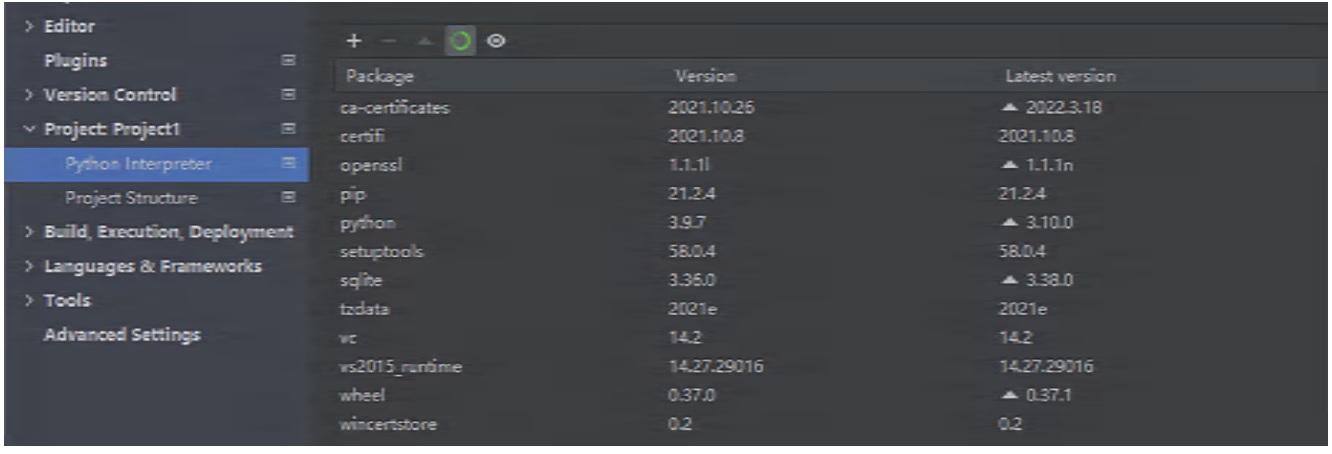


Fig. 10.16 Library information in the Python environment

Next, right-click on the newly created folder, and in the pop-up shortcut menu, select New → Python File, and name it hello world. By using the file extension.py, you inform PyCharm that the code in the file is written in Python, which enables it to know how to run the program and highlight the code in an effective way.

After the file is created, you can start entering the code in it, as shown in Fig. 10.17.

If you cannot see the above interface, it indicates that there are some issues with the program. Please check if you have used capital letters in the naming or if you have omitted the quotation marks in the parentheses. The syntax of programming languages is strict, and any non-compliance will result in an error.

10.3 Basic Elements of the Python Language

To delve deeper into learning and using Python [4], it is also necessary to understand the basic elements of the Python language. By mastering more fundamental elements, you will have a deeper understanding of Python and be able to apply this knowledge comprehensively in the subsequent chapters to complete specific data processing tasks [5].

10.3.1 Basic Data Types

1. Numerical value

In programming, we often use numbers to record scores, represent visualizations, store information for web applications, etc. Python can handle numbers in different ways depending on their usage. The four main numerical types in Python are integers, floating-point numbers, long integers, and complex numbers, among which integers and floating-point numbers are the two most commonly used types.

(1) Integer. Examples of integers are shown in Listing 10.1.

Listing 10.1 Integer Examples

```
1.x=9
2.print('#1:[1] '.format(x))      # Print the variable x and
output 9.
3.print('#2:[2] '.format(3*4))      # Print the calculation
result of 3*4 and output 12.
4.print('#3:[3] '.format(int(8.3)/int(2.7))    # Convert a
decimal number into an integer using int and perform division
operation, output 4.
```

Fig. 10.17 Writing a Python script

(2) Floating-point numbers. Examples of floating-point numbers are shown in Listing 10.2.

Listing 10.2 Examples of Floating-Point Numbers

```
1.print('#4:{0:.3f}'.format(8.3/2.7))      # output 3.074
2.y = 2.5*4.8
3.print('#5:{0:.1f}'.format(y))           # output 12.0
4.r = 8/float(3)
5.print('#6:{0:.2f}'.format(r))           # output 2.67
6.print('#7:{0:.4f}'.format(8.0/3))        # output 2.6667
```

#4 is very similar to #3, except that the two numbers being divided are kept as floating-point numbers. 8.3 divided by 2.7 gives an output result of 3.074. The "0.3 f" specifies that the printed output value should have three decimal places.

#5 indicates multiplying 2.5 by 4.8, assigning the result to the variable *y*, and then printing the result with one decimal place. The product of these two floating-point numbers is 12, so the printed value is 12.0. #6 and #7 represent two ways of calculating 8 divided by 3, both resulting in a floating-point number approximately equal to 2.67.

2. String

A string is a basic data type in Python. You can think of it this way: it usually refers to something that humans can read. More broadly speaking, it is a sequence of characters, and the characters only have meaning when they form this sequence. String type data is present in many business applications, such as the names and addresses of suppliers and customers, evaluation and feedback data, event logs, and document records. Some objects may seem like integers but are actually strings. For instance, postal codes. Postal code 11111 is different from the integer 1111. You cannot perform arithmetic operations on postal codes, so it is best to handle them as strings in the code.

Strings can be enclosed in single quotes, double quotes, three single quotes, or three double quotes. Simple string examples are shown in Listing 10.3.

Listing 10.3 String Examples

```
1.print('hello world!')
2.print("hello world!")
3.print("'hello world!"')
4.print("""hello world!""")
```

3. List

A list consists of a series of elements arranged in a specific order. You can create a list containing all the letters of the alphabet, all the numbers, or all the names of family members. You can also add anything to a list, and the elements within it do not need to have any relationship. Lists usually contain multiple elements, so it's a good idea to give them a plural name (such as letters, names). In Python, lists are represented by square brackets [4], and the elements within them are separated by commas.

Many business analyses use lists, such as various customer lists, product lists, asset lists, sales volume lists, etc. However, the list in Python (an ordered collection of objects) is much more flexible. The lists mentioned above contain similar objects (for example, strings containing customer names or floating-point numbers representing sales volumes), but Python lists are not that simple. They can contain any combination of numbers, strings, other lists, tuples, and dictionaries (introduced later in this chapter). Because lists are widely used, highly flexible, and play a significant role in business applications, it is extremely important to master how to operate lists in Python.

The code for creating the list is shown in Listing 10.4

Listing 10.4 Creating a List

```
1.”’Method1”’
2.list1 = [’hello’, ’world’, 123]     # Create a list
3.print(list1)                  # Output the list content
4.print(type(list1))              # Output type
5.”’Method 2: using the built-in function list()”’
6.list2 = list([’hello’, ’world’])
7.print(list2)
8.print(type(list2))
```

4. Tuple

Tuples are very similar to lists in all aspects except that they cannot be modified. Because tuples cannot be modified, there are no functions for modifying them. You might wonder why two such similar data structures were designed. This is because tuples have important functions that unchangeable lists cannot achieve, such as being used as dictionary keys.

Python refers to values that cannot be modified as immutable, and immutable lists are called tuples. Tuples look very much like lists, but they are identified by parentheses () instead of square brackets [].

The code for creating a tuple is shown in Listing 10.5.

Listing 10.5 Creating Tuples

```
1."'Method 1"'
2.tup1 = ('hello', 'world')          # Create a tuple
3.print(tup1)                      # Output tuple content
4.print(type(tup1))                  # Output type

5."'Method 2: Using the built-in function tuple()"'
6.tup2 = tuple(('hello', 'world'))
7.print(tup2)
8.print(type(tup2))
```

Tuples and lists can also be converted to each other through built-in functions. To convert a list to a tuple, simply place the list name inside the tuple() function. Similarly, to convert a tuple to a list, place the tuple name inside the list() function, as shown in Listing 10.6.

Listing 10.6 Conversion Between Tuples and Lists

```
1."'List to tuple conversion"'
2.list1 = [1, 2, 3, 4]
3.print(list1)
4.tup1 = tuple(list1)
5.print(tup1)

6."'Tuple to list conversion"'
7.tup2 = (5, 6, 7, 8)
8.print(tup2)
9.list2 = list(tup2)
10.print(list2)
```

5. Dictionary

In Python, a dictionary is essentially a list of various paired information with unique identifiers. Similar to a list, a dictionary is a mutable sequence. However, unlike a list, it is an unordered mutable sequence, and the stored content is in the form of "key-value pairs", which is similar to a Xinhua Dictionary. It can associate pinyin with Chinese characters and quickly find the desired Chinese character through the pinyin syllable table. In the Xinhua Dictionary, the syllable table is equivalent to the key (key), and the corresponding Chinese character is equivalent to the value (value). The key is unique, while the value can be multiple. Dictionaries are very useful when defining an object with multiple named fields.

Lists and dictionaries are both very important data structures, but there are significant differences between them. To effectively utilize them, one must understand

these distinctions. When using a dictionary, it is essential to be clear about these distinctions.

In a list, you can refer to a list value using consecutive integers known as indices or index values. In a dictionary, to refer to a dictionary value, you can use integers, strings, or other Python objects, collectively referred to as dictionary keys. This feature makes dictionaries more practical than lists when unique keys better reflect the meaning of the variable values.

(3) In a list, the list values are implicitly sorted because the indices are consecutive integers. In a dictionary, the dictionary values are not sorted because the indices are not just numerical. You can define a sorting operation for the items in a dictionary, but a dictionary does not have a built-in sort function.
(4) In a list, assigning a value to a non-existent position (index) is illegal. In a dictionary, however, new positions (keys) can be created when necessary.
(5) Because there is no sorting, the response time of a dictionary is faster when conducting searches or adding new values (when you insert a new item, the computer does not need to reassign index values). This is an important consideration when dealing with an increasing amount of data.

The code for creating a dictionary is shown in Listing 10.7.

Listing 10.7 Creating a Dictionary

```
1.”’Method 1”’
2.dic1 = {’name’: ’Python’, ’age’: 32} # Create a dictionary
3.print(dic1)  #Output dictionary content
4.print(type(dic1)) # Output type

5.”’Method 2: Using the built-in function dict()”’
6.dic2 = dict(name=’python’, age=32)
7.print(dic2)
8.print(type(dic2))
```

6. Set

Although sets are not as frequently used as other data types in Python, they are very practical and important. A set is an unordered collection of distinct elements and can be used as a key in a dictionary. Its purpose is to store different values together, and sets are used to perform relational operations between different sets without focusing on individual values within them.

A set can be created using curly braces {} or the set() function. It is particularly important to note that if you want to create an empty set, you must use set() instead of {}, as {} is used to create an empty dictionary.

The code for creating a collection is shown in Listing 10.8.

Listing 10.8 Creating a Set

```
1."'Method 1"'
2.set1 = {'Python', 32}  # Create a set
3.print(set1)                    # Output set content
4.print(type(set1)) # Output type

5."'Method 2:Using built-in function set()"'
6.set2 = set([1, 21, 33, 4])      #Contains unordered elements
7.print(set2)
8.print(type(set2))
```

10.3.2 If Statements

When programming, it is often necessary to check certain conditions and decide what measures to take accordingly. In Python, the if statement enables you to examine the current state of the program and take appropriate actions. The if statement makes a judgment based on the provided condition; if it is true, the program following the if statement is executed; otherwise, it is not.

1. The if-else statement

The logic provided by the if-else statement is: "If it is like this, do this thing; otherwise, do something else." else code Blocks are not necessary, but they can make the code clearer.

An example of the if-else statement is shown in Listing 10.9.

Listing 10.9 if-else Statement Example

```
1.x =5
2.if x > 1:
3.   print("True")
4.else:
5.   print("False, x is not greater than 1")
```

In this example, since *x* equals 5 and 5 is greater than 1, the condition x > 1 i*s true, so the print statement within the if code block is executed, outputting True;* if the condition in the if statement is false, the print statement within the else code block will be executed.

2. If-elif-else statements

An example of the if-elif-else statement is shown in Listing 10.10.

Listing 10.10 if-elif-else Statement Example

```
1.x =5
2.if x > 6:
3.  print('x is greater than 6')
4.elif x > 5:
5.  print('x is greater than 5V)
6.else:
7.  print('x is not greater than 5')
```

This example demonstrates a slightly more complex if-elif-else statement. Similar to the if-else statement example, it first checks if the if statement is true. If it is true, the judgment stops and the print statement in the if code block is executed. Since 5 > 6 is false, the execution continues and then checks if the elif statement is true. As 5 > 5 is also false, the print statement in the else code block is executed.

10.3.3 Loop Statements

1. For loop

The 'for' loop statement is a loop control statement in Python. It can iterate over the elements of any ordered sequence object, such as strings, lists, tuples, and other iterable objects. Although the 'if' statement discussed earlier has a different usage from the 'for' statement, it can be used in conjunction with the 'for' loop to perform conditional operations during iteration. It is used as a conditional statement in the for loop.

The code for the for loop example is shown in Listing 10.11.

Listing 10.11 Example of for Loop

```
1.numbers = [1, 2, 3, 4]      # Create a list.
2.for i in numbers: # Iterate through the list
3.print(i)                # Output each element
4.  Output result:
5.”’
6.1
7.2
8.3
```

```
9.4
10.”’
```

2. While loop

In Python programming, the ‘while’ statement is used to repeatedly execute a block of code as long as the condition is true. Once the condition becomes false, the program will move on to the next statement. The statement(s) to be executed can be a single statement or a block of statements. The condition can be any expression, and any non-zero or non-empty (not null) value is considered true. Additionally, the condition can be a constant value, indicating that the loop will always be executed. The loop terminates when the condition evaluates to false.

The while statement has two important commands: continue and break, which are used to skip loops. The continue command skips the current iteration, while the break command exits the current loop.

The code for the while loop example is shown in Listing 10.12.

Listing 10.12 While Loop Examples

```
1.i = 1
2.num = 0
3.while i<6:
4.num += i
5.i += 1
6.print(num)
```

This example calculates the sum of the integers from 1 to 5. The program first initializes *i* to 1, and then in the while loop, it checks the value of *i*. If *i* is <6, it executes the statements within the while block and increments *i* by 1. This process repeats: it checks again if *i* is still <6, and if so, it continues to execute the statements within the while block and increments *i* by 1. This continues until is no longer <6, at which point the while loop ends and the print statement below is executed to output the value of num.

The while loop is suitable for situations where you know exactly how many times the inner statements will be executed. When you are not quite sure how many times the inner statements need to be executed, you should use the for loop.

10.3.4 Functions

A function is a reusable program segment. In a program, if a certain function or operation needs to be executed multiple times, a function can be defined to achieve this.

If a certain function or operation is frequently used, the program segment that accomplishes this function or operation can be separated from the program and defined as a function. The original program that needs to execute this function or operation can then be replaced by a function call, thereby simplifying the program.

In some cases, you will find it more convenient and efficient to write your own functions rather than using Python's built-in functions or installing modules developed by others. For instance, if you find yourself constantly rewriting the same code snippet, it's a good idea to convert that snippet into a function. In some cases, the function you need may already exist in Python's standard library or in "importable" modules. If the function is already available, you should use these well-developed and extensively tested functions. However, in certain situations, the required function may not exist or be unavailable, and in such cases, you will need to create your own function.

In Python, you can create your own functions, which are called user-defined functions. The rules for defining them are as follows.

(1) Start with "def", followed by the name of the defined function and parentheses (), and end with a colon.
(2) The parentheses () can be empty or can contain parameters.
(3) The content that defines a function is indented in relation to the "def".
(4) The basic format for calling a custom function is "the name of the defined function()". If the content within the parentheses () is empty when defining the function, it should also be empty when calling it; otherwise, fill in the corresponding parameters.
(5) The 'return' expression ends a function and optionally returns a value to the calling function. A 'return' without an expression is equivalent to returning 'None'.

The code for the function definition example is shown in Listing 10.13.

Listing 10.13 Example of Defining a Function

```
1.def num(a, b):
2.    c = a + b
3.    return c
4.print(num(1, 2))
```

This example defines a simple addition function. The function has two parameters. Its execution content is to add the values of these two parameters together and return

the sum. After the definition, we can use this function. num(1, 2) is to replace the parameter *a* in the function with 1 and *b* with 2, add them together, and then return the sum 3. After that, the print statement outputs 3.

10.3.5 Class

Object-oriented programming is one of the effective methods for software development. In object-oriented programming, classes that represent things and scenarios in the real world are written, and objects are created based on these classes.

A class is a collection used to describe objects with the same attributes and methods. It defines the common attributes and methods of each object in the collection, and an object is an instance of a class.

A class binds data and functionality together. Creating a new class is to create a new object type, thereby creating new instances of that type. Class instances support attributes for maintaining their own state and methods (defined by the class) for modifying their own state. The rule for creating a class is: "class" followed by the class name, which is usually a word starting with a capital letter and ending with a colon.

The code for creating class instances is shown in Listing 10.14.

Listing 10.14 Create a Class

```
1.class Dog:
2.# Initialize attributes name and age
3.def __init__(self, name, age):
4.    self.name = name
5.    self.age = age
6.# Instance method
7.def sit(self):
8. print(f"{self.name} is sitting")
```

10.4 Building a Deep Learning Framework

To better learn the content of this book, in addition to installing Python on your computer, you also need to install a deep learning framework—TensorFlow or PyTorch.

10.4.1 Introduction to TensorFlow and PyTorch

TensorFlow is an end-to-end open-source machine learning platform. It features a comprehensive and flexible ecosystem that includes a variety of tools, libraries, and community resources to help researchers advance the study of advanced machine learning techniques and deep neural networks. TensorFlow offers multiple levels of abstraction, enabling users to easily build models, conduct reliable machine learning anytime and anywhere, and construct and train advanced models without sacrificing speed or performance.

PyTorch is an open-source Python machine learning library, whose predecessor is the well-known machine learning library Torch. In January 2017, Facebook AI Research (FAIR) launched PyTorch based on Torch. It is a deep learning framework for the Python language, which not only enables powerful GPU acceleration but also supports dynamic neural networks, a feature that many mainstream deep learning frameworks (such as TensorFlow, etc.) do not have. PyTorch can be regarded as a NumPy with GPU support or a powerful deep neural network with automatic differentiation. Besides Facebook, it has been adopted by Twitter, CMU, and Salesforce, among others. As a port of the classic machine learning library Torch, PyTorch offers Python users a comfortable choice for deep learning development.

Although both are deep learning frameworks, there are still significant differences in their specific implementations, as detailed below [6].

(1) Computational differences. To understand the distinctions between PyTorch and TensorFlow, it is essential to know the differences in their computational models.

PyTorch is a dynamic framework. A dynamic framework means that during the operation process, it will make reasonable arrangements in the optimal way according to different values.

Relatively speaking, TensorFlow is a static framework. A static framework means that a TensorFlow computation graph needs to be constructed before different data can be input for processing. This actually brings a very serious problem, that is, the computation process is in a fixed state. This inflexible operation mode will inevitably lead to relatively low operation efficiency. From the perspective of the operation process, PyTorch has obvious advantages.

(2) Applicable objects. Although these two procedures can achieve the same result, different operation processes will lead to different difficulties in the application of the program. PyTorch is relatively more capable of establishing results and solutions in a short time, and is more suitable for computer program enthusiasts, researchers, or small-scale projects. TensorFlow, on the other hand, is more suitable for large-scale operations, especially when it comes to cross-platform or embedded deployment. Therefore, when unsure whether to use PyTorch or TensorFlow, make a judgment based on your own goals and expected results.

10.4.2 Installing TensorFlow

1. Installation of TensorFlow

   Enter the following command in the terminal window.

   (1) Create a virtual environment: conda create -n tensorflow python = (version number), as shown in Fig. 10.18.
   (2) Enter the virtual environment: activate tensorflow, as shown in Fig. 10.19.
   (3) Connect to the Tsinghua University Open Source Software Mirror Station: "'conda config –add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/ conda config –set show_channel_urls yes'" As shown in Fig. 10.20.
   (4) Install TensorFlow: Run the command 'pip install –upgrade –ignore-installed tensorflow', as shown in Fig. 10.21. This process may take a while, so please be patient.

2. Check if the installation was successful.

   (1) To enter the TensorFlow environment: conda activate tensorflow.
   (2) Enter the Python interpreter: python.
   (3) To verify whether TensorFlow is installed successfully: import tensorflow as tf, as shown in Fig. 10.22.

If no errors occur, it indicates that TensorFlow has been installed successfully.

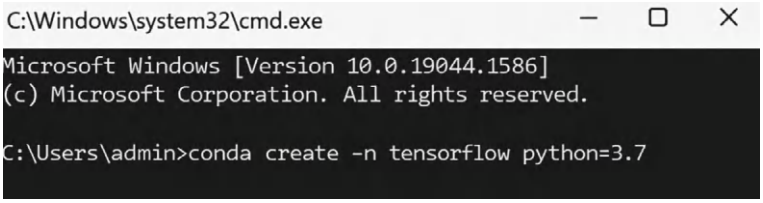


Fig. 10.18 Creating a virtual environment

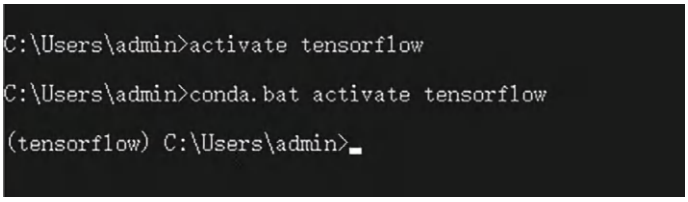


Fig. 10.19 Entering the virtual environment

Fig. 10.20 Connecting to the open source software mirror station of Tsinghua University

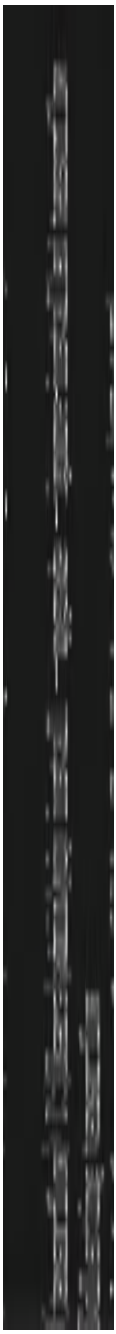


Fig. 10.21 Installing TensorFlow

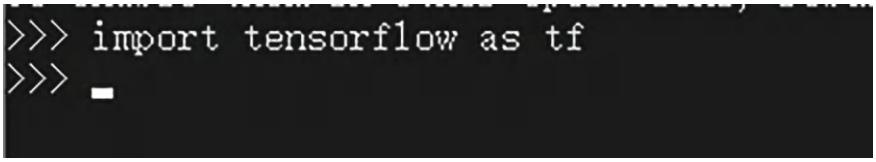


Fig. 10.22 Checking if TensorFlow is installed successfully

10.4.3 Installing PyTorch

PyTorch is mainly used for modeling and inference of deep learning algorithms. To speed up the training of algorithms, it is generally necessary to install PyTorch on a computer with a GPU. To accurately use the GPU in PyTorch, it is first necessary to install the GPU environment, including CUDA and cuDNN. After ensuring the correct installation of the GPU environment, PyTorch can be installed.

1. Install CUDA

With the development of graphics cards, GPUs have become increasingly powerful and are optimized for displaying images. In terms of computing, GPUs have surpassed general-purpose CPUs. Such powerful chips would be a waste if they were only used as graphics cards. Therefore, NVIDIA introduced CUDA, allowing graphics cards to be used for purposes other than image rendering and computing. CUDA stands for Compute Unified Device Architecture, which is a framework developed by NVIDIA for general-purpose parallel computing on GPU platforms. It includes the CUDA instruction set architecture (ISA) and the parallel computing engine inside the GPU. Developers can write programs for CUDA using languages such as C, OpenCL, FORTRAN, and C++. In simple terms, CUDA is a software driver provided by NVIDIA that enables parallel computing on graphics cards.

(1) By visiting the PyTorch official website, we can see the officially recommended PyTorch version and the corresponding CUDA version, as shown in Fig. 10.23.
(2) Download and install CUDA. Go to the CUDA official website to download the corresponding version of CUDA for PyTorch, as shown in Fig. 10.24. After downloading, simply extract and install it.
(3) Check if the installation was successful. Enter "nvcc-V" in the terminal command window. If you see the information as shown in Fig. 10.25, it indicates that the installation was successful.

2. Install cuDNN

NVIDIA cuDNN is a GPU-accelerated library for deep neural networks. It emphasizes performance, ease of use, and low memory overhead. NVIDIA cuDNN can be integrated into higher-level machine learning frameworks such as Caffe, TensorFlow, PyTorch, MXNet, etc. The simple plug-and-play design of cuDNN enables developers to focus on designing and implementing neural network models rather

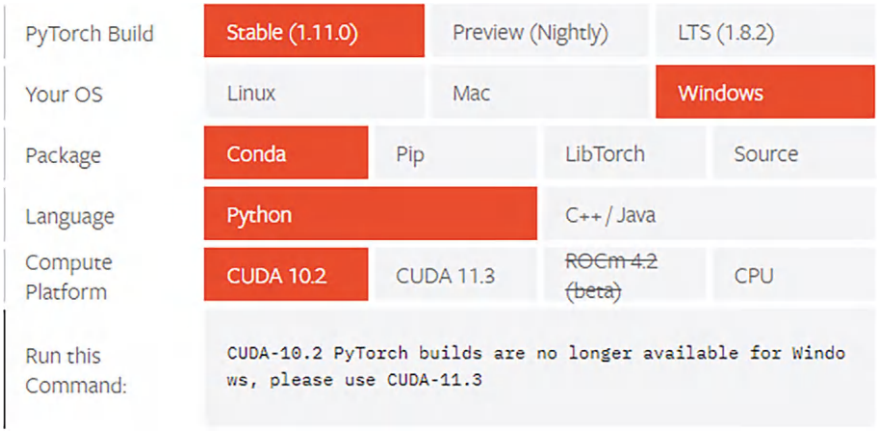


Fig. 10.23 PyTorch and CUDA versions

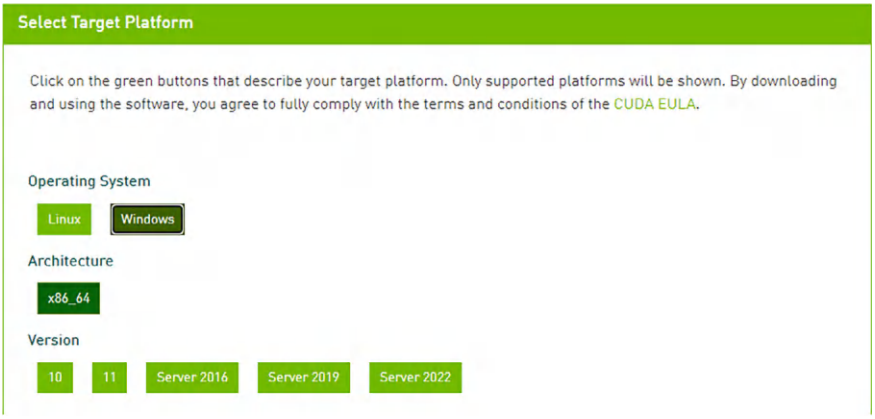


Fig. 10.24 Selecting the CUDA version and other information

Fig. 10.25 CUDA installation successful

Fig. 10.26 Corresponding versions of CUDA and cuDNN

than tuning performance, while also achieving high-performance modern parallel computing on GPUs.

(1) Download cuDNN. Find the cuDNN version corresponding to your CUDA version on the official website, as shown in Fig. 10.26.
(2) Install cuDNN. Unzip the downloaded installation package and directly copy the bin, include, and lib files from the cuDNN compressed package to the CUDA installation directory (e.g., C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1). The installation is then complete.

3. Install PyTorch

Select the corresponding version and run the command in "Run this Command" in the terminal to install it, as shown in Fig. 10.27.

4. Check whether the installation is successful.

As shown in Fig. 10.28, enter "import torch" in the terminal command window. If no error occurs, it indicates that the installation was successful. Then, you can check the PyTorch version and whether CUDA is installed successfully. If no errors are reported, the installation is complete.

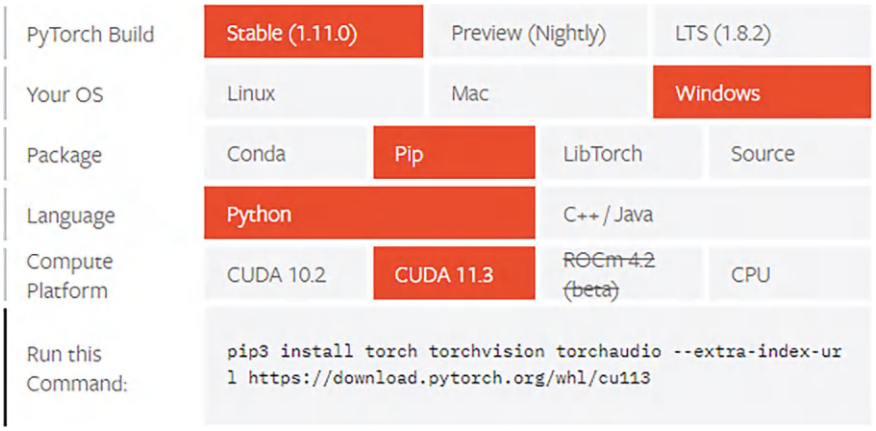


Fig. 10.27 PyTorch installation options

Fig. 10.28 PyTorch installation verifying

```
>>> import torch
>>> print(torch.__version__)
1.4.0
>>> print(torch.cuda.is_available())
True
>>>
```

References

1. Marvin, R., Ng'ang'a, M., Omondi, A.: Python Fundamentals: A Practical Guide for Learning Python, Complete with Real-World Projects for You to Explore. Packt Publishing Ltd (2018)
2. Matthes, E.: Python Crash Course (2016)
3. Chittibabu, P.: PyCharm IDE—A Handbook of Python Programming (2020)
4. Kennedy, M., Harrison, M.: Effective PyCharm: Learn the PyCharm IDE with a Hands-on Approach. Effectivepycharm. com (2019)
5. Anagnostopoulos, A.: Hands-On Software Engineering with Golang: Move Beyond Basic Programming to Design and Build Reliable Software with Clean Code. Packt Publishing Ltd (2020)
6. Chollet, F.: Deep Learning with Python. Manning (2021)

Chapter 11
Introduction to Commonly Used Databases

Drug design, also known as rational drug design, is a creative process of finding new drugs based on knowledge of biological targets. Specifically, drug molecule design is based on the research results of life sciences such as biochemistry, enzymology, molecular biology, and genetics. It designs reasonable drug molecules by targeting the drug action targets revealed in these basic studies and referring to the chemical structure characteristics of their endogenous ligands or natural substrates. Rational drug design mainly has two types: Ligand-based drug design (LBDD) and structure-based drug design (SBDD).

Rational drug design often requires information such as the three-dimensional structure of proteins, small molecules, or targets. Common machine learning algorithms and deep learning algorithms rely on biological or pharmaceutical data for training and learning to achieve the goal of artificial intelligence-assisted drug design. This biological information can be retrieved from some bioinformatics-related databases. Therefore, this chapter introduces some commonly used databases based on types of drugs, proteins and drug-targets.

11.1 Drug Database

11.1.1 PubChem Database

1. Introduction to the PubChem Database

PubChem is a public repository of chemical substances and their biological activity data, established in 2004 by an institute of the National Institutes of Health in the United States, aiming to promote the public utilization of small molecule data resources. PubChem mainly contains small molecules, but also includes nucleotides, carbohydrates, lipids, peptides, and chemically modified macromolecules. It collects

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_11

information on chemical structures, identifiers, chemical and physical properties, biological activities, patents, health, safety, and toxicity data, etc. It is a very popular information resource in biomedical research fields such as cheminformatics, chemical biology, medicinal chemistry, and drug repurposing. At the same time, PubChem is also a source of chemical big data and can be used in many machine learning and data science projects, such as virtual screening, computational toxicology, drug repurposing, and drug side effect prediction [1].

PubChem consists of three sub-databases, namely Substance (Substances), Compound (Compounds), and BioAssay (Bioassays). The Substance database contains descriptions of chemical samples provided by data depositors and is linked to information about their biological activities; the Compound database stores unique compounds extracted from the Substance database.

The PubChem database stores biological assay descriptions and test results, mainly from high-throughput screening experiments and scientific literature. As of August 2020, the PubChem database website contains over 279 million substance descriptions, 111 million unique chemical structures, and 293 million bioactivity data points from 1.2 million biological assay experiments. Additionally, PubChem has created a special dataset containing PubChem data related to COVID-19 and SARS-CoV-2. The homepage of the database website is shown in Fig. 11.1.

2. Application cases of PubChem

PubChem offers several commonly used functions, such as finding genes and proteins that interact with a given compound, identifying drug-like compounds similar to the query compound through 2D similarity search, calculating similarity scores between compounds, searching for drugs targeting specific genes, and obtaining bioactivity data of all chemicals tested against proteins, etc. [2].

On the homepage of the PubChem database website, we can search for the chemical and physical properties, biological activities, safety and toxicity information, patents, and literature citations of compounds by using key search terms such as

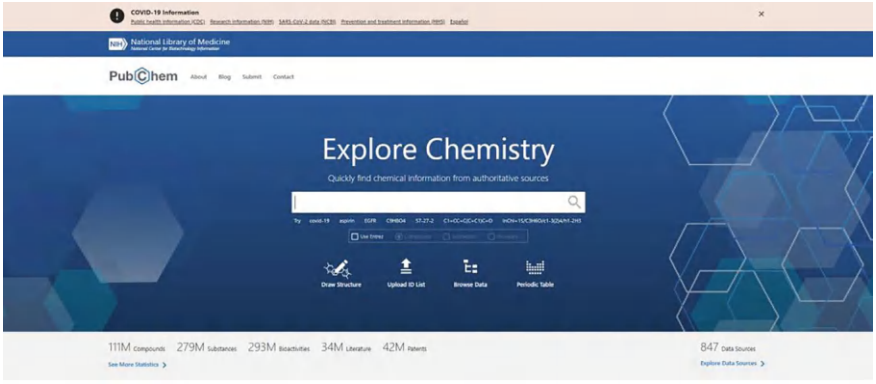


Fig. 11.1 Homepage of the PubChem database

compound names, chemical formulas, SMILES, and other identifiers. Additionally, structure-based search and batch search are also available. For structure-based search, click on "Draw Structure" on the homepage to open the "DRAW STRUCTURE" interface, draw the structure diagram to complete the search, as shown in Fig. 11.2. The search results are the same as those obtained through keyword search. For batch search, click on "Upload ID List" on the homepage to open the "UPLOAD ID LIST" interface, upload the data file to complete the search, as shown in Fig. 11.3. In this interface, you can learn through the sample files.

Let's take the common keyword search as an example to introduce the simple usage of the drug search function in the PubChem database. For instance, to search for the drug Ibrutinib, enter the drug name on the homepage of the PubChem database website and click the search button. The search results are shown in Fig. 11.4, including 23 compound structure information (including Ibrutinib monomer and its mixed drugs), 196 substance descriptions, 2688 drug activity information, and 408 patents, etc.

Click on the first item under the Compounds directory to view detailed information, as shown in Fig. 11.5. The right side of the interface displays the information directory, such as structure, name and identifier, chemical and physical properties, drug and drug information, etc. The left side lists the relevant information about Ibrutinib. On this website, we can see that Ibrutinib is an orally bioavailable small molecule Bruton tyrosine kinase inhibitor.

Bruton's tyrosine kinase (BTK) inhibitors, which have potential anti-tumor activity, are used as monotherapy for the treatment of relapsed or refractory mantle cell lymphoma (MCL) and chronic graft-versus-host disease (cGvHD) in adults. Additionally, clinical trials that have been conducted or are currently underway for such drugs can also be observed.

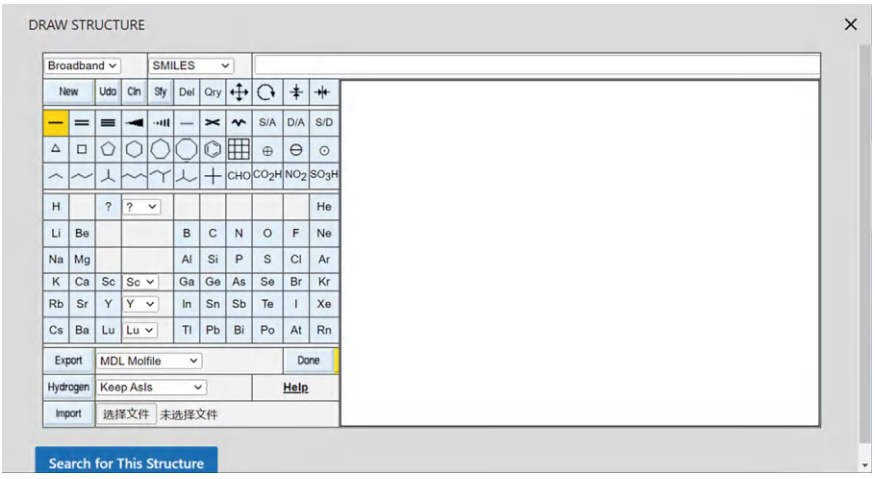


Fig. 11.2 DRAW STRUCTURE interface of PubChem

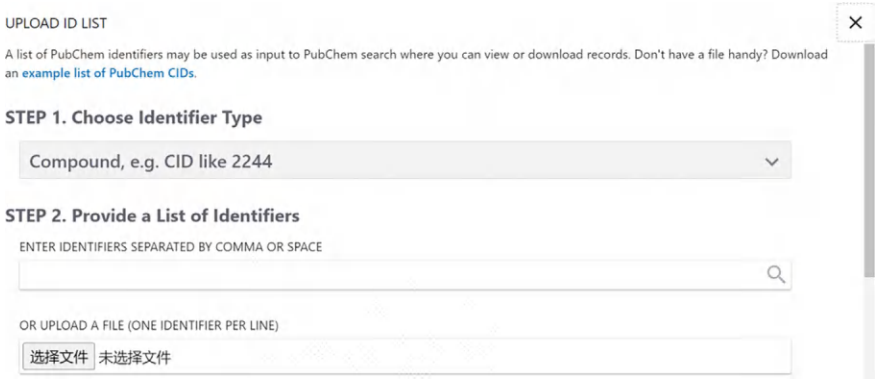


Fig. 11.3 UPLOAD ID LIST interface of PubChem

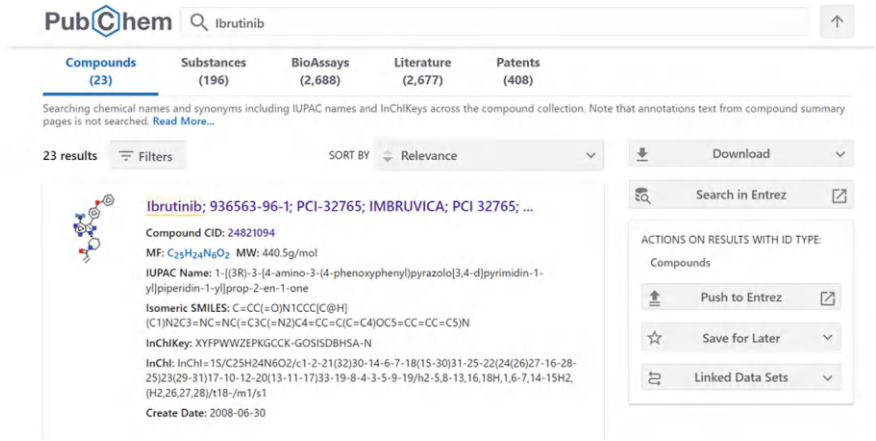


Fig. 11.4 PubChem search results for Ibrutinib

11.1.2 DrugBank Database

1. Introduction to DrugBank

DrugBank is a comprehensive, freely accessible online resource database that provides detailed information on FDA-approved drugs and experimental drugs undergoing the FDA approval process, including drug details, drug targets, drug actions, and drug interactions [3]. It was first established in 2006 in the laboratory of Dr. David Wishart at the University of Alberta as a project to assist academic researchers in obtaining detailed structured information on drugs. Currently, DrugBank has become one of the most widely used reference drug resources in the world, frequently utilized by the general public, educators, pharmacists, pharmacologists, medicinal chemists, drug researchers, and the pharmaceutical industry [4].

Fig. 11.5 Detailed information page of drug Ibrutinib in PubChem

DrugBank has two main applications, namely clinical and chemical. In the clinical-oriented aspect, DrugBank can provide detailed, up-to-date, quantitative analysis or molecular weight information on drugs, drug targets, drug indications, and the biological or physiological outcomes of drug actions. In terms of chemical applications, since the database was first released, DrugBank has been widely used in computer-aided drug "repurposing", computer retrieval of drug structure data, drug docking or screening, drug metabolism prediction, drug target prediction, and general pharmaceutical education. Additionally, DrugBank offers many built-in tools for viewing, sorting, searching, and extracting text, image, sequence, or structural data.

The DrugBank database includes four additional databases: HMDB, T3DB, SMPDB, and FooDB, which are also complete metabolite/chemical informatics databases. In the latest version 5.1.9 of DrugBank, there are 14,623 drug entries, including 2724 approved small molecule drugs, 1517 approved biologics (proteins, peptides, vaccines, and allergens), 132 nutraceuticals, and over 6677 experimental (discovery stage) drugs. Additionally, there are 5269 non-redundant proteins (i.e., drug targets).

The sequences of drug targets (enzymes, transporters, carriers) are associated with these drug entries. Each entry contains over 200 data fields, with half of the information dedicated to drug/chemical data and the other half to drug target or protein data. The database also has a dedicated COVID-19 section, providing a comprehensive summary of research related to the novel coronavirus, helping researchers quickly and effectively obtain the information they need.

2. The drug search application of DrugBank

The homepage of the website is shown in Fig. 11.6. On the DrugBank Online homepage, searches can be conducted based on drugs, targets, pathways, and indications.

Taking the drug Ibrutinib as an example, enter the drug name in the search box on the homepage, and the results are shown in Fig. 11.7. On the left side is the information directory, such as pharmacology, interactions, products, categories, clinical trials, performance, and targets; on the right side is the detailed information, where related background, drug indications, drug contraindications, structure, and target information can all be found.

In addition, you can also conduct a search by clicking the Browse or Search button. The search modes include chemical structure, molecular weight, drug-food interactions, target sequences, etc.

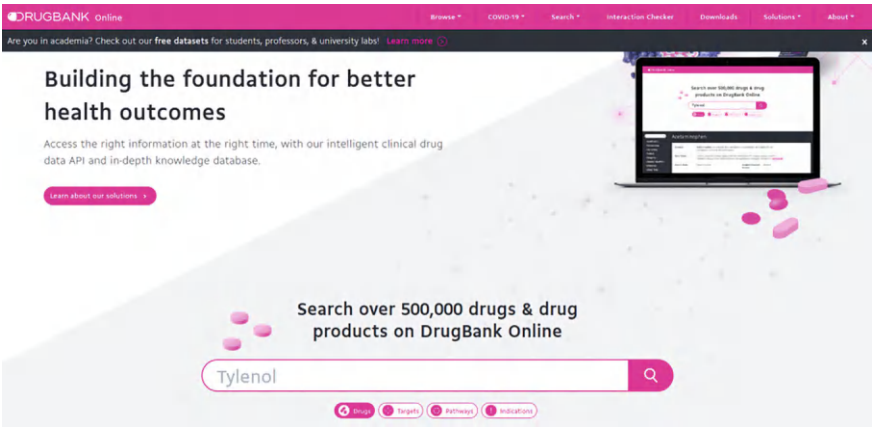


Fig. 11.6 Homepage of the DrugBank database

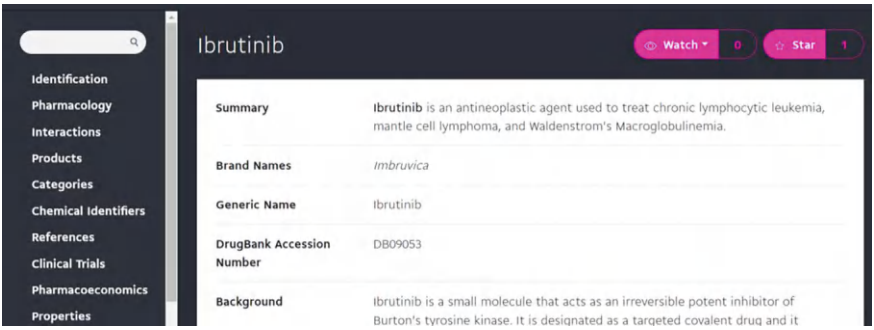


Fig. 11.7 DrugBank record of Ibrutinib

11.1.3 DGIdb

1. Introduction to DGIdb

Drug–Gene Interaction Database (DGIdb) was first released in 2013, aggregating drug-gene interaction and drug-related information from multiple sources, providing information on the association between genes and their known or potential drugs. It has now been updated to version 4.0, and its homepage is shown in Fig. 11.8. The genes included in DGIdb are mainly oncogenes, but there are also some related to other diseases (such as Alzheimer's disease, heart disease, diabetes, etc.). Users can obtain the information through a simple search interface, application programming interface (API), and TSV data download [5].

In DGIdb, the druggable genome is divided into two major categories. The first category includes genes with known drug interactions. This drug-gene interaction is useful when researchers have a list of candidate genes predicted to be activated in a disease and wish to identify drugs that may inhibit or modulate these genes. The second category includes genes that are "potentially" druggable based on their membership in gene classes related to druggability (such as kinases). The former represents established interactions between genes and drugs, mainly based on literature mining and obtained from existing public comments and databases. The latter represents genes with characteristics that make them suitable for drug targeting, but currently there may be no drugs targeting them. The sources of both types of drug genes are manually curated and semi-automatically imported. DGIdb can also assist clinical experts in prioritizing gene-level events, which ultimately helps in making treatment decisions.

In the latest version DGIdb 4.0, it contains over 40,000 genes and 10,000 drugs, involving more than 100,000 drug-gene interactions, and belongs to 42 potential

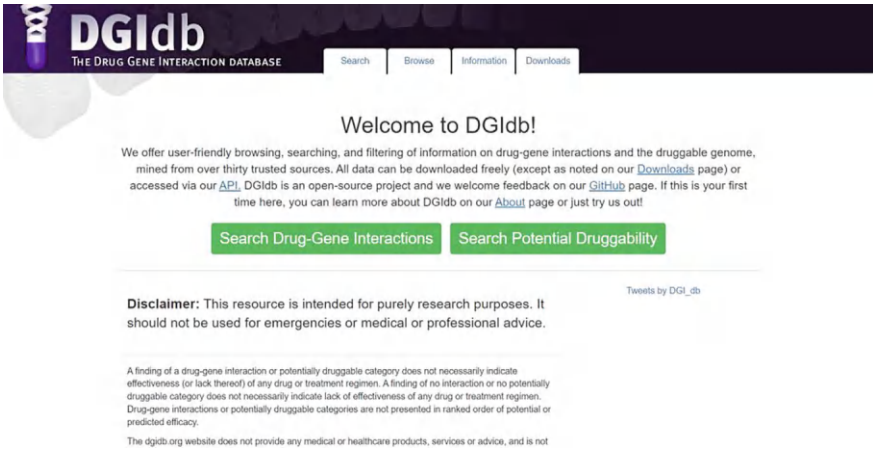


Fig. 11.8 Homepage of the DGIdb database

druggable gene categories. Users can input a list of genes to retrieve all known or potentially druggable genes in the list. Results can be filtered by source, interaction type or gene category. DGIdb is built on Ruby on Rails and PostgreSQL, with a flexible relational database schema that is sufficient to accommodate metadata from various sources.

2. The search application of DGIdb

DGIdb offers two types of search methods, namely Search Drug-Gene Interactions and Search Potential Druggability.

In the "Search Drug-Gene Interactions" module, users can search for drug-gene interactions by entering either the gene or drug name. On the right side, there are filtering options such as clinically actionable, pharmacogenomics, and drug resistance. There are also advanced filtering options, such as selecting different data sources, gene categories (e.g., DNA repair), and interaction types (e.g., cofactors and inducers), as shown in Fig. 11.9. In the "Search Potential Druggability" module, users can search for gene categories by entering a gene name. This module also provides a list of gene demonstrations, as illustrated in Fig. 11.10.

This example still takes Ibrutinib as an instance. In the "Search Drug-Gene Interactions" module, the search is conducted by the drug name. The search results are shown in Fig. 11.11, mainly consisting of three parts: Summary, Interactions, and Claims. The Summary contains drug information, while the Interactions section includes the interaction scores of Ibrutinib with different genes, the nature of the interactions, and interaction details (interaction mechanism, whether it is a direct interaction, detailed action information, and the source of PMID articles and data).

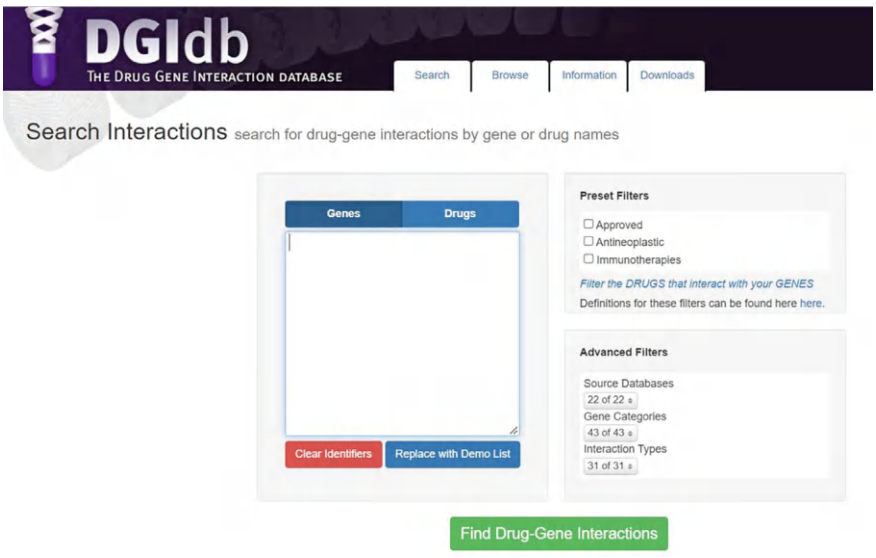


Fig. 11.9 Drug–gene interaction search interface in DGIdb

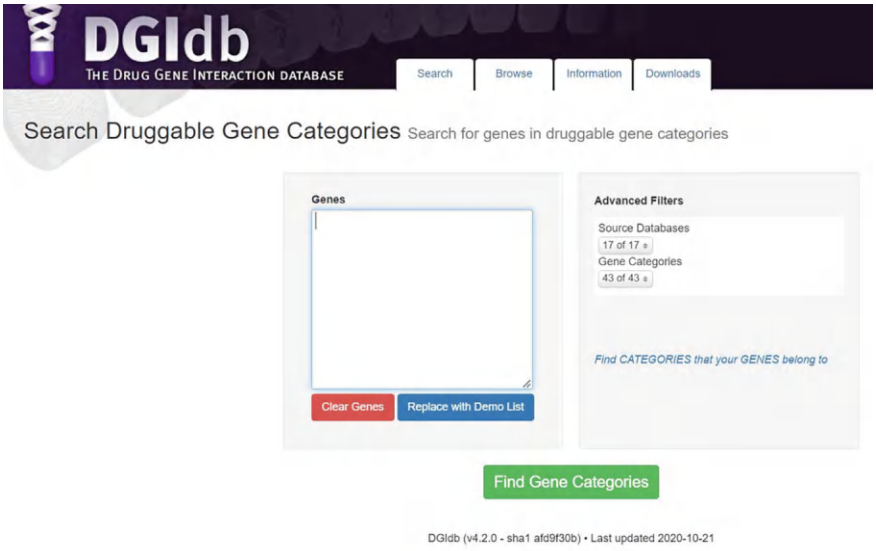


Fig. 11.10 Potential druggability search interface in DGIdb

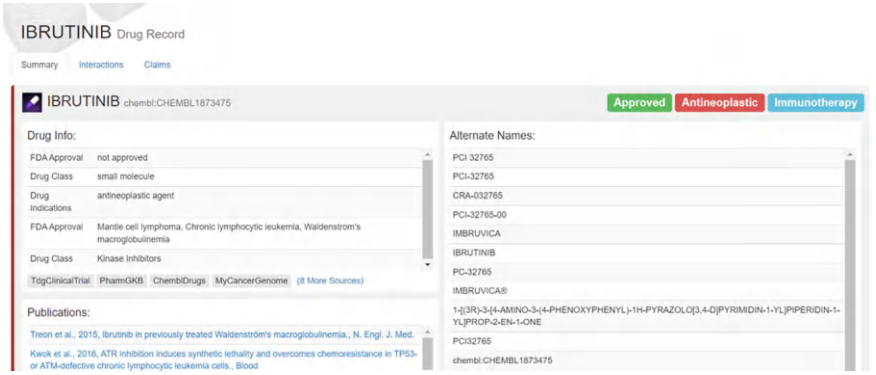


Fig. 11.11 Search results of Ibrutinib in DGIdb

11.1.4 ChEMBL

1. Introduction to ChEMBL

The ChEMBL database was established by the European Bioinformatics Institute in 2009. As a target-oriented and bioactive drug database, it aims to capture the data and knowledge of drug chemistry during the process of drug research and development. The homepage of the ChEMBL official website is shown in Fig. 11.12.

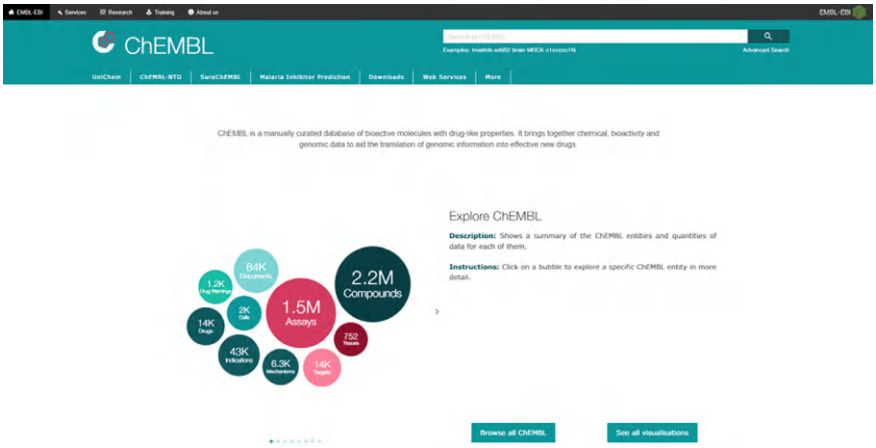


Fig. 11.12 Homepage of the ChEMBL database

The content of the ChEMBL database is about small molecules and their biological activities, which are mainly extracted from full-text articles of core medicinal chemistry journals such as Journal of Medicinal Chemistry, ACS Medicinal Chemistry Letters, European Journal of Medicinal Chemistry, and Journal of Natural Products, and combined with data on approved drugs and clinical development candidates, such as mechanisms of action and therapeutic indications. Biological activity data are also exchanged with other databases, such as the BioAssay and BindingDB subdatabases of PubChem, allowing users to benefit from a larger information body. The resulting database has a wide range of practical applications, including identifying chemical tools for targets of interest, evaluating compound selectivity, training machine learning models (e.g., for target prediction), assisting in generating drug repurposing hypotheses, assessing target tractability, and integrating with other drug repurposing resources.

The compounds in clinical development are mainly obtained from the United States Adopted Name (USAN) application and the ClinicalTrials.gov database. The approved drugs are mainly sourced from the FDA Orange Book database and the annual list of FDA New Drug Approvals, with additional information extracted from the British National Formulary (BNF) and the ATC classification. The indications of clinical trial drugs are obtained from the ClinicalTrials.gov database and mapped to disease IDs in the Medical Subject Headings (MeSH) and Experimental Factor Ontology through a combination of manual and automatic methods. The indications of approved drugs are obtained from the drug package labels on DailyMed and the ATC classification. The therapeutic targets of both approved drugs and clinical trial drugs are manually obtained from reference sources such as scientific literature, drug package labels, and pipeline information from pharmaceutical companies.

In the latest version, ChEMBL 30.0, the database has collected a total of 14,855 targets, 2,157,379 distinct compounds, and 84,092 patents.

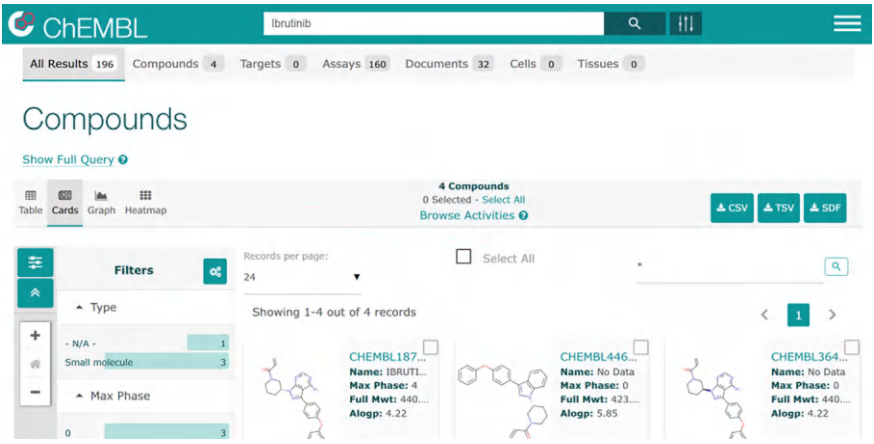


Fig. 11.13 Search results of Ibrutinib in ChEMBL

2. The retrieval application of the ChEMBL database

There are four ways to search for drug information in this database, namely drug name (compound), drawing compound structure, inputting sequence, and molecular or target ID. In this example, we take the drug Ibrutinib as an example and search by name. The results are shown in Fig. 11.13. On this page, the Filters window on the left is a filter that lists the statistics of various characteristics of these 46 small molecules, including molecular type (Type), Max Phase, the number of violations of the five rules (#RO5 violations), molecular weight (Molecular Weight), etc., and allows users to browse the original data subset within the given range by clicking the corresponding values.

It can be seen that the first one displayed is the drug we are looking for, Ibrutinib. After clicking on it, detailed information can be viewed, as shown in Fig. 11.14. Information such as the drug's name classification, source, molecular characteristics, indications, mechanism of action, and clinical data can all be found.

In addition, ChEMBL is the most commonly used and practical for searching drug-target data.

11.1.5 ETCM

1. Introduction to the ETCM Database

ETCM, which stands for The Encyclopedia of Traditional Chinese Medicine, was launched in 2018. The homepage of the ETCM database website is shown in Fig. 11.15. ETCM includes comprehensive and standardized information on

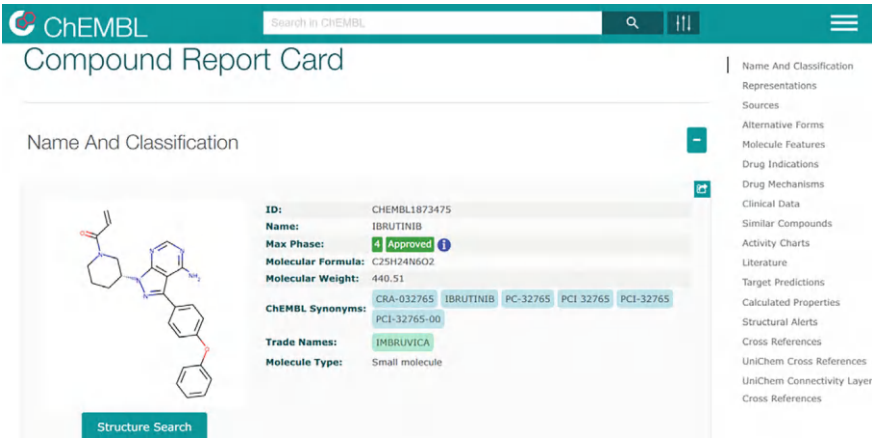


Fig. 11.14 Detailed information of Ibrutinib in ChEMBL

commonly used herbs and traditional Chinese medicine formulas and their components. The basic properties and quality control standards of herbs, formula compositions, component drug similarities, and many other pieces of information provided by ETCM can serve as a resource for users to conveniently obtain detailed information about herbs or formulas. To promote the research on the functions and mechanisms of traditional Chinese medicine, ETCM provides predicted target genes for traditional Chinese medicine components, herbs, and formulas based on the chemical fingerprint similarities between traditional Chinese medicine components and known drugs. ETCM has also developed systematic analysis functions, allowing users to explore the relationships or build networks among traditional Chinese medicines, formulas, components, gene targets, and related pathways or diseases.

At present, the database contains 402 kinds of Chinese herbal medicines, 3959 prescriptions, and 7284 effective components. In addition, it includes 2266 gene targets that the effective components of Chinese herbal medicines may target, as well as 4323 corresponding diseases.

2. The Application of the ETCM Database

Take the traditional Chinese medicine mugwort leaf as an example. On the right is the information directory of mugwort leaf, including basic information, identification information, characteristics, TCM properties, pharmacology, cross-references, etc. (Fig. 11.16).

Fig. 11.15 Homepage of the ETCM database

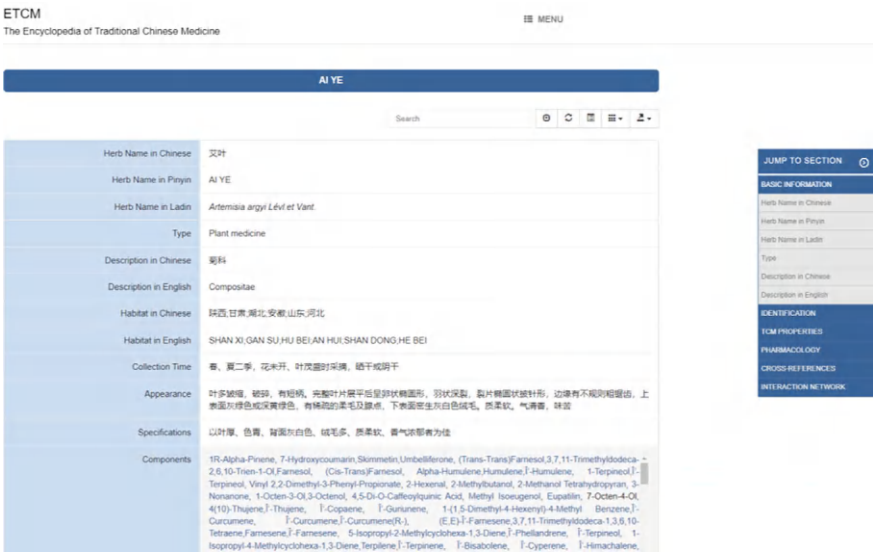


Fig. 11.16 Query results of AI YE in the ETCM database

11.2 Protein Databases

11.2.1 UniProt Database

1. Introduction to the UniProt Database

The UniProt database, also known as the UniProt Protein Resource, whose homepage is shown in Fig. 11.17, is edited and produced by the UniProt Consortium [6], which is composed of institutions such as the European Bioinformatics Institute (EBI), the Protein Information Resource (PIR) in the United States, and the Swiss Institute of Bioinformatics (SIB). It provides comprehensive, high-quality and free access to protein sequence and functional information resources. It includes three parts: the protein knowledge base UniProtKB, the protein sequence archive UniParc, and the protein sequence reference UniRef. Currently, it is one of the internationally renowned databases.

Swiss-Prot is an annotated protein sequence database produced by the UniProt Consortium. The database is composed of protein sequence entries, each of which contains a protein sequence together with reference literature information, taxonomic information, and curated annotations. These annotations include protein function, post-translational modifications, key sites and regions, structural features, sequence similarities, relationships between sequence variants and diseases, and sequence conflicts. Swiss-Prot aims to minimize redundancy and has established extensive cross-references to numerous external databases, including nucleic acid sequence libraries, protein sequence databases, and protein structure resources. Historically, the Sequence Retrieval System (SRS) was used to search Swiss-Prot and other EBI databases. Protein sequence submissions can be made through UniProt's web-based submission system and are reviewed by curators before inclusion.

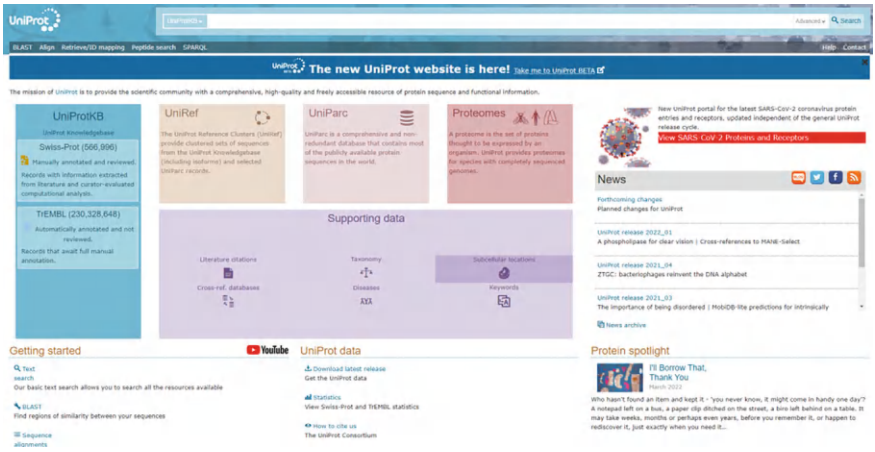


Fig. 11.17 Homepage of the UniProt database

UniParc, a comprehensive non-redundant protein sequence archive, encompasses the majority of publicly available protein sequences worldwide. Proteins may exist in different source databases or in multiple copies within the same database. UniParc avoids such redundancy by storing each unique sequence only once and providing it with a stable and unique identifier (UPI), enabling the identification of the same protein across different source databases. UPIs are never deleted, changed, or reassigned.

Protein sequences are classified into three datasets based on UniRef, namely UniRef100, UniRef90, and UniRef50. The data mainly come from the knowledge base UniProtKB, and also include some entries from the archive database UniParc.

2. The Application of the UniProt Database

In the UniProt database, not only can information about individual proteins be queried, but also batch queries for multiple proteins can be conducted. Taking the CD47 protein as an example, its related information can be queried, and the results are shown in Fig. 11.18. On the left side, the "Filter by" option allows for filtering between Swiss-Prot and TrEMBL. In the table on the right, those marked in yellow are from the Swiss-Prot database with manual annotations, while those in blue are from the TrEMBL database without annotations. In the "Popular organisms" section, the results can be classified by species. In this example, there are classifications such as Human, Mouse, Bovine, and Rat. The table on the right shows the protein ID, protein and gene names, species, and length. At the top of the table, there are BLAST and Align tools. BLAST can be used to perform BLAST analysis on the protein sequence of interest; Align can be used to compare the similarity of multiple sequences. When two or more sequences are selected, their homology can be compared. If the homology between two groups of sequences is high, their functions may be similar. Additionally, the protein sequence information can be downloaded in formats such as FASTA, Excel, and XML.

We selected the first entry in the "Human" group, labeled "Q08722", and the result is shown in Fig. 11.19. On the left side of the page is the entire web page's directory, which contains all the information about this protein, such as its function, cellular localization, PTM, interactions, advanced structure, sequence, and links to

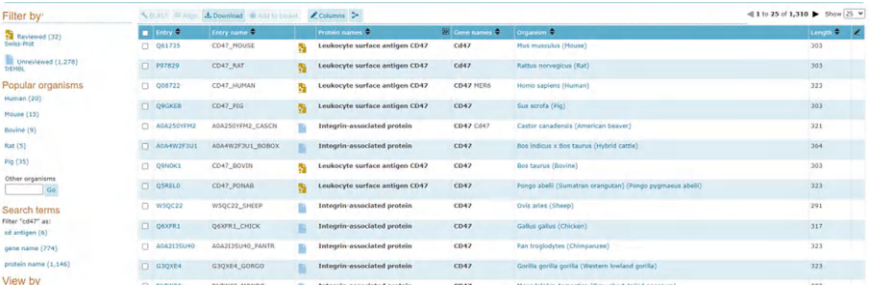


Fig. 11.18 Protein search results page in UniProt

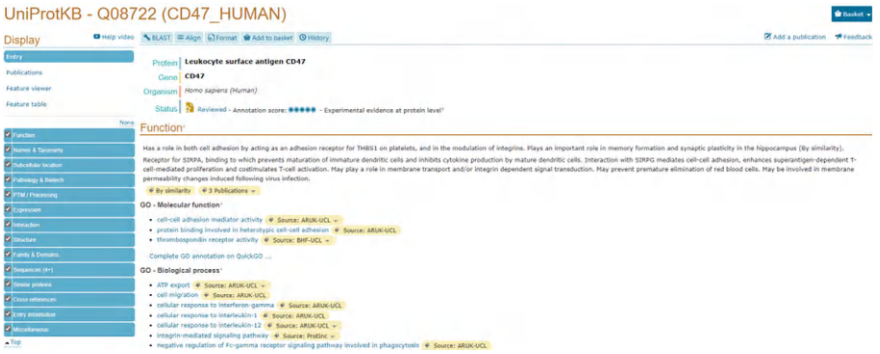


Fig. 11.19 Detailed Information of CD47 in UniProt

other databases about this protein. On the right side of the page, there is a detailed introduction to the CD47 protein. Its functions include: playing a role in cell adhesion by serving as the adhesion receptor for THBS1 on platelets, and also playing a role in the regulation of integrins; it is also important in hippocampal memory formation and synaptic plasticity (by similarity), etc.

11.2.2 Protein Data Bank

1. Introduction to PDB

The Protein Data Bank (PDB) was established in 1971 by the Brookhaven National Laboratory in the United States. At that time, it contained only seven X-ray crystal structures and was the first open-access digital biological data resource. Today, it remains the only international database storing three-dimensional structural data of biological macromolecules, which include proteins, nucleic acids, and complexes of nucleic acids and proteins. The data collected by the PDB are derived from X-ray crystallography and nuclear magnetic resonance (NMR) experiments and are regulated, reviewed, and annotated by the worldwide Protein Data Bank (wwPDB). Currently, the PDB is updated weekly, and its maintenance is handled by the Research Collaboratory for Structural Bioinformatics (RCSB).

Currently, in addition to three-dimensional atomic coordinates, each PDB entry contains experimental data and metadata that describe the molecular model and experimental details. The metadata includes protein name, sequence, source organism, small molecule information (such as chemical name, structure and formula), data collection information (such as instrument and data processing), and structure determination information (such as model building, refinement and validation methods and statistics). Additionally, the wwPDB provides value-added annotations, such as secondary structure, quaternary structure description and information about ligand binding sites. PDB data and resources are used in both basic and applied

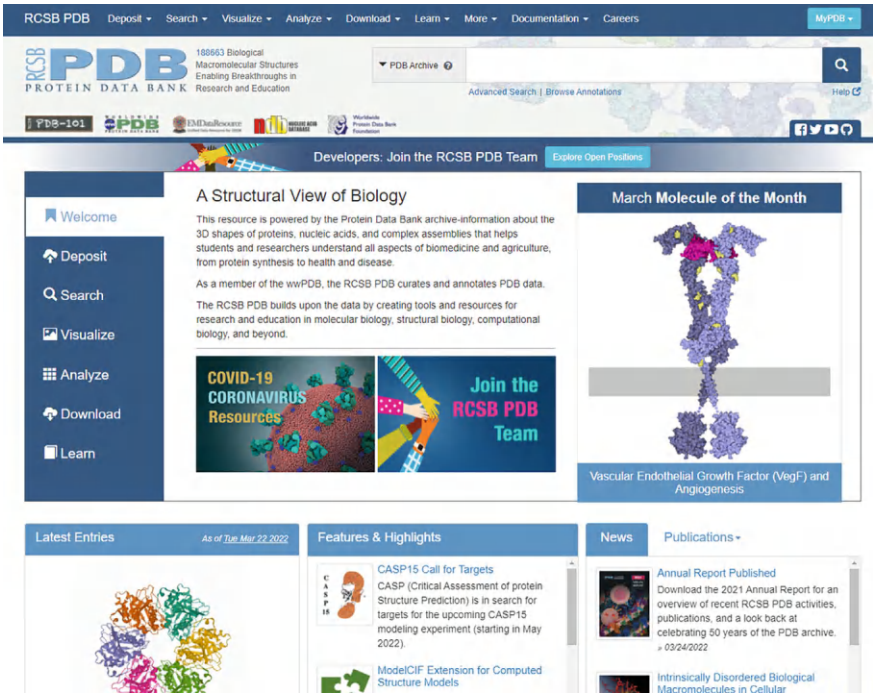


Fig. 11.20 Homepage of the PDB database

research, including science and education, experimental and computational method development, drug repurposing, etc. Besides experimental determined structures, the AlphaFold database provides over 214 million predicted protein structures.

2. The Use of PDB

The PDB enables users to conduct searches in various ways and through Boolean logic combinations. The searchable fields include functional categories, PDB codes, names, authors, space groups, resolutions, sources, deposit dates, molecular formulas, references, biological sources, etc. The homepage of the database's official website is shown in Fig. 11.20.

11.2.3 NCBI Database

1. Introduction to the NCBI Database

The full name of NCBI is the National Center for Biotechnology Information, which is a division of the United States National Library of Medicine (NLM) under the National Institutes of Health. It was established in 1988 with the aim of developing

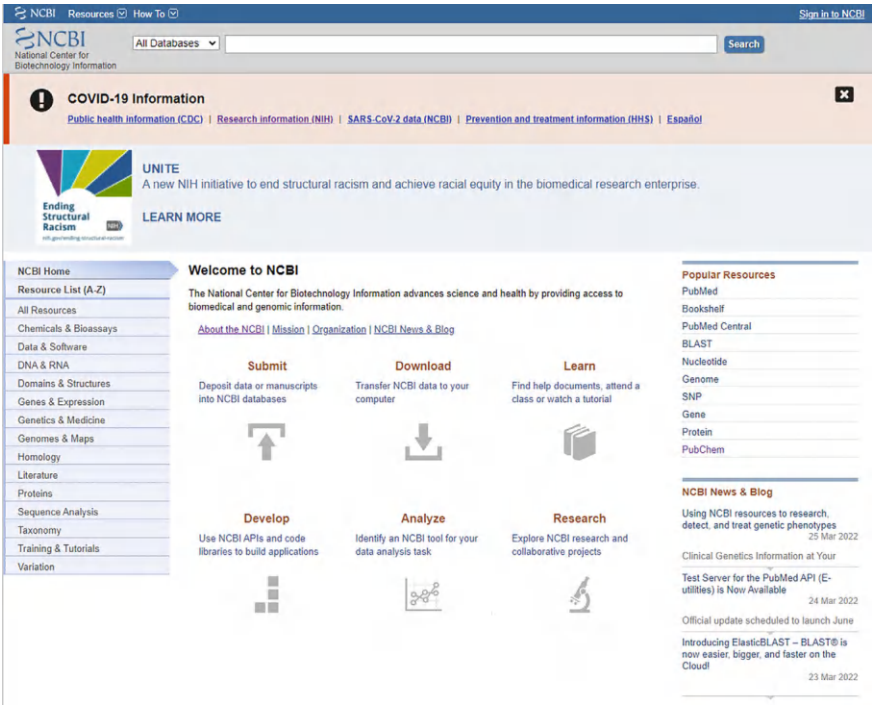


Fig. 11.21 Homepage of the NCBI database

information systems for molecular biology. This database provides a vast amount of online resources for bioinformatics and data, mainly including gene, DNA, RNA, protein sequences, literature indexes, and compound databases etc. The homepage of its official website is shown in Fig. 11.21.

The National Center for Biotechnology Information (NCBI) provides a comprehensive collection of bioinformatics resources and services, including data storage and maintenance, data retrieval and download, programmatic access through Application Programming Interfaces (APIs), integrated bioinformatics analysis tools, and educational and training resources. NCBI hosts more than 40 integrated databases covering biomedical literature, genomic sequences, molecular biology, and related research fields.

Among these resources, PubMed is a publicly accessible literature database that provides free access to biomedical abstracts and citations, supporting efficient literature search and exploration. GenBank, established by NCBI in 1992, is a comprehensive public repository of nucleotide sequence data with associated annotations, and serves as a foundational resource for genomic and molecular biology research. In addition, NCBI collaborates with the NCI through initiatives such as the Cancer Genome Anatomy Project to support cancer-related genomic studies.

In addition to hosting the GenBank nucleotide sequence database (whose data resources are sourced from several major DNA databases worldwide, including the Japanese DNA database DDBJ, the European Molecular Biology Laboratory database EMBL, and several other renowned research institutions), NCBI also offers a wide range of powerful data retrieval and analysis tools. Currently, NCBI provides a total of 46 functions, including the 1000 Genomes Browser, Batch Entrez, BLAST Link (Blink), CDTree, Genome BLAST, PubChem Structure Search, Sequence Viewer, etc. All of these can be found on the NCBI homepage through corresponding links, and most of them have evolved from the BLAST function.

The data of NCBI comes from direct submissions by researchers, national and international collaborations or agreements with data providers and research consortia, as well as internal curation efforts. Among them, protein sequence information is one of the important functions of NCBI, which contains textual information of protein sequences from different websites and servers, including the NCBI Reference Sequence Project, GenBank, PDB and UniProtKB/Swiss-Prot, etc. At the same time, it provides sequence-related gene, DNA/RNA sequence, biological pathway, expression and mutation information, references, etc. Researchers can search for almost all protein sequence information through NCBI.

2. A Simple Application of the NCBI Database

In the NCBI database, one can choose from Gene, Protein, Genome, GEO, etc. according to their needs. Taking the protein integrase as an example, on the homepage of the database, select Protein and search for integrase, and all the sequence information of integrase proteins can be obtained, as shown in Fig. 11.22. On the left side, one can select species and the source of the database to filter the sequence information. Taking the first sequence of integrase with 502 amino acids, Integrase [*Streptococcus pneumoniae*], as an example for introduction. The detailed information of this sequence is shown in Fig. 11.23, which includes GenBank code and sequence features. On the right side, one can choose RunBLAST (for sequence alignment), Identify Conserved Domains (to analyze conserved domains), and Highlight Sequence Features (to highlight sequence features).

Such as displaying sequence features, etc.

11.2.4 SMART Database

1. Introduction to the SMART Database

SMART, which stands for Simple Modular Architecture Research Tool, is a web resource for proteins.

The identification and annotation of protein domains as well as the analysis of protein domains. Protein domain databases remain important annotation and research tools. The SMART database integrates many manually curated Hidden Markov Models from various fields, along with a powerful web-based interface that offers a

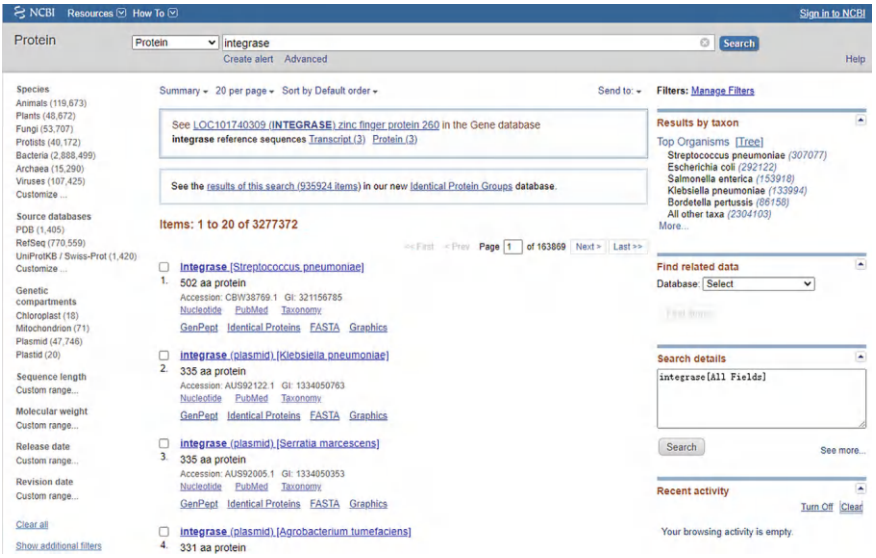


Fig. 11.22 Sequence information of integrase protein

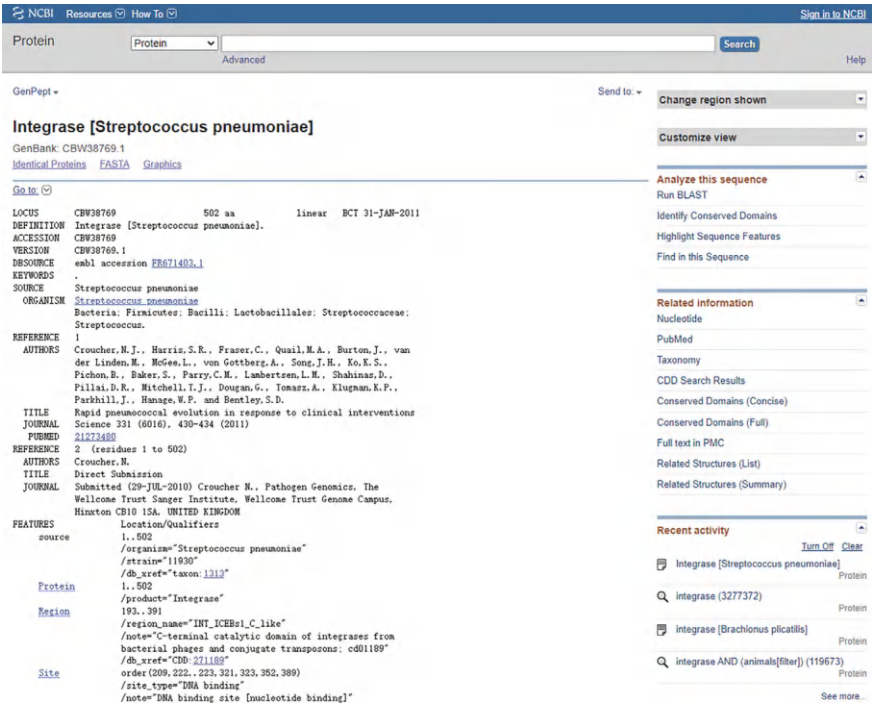


Fig. 11.23 Detailed information on the Integrase [Streptococcus pneumoniae] sequence

Fig. 11.24 Homepage of the SMART database

variety of analysis and visualization tools. The homepage of the official website is shown in Fig. 11.24.

SMART has two distinct modes: Normal and Genomic. The main difference between them lies in the databases they use. Normal SMART employs the databases Swiss-Prot, SP-TrEMBL, and stable Ensembl proteomes. Genomic SMART, on the other hand, utilizes whole genome sequences.

The main basic protein database in SMART combines the complete UniProt and stable Ensembl proteomes. The current version contains over 50 million proteins from approximately 460,000 species, subspecies and strains.

2. The Application of SMART Database

Since the SMART database has two different modes, you can choose the appropriate mode based on the color of the homepage. Here, we take the Genomic mode as an example for the introduction. The display page of the Genomic mode is shown in Fig. 11.25. In this mode, there are sequence analysis and structure analysis. Sequence analysis can be conducted by entering the ID (or ACC) of the UniProt/Ensembl protein sequence or the protein sequence to search for the protein domains; structure analysis can either directly input the domain into the Domain selection text box or use the GO terms query to obtain the domain list. Both modes have examples available for selection and operation.

11.2.5 Pfam Database

1. Introduction to the Pfam Database

The Pfam database is a repository of conserved protein families and domains. It is a widely used resource for classifying protein sequences into families and domains. The homepage of its official website is shown in Fig. 11.26.

Fig. 11.25 Genomic mode display page

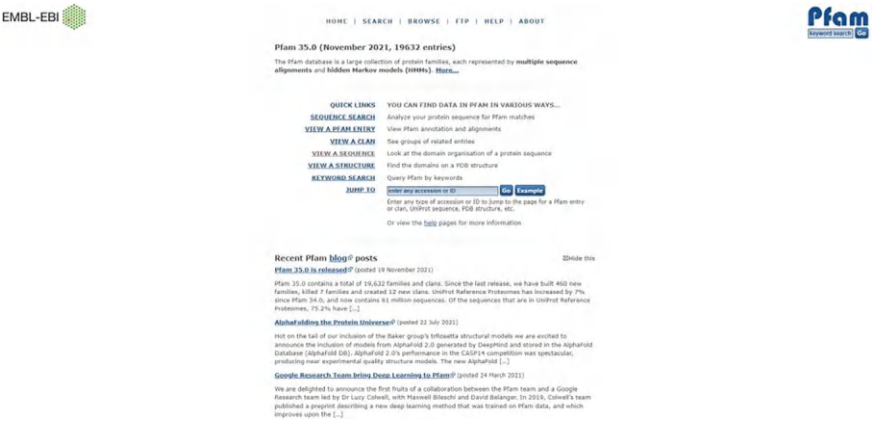


Fig. 11.26 Homepage of the Pfam database

The Pfam database is a repository of protein families, which classifies proteins into different families based on multiple sequence alignments and Hidden Markov Models. In this database, the following three levels of protein family information are provided. The first level is Family, each uniquely identified by a PF number. All Families can be classified into six types: Family, Domain, Repeat, Motif, Coiled-Coil, and Disordered. The second level is Clan, which groups multiple Families with similar three-dimensional structures or the same motifs into a Clan through similarity analysis. It can be regarded as the concept of a superfamily, and each Clan is identified by a CL number. The third level is Proteome, which provides information on the proteome of a species.

2. The Application of the Pfam Database

This database offers multiple input methods, allowing for sequence search, protein domain-related search, and keyword search, etc. Additionally, any type of Accession or ID can be entered to navigate to the pages of Pfam entries or clans, UniProt sequences, PDB structures, etc.

11.2.6 STRING

1. Introduction to the STRING Database

STRING (Search Tool for the Retrieval of Interacting Genes/Proteins) is a database developed by Peer Bork's team at the European Molecular Biology Laboratory (EMBL) for predicting protein–protein interactions. These interactions include both direct (physical) and indirect (functional) associations; they are derived from computational predictions, knowledge transfer between organisms, and interactions from other (main) databases.

In addition to predicting protein–protein interactions, there are many other applications, such as constructing interaction networks among multiple proteins and conducting protein function enrichment analysis.

The STRING database collects and evaluates evidence from multiple sources, such as automatic text mining of scientific literature, databases of interaction experiments and annotated complex pathways, computational interaction predictions based on co-expression and conserved genomic context, and systematic transfer of interaction evidence from one organism to another. Version 11.5 of the STRING database covers 67.6 M proteins from 5090 organisms.

2. The Application of STRING Database

The homepage of the STRING database is shown in Fig. 11.27. Clicking the SEARCH button leads to the search page, as illustrated in Fig. 11.28. On the left side of the page, there are many options for predicting interaction relationships. If we have a target protein and want to view its potential interacting proteins, we can choose "Protein by name"; if we have multiple proteins and want to view the interaction relationships among them, we can select "Multiple proteins".

11.2.7 Other Protein Databases

1. PROSITE database

The PROSITE database collects biologically significant protein sites and sequence patterns, and can quickly and reliably identify which protein family an unknown function protein sequence belongs to based on these sites and patterns. In the latest

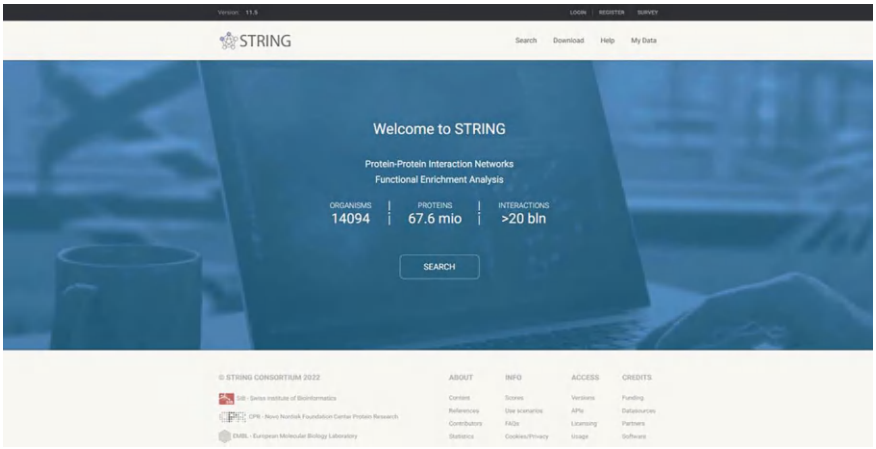


Fig. 11.27 Homepage of the STRING database

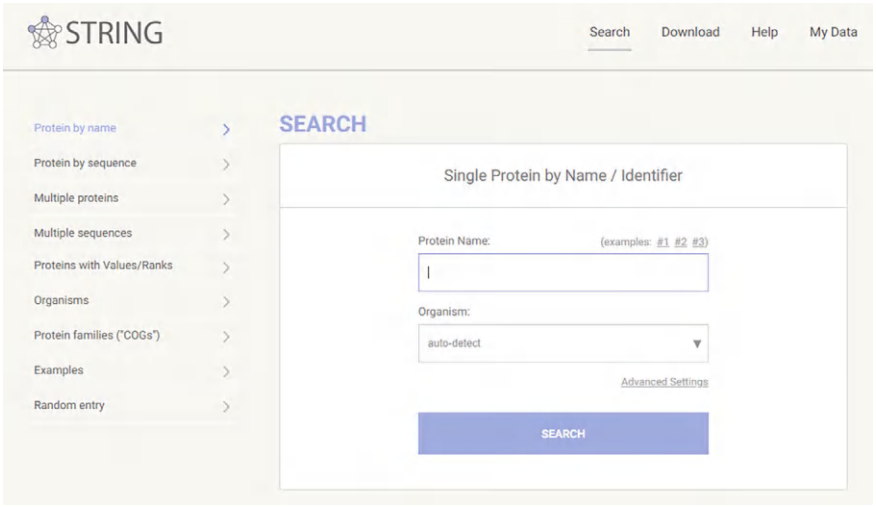


Fig. 11.28 Search page of the STRING database

version 2022_01, there are 1903 document entries, 1311 patterns, 1336 profiles and 1352 ProRules.

2. The SCOP database

The Protein Structure Classification Database (SCOP) aims to provide a detailed and comprehensive description of the structural and evolutionary relationships among proteins whose three-dimensional structures are known and deposited in the Protein Data Bank. It is classified according to Family, Superfamily, Fold, IUPR, Classes,

and Protein type. In the 2022 version, there are 71,812 non-redundant domains, representing 841,711 protein structures.

In addition, there are also databases such as the Database of Interacting Proteins (DIP), the Biological General Repository for Interaction Datasets (Bio-GRID), and the COG database that can be used as protein databases.

11.3 Drug-Target Database

A target refers to a structure located within a living organism that can be recognized or bound by other substances (ligands, drugs, etc.). Common drug targets include proteins, nucleic acids, and ion channels, etc. Understanding drug targets and early candidate drugs is helpful for researching the druggability of drugs, systems pharmacology, and developing drug repurposing tools, etc. Therefore, some commonly used drug-target databases are introduced below. The key to the efficiency of drug repurposing lies in choosing appropriate therapeutic targets and targeted drugs. The research on these targets and drugs and the knowledge obtained are very useful for accelerating the drug repurposing process.

11.3.1 TTD

1. Introduction to TTD

The homepage of Therapeutic Target Database (TTD) is shown in Fig. 11.29. TTD provides information on known and explored therapeutic protein and nucleic acid targets, associated diseases, pathway information, and corresponding drugs for each target. The core of TTD is target-centric data, around which other information is organized, including drug information, regulatory molecules, and related biological pathways. In TTD, targets can be categorized into four types: successful targets (with at least one approved drug), clinical-trial targets (with drugs in clinical trials but no approved drug), patent-recorded targets (reported in patents and subsequent literature), and literature-reported targets [7].

According to the 2020 release, TTD collected 38,760 drugs (2797 approved drugs, 10,831 clinical-trial drugs, and 20,123 experimental-stage drugs) and 3578 targets, including 498 successful targets, 1342 clinical-trial targets, and 1738 experimental-stage targets. In addition, 185 targets were recorded in patents as patent-recorded targets.

2. The Application of TTD

Users can search the database by target, drug, disease and biomarker, and can also use the drug similarity search tool to predict the targets of compounds without target information. Meanwhile, they can search for COVID-19 drugs and therapeutic targets in the advanced search module.

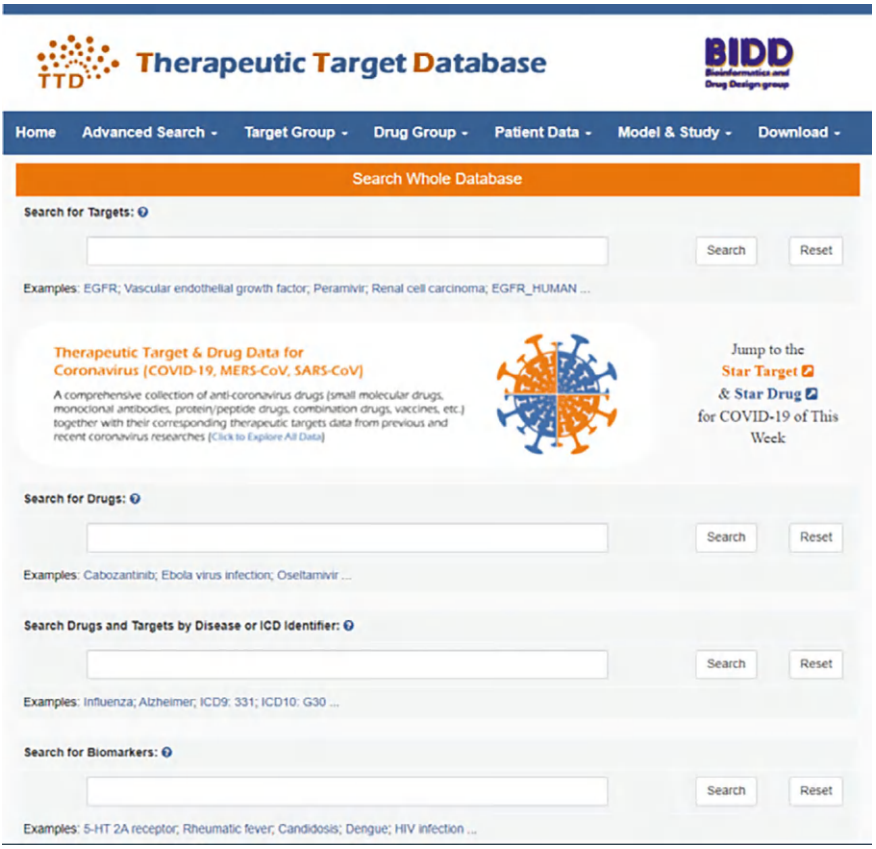


Fig. 11.29 Homepage of the TTD database

The TTD database provides search methods and examples on the homepage of its official website, such as target search. Taking EGFR (epidermal growth factor receptor, a member of the HER family) as an example. The search results are shown in Fig. 11.30. In the table, one can see the target information related to EGFR, including target name, target type, disease and drugs, and detailed drug information can also be viewed [8].

11.3.2 BindingDB

1. BindingDB

BindingDB was launched online in 2000 and is the first publicly accessible database for measuring binding affinities [9]. The focus of measuring binding affinities is

Fig. 11.30 Search results for the target EGFR in TTD

on the interactions between proteins as drug targets or potential drug targets and small molecules. The data in BindingDB is sourced from PDB-related literature reports, patent information, data from the BioAssays sub-database of PubChem, and recorded data in ChEMBL. The affinity data are derived from various measurement techniques, including enzyme inhibition activity and enzyme kinetics, isothermal titration calorimetry (ITC), nuclear magnetic resonance, and radioligand competition assays, among others. The types of data include ki, IC50, kd, EC50, etc.

Since its release, BindingDB has been updated weekly. Users can search through various methods such as target names, target sequences, drug names, drug structures, and pathway information. It also offers database data downloads and web service applications.

The programmatic interface enables users to retrieve and acquire server data.

As of March 19, 2022, BindingDB contains 41,296 entries (each with a DOI), including 8689 protein targets and 2,437,529 binding data for 1,047,667 small molecules. For proteins with 100% sequence identity, there are 5988 proteins with

BindingDB affinity measurements, as well as 11,442 protein–ligand crystal structures allowing for 85% sequence identity of the proteins.

2. The Application of BindingDB

The homepage of the BindingDB official website is shown in Fig. 11.31. By clicking on "Name" under "Target" on the left side, you can enter the search interface. Taking the target EGFR as an example, the relevant search page is shown in Fig. 11.32. The fourth item is the target information we need. Clicking on this item leads to the page shown in Fig. 11.33, where we can obtain detailed information about EGFR, such as the target name, small molecule BDBM number, chemical structure formula, English name, SMILES, and related literature information about this target.

In addition to retrieving information on targets, drugs, etc., there are other functions. Interested readers can refer to other materials.

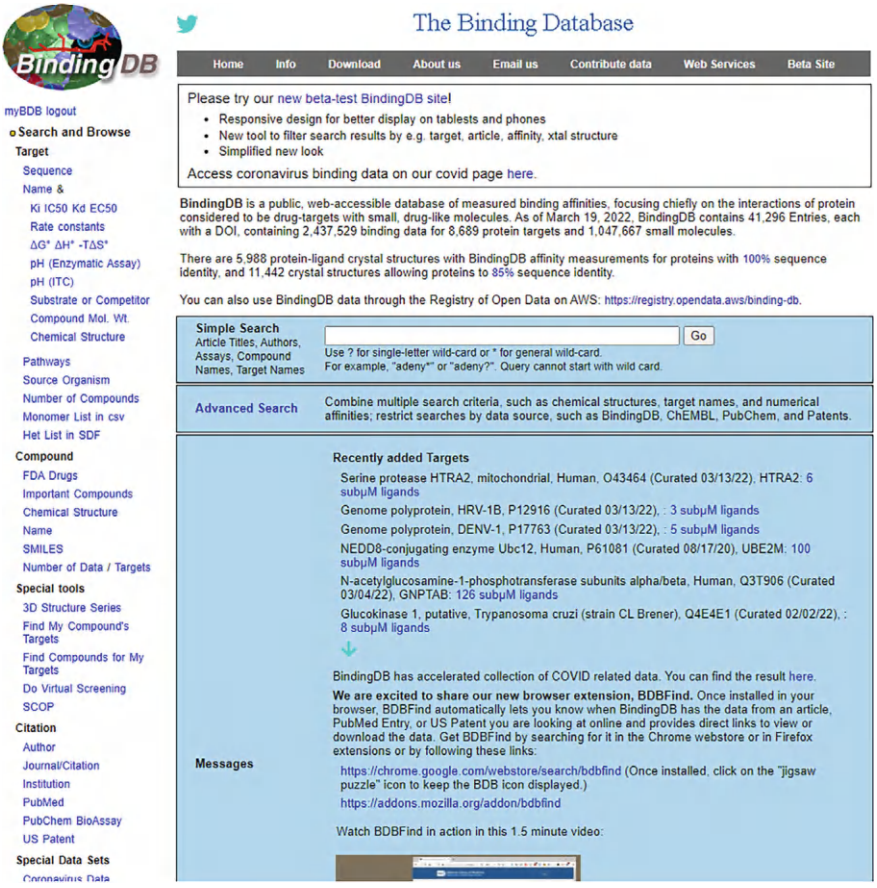


Fig. 11.31 Homepage of the BindingDB database

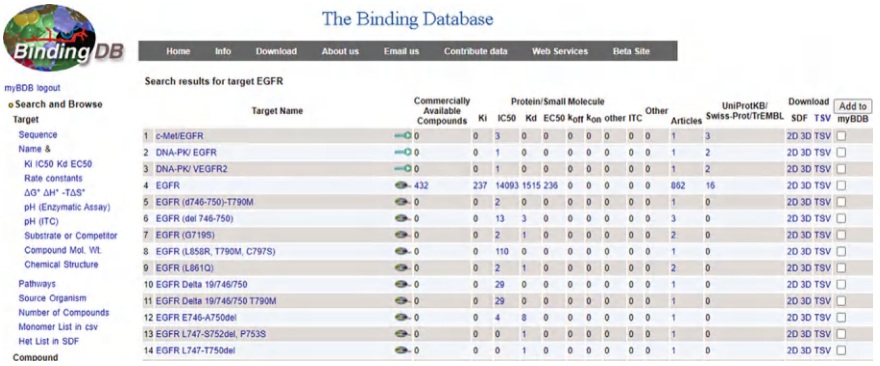


Fig. 11.32 Target EGFR search page

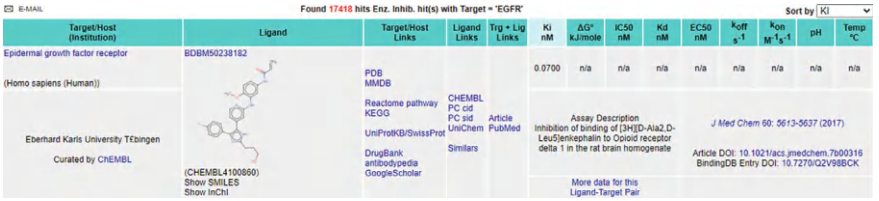


Fig. 11.33 Detailed information of Target EGFR

11.3.3 Other Drug-Target Databases

1. SuperTarget Database

The SuperTarget database is a network resource for analyzing drug-target interactions. It integrates drug-related information associated with drug indications, adverse drug reactions, drug metabolism, pathways, and Gene Ontology (GO) terms of target proteins. The SuperTarget website offers multiple methods for obtaining and viewing drug and target information:

(1) In order to improve quick access to data, a simple full-text search tool ("Targle") has been implemented, which returns information on drugs, targets, and pathways respectively.

(2) For each type of entity, there is also a dedicated search section, namely for drugs, targets, pathways, Gene Ontology, and CYP. Users can choose from predefined identifiers or input various search terms. For example, targets can be searched through synonyms, UniProtKB identifiers, PDB identifiers, KEGG target identifiers, or EC numbers. The results section provides detailed information for each instance, including SMILES and InChI strings, a list of presumed targets for drug results, and a list of similar targets for protein–protein interaction and target results. All different entities are cross-linked, so it is easy to

obtain detailed information on presumed targets and affected pathways from drug search results.

(3) For more complex searches, advanced search options can be used, which include general attributes (such as the required number of hydrogen bond donors), or features related to specific drugs or targets (such as affinity values). This option allows for any combination of different search conditions. Therefore, users can perform various complex searches.

2. The Open Targets database

The Open Targets database was jointly established by Biogen, the European Bioinformatics Institute, GlaxoSmithKline and the Wellcome Trust Sanger Institute (WTSI). It is a comprehensive open-source research tool that supports the systematic identification and prioritization of potential therapeutic drug targets. The platform integrates publicly available datasets and data generated by Open Targets to establish and score target-disease associations. It also includes annotation information about targets, diseases, phenotypes, and drugs.

In addition, HCDT database, PubChem database, DrugBank database, ChEMBL database, and others introduced in previous chapters can also be used as drug-target databases.

References

1. Cambrey, N., Ghang, D.A.: Accessibility of drug databases and resources with assistive technology. Curr. Pharm. Teach. Learn. **14**, 1081–1084 (2022)
2. Barnett, M.J., Khosraviani, V., Doroudgar, S., Ip, E.J.: A narrative review of using prescription drug databases for comorbidity adjustment: a less effective remedy or a prescription for improved model fit? Res. Social Adm. Pharm. **18**, 2283–2300 (2022)
3. Wishart, D.S., et al.: DrugBank: a knowledgebase for drugs, drug actions and drug targets. Nucl. Acids Res. **36**, D901–D906 (2008)
4. Masaki, A., Makoto, M., Yutaka, S.: Using drug descriptions and molecular structures for drug–drug interaction extraction from literature. Bioinformatics **37**, 1739–1746 (2021)
5. Nagashima, K., et al.: Analysis of the toxic and lethal doses of one over-the-counter drug product in humans and the ingredients that may be abused: Building a drug database to prevent drug overdoses. Global Health Med. **7**, 49–56 (2025)
6. The UniProt, C.: UniProt: the universal protein knowledgebase in 2025. Nucl. Acids Res. **53**, D609–D617 (2025)
7. Caffarena, E.R., daCostaGomes, M.F., Martins, R.S.: Combining network-based and matrix factorization to predict novel drug-target interactions: a case study using the Brazilian natural chemical database. Curr. Bioinform. **17**, 793–803 (2022)
8. Galletti, C., Bota, P.M., Oliva, B., Fernandez-Fuentes, N.: Mining drug–target and drug–adverse drug reaction databases to identify target–adverse drug reaction relationships. Database **2021**, baab068 (2021)
9. Liu, T., et al.: BindingDB in 2024: a FAIR knowledgebase of protein-small molecule binding data. Nucl. Acids Res. **53**, D1633–D1644 (2025)

Chapter 12
Molecular Docking

Drug design is a process of creating new drugs by applying research achievements in life sciences such as molecular biology and biochemistry. It involves designing rational drug molecules by studying the interactions between target sites and ligands in these achievements. With the advancement of technology, the method of designing drugs using computer technology has gradually matured [1]. Molecular docking is a theoretical simulation method that predicts the binding affinity between molecules by studying their mutual interactions. In recent years, with the advancement of technology, molecular docking has gradually become an important method in the field of computer-aided drug design.

12.1 The Concept of Computer-Aided Drug Design

Drug design is a highly complex and time-consuming task. Generally speaking, it takes about ten years for a new drug to go from research to commercialization, and the funds spent during this process are also considerable. Therefore, pharmaceutical companies and researchers all aim to minimize time and cost. Molecular docking not only can select suitable candidate drugs through specific filters, but also can detect the potential toxicity and side effects of drugs [2].

Based on the continuous improvement of the interaction principle between ligands and receptors, a new method for developing new drugs has emerged, Computer-Aided Drug Design (CADD). This method utilizes the fundamental principles of computational chemistry to simulate the correlation between the molecular structure of new drugs and active substances, and rationally set up new drug design schemes for new structures or lead compounds. In recent years, with the vigorous development of theoretical calculation formulas and molecular graphics technology, CADD technology has received increasing attention and has become an indispensable powerful tool in modern new drug research and development.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_12

CADD is an interdisciplinary integration of mathematics, chemistry, pharmacy, and computer science, and has gradually developed. As a mature and emerging scientific research field, it has now become an important technical means in new drug research and development. Generally, in computer-aided drug molecule design, it is first necessary to understand the type of disease targeted by the molecule, the pathogenesis, and the biochemical reaction indicators of the human body under pathological conditions. Secondly, based on certain abnormal physiological and biochemical reactions that occur in the pathological state, the therapeutic targets are determined, which are mostly large biomolecules with clear functions; then, through modern biological techniques [3]. The three-dimensional structure of the drug target is obtained through multiple steps, such as expression, separation, purification, and crystallization, and the structural information is stored in a specific file format. Subsequently, professional software is used to read, calculate, and statistically analyze the file to obtain detailed information about the drug molecule binding pocket, guiding the design of drug molecules. Finally, the designed molecules are scored through computer simulation and undergo structural optimization. After optimization, they are scored again and optimized again. After several rounds of this cycle, the final information results can be used to guide the synthesis of physical compounds and subsequent pharmaceutical research. If the structural information of the biomolecular target cannot be obtained, the binding mode of the drug and the target can also be indirectly inferred by analyzing the structural characteristics of known active drugs, guiding the design of drug molecules and the modification of compound structures.

In the 1980s, the method of CADD was proposed. Nowadays, with the rapid development of genomics, more and more genes related to diseases are known, and the number of targets that can interact with drugs has also increased; with the rapid development of computer technology, various methods of computer-aided drug design have been proposed one after another. According to whether the receptor structure is known, computer-aided drug design can be divided into direct drug design and indirect drug design [4].

The direct drug design approach, which is based on the structure of the target, can be carried out once the three-dimensional structure of the receptor or the complex formed by the receptor and ligand is discovered. If only the amino acid sequence of the receptor protein is known but not its structural arrangement, the homology protein modeling technique can be used to model its three-dimensional structure based on the known three-dimensional structure of a homologous protein, that is, to observe the structural consistency between the receptor and the homologous protein. However, all the above-mentioned detection methods have certain errors. In fact, only a few receptors have their structures determined at present, and they are generally detected by X-ray diffraction technology, which measures the three-dimensional molecular structure in the crystalline state. Since many receptor proteins are difficult to crystallize, their molecular structures cannot be detected by diffraction methods. In the solution environment, the receptor may not have a crystalline structure when interacting with the ligand. Therefore, although multi-dimensional nuclear magnetic resonance technology can also detect the conformation in the solution, it is still

technically limited to proteins with a relative molecular weight of less than 40,000 Da [5].

The commonly used methods in direct drug design include molecular docking and de novo design.

(1) Molecular docking is a technique for the mutual pairing, recognition and interaction between small molecule ligands and receptors. The main theoretical basis of molecular docking is the receptor theory: one is the occupancy theory, which holds that the active substances of drugs must first approach the target molecules closely and then cooperate in the required regions, and these cooperations reflect the complementarity of the receptors, namely, stereo complementarity, electrostatic complementarity and hydrophobic complementarity. The other is the induced-fit theory, which states that the conformations of small and large molecules are appropriately altered to obtain a stable complex conformation. Therefore, molecular docking is the process of determining the position, orientation and conformation of molecules in the complex [6].
(2) De novo design is a new drug design approach based on the structure of the receptor. According to the shape and characteristics of the active site in the receptor, computer methods are used to find active molecules that are complementary to the shape and characteristics of the target site from the entire chemical space occupied by compounds. These designs mainly rely on the three-dimensional structural features of the target receptor and involve searching and comparing in three-dimensional structure databases. The new ligand design strategy can be used to design new structures that are suitable for the active sites of target proteins [7].

Indirect drug design methods refer to the approach where, in the absence of the three-dimensional structure of the receptor, various bioactive molecules with activity at the same target are calculated and analyzed through computer technology to obtain a three-dimensional conformational relationship model. The spatial configuration of the receptor is inferred by displaying the conformations on a computer, and the three-dimensional structure of the receptor is virtually constructed for drug design. Therefore, it is also known as ligand-based drug design.

12.2 Principles and Classification of Molecular Docking

This section summarizes the core principles of molecular docking and how docking methods are commonly classified. In general, docking combines conformational sampling with scoring to predict plausible receptor–ligand binding poses. The following subsections first introduce the principles and then compare typical classification schemes.

12.2.1 Principles of Molecular Docking

Molecular docking predicts the best binding mode of ligands and targets based on their three-dimensional structures. Based on the three-dimensional structures of ligands and targets, molecular docking can predict the conformation of complexes and binding affinities. The process of molecular docking can be divided into two steps. The first step is sampling, which generates different conformations through the rigid three-dimensional structure of the ligand. During sampling, the conformational space of the ligand molecule needs to be explored to obtain all theoretically valid conformations. The second step is scoring, which assesses the binding affinity when the protein and ligand interact in a certain conformation. Although these are two different steps, they are also related. Scoring functions are usually used to guide sampling [8].

More than 100 years ago, Fisher proposed the "lock and key model" concept, which is considered the earliest idea in the history of molecular docking. According to the "lock and key model", drugs and human proteins or receptors have a recognition relationship similar to that of a key and a lock, and this recognition mainly relies on the geometric matching of the two. Therefore, this model is also called the receptor model. However, with the development of the "lock and key model", it was later discovered that when drugs bind to receptors, conformational changes occur. In the 1950s, Koshland proposed the "induced-fit" theory in the process of molecular recognition, stating that during the binding process of receptor molecules and ligand molecules, the receptor (or ligand) molecules will adopt a conformation that can ultimately bind to the ligand (or receptor) molecules to adapt to each other, thereby achieving the most comprehensive binding [9].

With the advancement of technology, the understanding of the interaction between receptors and ligands has been deepening continuously. For instance, it has evolved from the rigid model based on geometric matching to the flexible model that combines geometric and energy matching. Moreover, with the rapid development of computer technology and computational science, people have begun to handle large-scale data analysis problems. These two reasons have jointly led to the emergence of new molecular docking methods. Molecular docking is much more complex than the "lock and key" model. This approach starts from the known monomer structures of two molecular types to find the best binding mode between them. For well-bound receptors and ligands, they must meet the following complementary matching criteria: the principle of geometric shape complementarity, that is, there is a large contact volume between the complexes; complementary matching of electrostatic interactions; the complex interface contains as many hydrogen bonds and salt bridges as possible; and complementary matching of hydrophobic interactions.

The protein–protein docking method involves placing a monomeric protein molecule with a known three-dimensional structure at the active site of the target, and then optimizing the position and conformation of the compound to find the best interacting conformation. Through computer simulation, the binding mode and three-dimensional structure of the complex can be predicted [10].

The main aspects of molecular docking methods are twofold: one is to conduct a meticulous spatial search to identify a large number of complex conformations; the other is to score and evaluate these different conformations and select the most suitable one.

12.2.2 Classification of Molecular Docking

There are various classification methods for molecular docking. According to the complexity of different models, it can be roughly divided into three categories: rigid docking, semi-flexible docking, and flexible docking [11].

Rigid docking refers to the situation where the conformation of the studied system does not change during the docking process. A representative example is the FTdock molecular docking algorithm developed by the Katchalski-Katzir group. Rigid docking is suitable for large-scale projects such as protein–protein or protein-nucleic acid docking, as the calculation is relatively rough and the principle is relatively simple.

Semi-flexible docking refers to the process where the research system (especially the conformation of the ligand) is allowed to change within a certain range during the docking process. Representative methods include DOCK developed by Kuntz, and AutoDock developed by Olson. Semi-flexible docking is suitable for handling the docking between small molecules and large molecules. During the docking process, since small molecules are flexible, their conformations can change, while large molecules are rigid, and their conformations cannot be altered, such as target enzymes, where only the amino acids at the binding site are flexible, and the rest are rigid. In drug design, semi-flexible molecular docking methods are commonly used, and their computational efficiency is relatively high.

Flexible docking refers to the process in which the structure of the research system can be freely altered during the docking process. A representative method is the molecular docking method based on the principles of molecular mechanics and molecular dynamics developed by Accelrys, a company in the United States. Flexible docking is typically used to accurately describe the recognition between molecules. However, due to the continuous structural changes during flexible docking, this method usually takes a considerable amount of time [12].

According to the states of the two molecules during the docking process, molecular docking can also be classified into three types: docking in the bound state, docking in the unbound state, and pseudo-unbound state docking. In docking in the bound state, both molecules are directly taken from the corresponding parts of the complex crystal structure, that is, the bound state structure. In docking in the unbound state, the initial conformations of the ligand and the receptor are derived from the monomer conformations of the two molecules before the complex formation, that is, the unbound state conformations. Pseudo-unbound state docking lies between the former two, with one molecule coming from the unbound state and the other from the bound state [13].

12.3 Molecular Docking Operation Process

There are currently a large number of software available for molecular docking, including both commercial and free software, such as DOCK, Autodock, and Glide, etc. This section will introduce the operation steps for molecular docking using AutoDock4 [14].

1. Prepare protein/small molecule structure files:

AutoDock comes with a visualization software called AutoDock Tools, which can help us prepare the necessary structure files. AutoDock supports file formats such as.pdb and.pdbqt. However, the small molecule files we download are usually in the.sdf format. Therefore, we will convert both the protein and small molecule files to the.pdbqt format for use here.

(1) Prepare the protein in the.pdbqt format. First, download the required protein file from the PDB; then, prepare the structure file (remove water, solvent molecules and unnecessary metal ions, add/remove hydrogen atoms, repair missing side chain atoms of residues, keep one chain for homodimers, etc.).

Here, the Pymol software needs to be used. Pymol is one of the few open-source visualization tools that can be applied in the field of structural biology, and it is suitable for creating high-quality 3D structure images of small molecules or biological macromolecules [15].

Load the protein file we have downloaded in PyMOL. In the PyMOL command box at the top left, enter the command "fetch 1iep" (fetch + protein name). After a short while, the protein structure will be displayed. Here, you can delete the unnecessary structures, such as chloride ions and water molecules. Right-click on the part you want to delete and select "remove" from the pop-up shortcut menu, as shown in Fig. 12.1.

Since the protein structure is a homodimer with two identical chains, only one chain can be retained. Enter "remove chain B" in the command box to delete chain B. Enter "h_add" in the command box to add hydrogen atoms. After these operations are completed, the final structure needs to be saved. Click File → Export Molecule, and in the pop-up Save Molecule dialog box, click the Save button. In the pop-up Save Molecule As dialog box, save the file as 1iep_H.pdb.

Next, you can proceed with the operation in AutoDock Tools. Click on Grid → Macromolecule → Open, and you will see the existing files. Select the required file 1iep_H.pdb, and then click the Open button. You will be able to view the structure of the protein, and a dialog box will pop up indicating that non-polar hydrogen atoms have been merged, etc. Click the OK button, and a dialog box for saving the file will appear. Here, save the protein as a.pdbqt file and name it 1iep_receptor.pdbqt. Finally, click the Save button. As shown in Fig. 12.2, the.pdbqt format file of the protein has been set up successfully.

(2) Prepare the small molecule in the.pdbqt format file. First, load the downloaded small molecule file in PyMOL, as shown in Fig. 12.3. The displayed structure

Fig. 12.1 Deleting unnecessary structures

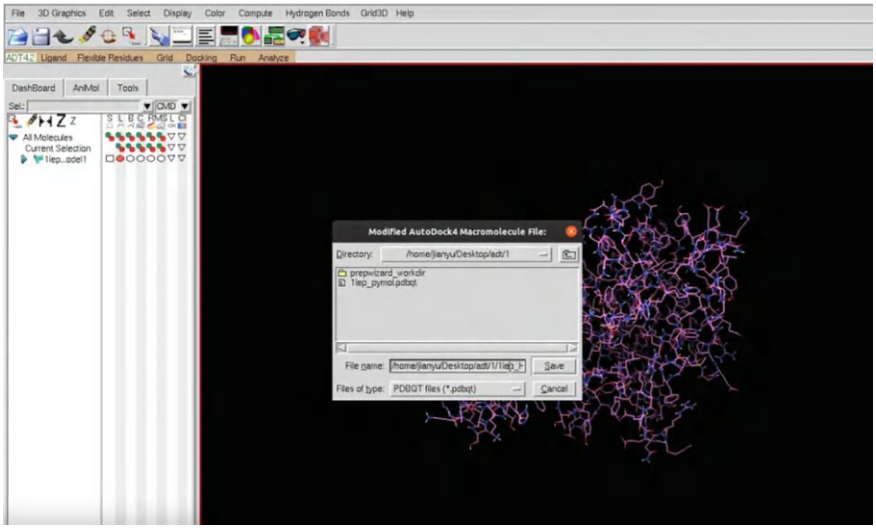


Fig. 12.2 Generation of.pdbqt format file

Fig. 12.3 Loading small molecule files

contains not only the small molecule but also the protein. Therefore, we need to retain only the small molecule structure.

Select the small molecule structure as shown in Fig. 12.4, click the A button in the sele on the right, copy to object → new, and copy this small molecule to a new project.

Next, only the small molecule structure remains. As shown in Fig. 12.5, enter the command "h_add" to add hydrogen atoms. After completion, click File → Export Molecule. In the pop-up Save Molecule dialog box, click the Save button. In the subsequent Save Molecule As dialog box, save the file as "ligand_H.pdb".

Import the small molecule file into AutoDock Tools as shown in Fig. 12.6. Open AutoDock Tools, click on Ligand → Input → Open, select the ligand_H.pdb file in the pop-up dialog box, click the Open button, and then click OK in the newly popped-up dialog box. The program will automatically perform operations such as adding charges.

Now the atom types of the small molecule have been corrected and all the required charges have been added. The next step is to define the Torsion to inform AutoDock which bonds can rotate. Click on Ligand → Torsion Tree → Detect Root, and a spherical structure will appear at the center of the small molecule, indicating that this position is the center of the small molecule. Click on Ligand → Torsion Tree → Choose Torsions again, and a new window will pop up (see Fig. 12.7).

The color of the bonds of small molecules also changes. Red indicates that the bond cannot rotate freely, while green indicates that it can. Additionally, the pink ones are amide bonds, and their color can be changed. Press and hold the Shift key

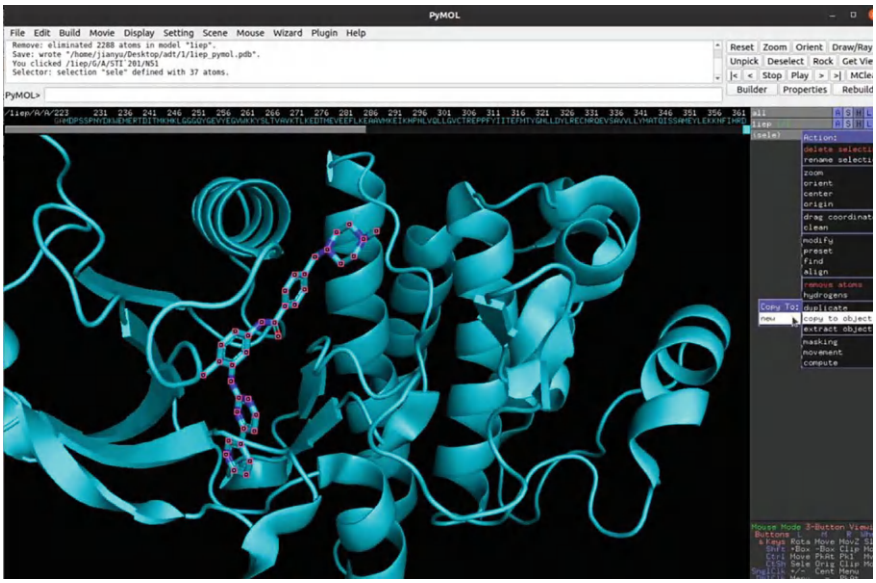


Fig. 12.4 Selecting small molecules

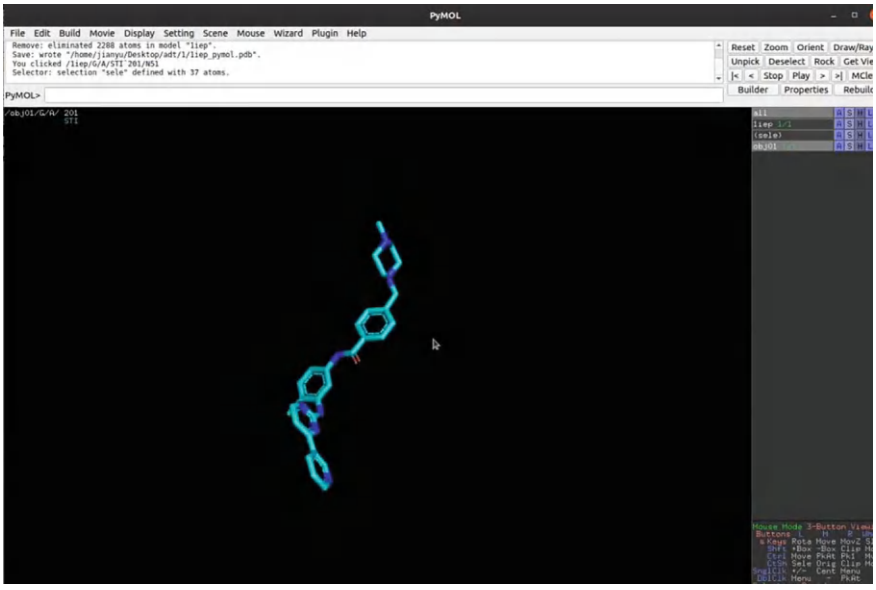


Fig. 12.5 Structure of small molecules

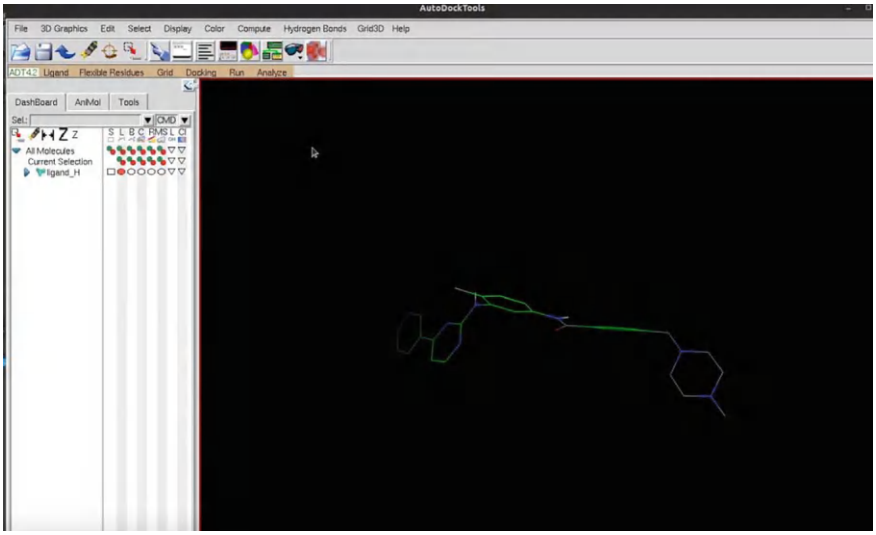


Fig. 12.6 Importing small molecule files into AutoDock Tools

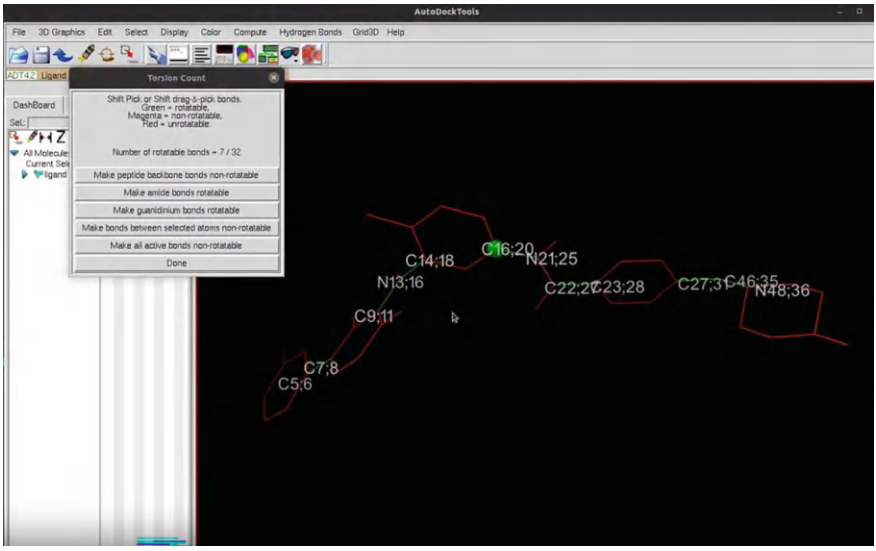


Fig. 12.7 Setting rotatable dihedral angles for small molecules

and click on the bond to change its color to green, and then it can be rotated freely. You can directly accept the default settings and click the Done button.

Click Ligand → Output → Save as PDBQT, and in the pop-up dialog box, name it as 1iep_ligand.pdbqt. Then click the Save button to save it. In this way, the.pdbqt format file of the small molecule is set up.

2. Perform molecular docking using AutoDock4:

AutoDock4 consists of two programs: one is AutoGrid; the other is AutoDock. AutoGrid can pre-compute the grid maps of interaction energies for various atom types of macromolecules, which are then used in the docking calculations of AutoDock, significantly saving time.

Before running these two programs, it is necessary to generate the parameter files they will use, namely the gpf file and the dpf file. Molecular docking consists of 8 steps: setting the path, loading files, generating the docking box, generating the gpf file, running AutoGrid (calculating the grid map), setting parameters, generating the dpf file and running AutoDock (docking).

Open AutoDock Tools, click File → Preferences → Set, and in the pop-up dialog box, set the working path. This working path will store the input files you need and the executable files required to run the program.

Load the protein/small molecule files (see Fig. 12.8). Click on Ligand → Input → Open, and then load the 1iep_ligand.pdbqt file. After that, click on Grid → Macromolecule → Open, and load the 1iep_receptor.pdbqt file. A dialog box will pop up asking whether to retain the previous charges. Just click the Yes button. Two more dialog boxes will appear subsequently; click OK on both of them.

Generate the parameters for the docking box. Click Grid → Set Map Types → Choose Ligand, then select the 1iep_ligand file in the pop-up dialog box. Next, click Grid → Grid Box. At this point, you will see a docking box, as shown in Fig. 12.9.

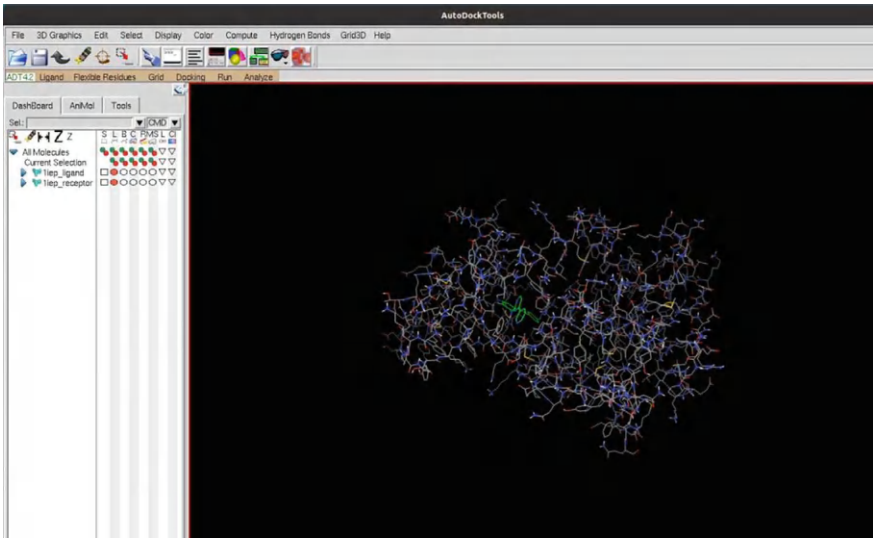


Fig. 12.8 Loading protein/small molecule files

If the position of the docking box is not ideal, you can click Center → Center on ligand. At this time, the size and position of the docking box will change, and the small molecule will be precisely enclosed within the docking box. Then click File → Close saving current. In this way, the size and position of the docking box will be saved.

Click Grid → Output → Save GPF to save it as a GPF file, name it 1iep.gpf, and click the Save button to save it. Then start running AutoGrid by clicking Run → Run AutoGrid. In the pop-up dialog box, select the 1iep.gpf file in the Parameter Filename field. The Log Filename field will automatically change to 1iep.glg, and the save path will be the same as that of the input file, as shown in Fig. 12.10. Then click the Launch button to start the run.

After the operation is completed, open the storage folder and you will see that there are now many more files inside, as shown in Fig. 12.11. Among them, the.map files are the grid maps generated by AutoGrid, which will be used in the AutoDock program.

Generate the parameters required by AutoDock and inform AutoDock which are the ligand and receptor protein to be docked. This molecular docking is a semi-flexible docking, where the protein is rigid and the ligand is flexible. Click on Docking → Macromolecule → Set Rigid Filename, and in the pop-up dialog box, select the 1iep_receptor.pdbqt file, then click the Open button. Next, click on Docking → Ligand → Choose, and in the pop-up dialog box, select 1iep_ligand. Then, in the new pop-up dialog box, click the Accept button.

Define some methods used for docking. Click Docking → Search Parameters → Genetic Algorithm. In the pop-up dialog box, you can modify the parameters.

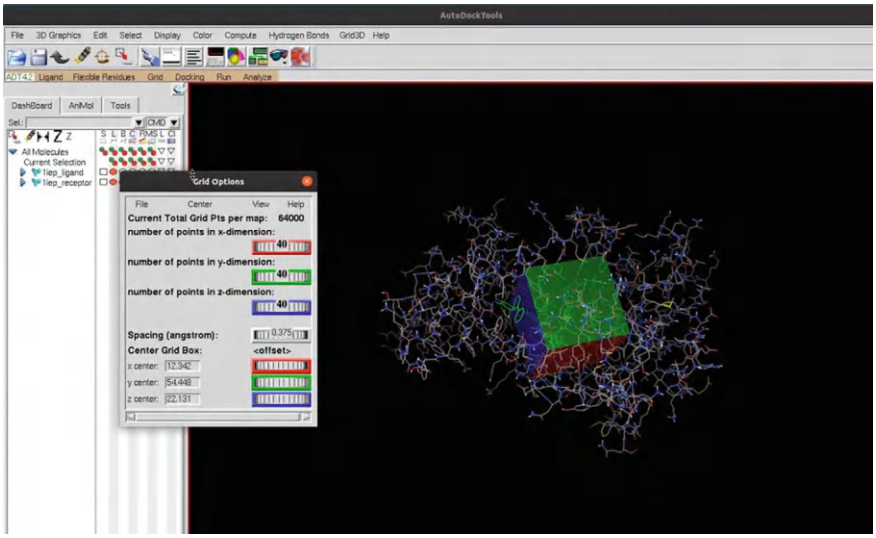


Fig. 12.9 Docking box

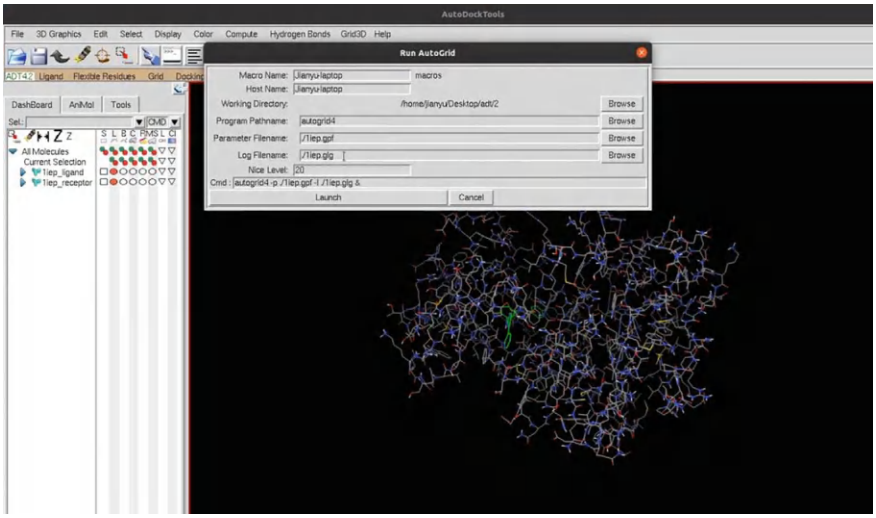


Fig. 12.10 Generation of GPF file

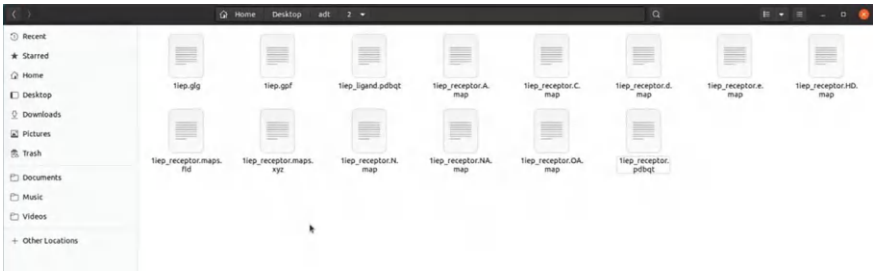


Fig. 12.11 Storage folder

Generally, only the number of runs in the first column needs to be changed. Here, change it to 50. After the modification is completed, click the Accept button. Next, output and save the completed parameters as a.dpf file. Click Docking → Output → Lamarckian GA. In the newly popped-up dialog box, name it 1iep.dpf and click the Save button.

Run AutoDock, click Run → Run AutoDock. In the pop-up dialog box, select the 1iep.dpf file in the Parameter Filename field, and the Log Filename field will automatically change to 1iep.dlg. Click the Launch button to start the operation. Then, just wait for the results to be generated. After generation, click Analyze → Conformations → Play, and in the newly popped-up panel, click the play button to view the conformations of these 50 molecular docking attempts. Thus, the semi-flexible molecular docking is completed. The molecular docking results are shown in Fig. 12.12.

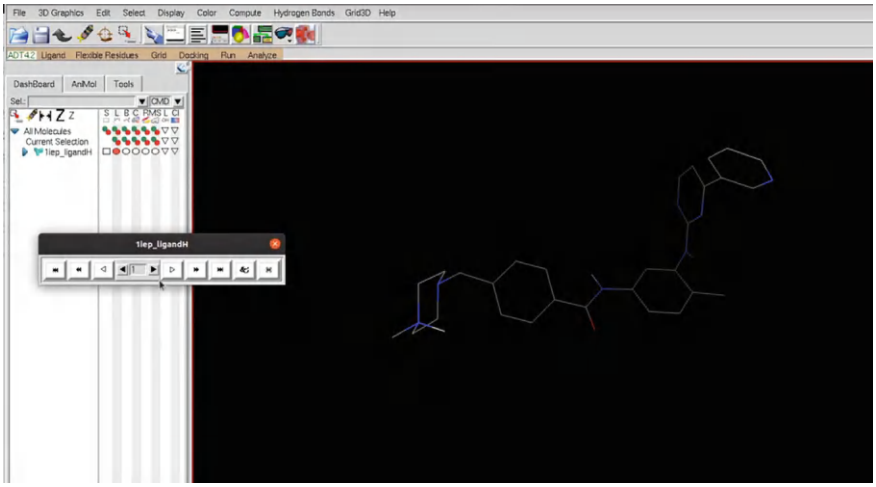


Fig. 12.12 Molecular docking results

12.4 Application of Artificial Intelligence in Molecular Docking

With the continuous development of artificial intelligence technology, AI has gradually permeated various fields, and interdisciplinary research has attracted increasing attention from more researchers. In the field of drug design, machine learning has been widely applied in various areas, including drug activity prediction, protein and peptide interaction prediction, drug similarity prediction, and protein–ligand binding affinity prediction. The general methods of machine learning include deep learning, neural networks, random forests, and support vector machines [16].

12.4.1 Scoring Function

All the software for molecular docking can rank the binding conformations of ligands according to the scoring function, whose purpose is to evaluate the binding free energy. Since the calculation of the binding free energy is computationally intensive, all scoring functions can provide relatively accurate scores to evaluate the results of molecular docking simulations. Additionally, scoring functions can also guide the sampling algorithm. Scoring functions can be classified into three categories, namely empirical scoring functions, physics-based scoring functions, and knowledge-based scoring functions.

1. Score function space

Based on the scoring function, we can obtain the optimal conformational model of protein–ligand binding. For the protein space, it was initially defined by sequence. Later, it was found that the definition of the protein space should also consider whether similar protein structures tend to fold. Therefore, for the chemical space of ligands, in research, it can be regarded as an aggregate of small molecule conformations. Each set containing the protein space and the chemical space can be regarded as an aggregate. All scoring functions can be regarded as existing within a scoring space. If there is a scoring function space that can predict the binding affinity between the set of protein space and the set of chemical space, then through computational research, this scoring function space can become the best scoring function for proteins [17].

2. Physics-based scoring function

Among the physics-based scoring functions, force field-based scoring functions are commonly used. These functions evaluate the free energy based on the weighted sum of energy terms. The selection of such functions depends on the force field used, which typically includes AMBER, GROMOS, OPLS, and CHARMM, among others. Physics-based scoring functions are widely used due to their relatively high accuracy in measuring atomic distances and calculating binding energies. Research has shown that among various physics-based scoring functions, those based on quantum mechanics have a promising future [18].

3. Experience-based scoring function

The empirical scoring function does not require extensive computations but achieves its purpose by evaluating the weighted sum of various parameters (such as the number of hydrogen bonds, hydrophilicity, hydrophobicity, etc.). The empirical scoring function can yield assessment results relatively quickly.

4. Knowledge-based scoring function

Initially applied in protein structure prediction, the scoring function uses statistical mechanics to obtain the complex structure of proteins and ligands. The binding free energy is expressed as an additive function, and the contribution to the binding free energy is calculated based on the distance between molecules.

12.4.2 Machine Learning in Protein–Ligand Molecular Docking

Machine learning can train new scoring functions to score complexes based on the current optimized scoring function or the structure of the complex as input. Additionally, machine learning is sometimes used in classification models for binding site detection and virtual screening. However, once the data is determined, a machine learning model can be developed. Currently, machine learning is developing very

rapidly in the field of molecular docking, and various methods that have emerged in recent years have significantly improved the docking performance of molecular docking.

1. Linear regression

Linear regression is a fundamental application in machine learning, which can determine the weights of linear equations, etc. For instance, tools for analyzing affinity take the protein–ligand interaction as a set of data and then parameterize the affinity of the complex through machine learning methods [19].

2. Random forest

Random forest is the first method of machine learning applied to molecular docking. It is an ensemble method based on the results of decision tree ensembles. Random forest can be applied to discrete value classification, continuous value regression, unsupervised learning clustering, outlier detection, etc.

3. Support vector machine

Before the development of deep learning, support vector machines were a commonly used method in machine learning. When the classification problem was introduced, using support vector machines as a model for regression, this process is known as the support vector machine regression model.

4. Convolutional neural network

The convolutional neural network contains a feature extractor composed of convolutional layers and subsampling layers. In each of its convolutional layers, there are multiple feature planes, and each feature plane is composed of multiple neurons. The neurons within the same feature plane can share weight parameters, that is, share convolution kernels. Convolution kernels are generally initialized in the form of random number matrices, and then the optimal weights are obtained through continuous training of the neural network.

12.4.3 Prediction Framework for Peptide-Protein Interactions Based on Deep Learning

Peptide-protein interactions are involved in a variety of fundamental cellular functions, and identifying them is crucial for designing effective peptide therapies. In recent years, many computational methods for predicting peptide-protein interactions have been developed. However, most of the existing prediction methods rely on high-resolution structural data.

Here, we introduce a deep learning-based framework for predicting peptide-protein interactions, CAMP [20]. The function of CAMP includes the prediction of peptide-protein interactions and the identification of corresponding peptide

binding residues. Comprehensive evaluations demonstrate that CAMP can successfully capture the binary interactions between peptides and proteins and identify the binding residues of peptides involved in the interactions. Moreover, CAMP outperforms other state-of-the-art methods in predicting peptide-protein interactions. CAMP can serve as a useful tool for predicting peptide-protein interactions and identifying important binding residues in peptides, thereby facilitating the discovery of peptide drugs.

This prediction method has constructed a comprehensive feature map of peptides and proteins based on their primary sequences, including secondary structure, hydrophobicity, hydrophilicity, polarity, intrinsic disorder tendency, and evolutionary information obtained from sequence alignment.

In addition, the framework designs a multi-channel feature extractor to learn the latent information of these physical, chemical and biochemical features. CAMP further utilizes convolutional neural networks and self-attention mechanisms to fully extract local and global information, predict the binary interaction of the input peptide-protein pairs, and identify the binding residues on the input peptide sequence. The rich and multi-level supervision information enables CAMP to accurately predict the interaction between peptides and proteins based solely on sequence-based input information. Through comprehensive evaluation on several benchmark datasets and an independent test dataset, it significantly outperforms other peptide-protein interaction prediction and state-of-the-art methods, and can accurately identify peptide binding residues. The ability of CAMP to handle three related tasks (peptide-protein interaction prediction, peptide-protein affinity estimation, and peptide virtual screening) is also examined, further demonstrating that CAMP performs better than baseline methods in handling these tasks. In summary, CAMP can provide a useful tool for predicting peptide-protein interactions using only sequence-based information as input.

The workflow of CAMP: Firstly, peptide-protein complex structures are extracted from RCSB PDB and known drug-target pairs are retrieved from DrugBank. Then, non-covalent interactions between peptides and proteins in each PDB complex are identified using ligand–protein interaction predictors, and only peptide-protein pairs with non-covalent interactions are retained as positive samples. Next, feature profiles based on the primary sequences of peptides and proteins are generated, along with residue-level structures and physicochemical properties, intrinsic disorder tendencies of peptides and proteins, and evolutionary information of proteins. Finally, multi-level labels, namely binary interaction labels of peptide-protein pairs and peptide-binding residue labels, are integrated for the training process.

The CAMP network architecture utilizes two multi-channel feature extractors to process the peptides and proteins obtained from the database, respectively; then, two convolutional neural network modules are employed to extract the hidden features of peptides and proteins. During this process, CAMP adopts the self-attention mechanism to understand the long-term dependencies between residues and the contribution of individual residues of proteins and peptides to the final interaction prediction. Finally, all the extracted features are combined, and three fully connected layers are used to predict whether there is an interaction between the given peptide-protein pair.

References

1. Ajaj, R., et al.: Aldose reductase inhibitory activity, molecular docking, ADMET, and density functional theory investigation of flavonoids isolated from *Euphorbia pulcherrima* Willd Ex Koltz. J. Mol. Struct. **1344**, 1–15 (2025)
2. Chen, J., Yu, X., Xiao, D., Xing, L.J.: Comments on "Integrating machine learning and molecular docking to decipher the molecular network of aflatoxin B1-induced hepatocellular carcinoma." Int. J. Surg. **111**, 6510–6511 (2025)
3. Durmaz, A., et al.: Phytochemical analysis, radical scavenging activity, and molecular docking studies of *Heliotropium sibiricum* (L.) JIM melo. Biomed. Chromatogr. **39**, e70147 (2025)
4. Godad, A., Sawant, R., Pahelkar, A.R., Pereira, G., Sathaye, S.J.M.N.: Exploring the potential mechanism of action of ursolic acid for Parkinson's disease: an integrative network pharmacology, docking and molecular dynamics study. Mol. Neurobiol. **21**, 1–14 (2025)
5. Cano-Gonzalez, C.N., et al.: Enzyme-assisted recovery of phenolic compounds from broccoli waste with antioxidant and antihyperglycemic activities evaluated through in vitro and molecular docking studies. Biocatal. Agric. Biotechnol. **121**, 103632 (2025)
6. Hartavi, A.J.I.C., Products, S.: Onosma obtusifolia extracts: antioxidant potential and molecular docking insights into acetylcholinesterase, α-amylase, and tyrosinase inhibition. Ind. Crops Prod. **233**, 121387 (2025)
7. Kali, V., Pasuparthy, S.D., Maiti, B.J.T.: [CMMIM][BF4] ionic liquid catalyzed efficient synthesis of henna based xanthene derivatives and their photophysical, theoretical and molecular docking studies. Tetrahedron **12**, 134774 (2025)
8. Dascălu, A.E., Ghinet, A., Boulanger, E., Shova, S., Lipka, E.J.C.: Molecular docking observations on enantiomeric retention trends and selection of chiral stationary phase. Chirality **37**, e70042 (2025)
9. Mhamdi, M., et al.: GC/MS analysis of the chemical composition of petroleum ether and chloroform extracts from coffea arabica seeds, along with molecular docking evaluation of the extracts antibacterial and anticancer activities. Biomed. Chromatogr. **39**, e70132 (2025)
10. Nagdeve, R.D., et al.: Larvicidal activity, molecular docking, and structural analysis of methoxyindolizine-1-carboxylate derivatives. J. Mol. Struct. **32**, 142919 (2025)
11. Nofianti, K.A., Ekowati, J., Diyah, N.W., Hakim, L.J.J., et al.: Benzoxazinone derivatives as antiplatelet: synthesis, molecular docking, and bleeding time evaluation. J. Pharmacol. Pharmacother. **241**, 305440 (2025)
12. Sable, Y.R., et al.: Exploring pyrazole integrated thiazole molecular hybrids for antitubercular activity: synthesis, spectral characterization, DFT, ADME, and docking studies. J. Mol. Struct. **59**, 142891 (2025)
13. Seleem, H., et al.: Coordinating behavior of UO_2(II), VO(II) and Pd(II) ions towards a quinoline-based hydrazone: synthesis, spectroscopic, DFT, antitumor and molecular docking studies. Inorg. Chem. Commun. **101**, 114847 (2025)
14. Shaikh, M.Z., et al.: Structural investigations, theoretical analysis, chemosensing characteristics and molecular docking studies of t-butyl carbazate based novel aldimine. J. Mol. Struct. **1344**, 142761 (2025)
15. Slimani, I., et al.: Synthesis of new imidazolidine derivatives: characterization, anti-cancer potential and molecular docking. J. Mol. Struct. **46**, 142887 (2025)
16. Teke Tunçel, Ş, et al.: Synthesis, conformational analysis, antioxidant and enzyme inhibition activity, molecular docking, and DFT studies of new chiral hydrazide–hydrazone derivatives. Chirality **37**, e70041 (2025)
17. Varikuppala, P., Adusumalli, M.S.K., Merugu, R., Mittapelli, V.J.C.: Synthesis, biological evaluation, and molecular docking studies of ((S)-7-(Dibenzylamino)-3-(1-Substituted-Pyrrolidin-3-yl)-3H-Imidazo [4, 5-d] pyrimidines as anti-inflammatory agents. ChemistrySelect **10**, e00192 (2025)
18. Wang, J., et al.: Reduction mechanism of XOR-mediated nitroaromatics hypoxia targeted AGT inhibitors: molecular docking, MD simulation and ONIOM (QM/MM) investigation. Int. J. Biol. Macromol. **21**, 145344 (2025)

19. Zhu, S., et al.: Identification of a novel and high affinity MIF inhibitor via structure-based pharmacophore modelling, molecular docking, molecular dynamics simulations, and biological evaluation. J. Enzyme Inhib. Med. Chem. **40**, 2501378 (2025)
20. Lei, Y., et al.: A deep-learning framework for multi-level peptide–protein interaction prediction. Nat. Commun. **12**, 5465 (2021)

Chapter 13
New Applications of Deep Learning in QSAR

Molecules are the basic units that constitute matter. The structural features of molecules within a compound and the combination mode of atoms determine the properties of the compound. That is, the physical and chemical properties and biological activities of a compound are expressed and explained based on the molecule as the main body. When the molecular structure of a compound changes, its properties and biological activities will also change accordingly. Computer-Aided Drug Design (CADD) is an emerging discipline that applies molecular simulation technology to drug research and development and integrates with traditional pharmacology. It comprehensively uses various theories and computational methods to study the physical and chemical properties and movement behaviors of drug targets or small molecules. After obtaining relatively accurate and detailed physical, chemical and biological information, it guides drug design through the integration, transformation and statistical analysis of the obtained data. Quantitative Structure–Activity Relationship (QSAR) is an important research method in modern drug design, which uses mathematical models to explain the quantitative relationship between the chemical structure parameters of ligands and their biological activity intensity, optimize the structure of ligands and predict the biological activity of new compounds of the same type.

13.1 QSAR

This section will first clarify the core definition, evolutionary context, and research process of Quantitative Structure–Activity Relationship (QSAR), laying a foundation for further in-depth study.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_13

13.1.1 Definition of QSAR

Quantitative Structure–Activity Relationship (QSAR), is the use of mathematical models to quantitatively describe the relationship between the molecular structure of organic compounds (such as two-dimensional structure, three-dimensional structure, electronic structure, etc.) and biological effects (such as drug activity, toxicity, pharmacodynamic properties, pharmacokinetic parameters and bioavailability, etc.).

QSAR studies are one of the earliest rational drug design methods developed by humans, featuring small computational load and good predictive ability. In recent years, QSAR has not only been applied to predict biological activity but also widely used to predict the physicochemical properties and toxicity of compounds. This is mainly reflected in the design of new drug candidates, the prediction of compound toxicity, the elucidation of the mechanism of enzyme-chemical interactions, and the prediction of the biological activity of unavailable compounds and untested active compounds.

When the structure of the receptor is unknown, the QSAR calculation results can be used to carry out more targeted structural modifications on bioactive substances. Before the explosive development of computer technology in the 1980s, QSAR was the most widely used and almost the only rational drug design approach. As QSAR is a statistical empirical model, the predictive ability of the QSAR method is largely dependent on the quality of the data and the model's construction. Due to the limitations of the accuracy of experimental data, QSAR methods often face the criticism of "statistical method fraud". At the same time, they cannot clearly provide the physical meaning of the regression equation and the interaction mode between drugs and receptors. The ambiguity of physical meaning is also one of the main criticisms of QSAR methods [1].

13.1.2 A Brief Introduction to the Development of QSAR

In 1868, Scottish organic chemists Crum-Brown and Frazer proposed the fundamental idea of QSAR [2], suggesting that there exists a certain functional relationship between the molecular structure x of a compound and its biological activity y, as shown in Eq. (13.1).

$$y = f(x) \tag{13.1}$$

However, they did not find any examples to establish a clear functional model. The earliest quantitative structure–activity relationship was the Hansch equation proposed by Professor Corwin Hansch. During this period, different scholars successively studied the relationship between chemical structure and biological activity. For instance, around 1900, German pharmacologist Meyer and British physiologist Overton respectively measured the solubility of inhalation anesthetics such as Et_2O,

N_2O and $CHCl_3$ in olive oil, and compared the relationship between their lipophilicity and anesthetic efficacy. The results showed that the lipophilicity of the molecule was positively correlated with its anesthetic efficacy. In 1939, Ferguson expressed the quantitative relationship between chemical structure and biological effect by using Eq. (13.2).

$$\lg \frac{1}{c_i} = m \lg A_i + k \tag{13.2}$$

Among them, c_i represents the concentration of the i-th compound that produces a specified biological effect; A_i represents the physicochemical parameters of the compound such as solubility (S), lipid-water partition coefficient ($\log P$), or vapor pressure (P); the coefficient m and the constant k are characteristic values of this type of compound in a specific biological system.

In 1962, Professor Hansch established the Linear Free Energy Relationship model (LFER) [3] based on the relationship between substituents and activity. At the same time, Free and Wilson [4] proposed the substituent contribution model of QSAR, marking the beginning of the 2D-QSAR era. The Hansch equation originated from the Hammett equation proposed by the American physical chemist Hammett in 1935 for calculating the dissociation constant of substituted benzoic acid, and the improved Taft equation in 1952 for calculating the rate constant of the hydrolysis reaction of esters. The Hammett equation is an empirical equation for calculating the dissociation constant of substituted benzoic acid, which establishes a linear relationship between the logarithm of the dissociation constant and the electronic parameter of the substituent group. The Taft equation is an empirical equation for calculating the rate constant of the hydrolysis reaction of aliphatic esters, which is improved on the basis of the Hammett equation and establishes a linear relationship between the logarithm of the rate constant and the electronic and steric parameters.

The Hansch method, the Free-Wilson method, and others are also known as two-dimensional quantitative structure–activity relationship (2D-QSAR), all of which consider the properties of a molecule as a whole. Because 2D-QSAR is not able to accurately describe the relationship between the three-dimensional structure of a compound and its biological activity, it is necessary to follow the development of computer technology led to the proposal of "3D-QSAR based on distance geometry" by Crippen [5] and "molecular shape analysis method" by Hopfinger in 1979–1980. In 1988, Cramer et al. [6] proposed the "Comparative Molecular Field Analysis (CoMFA) method" for drug design. Once introduced, the Comparative Molecular Field Analysis method quickly swept through the field of drug design and became a widely used drug design method based on quantitative structure–activity relationship (QSAR). In the 1990s, new three-dimensional QSAR methods, such as the "Comparative Molecular Similarity Indices Analysis (CoMSIA) method" [7], an improvement of the Comparative Molecular Field Analysis method, and the "virtual receptor method" [8], emerged based on 3D-QSAR of distance geometry. Thus, QSAR research entered the 3D era.

With the widespread use of computers, the research on QSAR has become increasingly diverse and rich since 1990. As more factors are taken into account in QSAR studies, QSAR is evolving towards higher dimensions. Romeiro et al. [9] introduced the concept of 4D-QSAR by incorporating the conformational integration parameters of compounds as the fourth dimension. Vedani and Dobler [10], Damale et al. [11] successively proposed the concepts of 5D-QSAR and 6D-QSAR by considering the induced fit effect between receptors and ligands as the fifth dimension and the hydration and de-hydration solvation effects during the interaction between receptors and ligands as the sixth dimension. However, in fields such as pharmacology and environmental toxicology, the application of 2D-QSAR and 3D-QSAR is more widespread.

13.1.3 QSAR Model Research Methods

Typically, the research content of QSAR mainly starts from four aspects: data set collection and preprocessing, molecular structure construction and optimization, calculation and selection of molecular descriptors, and model establishment and validation. The following is a detailed introduction to each aspect.

1. **Data Collection and Preprocessing of the Dataset**

The accuracy and reliability of the dataset play a very crucial role and have a decisive impact on the model. Generally, the data can be collected from published literature or public databases, or from within the research group or from collaborators. Currently, commonly used databases and websites include: (1) OECD QSAR Toolbox, which contains 57 databases, 84,291 chemicals and 2.4 million experimental data, covering physical and chemical properties, human health hazard data, etc.; (2) QSAR Databank (QsarDB), which contains over 500 QSAR models and the structural formulas of 40,000 compounds. For data preprocessing, duplicate data need to be removed. The biological activity data should preferably be obtained under the same or similar conditions. If the activity values in some datasets span several orders of magnitude, to obtain a better mathematical model, the activity parameters are usually converted to negative logarithms for statistical analysis, such as converting IC_{50} and LD_{50} to pIC_{50} and pLD_{50} forms. Finally, the dataset needs to be randomly divided into a training set and a test set. Generally, the ratio of the training set to the test set is 4:1, with the training set used for model training and the test set for model performance testing and improving the model's predictive ability.

In addition to the above-mentioned activity parameters, common ones also include the minimum inhibitory concentration (MIC), half-maximal inhibitory concentration (IC_{50}), median lethal concentration (LC_{50}), half-maximal effective concentration (EC_{50}), median lethal dose (LD_{50}), and median effective dose (ED_{50}), etc. All activity parameters must use the amount of substance as the unit of measurement to eliminate the influence of molecular weight and truly reflect the physiological activity at the molecular level.

2. **Construction and Optimization of Molecular Structures**

The substances in the dataset can be represented by chemical structures, commonly using methods such as nomenclature, CAS numbers, 2D structures, and 3D structures. Common software includes Chem Office, Chem Draw, ISIS Draw, and SYBYL, which can draw the real structure of molecules and generate electronic spreadsheets. However, 2D structures cannot accurately represent the spatial structure of molecules or distinguish chirality, while continuous optimization of 3D structures can make them very close to the actual molecules. At the same time, further optimization of the molecular structure is needed to make it closer to the real one.

3. **Calculation and Selection of Molecular Descriptors**

The molecular structure features of compounds are transformed into various parameters, which are used to represent the molecular structure. That is, various descriptive parameters are obtained from the molecular structure to express the structural characteristics of substances. Such descriptive parameters are called descriptors. Generally, commonly used descriptors mainly include experimental descriptors and theoretical calculation descriptors. Experimental descriptors include lipid-water partition coefficient, molar refractivity, dipole polarizability, etc. Theoretical calculation descriptors include composition descriptors, topological descriptors, quantum chemical descriptors, charge-related descriptors, and geometric descriptors, etc. Among them, composition descriptors can reflect the structural information of molecular composition, such as the number of bonds, the number of rings, atomic weight, etc. of the molecule; topological descriptors refer to the connection situation between atoms in the molecule, reflecting the size and number of rings, such as Wiener index, Hosoya index, edge adjacency index and other types of indices; quantum chemical descriptors can reflect the orbital energy information of the molecule, such as the energy, reaction index and dipole moment of the molecule; charge-related descriptors reflect the charge distribution of the molecule, such as molecular polarity, maximum and minimum partial charges and molecular surface area and other parameters; geometric descriptors can reflect the shape of the molecule, such as molecular volume, molecular moment of inertia and surface area and other information.

At present, there are many software programs available for calculating molecular descriptors. This can be achieved either through dedicated descriptor calculation software such as Dragon and Power MV, or by using the descriptor calculation modules in molecular simulation software packages like SYBYL and Cerius2.

Before variable selection, it is necessary to conduct a pre-screening of these descriptors to reduce redundant and irrelevant information. This not only reduces the computational load but also avoids the interference of molecular descriptors that are not related to activity. At the same time, it can reduce the accidental correlation in modeling. Traditional variable selection methods mainly include heuristic search algorithms, principal component analysis, and least squares methods. With the wide development of computer technology, swarm intelligence algorithms such as genetic algorithms, ant colony algorithms, and particle swarm algorithms have also been applied to the screening of molecular descriptors.

4. **Model Establishment**

In the early QSAR models, a formula was usually determined first, and then the coefficients of the formula were solved by using multivariate statistical analysis methods in chemometrics, including Multiple Linear Regression (MLR), Principal Component Analysis (PCA), and partial least squares regression. Multiple linear regression is a classic modeling method that can optimize the activity of lead compounds. Principal component analysis was first proposed by Hotelling in 1933 and is used to select several principal components that have a significant impact on activity to establish a QSAR model. Partial Least Squares (PLS) regression combines the advantages of multiple linear regression and principal component analysis, considering both independent and dependent variables simultaneously, and proposes a regression modeling method for multiple dependent variables and multiple independent variables, which better reflects the overall nature of the model.

With the development of computer technology, machine learning and deep learning methods such as Artificial Neural Networks (ANN), Genetic Algorithms (GA), and Support Vector Machines (SVM) are often used to build models. These methods are not only used to construct quantitative regression models but also qualitative classification models.

5. **Model Validation**

Refers to the evaluation of the model. Common statistical indicators for evaluating the fitting ability of QSAR models include the correlation coefficient r^2, root mean square error (RMSE), and Fisher test value F. The larger the r^2 and F values, and the smaller the RMSE value, the better the fitting ability of the model. The size of the cross-validation correlation coefficient q^2 is used to evaluate the predictive ability of the model. Generally, the larger the q^2 value, the better the predictive ability of the model. The relevant formulas are shown in Eqs. (13.3)–(13.6).

$$r^2 = 1 - \frac{\sum \left(y_{exp} - y_{cal}\right)^2}{\sum \left(y_{exp} - y_{mean}\right)^2} \tag{13.3}$$

$$RMSE = \sqrt{\frac{\sum \left(y_{cal} - y_{exp}\right)^2}{n-1}} \tag{13.4}$$

$$F = \sqrt{1 - \frac{r^2(n-k-1)}{k(1-r)}} \tag{13.5}$$

$$q^2 = 1 - \frac{\sum \left(y_{exp} - y_{pred,CV}\right)^2}{\sum \left(y_{exp} - y_{mean}\right)^2} \tag{13.6}$$

Here, n represents the number of samples; k represents the number of variables; y_{exp} represents the experimental value of the compound's property; y_{cal} represents the calculated value of the compound's property; y_{mean} represents the average of the

experimental values of the property for all compounds in the training set; and $y_{pred,CV}$ represents the predicted value of the compound's property in the cross-validation.

13.2 Traditional QSAR

This section will focus on and elaborate on the two core branches of traditional QSAR. First, it will analyze the fundamental principles of 2D-QSAR, highlighting the mathematical models, parameter meanings, and application characteristics of classic methods such as the Hansch equation; subsequently, it will delve into the core logic of 3D-QSAR, elaborating on the principles, calculation processes, and advantages of mainstream methods including Comparative Molecular Field Analysis (CoMFA) and Comparative Molecular Similarity Index Analysis (CoMSIA). Meanwhile, it will briefly introduce other 3D-QSAR methods such as distance geometry and molecular shape analysis, to demonstrate the diversity and applicable scenarios of traditional QSAR methods.

13.2.1 Basic Principles of 2D-QSAR

2D-QSAR is a drug design method that takes the overall structural properties of molecules as parameters to conduct regression analysis on their physiological activities and establish a correlation model between chemical structure and physiological activity. Common methods include the Hansch equation, the Free-Wilson method, and the Molecular Connectivity Index (MCI) method, among others. Among them, the Hansch equation is particularly well-known and widely applied.

Subsequently, Hansch, along with visiting Japanese scholar Tominaga Toshio and others, improved the mathematical model of the Hansch equation by introducing indicator variables, parabolic models, and bilinear models, among other corrections, which enhanced the predictive power of the equation.

Both the Hansch equation and the Free-Wilson method employ regression analysis. The Hansch equation is very similar in form to the Hammett equation and the Taft equation. It uses the median effective dose of physiologically active compounds as the activity parameter and the electronic, steric, and hydrophobic parameters of the molecule as the independent variables in linear regression analysis. The classic form of the Hansch equation is shown in Eq. (13.7).

$$\log\left(\frac{1}{C}\right) = a\pi + b\sigma + cE_s + k \tag{13.7}$$

Here, C represents the concentration of a compound that produces a certain biological effect within a given time period, such as the half-maximal inhibitory concentration

(IC_{50}), half-maximal effective concentration (EC_{50}), and The median lethal concentration, etc.; σ is the Hammett electronic parameter, E_s is the Taft steric parameter, a, b, c, and k are all regression coefficients, π is the hydrophobic coefficient of the molecule, and PH is the partition coefficient of the standardized compound, which has a relationship with the lipophilic-hydrophilic partition coefficient P_X of the molecule as shown in Eq. (13.8).

$$\pi = \log P_x - \log P_{xH} \tag{13.8}$$

Subsequently, Japanese scholar Toshio Fujita, along with Hansch and others, made certain improvements to the classic Hansch equation, introducing indicator variables, parabolic models, and bilinear models to modify the equation, as shown in Eq. (13.9).

$$\log\left(\frac{1}{C}\right) = a\pi + b\pi^2 + +c\sigma + dE_s + k \tag{13.9}$$

This model has a better fitting effect. The Hansch equation further describes the relationship between hydrophobicity and activity with a double linear model, as shown in Eq. (13.10).

$$\log\left(\frac{1}{C}\right) = a\log P - b\log(\beta P + 1) + D \tag{13.10}$$

Among them, P represents the lipophilicity of the molecule, a, b, and β are regression coefficients, and D stands for the other parts of the equation. The predictive ability of the double linear model is further enhanced compared to the parabolic model. Since all parameters in the equation are related to the free energy of the compound, the Hansch equation is also known as the linear free energy relationship method or the supramolecular thermodynamic correlation model.

In the Free-Wilson method, pure structural parameters are used. These parameters take a specific molecular structure as a reference standard and encode the molecular structure based on the presence or absence of functional groups on the structural parent ring. Regression analysis is then conducted to calculate the regression coefficient for each functional group, thereby obtaining a quantitative structure–activity relationship model.

Free and Wilson conducted a study on the correlation between the structural information of organic compounds and their biological activities using multivariate regression analysis, and established a method that does not require the physicochemical parameters of the compounds. This method assumes that the biological activity of a group of homologous compounds with the same parent nucleus is the sum of the activity contribution of the parent structure and the activity contributions of each substituent, and is thus also known as the group contribution method. The equation form of this method is shown in Eq. (13.11).

$$\log\left(\frac{1}{C}\right) = \sum_i \sum_j G_{ij} X_{ij} + \mu \tag{13.11}$$

Among them, X_{ij} is the structural parameter. If the i-th position in the structural parent ring has the j-th type of substituent, the structural parameter takes the value of 1; otherwise, it is 0. μ is the activity parameter of the reference molecule; G_{ij} is the regression coefficient.

The molecular connectivity method, also known as the "MCI method" [12], was first proposed by Randic in 1975 and further developed by Kier, Hall, and others, thus forming a complete series of molecular connectivity indices. This method characterizes the chemical structure of molecules using topological parameters, that is, it describes the structural properties of molecules by the arrangement or connection mode of the skeleton atoms within each compound molecule, and links the structure of the compound with its biological activity through multiple linear regression analysis. As a topological parameter, MCI has zero-order terms, first-order terms, and second-order terms, etc., which can be calculated from the molecular structural formula and have a good correlation with the toxicity data of organic compounds [13]. Although MCI has a significant advantage in reflecting the three-dimensional structure of molecules, its application is somewhat limited due to the lack of a clear physical meaning.

13.2.2 The Basic Principles of 3D-QSAR

3D-QSAR is based on the three-dimensional structural features of ligands and receptors, and uses mathematical models to describe the relationship between molecular structure and certain biological activities of molecules. Its basic assumption is that the molecular structure of a compound contains information that determines its physical, chemical, and biological properties. This method indirectly reflects the nonbonding interaction characteristics between drug molecules and macromolecules during their interaction process. Compared with 2D-QSAR, it has a more explicit physical meaning and richer information content. Since the 1980s, 3D-QSAR has gradually replaced the position of 2D-QSAR and become one of the main methods for mechanism-based rational drug design. However, both are based on energy changes, so the features described by the two methods can complement each other. The most commonly used methods in 3D-QSAR are comparative molecular field analysis and comparative molecular similarity indices analysis. In addition, there are distance geometry method (DG 3D-QSAR), molecular shape analysis (MSA), and virtual receptor, etc.

1. **Overview of CoMFA**

CoMFA is one of the widely used rational drug design methods. This method is mainly used to study the relationship between the biological activity of compounds and their molecular steric and electrostatic fields. Most applications are in the field

of protein–ligand interactions, describing affinity or inhibition constants. In 1979, Cramer and Milne first attempted to compare molecules by aligning them in space and mapping their molecular fields onto a 3D grid. In the following years, this method was further developed into the Dynamic Lattice Oriented Molecular Modeling System (DYLOMMS) method, but it was not widely accepted by the scientific community. In 1986, Svante Wold proposed using Partial Least Squares (PLS) analysis to correlate field values with biological activity. In 1988, Cramer proposed CoMFA, mainly replacing the PCA method with PLS and combining it with cross-validation.

The CoMFA study begins by superimposing molecules with the same structural parent ring in space to make their spatial orientations as consistent as possible. Then, a probe particle moves around the space surrounding the molecule, calculating the interaction between the probe particle and the molecule, and recording the energy values of the interaction at different spatial coordinates, thereby obtaining the molecular field data. Different probe particles can detect different properties of the molecular field around the molecule. For example, methane molecules as probes can detect steric fields, water molecules as probes can detect hydrophobic fields, and hydrogen ions as probes can detect electrostatic fields, etc. Some mature comparative molecular field programs can provide dozens of probe particles for users to choose from.

The molecular force fields considered in CoMFA include steric interactions (Lennard–Jones potential energy) and Coulombic electrostatic interactions (Coulomb potential energy). The formulas for the steric force field and the electrostatic force field are shown in Eqs. (13.12) and (13.13), respectively.

$$E_{vdm} = \sum_{i=1}^{n} \left(\frac{a_{ij}}{r_{ij}^{12}} - \frac{b_{ij}}{r_{ij}^{6}} \right) \tag{13.12}$$

$$E_c = \sum_{i=1}^{n} \left(\frac{q_i q_j}{D r_{ij}} \right) \tag{13.13}$$

Here, a and b are constants related to the atomic nature (which can be obtained from the Tripos force field); r_{ij}represents the charge between atom i and probe j; q_i and q_j respectively denote the charges carried by atom i and probe j; D is the dielectric constant of the medium.

The vast amount of molecular field information obtained through probe particle detection is used as an independent variable to participate in the regression analysis of molecular physiological activity data. Because the molecular field information data volume is large and it belongs to high-dimensional chemical data, data dimension reduction measures must be taken during the regression analysis process. The most commonly used method is partial least squares, and principal component analysis is also used for data analysis.

The results of statistical analysis can be graphically output on the molecular surface to guide researchers on how to selectively modify the structure of lead compounds. In addition to the intuitive graphical results, CoMFA can also obtain

regression equations to quantitatively describe the relationship between molecular fields and activity.

2. Overview of CoMSIA

CoMSIA was developed by Klebe et al. and is often used in drug research and development to discover important common features when binding to related biological receptors. In CoMSIA, spatial and electrostatic features, hydrogen bond donors, hydrogen bond acceptors, and hydrophobic fields are considered. CoMSIA is an improvement on the CoMFA method, as it modifies the calculation formula for the interaction energy between probe particles and drug molecules, thereby obtaining better molecular field parameters.

The similarity index $A_{F,k}^{q}(j)$ is calculated through the following formula for molecule j in the dataset at grid point q:

$$A_{F,k}^{q}(j) = -\sum_{i=1}^{n} w_{probe,k} w_{ik} e^{-\alpha r_{iq}^{2}} \tag{13.14}$$

Among them, A is the total sum of the similarity indices of all atoms i of molecule j at grid point q, $w_{probe,k}$ is the probe atom, w_{ik} represents the actual value of the physicochemical property k of atom i, r_{iq}^{2} indicates the square of the distance between the probe atom at grid point q and atom i of the molecule under investigation, and α is the attenuation factor, with a default value of 0.3 and an optimal value ranging from 0.2 to 0.4. A higher attenuation factor will cause the Gaussian function to be more jittery, the attenuation of the distance-dependent effect of molecular similarity to be stronger, and the importance of the overall molecular properties to be reduced.

Compared with CoMFA, the computational process of CoMSIA is roughly similar, as are the statistical methods and the display methods of the models. However, the description of the interaction fields, the number of fields, and the calculation functions are different. That is, CoMSIA uses similarity indices to describe a certain property of the interaction between the ligand and the receptor; meanwhile, on the basis of the steric field and the electrostatic field, it adds the hydrophobic field, the hydrogen bond donor field, and the hydrogen bond acceptor field. CoMSIA uses distance-dependent Gaussian functions to calculate the molecular similarity indices, effectively avoiding the huge potential energy jumps between adjacent grid points on the molecular surface.

3. Other 3D-QSAR Methods

Strictly speaking, distance geometry three-dimensional quantitative structure–activity relationship (QSAR) is a method that lies between two-dimensional and three-dimensional QSAR. This approach holds that the interaction between a drug and its receptor is achieved through the direct interaction of the drug's active groups with the corresponding binding sites on the receptor. The activity of a drug is measured by the binding energy between its active groups and the receptor binding sites, which is related to the nature of the drug's active groups and the type of receptor

binding sites. By choosing a reasonable distribution model of receptor binding sites and the binding mode of drug molecules, and achieving the best fit with experimental data, a set of energy parameters related to the drug's active groups and the type of receptor binding sites can be obtained. After determining the binding mode of a new compound, these energy parameters can be used to quantitatively predict its binding energy and thereby infer its pharmacological activity.

The molecular shape analysis method was proposed in 1980 and is a product of the combination of molecular conformation analysis and the Hansch approach QSAR. This method holds that the pharmacologically active conformation of a drug molecule is the key to determining its activity. Flexible molecules can have multiple conformations and thus different shapes. By comparing the shapes of drug molecules with the same mechanism of action and using data such as the overlapping volume between molecules as structural parameters for statistical analysis, a structure–activity relationship model is obtained.

The virtual receptor method is an extension and development of distance geometry and CoMFA methods. Its basic idea is to establish a virtual receptor environment around the drug molecule by using multiple probe particles, thereby studying the correlation between the activity and structure of different drug molecules. Compared with the CoMFA method, its principle is more reasonable and it is one of the hotspots in quantitative structure–activity relationship research.

13.3 Steps for QSAR Model Construction

This section will take the commonly used SYBYL molecular modeling software as the practical tool to systematically dissect the construction process of 3D-QSAR models. Starting from dataset design and molecular energy optimization, the critical molecular alignment operation, the generation of molecular tables and import of biological activity data, to the parameter setting of CoMFA/CoMSIA, the implementation of Partial Least Squares (PLS) analysis, and further to the generation of contour maps and prediction of biological activity values, we will elaborate on the key operational points, logic of parameter configuration, and precautions for each step one by one, providing readers with practical and actionable guidelines for model construction.

13.3.1 Software Introduction

At present, there are many software programs available for calculating relevant parameters or descriptors, such as TOPIX, Hyper Chem, Discovery Studio, and SYBYL. This article takes SYBYL software as an example to introduce the construction steps of 3D-QSAR models.

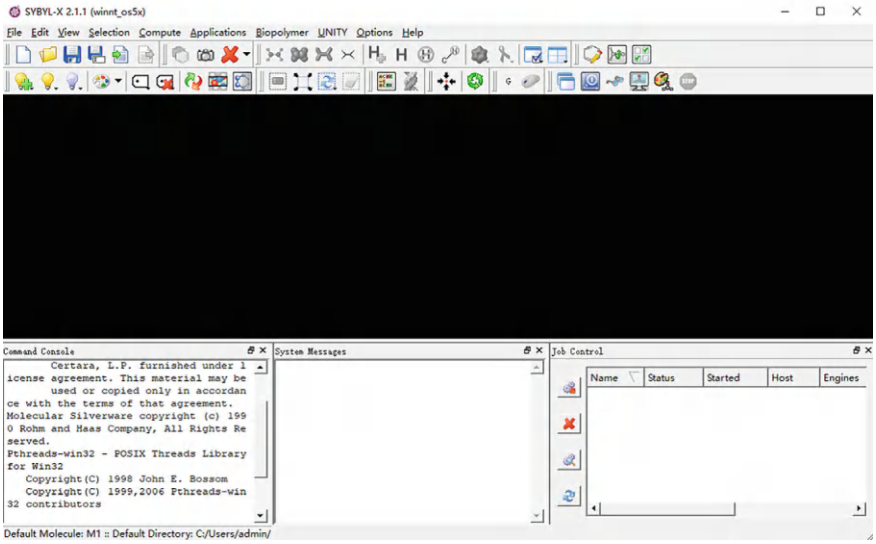


Fig. 13.1 Interface of SYBYL software

SYBYL is an integrated molecular simulation software developed by Tripos Inc. of the United States for the fields of small molecule drugs and biological macromolecules. It is widely used in the research and development of drugs and biological molecules. The functions it provides mainly include molecular drawing, molecular docking, complex conformation analysis, and QSAR, which can assist researchers in conducting comprehensive compound design. SYBYL is a commonly used homology modeling software, covering all aspects of drug design, including molecular docking, virtual screening, quantitative structure–activity relationship, pharmacophore modeling, combinatorial library design and optimization, and compound management, among many others (Fig. 13.1).

13.3.2 3D-QSAR Operating Procedures

At present, among quantitative structure–activity relationship (QSAR) methods, three-dimensional QSAR is the most widely used. Among them, CoMFA and CoMSIA are particularly prominent It is a major research method of 3D-QSAR, implemented as a module in large-scale molecular simulation software such as SYBYL and Discovery Studio, and is widely applied in drug design, toxicological testing, and other fields. Taking the relevant operations of CoMFA as an example, the specific steps are as follows.

(1) Dataset design. Biological activity data are obtained through database search, literature collection and experimental determination, and then preprocessed.

The biological activity data are preferably obtained under the same or similar conditions. For the data of the same compound from different sources, the average value can be taken, while those with significant differences should be excluded. To ensure the predictive ability of the model, the CoMFA study needs to be divided into a training set and a test set. Usually, the two datasets can be plotted in SYBYL software or other software such as Chem Draw. It is worth noting that in SYBYL software, the default path needs to be set at the very beginning. Finally, to ensure the reliability of the QSAR analysis results, energy optimization of the molecules in the dataset is required.

In the menu bar of SYBYL, select "Compute", and select the "Molecule" option of the first item "Minimize", and set the required parameters through the pop-up dialog box. dialog box to set the required parameters. In the Minimize dialog box that pops up, set the required parameters as shown in Fig. 13.2. Generally, energy minimization is performed through the Tripos force field, specifying Gasteiger-Hückel charges, and setting the values of Max Iterations and Gradient. The final conformation of the molecule is then saved for model construction. Energy minimization can also be carried out uniformly after generating the molecular table.

(2) Manual molecular superposition. The quality of molecular superposition is a very important factor affecting the quality of the model. Generally, superposition is carried out by aligning the key atoms or functional groups of the molecules. Superposition can be divided into two methods: backbone superposition and field superposition. The rules for backbone superposition are as follows: On the premise of understanding the mechanism of action, the structure of the remaining molecules can be constructed using the known active conformation as a template, and then locally optimized and superimposed with the known active conformation; when the active conformation is unknown, the low-energy conformation of the most active molecule can be used as a template to construct and locally optimize the structures of other molecules, and then superimpose them with the template molecule; or the active an alog method can be used to conduct a systematic conformational search of the molecules to find their common conformation and thereby determine the active conformation for superposition. At the same time, when choosing a common backbone, it should be noted that the backbone must be shared by all compounds in the database, otherwise the molecules cannot be automatically superimposed; the selected backbone structure should not contain hydrogen atoms. The specific operation is as follows.

Click on File → Database → Align Database in the menu bar. In the Align Database dialog box that pops up (as shown in Fig. 13.3), select the database file to be aligned in the Database to Align field. Then, choose the most active compound in the database as the template molecule in the Template Molecule field. Enter the common substructure for alignment in the Common Substructure field. There are two ways to select the common substructure: one is to check it in the Atom Expression dialog box that pops up; the other is to select it on the template molecule by holding

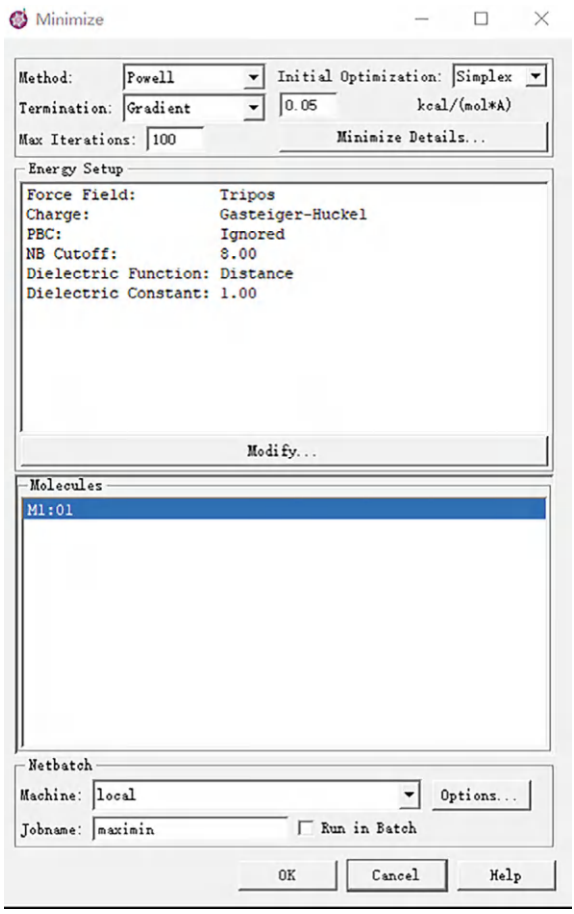


Fig. 13.2 Parameter settings for molecular energy minimization

down the left mouse button and the Shift key. After the selection is completed, the operation interface will retain the common substructure part, and the unselected part will be hidden.

Keep the following three items as default: select "Inertial" in the "Grid Orientation" column, "New Database" in the "Put Into" column, and "All Molecules" in the "Align" column. Then click the "Apply" button to start the molecular superposition. If molecules without common structures are encountered during the superposition process, a dialog box will pop up for prompt. After confirmation, the system will continue the superposition. Once the superposition is completed, enter the name of the new database in the popped-up dialog box and click the "OK" button.

It is worth noting that in actual QSAR studies, a test set is needed to verify the predictive ability of the model. The molecules in the test set also need to be superimposed, and the template molecule should be the same as the one used for superimposing the training set. Meanwhile, to avoid repeating this step, you can

Fig. 13.3 Selection of superimposition parameters

click on the menu bar's View → Transformation, and then click the Freeze All button to freeze the conformation of the template molecule.

(3) Generation of Molecular Tables. When selecting the database file for generating the table in this step, the newly generated database file after molecular superposition should be chosen, such as the db1.mdb file (see Fig. 13.4).

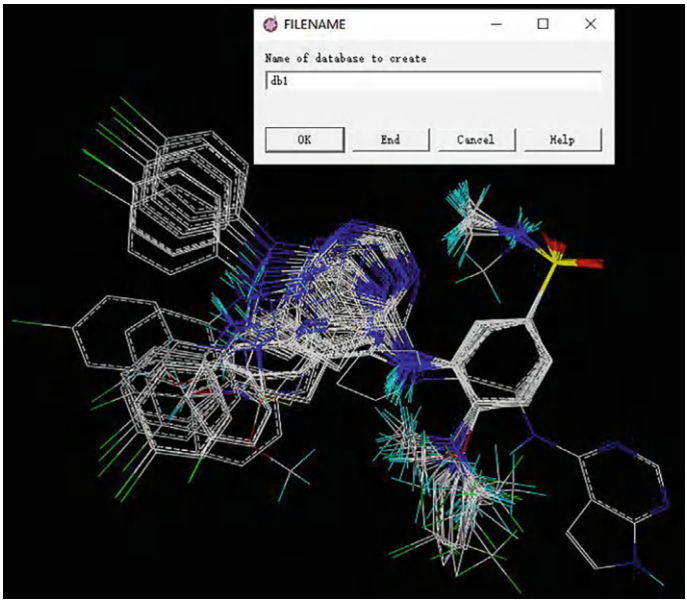


Fig. 13.4 Molecular superposition results

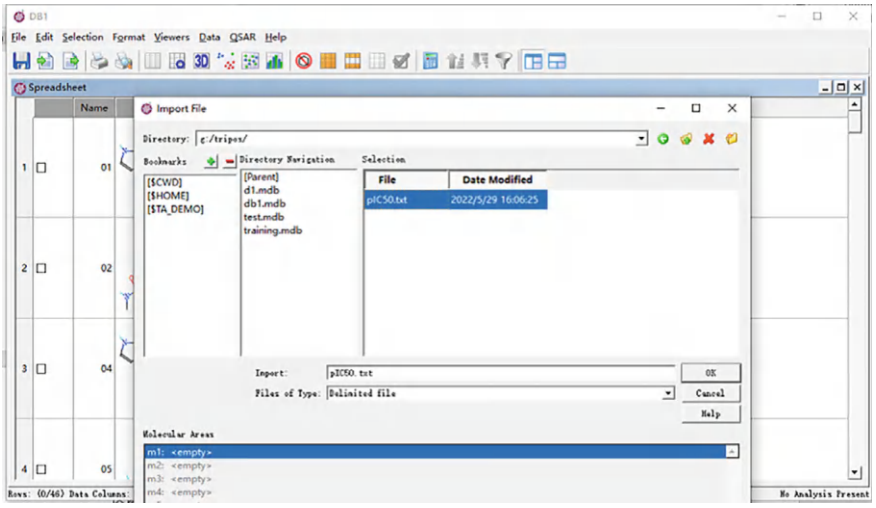


Fig. 13.5 Selection of the bioactive parameter file

The newly generated table does not contain bioactivity data and needs to be added manually. The data to be added should be in the form of negative logarithms, such as pIC50. In the new table db1.mdb, click on File → Import into spreadsheet in the menu bar. In the pop-up dialog box, select the text document containing the bioactivity data (*.txt), and choose Delimited file format for the file type. Then, click the OK button, as shown in Fig. 13.5. It is essential to note that the format in the text document must be consistent.

(4) Parameter Settings. The SYBYL software has an inbuilt QSAR module. After importing the biological activity parameters, simply select CoMFA or CoMSIA for calculation. Click the "Calculate Properties" button in the menu bar, and then choose the CoMFA option under QSAR from the pop-up dialog box. If you are establishing CoMSIA, select the CoMSIA option instead, as shown in Fig. 13.6. Other options can be modified as needed. Click the "Advanced Details" button for further settings, as illustrated in Fig. 13.7. After completing the relevant parameter settings, click the "Calculate" button to finish the calculation.

(5) Partial least squares analysis. Partial least squares analysis can determine the best function match for a set of data by minimizing the sum of the squared errors. Firstly, cross-validation is conducted using the leave-one-out (LOO) method to obtain the optimal number of components (ONC) and the coefficient of determination r^2. Then, based on the determined ONC, a 3D-QSAR model is established without cross-validation to obtain the correlation coefficient r^2, root mean square error (RMSE), and Fisher test value F. The above parameters obtained through partial least squares analysis can be used to evaluate the predictive ability and stability of the model and predict the activity of each molecule in the test set.

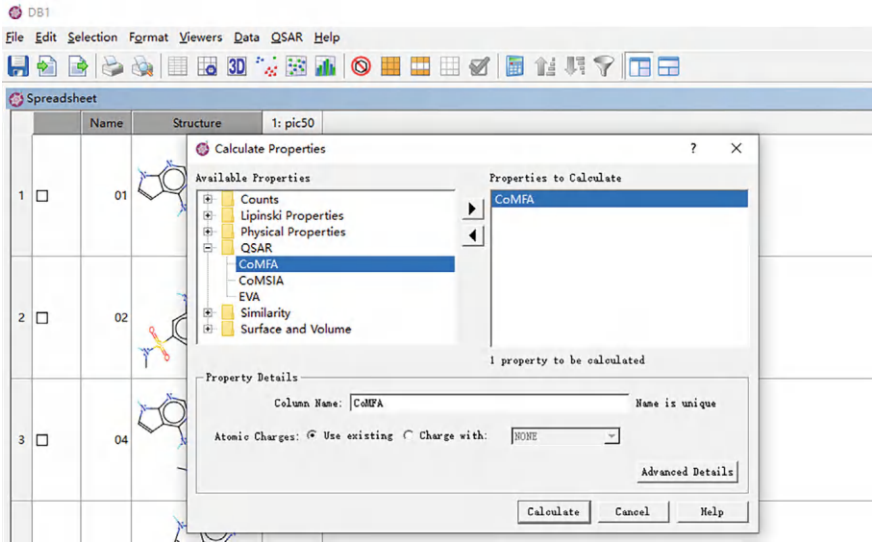


Fig. 13.6 CoMFA/CoMSIA calculation selection

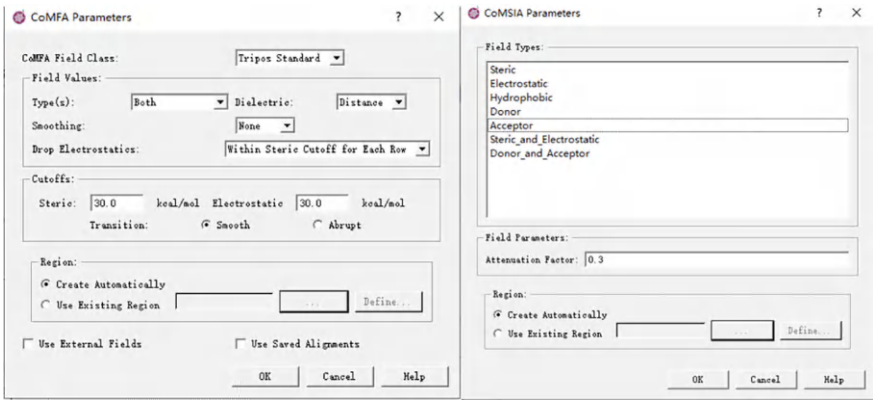


Fig. 13.7 Parameter settings for CoMFA/CoMSIA

Select the bioactivity data column (pIC50) and the CoMFA column. Then, click on QSAR → Partial Least Squares in the menu bar. In the dialog box shown in Fig. 13.8 that pops up, select the Leave-One-Out radio button in the Validation column; check the Column Filtering checkbox and set its value to 2.0; in the Components column, select the Components radio button, and its default value is 6, which can be adjusted as needed; in the Scaling dropdown list, select CoMFA Standard; uncheck the Use SAMPLS checkbox. After completing the settings, click the "Do PLS" button to

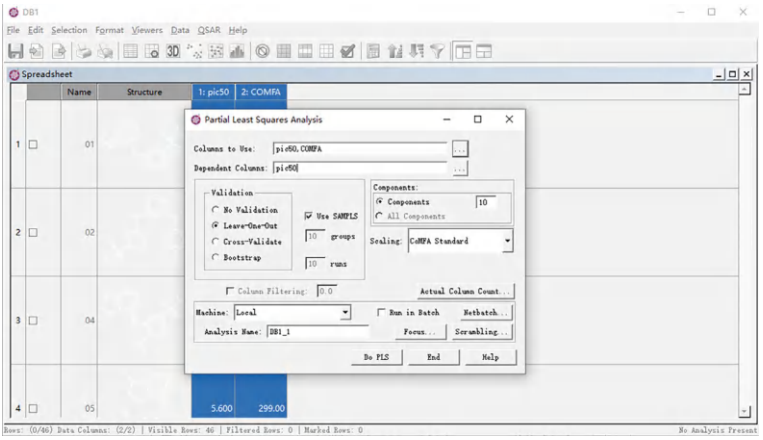


Fig. 13.8 Parameter settings for cross-validation in CoMFA

```
Crossvalidated R2 for 10 components:
 0.356  0.511  0.354  0.520  0.690  0.675  0.664  0.665  0.629  0.620
-- optimum is  0.690 at 5 components

-------------------------------------------------
```

Fig. 13.9 CoMFA cross-validation results

perform cross-validation. Once the validation is complete, the results of the cross-validation can then be seen in the "Command Console" window at the lower left corner, as shown in Fig. 13.9. The results of the cross-validation are the optimal number of principal components and the cross-validation coefficient.

Return to the Partial Least Squares Analysis dialog box, and then perform a non-cross-validation based on this result. Change the Components value to the number of principal components obtained from the cross-validation. In the Validation column, select the No Validation radio button and uncheck the Column Filtering checkbox, as shown in Fig. 13.10. Click the Do PLS button to conduct the regular regression analysis and save the analysis results in the *.pls format. After completing the above two steps, you can view the results of the CoMFA regular regression analysis in the command window, as shown in Fig. 13.11.

In the Partial Least Squares Analysis dialog box, click the End button to close the dialog box and complete the model establishment. The operation of CoMSIA is similar to that of CoMFA.

(6) Generation of equipotential maps. Click on QSAR → View QSAR → CoMFA in the menu bar, as shown in Fig. 13.12. A View CoMFA dialog box will pop up. If CoMSIA is selected, a View CoMSIA dialog box will appear, as shown in Fig. 13.13. The similarities between the two are as follows: In the Type of Field to Display section of the dialog box, check the PLS Analysis checkbox, and

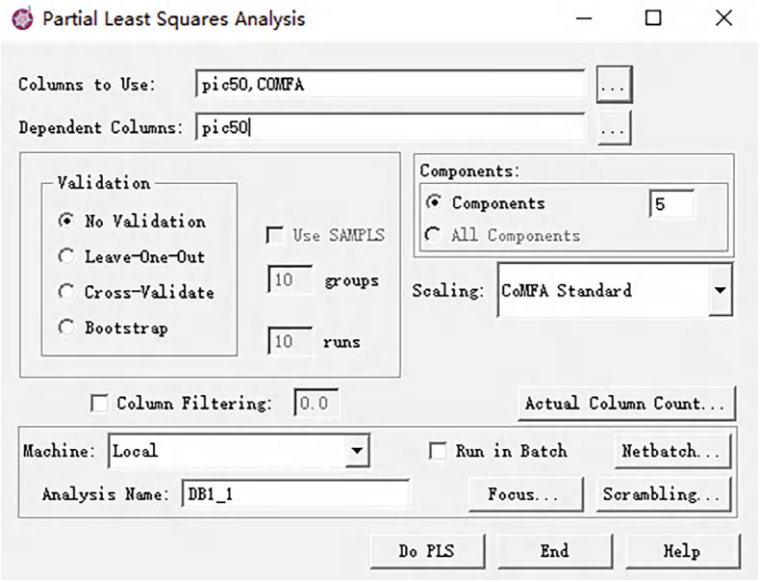


Fig. 13.10 Parameter settings for conventional regression analysis of CoMFA

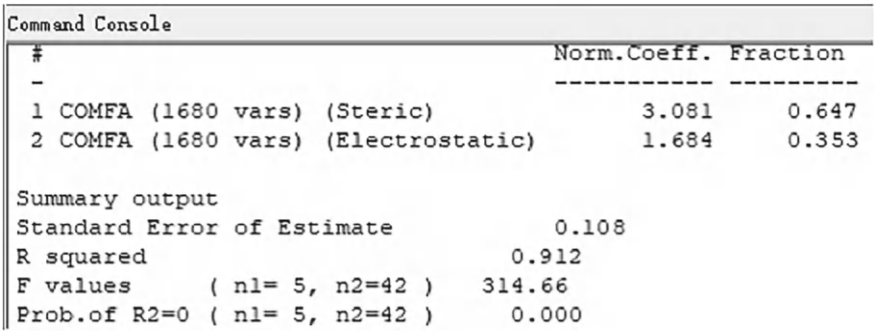


Fig. 13.11 Results of the conventional regression analysis of CoMFA

then select the file saved from the regular regression analysis. In the Contour Specifications section, select Contribution from the Contour by dropdown list and Solid from the Display as dropdown list. In the absence of special requirements, the favored cutoff value for each molecular field is set to 80.0, and the disfavored cutoff value is set to 20.0. The difference lies in the molecular fields. In the CoMFA model, there are only two molecular fields, namely Sterics and Electrostatics, and the color blocks representing each molecular field cannot be changed. In the CoMSIA model, in addition to the two molecular fields in the CoMFA model, there are three more molecular fields: hydrophobic field,

hydrogen bond donor field, and hydrogen bond acceptor field. However, the colors representing the molecular fields in this model can be changed, as shown in Fig. 13.14. According to the drawing requirements, check the corresponding molecular fields, and they will be displayed on the operation interface. You can draw them separately or combine them. The CoMFA equipotential map is shown in Fig. 13.15.

(7) Prediction of biological activity values. Use the constructed 3D-QSAR model to predict the biological activity of the training set compounds. Right-click an empty column in the spreadsheet, select Add A Computed Column from the pop-up menu, choose PREDIC in the dialog box shown in Fig. 13.16, and click the OK button; select the spreadsheet DB1 for prediction, and click the OK button.

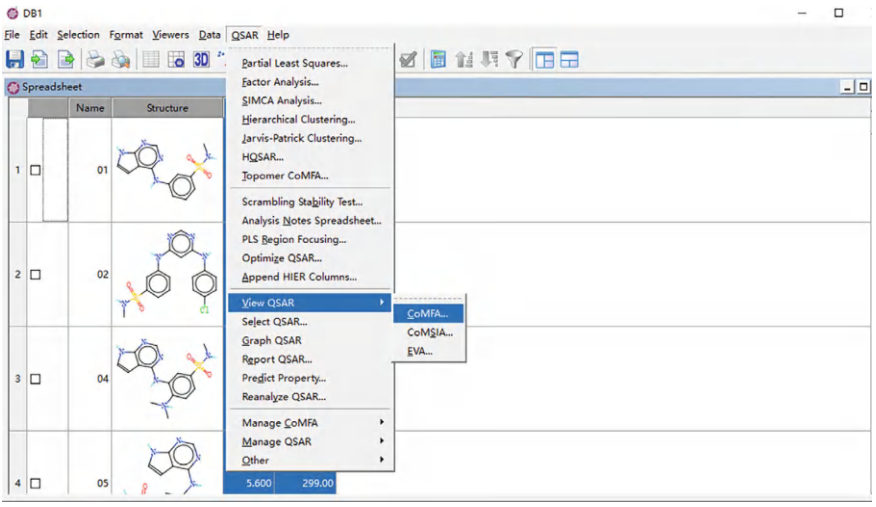


Fig. 13.12 Three-dimensional equipotential surface menu

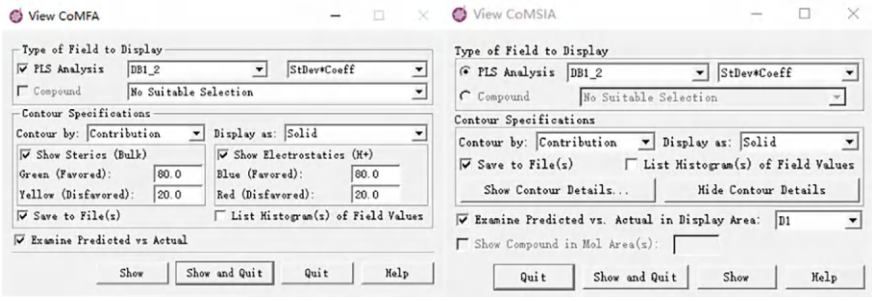


Fig. 13.13 Parameter settings for CoMFA/CoMSIA contour maps

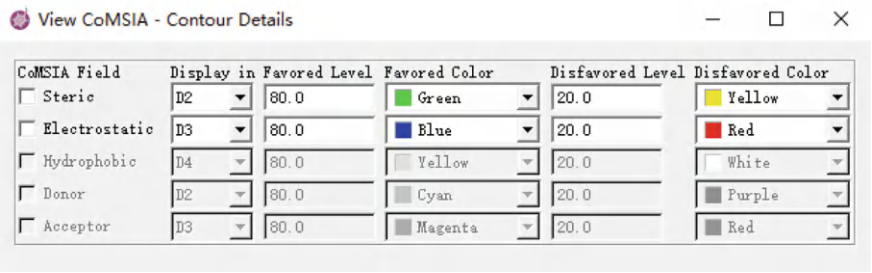


Fig. 13.14 Color settings for CoMSIA potential maps

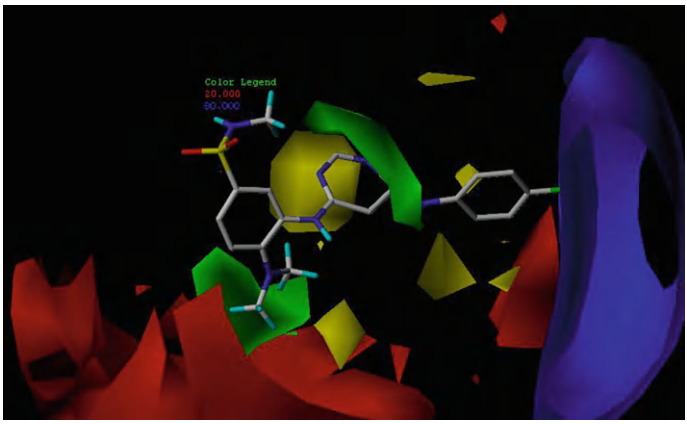


Fig. 13.15 CoMFA equipotential map

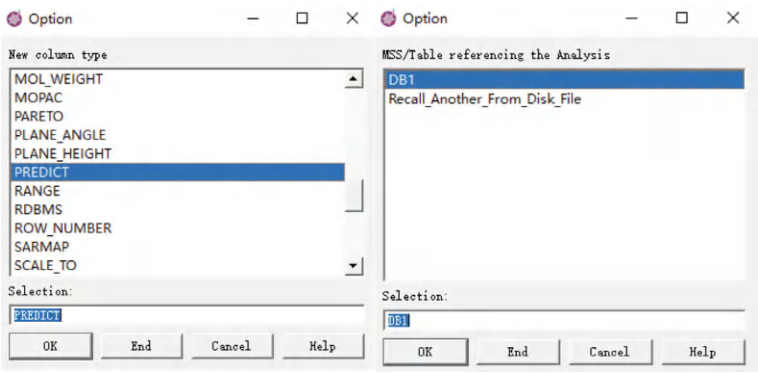


Fig. 13.16 Settings before predicting the bioactivity value

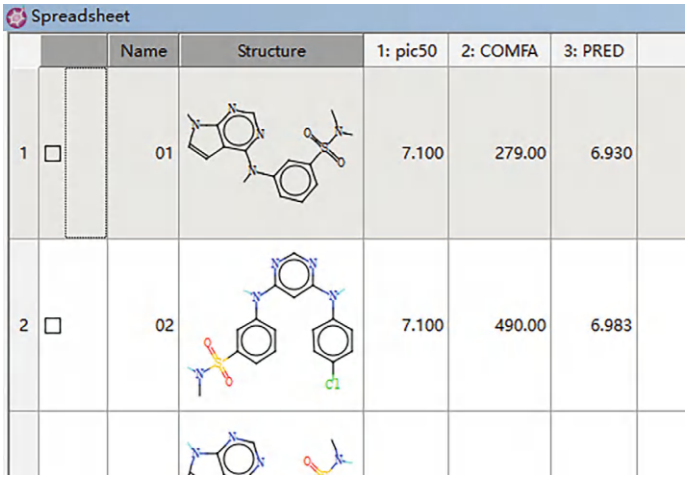


Fig. 13.17 CoMFA predicted bioactivity results

Select the model used to calculate the predicted values, click the OK button, and the system will automatically calculate and fill the prediction results into the spreadsheet to be predicted. The CoMFA biological activity prediction results are shown in Fig. 13.17. When predicting the test set, the spreadsheet containing the training set information must be opened; otherwise, the prediction cannot be performed. After obtaining the predicted values, they can be imported into the graphing software along with the biological activity values to draw the correlation graph of predicted activity and experimental values.

13.4 QSAR in the Context of Machine Learning

Drug discovery aims to identify new compounds with specific chemical properties for treating diseases. With the goals set and new challenges arising from the Precision Medicine Initiative, it is necessary to establish robust, standardized, and reproducible computational methods to achieve the set objectives [14]. Traditional QSAR methods are not suitable for big data due to their characteristics, such as large data volume, rapid data growth, diverse sources, and high data uncertainty. Compared with machine learning, deep learning methods that explain the vanishing gradient effect are more suitable for raw high-dimensional data. Therefore, as a result of data-driven and computing power-driven research, machine intelligence in drug discovery. It has developed to a new position. Currently, machine learning-based predictive models play an important role in preclinical research. Since the beginning of the twenty-first century, previously established ML methods, such as Support Vector Machines (SVM), Artificial Neural Networks (ANN) and Random Forests

(RF), have been used to develop QSAR models to predict biological properties. With the development of neural network technology, convolutional neural networks and others have also gradually been applied to drug design.

13.4.1 Common Machine Learning Methods

Support vector machines (SVMs) are one of the most widely used models in bioinformatics because they can handle complex, nonlinear, high-dimensional and noisy problems. For linear problems, the SVM model distinguishes different categories by mapping points in space to maximize the difference between points of different categories. For nonlinear problems, SVMs use kernel mapping to transform nonlinear datasets into high-dimensional feature spaces to achieve linear classification.

In the past few years, the number of studies on drug discovery using support vector machines has significantly increased, especially those related to QSAR and virtual screening. As one of the methods for constructing QSAR models, support vector machines can be combined with various machine learning techniques, such as elastic net, random forest, neural networks, MLR, partial least squares, k-nearest neighbors, etc. [15]. For instance, Ai et al. [16] established a QSAR model for over 450 compounds regarding LC_{50} values using BCF data, in which they employed the recursive feature elimination method combined with three machine learning algorithms: random forest, support vector machine, and XGBoost. The model indicated that aromatic structures, hydrogen bond groups, and partition coefficients are relatively important influencing factors.

Decision trees are a transparent and interpretable machine learning method. Although there are implementations for different types of problems (such as regression, survival or outlier detection), the methods used for classification problems are particularly prominent. In the hierarchical structure of a decision tree, root nodes, internal nodes or terminal nodes can be identified. The root node is usually located at the top of the tree, with no branches leading to it, and one or more branches starting from it. For internal nodes, there is one branch leading to them and two or more branches starting from them to the next level of the hierarchy. Finally, terminal nodes have no branches starting from them because they are at the last level of the hierarchy. Decision trees are used to simulate the absorption, distribution and metabolic characteristics of drugs and their toxicity [17]. For example, to assess the toxicity of volatile organic compounds, Gupta et al. [18] used the DT-forest and DT-boost algorithms to simulate the sensory irritation potency of volatile organic compounds. The former combines decision trees with bagging technology, while the latter combines decision trees with gradient boosting algorithms. Both models have improvements over standard decision trees.

Random forest is an ensemble modeling method and one of the widely used algorithms in machine learning. It operates by constructing multiple decision trees as base learners. Compared with decision trees, random forests can prevent overfitting of data. Random forest has been widely applied in bioactivity classification, toxicity

modeling, prediction of protein–ligand binding affinity, and drug target identification, etc. [19]. For instance, Mistry et al. were the first to model the drug-carrier toxicity relationship using random forest and decision trees. Their dataset included 227,093 potential candidate drugs and 39 potential carriers, and the resulting model predicted the mitigation of drug toxicity by specific carriers.

K-Nearest Neighbors (KNN) is an unsupervised algorithm used for classification and regression. In most cases, KNN is employed for classification by calculating the categories of the k nearest neighbors in the feature space. Therefore, the KNN algorithm is one of the simplest and easiest-to-implement algorithms among all machine learning algorithms and is often integrated with other feature selection algorithms. To identify antiviral drugs, Weidlich et al. [20] combined KNN with simulated annealing and random forest to study 679 drug molecules. Their results indicated that this improved KNN model outperformed the random forest model. Mansouri et al. established a QSAR model to predict biodegradable chemicals using different modeling methods and various types of molecular descriptors. Different classification models (such as KNN, Partial Least Squares Discriminant Analysis, Support Vector Machine, etc.) were applied to classify biodegradable and non-biodegradable chemicals, and it was found that KNN could perform better. Olotu et al. [21] developed a QSAR model based on FDA approved inhibitors using appropriate molecular descriptors, and employed random forest, K-means, decision trees, linear regression, hierarchical clustering, and Bayesian classifiers to explain the structural requirements for HIV-1 inhibitory activity.

13.4.2 Deep Learning Methods

Although machine learning methods have advantages and popularity in drug design, in recent years, in some cases, machine learning has been replaced by deep learning. Deep learning can not only handle big data but also does not require manual feature extraction; it can automatically learn features. This process is usually achieved by stacking multiple layers of simple neural networks with nonlinear input–output mappings. These mainly include deep neural networks (DNN), convolutional neural networks (CNN), recurrent neural networks (RNN), and other deep networks, which have been applied in chemoinformatics and bioinformatics to handle various tasks, such as water solubility prediction, quantitative structure–activity relationship analysis, and predicting the sequence specificity of DNA/RNA-binding proteins, etc. [22].

Among the quantitative structure–activity relationship (QSAR) molecules, the application of deep neural networks is the most widespread. For instance, Ma et al. [23] utilized 15 different QSAR datasets and demonstrated through experiments that in most cases, DNNs could make better predictions than random forests. They also proved that for most datasets, there could be a set of well-performing tunable parameters, and it was not necessary to optimize the parameters for each dataset separately. This makes DNN a practical method for QSAR in drug design. Additionally,

Karpov et al. [24] proposed a Transformer-CNN model that normalizes SMILES into a Seq2Seq problem. Based on SMILES embeddings, the CharNN architecture was used, generating higher-quality interpretable QSAR/QSPR models on various benchmark datasets for regression and classification tasks.

Although deep learning models have high prediction accuracy, the intermediate running process is like a "black box", making it difficult to specifically reveal the biological mechanisms integrated in the modeling data. In general, as a newly developed machine intelligence technology, deep learning has shown potential applications in the new era of big data in drug discovery. With more and more data becoming available and new methods being developed, deep learning methods will become the main computer-aided drug design methods in the near future.

References

1. Verma, J., Khedkar, V.M., Coutinho, E.C.: 3D-QSAR in drug design-a review. Curr. Top. Med. Chem. **10**, 95–115 (2010)
2. Missner, A., Pohl, P.: 110 years of the Meyer-Overton rule: predicting membrane permeability of gases and other small compounds. ChemPhysChem **10**, 1405–1414 (2009)
3. Debnath, A.: Quantitative structure-activity relationship (QSAR) paradigm-Hansch era to new millennium. Mini Rev. Med. Chem. **1**, 187–195 (2001)
4. Free, S.M., Wilson, J.W.: A mathematical contribution to structure-activity studies. J. Med. Chem. **7**, 395–399 (1964)
5. Crippen, G.M.: Distance geometry approach to rationalizing binding data. J. Med. Chem. **22**, 988–997 (1979)
6. Cramer, R.D., Patterson, D.E., Bunce, J.D.: Comparative molecular field analysis (CoMFA). 1. Effect of shape on binding of steroids to carrier proteins. J. Am. Chem. Soc. **110**, 5959–5967 (1988)
7. Klebe, G., Abraham, U., Mietzner, T.: Molecular similarity indices in a comparative analysis (CoMSIA) of drug molecules to correlate and predict their biological activity. J. Med. Chem. **37**, 4130–4146 (1994)
8. Ballante, F., Kooistra, A.J., Kampen, S., de Graaf, C., Carlsson, J.: Structure-based virtual screening for ligands of G protein–coupled receptors: what can molecular docking do for you? Pharmacol. Rev. **73**, 1698–1736 (2021)
9. Romeiro, N.C., Albuquerque, M.G., de Alencastro, R.B., Ravi, M., Hopfinger, A.J.: Construction of 4D-QSAR models for use in the design of novel p38-MAPK inhibitors. J. Comput. Aided Mol. Des. **19**, 385–400 (2005)
10. Vedani, A., Dobler, M.: 5D-QSAR: the key for simulating induced fit? J. Med. Chem. **45**, 2139–2149 (2002)
11. Damale, G., Harke, M.N., Khan, S.A.K., Shinde, F.B., Sangshetti, N.: Recent advances in multidimensional QSAR (4D–6D): a critical review. Mini Rev. Med. Chem. **14**, 35–55 (2014)
12. Randic, M.: Characterization of molecular branching. J. Am. Chem. Soc. **97**, 6609–6615 (1975)
13. Yang, L., et al.: Data driven toxicity assessment of organic chemicals against Gammarus species using QSAR approach. Chemosphere **328**, 138433 (2023)
14. Carracedo-Reboredo, P., et al.: A review on machine learning approaches and trends in drug discovery. Comput. Struct. Biotechnol. J. **19**, 4538–4558 (2021)
15. Maltarollo, V.G., Kronenberger, T., Espinoza, G.Z., Oliveira, P.R., Honorio, K.M.: Advances with support vector machines for novel drug discovery. Expert Opin. Drug Discov. **14**, 23–33 (2019)

16. Ai, H., et al.: QSAR modelling study of the bioconcentration factor and toxicity of organic compounds to aquatic organisms using machine learning and ensemble methods. Ecotoxicol. Environ. Saf. **179**, 71–78 (2019)
17. Newby, D., Freitas, A.A., Ghafourian, T.: Decision trees to characterise the roles of permeability and solubility on the prediction of oral absorption. Eur. J. Med. Chem. **90**, 751–765 (2015)
18. Gupta, S., Basant, N., Singh, K.P.: Estimating sensory irritation potency of volatile organic chemicals using QSARs based on decision tree methods for regulatory purpose. Ecotoxicology **24**, 873–886 (2015)
19. Singh, H., Singh, S., Singla, D., Agarwal, S.M., Raghava, G.P.S.: QSAR based model for discriminating EGFR inhibitors and non-inhibitors using Random forest. Biol. Direct **10**, 10 (2015)
20. Weidlich, I.E., et al.: Inhibitors for the hepatitis C virus RNA polymerase explored by SAR with advanced machine learning methods. Bioorg. Med. Chem. **21**, 3127–3137 (2013)
21. Olotu, F.A., Agoni, C., Soremekun, O., Soliman, M.E.S.: The recent application of 3D-QSAR and docking studies to novel HIV-protease inhibitor drug discovery. Expert Opin. Drug Discov. **15**, 1095–1109 (2020)
22. Tropsha, A., Isayev, O., Varnek, A., Schneider, G., Cherkasov, A.: Integrating QSAR modelling and deep learning in drug discovery: the emergence of deep QSAR. Nat. Rev. Drug Discovery **23**, 141–155 (2024)
23. Ma, J., Sheridan, R.P., Liaw, A., Dahl, G.E., Svetnik, V.: Deep neural nets as a method for quantitative structure–activity relationships. J. Chem. Inf. Model. **55**, 263–274 (2015)
24. Karpov, P., Godin, G., Tetko, I.V.: Transformer-CNN: Swiss knife for QSAR modeling and interpretation. J. Cheminform. **12**, 17 (2020)

Chapter 14
Molecular Representations

This chapter focuses on molecular representations and molecular feature engineering, a foundational discipline in computer-aided drug discovery and chemoinformatics. It systematically presents the core concepts, methodologies, and applications of drug molecular representation. First, it establishes the structural foundation, covering molecular composition (scaffolds, linkers, side chains), chemical bond types (covalent/non-covalent bonds), 3D conformations, and molecular chirality—highlighting chirality's critical impact on drug efficacy and safety. Subsequently, it introduces molecular descriptors as mathematical quantifications of physicochemical properties, detailing their classification, encoding rules of Simplified Molecular Input Line Entry System (SMILES) and SMILES Arbitrary Target Specification (SMARTS) strings, and selection criteria for high-quality descriptors. It then delves into molecular fingerprints (Molecular ACCess System (MACCS keys), Extended-Connectivity Fingerprints (ECFP), and Functional-Class Fingerprints (FCFP)), expounding their principles, generation processes, and applications in molecular similarity analysis with code examples. Finally, it outlines drug molecular feature engineering, distinguishing manual and deep learning-acquired representations, presenting specialized methods (grid features, graph convolution), and systematically explaining feature selection strategies (supervised/unsupervised/semi-supervised, filter/wrapper/embedded methods). This chapter constructs a complete theoretical and methodological framework, providing essential technical support for computational modeling, virtual screening, and predictive analysis in drug development.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_14

14.1 Drug Molecular Structure

This section focuses on the core structural basis of drug molecules, sequentially elaborating on the essential definition of molecules, the compositional division of "backbone-side chain", the types of chemical bonds (covalent bonds and non-covalent bonds) linking atoms, the conformational characteristics of molecules in three-dimensional space, and the molecular chirality with enantiomeric properties. Meanwhile, typical cases are integrated to illustrate the critical impact of chirality on drug efficacy, laying a foundation for understanding the correlation between molecular structure and function.

14.1.1 What Is a Molecule?

Molecules are formed by the combination of multiple atoms in a specific arrangement through the action of electromagnetic forces. They are the smallest basic units in compounds that can participate in chemical reactions and also the fundamental units that maintain physical properties. Atoms within a molecule are bonded together through chemical bonds. Interconnected by chemical bonds, which restrict their movement relative to each other, molecules have a wide range of sizes, from a few atoms to thousands of atoms.

The structures of drug molecules are diverse, but they share some common features. We can divide a molecule into a scaffold and side chains. The scaffold can be further divided into ring systems and linkers. Ring systems are one or more complete rings formed without breaking any chemical bonds, and they can share a chemical bond or an atom. Linkers refer to the atoms or chemical bonds that connect two different ring systems. Side chains are the atoms and chemical bonds that do not belong to the ring systems or linkers. Note that non-cyclic molecules do not have a scaffold [1]. The structure of a drug molecule is shown in Fig. 14.1.

14.1.2 What Are Molecular Bonds?

A molecule is composed of several atoms, which are connected together by chemical bonds. Essentially, chemical bonds link atoms through the electrons in the peripheral region of the atoms. The structure of molecules is diverse, so the types of molecular bonds are also diverse, which can be divided into covalent bonds and non-covalent bonds [2].

Elements with very high ionization energies cannot transfer electrons, while those with very low electron affinities cannot absorb electrons. The atoms of these elements tend to share their electrons with the atoms of other elements or with other atoms of the same element to achieve stability. This association through the sharing of electron

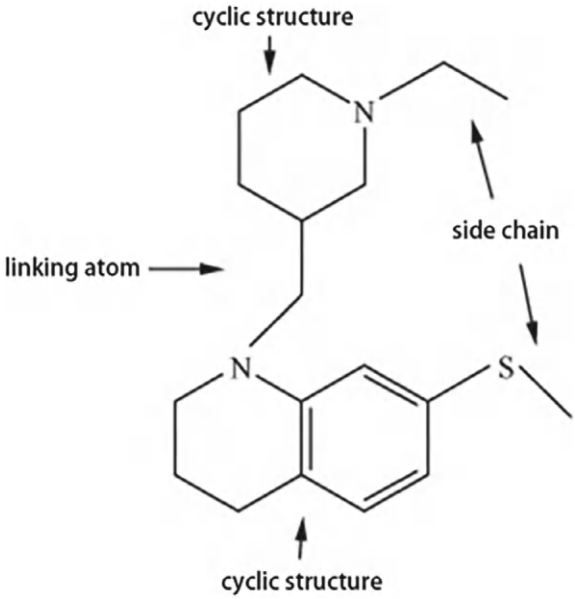


Fig. 14.1 Structure of the drug molecule

pairs between different or the same kinds is called a covalent bond. Covalent bonds are highly stable and do not form or break easily.

Covalent bonds can be formed in two ways: by sharing electrons between atoms of the same kind or between atoms of different kinds. For instance, molecules like H_2 and O_2 involve the sharing of electrons between atoms of the same kind, while molecules such as CH_4, H_2O, and NH_3 involve the sharing of electrons between atoms of different kinds [3].

Covalent bonds are classified into single bonds, double bonds and triple bonds based on the number of shared electron pairs. A single bond is formed when there is only one pair of electrons shared between atoms. It has the lowest density but is the most stable. Double bonds and triple bonds are formed when there are two and three pairs of electrons shared between atoms respectively. The triple bond is the least stable covalent bond.

Non-covalent bonds are exactly the opposite of covalent bonds. They do not involve the direct sharing of electrons between atoms. Non-covalent bonds are also not as strong as covalent bonds and can constantly break and re-form. The three main types of non-covalent bonds are ionic interactions, hydrophobic interactions, and hydrogen bonds. Compared with covalent bonds, the interactions of non-covalent bonds are relatively weak, but a large number of non-covalent bonds can fix the structure.

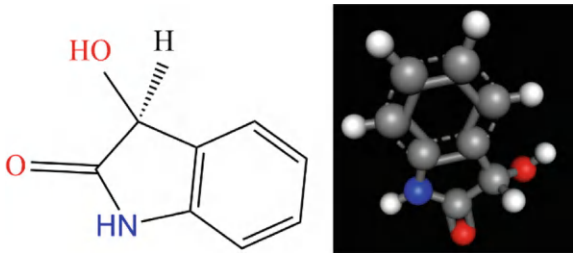


Fig. 14.2 Ligand molecules are represented in both two-dimensional molecular graphs and three-dimensional conformations

14.1.3 What Is Molecular Conformation?

Prior to delving into molecular conformations, it is necessary to first clarify the definition of a molecular graph. A molecular graph is a topological structure constituted by interconnected nodes and edges. Chemical molecules can be represented in the form of graphs: in a molecular graph, atoms serve as nodes, while chemical bonds correspond to edges. Notably, any chemical molecule can be universally represented by a corresponding molecular graph.

A molecular graph describes what components a molecule is composed of and how they are connected. However, with the advancement of scientific research and the demands of technology, we are no longer satisfied with the two-dimensional representation of molecular graphs. We are more interested in knowing how atoms and chemical bonds are positioned and connected in three-dimensional space. This three-dimensional representation in the form of a molecular graph is termed a molecular conformation [4].

Molecular conformation refers to any spatial arrangement of atoms within a molecule that can be interconverted by rotation around single bonds. Any molecule may have different conformations, with at least one atom in each molecule not lying on the axis of the relevant single bond, where a single bond connects two polyatomic groups. Biomacromolecules, such as polynucleotides, polypeptides or polysaccharides, can change their conformations in response to environmental variations [5].

Figure 14.2 shows the two-dimensional molecular graph and three-dimensional conformation of a ligand molecule.

Figure 14.3 shows the three-dimensional conformation of a protein macromolecule. As the molecule becomes larger, the number of feasible conformations required will increase significantly.

14.1.4 What Is the Chirality of Molecules?

Molecular chirality refers to the phenomenon where a molecule exists as two enantiomers—mirror-image stereoisomers that are non-superimposable on each other. As

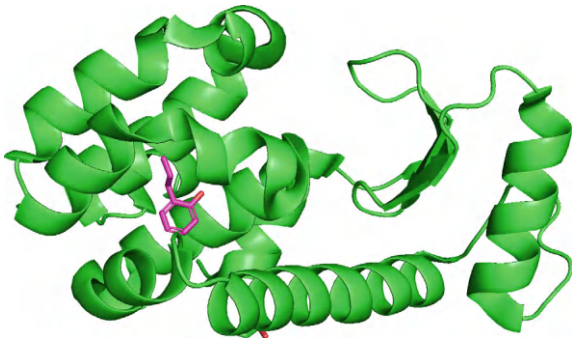


Fig. 14.3 Three-dimensional conformation of proteins. The conformation of proteins is particularly complex, featuring multiple three-dimensional geometric motifs, which also well illustrates that molecules have not only chemical formulas but also high-dimensional geometric structures

the term implies, chirality is manifested in two distinct configurations: the dextrorotatory (D) and levorotatory (L) forms, which are alternatively designated as the (R) and (S) configurations based on the Cahn-Ingold-Prelog (CIP) priority rules. Notably, most naturally occurring amino acids exhibit intrinsic chirality, with glycine being the sole exception due to its achiral α-carbon. Figure 14.4 shows the chiral diagram of helical compounds.

In organic chemistry, chiral compounds are identical in their compositional makeup, and their spatial structures are non-superimposable mirror images. They also share many physicochemical properties. As a result, it is often difficult to distinguish between the chiral versions (R or S enantiomers) of a given molecule in experiments, making the manufacturing process highly complex to obtain a single "R" or "S" form. This challenge extends to computational models as well. Since the two chiral forms are identical in composition and structure, machine learning models relying on molecular graphs as training data also struggle to differentiate between the two chiral compounds.

Most drugs in current use are chiral drugs, and the majority of pharmaceutical molecules exhibit chirality. Different chiral configurations of a drug may bind to distinct proteins in the body, potentially leading to two entirely divergent outcomes.

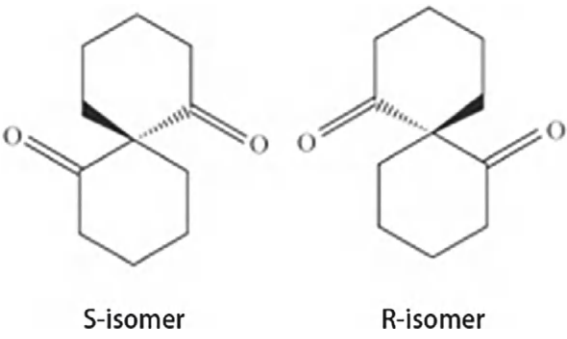


Fig. 14.4 Enantiomers of a chiral molecule (R and S)

Typically, only one enantiomer (R or S configuration) of the drug molecule is therapeutically active, while the other enantiomer lacks therapeutic efficacy. This inactive form may cause side effects or even exhibit toxicity.

The most renowned example of a drug with differing effects due to chiral isomerism is thalidomide, also known as "Contergan." First produced in 1957 by West Germany, the drug was initially marketed as a remedy for morning sickness and as a sedative for pregnant women. However, its release led to over 10,000 cases of neonatal deformities in Europe. Subsequent research confirmed that the "R" enantiomer of thalidomide acts as an effective sedative, while the "S" enantiomer causes teratogenic effects. This tragedy prompted rigorous protocols in pharmaceutical production, mandating the elimination of mirror-image isomers through extensive biological testing and clinical trials before drug approval, thereby preventing similar disasters like the thalidomide incident.

14.2 Molecular Descriptors

This section first clarifies the core concept of molecular descriptors—i.e., the mathematical quantitative representation of the physicochemical properties of molecules. Subsequently, it details various classification methods (by dimension, physical significance, etc.), focusing on the encoding rules and application scenarios of SMILES (Simplified Molecular Input Line Entry System) and SMARTS (SMILES Arbitrary Target Specification) structural query strings. Additionally, the selection criteria for high-quality molecular descriptors are defined, providing methodological support for the accurate featurization of molecular features in subsequent studies.

14.2.1 What Are Molecular Descriptors?

Molecular descriptors can be defined as mathematical representations of molecular properties generated by algorithms. The numerical values of molecular descriptors are used to quantitatively describe the physical and chemical information of molecules. Molecular descriptors are very useful in similarity searches within molecular libraries because they can identify molecules with similar physical or chemical properties based on the similarity of descriptor values. Molecular descriptors can describe simple features of molecules, such as molecular volume; they can also describe complex features, such as multiple physical, chemical, and structural properties of compounds.

14.2.2 Classification of Molecular Descriptors

In the field of computer-aided drug discovery, many applications of chemoinformatics rely on representing molecules by capturing their structural features and properties through descriptors. Meanwhile, computational methods for drug design and molecular diversity often require the symbolization of the abstract structure of molecular structures and properties [6]. For instance, the Schrödinger software suite offers the functionality to calculate molecular descriptors. Through the Maestro module within Schrödinger, one can compute Topological Descriptors, Mopac Descriptors, and QikProp Descriptors.

Molecular descriptors can be categorized in a variety of ways, roughly according to the quantitative and qualitative classification, according to the data type of the descriptor, according to the molecular structure dimension required for the calculation of the descriptor, according to the difference in the physical meaning, according to the experimental and theoretical classification of the five ways of classification [7].

According to the classification of quantitative and qualitative, they can be divided into quantitative descriptors and qualitative descriptors, among which qualitative descriptors are generally called molecular fingerprints. According to the classification of data types of descriptors, they can be divided into boolean values (such as whether chiral), integers (such as the number of rings), real numbers (such as molecular weight), vectors, tensors, etc. According to the classification of physical meaning differences, they can be divided into composition descriptors, molecular property descriptors, topological descriptors, and geometric descriptors. According to the classification of experimental and theoretical, they can be divided into experimental descriptors and theoretical descriptors [8].

Most molecular descriptors are classified according to the required molecular structure dimensions, where "dimension" refers to the molecular representation used to calculate the descriptor values. Zero-dimensional (0-D) descriptors combine all those that do not provide information about the molecular structure or atomic connectivity. They can represent a molecule as simply as a chemical formula. Common 0-D descriptors include the number of atoms, the number of bonds, and the molecular weight. 0-D descriptors are easy to obtain but are often used in combination with other descriptors. One-dimensional (1-D) descriptors can capture a large number of properties, and molecular fingerprints are the most common 1-D descriptors. At the same time, 1-D fingerprints are also easy to obtain. Two-dimensional (2-D) descriptors describe properties that can be calculated from the two-dimensional representation of a molecule, such as its size, shape, and symmetry; while three-dimensional (3-D) descriptors depend on the conformation of the molecule. The most common 3-D descriptors are molecular matrices and 3D-MoRSE descriptors [9]. 3-D descriptors provide a large amount of information about a molecule and have the advantage of distinguishing isomeric molecules. However, they also have a drawback.

A significant drawback is that due to the complexity issue, the calculation can be quite time-consuming. Some "implicit" three-dimensional descriptors representing

surface area or shape indices can be calculated from the two-dimensional representation of molecules or molecular graphs. Four-dimensional (4-D) descriptors are also known as "grid-based descriptors". Four-dimensional descriptors typically characterize the interaction between a molecule and the receptor's active site or multiple conformational states of a molecule. Common four-dimensional descriptors include CoMFA and GRID. A key advantage of four-dimensional (4D) descriptors is that they offer more comprehensive information than conventional lower-dimensional (0D–3D) descriptors, while consistently yielding distinct numerical values for molecules with distinct chemical structures. Similar to three-dimensional descriptors, four-dimensional descriptors are difficult to obtain due to their high complexity.

The requirements for selecting molecular descriptors are as follows:
(1) It has structural interpretability and a good correlation with at least one property
(2) It can distinguish isomers while being distinct from other descriptors
(3) It can be applied to local structures and also change along with structural alterations
(4) It has good independence

14.2.3 SMILES String

String representation is usually composed of characters in the American Standard Code for Information Interchange (ASCII) character encoding standard. Compared with other representation methods, string representation is more compact and easier for humans to read and write.

SMILES character strings are a popular method for specifying molecules in text strings today. SMILES strings describe the atoms and chemical bonds in a molecule in a concise and rather intuitive way. Letters are used to represent atoms, while symbols and numbers are used to encode bond types, connections, branches, and stereochemistry [10].

The representation method of SMILES strings has its own set of rules. Firstly, there are no space characters throughout the entire string, and all characters are continuous. When representing molecules, hydrogen atoms can be omitted. For example, CH4 can be represented as C. Chemical element symbols enclosed in square brackets represent atoms. For instance, [Au] represents a gold atom. Two adjacent atoms in a SMILES string indicate that they are connected. Molecules contain a large number of chemical bonds, and each type of chemical bond has its unique representation in SMILES strings. For example, a single bond is represented by the symbol "−", a

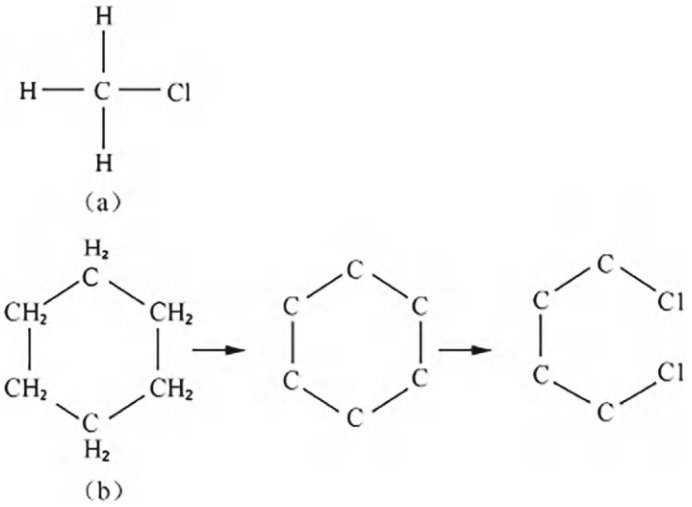


Fig. 14.5 The SMILES string for chloromethane in Figure (**a**) is "CCl"; the SMILES string for cyclohexane in Figure (**b**) is "C1CCCCC1"

double bond by "=", a triple bond by "#", and an aromatic bond by ":". If adjacent atoms are connected by a single bond and an aromatic bond, they can be omitted. For molecules with branched structures, the branched parts are represented within "()", and usually, the main molecular part is on the left side. For cyclic molecules, one chemical bond needs to be broken first, and then represented by a string. The broken bond can be any one, and the two atoms at the break point are marked with the same number. At this point, the structure is no longer a closed loop but an open chain structure, and the chain structure is described according to the above representation method. Compounds with separate structures and no connections are represented by ".". The order of the listed strings is also arbitrary. As shown in Fig. 14.5, the molecular structure of chloromethane is depicted and has a branch. It can be represented by the SMILES string "C–H–H–H–Cl", and after omitting single bonds and hydrogen atoms, it can be written as "CCl". Figure 14.5 shows the cyclic molecular structure of cyclohexane, which can be represented by the SMILES string "C1CCCCC1".

14.2.4 SMARTS Strings

After improving the SMILES system, the SMARTS system was developed. The highlight of the SMARTS system is that it can use wildcards to represent atoms and chemical bonds. SMARTS strings can specify molecules with specific structures when searching and identifying molecules, and superimpose a set of molecules based on their common substructures to improve structural visualization. Compared

with SMILES strings—whose core function is to encode molecular structures in a machine-readable format—the most prominent advantage of SMARTS strings lies in their inherent ability to perform substructure search via pattern matching, rather than merely serving as a linear representation of molecular structures (a limitation of SMILES that requires additional descriptor generation for structural comparison). The atomic semantics in SMARTS strings are shown in Table 14.1.

We can take a look at how SMILES strings define molecules, display molecules, and highlight atoms that match SMARTS patterns from the code example shown in Listing 14.1.

Listing 14.1 SMARTS Pattern Matching Code

```
1.# Featurize the molecule.
  2.from rdkit import Chem.
  3.from rdkit.Chem.Draw import MolsToGridImage.
  4.smiles_list = ["CCCCC","CCOCC","CCNCC","CCSCC"].
  5. mol_list = [Chem.MolFromSmiles(x) for x in
  smiles_list]
  6.# Compare the SMILES string and the SMARTS
string and match the same structure
  7.# Match with "CCC".
  query = Chem.MolFromSmarts("CCC")
  8.match_list = [mol.GetSubstructMatch(query) for
mol in mol_list]
  9.MolsToGridImage(mols = mol_list,
  10.       molsPerRow = 4,
  11.       highlightAtomLists = match_list).
  12,# Use wildcards to match the atomic set.
  13.# Matches the numerator with "C*C" in the
SMARTS string
```

Table 14.1 SMARTS atomic semantics

Symbol	Explain	Atomic properties
*	Wildcard	Any atom
a	Aromatic	Aromatic atom
A	Aliphatic	Aliphatic atom
v <n>	Valence	Atom with total valence <n>
− <n>	Negative charge	Atom with a charge of − <n>
+ <n>	Positive charge	Atom with a charge of + <n>
#n	Atomic number	Atom with an atomic number of <n>
@	Chirality	S-configuration Chirality
@@	Chirality	R-configuration Chirality

```
  14. query = Chem.MolFromSmarts("C*C").
  15. match_list = [mol.GetSubstructMatch(query)
for mol in mol_list]
  16. MolsToGridImage(mols = mol_list, molsPerRow
= 4,
  17. highlightAtomLists = match_list).
  18.# SMARTS strings can be extended to specific
atoms.
  19.#Carbon, oxygen, or nitrogen that is connected
to carbon is connected to another carbon
  20. match_list = [mol.GetSubstructMatch(query)
for mol in mol_list]
  21. MolsToGridImage(mols = mol_list,
  22.       molsPerRow = 4,
  23.       highlightAtomLists = match_list).
```

14.3 Molecular Fingerprint

This section delves into molecular fingerprints, an important type of qualitative descriptor. It first clarifies their core representation form as binary strings and application scenarios, then separately elaborates on the principles, generation processes, and core advantages of SMARTS-based MACCS (Molecular Access System) structural key fingerprints, the widely used Extended Connectivity Fingerprints (ECFP), and their variant Functional Class Fingerprints (FCFP). Meanwhile, code examples are provided to demonstrate the calculation and matching methods of fingerprints, highlighting their central role in molecular similarity analysis.

14.3.1 What Is Molecular Fingerprint

First, we need to understand what a molecular fingerprint is. Molecular fingerprints, also known as "qualitative descriptors", are representations of chemical structures. A fingerprint that describes the structure and characteristics of a molecule is a particularly complex form of descriptor [11]. During encoding, fingerprints are usually represented as binary strings, and their settings generate a specific molecular fingerprint string in different ways. Each bit in the string represents a part of the numerically encoded descriptor, or in other words, it can define a fragment or structural bond of a molecule. Molecular fingerprints were originally designed to illustrate different sets of molecular descriptors, structural fragments, and possible connection pathways

through a molecule, but later they were used for analytical tasks such as similarity search, clustering, and classification [12].

14.3.2 Molecular Access System Structure Key

Molecular Access System Structural Keys (MACCS Keys) are structures encoded based on SMARTS. MACCS Keys are divided into two types of structures according to the length of the binary vector: one is a binary vector of length 166, and the other is a binary vector of length 960. Among them, the MACCS Keys of length 166 are the most commonly used because of their shorter length, and each bit represents different structural features of compounds. Each 1 and 0 in the binary vector has a one-to-one mapping with a substructure, indicating the presence or absence of a specific substructure in the molecule. MACCS Keys fingerprints can be used for rapid substructure screening in molecular databases [13].

14.3.3 Extended Connectivity Fingerprint

Extended Connectivity Fingerprint (ECFP) is a two-dimensional fingerprint used for molecular description and is one of the widely used chemical fingerprinting techniques at present [14]. ECFP is a circular fingerprint with many useful features, such as being computationally fast; not predefined, it can represent an infinite number of different molecular features (including stereochemical information); its features represent the presence of specific substructures, making it easier to interpret the analysis results; the ECFP algorithm can be customized to generate different types of circular fingerprints, optimized for various applications [15].

The ECFP algorithm is a modified and standardized implementation of the classic Morgan algorithm. It was proposed to address the issue of molecular isomorphism—i.e., to ensure that topologically isomorphic molecules (with identical atomic composition and connectivity but distinct atom numbering) generate consistent fingerprints—thereby reducing fingerprint redundancy and improving the reliability of structural representation. Comparing molecules can be difficult, but comparing strings is simple. Before comparing molecules, they just need to be quantified. The ECFP compares molecular fingerprints, which can be simply described as extracting the fingerprints of two molecules and comparing the corresponding elements. The more matching elements there are, the more similar the molecules are [16].

The generation of ECFP mainly undergoes three stages: initial atomic hash assignment, iterative hash value update, and final hash value deduplication. In the first stage, all information is a description of the atoms constituting the molecule and their environment, and each atom is assigned an identifier based on its characteristics. In the second stage, for a fixed number of iterations, the identifier of each atom is updated based on the identifiers of its neighboring atoms. Each identifier corresponds to a

substructure in the molecule. The number of iterations determines the size of the substructures that the algorithm can recognize. In the third stage, the same features that appear multiple times in the final generated feature list are simplified to a single representative. The purpose of ECFP is to capture the substructure features of the precise atomic environment [17].

The code for displaying the molecular ECFP is shown in Listing 14.2:

Listing 14.2 Calculate the ECFP Fingerprint of the Molecule

```
132. import deepchem as dc.
  133. from rdkit import Chem
  134. smiles= ['C1CCCCC1', 'O1CCOCC1'] .
  # Cyclohexane and Dioxane.
  135. mols = [Chem.MolFromSmiles(smile) for smile
in smiles]
  136. feat = dc.feat.CircularFingerprint(size=1024)
  137.arr = feat.featurize(mols).
  # arr is an array of 2×111 containing the proper
ties of 2 molecules
```

Functional-Class Fingerprint (FCFP) is a variant of ECFP. ECFP assigns values to atoms to generate structural features, but this specific approach is sometimes not desirable. In some cases, we prefer to see the types and roles of atoms within a molecule. The purpose of FCFP is to capture the general functional roles and types of atoms and incorporate them into several general atomic classes, such as hydrogen bond donors, acceptors, aromatic, halogen, etc. [18].

Characteristics of ECFPs
(1) ECFPs are computationally lightweight with fast processing speeds, effectively capturing a molecule's salient features
(2) They do not rely on predefined molecular features, enabling robust representation of novel structural variations
(3) While ECFPs encode extensive molecular information, some data loss is inherent. Consequently, a single fingerprint cannot uniquely trace back to its original molecule

14.4 Feature Engineering of Drug Molecules

This section systematically organizes the complete workflow of feature engineering for drug molecules. It first distinguishes the differences and characteristics between manual descriptors (e.g., fingerprints, SMILES) and deep learning-acquired descriptors, supplements specialized representation methods such as grid features and graph convolution, and then focuses on explaining the core objectives, four-step workflow of feature selection, and classification strategies, including supervised/unsupervised/semi-supervised, filter/wrapper/embedded approaches. Ultimately, it provides comprehensive guidance for efficiently screening effective molecular features and improving model performance.

14.4.1 What Are Molecular Representations?

We refer to all the different ways of describing a molecule using vectors as molecular representation or molecular features. Molecular representation is also known as "descriptors" of feature vectors. Molecular representation, with its core embodied in the descriptors of feature vectors, can be categorized into manual representation and learned representation. Manual representation entails designing a computer-interpretable molecular representation followed by manual engineering. Two widely adopted molecular representation approaches are molecular fingerprints and SMILES strings. The most prevalent type of molecular fingerprint is a binary vector composed of 1 s and 0 s, where each bit denotes the presence or absence of specific structural or functional features within the molecule. Although molecular fingerprints are flexible and easy to calculate in reaction prediction, a large amount of encoding of molecules may lead to the loss of some information [19]. SMILES strings are a single-line text format for molecular encoding. However, using text sequences to represent molecules has a significant drawback, which is the fragility of text sequences. A slight change in the text sequence may lead to a significant change in the molecular structure. Compared with manually crafted representations, molecular representations obtained in deep learning have better generalization ability and stronger expressive power, but they usually lack interpretability. We cannot explain the process by which deep learning generates molecular representations and what they represent. In learned representations, they can be divided into task-independent representations and task-specific representations. The former learns molecular features in an unsupervised manner, while the latter learns features related to specific tasks [20].

14.4.2 Other Molecular Representation Methods

In Sect. 14.3, we introduced a method for featurizing molecules, molecular fingerprints. Besides molecular fingerprints, there are also other methods for featerizing molecules, such as grid features and graph convolution [21].

Grid features were originally designed for the PDBBind database, specifically addressing the problem of feature extraction for protein–ligand binding structures. By considering the chemical interactions within the binding pocket and separately taking into account the features of both the protein and the ligand, the structural information of the ligand and the target protein is combined. Compared with other feature methods, grid features have been optimized in terms of computational speed, robustness, and universality. The intermolecular interactions listed by grid featureization, in addition to SPLIF fingerprints, also include salt bridges and hydrogen bonds between the protein and the ligand, cyclic molecular fingerprints of the ligand, pi-pi stacking interactions, and other chemical information [22].

Molecular fingerprints are manually featurized, while graph convolutional feature extraction is about learning representations. Graph convolution is similar in principle to convolutional neural networks, which perform feature extraction on the input image. After multiple layers of convolution operations, the resulting feature image becomes more abstract. Each pixel's feature vector in the feature map represents multiple features of a certain local area [23]. Unlike a convolutional neural network, the graph convolution input is a molecular graph. When the graph represents a molecule, each feature vector in the graph represents all the properties of an atom, including the atom type, charge, and hybridization type. The convolutional layer of a convolutional neural network computes the input features to generate new features. The same is true for a graph convolutional network, but a molecular graph is composed of nodes and edges. Graph convolution is performed on nodes and edges to generate new feature vectors. In a molecular graph, the local area is defined by the edges connecting the nodes. Graph convolutional networks are powerful tools for analyzing molecules, but they have an important limitation: the computation is based only on the molecular graph [24].

14.4.3 Feature Selection

Feature selection, also known as "feature subset selection", involves choosing appropriate and effective feature groups from a large number of original features to form new feature subsets. It is a process of detecting relevant features and eliminating irrelevant, redundant or noisy data [25]. The process of feature selection can generally be divided into four steps: generating feature subsets, evaluating feature subsets, stopping conditions, and validating results. The generated feature subsets can be classified into three types: no feature, all features, or random features. The evaluation of feature subsets is based on their dependence and independence on the algorithm.

Stopping conditions can be divided into generation conditions and evaluation conditions. Generation conditions include "whether the predetermined number of features has been reached or the predetermined number of iterations has been completed", while evaluation conditions include "whether the next subset is better or whether the current subset meets the evaluation criteria". The validity of the selected feature subsets is verified by comparing them with artificial datasets or real datasets. Reasonable feature selection can ensure the effectiveness of data processing and improve processing efficiency. The purpose of feature selection is to build simple and understandable generalized models, reduce model training time (improve computational efficiency), lower dimensionality, and reduce overfitting [26].

From the perspective of the effectiveness of supervision, feature selection methods can be classified into three types: supervised feature selection methods, unsupervised feature selection methods, and semi-supervised feature selection methods [27].

Supervised feature selection methods are mostly used in classification and regression tasks, aiming to select a subset of features for classifying or regressing different types of samples. Supervised feature selection evaluates the relevance of features based on their association with label information. A key characteristic of this approach is that model training exhibits strong dependence on the filtered features; additionally, the process inherently requires data partitioning. Notably, supervised feature selection methods are specifically designed for labeled datasets.

Unsupervised feature selection methods are typically applied to clustering problems. In unsupervised feature selection, the data is unlabeled, thus more time is required to obtain label information [28]. Unsupervised feature selection needs to evaluate the importance of features using information without labels. Utilizing unsupervised feature selection can provide better data description and reliability compared to unsupervised learning alone.

The data labels of semi-supervised feature selection methods lie between those of supervised and unsupervised feature selection methods. Unlike supervised feature selection methods, it does not have a large amount of label information, nor does it lack label information as in unsupervised feature selection methods. With a limited number of label information, it can complete feature selection. From an application perspective, semi-supervised feature selection methods are more in line with the actual situation.

Depending on the different feature selection strategies, feature selection methods can be classified into filter methods, wrapper methods, and embedded methods [29].

Filter methods are often used in classification and clustering tasks. As a preprocessing step, filter methods rank features [30]. The filter method is divided into two steps. First, the importance of features is ranked according to the feature evaluation criteria. Then, all the features of lower ranks are filtered out, and the higher-ranked features are selected and applied to the predictor. The advantage of the filter method is that it has a low computational cost and avoids overfitting. However, it also has a drawback, that is, there may be varying degrees of redundant information in the selected optimal feature subset. The filter method has been applied in related feature algorithms, minimum redundancy maximum relevance, and information gain.

In wrapper methods, the criterion for feature selection is the performance of the predictor, that is, the predictor is wrapped in a search algorithm that finds a subset with the best performance. Similar to filter methods, wrapper methods also consist of two steps: first, search for a feature subset that matches the desired criteria, and then evaluate the performance of the selected feature subset. Unlike filter methods, these two steps in wrapper methods need to be repeated until the iteration stops when the performance of the searched feature subset is optimal. The biggest drawback of wrapper methods is that they require a large amount of computation when selecting feature subsets. Therefore, in terms of computation, filter methods are usually more computationally efficient than wrapper methods.

Embedded methods lie between filter methods and wrapper methods. Their emergence inherits the advantages of both wrapper and filter methods and also makes up for their shortcomings. To reduce the computational load of repeated iterations in wrapper methods, embedded methods incorporate feature selection as part of the training process, rather than splitting the data into training and test sets [31].

In the current research on feature selection, some researchers have combined the three methods of filter, wrapper, and embedded methods, calling it the "hybrid feature selection method", and have included it as the fourth method of feature selection strategy. This approach addresses the instability and perturbation issues during feature selection, enhancing the robustness and credibility of the selected features in machine / deep learning.

References

1. Wessel, M.D., Jurs, P.C., Tolan, J.W., Muskal, S.M.: Prediction of human intestinal absorption of drug compounds from molecular structure. J. Chem. Inf. Comput. Sci. **38**, 726–735 (1998)
2. Hoffmann, R., Laszlo, P.: Representation in chemistry. Angew. Chem. Int. Ed. Engl. **30**, 1–16 (1991)
3. Fulton, R.L.: Sharing of electrons in molecules. J. Phys. Chem. **97**, 7516–7529 (1993)
4. Shenkin, P.S., McDonald, D.Q.: Cluster analysis of molecular conformations. J. Comput. Chem. **15**, 899–916 (1994)
5. Atkins, E.: Conformations in polysaccharides and complex carbohydrates. J. Biosci. **8**, 375–387 (1985)
6. Baldi, A.: Computational approaches for drug design and discovery: an overview. Syst. Rev. Pharmacy **1**, 99 (2010)
7. Atz, K., Grisoni, F., Schneider, G.: Geometric deep learning on molecular representations. Nat. Mach. Intell. **3**, 1023–1032 (2021)
8. Estrada, E.: Characterization of 3D molecular structure. Chem. Phys. Lett. **319**, 713–718 (2000)
9. O'Boyle, N.M.: Towards a universal SMILES representation-A standard method to generate canonical SMILES based on the InChI. J. Cheminform. **4**, 22 (2012)
10. Weininger, D.: SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. J. Chem. Inf. Comput. Sci. **28**, 31–36 (1988)
11. Stepišnik, T., Škrlj, B., Wicker, J., Kocev, D.: A comprehensive comparison of molecular feature representations for use in predictive modeling. Comput. Biol. Med. **130**, 104197 (2021)
12. Xue, L., Bajorath, J.: Molecular descriptors in chemoinformatics, computational combinatorial chemistry, and virtual screening. Comb. Chem. High Throughput Screening **3**, 363–372 (2000)

13. Rogers, D., Hahn, M.: Extended-connectivity fingerprints. J. Chem. Inf. Model. **50**, 742–754 (2010)
14. Taylor, R.D., MacCoss, M., Lawson, A.D.: Rings in drugs: Miniperspective. J. Med. Chem. **57**, 5845–5859 (2014)
15. Pozzan, A.: Molecular descriptors and methods for ligand based virtual high throughput screening in drug discovery. Curr. Pharm. Des. **12**, 2099–2110 (2006)
16. Chuang, K.V., Gunsalus, L.M., Keiser, M.J.: Learning molecular representations for medicinal chemistry: miniperspective. J. Med. Chem. **63**, 8705–8722 (2020)
17. Rogers, D., Brown, R.D., Hahn, M.: Using extended-connectivity fingerprints with Laplacian-modified Bayesian analysis in high-throughput screening follow-up. J. Biomol. Screen. **10**, 682–686 (2005)
18. Andrews, L.J.: Aromatic molecular complexes of the electron donor-acceptor type. Chem. Rev. **54**, 713–776 (1954)
19. Rutten, M.G., Vaandrager, F.W., Elemans, J.A., Nolte, R.J.: Encoding information into polymers. Nat. Rev. Chem. **2**, 365–381 (2018)
20. Miao, J., Niu, L.: A survey on feature selection. Procedia Comput. Sci. **91**, 919–926 (2016)
21. Dou, B., et al.: Machine learning methods for small data challenges in molecular science. Chem. Rev. **123**, 8736–8780 (2023)
22. Martinez, C.R., Iverson, B.L.: Rethinking the term "pi-stacking." Chem. Sci. **3**, 2191–2201 (2012)
23. Wu, Z., et al.: MoleculeNet: a benchmark for molecular machine learning. Chem. Sci. **9**, 513–530 (2018)
24. Pogliani, L.: From molecular connectivity indices to semiempirical connectivity terms: recent trends in graph theoretical descriptors. Chem. Rev. **100**, 3827–3858 (2000)
25. Yu, L., Liu, H.: Efficient feature selection via analysis of relevance and redundancy. J. Mach. Learn. Res. **5**, 1205–1224 (2004)
26. Subramanian, J., Simon, R.: Overfitting in prediction models–is it a problem only in high dimensions? Contemp. Clin. Trials **36**, 636–641 (2013)
27. Das, S., Das, A.K.: A graph-based approach for feature selection from higher order correlations. Int. J. Comput. Syst. Eng. **4**, 66–71 (2018)
28. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Comput. Electr. Eng. **40**, 16–28 (2014)
29. Vipin, K., Sonajharia, M.: Feature selection: a literature review. Smart Comput. Rev **4**, 211–229 (2014)
30. Li, J., et al.: Feature selection: a data perspective. ACM Comput. Surveys (CSUR) **50**, 1–45 (2017)
31. Noorizadeh, H., Sobhan Ardakani, S., Ahmadi, T., Mortazavi, S., Noorizadeh, M.: Application of genetic algorithm-kernel partial least square as a novel non-linear feature selection method: partitioning of drug molecules. Drug Testing Anal. **5**, 89–95 (2013)

Chapter 15
Prediction of Molecular Drug Properties

In drug chemistry research, elucidating the relationship between molecular structure and activity has always been of great significance. Early studies generally analyzed this issue by constructing quantitative structure–activity relationship models. In recent years, new methods based on artificial intelligence, especially deep learning models, have achieved remarkable results in many fields and have attracted increasing attention in the fields of chemistry and pharmacy [1]. Molecular property prediction is one of the key tasks in the computer-aided drug discovery process and plays a crucial role in many downstream applications, such as drug screening and drug design. Its main purpose is to predict the physical and chemical properties of molecules based on internal information such as atomic coordinates and atomic numbers, enabling the identification of compounds with desired properties from a large number of candidates and accelerating the speed of drug screening and design [2].

15.1 Pharmacokinetics

15.1.1 Introduction to Pharmacokinetics

Pharmacokinetics, often abbreviated as PK or pharmacokinetics, is a discipline that studies the dynamic laws of drugs in the body, including the time-dependent changes in the absorption, distribution, metabolism, and excretion (ADME) of drugs. It quantitatively describes this process using kinetic principles and mathematical processing methods. Currently, both the theory and methods of pharmacokinetics have permeated various fields such as toxicology, biopharmaceutics, and pharmacology, playing a significant role in drug research and development and serving as a key indicator for measuring the druggability.

The development of innovative drugs is a high-risk, high-investment and high-return industry. From laboratory research to the launch of a new drug on the market

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_15

is a long process, which involves a series of procedures such as synthesis and extraction, biological screening, preclinical trials including pharmacology and toxicology, formulation prescription, and stability tests, bioavailability tests and scale-up tests. It also requires going through many complex stages such as human clinical trials, registration and listing, and post-market supervision [3]. Such a complex process can bring about many unforeseen situations, and failure can occur at any stage. Once a company fails in its development, its huge investment will be lost. At present, new drug research and development shows characteristics such as longer and longer research and development time, higher and higher research and development costs, and more and more difficult discovery of new active entities. All these have led to a high failure rate in new drug research and development.

Pharmacokinetic studies play a crucial role in the development of innovative drugs. After a candidate drug enters clinical trials, it is reported that approximately 40% of candidate compounds are eliminated due to pharmacokinetic factors [4]. For instance, many candidate compounds are considered promising in in vitro studies, but in in vivo studies, they show very low or no in vivo activity, or have significant toxicity in vivo, which are situations that researchers do not wish to see.

Tips

The lack of in vivo activity of a compound is often attributed to suboptimal pharmacokinetic properties, such as excessive first-pass elimination, poor intestinal mucosal absorption leading to low bioavailability, rapid metabolism, short half-life, or inadequate penetration into target organs. Toxicity of a compound in vivo is primarily due to the formation of toxic metabolites during its metabolic processing.

Drugs absorbed from the gastrointestinal tract into the portal venous system must pass through the liver before entering systemic circulation. If the liver exhibits strong metabolic capacity toward the drug or if a significant portion is excreted into bile, the effective amount of the drug reaching systemic circulation is markedly reduced. This phenomenon is termed first-pass elimination. In some cases, drugs may also undergo partial metabolism within the intestinal wall cells during absorption, which is also classified as first-pass elimination. It is alternatively referred to as first-pass metabolism or the first-pass effect.

15.1.2 Introduction to ADMET

The five letters of ADMET stand for Absorption (absorption), Distribution (distribution), Metabolism (metabolism), Excretion (excretion), and Toxicity (toxicity) of drugs. A measures the process by which drugs enter the systemic circulation from the site of action; D represents the process by which drugs are transported to various

tissues, organs, or body fluids after absorption through cell membrane barriers; M refers to the structural transformation of drugs in the body due to their interaction with receptors or intestinal flora; E is the process by which drugs are excreted from the body in their original form or as metabolites; T is the toxicity of drugs to the body [5]. ADMET is a comprehensive study of the absorption, distribution, metabolism, excretion, and toxicity of drugs.

The properties of ADMET are very important factors in contemporary drug design and drug screening, which are related to whether drugs can be absorbed and metabolized by the human body. The evaluation of ADMET properties can effectively solve the problem of species differences, significantly increase the success rate of drug research and development, reduce the cost of drug development, decrease the occurrence of drug toxicity and side effects, and guide the rational use of drugs in clinical practice.

An ideal compound should possess high in vitro activity, low toxicity and favorable pharmacokinetic properties. Favorable pharmacokinetic properties refer to high bioavailability and an ideal half-life. To obtain ideal candidate compounds, researchers will conduct numerous rounds of screening based on pharmacology, toxicology and pharmacokinetics, and modify or optimize the lead compounds according to the results of each screening. This process is repeated until a batch of optimal candidate compounds is produced. It is evident that preclinical pharmacokinetic studies play a crucial role in the drug development system. Together with clinical pharmacology and toxicology studies, they form a complete and integrated drug screening and evaluation system.

Tips

A lead compound (or lead) refers to a chemically defined substance with specific biological activity, derived via diverse strategies, that serves as the foundation for further structural modification and optimization in modern drug discovery. It represents the critical starting point for developing novel therapeutic agents. During drug research, the identification of bioactive lead compounds through activity screening is fundamental to innovative drug discovery.

The early calculation of ADMET properties mainly focused on human or humanized functional proteins as “drug targets”, and combined in vitro experimental techniques with computer simulation methods to study the interaction between drugs and in vivo biophysical and biochemical barrier factors. Nowadays, the prediction of ADMET properties can be achieved by artificial intelligence methods such as machine learning or deep learning. Sections 15.3 and 15.4 will respectively introduce two typical methods of artificial intelligence for predicting the properties of drug molecules.

15.1.3 Dissociation Constant

The dissociation constant (pKa) is a very important property of organic compounds, determining their existence forms in media and thereby influencing their solubility, lipophilicity, bioaccumulation and toxicity. For drug molecules, pKa also affects their pharmacokinetic and biochemical properties [6]. In fields such as environmental chemistry, biochemistry, medicinal chemistry and drug development, accurately predicting the pKa values of organic compounds is of great significance.

pKa, as an important physicochemical property of drug molecules, is not only a significant indicator for evaluating their biological activity but also a crucial factor influencing the drug's absorption, distribution, metabolism, excretion, and toxicity in the body. Determining the pKa of a molecule through biological experiments is both time-consuming and labor-intensive. Therefore, accurately predicting the pKa of drugs during the drug development process can effectively reduce the risks and control the costs of drug research and development [7]. For known drug molecules, using high-performance computers to predict the properties of drug molecules based on their structures has been a hot topic in the field of bioinformatics in recent decades.

15.2 Lipinski's Rule

15.2.1 Introduction to Lipinski's Rule

The Lipinski's rule, also known as the Rule of Five, was proposed by Christopher A. Lipinski, a senior drug chemist at Pfizer, in 1997. It is used to evaluate whether drug analogues or compounds with definite pharmacological and biological activities have the potential to become orally active drugs in humans.

The specific content of Lipinski's rule is as follows: A small molecule drug should possess the following properties:

(1) Molecular weight <500;
(2) The number of hydrogen bond donors is <5;
(3) The number of hydrogen bond acceptors is <10;
(4) The lipid-water partition coefficient is <5;
(5) The number of rotatable keys shall not exceed 10.

In the field of drug development, the Lipinski rule is used to conduct an initial screening of compound databases, with the aim of eliminating molecules that are not suitable for drug development, narrowing the scope of screening and reducing the cost of drug research and development [8]. Over the long-term practice, medicinal chemists have simplified the Lipinski principle, forming the "four rules" and "three rules". However, the "four rules" and "three rules" are sometimes still referred to as

the "five rules", where the "five" refers to the threshold values of each rule being 5 or 500. The simplified "four rules" removed the limit on the number of rotatable bonds; the "three rules" further removed the limit on the number of hydrogen bond acceptors [9]. These are empirical rules summarized from oral drugs and are effective guiding principles for small molecule drug design.

15.2.2 Simple Program Implementation of Lipinski's Rule

Compound drugs are a major type of medication. When conducting related research, it is usually necessary to manipulate and display their structures, as well as calculate molecular weight, chemical descriptors, etc. RDKit is an open-source toolkit for chemoinformatics, which is based on the manipulation of 2D and 3D molecular structures of compounds and uses machine learning methods to generate compound descriptors, fingerprints, calculate compound structural similarity, and display 2D and 3D molecules, etc. In Listing 15.1, we have demonstrated how to compute the Lipinski rule using a simple code.

Listing 15.1 Simple Program Implementation of Lipinski's Rule

```
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import rdMolDescriptors as rdes
drugbank = Chem.SDMolSupplier(
   './data/drugbank/drugbank_approved_logP.sdf')
def lipinski_wt_limit(m):
   return Descriptors.MolWt(m) <= 500
def lipinski_logp_limit(m):
   return Descriptors.MolLogP(m) <= 5
def lipinski_hba_limit(m):
   return rdes.CalcNumLipinskiHBA(m) <= 10
def lipinski_hbd_limit(m):
   return rdes.CalcNumLipinskiHBD(m) <= 5
def lipinski_violations(m):
   return 4-sum((lipinski_wt_limit(m),
           lipinski_logp_limit(m),
           lipinski_hba_limit(m),
           lipinski_hbd_limit(m)))
```

Next, we test the implementation program of Lipinski's rule and count the number of small molecules that meet each condition respectively. The code is shown in Listing 15.2.

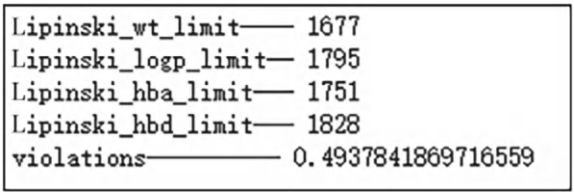


Fig. 15.1 Prediction of drug molecule properties in machine learning

Listing 15.2 Test

```
# exp-1
print("ipinski_wt_limit----", sum(lipinski_wt_limit(m)
    for m in drugbank if m))
# exp-2
print("ipinski_logp_limit--", sum(lipinski_logp_limit(m)
    for m in drugbank if m))
# exp-3
print("ipinski_hba_limit---", sum(lipinski_hba_limit(m)
    for m in drugbank if m))
# exp-4
print("ipinski_hbd_limit---", sum(lipinski_hbd_limit(m)
    for m in drugbank if m))
# exp-4
violations = [lipinski_violations(m) for m in drugbank if m]
print("violations----------",sum(violations)/
len(violations))
```

The program's running result is shown in Fig. 15.1

15.3 Prediction of Drug Molecule Properties in Machine Learning

One of the most important steps in applying machine learning to molecular data is to transform the data into a form suitable for learning algorithms. This process is called "featureization" and involves converting a molecule into some kind of vector or tensor [10]. There are many different ways to do this, and the choice of featureization typically depends on the problem at hand. Common methods include molecular fingerprints and ConvMol objects for graph convolution. Here, we will use molecular fingerprints to featureize the data and predict molecular properties.

15.3.1 Processing Input Data

To build a local database containing the pKa values of drug molecules, we selected the drug molecule database DrugBank for data organization and mining. The drug molecule dataset in the database is saved in SDF file format, which includes relevant information about the drug molecules [11]. By using a scripting language, we can extract the structures of the drug molecules and other information we need from the database [12].

In Listing 15.3, the LoadSDF method from the rdkit.Chem.PandasTools module is used to read the drug molecule file, and the relevant properties of the drug molecules are formatted and displayed, as shown in Fig. 15.2. The presentation of the original data in the SDF file is shown in Fig. 15.3.

Listing 15.3 Import Relevant Tool Modules and Read Data

```
from rdkit.Chem import PandasTools
df = PandasTools.LoadSDF('drugbank_approved_logP.sdf').
pd.set_option('max_columns', None)
print(df.head())
```

Import the relevant tool modules, as shown in Listing 15.4.

DATABASE_ID	DATABASE_NAME	SMILES	INCHI_IDENTIFIER
DB00006	drugbank	CC[C@H](C)[C@H](NC(=O)[C@H](CCC(O)=O)NC(=O)[C@...	InChI=1S/C98H138N24O33/c1-5-52(4)82(96(153)122...
DB00014	drugbank	CC(C)C[C@H](NC(=O)[C@@H](COC(C)(C)C)NC(=O)[C@H...	InChI=1S/C59H84N18O14/c1-31(2)22-40(49(82)68-3...
DB00027	drugbank	CC(C)C[C@@H](NC(=O)CNC(=O)[C@@H](NC=O)C(C)C)C(...	InChI=1S/C96H135N19O16/c1-50(2)36-71(105-79(11...
DB00035	drugbank	NC(=O)CC[C@@H]1NC(=O)[C@H](CC2=CC=CC=C2)NC(=O)...	InChI=1S/C46H64N14O12S2/c47-35(62)15-14-29-40(...
DB00050	drugbank	CC(C)C[C@H](NC(=O)[C@@H](CCCNC(N)=O)NC(=O)[C@H...	InChI=1S/C70H92ClN17O14/c1-39(2)31-52(61(94)82...

Fig. 15.2 Partial attributes of drug molecules in the file

```
  Mrv0541 02231216082D

155160  0  0  1  0            999 V2000
  -19.1798  -19.8286    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
  -18.5326  -19.3169    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -17.7659  -19.6216    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
  -18.6521  -18.5006    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
  -19.4188  -18.1960    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -19.5383  -17.3797    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -20.3050  -17.0751    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -20.4245  -16.2588    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
  -21.1912  -15.9541    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
  -21.8384  -16.4658    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -21.7189  -17.2821    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
  -22.6051  -16.1611    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
  -23.3022  -16.6023    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -23.9372  -16.0757    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -23.6326  -15.3090    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -22.8093  -15.3618    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
  -22.2826  -14.7268    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  -21.4693  -14.8653    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
  -22.5692  -13.9531    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
```

Fig. 15.3 Screenshot of the data presentation part of the SDF file

Listing 15.4 Importing Relevant Tool Modules

```
import pandas as pd
from rdkit import Chem
from deepchem import feat as fp
```

In Code Listing 15.5, the 'Chem.SDMolSupplier' is used to obtain the file reader object 'suppl2'. It is important to note that when iterating through 'suppl2', each 'mol' must also be converted into an iterable object to meet the input requirements of subsequent featurization methods. The featurization process involves two steps: defining a featurization object ('CircularFingerprint') with specified constraint rules (parameters such as radius, size, and chiral), and then applying the featurization method to generate features, which are added to a list. In this example, we set the radius to 2 and the fingerprint length to 2048. Storing the generated features in a list prepares the data for uniform formatting and storage in a file. The Pandas data processing framework is then used to structure the data and save it to 'ecfp-drugbank-all.csv'. The data in the file 'ecfp-drugbank-all.csv' is illustrated in Fig. 15.4.

	0	1	2	3	4	5	6	7	8	9	...	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8650	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8651	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8652	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8653	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0	0	0
8654	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

8655 rows × 2048 columns

Fig. 15.4 Parts of ECFP fingerprint

Listing 15.5 Importing Relevant Tool Modules

```
Suppl2 = Chem.SDMolSupplier('./data/drugbank/drugbank_
approved_logP.sdf').
Feature list = [].
for mol in suppl2:
mol = [mol].
engine = fp.CircularFingerprint(radius=2, size=2048,
chiral=False,
bonds=True,features=False,sparse=False, smiles=False)
feature = engine(mol)
feature_list.extend(feature)
ID_feature_df = pd.DataFrame(feature_list)
ID_feature_df.to_csv('ecfp-drugbank-all.csv',index=0)
```

The running result is shown in Fig. 15.4. The total length of the fingerprint is 2048, which cannot be fully displayed in the figure.

15.3.2 Machine Learning for Predicting Properties of Drug Molecules

We will use the relevant modules from Pandas, Numpy, Sklearn and Scipy, as shown in Listing 15.6.

```
[-0.55 -0.96 -2.67 ...  3.05  3.82  2.61]
len(Y): 8655
```

Fig. 15.5 Dataset Y is a one-dimensional array

Listing 15.6 Importing Relevant Toolkits and Modules

```
from pandas import read_csv
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import scipy
import matplotlib.pyplot as plt
```

In Listing 15.7, the read_csv method in pandas is used to read the data from the file. Since the data in the CSV file is separated by commas, the parameter delimiter = ',' needs to be set. The resulting data object, dataframe, is in the DataFrame format, and the dataset is obtained from it. We take all the data in the dataset as the input values X, and the data in the file drugbank_all_logP.AlogP.value under the specified folder as the result values Y. The dataset Y is a one-dimensional array, as shown in Fig. 15.5. The train_test_split method is used to randomly split the dataset into a training set and a test set in an 8:2 ratio. The first two parameters of the train_test_split method are the input dataset and the result dataset, respectively. The test_size parameter is the ratio of the dataset split, and the random_state parameter controls the mode of splitting the training set and the test set each time.

Tips

Specifying random_state as a number can ensure that the dataset is split in the same way at any time and on any device, meaning that the resulting training and test sets are consistent. However, when random_state is set to different numbers, the effect of splitting the dataset using the train_test_split method will definitely be different. This shows that the so-called randomness in computers is not truly random.

Listing 15.7 Reading the Data File and Dividing It into Training and Test Sets

```
dataframe = read_csv("./data/drugbank/ecfp-drugbank-
all.csv", delimiter=',')
dataset = dataframe.values
X = dataset[:,:]
Y = np.loadtxt("./data/drugbank/drugbank_all_
logP.AlogP.value")
X_train, X_test, Y_train, Y_test = train_test_split(
X, Y, test_size=0.2, random_state=42)
```

Define the random forest prediction model object 'estimators', where the number of sub-trees to be built is set to 90, the maximum depth of each tree is 24, at least 6 nodes are required for splitting nodes based on attributes, the minimum number of samples in each leaf node is 5, and the maximum number of features used by a single decision tree is the square root of the total number of features. This model uses 'oob_score', which is a very powerful model validation technique specifically designed for the random forest algorithm to achieve the minimum variance result. Then load the training set 'X_train' and 'Y_train' into the model, and save the run model as 'history'. Through the 'oob_score_' attribute of 'history', we can intuitively understand the accuracy of this model for prediction. This process is shown in Listing 15.8. The program's running result is 'training accuracy = 0.671'.

Listing 15.8 Training the Dataset with Random Forest Regression and Evaluating the Model Performance

```
estimators=RandomForestRegressor(n_estimators=90, max_
depth=24,
min_samples_split = 6,min_samples_leaf = 5,max_features =
'sqrt', oob_score = True).
history = estimators.fit(X_train, Y_train)
print('training accuracy:', history.oob_score_)
```

In Listing 15.9, the trained model is used to make predictions on the test set, and the predictions are stored in a one-dimensional array. The results of the test set after being predicted by the model are stored. By drawing a scatter plot with the predicted values on the x-axis and the true values on the y-axis, we can visually observe the overall prediction effect of the model on the test set data, as shown in Fig. 15.6. The closer the scatter points are to the line y = x, the more accurate the prediction is.

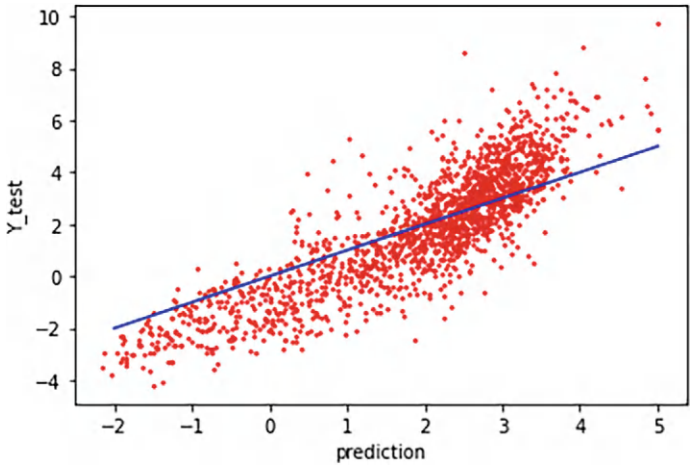


Fig. 15.6 Scatter plot of predicted data on the test set

Listing 15.9 Predicting the Test Set Using the Model

```
prediction = estimators.predict(X_test)
plt.scatter(prediction,Y_test, color='red',s=2)
x = np.linspace(-2, 5, 10000)
plt.plot(x,x,color="blue")
plt.xlabel("prediction")
plt.ylabel("Y_test")
plt.show()
```

15.4 Prediction of Drug Molecule Properties in Deep Learning

15.4.1 Feature Processing and Dataset Partitioning

The ROMol attribute in the SDF file records the topological structure of the drug molecule. We will use the ROMol attribute values of the molecules in the database to predict the pKa values of the drug molecules. First, we need to import the relevant tool modules, as shown in Listing 15.10.

	DATABASE_ID	SMILES	INCHI_IDENTIFIER	New_pKa
0	DB00006	CC[C@H](C)[C@H](NC(=O)[C@H](CCC(O)=O)NC(=O)[C@...	InChI=1S/C98H138N24O33/c1-5-52(4)82(96(153)122...	3.177101
1	DB00014	CC(C)C[C@H](NC(=O)[C@@H](COC(C)(C)C)NC(=O)[C@H...	InChI=1S/C59H84N18O14/c1-31(2)22-40(49(82)68-3...	9.818594
2	DB00027	CC(C)C[C@@H](NC(=O)CNC(=O)[C@@H](NC=O)C(C)C)C(...	InChI=1S/C96H135N19O16/c1-50(2)36-71(105-79(11...	11.945820
3	DB00035	NC(=O)CC[C@@H]1NC(=O)[C@H](CC2=CC=CC=C2)NC(=O)...	InChI=1S/C46H64N14O12S2/c47-35(62)15-14-29-40(...	11.344259
4	DB00050	CC(C)C[C@H](NC(=O)[C@@H](CCCNC(N)=O)NC(=O)[C@H...	InChI=1S/C70H92ClN17O14/c1-39(2)31-52(61(94)82...	11.590298

Fig. 15.7 Partial attribute data in the df data table

Listing 15.10 Importing Relevant Tool Modules

```
from rdkit import Chem
import pandas as pd
from rdkit.Chem import Descriptors
from rdkit.ML.Descriptors import MoleculeDescriptors
from rdkit.Chem import PandasTools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

In Listing 15.11, the SDF file drugbank_approved_logP.sdf is loaded, the values of the column with the label JCHEM_PKA are read, converted to the float type, and then saved under the label New_pKa. Figure 15.7 shows some of the attribute data in the df data table.

Listing 15.11 Reading Data Files

```
df=PandasTools.LoadSDF('drugbank_approved_logP.sdf')
pd.set_option('max_columns', None)
df["New_pKa"] = [float(i)foriin df["JCHEM_PKA"].values].
print(df.head())
```

Calling Descriptors.descList can retrieve the attribute names of molecular descriptors, as shown in Fig. 15.8.

In Listing 15.12, a descriptor calculation object calc is created using MolecularDescriptorCalculator(), which can query all the property values of a molecule. The header variable stores the property labels that the calc object can query, which will serve as the header of the final feature table. The calc object is used to calculate the ROMol column in the df table, and the query results are saved in the list rdkit_

```
['MaxEStateIndex', 'MinEStateIndex', 'MaxAbsEStateIndex', 'MinAbsEStateIndex', 'qed',
'MolWt', 'HeavyAtomMolWt', 'ExactMolWt', 'NumValenceElectrons', 'NumRadicalElectrons',
'MaxPartialCharge', 'MinPartialCharge', 'MaxAbsPartialCharge', 'MinAbsPartialCharge',
'FpDensityMorgan1', 'FpDensityMorgan2', 'FpDensityMorgan3', 'BCUT2D_MWHI', 'BCUT2D_MWLO
W', 'BCUT2D_CHGHI', 'BCUT2D_CHGLO', 'BCUT2D_LOGPHI', 'BCUT2D_LOGPLOW', 'BCUT2D_MRHI',
'BCUT2D_MRLOW', 'BalabanJ', 'BertzCT', 'Chi0', 'Chi0n', 'Chi0v', 'Chi1', 'Chi1n', 'Chil
v', 'Chi2n', 'Chi2v', 'Chi3n', 'Chi3v', 'Chi4n', 'Chi4v', 'HallKierAlpha', 'Ipc', 'Kapp
a1', 'Kappa2', 'Kappa3', 'LabuteASA', 'PEOE_VSA1', 'PEOE_VSA10', 'PEOE_VSA11', 'PEOE_VS
A12', 'PEOE_VSA13', 'PEOE_VSA14', 'PEOE_VSA2', 'PEOE_VSA3', 'PEOE_VSA4', 'PEOE_VSA5',
```

Fig. 15.8 Partial attribute names of molecular descriptors

2d_desc. Then, the query property labels header and the query results are combined into a feature table df_final, and a new column attribute New_pKa is added at the end of the table. Before the feature table df_final is used as a dataset, it is necessary to check if there are any null values in the table. If there are, the corresponding rows are removed. The program's output is shown in Fig. 15.9.

Listing 15.12 Feature Extraction

```
rdkit_2d_desc = []
calc = MoleculeDescriptors.MolecularDescriptorCalculator([x[0]
for x in
Descriptors._descList])
header = calc.GetDescriptorNames()
for i in range(len(df)):
ds = calc.CalcDescriptors(df["ROMol"][i])
rdkit_2d_desc.append(ds)
df_final = pd.DataFrame(rdkit_2d_desc,columns=header)
df_final=pd.concat([df_final,df["New_pKa"]],axis=1)
is_NaN = df_final.isnull()
row_has_NaN = is_NaN.any(axis=1)
rows_with_NaN = df_final[row_has_NaN]
```

fr_thiazole	fr_thiocyan	fr_thiophene	fr_unbrch_alkane	fr_urea	New_pKa
0	0	0	1	0	3.177101
0	0	0	1	1	9.818594
0	0	0	0	0	11.945820
0	0	0	1	0	11.344259
0	0	0	2	1	11.590298

Fig. 15.9 Partial screenshot of the df final data table

```
df_final.drop(index=rows_with_NaN.index,inplace=True)
print(df_final.head())
```

In the data table df_final, the first $n - 1$ columns are the input data for the neural network, and the sixth column is the corresponding real result data. Due to the different natures of each evaluation index, they usually have different dimensions and magnitudes. When the levels among the various indicators vary greatly, if the original index values are directly used for analysis, the indicators with higher values will be highlighted in the comprehensive analysis, while the indicators with lower values will be relatively weakened. Therefore, to ensure the reliability of the results, it is necessary to normalize the original index data. Here, the Min–Max normalization method is used. Subsequently, the train_test_split is used to divide the training set and the test set, as shown in Listing 15.13.

Listing 15.13 Splitting the Dataset

```
X=df_final.values[:,0:-1]
y=df_final.values[:,-1]
scaler=MinMaxScaler()
X=scaler.fit_transform(X)
X_train1,X_test1,y_train, y_test = train_test_split(X, y,
test_size=0.2,
random_state=101)
```

15.4.2 Deep Learning for Predicting Properties of Drug Molecules

The Sequential model structure is a linear stack of network layers. It is a simple linear structure without any extra branches, consisting of multiple layers stacked together [13]. As can be seen from the code, this is a 7-layer network structure, including 4 fully connected layers and 3 The model employs a dropout layer and selects the Adam optimizer along with the mean squared error (MSE) loss function.

The data is loaded into the model, with the first two parameters being the input and output of the training data respectively, and 'validation_data' acting as the test set [14]. The number of samples required for one training of the network is 'batch_size = 128', and the total number of iterations for training is 400. The specific implementation is shown in Listing 15.14. The detailed information during the model training process is shown in Fig. 15.10.

```
6/6 [==============================] - 0s 5ms/step - loss: 3.7753 - val_loss: 4.1367
Epoch 393/400
6/6 [==============================] - 0s 5ms/step - loss: 3.3696 - val_loss: 4.4091
Epoch 394/400
6/6 [==============================] - 0s 4ms/step - loss: 3.2914 - val_loss: 4.1826
Epoch 395/400
6/6 [==============================] - 0s 5ms/step - loss: 3.3773 - val_loss: 4.2328
Epoch 396/400
6/6 [==============================] - 0s 4ms/step - loss: 3.3208 - val_loss: 4.4095
Epoch 397/400
6/6 [==============================] - 0s 5ms/step - loss: 3.3318 - val_loss: 4.3793
Epoch 398/400
6/6 [==============================] - 0s 4ms/step - loss: 3.6522 - val_loss: 4.0488
Epoch 399/400
6/6 [==============================] - 0s 5ms/step - loss: 3.3757 - val_loss: 4.0351
Epoch 400/400
6/6 [==============================] - 0s 4ms/step - loss: 3.4886 - val_loss: 4.6354
```

Fig. 15.10 Detailed information during model training process

Listing 15.14 Building a Neural Network Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
model=Sequential()
model.add(Dense(200,activation="tanh"))
model.add(Dropout(0.2))
model.add(Dense(100,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(50,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1,activation="linear"))
model.compile(optimizer='adam',loss='mse')
model.fit(x=X_train,y=y_train,
validation_data=(X_test,y_test),batch_size=128,
epochs=400)
```

The 'fit' function of the model returns a 'history' object, whose 'history.history' attribute records the values of the loss function and other metrics as the number of epochs increases. To visualize the changes in the loss function, the code is shown in Listing 15.15. The trend of the loss function values can be more easily observed through the loss function change graph shown in Fig. 15.11. Among them, 'loss' is the loss value of the training set, and the 'loss' value is an important basis for updating the parameters of the neural network [15]. 'val_loss' is the loss function of the test set, and the 'val_loss' value does not affect the parameters of the neural network.

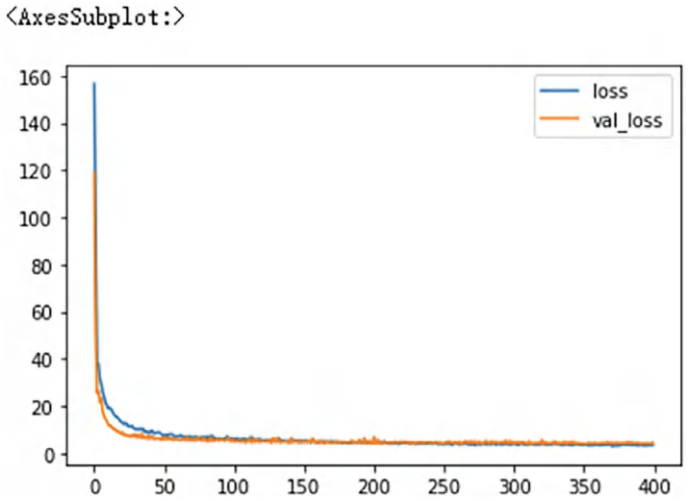


Fig. 15.11 Graph of the change in loss function

Listing 15.15 Visualization Display of Loss Function Changes

```
loss=pd.DataFrame(model.history.history)
loss.plot()
```

In Listing 15.16, the trained model is used to predict the test set, and the results are saved in the list y_predict. A scatter plot is drawn with the predicted values on the x-axis and the true values on the y-axis, as shown in Fig. 15.12. We can visually observe the overall prediction effect of the model on the test set data. The closer the scatter points are to the line y = x, the more accurate the prediction is.

Listing 15.16 Evaluation of Model Prediction Effectiveness

```
import matplotlib.pyplot as plt
import numpy as np
y_predict=model.predict(X_test)
plt.scatter(y_predict,y_test, color='red',s=4)
x = np.linspace(0,20, 10000)
plt.plot(x,x,color="blue")
plt.xlabel("y_predict")
plt.ylabel("Y_true")
plt.show()
```

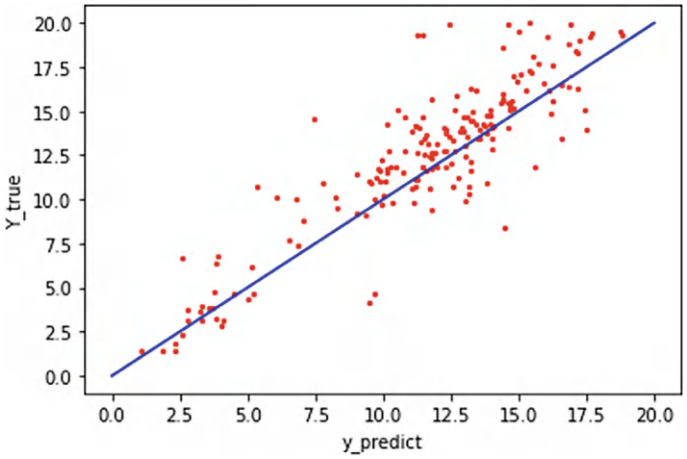


Fig. 15.12 Overall prediction result of the model on the test set

Recently, large language models (LLMs) have also been used as molecular descriptors. For example, a framework named KFLM2 integrates fine-tuned LLMs with molecular graph representations to enhance molecular property prediction [16]. The work highlights that fine-tuning significantly enhances the LLM's ability to generate informative SMILES embeddings, which capture semantic molecular features not explicitly encoded in structural graphs. Furthermore, fusing these LLM-derived embeddings with molecular graph representations consistently improves predictive accuracy across both regression and classification tasks compared to using either modality alone. This synergy provides complementary information, with attention-based fusion mechanisms often yielding the highest performance gains.

References

1. Klebe, G., Abraham, U.: On the prediction of binding properties of drug molecules by comparative molecular field analysis. J. Med. Chem. **36**, 70–80 (1993)
2. Shen, J., Nicolaou, C.A.: Molecular property prediction: recent trends in the era of artificial intelligence. Drug Discov. Today Technol. **32**, 29–36 (2019)
3. Urso, R., Blardi, P., Giorgi, G.: A short introduction to pharmacokinetics. Eur. Rev. Med. Pharmacol. Sci. **6**, 33–44 (2002)
4. Zeng, X., et al.: Accurate prediction of molecular properties and drug targets using a self-supervised image representation learning framework. Nat. Mach. Intell. **4**, 1004–1016 (2022)
5. Hodgson, J.: ADMET—turning chemicals into drugs. Nat. Biotechnol. **19**, 722–726 (2001)
6. Lee, D., Kim, J., Lee, G.: Simple methods to determine the dissociation constant, Kd. Mol. Cells **47**, 100112 (2024)
7. Stangherlin, S., Ding, Y., Liu, J.: Dissociation constant (Kd) measurement for small-molecule binding aptamers: homogeneous assay methods and critical evaluations. Small Methods **9**, 2401572 (2025)

8. Lohit, N., et al.: Description and in silico ADME studies of US-FDA approved drugs or drugs under clinical trial which violate the Lipinski's rule of 5. Lett. Drug Des. Discov. **21**, 1334–1358 (2024)
9. Nhlapho, S., et al.: Druggability of pharmaceutical compounds using Lipinski rules with machine learning. Sci. Pharm. **3**, 177–192 (2024)
10. Cheng, F., Zhao, Z.: Machine learning-based prediction of drug–drug interactions by integrating drug phenotypic, therapeutic, chemical, and genomic properties. J. Am. Med. Inform. Assoc. **21**, e278–e286 (2014)
11. Menden, M.P., et al.: Machine learning prediction of cancer cell sensitivity to drugs based on genomic and chemical properties. PLoS ONE **8**, e61318 (2013)
12. Walters, W.P., Barzilay, R.: Applications of deep learning in molecule generation and molecular property prediction. Acc. Chem. Res. **54**, 263–270 (2020)
13. Aly, M., Alotaibi, A.S.: Molecular property prediction of modified gedunin using machine learning. Molecules **28**, 1125 (2023)
14. Galushka, M., et al.: Prediction of chemical compounds properties using a deep learning model. Neural Comput. Appl. **33**, 13345–13366 (2021)
15. Pang, C., Tong, H.H., Wei, L.: Advanced deep learning methods for molecular property prediction. Quant. Biol. **11**, 395–404 (2023)
16. Xie, L., Jin, Y., Xu, L., Chang, S., Xu, X.: Fusing domain knowledge with a fine-tuned large language model for enhanced molecular property prediction. J. Chem. Theory Comput. **21**, 6743–6758 (2025)

Chapter 16
De Novo Molecular Generation

This chapter focuses on de novo molecular generation, covering core content such as lead compound optimization (traditional and computer-aided methods), drug molecule design principles, de novo molecular generation concepts, classification, deep learning applications, and current challenges. It systematically elaborates on the theoretical basis, technical methods, and practical applications of de novo molecular generation, providing a comprehensive overview for related research and practice.

16.1 Optimization of Lead Compounds

The discovered lead compounds cannot be immediately applied in clinical practice because they may have problems such as an unsatisfactory chemical structure, weak binding to the target, or weak biological activity. Chemical modifications are needed to make the lead compounds achieve the expected pharmacological effects and optimize them into ideal drugs. We call this process lead compound optimization [1].

Before conducting the optimization of lead compounds, two points need to be clarified: First, the optimization objective should be defined based on the expected pharmacology, the modification targets should be determined, and an appropriate optimization route should be formulated, while ensuring that specific problems are analyzed specifically. Second, a thorough understanding of the properties, characteristics, and existing issues of the lead compounds to be optimized is necessary.

The following three points should be noted when optimizing lead compounds:

(1) The principle of minimal modification means that when optimizing lead compounds, priority should be given to analogues with molecular structures similar to those of the lead compounds, or to compounds whose properties can be altered with only minor structural changes.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_16

(2) Improve the pharmacological activity and ADMET properties of compounds.
(3) Economic evaluation. In addition to intellectual property issues, the optimization of lead compounds should also take into account a comprehensive assessment of whether the synthetic route is simple and whether the synthetic raw materials or intermediates are inexpensive and readily available.

With the development of computer technology, molecular generation technology has also made significant progress, giving rise to traditional combinatorial drug chemistry methods and artificial intelligence-based de novo generation methods.

16.2 Principles of Drug Molecule Design

This section begins with an introduction to the basic idea of drug design.

16.2.1 Prodrug Design

Prodrugs are compounds that need to undergo biotransformation before exerting their pharmacological effects [2]. Generally speaking, the structural modification types of prodrug design mainly include ester prodrugs, phosphate or phosphoester prodrugs, carbonate and carbamate prodrugs, amide prodrugs, and oxime prodrugs.

Prodrugs can be roughly classified into two major categories: carrier prodrugs and bio-precursor prodrugs. Bio-precursor prodrugs are new compounds obtained by molecular modification of active compounds, and this new compound serves as the substrate of metabolic enzymes [3]. The expected active molecule is the active metabolite produced through enzymatic metabolism. The principle of carrier prodrugs is to covalently link biologically active drugs with carriers to influence the physicochemical properties of the drugs, and finally produce active drugs under the influence of enzymes.

In drug design, prodrug design has become a widely used and effective approach. Prodrug design not only enables structural modification of classic drugs containing carboxyl, hydroxyl, and amino groups to form prodrugs such as esters, carboxylic acid esters, and amino acid esters, but also allows for the creation of novel prodrugs with structures like azo-type, nitric oxide-type, and open-ring/closed-ring types, which can maintain or enhance the efficacy of the original drugs while addressing certain limitations of the parent compounds.

The four principles of prodrug design are as follows [4].

(1) After covalent bonding, it can be broken down through chemical reactions and enzymatic hydrolysis in the human body.
(2) The process of preparing prodrugs should be easy to achieve, and the prepared prodrugs should be inactive or have lower activity than the original drugs.

(3) The carrier molecules should be readily available and inexpensive. At the same time, the obtained carrier molecules should be non-toxic and have no physiological activity.
(4) Prodrugs can be quantitatively converted in the human body and can provide sufficient feedback kinetics fast enough.

The purpose and application of prodrug design can be summarized into the following four aspects.

(1) Enhance drug absorption and biofilm permeability. The physicochemical properties of drugs determine their biofilm permeability [5], especially the drug's lipid-water partition coefficient. Based on this, connecting drugs with lipophilic carriers can significantly increase drug absorption, making it easier for them to cross cell membranes through passive diffusion. There are many drug administration methods where drug absorption relies on passive diffusion, such as oral absorption, rectal administration, ocular administration, and transdermal administration, etc.
(2) Targeted drug delivery of prodrugs. The ultimate goal of drug administration is to achieve targeted delivery, and prodrugs also need to fulfill this objective. Targeted delivery of prodrugs can be accomplished through four pathways: transporter-mediated delivery, passive enrichment within organs, selective metabolic activation by enzymes, and antigen-targeting. There are generally two approaches to using prodrugs to target drugs to specific sites in the human body: one is to selectively transport the parent drug to the site of action; the other is to enable the prodrug to reach different parts of the body while only achieving biological activity and demonstrating biological effects at the target organ.
(3) Prolonging the duration of drug action. Unless a drug can accumulate in fat, the duration of action of an orally administered drug is usually much shorter than its transit time through the gastrointestinal tract. For drugs that are rapidly cleared from the body, the duration of action is often even shorter, so they must be administered repeatedly within 24 h to maintain the blood drug concentration at the therapeutic level. To prolong the duration of action of such drugs, they can be formulated as lipophilic prodrugs and rapidly dissolved or suspended in an oily medium, then administered by deep intramuscular injection. For example, after injection of the drug fluphenazine decanoate [6], it usually takes 24–72 h to take effect, and its efficacy can last for an average of 3–4 weeks.
(4) Enhance the water solubility and stability of drugs. By applying the prodrug principle and introducing hydrophilic groups into the molecular structure, the water solubility can be improved, addressing the issue of insufficient hydrophilic groups and low water solubility in some drug molecules. For instance, steroid anti-inflammatory drugs such as betamethasone and dexamethasone can be esterified with phosphoric acid or organic dicarboxylic acids through the hydroxyl groups in their molecules to form salts with good water solubility. In the body, these salts are enzymatically hydrolyzed to release the parent compounds and exert their effects.

16.2.2 Twin Drug Design

A twin drug is a formulation in which two pharmacophores are linked through a covalent bond. Twin drugs are not only a way to optimize lead compounds but also a typical approach for new drug design [7]. By combining to form twin drugs, if the twin drugs can be cleaved into the original two drugs in the body, it can change the pharmacokinetic and pharmacodynamic properties; if the twin drugs do not cleave in the body but act as a whole molecular structure, it can produce completely different pharmacological effects.

There are two types of twin drugs: identical twin drugs and different twin drugs. The way to produce twin drugs by splicing can be achieved through various connecting groups to adjust the spatial distance between the two pharmacophores. Two atoms can also be directly connected without a connecting group, or even overlap to form a certain structure and bond. Identical twin drugs refer to twin drugs formed by connecting two identical groups or molecules. Homologous twin drugs are new molecules produced by covalently bonding many identical elements or fragments within one atom, mainly aiming to obtain drugs with higher activity and selectivity than single-molecule or monomer drugs. Different twin drugs are twin drugs formed by connecting two different groups or molecules. By covalently bonding multiple different molecules or fragments to form new molecules, when diseases mediated by multiple enzymes or receptors exist, these heterologous twin drugs can simultaneously stimulate multiple targets, thereby enhancing the efficacy of the drug.

16.2.3 Soft Drug Design

The principle of soft drugs is based on in-depth research into the theoretical process of human metabolism. According to the mechanism of the metabolic process, it involves the research and design of active metabolites, inactive metabolites, soft analogues, active soft substances, and natural soft substances designed through controlled-release methods, all of which are biologically active drug molecules [8]. After exerting their effects in the human body, soft drugs are gradually metabolized and inactivated through predictable and controllable metabolic functions, transforming into inactive and non-toxic compounds, thus avoiding side effects. Compared with the original drugs, their efficacy is only higher, and they significantly reduce the toxicity and side effects of the original drugs. This is a highly recommended new drug development method.

The main differences between soft drugs and prodrugs lie in two aspects: First, their lead compounds are different. Prodrugs are all based on the original drug as the lead compound, while the lead compound of soft drugs can be either the original drug or its intermediate metabolite. Second, their modes of action are different. Prodrugs are designed to be inactive in vitro and only become active after the original drug is produced at the target site, while soft drugs are designed to be active in vitro but

lose their activity through further metabolism after exerting therapeutic effects at the target site.

16.3 Traditional Lead Compound Optimization

This section introduces traditional lead compound optimization and traditional methods from two perspectives.

16.3.1 Replacement Using Bioelectronic Isosteres

Langmuir proposed that as long as molecules, atoms, or groups (ions) have the same number of atoms and electrons, and their electron arrangement is also the same, then these molecules, atoms, or groups (ions) are called isoelectronic. Isoelectronic substances have similar physical and chemical properties. There are also some equivalent concepts, such as the "ring equivalent part" proposed by Hinsberg. The concept of "equivalence", that is, the equivalent parts of an aromatic ring can be mutually substituted without significantly altering its physical and chemical properties. Hückel expanded this equivalence theory, suggesting that they be regarded as equivalent substituents.

With the passage of time, the concept of electronic isosteres has been continuously expanded and refined. In 1979, Thornber summarized and generalized the concept of electronic isosteres, stating that functional groups or molecules with similar biological effects, physical and chemical properties can all be referred to as electronic isosteres.

16.3.2 Classification of Bioelectronic Isosteres

The application and scope of the bioisosteric principle have been gradually broadened. Bioisosteres are classified into two categories: classical and non-classical bioisosteres. Classical bioisosteres can generally be divided into four types: monovalent, divalent, trivalent, and tetravalent.

Monovalent bioelectronic isosteres: Halogens are a frequently used type of bioelectronic isosteres, which can influence the charge distribution of drugs and enhance the electrostatic binding with receptors; introducing halogens onto a benzene ring can increase the lipophilicity of the molecule. Substituting F, Cl, Br, and I for each other can produce compounds with similar or enhanced activities and the same effects. For instance, in the anti-estrogen drug tamoxifen shown in Fig. 16.1a, the benzene ring is replaced by a methyl group to form the compound in Fig. 16.1b, and

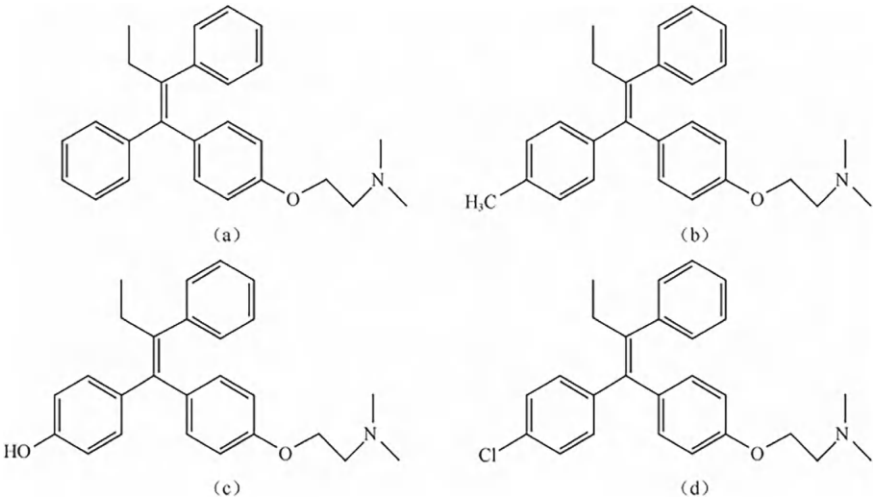


Fig. 16.1 The process of tamoxifen optimization

further substitution of $–CH_3$ with –OH and –Cl results in the compounds shown in Fig. 16.1c, d, respectively.

Divalent bioisosteres: Divalent atoms or groups mainly include O, S, NH, etc. Their bond angles are similar, but their hydrophobicities vary greatly. After substitution, changes will occur in hydrophobicity, stereochemistry, and electrical properties. For example, when –S– and –NH– are used to replace –O– in the ester group of the local anesthetic procaine, tetracaine, and procaineamide can be obtained. Among them, the local anesthetic effect of tetracaine is more than twice that of procaine, while the local anesthetic effect of procaineamide is only 1% of that of procaine. Therefore, it is currently mostly used to treat arrhythmia. This shows the impact of bioisosteric substitution on pharmacological effects.

Trivalent bioelectronic isosteres: Trivalent bioelectronic isosteres are a commonly used type of isostere in new drug design. The trivalent groups mainly involve the interchange of =CH–and =N–. For instance, pilocarpine is a cholinergic M receptor agonist. However, its lactone structure makes it highly susceptible to hydrolysis and instability. By replacing the =CH– on the pilocarpine ring with the isostere =N–, it transforms into a carbamic acid lactone structure. Carbamic acid lactones have better stability than formate esters, and this modification enhances the drug's stability.

Tetra-valent bioelectronic isosteres and ring system isosteres: The main tetravalent bioelectronic isosteric groups are Si and C. In the study of bioelectronic isosteres, due to significant differences between C and Si atoms in terms of atomic size, electronegativity, and lipophilicity, their application is not very widespread. Ring system isosteres mainly involve the substitution of aromatic substances.

Non-classical electronic isosteres have significant differences from classical electronic isosteres in terms of electronic and steric properties. Even if the number of atoms, electrons, and structures of the groups are different, as long as there are

similarities in certain key properties and they can form similar biological activities, they can be called electronic isosteres, that is, non-classical bioelectronic isosteres. Commonly substitutable non-classical bioelectronic isosteres include –CH=, –S–, –NH–, and –O–. Further classification can be made into two types: cyclic and acyclic substituents and exchangeable groups.

Ring and non-ring structures and conformational restrictions. A classic example is that diethylstilbestrol and estradiol are ring and non-ring bioelectronic isosteres. The double bond of diethylstilbestrol plays a key role in the correct spatial distribution and orientation of the phenolic hydroxyl group and the vinyl group at the receptor site. In terms of exchangeable groups, there are also corresponding applications. For instance, amino benzenesulfonic acid and aminobenzoic acid have been confirmed to be similar in electronic distribution and configuration, and they also share similarities in physical and chemical properties.

16.4 Computer-Aided Optimization of Lead Compounds

In addition to the traditional lead compound optimization methods, with the development of computer technology, computer-aided lead compound optimization has gradually been integrated into daily research. Common computer-aided lead compound optimization methods include QSAR and scaffold hopping, etc.

16.4.1 QSAR in Lead Optimization

QSAR specifically represents the relationship between the molecular structure of compounds and their biological activities, and combines theoretical calculation formulas with various data analysis methods to explore the quantitative relationship between chemical structure and biological effects. A relatively effective QSAR model can predict various properties of unknown compounds and identify the constituent elements that determine the properties of compounds, thereby understanding the different impacts of changes in the microscopic structure of compounds on their macroscopic properties. QSAR is not only applied in the search for lead compounds but also plays an important role in the optimization process of lead compounds.

The process of QSAR research and development mainly includes: data collection; improving the internal structure of the molecule and optimizing the energy; Molecular description; calculating symbols; Construct feature selection model; Test the model. The main process of QSAR research and development is shown in Fig. 16.2.

(1) Data collection. To ensure the validity of the subsequent model establishment, it is essential to guarantee the accuracy of the obtained data. Generally, data have three main sources: the first is from experiments; the second is from papers published in various scientific research journals; the third is from various

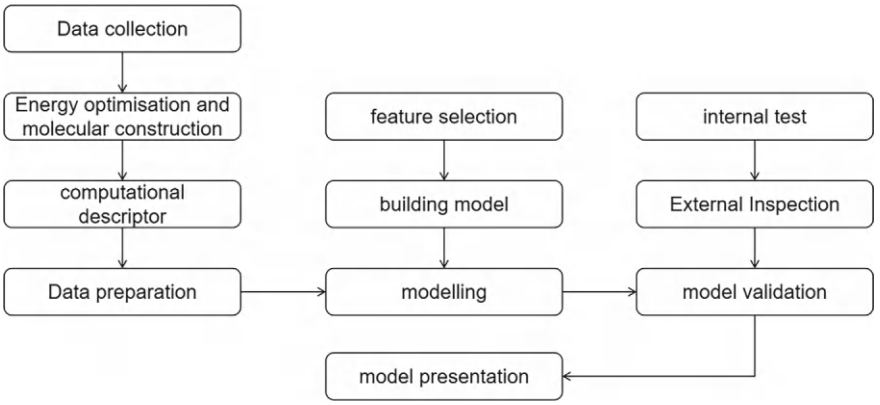


Fig. 16.2 Main process of QSAR research and development

database systems. The ideal data are those obtained under the same experimental conditions. The data were measured under certain conditions or by multiple scholars in the same laboratory. Among them, the activity data are preferably obtained through experiments using a standard sample that is widely recognized in the current academic community, as such data have smaller experimental deviations and higher quality. Additionally, the number of samples collected for experimental data is usually no less than 10.

(2) Molecular construction and energy optimization. Molecular structures can be drawn using software such as Chem Office or downloaded from online databases like PubChem and Zinc. After establishing the molecular structure, it is necessary to adjust it to ensure the lowest energy structure is obtained. The conventional approach to achieving the lowest energy structure is to first perform basic geometry optimization using molecular mechanics methods, followed by further optimization using quantum mechanics methods.

(3) Calculation of molecular descriptors. Molecular descriptors are numerical vectors calculated directly from the molecular structure using specific algorithms and can be used to describe the structural information of compounds. A particularly important step in molecular feature extraction is the calculation of molecular descriptors from the chemical structure. Common methods for characterizing molecular structure and generating molecular descriptors include the group contribution method and the topological index method. Both of these methods can obtain the required descriptor by analyzing the structural features of the molecule. Using molecular descriptors (i.e., parameters based on molecular structure) to characterize the structure of molecules can successfully achieve the construction of QSAR models. Molecular descriptors are indicators that measure certain properties of a molecule. The calculation of molecular descriptors is the foundation and prerequisite for obtaining a relatively efficient QSAR.

Since there are already more than 5,000 types of descriptors to date, it is necessary to select the most appropriate descriptors based on the research object in the study.

(4) Feature selection. Due to the excessive number of current molecular descriptors, feature selection is a crucial step. If too few descriptors are selected, the main structural features of compound activity cannot be fully covered. However, if too many descriptors are chosen, data redundancy will occur, leading to overfitting. Therefore, feature selection is necessary. Common feature selection methods include principal component analysis, heuristic methods, and genetic algorithms.

(5) Modeling methods. In QSAR, mathematical methods such as multiple linear regression and partial least squares are generally used to build a model that can quantify molecular properties based on chemometrics methods.

(6) Model validation. Model validation mainly consists of two aspects: one is internal validation, which examines the model's fitting ability and robustness; the other is external validation, which assesses the model's predictive power. The coefficient of determination R^2 is the most commonly used indicator to test the model's fitting ability, and its value in a regression model determines the goodness of fit. The cross-validation correlation coefficient q^2 is a coefficient used to evaluate the model's predictive ability from the perspective of internal validation.

16.4.2 Skeletal Transition

Scaffold hopping, also known as lead hopping, is a strategy for discovering compounds with novel structures. The scaffold hopping approach starts with an active compound and modifies the core structure of the molecule to obtain a new chemical structure. The main goal is to find compounds with similar activities but different core structures. In addition to activity, other molecular properties should also be taken into consideration. Therefore, the objective of scaffold hopping is to identify structurally diverse compounds that are similar in terms of activity and molecular properties.

There are two distinct features of skeleton transition: first, the new compound and the parent compound have different parent structures; second, the new compound and the parent compound have similar properties. These two requirements obviously conflict with the principle of structural similarity, that is, compounds with the same chemical structure generally have similar physicochemical properties. This can be explained as follows: although the ligands belong to different chemical types, there may be specific chemical structural similarities among the ligands combined in one pocket, such as similar shapes and the same potential surface.

As shown in Fig. 16.3, by modifying the skeleton of compound A, compound B was obtained. The overall difference between the newly obtained compound B and the original compound A is not significant; only a small part was modified. However,

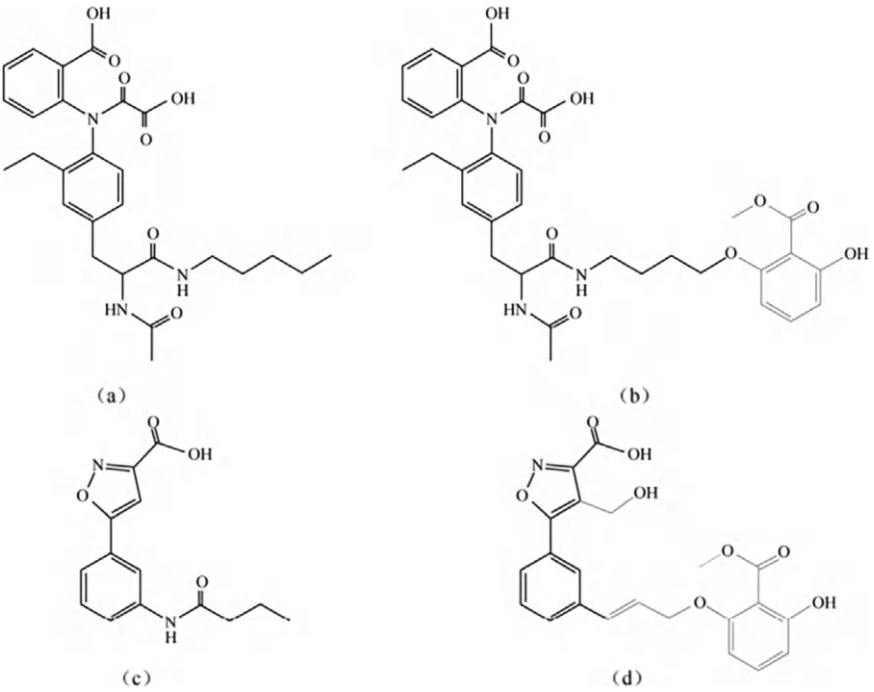


Fig. 16.3 Skeletal transition of compounds

at this point, the k_i value changed from the original 1.2–0.018 μM. Another example is that by modifying the skeleton of compound C, compound D was obtained. Two parts of the skeleton of compound C were modified, and the k_i value changed from the original 148–0.92 μM, showing a significant improvement.

16.5 Introduction to De Novo Molecular Generation

16.5.1 What Is De Novo Molecular Generation

Molecular de novo generation is a process that relies on algorithms to automatically propose new chemical structures in the best way to meet the required molecular characteristics [9]. In drug discovery, the target molecular characteristics are aimed at achieving the expected ideal biological effects while maintaining pharmacokinetic properties within an acceptable range. Due to the growing popularity of generative models in artificial intelligence, molecular de novo generation is also known as generative chemistry.

Traditionally, virtual screening [10] is used to identify molecules that can exhibit the desired experimental results. An important difference between virtual screening and de novo design lies in the source of the molecules: in virtual screening, the structures are a priori, while in de novo design, we seek to generate the structures to be evaluated. Because the amount of information in the chemical space that encompasses all possible molecules is obviously immense and impossible to fully explore, although the scale of current virtual screening databases is becoming increasingly large, with many databases containing over a billion molecules, these databases still only represent a very small part of the chemical space. Moreover, when large-scale compound databases need to be considered, the evaluation methods may sacrifice the effectiveness of the predictions.

The task of molecule generation: By using de novo design to generate compounds in a targeted manner, medicinal chemists hope to traverse the chemical space more efficiently and obtain the best molecules. Additionally, for a given target, there may be many acceptable regions in the chemical space, and one of the tasks of molecular design methods is to find solutions that balance global optimization and the utilization of local minima. De novo design methods are usually evaluated based on their performance in independent tasks, such as calculating the octanol–water partition coefficient. Although these goals are insignificant for demonstrating the optimizer's ability to generate target molecules, they are not effective in capturing the complexity of real-world drug discovery. In contrast, another way to evaluate de novo design methods is to prove their practicality through experiments.

The benchmark for molecular generation evaluation: Establishing a benchmark is particularly crucial for the continuous measurement of the progress of automated methods for generating chemical structures one step at a time. The release and availability of the benchmark standardized the assessment of the de novo design method from an electronic perspective. The molecular set benchmark includes a set of distribution learning tasks, as well as metrics for molecular validity and uniqueness. The purpose of the distribution learning tasks is to measure the structural diversity and relevance of compounds by comparing the generated chemical structures with known chemical structures. For example, the benchmark model for molecular generation, GuacaMol, in addition to distribution benchmark points, also includes a set of more practical goal-oriented tasks and feature scoring methods, with scoring functions such as physicochemical scoring functions, similarity scoring functions, and heterogeneity scoring functions. Using this model to optimize the generated molecules, a method better than virtual screening was established.

The input for molecule generation: The computational methods for evaluating chemical structures rely on a molecular representation that meets expectations, that is, the form of the molecular structure that subsequent algorithms can observe. Molecular characterization is a broad topic. For instance, these methods can encode the presence or absence of functional groups, represent molecules as topological graphs, or include 3D information describing bond angles. In de novo design methods, common molecular representations are text-based, such as the Simplified Molecular Input Line Entry System [11] (SMILES), and graph-based, where the molecule generator can

explicitly operate on the molecular topology. Text-based methods benefit from extensive research in natural language processing (NLP), while graph-based methods more naturally reflect the representation of molecular structures. Additionally, the choice of molecular characterization has other implications, such as whether the representation is discrete, continuous, or reversible. Recent explorations of de novo design methods have mainly focused on molecular characterization from the perspective of generative model architectures.

Molecular generation optimization: In the de novo generation of molecules, we also need to optimize the molecules. There are two methods for molecular optimization, namely gradient-free molecular optimization and gradient-based molecular optimization. Different methods have their own advantages and disadvantages. Molecules can be represented by encoding each atom and chemical bond in them, or by maintaining the representation form of their substructures and connectivity unchanged. For example, a phenyl group with substitution patterns can be regarded as a single group. Or the target molecule can be considered as the product of reactants and reaction conditions, and then the reaction is encoded.

16.5.2 Classification of Molecular Generation

There are various methods for molecular generation design, such as atom-based, fragment-based, and reaction-based molecular generation design. In practice, each of these methods—atom-based, fragment-based, and reaction-based—has distinct advantages and disadvantages. However, many methods blur the boundaries between these classifications.

Gradient-based molecular optimization is an atom-based molecular generation model that utilizes SMILES as the molecular representation. Given that SMILES is a text-based representation, it can leverage deep learning architectures suitable for sequences, such as recurrent neural networks [12], for molecular generation. By training on large molecular structure corpora, the generation model can perform prior learning, thereby encapsulating the grammar and syntax of valid SMILES. Early studies employed transfer learning to bias generation towards the chemical space of interest, while it is now common to combine the generation task with reinforcement learning algorithms, which learn to traverse the space to obtain more optimized molecules.

A classic example of gradient-free molecular optimization in de novo design based on atoms is the graph-based genetic algorithm, which uses reaction intelligence to perform mutation and crossover on a set of data, while the natural selection program ensures that the most optimized molecules are maintained in the population.

In gradient-free molecular optimization, the fragment-based de novo molecular generation method restricts the generation of new compounds, making them contain only previously existing fragments, the relevant substructures of knowledge, a fragment library containing one or more atoms can be used to construct new

molecules. MOARF [13] is a fragment-based de novo design method that uses a set of retrosynthetic cleavage rules and evolutionary algorithms.

In gradient-based molecular optimization, although the generation models based on pre-trained atoms have strong prior knowledge of substructures in the training data, they can still modify each atom in the molecule independently. Although the reinforcement learning models have flexibility and strong expressive power, and also expand the coverage of the chemical space, the ligand-based methods use simpler molecular representations to limit the search space. An early example of a simple molecular representation is the simplified graph, which was originally developed to represent Markush structures in patents. Later, researchers at GlaxoSmithKline reported the use of a sequence-to-sequence model that can learn the conversion between simplified graphs and SMILES. Jin et al. proposed a model called JT-VAE, which is a two-step generation process. In this process, the first step constructs a connection tree very similar to the simplified graph to represent the composition of substructures in the molecule, and the second step uses a graph message passing network to decode the final molecular structure.

Among the reaction-based molecular optimization methods, the most practical strategy for de novo design is to conduct forward reactions with the aid of computers. Recently developed AutoGrow4 [14] utilizes genetic algorithms and a reaction library, which mainly consists of organic reactions of the variant molecules in the population. However, there is a notable drawback to these reactions: they match the reaction handles but do not take into account other reactive groups in the molecule, which may affect the reactions in reality.

In gradient molecular optimization, ChemBO represents a stochastic structure, which randomly generates candidate structures by randomly selecting reactants and conditions, and then evaluates their performance. The entire operation process generates molecules through multi-step chemical synthesis.

16.6 De Novo Molecular Generation

16.6.1 Background of De Novo Molecular Generation

The de novo generation of molecules is confronted with the challenges of traversing a vast chemical space and the discontinuity in the field of drug optimization. As a result, the early stage of molecular generation consumes a considerable amount of time and human resources, and the outcome may not meet expectations. With the enhancement of computing power and the advancement of theories, deep learning methods have provided an effective approach for automatically learning tasks and extracting relevant features in molecular generation. Although computing capabilities have rapidly improved in modern times, the number of drug-like molecules varies from $10^{\wedge 23}$ to $10^{\wedge 180}$ depending on different estimation methods. Such a large number of molecules

still requires significant investment of time and funds to discover potential candidate compounds.

In early studies, the molecular generation task was treated as string generation, represented in a textual form, with a notable example being the SMILES generation model. However, the SMILES generation model has two drawbacks: one is that different SMILES can represent the same molecule, and the other is that it is not easy to express basic chemical properties. With the development of graph neural network algorithms, methods based on molecular graphs have shown better performance than string-based approaches. Additionally, traditional graph-based generation models generate molecular graphs in the form of nodes and edges. However, point-by-point generation based on graphs is not an ideal method for molecular generation, as generating molecules one atom at a time may result in chemically invalid intermediates. Some methods can generate molecular graphs by connecting trees and nodes within the trees. Subgraphs are used to solve these problems. Common generative architectures include recurrent neural networks (RNN), autoencoders (AE), variational autoencoders (VAE) [15], adversarial autoencoders (AAE), and generative adversarial networks (GAN) [16], etc. Among them, VAE and GAN are widely used to train molecular generation models. The generative model based on VAE consists of an encoder and a decoder. The encoder is responsible for rendering molecules in a continuous space, while the decoder can map the optimized continuous representation from the space to molecules with better performance. GAN is composed of a generator and a discriminator and is a useful tool for de novo drug design. The generator generates new data based on the learned probability distribution to deceive the discriminator, whose main function is to distinguish the real data from the generated data of the generator. The generator and the discriminator evolve mutually during the training process. Generally, GANs applied to de novo molecular generation are used in combination with other models. For example, the convolutional generative adversarial network (DCGAN) [17] applies convolutional neural networks to GAN to train generative models for de novo drug design with the desired characteristics.

The commonly used molecular generation model architectures mainly include the following five: Recurrent Neural Network (RNN), Variational Auto Encoder (VAE), Generative Adversarial Network (GAN), and generative glow-based model, etc. Each model has its own advantages and limitations due to its architectural features.

Figure 16.4 shows a schematic diagram of molecule generation based on a recurrent neural network. Recurrent neural networks perform significantly well for data with temporal dependencies. They can remember the information from previous input data and apply it to the current output calculation. Therefore, to achieve the above results, it is required that the nodes between the hidden layers of the recurrent neural network are connected rather than unconnected; that is, the input of the hidden layer consists of two parts—the output of the input layer and the output of the hidden layer at the previous moment.

The schematic diagram of molecule generation based on GAN is shown in Fig. 16.5. GAN consists of two parts: the generator and the discriminator. When dealing with real data, it does not directly calculate the probability distribution of the

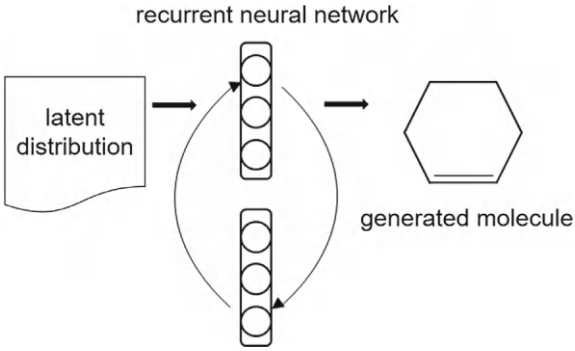


Fig. 16.4 Schematic diagram of molecule generation based on a recurrent neural network

real data but uses the adversarial approach between the generator and the discriminator to fit the probability distribution of the real data. Therefore, it can save a lot of time and computational costs.

The most significant difference between flow models and GANs is that when faced with real data, flow models choose to calculate the probability distribution of the real data, that is, they can transform distribution A into the real data's distribution B, and this transformation is usually reversible. As shown in Fig. 16.6, flow models can not only transform distribution A into the real distribution B but also convert the real distribution B back into distribution A. However, this transformation is subject to a restriction, which is that the data dimensions of the two distributions must be consistent.

The variational autoencoder (as shown in Fig. 16.7) is a combination of variational inference and neural networks, and is thus typically applied in complex scenarios for inference and as a generative model for generating continuous data. The variational autoencoder can simultaneously train the generative model and the inference model, and has many advantages in both inference and generation. For instance, in inference problems, it is fast to train and has low cost; in generation, it is fast to train while maintaining high stability and diversity.

Having mastered the basic algorithmic tools, the next step is to apply them. The first problem to be solved is to convert the chemical structures that are simple to

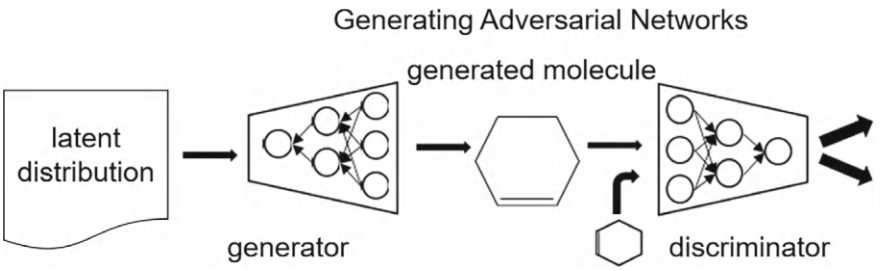


Fig. 16.5 Schematic diagram of molecule generation based on GAN

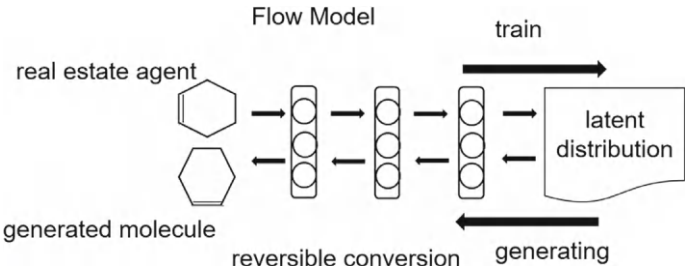


Fig. 16.6 Flow model

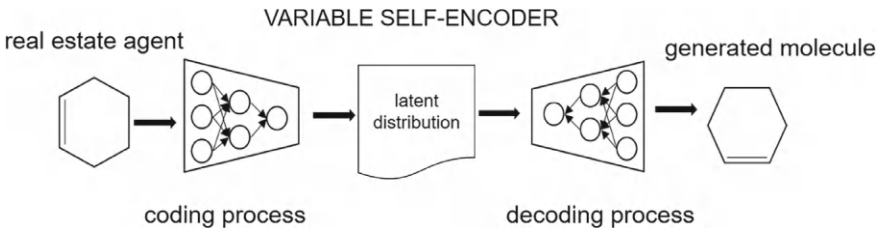


Fig. 16.7 Variational autoencoder schematic

humans into information that can be recognized by computers. Only when this condition is met can we use the mature algorithms at the bottom layer to train models and achieve the expected results. Currently, various methods of molecular characterization of multiple compounds are used for computer recognition of chemical molecular structures and their features, thus giving rise to multiple levels of molecular generation methods.

16.6.2 De Novo Molecular Generation Models

Molecular de novo generation models can be classified into one-dimensional molecular generation models, two-dimensional molecular generation models, and three-dimensional molecular generation models based on the dimension of molecular structure.

The most common one-dimensional molecular generation model is the chemical molecular descriptor, which represents chemical structures as strings. The advantage of RNN models lies in their ability to process sequence attributes, such as natural language recognition and music generation. The combination of the two enables the RNN model to learn the probability of specific atomic characters appearing at each position in the SMILES sequence through the training set, thereby obtaining the distribution rules of chemical structures and chemical space, which can guide the operation of character replacement in existing strings or the generation of new strings.

This is the working mode of one-dimensional generation models. One-dimensional molecular generation models can also be implemented using VAE networks, which control the structure of molecules by manipulating hidden variables. Comparisons have shown that the hidden representations obtained through VAE training have a high correlation with the structure.

Currently, there is a model for generating molecules using SMILES strings-BIMODAL [18]. Since the input data is in the form of text-based SMILES strings, many experimental results have shown that RNNs perform best in sequence-based methods and in tasks that match the distribution of structural and biological feature training data. The general form of RNNs is a forward data reading method, that is, training, reading, and generating SMILES from left to right. Meanwhile, SMILES can start from non-hydrogen atoms and be generated in any direction. Its non-uniqueness and undirected nature are the prerequisites for a new way of bidirectional sequence generation, that is, reading and generating SMILES in both forward and backward directions. Combining two proposed bidirectional RNNs—the synchronous bidirectional RNN (FB-RNN) and the neural autoregressive distribution estimator (NADE)—results in the BIMODAL model. You can enhance the effect of molecule generation by adjusting the parameters in Table 16.1.

In addition to the RNN model, variations of the GAN model can also be applied to the sequence-based method SMILES. The modified GAN model is called SMILES-MaskGAN. This model is derived from the MaskGAN architecture and, in addition to the traditional GAN generator and discriminator, it adds an evaluator. All three modules have the same architecture, which is a seq2seq with an attention mechanism. The attention mechanism consists of an encoding and a decoding module. During generation, the vector enters the generator for encoding first, and then decoding to obtain the generated molecule. The discriminator and evaluator follow the same process. In GANs, the key lies in the update of the generator's parameters. SMILES-MaskGAN uses the evaluator to update the generator's parameters, reducing the high variance of gradient updates to accelerate the convergence of the generator, thereby improving the efficiency and effectiveness of the model.

Given the limitations of one-dimensional generative models, we need a more expressive molecular model. Two-dimensional molecular generative models have driven research towards more expressive "molecular graph" methods. A molecular graph is a labeled graph composed of points corresponding to atoms and edges corresponding to chemical bonds, and they are connected to a set through certain connection relationships. Graph generative models have unique advantages in the

Table 16.1 Parameter adjustment of BIMODAL

Parametric	Adapt	Final results
Starting position	Secure	Ability to enhance molecular novelty
Data enhancement	Reinforce	Increased molecular uniqueness and novelty, improved modelling
Network size	Rise	Network size is proportional to model performance

field of molecular graph generation and focus on solving the following two problems: the first problem is to design neural network architectures that can directly handle graph data structures; the second problem is to design the generation process of molecular graphs.

The basic idea of two-dimensional molecular generation models is to generate new skeletons by splicing together building units based on fragments. VAE can be used to encode molecules into several building units and the connection methods of substructure nodes. Through the training set, the characteristics and occurrence frequency of the building units, as well as the splicing rules of the nodes, can be learned. Finally, the newly assembled substructure combinations are decoded by the decoder, and a brand-new molecular graph is obtained.

One of the classic models for de novo molecular generation based on images is also a variation of GAN, namely the MolGAN model. This model directly generates small molecule graph structures by combining GAN with reinforcement learning strategies. As shown in Fig. 16.8, the generator takes a random noise as input and implements it through a multi-layer perceptron, while the discriminator is implemented by a graph neural network and obtains the graph vector through the aggregation of node vectors as the input of the discriminator.

The process of MolGAN begins with sampling from a prior distribution and feeding it into the generator. The generator outputs two matrices in total. One is the adjacency matrix $\boldsymbol{A}$, which contains information about the types of edges; the other is the annotation matrix $\boldsymbol{X}$, representing the atomic types corresponding to each node of the molecule. Both $\boldsymbol{A}$ and $\boldsymbol{X}$ are continuous and dense matrices, with values ranging between 0 and 1, representing probabilities. After classification sampling, discrete and sparse $\boldsymbol{A}^*$ and $\boldsymbol{X}^*$ matrices are obtained. Then, the generated image is input into the discriminator to determine if the compound is reasonable. At the same time, it is

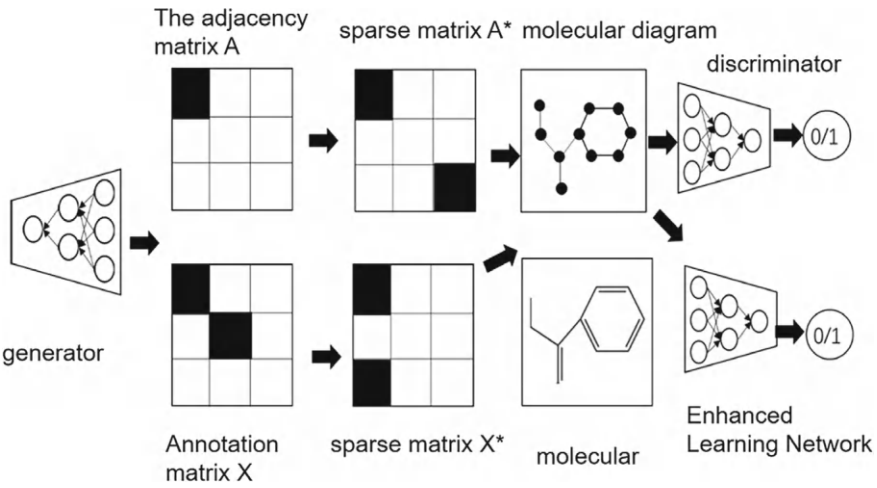


Fig. 16.8 Schematic diagram of MolGAN

also input into the reinforcement learning network to evaluate whether the compound possesses certain desired properties, such as improved biological activity.

Both of the aforementioned molecular generation modes focus on extracting the structural features of ligands. Considering the three-dimensional conformations of both the target pocket and the chemical molecules for molecular design represents a shape-based three-dimensional molecular generation model. The application of GAN can generate three-dimensional ligand structures complementary to the binding pocket. On one hand, the binding pocket is input and encoded as a hidden vector in the form of a graph; on the other hand, the network is trained to generate ligand shapes that fit the pocket structure using the hidden vector information. For instance, DeepligBuilder [19] can generate three-dimensional ligand molecules within the binding sites of target proteins.

16.6.3 Challenges in De Novo Molecular Generation

As artificial intelligence and machine learning play an increasingly significant role in drug discovery and can explore the vast chemical space more effectively than virtual screening or human experts, there are still many opportunities to improve de novo molecular design models. Current algorithms also have issues with the instability, reactivity, manipulability, and synthetic feasibility of the generated molecules. Developing reliable generative model algorithms, optimizing methods for three-dimensional molecular characterization, and enhancing the interpretability of generative models will be more conducive to generating drug molecules with specific functions.

The biggest challenge in the field of de novo molecular generation is how to evaluate whether our generators and optimization objectives are useful for drug design. Most of the methods are still black-box models. Although there are some scoring functions for evaluating generated molecules, these functions mainly score the physicochemical properties of ligand molecules and the rationality of the molecules. How to better evaluate the biological activity of molecules still needs to be improved.

References

1. Prokai, L., Prokai-Tatrai, K., Zharikova, A., Li, X., Rocca, J.R.: Combinatorial lead optimization of a neuropeptide FF antagonist. J. Med. Chem. **44**, 1623–1626 (2001)
2. Wen, Z., et al.: Prodrug-based nano-drug delivery system for co-encapsulate paclitaxel and carboplatin for lung cancer treatment. Drug Delivery **23**, 2575–2580 (2016)
3. Wang, H., et al.: Synthesis and evaluation of an injectable everolimus prodrug. Bioorg. Med. Chem. Lett. **27**, 1175–1178 (2017)
4. Felix, K., Müller, I.A., Claudia, R., André, W.: Prodrug strategies in anticancer chemotherapy. ChemMedChem **3**, 20–53 (2008)
5. Xianghua, W., et al.: A H_2S-triggered two-photon ratiometric fluorescent theranostic prodrug for bio-imaging. Chin. Chem. Lett. **32**, 2380–2384 (2021)

6. Marder, S.R., et al.: Fluphenazine plasma level monitoring for patients receiving fluphenazine decanoate. Schizophr. Res. **53**, 25–30 (2002)
7. Ting, Z., et al.: Self-assembled nanoparticles of amphiphilic twin drug from floxuridine and bendamustine for cancer therapy. Mol. Pharm. **12**, 2328–2336 (2015)
8. Bell, M., et al.: Discovery of super soft-drug modulators of sphingosine-1-phosphate receptor 1. Bioorg. Med. Chem. Lett. **28**, 3255–3259 (2018)
9. Francesca, G., Gisbert, S.: De novo molecular design with chemical language models. Methods Mol. Biol. (Clifton, N.J.) **2390**, 207–232 (2022)
10. Haga, H., Kohei, I., Susumu, D.: Virtual screening techniques and current computational infrastructures. Curr. Pharmaceut. Des. **22**, 3576–3584 (2016)
11. Toropov, A.A., Emilio, B.: SMILES in QSPR/QSAR modeling: results and perspectives. Curr. Drug Discovery Technol. **4**, 77–116 (2007).
12. Vathsala, M.K., Holi, G.: RNN based machine translation and transliteration for Twitter data. Int. J. Speech Technol. **23**, 1–6 (2020)
13. Firth, N.C., Butrus, A., Nathan, B. & Julian, B.: MOARF, an integrated workflow for multi-objective optimization: implementation, synthesis, and biological evaluation. J. Chem. Inf. Model. **55**, 1169–1180 (2015)
14. Spiegel, J.O., Durrant, J.D.: AutoGrow4: an open-source genetic algorithm for de novo drug design and lead optimization. J. Cheminform. **12**, 168–178 (2020)
15. Han, K., et al.: Variational autoencoder: an unsupervised model for encoding and decoding fMRI activity in visual cortex. Neuroimage **198**, 125–136 (2019)
16. Mosser, L., Dubrule, O., Blunt, M.J.: Stochastic reconstruction of an oolitic limestone by generative adversarial networks. Transp. Porous Media **125**, 81–103 (2018)
17. Abdurrahman, Ö., Lale, Ö.: Supervised deep convolutional generative adversarial networks. Neurocomputing **449**, 389–398 (2021)
18. Francesca, G., Michael, M., Robin, L., Gisbert, S.: Bidirectional molecule generation with recurrent neural networks. J. Chem. Inf. Model. **60**, 1175–1183 (2020)
19. Yibo, L., Jianfeng, P., Luhua, L.: Structure-based de novo drug design using 3D deep generative models. Chem. Sci. **12**, 13664–13675 (2021)

Chapter 17
Protein Structure Prediction

Proteins are of vital importance to life. Understanding their structure can further enhance our comprehension of their functions. As a key component of living organisms, proteins play a crucial role in their daily activities. The study of protein structure and function is a major focus of research both at home and abroad. The basic unit of proteins is the amino acid, and there are 20 types of amino acids. These 20 amino acids form an amino acid sequence through dehydration synthesis using mRNA as a template, which constitutes the primary structure of a protein. Since different proteins have different amino acid sequences, all proteins can fold into specific three-dimensional structures based on their one-dimensional structure. Understanding the three-dimensional structure of proteins is fundamental to studying their biological functions and the mechanisms of their activity.

17.1 Protein Structure and Function

This section will briefly discuss protein structures and functions. The primary structure of a protein determines its secondary, tertiary and quaternary structures. Natural proteins often have their own biological activities.

17.1.1 Protein Structural Hierarchy

1. The primary structure of protein

There are 20 standard amino acids used by cells for the construction of protein substances. Amino acids contain one amino group and one carboxyl group, which enables individual amino acids to link up into long chains by forming peptide bonds, where a peptide bond refers to the amide group between the $–NH_2$ of one amino acid

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_17

and the COOH of another. Peptides are usually sequences of less than 50 amino acids, while proteins are composed of one or more polypeptide molecules. The carboxyl terminus or C-terminus refers to the end of a protein sequence or polypeptide with a free carboxyl group, and the amino terminus or N-terminus refers to the end of a polypeptide or protein sequence with a free α-amino group.

The primary structure of a protein refers to the sequence of amino acid residues that make up the protein's peptide chain and is the most fundamental structure of a protein. The sequence of genetic codes on genes can determine the primary structure of a protein, and various amino acids are linked through peptide bonds based on the sequence of genetic codes to form polypeptide chains. Therefore, the main bond in the structure of proteins is the peptide bond. Currently, the primary structures of over a thousand proteins have been determined, such as insulin, pancreatic ribonuclease, and proinsulin.

The amino acid sequence of the peptide chain can determine the structural characteristics of the biological activity of each protein. Since the composition of proteins, the 20 amino acids in the protein each have a unique side chain. Different substituents on the amino acid side chains make the amino acids structurally distinct. These side chains endow peptides or proteins with different chemical, physical and structural properties. When the side chain groups are combined in different sequence relationships, various spatial structures and protein molecules with different biological activities can be formed. According to the nature of the side chain substituents, amino acids can be classified into acidic amino acids, basic amino acids and neutral amino acids.

The polypeptide chains of protein molecules are not linearly extended but constantly fold and coil until they form a relatively stable spatial structure with certain characteristics. The integrity of the spatial structure determines the biological activity and physicochemical properties of the protein. However, the amino acid composition and sequence of the protein molecule alone cannot fully explain the biological activity and physicochemical properties of the protein molecule. Therefore, the primary structure of a protein is a one-dimensional structure without considering its spatial folding, while the spatial structure of a protein refers to its secondary, tertiary and quaternary structures.

2. The secondary structure of proteins

The linear extended chain segments of proteins or peptides have certain characteristic local structural conformations, which constitute the secondary structure of proteins. The formation of secondary structure mainly depends on hydrogen bonds. There are two types of secondary structures: α-helix and β-sheet. The α-helix is a right-handed helix. In the α-helix, the side chain substituents of amino acids extend outward, and the formation of hydrogen bonds makes this structure very stable, with the side chain substituents of amino acids located outside the N-H groups. In the β-sheet, hydrogen bonds form between peptide chains (between strands) rather than within peptide chains (within strands). The folded conformation is mainly composed of pairs of parallel or antiparallel strands. If the directions of the strands are the same, the two strands can be parallel; if the directions are different, the two strands are

antiparallel. The antiparallel β-sheet has a more regular secondary structure due to the more orderly arrangement of hydrogen bonds.

3. The tertiary structure of proteins

The tertiary structure of a protein is a stable and regular spatial structure formed by the further coiling or folding of the polypeptide chain on the basis of its secondary structure. The conformation of a protein is the complete three-dimensional structure composed of the regular secondary structures and other irregular peptide segments in the polypeptide chain. This is the tertiary structure of the protein, which involves the spatial relationships between amino acid residues that are far apart in the linear sequence.

The side-chain conformations form microdomains, which are hydrophobic and hydrophilic regions for proteins. Under physiological conditions, the hydrophobic side chains of neutral nonpolar amino acids are located inside the protein molecule, thus they are not affected by water. While acidic or basic amino acids' side chains are exposed to the exterior of the protein, allowing them to interact with the aqueous environment, acidic or basic side chains are hydrophilic and typically exposed on the protein surface. In addition, hydrogen bonds can form between different side chain groups, and these hydrogen bonds can align two parts of a chain in sequence. Salt bridges—the ionic interactions between amino acid side chains—play a certain stabilizing role in the tertiary structure of proteins.

4. The quaternary structure of proteins

Protein subunits are composed of multiple polypeptide chains. The quaternary structure of the protein with ID 4KQL in the PDB is shown in Fig. 17.1. A homodimer refers to the situation where the subunits are the same, while a heterodimer refers to the situation where the subunits are different. The quaternary structure of a protein refers to an aggregated protein complex. It is formed by the interaction and arrangement of protein subunits. The shape of the protein complex is stabilized by various interactions such as hydrogen bonds, disulfide bridges and salt bridges.

17.1.2 Functions of Proteins

Proteins are biopolymers of special status in living organisms. They are the basic building blocks of living things and play a significant role in many important biological processes. There are various types of proteins in living organisms, and they have many different functions in the body, such as providing mechanical support, acting as catalysts, transporting and storing other molecules, etc. [1].

The influence of protein structure on its function can be seen from the degree of change in protein structure. Any change in protein at all structural levels may cause it to lose certain functions. The function of a protein is directly related to its three-dimensional structure formed by folding according to a certain sequence. Proteins interact with each other or combine with other biological macromolecules to form

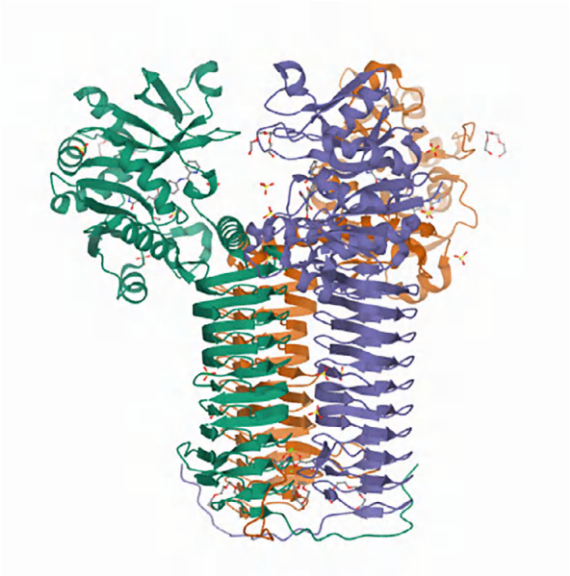


Fig. 17.1 The quaternary structure of proteins, illustrated by the protein with ID 4KQL in the PDB

complexes, and within these complexes, proteins can perform independent functions such as DNA replication or cell signal transduction. The most important biological function of proteins is to act as enzymes to catalyze various metabolic processes in the body, and they are also important structural components of organisms. The main functions of proteins are as follows [2].

(1) Enzymatic catalysis. Many reactions within living organisms can hardly take place without the catalytic action of enzymes, and most of the enzymes that have been discovered are proteins.
(2) Transport and storage of substances. The most typical example is the transport of oxygen by hemoglobin. Myoglobin transports oxygen in the muscles. Transferrin in the plasma carries iron and forms a complex with transferrin for storage when it reaches the liver.
(3) Immune protective function. Antibodies with immune functions are highly specific proteins that can recognize the cells of bacteria, viruses and other organisms and bind to foreign substances.

The properties of proteins are also highly variable. Some proteins are relatively rigid, while others are flexible. Rigid proteins may play a significant role in the cytoskeleton or connective tissue, while flexible proteins can act as springs or levers to assist the functions of other proteins. In addition, proteins have functions such as movement coordination, growth, or differentiation control. With the continuous development of science and technology, people's understanding of proteins will deepen. Not only can they conduct in-depth learning on the known functions of proteins, but they can also constantly discover new functions of proteins.

17.2 Introduction to Protein Folding

This section mainly introduces protein folding.

17.2.1 Protein Folding

An active protein molecule has a specific amino acid sequence and is also in a three-dimensional spatial structure. If the stability or integrity of the three-dimensional structure is disturbed by external forces, the biological activity will change. Some minor changes or damages may cause the complete loss of biological activity. According to the central dogma of molecular biology, the genetic information of life is transferred within the nucleus from DNA to RNA and then to proteins. DNA is transcribed into RNA, and the amino acid sequence of the protein polypeptide chain is translated on the ribosomes in the cytoplasm. In fact, what is synthesized is a nascent polypeptide chain rather than a protein. At this point, the nascent polypeptide chain needs to undergo a complex processing process, including chemical modifications such as disulfide bond formation and carboxylation of amino acid residues. The nascent polypeptide chain needs to fold to form a certain spatial structure, and each nascent chain needs to be transported to a specific location or secreted outside the cell to exert its function. Then, the nascent polypeptide chain needs to be transported, and the transport process often involves one or more transmembrane processes. However, fully folded proteins cannot cross the membrane structure. Therefore, when premature or excessive folding occurs, the protein needs to be unfolded first before being transported across the membrane to other specific biological functional sites, and then folded into a functional protein with a special spatial structure. The nascent peptide acquires biological functions by folding into a specific three-dimensional structure and matures into a functional protein molecule. From this, we can see that folding not only refers to the coiling and curling of the peptide chain in space but also includes the entire maturation process of the peptide chain, such as transmembrane transport and chemical modification [3].

The research on the kinetic control of protein folding is a further study and improvement of protein folding based on the thermodynamic hypothesis. For different proteins, their folding processes are also different, each with its own characteristics. Protein folding follows the ‘thermodynamic hypothesis’, that is, it can shift from a high-energy state to a low-energy state. During this process, protein folding is controlled by kinetics. In the folding reaction of protein polypeptide chains, thermodynamic control and kinetic control are unified. For small proteins with a simple structure, their folding is simple, and they can undergo reversible denaturation and renaturation under thermodynamic control.

17.2.2 Protein Folding Dynamics

1. The movement of protein molecules

The three-dimensional conformation of natural protein molecules is constantly changing in solution. The presence of temperature promotes the movement of atoms. Due to the interaction between various atomic groups within the molecule and the influence of solution molecules, the internal movement of protein molecules is quite complex. The basis of biological functions lies in the spatial structure of proteins, and the spatial structure among atoms, groups, secondary structures, and subunits of tertiary structures within protein molecules enables biological functions to be exerted. The internal movement of protein molecules follows certain patterns, which are closely related to their biological functions. To study protein molecules, we need to have a clear understanding of their structure, function, and interactions.

To study the movement of protein molecules from a theoretical and computational perspective, we first need to introduce the concept of potential energy landscape. Atoms have precise energy and structure in their ground state, and structural changes relative to the ground state can reflect excited states and corresponding energy level sets. As a physical system, proteins also have their structural space and corresponding energy. The total energy of the protein system is spread out in the space based on the three-dimensional structure of the molecule, forming the energy surface of the protein. However, the energy surface of proteins is extremely complex. Proteins in their natural state are at the global minima of the energy surface. The movement and conformational changes of protein molecules on the energy surface correspond to perturbations near the global minimum, or jumps from the global minimum to nearby local minima, or significant gravitational processes from equilibrium to non-equilibrium.

Atom-scale first-principles molecular dynamics simulations can model the real movement of protein molecules, but the huge computational load will make the basic time step of the simulation very small. However, a too small-time step in the calculation process is very inefficient for modern computing. It is difficult to accomplish. Therefore, regular pattern analysis can provide a computational simulation of protein movement.

2. Regular pattern analysis

Suppose that at temperatures not too high, a typical semi-empirical potential energy is expressed as Eq. (17.1).

$$E_P = \frac{1}{2}\sum_{bonds} k_b(b-b_0)^2 + \frac{1}{2}\sum_{\substack{bond\\angles}} k_\theta(\theta-\theta_0)^2 + \frac{1}{2}\sum_{\substack{dihedral\\angles}} k_\phi(1+\mathrm{Cos}(n\phi-\delta))$$

$$+ \sum_{\substack{nonbond \\ bonds}} \left\{ 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r} \right)^{12} - \left(\frac{\sigma_{ij}}{r} \right)^{6} \right] + \frac{q_i q_j}{\varepsilon r} \right\} \tag{17.1}$$

The first three terms in Eq. (17.1) describe the energy magnitudes of the bond length, bond angle and dihedral angle between the two atoms forming a chemical bond, while the last two terms represent the van der Waals force and electrostatic force respectively. The electrostatic force is a long-range force that acts on non-bonded atom pairs. There are different constants k in the formula, and the degree of difference varies according to the different valence bond interactions between atoms of different elements. The van der Waals force constant and the charges carried by each atom will also produce different results according to different interactions. Generally speaking, the force fields mainly used for biological macromolecules are AMBER, CHARMM, GROMOS, etc.[4].

Based on the existing potential function, we can select a set of generalized coordinates, at which point the potential function can be expressed as Eq. (17.2).

$$\begin{aligned} E_P &= \frac{1}{2} \sum_{ij}^{n} q_i F_{ij} q_j \\ F_{ij} &= \frac{\partial^2 E_P}{\partial q_i \partial q_j} \end{aligned} \tag{17.2}$$

Here, F_{ij} is the generalized force matrix, and q_i are generalized coordinates.

The first theory to conduct standard mode calculations on protein systems achieved considerable success. The model provided relatively good results when simulating the anharmonic motion of protein molecules without high temperatures. However, as experimental equipment has gradually improved, the three-dimensional molecular structure of protein systems can now be obtained using new technologies. Nevertheless, when simulating large molecular-weight protein polymer systems, applying traditional standard mode analysis imposes high demands on computational conditions, especially memory requirements.

17.3 Protein Structure Prediction Algorithms

This section will mainly introduce protein structure prediction algorithms.

17.3.1 Genetic Algorithm

Genetic algorithms are a kind of random search method based on the foundation of evolution and constantly evolving. They were first proposed by Professor J. Holland

of the United States in 1975. The problem to be solved is simulated as a biological evolution process. Through operations such as replication and mutation, the next generation of solutions is generated, and solutions with higher fitness function values are gradually increased while those with lower values are eliminated. This process continues until a solution with a very high fitness function value is obtained. The main features of genetic algorithms are that they can directly operate on structural objects, possess certain implicit parallelism and global optimization capabilities. However, in practical applications, they have some drawbacks such as poor local optimization ability, premature convergence and convergence issues.

The basic operation process of the genetic algorithm is as follows.

(1) Initialize the data. Set the counter of the evolutionary generation $s = 0$, set the maximum number of evolutionary generations S, and then randomly generate N individuals as the initial population $P(0)$.
(2) Individual evaluation. Calculate the fitness of each individual in the population $P(s)$.
(3) Selection operation. Its purpose is to directly pass on the optimized individuals to the next generation or to cross-pair them to generate new individuals and then pass them on to the next generation. Here, the selection operator is applied to the population and is based on the fitness evaluation of the individuals within the population.
(4) Crossover operation. The core function in genetic algorithms.
(5) Mutation operation. Alter the gene values at certain loci of the individual strings within the population. That is to say, after the operations of selection, crossover and mutation are performed on the population $P(s)$, the next generation population $P(s + 1)$ is obtained.
(6) Termination condition judgment. If $s = S$, the individual with the maximum fitness obtained during the evolutionary process is output as the optimal result, and then the calculation is terminated.

Genetic operations mainly include the following three points: selection, crossover and mutation. Among them, selection refers to choosing the best individuals from the population and eliminating the inferior ones. Common selection operators include fitness proportion method, random traversal sampling method and local selection, etc. Crossover means generating new individuals by replacing and recombining the partial structures of two parent individuals, at which point the search ability of the genetic algorithm is enhanced. Mutation refers to making changes to the gene values at certain loci on the individual strings in the population. According to the individual coding representation method, it can be divided into real value mutation and binary mutation. The reason for introducing mutation in the genetic algorithm is to endow it with local random search ability. The algorithm terminates when the fitness of the optimal individual reaches the given threshold.

17.3.2 Simulated Annealing Algorithm

The Simulated Annealing algorithm is based on the annealing principle of solids. When the temperature of a solid is high, its internal energy is relatively large, and the particles inside the solid move rapidly and disorderly. As the internal energy of the solid decreases, the disorderly movement gradually becomes orderly. When the solid is at room temperature, the internal energy reaches its minimum, and the particles are most stable. The Simulated Annealing algorithm starts from a certain temperature, which is the initial temperature. As the temperature parameter continuously decreases, the solution tends to be stable, but this solution is a local optimal solution. It uses the local optimal solution to search for the global optimal solution of the objective function. The Simulated Annealing algorithm has a strong local optimization ability, but its ability to control the entire search process is insufficient.

The process of the simulated annealing algorithm is as follows.

(1) Randomly select a unit t and give it a random displacement to calculate the energy change ΔEt it generates.
(2) If $\Delta Et_{\leq} 0$, this displacement can be taken as the starting point for the next change in the system state. If $\Delta E_t > 0$, then after the displacement, the probability of the system state is $P_t = \frac{1}{1+e^{-\Delta E_t/T}}$. Here, T represents the temperature, which is uniformly distributed within the range of (0,1). Generate a random number Q uniformly distributed in the range (0, 1).If $Q < Pt$, *th*en the changed state at this time will be the starting point for the next round.
(3) Then return to step (1) and continue the process until a balanced state is reached.

17.3.3 Homology Modeling Method

1. Basic idea

Homology modeling, also known as comparative modeling, predicts the structure of a protein based on its sequence homology with a known structure. This involves calculating the similarity between the target protein sequence and the template protein sequence, presenting the sequence similarity in the form of sequence alignment, and then constructing the spatial structure of the protein starting from the alignment.

2. Sequence alignment algorithm

Typical homology modeling methods include FFAS, ORFeus, SAM-T99, etc. In terms of protein sequence representation, initially, the residue sequence was used, followed by the sequence profile containing homologous sequence information, and then the Hidden Markov Model (HMM) and Markov Random Field (MRF) were adopted. Different sequence similarity calculation methods can lead to different protein sequences.

FFAS presents the target protein and the template protein in the form of sequence profiles and measures the similarity between two sequences by the differences

between sequence profiles. The advantage of this approach lies in its ability to identify the possibility of various amino acids and gaps occurring at each residue position [5].

ORFeus, on the other hand, incorporates predicted secondary structure information into the sequence alignment. That is, ORFeus first predicts the secondary structure propensity of each residue in the protein; then, in the sequence alignment, it takes into account the similarity between the predicted secondary structure of the target sequence and the actual secondary structure of the template sequence to enhance the accuracy of the sequence alignment. The advantage of sequence profile lies in its ability to consider various variations at each position. However, the existence of gap penalties leads the sequence profile alignment method to prefer 'block-like' alignments between the target sequence and the template sequence.

SAM-T99 represents the template protein sequence by using the sequence profile based on the Hidden Markov Model (HMM), and avoids the drawbacks of sequence profile alignment by assigning specific gap penalties to each position. We can also view it from another perspective: SAM-T99 establishes a generative model, and the optimal alignment between the target protein and the template can be transformed into a labeling problem of the Hidden Markov Model, which is then solved using the dynamic programming algorithm [6].

HHpred extends this problem by using sequence-profile hidden Markov models to simultaneously represent the template protein sequence and the target protein sequence; the optimal alignment problem between the template protein sequence and the target protein sequence is transformed into the optimal alignment problem between two hidden Markov models. To calculate the coupling between the two hidden Markov models, HHpred constructs a finite state automaton (FSA) containing multiple sets of 'paired states'; in each paired state, one state comes from the hidden Markov model of the target protein, and the other state comes from the hidden Markov model of the template protein. In each paired state, two residues are emitted simultaneously, as shown in Fig. 17.2. The similarity between the two hidden Markov models can be calculated by the probability that two models generate the same sequence of residues [7].

3. Summary of Homology Modeling Methods

Homology modeling methods are applicable to target proteins with homologous sequences in the template library. The differences among existing homology modeling methods lie in how sequence information is described. In fact, the continuous improvement of sequence information representation schemes is part of the development history of homology modeling methods. For instance, the earliest BLAST simply used amino acid sequence information; PSIBLAST and FFAS further depicted site-specific sequence information; SAMT99 employed sequence profile Hidden Markov Models to describe the sequence information of template proteins, allowing site-specific gap penalties; while HHpred established a sequence profile Hidden Markov Model for both the target protein sequence and the template protein sequence. It should be noted that the sequence features considered in homology modeling methods describe the local characteristics of residues and do not take into

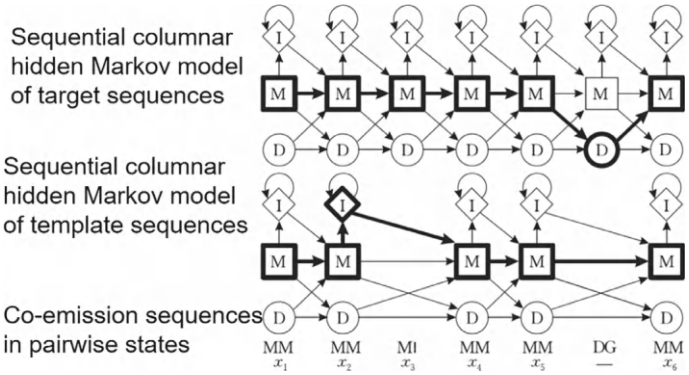


Fig. 17.2 Optimal alignment between sequence-type hidden markov models

account the associations between residues far from the sequence, thus enabling efficient calculation of sequence associations. Swiss-model, Modeller, and I-TASSER are relatively reliable and commonly used protein homology modeling methods.

17.4 Disruptive Developments in Protein Structure Prediction

At present, approximately 100,000 protein structures have been determined, which is only a tiny fraction of the billions of known protein sequences. The problem of predicting three-dimensional structures from amino acid sequences has been studied for over 50 years. Although many methods have made some progress, they still cannot achieve atomic-level accuracy, especially in the absence of homologous templates. In the CASP14 critical assessment competition for protein structure prediction in 2020, the AlphaFold2 proposed by the DeepMind team could predict protein structures with atomic-level accuracy even without homologous templates. AlphaFold2 integrates physical and biological knowledge about protein structures, multiple sequence alignments, and deep learning algorithms, ultimately achieving very high accuracy.

Traditional protein structure prediction methods require multiple sequence alignment based on protein sequences first, and then apply deep learning. The model obtains the distance map of the sequence and then predicts the structure of the sequence based on the distance map. The theoretical basis for this is the co-evolutionary information of proteins. The AlphaFold2 network uses the primary amino acid sequence and the alignment sequence of homologues as input to directly predict the three-dimensional coordinates of all heavy atoms of the given protein. The network structure diagram is shown in Fig. 17.3 and can be divided into four parts: the model input part, the recycling part, the Evoformer part, the structure module part, and the final output.

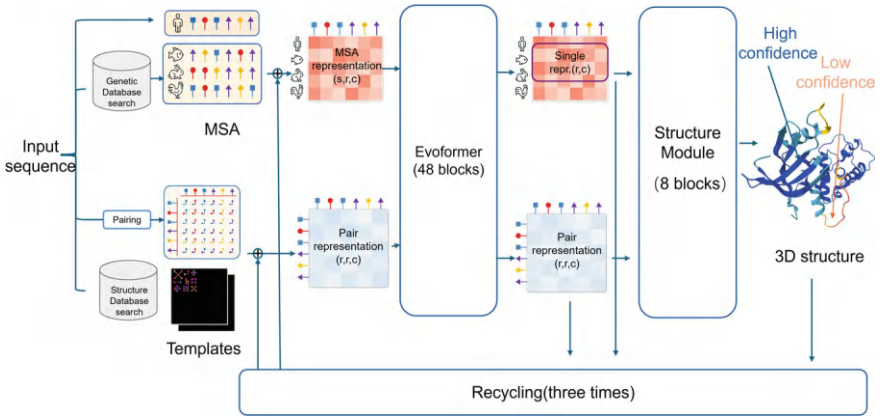


Fig. 17.3 Network structure of AlphaFold2

The AlphaFold2 network consists of two main parts. The first part is the backbone of the network, which processes the input through a series of repeated layers of a novel neural network block, which we call the Evoformer. This module generates an array (where N_{seq} represents the number of sequences and N_{res} represents the number of residues), representing a processed MSA, and an Nres array that can be used to represent residue pairs. The MSA representation is initialized with the original MSA, and the Evoformer module contains many novel attention-based and non-attention-based components. Following the backbone of the network is the second part, the structure module, which introduces an explicit three-dimensional structure in the form of rotations and translations for each residue of the protein (in a global rigid body frame). These representations are initialized in a trivial state, with all rotations set to be consistent and all positions set to the origin, but they rapidly develop and refine into highly accurate protein structures with precise atomic details. Key innovations in this part of the network include breaking the chain atomic structure to allow simultaneous local refinement of all parts of the structure, a new equilateral transformer that enables the network to implicitly reason about unrepresented side chain atoms, and a loss term that places significant weight on the correctness of the direction of pairwise evolutionary correlation residuals. Both within the structure module and throughout the entire network, we reinforce the concept of iterative refinement by repeatedly applying the final loss function to the output and recursively feeding the output back into the same module. Iterative refinement using the entire network (which we call "recycling", related to methods in computer vision) makes a significant contribution to accuracy, requiring only a small amount of additional training time [8].

At the same time as AlphaFold2 was published, the Baker team released an article on RosettaFold. This method employs a three-track neural network mechanism, focusing on extracting and integrating information about the primary amino acid sequence, the two-dimensional amino acid residue side-chain distances and

conformational maps, and the tertiary backbone coordinates of proteins. By adding multiple connections among the three, the entire neural network can simultaneously learn information at three levels. In this architecture, information flows back and forth between one-dimensional amino acid sequence information, two-dimensional distance maps, and three-dimensional coordinates, allowing the neural network to reason about the relationships within sequences, distances, and coordinates.

Meanwhile, the article points out that the main advancements of the AlphaFold2 method lie in the following five aspects: First, it starts from multiple sequence alignment (MSA) rather than from processing more features (such as the inverse covariance matrix derived from MSA); second, it uses the attention mechanism instead of two-dimensional convolution to better represent the interactions between residues far apart in the sequence; third, it adopts a dual-track network structure, iteratively converting and passing information between one-dimensional sequence-level and two-dimensional distance map-level; fourth, it uses an equivariant SE(3)-Transformer network to directly refine the atomic coordinates generated by the dual-track network (rather than previous methods that utilized two-dimensional distance maps); fifth, it employs end-to-end learning, optimizing all network parameters by backpropagating from the final generated three-dimensional coordinates to all network layers and back to the input sequence. Although there is still much room for improvement in the combination of protein structure prediction and deep learning methods, there are also certain limitations, such as a heavy reliance on existing structural information and the inability to achieve 100% accuracy.

While AlphaFold2 and early iterations of RosettaFold achieved distinct breakthroughs in predicting monomeric and multimeric protein structures, they were primarily optimized for protein monomers and complexes optimized for protein monomers and complexes, whereas modeling non-protein components required specialized extensions in modeling complex biological systems containing non-proteinaceous components—such as DNA, RNA, small molecule ligands, ions, and post-translational modifications. In response to this bottleneck, the field of structural biology prediction underwent a paradigm shift in 2024, transitioning from a singular focus on "protein folding" to the comprehensive simulation of "holo-biomolecular systems."

The network structure diagram is shown in Fig. 17.4. The release of AlphaFold 3 by DeepMind marks a definitive milestone in this evolution [9]. Architecturally, AlphaFold 3 diverges from the rotation-and-translation structural modules of its predecessors, instead integrating a Diffusion Network rooted in generative artificial intelligence. This network directly generates raw three-dimensional structures by progressively denoising Gaussian noise within atomic coordinates, thereby enabling the unified modeling of proteins, nucleic acids, and diverse chemical modifications. Notably, in the realm of protein–ligand interaction (PLI) prediction—a critical domain for pharmaceutical research—AlphaFold 3 has demonstrated precision far superior to traditional molecular docking methodologies, achieving high-fidelity complex modeling without the necessity for predefined binding pockets.

Concurrently, the introduction of RosettaFold All-Atom (RFAA) by David Baker's team has further bridged the dichotomy between biological macromolecules

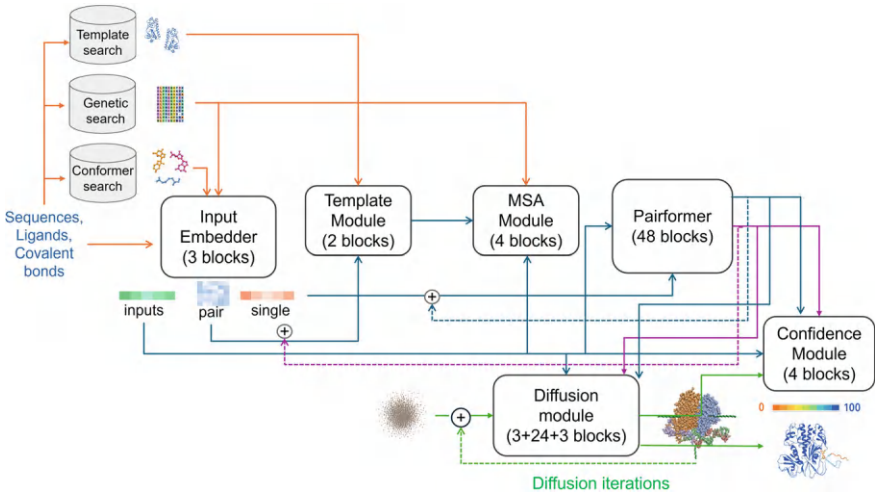


Fig. 17.4 Network structure of AlphaFold3

and chemical small molecules. By constructing a unified tensor representation space, RFAA integrates amino acids, nucleotides, and small molecules of arbitrary chemical structure into a singular prediction framework. Leveraging a fine-tuned diffusion model to refine full-atom structures, RFAA not only overcomes longstanding challenges in identifying metal ions and covalent modifications but also exhibits significant potential for de novo drug design, particularly in the inverse generation of binding molecules tailored to specific target structures.

This evolutionary stage signifies that the application of deep learning in structural biology has extended beyond backbone topology prediction to encompass the simulation of physicochemical interactions at the atomic level. Generative architectures based on diffusion models are now providing computational tools with enhanced physical realism, which are instrumental in elucidating the mechanistic underpinnings of cellular molecular machines and accelerating Structure-Based Drug Discovery (SBDD).

17.5 Drug Design Based on Protein Structure Prediction

Protein structure-based methods are helpful for predicting the binding mode and relative affinity of small molecules. After obtaining the structure of a protein, various structure-based drug design studies can be carried out. In recent years, computer-aided drug design methods have developed rapidly, and their accuracy has reached a level that can be used in daily practical applications in the drug discovery process. By conducting high-throughput docking of up to 10^6 small molecules and then scoring

based on implicit solvent force fields, micro-molar binders can be robustly identified using rigid protein targets. Molecular dynamics with explicit solvents is a low-throughput technique used to characterize flexible binding sites and accurately evaluate binding pathways, dynamics and thermodynamics.

17.5.1 Molecular Docking

Molecular docking refers to the process by which two or more molecules recognize each other through geometric and energy matching. As an important molecular simulation technique, it has a wide range of applications, such as virtual screening, drug development, binding site identification, protein–protein or protein-nucleic acid interactions, etc. It is particularly significant for drug design because if a drug molecule is to exert its pharmacological effect, it needs to bind to the target protein. During the binding process, the two molecules take appropriate measures to approach each other, cooperate and interact at necessary sites, and then undergo appropriate conformational adjustments to ultimately form a stable complex conformation. At this point, the correct position and orientation of the protein–ligand molecule can be determined through the conformation of the complex formed by molecular docking, allowing for the study of the conformations of the two molecules.

According to the molecular type, the methods can be classified into protein–ligand docking and protein–protein docking. Protein–ligand docking can be further divided into three categories based on the degree of molecular flexibility: rigid docking, semi-flexible docking, and flexible docking. Rigid docking refers to the situation where neither the protein nor the ligand molecule's conformation changes during the docking process. Semi-flexible docking means that the protein molecule's conformation remains unchanged during the docking process, but the ligand molecule's conformation can change within a certain range. Flexible docking indicates that both the protein and ligand molecules' conformations can change. In terms of simulation accuracy, flexible docking requires considering many variables during the calculation process, thus incurring a relatively high computational time cost. However, flexible docking can precisely describe the interaction between molecules.

Protein–protein docking methods can be classified into rigid docking and flexible docking based on the degree of molecular flexibility. Among them, Flexible docking can be further classified into three types based on the different levels of flexibility: side-chain flexibility, backbone flexibility, and domain flexibility. Side-chain flexibility takes into account the changes of side-chains during the docking process; backbone flexibility considers the changes of the backbone during the docking process; and domain flexibility accounts for the relative changes of large-scale domains during the docking process.

The principle of virtual screening is to assess the possibility of molecules in a library binding to a protein and to select those most likely to bind with the highest affinity. As mentioned above, the main challenge is not to identify a few nanomolar binders from a small molecule library, but to reduce the number of false positives in

the subset of compounds selected for in vitro validation. Few studies have systematically analyzed the success rate of docking activities, that is, the percentage of compounds correctly predicted to bind to the protein target. Although many papers report very good hit rates, the criteria for defining a hit are always subjective and depend on the study. The most stringent criterion is to consider only those hits confirmed by the crystal structure of the target-ligand complex as valid. In this case, it must be noted that even millimolar binders can have their crystal structure of the complex with the target protein determined. Moreover, obtaining the crystal structure of a complex with a potent ligand (such as nanomolar affinity) can be very difficult or even impossible, as the binding site may be occluded by crystal contacts or the ligand may not be accessible due to the tight packing of protein molecules in the crystal. The most commonly used hit rate criterion is based on the affinity measured in in vitro biochemical or biophysical experiments (typically KD or IC50 below 100 μM) or semi-quantitative data, such as ligand-based NMR spectroscopy. Such success cases should be treated with caution, as it is clear that the selection process often involves visual inspection and checking by users with significant expertise and may be biased towards scaffolds previously disclosed in the literature. To correctly measure the performance of different software suites, common standards should be introduced and human intervention should be minimized, but this is not simple given the complexity of the binding pose analysis and the associated costs of in vitro validation.

At present, molecular docking aims to obtain the best binding mode between molecules, which involves two main issues: one is how to determine the binding strength between molecules; the other is how to find the most suitable binding position, that is, the description and optimization problems of intermolecular forces. The prediction of binding sites is quite important for molecular docking. The ways to use this predictive information to guide docking generally include two types: front-end use and back-end use, which complement each other. Front-end use is to restrict the search space to the local region of the protein in order to accelerate the search process. Back-end use is to predict information to assist in ranking the docking conformations after the search process to reduce false positive results.

17.5.2 Drug-Target Interaction Research Based on Semi-Supervised Learning

Identifying drug-target interaction (DTI) candidates is a key step in drug repositioning. However, the existing databases often store only positive samples and lack negative samples, which poses a challenge for the computational prediction of DTI. To overcome this difficulty, researchers introduce many negative samples into the unlabeled drug-target pairs, among which there are false negative samples. In current studies, the training dataset contains both positive and negative samples, and the classification accuracy and robustness of supervised learning depend on the training

dataset. However, to identify potential DTI, it is difficult to achieve without using a small number of positive samples and negative samples (non-interacting drug-target pairs), and it may even be impossible to obtain a discriminative result. Therefore, negative samples need to be randomly generated from the unlabeled drug-target pairs. However, these unlabeled datasets may contain both positive and negative samples. Thus, the sample selection method is directly related to the discriminative model, and extracting highly reliable negative samples is an important step in predicting DTI.

Semi-supervised learning can also be applied to obtain high-confidence negative samples, which can lead to better results. In existing research materials, unknown DTIs are treated as unlabeled samples, and multiple methods are used to extract negative samples. Currently, positive unlabeled learning (PU-Learning) and target similarity information are also commonly used methods for generating negative samples. For classification applications with unlabeled data, PU-Learning is the most widely used method. Strategies for handling unlabeled samples mainly fall into two categories. One approach is to extract reliable negative samples from the unlabeled data and train the classifier using these reliable negative samples. Spy-EM and Roc-SVM are two representative techniques. Spy-EM is based on the Expectation Maximization (EM) algorithm and the Naive Bayes classifier, while Roc-SVM classifies unlabeled samples by integrating Rocchio technology and support vector machines. Since both known positive samples and extracted negative samples can be utilized, these two techniques exclude ambiguous samples, thereby limiting their capabilities. The other approach fully utilizes ambiguous samples and, based on PU-Learning, selects high-quality negative samples and still uncertain samples from the unlabeled samples for classification. By combining global and local information, ambiguous samples are identified, and the generation method of negative samples also requires further in-depth research.

17.5.3 Drug-Target Binding Affinity Study

Due to the varying degrees of abnormal protein levels within individuals and the different responses of patients to different drugs, it is difficult to use a single drug to treat all patients' lesion sites. Therefore, obtaining biological information data from the database and quickly obtaining a candidate molecule set through computational methods can accelerate the discovery of drugs. The process of predicting drugs through a computational platform can significantly reduce experimental costs. The identification of DTI binding affinity is a crucial step in the process of new drug development. The new drugs generated by the generative model need to obtain valuable information from the biomolecular recognition patterns between drugs and target proteins.

The research on DTI is currently also consistently treated as a binary classification problem, that is, whether there is an interaction between a drug and a target. Some researchers have conducted measurements and studies on the intensity values

of DTI, comparing the intensity values with the threshold to determine whether an interaction can occur between a drug and a target protein. At this point, the interaction intensity value of the drug and the target protein within a certain confidence interval can be obtained, all of which can serve as references for drug design. The binding affinity of drugs to targets is usually expressed by the dissociation constant (K_d), half maximal inhibitory concentration (IC_{50}), and inhibition constant (K_i), which provide certain information for drug-target interaction (DTI) intensity. Computational methods such as molecular docking and deep learning models are widely used in the field of DTI research. In molecular docking, the 3D structural features of drugs and target proteins are manipulated through simulation, but the drawback is that it cannot be applied to large-scale datasets. To overcome this shortcoming, KronRLS and SimBoost based on similarity methods were proposed. However, similarity methods also have disadvantages, such as limiting the maximum number of compound molecules in the training process and using molecules with weak similarity for expression, which reduces the prediction accuracy. To overcome these shortcomings, researchers proposed a deep learning-based model, DeepDTA, which uses an end-to-end convolutional neural network to learn feature representations from the original drug-target biological sequences and then uses fully connected layers to predict drug-target binding affinity. The advantage of this model is that it can use deep learning to automatically find useful information in biological sequences, thus eliminating the need for cumbersome feature engineering steps. This demonstrates the advantages of deep learning models in the field of bioinformatics. Additionally, the DeepDTA model has a single data representation method and only uses feature expressions based on biological characters when learning features from the original sequences. Therefore, researchers proposed a protein sequence feature expression method based on chemical language, using chemical-language-inspired models to encode the features of the target protein sequence, similar to how ligand structures are represented and has a high binding affinity with the corresponding target protein. In fact, how to more effectively utilize feature information to extract modules and discover potential connections between biomolecules from biological sequences is also a current research focus.

References

1. Brian, K., Philip, B.: Advances in protein structure prediction and design. Nat. Rev. Mol. Cell Biol. **20**, 681–697 (2019)
2. Śledź, P., Caflisch, A.: Protein structure-based drug design: from docking to molecular dynamics. Curr. Opin. Struct. Biol. **48**, 93–102 (2018)
3. Molecular, P.I., et al.: Ligand-promoted protein folding by biased kinetic partitioning. Nat. Chem. Biol. **13**, 369–371 (2017)
4. Guo, J.S., Mi, D., Sun, Y.Q.: Folding kinetics of two-state proteins based on the model of general random walk in native contact number space. Physica A **389**, 761–766 (2009)
5. Afify, H.M., Abdelhalim, M.B., Mabrouk, M.S., Sayed, A.Y.: Protein secondary structure prediction (PSSP) using different machine algorithms. Egypt. J. Med. Human Genet. **22** (2021)

6. Mario, C., Emidio, C.: Computational and theoretical methods for protein folding. Biochemistry **52**, 8601–8624 (2013)
7. Eliodoro, C., et al.: Intrinsic map dynamics exploration for uncharted effective free-energy landscapes. Proc. Natl. Acad. Sci. U.S.A. **114**, E5494–E5503 (2017)
8. Jumper, J., et al.: Highly accurate protein structure prediction with AlphaFold. Nature **596**, 583–589 (2021)
9. Abramson, J., et al.: Accurate structure prediction of biomolecular interactions with AlphaFold 3. Nature **630**, 493–500 (2024)

Chapter 18
Deep Learning of Protein–Ligand Interaction Prediction

Protein-small-molecule ligand interactions are central to structure-based drug design, where the core challenge lies in accurately characterizing binding strength and binding mechanisms. This work provides a systematic overview of key binding descriptors—binding constants (Kd/Ka, Ki, and IC50) and binding free energy (ΔG)—and explains how noncovalent interactions, including hydrogen bonding, salt bridges, hydrophobic effects, metal coordination, and cation–π interactions, collectively drive complex formation through structural and energetic complementarity. Methodologically, three mainstream routes for binding free energy/affinity estimation are summarized: physics-based molecular simulation and free-energy calculations (e.g., PMF, enhanced sampling, TI/FEP, MM/PBSA, and MM/GBSA), empirical scoring functions, and knowledge-based/statistical potential approaches. The review further highlights deep learning advances in structure-based affinity prediction and docking re-scoring, covering dataset construction (e.g., PDBbind and its benchmark subsets such as CASF), common molecular representations (molecular descriptors and fingerprints, 3D grid/voxel encodings), and the typical modeling workflow from training and hyperparameter tuning to evaluation. Representative models—including RF-Score, AtomNet, Pafnucy, Kdeep, OnionNet, and PLEC—are discussed for performance comparison. Finally, using feature extraction examples from DeepChem (grid features and fingerprints), this work illustrates random forest and multitask neural network modeling on PDBbind with correlation-based assessment, demonstrating the strong potential of deep learning to improve affinity prediction accuracy and accelerate virtual screening.

S. Chang and L. Xie, *AI-Assisted Drug Design*,
https://doi.org/10.1007/978-981-95-8364-5_18

18.1 Basic Concepts of Drug Targets

The prerequisite for a drug to exert its effect is that it can reach specific tissues in the body and interact with specific biomacromolecules, which are usually referred to as 'drug design targets'. Common targets include receptors, enzyme proteins, structural proteins, ion channels, genes, etc. Currently, most drugs target proteins, followed by DNA and RNA. In the body, enzyme proteins play a crucial role by catalyzing reactions. After binding to substrates, they can efficiently catalyze chemical reactions. If a substrate analogue is designed to competitively bind to the enzyme protein, it can prevent the enzyme from performing its normal function. This drug design approach based on the enzyme catalytic mechanism has achieved considerable success due to its clear reaction mechanism. For instance, kinase inhibitors can act as competitive inhibitors of adenosine triphosphate binding to kinases.

After the target and the target site have been determined, structure-based drug design can be carried out [1]. The process of finding new drugs includes the screening of hit compounds, the discovery of lead compounds, and the selection of candidate drugs. At the beginning of the drug search process, millions of compounds may be screened. Compounds that show activity against the target in initial biological tests are marked as hit compounds. In the subsequent process from hit compounds to lead compounds, the physicochemical properties of the molecules are improved based on their physiological and pharmacological activities and toxicity analysis, and this step will screen out 100 lead compounds. Finally, 1–2 candidate compounds are selected based on the optimization results from animal model tests. For example, the allosteric inhibitor of BCR-ABL1 developed by the globally renowned pharmaceutical company Novartis was obtained from the screening of 50,000 compounds.

The development of a new drug may take over a decade and cost several hundred million dollars. Among them, the discovery of drug lead compounds is the starting stage of finding new drugs. The strategies relied on in this stage are random high-throughput screening or searching from known compounds. Although high-throughput screening has been automated and can screen hundreds of thousands of compounds, it requires huge investment but brings a very low success rate. Random high-throughput screening is an extremely challenging process. If the structure of the target has been obtained, computer-aided virtual screening can be used, which will greatly reduce the cost of blind trial and error in drug screening. Virtual screening is a process of finding ligand molecules based on biological structures using computer and theoretical methods. Currently available computational methods include quantitative structure–activity relationship (QSAR), molecular docking, free energy calculation, machine learning and deep learning methods. The development of these methods all depends on the establishment of databases, including databases of target proteins, small molecule compounds and the interaction between proteins and ligand molecules, etc.

The main terms involved in this chapter are as follows.

- Target: The target of a drug is the site within the body where the drug binds to a biomolecule and exerts its effective interaction.
- Ligand: A small molecule capable of binding to a biomacromolecule.
- Inhibitor: A molecule that can directly or indirectly prevent the substrate from binding to the protein is called an inhibitor.
- Receptor: A membrane protein or protein complex that, upon binding with an agonist, can trigger downstream biological effects.

18.2 Interaction Between Drug Targets and Small Molecules

The binding strength between target proteins and ligand molecules can be measured, and the binding strength can be characterized by the binding constant. According to statistical theory, after the binding of proteins and ligand molecules in solution reaches dynamic equilibrium, the binding constant follows the rule shown in Eq. (18.1).

$$K_{eq,aq} = \frac{[\mathrm{P} \cdot \mathrm{L}]_{\mathrm{aq}}}{[\mathrm{P}]_{aq}[\mathrm{L}]_{aq}} \tag{18.1}$$

Here, $[P]_{aq}$ represents the concentration of the receptor molecule, $[L]_{aq}$ represents the concentration of the ligand molecule, and $[P \cdot L]_{aq}$ represents the concentration after the receptor-ligand binding. The binding constant and the dissociation constant are reciprocals of each other. For enzyme proteins, we usually look for small ligand molecules that inhibit the enzyme's activity. In this case, the inhibition constant K_i shown in Eq. (18.2), is used to measure the binding strength between the protein and the inhibitor molecule.

$$K_i = \frac{[\mathrm{P}]_{aq}[\mathrm{L}]_{aq}}{[\mathrm{P} \cdot \mathrm{L}]_{\mathrm{aq}}} \tag{18.2}$$

It can be seen that the smaller the value of K_i, the stronger the binding ability of the ligand to the protein. In experiments, K_i is usually replaced by a half-maximal inhibitory concentration value. The IC50 of an inhibitor is affected by the affinity of the substrate, as the inhibitor and the substrate competitively bind to the same site on the enzyme protein. Because IC50 is easier to obtain, it is more suitable for characterizing the relative binding ability of inhibitor molecules. There are various biological experimental methods to measure the binding strength of small molecules to targets, such as fluorescence polarization energy transfer (FRET), bio-layer interferometry (BLI), magnetic resonance, and surface plasmon resonance (SPR). These experimental techniques provide effective and reliable experimental results. However, in actual experiments, solvents and proteins and other reagents are required, and

high-throughput screening is conducted. Proteins and biological reagents are not cheap, and the entire process is costly and time-consuming for researchers. Developing efficient drug screening methods has become a concern for pharmaceutical companies.

Computer-aided drug design has played a significant role in the development of new drugs. By using physical models or artificial intelligence methods such as deep learning, the binding ability of ligand molecules to target proteins is calculated and ranked. The top-ranked ligand molecules are then selected for biological experiments, thereby reducing the workload of blind screening. To accurately calculate the binding ability of ligand molecules to target proteins, it is necessary to analyze the mechanism of interaction between the ligand and the protein. An important prerequisite for the binding of ligand molecules to the target is to have an appropriate size and shape. To achieve binding with the protein, ligand molecules can adjust their shape through the rotation of chemical bonds to fit the binding pocket of the target protein. The interaction between ligand molecules and targets should have high affinity and high selectivity so that drug molecules can exert the desired effect without causing unnecessary side effects. On the basis of structural complementarity, the protein and ligand molecules must also achieve energy complementarity.

The functional groups of the ligand need to find complementary interactions with the amino acids of the protein. In-depth analysis and understanding of the interaction between the protein and the ligand are conducive to the discovery of new lead compounds and the rational design of lead compounds. Proteins and ligands are bound together through intermolecular non-covalent interactions, and the interaction forces include electrostatic forces, van der Waals forces, and the interaction energy obtained due to hydrophobic interactions. The description of non-covalent interactions has unique action modes. The main interaction modes between the target and the small molecule ligand include hydrogen bonds, ionic interactions, metal complex interactions, hydrophobic interactions, and cation-π interactions, etc. The types of interactions between proteins and ligands and their descriptions are shown in Table 18.1.

18.3 Calculation of the Binding Free Energy for Protein–Ligand Molecules

The binding ability between drug molecules and their targets is the primary criterion for evaluating the effectiveness of drug molecules [2]. Thermodynamic free energy is widely applied in physics, chemistry, and biology, serving as a criterion for determining whether a process occurs spontaneously. Free energy is the most commonly used physical quantity in biophysics and can be utilized in the study of enzyme catalytic reaction mechanisms, protein structure folding, and protein–ligand binding mechanisms. Therefore, the development of free energy calculation methods is a popular research field internationally. Currently developed methods can

Table 18.1 Types and descriptions of protein–ligand interactions

Type	Description
Hydrogen bond	Hydrogen bonding is a common way of interaction between proteins and ligands, and the proton-carrying part of a biomolecule has an NH or OH group, called a donor of hydrogen bonding; The electronegative atoms with a partial negative charge are called hydrogen bond acceptors, like atoms like O, N, etc. The interaction of hydrogen bonds is usually strong, and the distance between the hydrogen bond acceptor and the donor is within 3 Å and is directional
Ion (salt bridge)	A strong interaction between the charged group of the ligand and the oppositely charged group in the protein. This interaction is caused by electrostatic interactions, and in protein–ligand complexes, ligand binding is determined by this ionic interaction
Metal composites	Some proteins contain metal ions that stabilize the structure of the protein or participate in enzyme-catalyzed reactions. These metal ions can bond donors (e.g., thiols) or bands with hydrogen in ligands, and oppositely charged groups (such as carboxylic acid groups) form particularly stable metal complexes that are compatible with ligands. The binding of proteins forms a decisive influence
Hydrophobic	The lipophilic groups in the ligand and the non-polar amino acids in the protein are close to each other to form hydrophobic interactions. Lipophilic groups include aliphatic hydrocarbons, aromatic hydrocarbons, heterocycles, etc. Hydrophobicity is mainly caused by the proximity of two hydrophobic groups during the binding process, which excludes water from the binding pocket environment. Hydrophobicity has no aromatic properties
Cation-π	Positively charged ionic groups can form strong interactions with aromatic rings. A cation can be a positive group in a ligand or a cation in a protein, such as an ammonium group with a positive charge

calculate solvation free energy, conformational free energy, and binding free energy respectively. Among them, accurately calculating binding free energy is one of the core issues in structure-based drug design and has significant application value. This section will introduce representative traditional free energy calculation methods as well as emerging free energy prediction methods based on machine learning and deep learning.

The binding free energy of a protein and a ligand small molecule describes the free energy change during the process where the ligand small molecule enters the protein receptor cavity or channel from the solution, including the conformational changes of the ligand and receptor during the binding process, as well as the solvent rearrangement at the receptor cavity and ligand surface.

The strength of the combination is expressed as the Gibbs free energy ΔG, as shown in Eq. (18.3).

$$\Delta G = -RT \ln K_{eq,aq} = -RT \ln \frac{[\mathrm{P} \cdot \mathrm{L}]_{\mathrm{aq}}}{[\mathrm{P}]_{aq}[\mathrm{L}]_{aq}} \tag{18.3}$$

The calculation of binding free energy can help to understand the binding mechanism of proteins and ligand molecules in microscopic processes.

The binding free energy of protein–ligand can be divided into the free energy changes of the interaction between the protein and the ligand and the solvation. Currently, the precise calculation of binding free energy is an important research direction in the field. There are three commonly used methods: the first category is to strictly solve the state function based on physical models or estimate the binding free energy through approximate formulas; the second category is to obtain empirical scoring functions through the statistics and fitting of experimental data; the third category is to estimate the binding ability through machine learning or deep learning based on existing experimental data.

18.3.1 Physics-Based Models

The method based on physical models is derived from fundamental formulas. According to the definition of Gibbs free energy ΔG in thermodynamics, the formula shown in Eq. (18.4) can be obtained.

$$\Delta G = \Delta H - T\Delta S \tag{18.4}$$

Among them, ΔH describes the energy change of the system, and $T\ \Delta S$ describes the entropy change of the system. In the protein–ligand binding system, if the free energy of each state can be calculated, the binding free energy can be expressed as Eq. (18.5).

$$\Delta G_{bincl} = \Delta G_{complex} - \left(\Delta G_{protein} + \Delta G_{ligancl}\right) \tag{18.5}$$

The binding free energy is a state function. By calculating the free energy changes of the protein, ligand and complex before and after binding respectively, the change in binding free energy can be obtained. The binding free energy can be decomposed into different components, and the components of the free energy changes of each state can be obtained through the master equation, as shown in Eq. (18.6).

$$\Delta G_{bind} = \Delta G_{\text{int}} + \Delta G_{solv} + \Delta G_{motion} + \Delta G_{conf} \tag{18.6}$$

Among them, ΔG_{int} mainly derived from the van der Waals and electrostatic interactions between the target and the ligand. These interactions are usually local interactions near the binding site. ΔG_{solv} *is* the change in solvation. Water is also an important factor affecting the binding process between the protein and the ligand. During the process of the ligand approaching the protein binding pocket, the water molecules on the protein and ligand surfaces are displaced. ΔG_{motion} *is* the change After the complex structure is formed, the three rotational and three translational degrees of freedom of the molecules disappear. ΔG_{conf} is the change in free energy caused before and after binding. Decomposing the binding free energy into the main equation form can help us better understand which part of the energy change is

favorable for the binding of the ligand and the target, thereby achieving more rational drug design.

Based on the derivation of physical models, the calculation of binding free energy through computer simulation has been achieved. To apply the physical models to the study of protein–ligand interactions, a method based on force field parameters was developed. The potential energy of biological macromolecules such as proteins is described using the parameters of molecular force fields, and the movement and changes of each atom in biomolecules are simulated using Newtonian mechanics. Molecular dynamics simulation is a commonly used method for simulating the structural properties of biomolecules such as proteins. Molecular dynamics simulation can dynamically describe the process of ligand binding to protein receptors. For instance, the Shaw research group was the first to achieve molecular dynamics simulation of protein–ligand binding, obtaining the dynamic process of ligand binding to the target site from long-term molecular dynamics simulation. Free energy calculation based on the process, in simulations, the Potential of Mean Force (PMF) function is widely used to describe the dissociation/association process of receptor-ligand. By calculating the free energy change along the reaction path, the free energy change during the ligand binding process can be obtained. The calculation of PMF is achieved through the probability distribution function, as shown in Eq. (18.7).

$$F = -k_B T \ln \rho \tag{18.7}$$

For more complex binding processes, the time and space scales of ordinary molecular dynamics simulations are limited, leading to the problem of sampling efficiency in the calculation of PMF. Enhanced sampling techniques overcome the problem of insufficient sampling in traditional molecular dynamics and obtain dynamic information of protein conformational transitions within affordable computational resources, such as Umbrella Sampling (US), Integrated Tempering Sampling (ITS), Temperature Accelerated Molecular Dynamics (TAMD), and Markov State Model (MSM). Enhanced sampling methods have been applied to the study of protein–ligand binding processes, by constructing a series of continuous intermediate states between the initial and final states and calculating the free energy changes along the reaction path, which to some extent takes into account the structural changes during the binding process of the receptor and ligand.

In addition to directly reproducing the binding process from molecular dynamics simulations, there are two popular physics-based implementation methods. One is the thermodynamic path-based approach, such as the thermodynamic integration method and the free energy perturbation method. These methods, the thermodynamic integration method and the free energy perturbation method, calculate the relative binding free energy based on the thermodynamic path. They require the system to be transformed from the initial state to the final state through chemical changes in the system's energy function. For instance, this could involve transitioning from two states of ligand A to two states of ligand B, or from the molecule of ligand A to that of ligand B, to compare the different binding free energies of the two states or two similar ligands. Alternatively, ligand A can be mutated to "none", which can provide

the absolute binding free energy. Due to their high computational accuracy, these two methods are widely applied in the optimization process of lead compounds.

Another approach is the endpoint-based method, such as the widely used MM/PBSA or MM/GBSA methods. Compared with the free energy perturbation method, MM/GBSA and MM/PBSA have less computational load and can be applied in the screening and ranking of multiple ligands for a single target. Both methods use implicit solvent models to account for solvent molecules and employ dielectric continuum models to obtain the electrostatic component of the solvation free energy. The difference lies in the models used to calculate the solvation free energy ΔG_{solv} with the GB model or PB model being used to calculate polar solvation ($\Delta G_{PB/GB}$), respectively. The non-polar contribution from both the ligand and the protein is calculated based on the size of the solvent-accessible surface area (ΔG_{SASA}).

18.3.2 Empirical Scoring Function

The empirical scoring function method was an early research approach in computer-aided drug design. This method expresses the interaction between proteins and ligands as empirical formulas and fits the parameters in the empirical formulas using existing experimental results. Empirical formulas are typically composed of simple potential energy functions, such as van der Waals potential, hydrogen bonds, and electrostatic potential. The coefficients in the formulas are fitted from experimentally determined binding affinities. The fitting process aims to maximize the correlation between the calculated binding affinities and the experimental ones. The scoring functions used in molecular docking software are empirical scoring functions. For instance, the interaction energy in AutoDock software is the sum of the following intermolecular interactions, as shown in Eq. (18.8).

$$E_{AutoDock} = E_{vchv} + E_{H-bond} + E_{elec} + E_{inter} \tag{18.8}$$

Among them, E_{vdw} represents the van der Waals potential energy, E_{H-bond} represents the energy related to hydrogen bonds, E_{elec} represents the electrostatic potential energy, and E_{inter} represents the intramolecular potential energy of the ligand.

The advantage of the empirical scoring function method is its fast calculation speed, which enables high-throughput drug screening. Among them, empirical scoring functions and machine learning methods rely on the quality and quantity of existing data. Especially for scoring functions that have been optimized for multiple known ligands of a certain target, the scoring results are the best when screening for this target. Such scoring functions derived from empirical fitting coefficients inevitably have certain limitations. When dealing with systems not present in the training dataset, the accuracy of free energy estimation needs to be improved.

18.3.3 Knowledge-Based Approaches

The knowledge-based approach involves the statistical analysis of existing knowledge and the extraction of inherent patterns to address new problems. The existing knowledge encompasses the crystal structure of molecules, crystal packing modes, protein structures, protein sequences, and the structures of complexes formed by proteins and ligand molecules. The commonly used database is PDB. Within these biological data, by setting algorithms, the structure–activity relationship of molecules can be established. This type of method can be regarded as establishing a statistical potential energy. This statistical potential energy is based on the energy minimum point of the structures in PDB being in thermodynamic equilibrium, so the distribution of atoms in the complex structure conforms to the Boltzmann distribution. When predicting the binding energy of proteins and molecules, the relationship between the occurrence frequency of certain types of atoms and the binding free energy can be statistically analyzed from the structure of the protein and the structure of the complex formed by the protein and the ligand, as shown in Eq. (18.9).

$$A_{ij}(r) = -k_B T \ln\left[\frac{\rho_{ij}(r)}{\rho_{ij}(bulk)}\right] \tag{18.9}$$

Among them, $\rho_{ij}(r)$ represents the probability of the defined atomic pair occurring within a distance r when in a bound state, and $\rho_{ij}(bulk)$ represents the probability of the defined atomic pair occurring within a distance r when in an unbound state. The knowledge-based approach requires the pre-definition of a formula, which utilizes human experience and knowledge to mine hidden patterns in the data. Due to the complexity of biological data, it is sometimes difficult to find a formula that can summarize all interactions. This can be easily achieved by using artificial intelligence methods.

18.3.4 Protein–Ligand Molecular Binding Affinity Based on Deep Learning

Based on the structure of protein–ligand binding and the experimentally determined binding constants, we can conduct structure-based deep learning, that is, using deep learning algorithms to discover hidden patterns from the data such as structures and corresponding binding constants. Compared with the empirical formula-based models, the organic combination of artificial intelligence methods and big data has changed the field of computer-aided drug design and has become a new drug design method that surpasses traditional computational models. In recent years, artificial intelligence technology has been gradually applied to the prediction of the binding strength between proteins and drug molecules. With the help of machine learning or deep learning methods, the intrinsic rules can be obtained from the complex structures

of protein-small molecule interactions, thereby predicting the binding free energy of proteins and ligand molecules. Deep learning methods have been applied in both scoring functions and predicting binding modes.

The general steps for predicting the binding constant of protein–ligand molecules based on deep learning are as follows.

(1) Dataset preparation. For different problems, relevant datasets need to be selected. In predicting protein–ligand interactions, among the problems of interaction, the most frequently used dataset is PDBbind. A subset with higher accuracy, CASF, was selected from this dataset. The latest version is CASF-2016, which has played an important role in the optimization of scoring functions and is currently an important benchmark dataset in the field of deep learning.
(2) Molecular feature representation. Transform the three-dimensional coordinate information of proteins into digital information that is easy for computers to process. Nowadays, it is possible to autonomously learn molecular representations through deep learning methods, eliminating the need for feature engineering. As a result, various methods for molecular descriptors or molecular fingerprints have emerged.
(3) Establish a predictive model. The choice of the predictive model depends on whether the problem is a classification, regression or generation problem. In the case of predicting protein–ligand binding constants, the predictive model can be selected from machine learning methods (such as support vector machines, decision trees, etc.) or deep learning methods (such as deep neural networks and convolutional neural networks, etc.).
(4) Optimize the model's hyperparameters. The optimization of hyperparameters directly determines the model's predictive performance. A set of hyperparameters can be found through grid search or Bayesian optimization. Various artificial intelligence methods have achieved good results. A notable issue is how to prevent overfitting of the model. Models typically perform well on the training set but poorly on the test set, which indicates the occurrence of overfitting.
(5) Analysis of prediction results. The purpose of training an artificial intelligence model is to achieve predictions on the issues of concern. For instance, using the optimized model to re-rank the structures of molecular docking, or to classify ligand molecules as either active or inactive.

In this process, the key step is how to achieve the digital encoding of the structure of biological molecules such as proteins. The encoding of proteins can be roughly divided into two methods: molecular descriptors and molecular fingerprints. Molecular descriptors use predefined atomic types, atomic charges, and physical and chemical molecular features such as the interaction between proteins and atoms to count the occurrence of different descriptors within a certain distance, thereby converting structural information into digital information. Molecular fingerprints are a digital representation of molecules, encoding them as a series of bit strings. For example, the Extended Connectivity Fingerprint (ECFP) generates a fingerprint of a specific length by hashing the identifiers corresponding to all molecular fragments on the

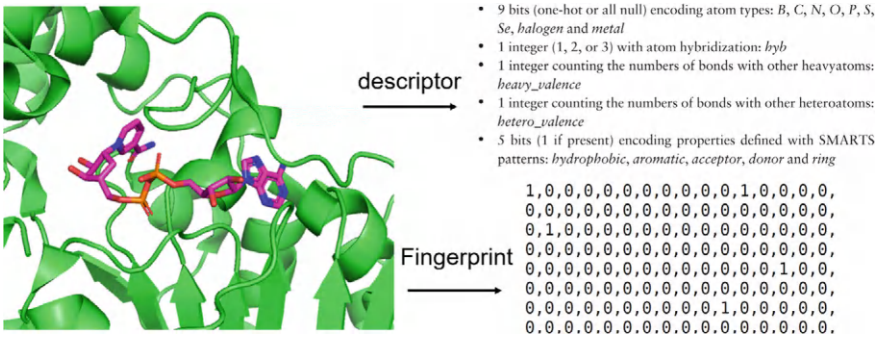


Fig. 18.1 Schematic diagram of transforming protein–ligand molecules into molecular descriptors and molecular fingerprints respectively

paths starting from an atom until a specified number of bonds are reached, as shown in Fig. 18.1.

So far, a variety of structure-based deep learning models have been reported. Representative methods include RF-Score and AtomNet. and Pafnucy et al. defined 9 types of atoms for both proteins and ligands and used the occurrence frequency of atom types between protein–ligand molecules as descriptors, resulting in a total of 81 pairs of atom types. By applying the random forest method to perform regression analysis on the 81 descriptors and binding constants, the goal of predicting binding constants through machine learning was achieved. In 2015, Wallach et al. reported the first deep learning model based on structure, AtomNet, for predicting the biological activity of small molecules using convolutional neural networks. AtomNet uses a 3D grid with a 1 Å interval, defines the ligand binding position as the origin, and calculates whether basic structural features appear within 20 Å around the origin to extract structural information that can be used in deep convolutional neural networks. AtomNet [3] is a classification model that outputs 1 if the ligand molecule is active and 0 otherwise. In 2018, Pafnucy used the 3D Cartesian coordinates around the ligand molecule and 19 molecular features (atom types, bond numbers, heteroatoms, charges, etc.) to describe the structure of protein–ligand complexes [4]. K_{deep} converts the structure of protein–ligand complexes into 3D pixel information using 3D molecular descriptors, which include hydrophobicity, aromaticity, hydrogen bond acceptors, hydrogen bond donors, positive ions, negative ions, metals, and atomic van der Waals volumes. This 3D pixel information is input into a 3D convolutional neural network for training to predict the binding constants of protein–ligand complexes. PLEC is a method based on ECFP [5], applying ECFP to the structural encoding of protein–ligand molecules. OnionNet converts the ligand and the surrounding protein amino acids into specified molecular features and then uses convolutional neural networks for training and prediction. The comparison of the prediction performance of different models on the PDBbind dataset is shown in Table 18.2. It can be seen that the relative error RMSE calculated by the deep learning-based methods (the first 5 items in the table) are smaller than that of the molecular docking method

Table 18.2 Comparison of prediction performance of different models

Model	RMSE	R
RF-score-v3	1.39	0.80
Pafnucy	1.42	0.78
K_{deep}	1.27	0.82
OnionNet	1.278	0.816
PLEC	1.25	0.83
AutoDock Vina (2013)	1.90	0.54

AutoDock Vina, and the correlation coefficient between the predicted values and the experimental values is higher than that of the molecular docking methods.

18.4 Practical Application of AI in Predicting Protein–Ligand Binding Affinity

This section will mainly discuss the practical applications of artificial intelligence in predicting protein–ligand binding affinity.

18.4.1 Feature Extraction of Targets and Small Molecules

The key step in applying artificial intelligence methods to the prediction of protein–ligand binding energy is to convert the three-dimensional structural information into digital information that can be processed by computers. This requires the use of molecular characterization methods to bridge biological molecules and digital information, that is, the encoding of biological molecules. Currently, there are many methods in the industry that can encode the interaction between proteins and ligands, such as PLEC and SPLIF, which extend ECFP to protein–ligand molecules. The PDB contains the three-dimensional structures of biological macromolecules. The PDBbind database is a specialized repository that collects and organizes binding coefficients of protein and ligand molecules. We can obtain the three-dimensional complex structure information and binding constants of protein–ligand interactions from the PDBbind database. The experimental data obtained are used to train the deep learning model.

There are various ways to extract features of the interaction between drug targets and small ligand molecules. For ease of learning, here we take the Grid Featurization method developed by the Pande group as an example. The Grid Featurization encoding method contains both physicochemical information and molecular fingerprint information, and has also shown good performance in predicting the binding constants of protein–ligand molecules. This method has been incorporated into the

DeepChem software, which is a well-packaged software for artificial intelligence-assisted drug screening and material design. The extraction of molecular features and the artificial intelligence model are all integrated within one software, and the required functions can be achieved through simple scripts.

18.4.2 Fingerprint Extraction Based on Protein–Ligand Interactions

Use DeepChem to convert the ligand molecules into ECFP, as shown in Listing 18.1.

Listing 18.1 Obtaining the ECFP Fingerprints of Ligand Molecules

```
from rdkit import Chem
from rdkit.Chem import AllChem
import deepchem as dc

fp_featurizer = dc.feat.CircularFingerprint(size=2048)
ligands = ['ligand_1u1b.pdb']

features = fp_featurizer.featurize([Chem.MolFromPDBFile(l)
for l in ligands])
```

The fingerprints of ligand molecules are more often applied to the prediction of the physical and chemical properties and pharmacokinetics of the ligand molecules themselves or to the search for molecules with similar structures. If one wants to predict the binding free energy of protein–ligand molecules, it is necessary to incorporate the information of the protein into the molecular fingerprints, because the interaction between molecules plays an important role in the binding process. If one wants to obtain the fingerprint of the protein–ligand complex structure, the code shown in Listing 18.2 can be used.

Listing 18.2 Molecular Fingerprints of Protein and Ligand Molecules

```
from rdkit import Chem
from rdkit.Chem import AllChem
import deepchem as dc

fp_featurizer = dc.feat.RdkitGridFeaturizer(voxel_
width=16.0,
    box_width = 16.0,
```

```
        feature_types=['hbond', 'salt_bridge', 'pi_stack',
'cation_pi',
     'ecfp', 'splif'],
     ecfp_power=9,
     splif_power=9,
     sanitize=True, flatten=True)
proteins = ['1u1b_pocket.pdb']
ligands = ['ligand_1u1b.pdb']
features = fp_featurizer.featurize(zip(ligands, proteins))
print(features)
print(features.shape)
```

Among them, 'hbond', 'salt_bridge', 'pi_stack', 'cation_pi' in feature_types are structural information based on chemical interactions, while 'ecfp' and 'splif' are information of molecular fingerprints.

18.4.3 Application of AI for Protein–Ligand Binding Interaction Prediction

After obtaining the encoded information of protein–ligand complex structures, the artificial intelligence model can be trained, and then the trained model can be used for subsequent predictions. For instance, we can utilize the trained AI model to re-score and re-rank the structures obtained from molecular docking, thereby enhancing the scoring capability of molecular docking. We selected the "refined" subset from the PDBbind database as the training and testing dataset. Compared with CASF-2016, the "refined" dataset contains more protein–ligand binding structures. Since deep learning is a greedy algorithm, the more data available, the better the prediction performance. We used Listings 18.3 and 18.4 to compare the prediction performance of the typical machine learning algorithm, the random forest model, and the deep learning algorithm, the multi-task neural network model, in predicting protein–ligand binding constants.

Listing 18.3 presents an example of training and prediction using the Grid Featurization molecular fingerprint of protein–ligand complex structures with a random forest. Grid Featurization is a specialized feature calculation method developed by the Pande group for encoding protein–ligand complex structures.

Listings 18.3 Training a Random Forest Model Using Protein and Ligand Molecular Fingerprints

```
from rdkit import Chem
import os
import deepchem as dc
```

```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from deepchem.molnet import load_pdbbind
from sklearn.model_selection import train_test_split
# For stable runs
np.random.seed(123)

split = "random"
feat="grid"
dataset="refined"

current_dir = os.path.dirname(os.path.realpath(__file__))
pdb_dir = os.path.join(current_dir, "%s_%s" % (dataset,
feat))

pdbbind_tasks = ["-logKd/Ki"]
dataset = dc.data.DiskDataset(pdb_dir)
transformers = []

X, y, w, ids = dataset.X, dataset.y, dataset.w, dataset.ids
print(len(X), X.shape)
print(X[:,0])
print(y[:10], ids[:10])

X_train, X_test, Y_train, Y_test = train_test_split(X, y,
test_size=0.2, random_state=123)

#sklearn model
sklearn_model = RandomForestRegressor(n_estimators=500)

sklearn_model.fit(X_train, Y_train)
print("train score: ", sklearn_model.score(X_train, Y_
train))
print("test score: ", sklearn_model.score(X_test, Y_test))

Y_predict = sklearn_model.predict(X_test)
import scipy
from scipy import stats
print("pearson:     ",scipy.stats.pearsonr(Y_test.reshape(-
1),Y_predict))
from sklearn.metrics import r2_score
print(r2_score(Y_test.reshape(-1),Y_predict))
```

The output result is as follows:
Train score: 0.9349345760655188.
Test score: 0.5340532857885314.
Pearson: (0.741329666049171, 1.97912286664592e-125).

On the training set, the score of the random model we trained reached 0.93, while on the test set, it was 0.53. The random forest model in this calculation needs further optimization. For this purpose, the Pande group developed a multi-task neural network model with stronger fitting ability. Listing 18.4 shows an example of training and prediction of the Grid Featurization molecular fingerprints of protein–ligand complexes using the multi-task neural network model.

Listings 18.4 Training a Multi-task Neural Network Model Using Protein and Ligand Molecular Fingerprints

```
import os
import deepchem as dc
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from deepchem.molnet import load_pdbbind
from sklearn.model_selection import train_test_split
# For stable runs
np.random.seed(123)

split = "random"
feat="grid"
dataset="refined"

current_dir = os.path.dirname(os.path.realpath(__file__))
pdb_dir = os.path.join(current_dir, "%s_%s" % (dataset,
feat))

pdbbind_tasks = ["-logKd/Ki"]
dataset = dc.data.DiskDataset(pdb_dir)
transformers = []

#sklearn model
X, y, w, ids = dataset.X, dataset.y, dataset.w, dataset.ids
print(len(X), X.shape)
print(X[:,0])
print(y[:10], ids[:10])

#deepchem model
dataset = dc.data.NumpyDataset(X=X, y=y)
splitter = dc.splits.RandomSplitter()
train_dataset, valid_dataset, test_dataset = splitter.train_
valid_test_split(dataset, frac_train=0.8)

metric = dc.metrics.Metric(dc.metrics.pearson_r2_score)
n_features=X.shape[1]
model = dc.models.MultitaskRegressor(
  1,
```

```
    n_features,
    layer_sizes=[1000],
    dropouts=[.25],
    learning_rate=0.001,
    batch_size=50,
    verbosity="high")

# Fit trained model
model.fit(train_dataset)

import scipy
from scipy import stats

dnn_predicted_train = model.predict(train_dataset, trans-
formers)
dnn_true_train = train_dataset.y

dnn_predicted_test = model.predict(test_dataset, trans-
formers)
print(dnn_predicted_test.shape)
dnn_true_test = test_dataset.y
print(dnn_true_test.shape)

from scipy import stats
print("train pearson: ",scipy.stats.pearsonr(dnn_true_
train.reshape(-1),dnn_predicted_train.reshape(-1)))
print("test pearson: ",scipy.stats.pearsonr(dnn_true_
test.reshape(-1),dnn_predicted_test.reshape(-1)))
```

The calculation results are as follows:

Train Pearson: (0.9406207609910671, 0.0).

Test Pearson: (0.7563313666541416, 2.096440771445612e-67).

Deep learning methods achieve better performance than machine learning method. It shows better correlation on the test set. The correlation graph of the predicted values and the experimental values is shown in Fig. 18.2.

The deep learning method provides a more effective tool for evaluating the binding ability of ligands to proteins. With more and more data available, the resolution of protein–ligand complex structures and the increase in biological information will make the performance of deep learning, as a greedy algorithm, more accurate. Deep learning will play a greater application value in predicting binding constants and binding modes based on structural prediction, and is expected to further accelerate the process of drug design.

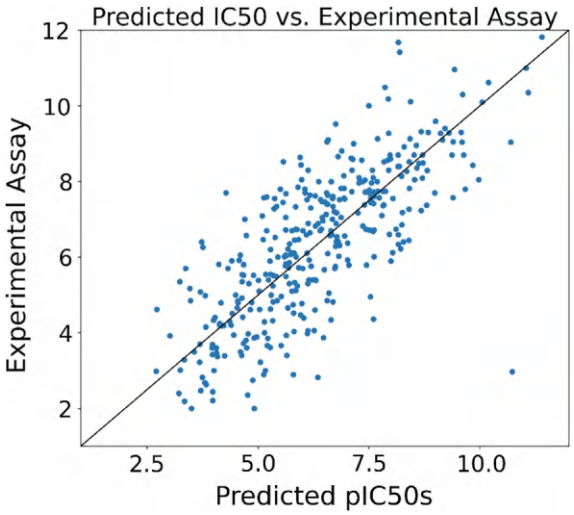


Fig. 18.2 Correlation graph of the predicted values and experimental values by the multi-task neural network

Notice

Listings 18.1 and 18.2 calculate molecular fingerprints for ligands and protein–ligand complexes, respectively. Listing 18.2 implements the conversion of protein–ligand complex information into molecular fingerprints. In this example, the default parameters of DeepChem are used. In practical applications, the parameters of dc.feat.RdkitGridFeaturizer need to be optimized and adjusted according to the specific system and GPU memory.

References

1. Wang, X., Song, K., Li, L., Chen, L.: Structure-based drug design strategies and challenges. Curr. Top. Med. Chem. **18**(12), 998–1006 (2018)
2. Buchwald, P., Bodor, N.: Computer-aided drug design: the role of quantitative structure-property, structure-activity and structure-metabolism relationships (QSPR, QSAR, QSMR). Drugs Fut. **27**, 577–577 (2002)
3. Wallach, I., Dzamba, M., Heifets, A.: AtomNet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery. CoRR **1510**, 02855 (2015)
4. Kumar, P., Bankapur, S., Patil, N.: An enhanced protein secondary structure prediction using deep learning framework on hybrid profile based features. Appl. Soft Comput. J. **86**, 105926–105926 (2020)
5. Maciej, W., Michał, K., et al.: Development of a protein-ligand extended connectivity (PLEC) fingerprint and its application for binding affinity predictions. Bioinformatics **35**, 1334–1341 (2019)