Transactions on Computer Systems and Networks

Govindakumar Madhavan

Mastering Machine Learning: From Basics to Advanced

Springer

Transactions on Computer Systems and Networks

Transactions on Computer Systems and Networks is a unique series that aims to capture advances in evolution of computer hardware and software systems and progress in computer networks. Computing Systems in present world span from miniature IoT nodes and embedded computing systems to large-scale cloud infrastructures, which necessitates developing systems architecture, storage infrastructure and process management to work at various scales. Present day networking technologies provide pervasive global coverage on a scale and enable multitude of transformative technologies. The new landscape of computing comprises of self-aware autonomous systems, which are built upon a software-hardware collaborative framework. These systems are designed to execute critical and non-critical tasks involving a variety of processing resources like multi-core CPUs, reconfigurable hardware, GPUs and TPUs which are managed through virtualisation, real-time process management and fault-tolerance. While AI, Machine Learning and Deep Learning tasks are predominantly increasing in the application space the computing system research aim towards efficient means of data processing, memory management, real-time task scheduling, scalable, secured and energy aware computing. The paradigm of computer networks also extends it support to this evolving application scenario through various advanced protocols, architectures and services. This series aims to present leading works on advances in theory, design, behaviour and applications in computing systems and networks. The Series accepts research monographs, introductory and advanced textbooks, professional books, reference works, and select conference proceedings.

Govindakumar Madhavan

Mastering Machine Learning: From Basics to Advanced

Springer

Govindakumar Madhavan
SeaportAi
Chennai, India

ISSN 2730-7484 ISSN 2730-7492 (electronic)
Transactions on Computer Systems and Networks
ISBN 978-981-97-9913-8 ISBN 978-981-97-9914-5 (eBook)
https://doi.org/10.1007/978-981-97-9914-5

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

Contents

About the Author

Govindakumar Madhavan is the founder and CEO of SeaportAi. He completed his B.Tech. and MBA from top institutes in India. He also holds certifications in six sigma and project management. He has over two decades of experience managing technology, operations, and quality in large MNCs and start-ups. He has led high performance multi-national teams and managed businesses across the Asia Pacific region. He has successfully incubated Centers of Excellence for fraud prevention and service analytics. He has significant experience in design thinking-based product development and management. He has also played a critical role in developing products for emerging markets and won global awards in the areas of Customer Experience, Leadership Excellence, Quality, and Technology. He is a prolific author of about 40 video courses and 10 books.

Chapter 1
About the Book

Abstract This chapter introduces the book, what to expect besides links to code and datasets.

Keyword Machine learning

The interest in machine learning (ML) as well as the application of machine learning is growing rapidly. Since machine learning is at the intersection of mathematics, programming, and domain knowledge, machine learning is best learnt from an application perspective rather than from a theoretical perspective. This is especially true for beginners as well as young practitioners.

This book attempts to bring application and industry perspectives to both the mathematics aspects and the programming aspects of machine learning. All the codes associated with algorithms are explained at length (Fig. 1.1).

Unique attempt: video lectures

In a distinctive approach, the book's content is complemented by video lectures that are available on iversity.com. This offers readers a multimedia learning experience, accommodating different learning preferences, and reinforcing the material through visual and auditory means.

The book covers the following areas.

- Machine learning concepts.
- Explanation of mathematical concepts relevant to machine learning, explained from an application perspective.
- Supervised learning algorithms.
- Unsupervised learning algorithms.
- Deep learning algorithms.

Depending on your interest and knowledge level, you can either go through the book in the above order or jump to specific areas.

---

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_1

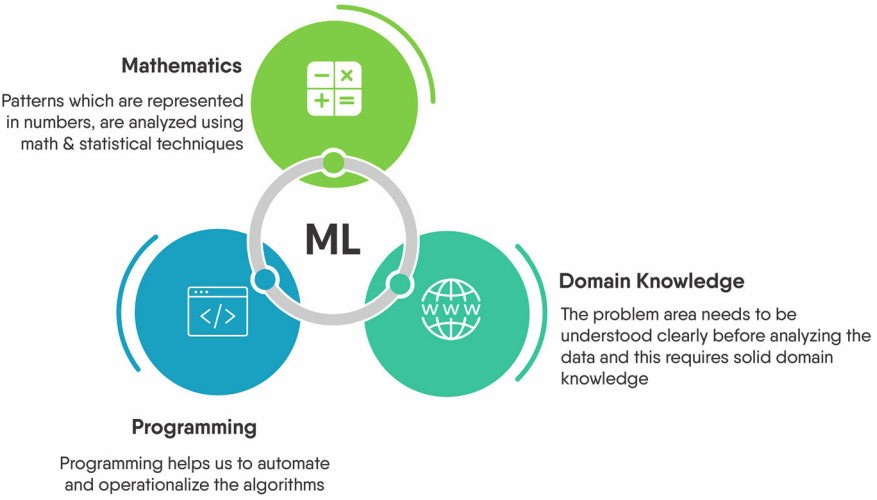


Fig. 1.1 Machine learning at intersection of three disciplines

Who this book is for:

- **Students**: This book is ideal for students at various stages of their education who are interested in entering the field of machine learning. It provides foundational knowledge that is crucial for beginners.
- **Young professionals in machine learning**: Young professionals who have recently started their careers in machine learning will find this book beneficial for consolidating their basic understanding and enhancing their practical skills.
- **Professionals from other disciplines wanting to get into machine learning**: For those looking to pivot into machine learning from different professional backgrounds, this book serves as a comprehensive guide to the basics and practical applications of ML concepts.

Who this book is not for:

- **Experts in machine learning**: Individuals who already have an advanced understanding of machine learning and are proficient in the latest ML techniques and theories might not find this book useful.
- **Those looking for advanced concepts** in machine learning: Readers in search of in-depth exploration of advanced machine learning topics may need to look beyond this book, which is tailored more toward foundational and intermediate knowledge.

Pre-requisites:

- Basic knowledge of Python programming including Pandas.

Datasets and codes

- The datasets used in this book along with the codes will be available at https://github.com/amgovindkumar/springer.

Errata

The book has undergone multiple revisions aimed at accuracy and clarity, thus minimizing the likelihood of errors. However, should readers encounter any mistakes, they are encouraged to report them to info@seaportAi.com so that the book can be further improved in future editions. This open channel for feedback demonstrates the author's commitment to the quality and reliability of the material and openness to receive feedback.

Chapter 2
Introduction to AI

AI Is the New Electricity, Illuminating Paths in Data Darkness

Abstract This chapter introduces readers to Artificial Intelligence (AI), exploring its fundamental nature as systems designed to emulate human cognition, problem-solving, and decision-making capabilities. Readers can look forward to gaining clarity on how Machine Learning enables AI systems to independently recognize patterns and improve performance from data without direct programming. A concise overview of AI history highlights milestones, periods of significant advancement, setbacks, and the influence of evolving technology. Special attention is given to the current "ChatGPT Era," showcasing how sophisticated language models have transformed communication and interactions between humans and machines. Additionally, the chapter explains the significance of the Turing Test, devised by Alan Turing, as a benchmark for evaluating a machine's intelligence based on human-like conversational ability.

Keywords Artificial intelligence · ChatGPT · Turing Test · Traditional programming · Machine learning

2.1 What Is Artificial Intelligence?

Artificial intelligence (AI), in simple terms, refers to the capability of a computer or a machine to imitate intelligent human behavior. It encompasses a broad range of technologies that enable machines to sense, comprehend, act, and learn. AI can be as simple as a chess program or as complex as a self-driving car.

Artificial intelligence (AI) has evolved significantly since its inception. The term "Artificial Intelligence" was first coined in 1956 at a conference at Dartmouth College. Initially, AI research focused on problem-solving and symbolic methods. In the 1960s, the US Department of Defense took interest in this type of work and began training computers to mimic basic human reasoning. However, the limitations of the then computing power slowed progress.

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_2

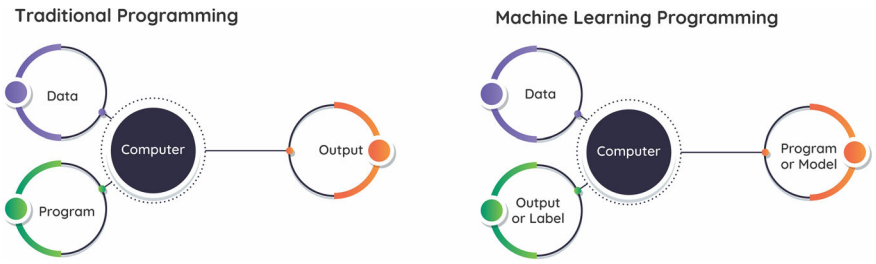


Fig. 2.1 Difference between traditional programming versus machine learning programming

As AI developed, a new branch emerged—machine learning (ML), which is a subset of artificial intelligence. Machine learning involves the development and use of algorithms that allow computers to learn from data. This learning is subsequently used to make predictions or decisions. This approach differs from traditional programming, where humans explicitly code the computer's every action (Fig. 2.1).

2.2 How Does the Machine Learn?

Machine learning algorithms use statistical techniques and equations to enable machines or software to improve at tasks with experience. The process starts with feeding the computer a large set of data. The machine then uses this data to train itself to recognize patterns and make decisions. Over time, as it is exposed to more data, its ability to make decisions and predictions improves.

At the heart of machine learning is pattern recognition. Computers are taught to recognize patterns and make connections between them. This ability is what enables a facial recognition system to identify faces or a recommendation system to suggest products you might like. Pattern recognition involves understanding the linkages in the historical data, which the machine uses to make predictions about new data.

2.3 History of Artificial Intelligence

The history of artificial intelligence (AI) is a tale of dreams, innovation, and milestones that stretch back much further than many people realize. It all began in the realm of science fiction, with stories of intelligent machines that could think and act on their own. But the real journey of AI started in the mid-twentieth century.

In the 1950s, a mathematician named Alan Turing asked a groundbreaking question: "Can machines think?" This question kicked off serious scientific efforts to create intelligent machines. Turing also developed the Turing Test, a method for determining whether a machine is capable of thinking like a human.

The term “artificial intelligence” was first coined by John McCarthy in 1956 at a conference at Dartmouth College. Here, McCarthy and others believed that machines could be made to simulate every aspect of learning or any other feature of intelligence.

The 1960s and 70s saw the first wave of AI, where researchers were optimistic about the future. They built the first AI programs that could solve algebra problems, prove theorems, and even understand natural language to a certain extent.

But then, AI hit its first roadblocks in the 1970s, a period known as “AI winter,” and again in the 1980s. Funding dried up, and the high expectations were not met primarily because of the limitations in computer power and complexity of human intelligence.

The 1990s brought a resurgence of interest and progress in AI, thanks to improved computer technology and new approaches like machine learning, where computers could learn from data. The victory of IBM’s Deep Blue over chess champion Garry Kasparov in 1997 was a significant milestone that showed the world what AI could do.

The twenty-first century has seen rapid progress in AI. We have voice-activated assistants, cars that can drive themselves, and algorithms that can diagnose diseases. AI has become a part of everyday life, with systems that can understand and process human language, recognize faces, and make decisions.

In recent years, the field of AI has continued to advance, with technologies like deep learning, where neural networks—systems inspired by the human brain—can learn from vast amounts of data. This has led to improvements in image and speech recognition that were hard to imagine just a few decades ago.

As we move forward, AI continues to push the boundaries of what machines are capable of, creating new possibilities and raising important questions about the future of technology and society. The story of AI is still being written, and the next chapters are likely to be even more astonishing as the line between human and machine intelligence becomes increasingly blurred.

2.4 The ChatGPT Era

As the history of artificial intelligence unfolded, a new chapter began with the arrival of ChatGPT, a sophisticated language model developed by OpenAI. ChatGPT, which stands for “Chat Generative Pre-trained Transformer,” represents a significant leap forward in the field of natural language processing (NLP).

The roots of ChatGPT lie in the development of machine learning models known as transformers, introduced in a 2017 paper titled “Attention is All You Need.” These models revolutionized the way machines understand and generate human language. They could pay “attention” to different parts of a sentence to understand context and meaning more effectively than ever before.

Building on this, ChatGPT was trained on a vast corpus of text data, allowing it to generate human-like text. It can engage in conversations, answer questions, write essays, and even create poetry. This is not just about finding the right words; it’s

about understanding the nuances of human language—a task that has traditionally been incredibly complex for machines.

The era of ChatGPT began when OpenAI released versions of the model to the public, where it was met with both enthusiasm and astonishment. Users found that they could have meaningful and coherent conversations with an AI. This was a far cry from the rule-based chatbots of the past that could only handle very specific queries.

ChatGPT has found its place in various applications:

- Customer Service: Automating responses to customer inquiries with a level of understanding that standard chatbots can't match.
- Education: Assisting in learning and tutoring by providing explanations to complex concepts and answering student questions.
- Content Creation: Helping writers by suggesting ideas, drafting content, and even writing code.
- Entertainment: Engaging users in interactive storytelling and creating personalized gaming experiences.

However, the ChatGPT era is not without its challenges. Concerns about misinformation, the potential for generating biased content, and the implications for job displacement are part of ongoing discussions. Furthermore, the technology raises important ethical questions about the use and misuse of AI-generated text.

The development of ChatGPT has also sparked a broader conversation about the future of AI and its integration into society. As AI becomes more advanced, issues of governance, security, and the societal impact of AI will need to be addressed more fully.

Despite these challenges, the ChatGPT era is an exciting time in the evolution of artificial intelligence. It marks a moment where AI began to interact with us in a way that feels more human, reshaping our relationship with technology and opening up new possibilities for how we communicate, work, and create. The story of ChatGPT continues to unfold, and its full impact on society remains to be seen.

2.5 The Turing Test

The Turing Test is a famous concept in the world of artificial intelligence, introduced by the British mathematician and computer scientist Alan Turing in 1950. The test was a part of Turing's paper, "Computing Machinery and Intelligence," where he pondered the question: "Can machines think?"

The Turing Test is a simple yet powerful idea. Turing proposed a test to determine whether a machine can exhibit intelligent behavior that is indistinguishable from that of a human. The test involves a human judge who interacts with two participants: one human and one machine. The interactions are typically text-based, so the judge cannot see or hear the participants. The judge asks questions, and both the human and the machine respond (Fig. 2.2).

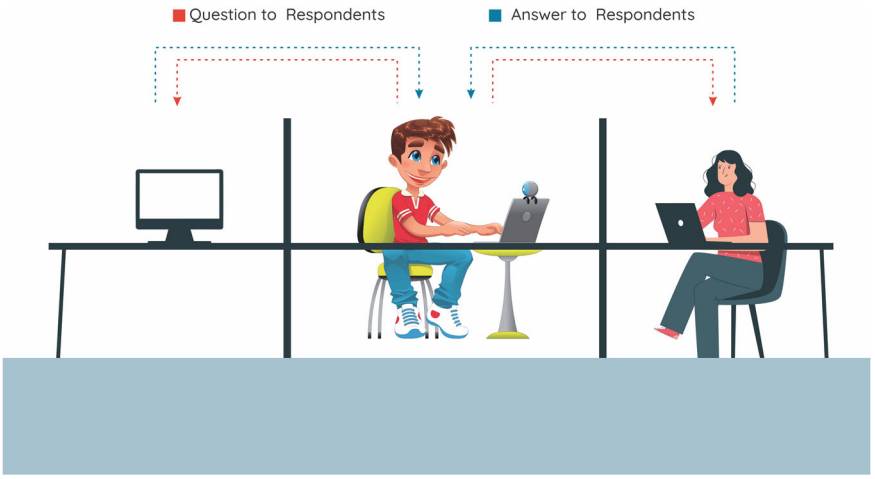


Fig. 2.2 Turing Test

The essence of the Turing Test is deception. If the judge can't reliably tell which participant is the machine and which is the human, the machine is said to have passed the Turing Test. In other words, if the machine can mimic human responses well enough that it fools the judge, it's considered a sign of intelligence.

The Turing Test has been both influential and controversial. It shifted the focus from whether machines can think to whether machines can behave like they think. It's more about the appearance of intelligence than intelligence itself. This has led to debates about what it truly means to be intelligent and whether the Turing Test is a valid measure of AI.

Despite its simplicity, the Turing Test has had a significant impact on the development of AI. It has been used as a benchmark for AI progress, inspiring the creation of more sophisticated and human-like AI systems.

However, the test has its limitations. Critics argue that passing the Turing Test doesn't necessarily mean a machine is truly intelligent. It might just be good at imitating human speech. Also, the test doesn't account for non-verbal intelligence or other forms of intelligence that don't involve human-like conversation.

Do you think ChatGPT will pass the Turing Test with flying colors?

Chapter 3
Pattern Recognition

Patterns Are Nature's Whispers, Revealing Secrets in Data's Chaos

Abstract In the previous chapter, readers explored foundational concepts of AI and learned how machines use data-driven approaches to emulate human intelligence. Building on this understanding, this chapter introduces readers to the core concept of pattern recognition—a vital process underlying machine learning. Through engaging quizzes, including classic problems such as predicting the next number in a series and the Fibonacci sequence, readers actively practice identifying and interpreting patterns. Each exercise encourages learners to follow key steps: understanding the given data, uncovering hidden relationships or patterns, and utilizing those insights to make accurate predictions. This approach mirrors exactly what occurs in machine learning, where algorithms analyse data to detect patterns and predict outcomes. By the chapter's end, readers will recognize how human pattern recognition serves as a blueprint for the techniques employed by intelligent systems.

Keywords Fibonacci series · Pattern recognition

3.1 Identify the Next Number in the Series

Question 1: What is the next number in the series: 2, 4, 8, 16, 32, …?

(A) 40
(B) 48
(C) 64
(D) 56

Question 2: What is the next number in the series: 5, 10, 20, 40, 80, …?

(A) 100
(B) 120
(C) 160
(D) 200

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_3

Question 3: What is the next number in the series: 1, 1, 2, 3, 5, 8, …?

(A) 10
(B) 11
(C) 13
(D) 15

Question 4: What is the next number in the series: 3, 9, 27, 81, …?

(A) 162
(B) 243
(C) 324
(D) 405

Question 5: What is the next number in the series: 2, 3, 5, 8, 12, …?

(A) 15
(B) 17
(C) 18
(D) 20

3.2 Answers and Explanation to the Multiple-Choice Questions

Question 1

Series: 2, 4, 8, 16, 32, …

Answer: (C) 64

Explanation: The pattern in this series is each number being multiplied by 2. So, $32 \times 2 = 64$.

Question 2

Series: 5, 10, 20, 40, 80, …

Answer: (C) 160

Explanation: Similar to the first, this series multiplies each number by 2. Thus, $80 \times 2 = 160$.

Question 3

Series: 1, 1, 2, 3, 5, 8, …

Answer: (C) 13

Explanation: This is the Fibonacci sequence, where each number is the sum of the two preceding ones. Therefore, $8 + 5 = 13$. The golden ratio as shown below gets repeated across numbers (Fig. 3.1).

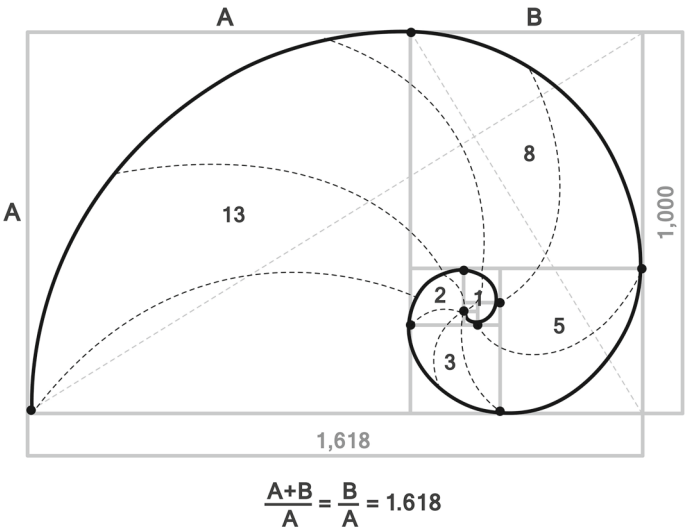


Fig. 3.1 Fibonacci series

Question 4

Series: 3, 9, 27, 81, …

Answer: (B) 243

Explanation: This series involves multiplying each number by 3. Hence,

$81 \times 3 = 243$

Question 5

Series: 2, 3, 5, 8, 12, …

Answer: (B) 17

Explanation: This series adds 1, 2, 3, 4, … to each number. So, the next number is

$12 + 5 = 17$

Each series follows a distinct numerical pattern, and identifying the pattern is key to predicting the next number.

3.3 Pattern Recognition in Machine Learning

Pattern recognition is a crucial part of machine learning, much like solving a puzzle by recognizing shapes and colors. In machine learning, pattern recognition is about teaching a computer to spot patterns and make sense of them.

The multiple-choice questions (MCQs) about predicting the next number in a series are simple examples of pattern recognition. In each series, there's a hidden rule. If you can find the rule, you can predict the next number. This is similar to what we ask computers to do in machine learning, but with more complex patterns.

In machine learning, pattern recognition involves algorithms that look at data and learn to recognize patterns. This can be anything from spotting a cat in a photo to predicting what a customer will buy next. The machine looks at examples (data) and starts to see patterns, like how numbers increase in a series or what features make up a cat's face.

The link between those MCQs and machine learning is the concept of identifying a pattern. In the MCQs, you might recognize that numbers are doubling or following the Fibonacci sequence. In machine learning, an algorithm might learn to recognize that certain email messages are spam or which transactions are likely fraudulent.

But in machine learning, the patterns can be much more complex, and the data can be huge. That's why we need computers and programming. They can analyze vast amounts of data and find patterns that might be too subtle or complicated for a human to see.

Pattern recognition is essential in machine learning because it's the foundation of learning from data. Whether it's a simple number series or a complex problem like speech recognition, the ability to recognize patterns helps machines to learn and make decisions. This capability is at the heart of making machines smarter and more useful in our daily lives.

Chapter 4
Introduction to Machine Learning

Teaching Machines to Learn, One Algorithm at a Time

Abstract In the previous chapter, readers discovered how recognizing patterns underpins the foundation of machine learning. Building upon this, the current chapter provides an introduction to machine learning (ML), clearly differentiating between supervised and unsupervised learning—the primary focus areas of this book. Readers will explore industry specific ML applications spanning diverse sectors such as agriculture, education, energy, entertainment, finance, manufacturing, and healthcare, highlighting ML's wide-ranging impact and utility. Additionally, the chapter briefly introduces reinforcement learning and semi-supervised learning, offering learners a broader perspective of the ML landscape. To simplify complex concepts, everyday analogies and relatable examples illustrate how ML models and algorithms function, making the topic approachable even to beginners. This chapter thus sets the stage for deeper exploration of supervised and unsupervised methods, preparing readers effectively for subsequent chapters dedicated to these key ML approaches. Readers will also understand why choosing the right ML technique is crucial for solving real-world problems. Finally, the chapter encourages readers to appreciate the power and potential of machine learning in driving innovation and improving decision-making across various domains.

Keywords Machine learning · Spam filter · Supervised learning · Unsupervised learning · Semi supervised learning · Reinforcement learning · Application

4.1 An Introduction to Machine Learning

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn. Machine learning models are a class of computer algorithms designed to improve their performance on specific tasks by learning from data. This sets them apart from

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_4

traditional algorithms, where improvements typically involve changes to the software code.

4.1.1 The Example of Spam Filters

Consider the case of spam filters as an example of machine learning in action. Two decades ago, these filters struggled to accurately identify spam mails due to their limited exposure to examples and a lack of human-labeled data.

However, as the volume of spam emails increased, users marked them as junk or spam, providing valuable training data to machine learning algorithms.

Over time, these algorithms gained experience and improved their ability to distinguish between legitimate and unwanted emails. This exemplifies how machine learning models thrive on data, becoming more proficient with increased exposure and labeled examples.

They continuously refine their decision-making processes, making them invaluable tools in addressing complex problems across various domains (Fig. 4.1).

Machine learning algorithms are trained to find relationships and patterns in data. They use historical data as input to make predictions, classify information, cluster data points, reduce dimensionality, and even help generate new content, as demonstrated by new ML-fueled applications such as ChatGPT, Dall-E, and GitHub Copilot.

Fig. 4.1 Spam filters

4.2 Working of ML Models/Algorithms

As discussed earlier, ML models learn from experience. A computer program is said to learn from experience E with respect to a set of tasks T, if its performance measure P increases with increase in E.

Task (T): Image Classification—program's task is to correctly classify images into different categories, such as recognizing fruits in pictures.

Performance Measure (P): Classification Accuracy—measures the program's accuracy in correctly classifying images. It is defined as the ratio of correctly classified images to the total number of images.

Experience (E): Training Data E—consists of a large dataset of labeled images. Labeled images are images with annotations or references.

As the program gains more experience by training on a larger and more diverse dataset (E), its performance in accurately classifying images (P) typically improves for various tasks (T), such as identifying different types of fruits in images. This demonstrates that the program is learning from experience.

The process can be expressed as P(T) = f(E, T).

This function can be linear or non-linear depending on factors like the quality of the data, the complexity of the model, and the specifics of the task.

4.2.1 Understanding the Working of ML Models/Algorithms

A model is the core component of machine learning, and ultimately what we are trying to build. A model might estimate how old a person is from a photo, predict what you might like to see on social media, or decide where a robotic arm should move.

Models are simple functions (equations) you are already familiar with. Models in machine learning or any computational framework often take inputs and apply certain logic to produce outputs.

Imagine you have a model to estimate the daily calorie needs based on age, height, and weight. The logic would be 'multiply the age by parameter_1', 'multiply the weight by parameter_2', 'multiply the height by parameter_3', and 'adding the three values' to calculate the daily calorie needs.

- Input: Age (e.g., 30), Weight (e.g., 80 kg), and Height (e.g., 1.7 m).
- If parameter_1 has a value of 15, parameter_2 has a value of 20 and parameter_3 has a value of 300.
- Daily calorie needs = 30 * 15 + 80 * 20 + 1.7 * 300 = 2560.

This model takes age, weight, and height as input and uses the parameters to calculate the total calorie needs. The values of the parameters are finalized during the process of model building.

4.2.2 Selecting an ML Model

There are many model types, some simple and some complex. Which kind of model you should choose depends on your goal. For example, medical scientists often work with models that are relatively simple, because they are reliable and intuitive. In contrast, AI-based robots typically rely on complex models.

4.3 Different Types of Machine Learning

Machine learning can broadly be classified into four different categories:

1. Supervised learning.
2. Unsupervised learning.
3. Semi-supervised learning.
4. Reinforcement learning.

Let us explore each of these in brief.

4.3.1 Supervised Learning

Supervised learning is a category in which we feed labeled data (input data with annotations or references is known as labeled data) as input into the ML model. We also know the kind of output that we are expecting (Fig. 4.2).

These labels provide the algorithm with the correct answers for a given set of inputs. For example, in image classification, each image is labeled with a corresponding category (e.g., "apple"). The primary goal of supervised learning is to learn a mapping or a function from the input data to the correct output labels. The

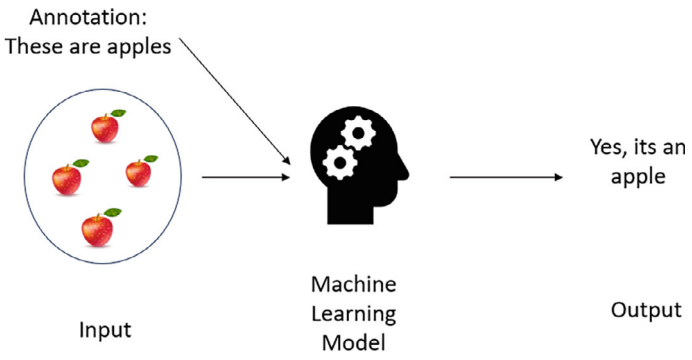


Fig. 4.2 Supervised learning

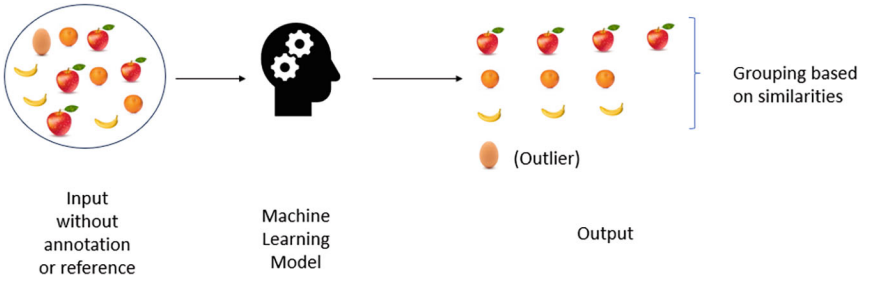


Fig. 4.3 Unsupervised learning

algorithm generalizes from the training data to make predictions or classifications on new, unseen data. It aims to find patterns, relationships, and decision boundaries that enable it to make accurate predictions.

Use Cases: Recommendation Systems, Object Detection, Image Classification.

4.3.2 Unsupervised Learning

Unsupervised learning is a category in which we only have input data to feed the model but no corresponding output data (Fig. 4.3).

The model is expected to discover hidden patterns, relationships, and structures within the input data. It focuses on unveiling the intrinsic characteristics of the data. In this case, the model has grouped fruits based on similarity into apples, oranges, and bananas. Since egg differs from fruits, it stands out in the output as an outlier or anomaly.

Use Cases: Clustering (Customer Segmentation), Dimensionality reduction, Anomaly Detection (Fraud Detection), Market Basket Analysis.

4.3.3 Semi-supervised Learning

It is a category in which the model combines small labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning and supervised learning (Fig. 4.4).

Use Cases: Semi-supervised learning is particularly valuable in situations where obtaining labeled data is costly, time-consuming, or impractical.

Fig. 4.4 Semi-supervised learning

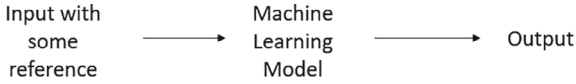


Fig. 4.5 Reinforcement learning

4.3.4 Reinforcement Learning

Reinforcement learning is a type of machine learning where an agent interacts with an environment. The agent takes actions in the environment, and the environment responds with rewards and new states. Reinforcement learning is often characterized by trial-and-error learning. The agent does not have access to labeled data or explicit instructions but must explore the environment, take actions, and learn from the consequences (Fig. 4.5).

Use Cases: Gaming, Robotics, Recommendation System.

4.3.5 Application of Supervised and Unsupervised Learning

Supervised and unsupervised learning techniques are extensively utilized across industries.

In the agriculture industry, supervised learning is utilized for predicting crop yields based on soil and weather data, while unsupervised learning is employed to identify different crop types from satellite imagery.

In education, supervised learning methods are used to create grading systems based on student performance data. In contrast, unsupervised learning is applied for grouping students according to their learning styles, facilitating customized teaching approaches. The energy sector uses supervised learning for forecasting power demand, which is critical for efficient grid management. Simultaneously, unsupervised learning is deployed for clustering energy consumption patterns, aiding in targeted marketing strategies.

In entertainment, supervised learning algorithms predict movie box office revenues, whereas unsupervised learning techniques identify viewer preferences to guide content production. Finance industries rely on supervised learning for developing credit scoring models to predict loan defaults. Concurrently, unsupervised learning is used for detecting unusual patterns in financial transactions to identify potential fraud.

Healthcare utilizes supervised learning to predict patient diagnoses from medical imaging data. In parallel, unsupervised learning is used to cluster patients with similar symptoms, enhancing understanding of health conditions. In manufacturing, supervised learning aids in predictive maintenance of machinery using sensor data. On the other hand, unsupervised learning analyzes patterns in production line data to optimize manufacturing processes.

Table 4.1 Supervised and unsupervised learning examples

Industry	Supervised learning example	Unsupervised learning example
Agriculture	Predicting crop yields based on soil and weather data	Identifying different crop types from satellite imagery
Education	Grading systems based on student's past performance data	Grouping students based on learning styles for customized teaching approaches
Energy	Forecasting power demand to manage grid resources efficiently	Clustering of energy consumption patterns for targeted marketing
Entertainment	Movie box office revenue prediction	Identifying viewer preferences to tailor content production
Finance	Credit scoring models to predict loan defaults	Detecting unusual patterns in financial transactions for fraud detection
Healthcare	Predicting patient diagnoses based on medical imaging data	Identifying patient clusters with similar symptoms
Manufacturing	Predictive maintenance of machinery based on sensor data	Identifying patterns in production line data to optimize processes
Retail	Product recommendation systems based on customer purchase history	Market segmentation using customer buying patterns
Technology	Speech recognition in virtual assistants	Natural language processing to understand themes in large text datasets
Transportation	Route optimization based on traffic data	Analyzing transportation patterns to improve city planning

The retail sector employs supervised learning for developing product recommendation systems based on customer purchase history, while unsupervised learning is used for market segmentation by analyzing customer buying patterns. Technology industries leverage supervised learning for speech recognition in virtual assistants. In contrast, unsupervised learning is applied for natural language processing and understanding themes in large text datasets.

Lastly, in transportation, route optimization is achieved through supervised learning based on traffic data, and unsupervised learning is used to analyze broader transportation patterns, aiding in city planning and improvement initiatives. Table 4.1 summarizes the examples explained above.

4.4 Focus of This Book

The author would like to emphasize that this book is going to focus on supervised and unsupervised machine learning only. Figure 4.6 summarizes the two main types of machine learning.

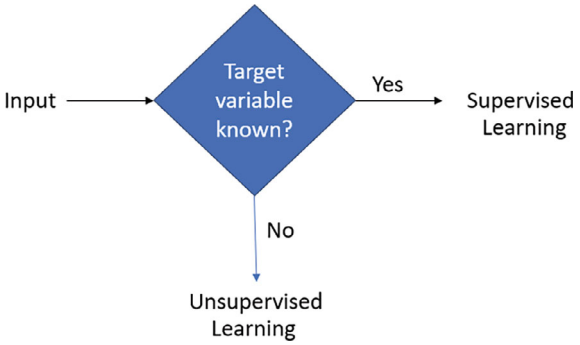


Fig. 4.6 Supervised and unsupervised machine learning

Chapter 5
Variables and Data Types

In the Dance of Data, Dependent Variables Follow the Lead of Their Independent Partners

Abstract This chapter introduces readers to the foundational elements of machine learning—variables and data types. Central to this discussion is the most fundamental equation in machine learning, $y = f(X)$, emphasizing the relationship between dependent (y) and independent (X) variables. Readers will clearly understand the distinction between dependent and independent variables through practical, real-world examples. Additionally, the chapter provides historical context behind the equation $y = f(X)$, illustrating its enduring significance in predictive modelling and analytics. Structured and unstructured data are briefly explored, enabling readers to differentiate between organized datasets versus data lacking predefined formats. Furthermore, the chapter introduces readers to numerical and categorical data types, crucial for informed data handling in machine learning. Through relatable examples, readers will grasp concepts of discrete and continuous numerical data, along with categorical types such as binary, nominal, and ordinal data. The importance of properly identifying data types before applying ML algorithms is emphasized to reinforce understanding. Readers will also learn why encoding non-numerical data into numerical form is essential for machine learning models to process and interpret information effectively. Practical scenarios demonstrating the need and methods for encoding categorical data are discussed. The chapter highlights common encoding techniques, preparing learners for detailed exploration in subsequent chapters. By the chapter's end, readers will appreciate how selecting appropriate data types and encoding strategies directly impact model accuracy and performance. This fundamental knowledge sets the stage for deeper exploration of feature engineering and data preprocessing in upcoming chapters. Ultimately, learners will be equipped with essential terminology and clarity required to handle diverse datasets in real-world ML projects. Mastering these concepts early will empower readers to confidently approach complex machine learning tasks.

Keywords Dependent variable · Independent variable · Function · Structured data · Unstructured data · Semi structured data · Meta data · Data types · Norminal data · Ordinal data · Binary data · Encoding · Time series data

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_5

5.1 Introduction

Machine learning models and algorithms get better with experience as they learn from 'data.' They also predict unseen or unknown values from a set of known values. They are called 'dependent variables' and 'independent variables' in mathematics and machine learning.

We will dive deeper and explore what exactly these variables are and what is the kind of data that goes into ML Models.

5.2 Independent and Dependent Variables

Let us take an example of The Impact of Watering Frequency on Plant Growth. In this case, the independent variable is the "watering frequency." Watering frequency is something that a gardener or an analyst can control and change. It refers to how often a plant is watered. The dependent variable is "plant growth." Plant growth is the outcome that machine learning professionals are interested in understanding, and it depends on various factors like watering frequency.

Machine learning professionals may hypothesize that the frequency of watering will have an impact on the growth of a plant. They might predict that plants that are watered more frequently will exhibit better growth compared to those that are watered less often.

Therefore, the dependent variable (say y) can be viewed as a function of the Independent Variable (say X). We will now end up with the equation $y = f(X)$ which is a measure of how much y changes with respect to X, i.e., how much does watering a plant affects its growth.

5.3 Mathematical Understanding of Variables

- Representation of Unknowns: Variables are used in algebraic equations to represent unknown quantities that need to be determined.
- Alphabet Range: Variables can take on any alphabet from 'a' to 'z,' although certain letters like 'x' and 'y' are more frequently used in equations.
- Arithmetic Calculations with Variables: Performing arithmetic calculations with variables allows for solving a wide range of problems in a single equation. For instance, quadratic equations can be solved by substituting numerical values for the coefficients into the variables representing them.
- Significance in Mathematics: The concept of a variable is fundamental in mathematics. Many mathematical functions are expressed as $y = f(X)$, where 'y' represents the value of the function, and 'X' represents the argument or input to the function. When the argument (variable 'X') changes, the value ('y') also

changes accordingly, illustrating the dynamic nature of variables in mathematical relationships.

5.4 History of the Function $y = f(X)$

The history of the function $y = f(X)$ is closely tied to the development of mathematics and its applications throughout human history. The concept of functions and their notation has evolved over centuries, and it has played a fundamental role in various scientific and engineering fields, now extended even to machine learning.

The articles '*Functions: How they have changed through History*' (Tall 1992) and '*The Function Concept: A Historical Perspective*' (Dhombres 2005) trace the history of the concept of functions, starting with the Babylonians who recorded tables of functions without recognizing the relationships between the numbers and their squares or cubes. It goes on to discuss the contributions of ancient mathematicians like Ptolemy, Oresme, Galileo, Descartes, Newton, and Leibniz in understanding and defining functions in different ways.

In the fourteenth century, Oresme made significant contributions that laid the foundation for our understanding of functions. He described natural laws in terms of relationships between different quantities. Importantly, he introduced the concept of latitudes, which was a pioneering way to think about dependent variables and how they vary in relation to others. This early insight into variable relationships was a crucial step forward in the mathematical and scientific understanding of functions.

Galileo, through his studies of motion and mapping between concentric circles, demonstrated a clear understanding of the connection between variables, showcasing an early appreciation for the mathematical concept of functions.

Descartes, in his work on algebraic geometry, emphasized the dependence between variable quantities represented by curves and introduced the derivative, advancing the understanding of functions by providing a means to find tangents to curves.

Newton's pioneering work in calculus extended the concept of functions with infinite power series (Fig. 5.1).

Leibniz played a pivotal role in advancing our comprehension of functions. He was the first to introduce the term "function" and introduced symbolic notation, which marked a significant evolution in mathematical language. In his approach, he viewed functions as dependencies of geometrical quantities on the shapes of curves. This perspective opened up new ways of understanding mathematical relationships. Furthermore, in his influential paper "De geometria recondita," Leibniz introduced terms such as "constant," "variable," and "parameter." These terms provided a more nuanced language to describe different aspects of functions, enriching the concept and its application in mathematics. His contributions were instrumental in shaping the modern understanding of functions and their role in mathematical analysis.

Euler's work, which followed those of his predecessors, marked another significant advancement in the field. He stated that "*If some quantities so depend on other*

Fig. 5.1 Functions with infinite power series

$$1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{n=0}^{\infty} x^n$$

$$2 + 2x + 2x^2 + 2x^3 + 2x^4 + \cdots = \sum_{n=0}^{\infty} 2x^n$$

$$1 + \frac{x}{3} + \frac{x^2}{9} + \frac{x^3}{27} + \frac{x^4}{81} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{3^n}$$

Fig. 5.2 Absolute value function

$$y = |x| = \begin{cases} x & \text{for } x \geq 0 \\ -x & \text{for } x < 0 \end{cases}$$

quantities that if the latter are changed, the former undergoes change, then the former quantities are called functions of the latter."

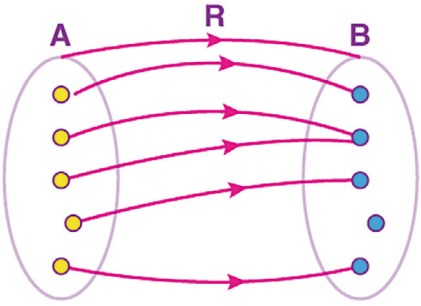
The clearest example of such a function was given by Cauchy in 1844 when he noted that the absolute value function (Fig. 5.2).

5.4.1 Functions in Modern-Day Mathematics

A function is a vital mathematical concept that establishes a clear association between elements in one set (the domain, set *A*) and elements in another set (the codomain, set *B*). This relationship is defined by a rule or process, ensuring that each input from set A corresponds uniquely to an output in set *B* (Fig. 5.3).

Consequently, elements in set *A* are referred to as independent variables, while elements in set B are dependent variables.

Fig. 5.3 Functions representing relationships



Functions can be represented in various notations, such as $f(x)$, $f: A \rightarrow B$, or as subsets of the Cartesian product $A \times B$. These notations convey the same essential information about the mapping between sets.

Functions are a foundational concept in mathematics, applied across numerous disciplines like calculus, linear algebra, and number theory. They serve as powerful tools for modeling real-world phenomena, solving equations, and comprehending complex problems of today.

5.5 Variables in Machine Learning

The concept of dependent and independent variables has been essential for the development of machine learning. In machine learning, we are often interested in predicting a target variable (dependent variable) based on a set of input variables (independent variables).

One of the most common types of machine learning algorithms is supervised learning. Supervised learning algorithms learn from a set of training data, where each data point consists of input variables and a target variable. The goal of the algorithm is to learn a function that can predict the target variable for new data points. An example would be predicting the price of a house (y) based on its size (x) (Fig. 5.4).

The concept of functions is also essential for most types of machine learning algorithms. For example, in unsupervised learning (Obulesu et al. 2018), we might use functions to cluster data points into groups or to identify patterns in data. In reinforcement learning, we might use functions to represent the state of an environment and to predict the rewards for taking different actions.

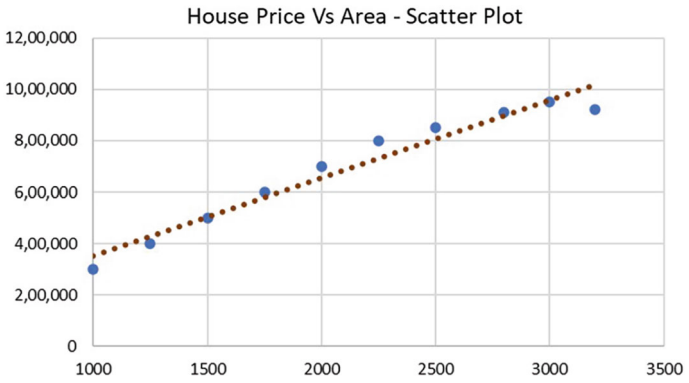


Fig. 5.4 House price prediction

5.6 Visualizing Variables Using Graphs

Graphs are a valuable tool for visualizing the relationships between variables. In traditional graphs, independent variables are typically placed on the horizontal *X*-axis, while the dependent variable is plotted on the vertical *Y*-axis. The type of graph used depends on the nature of the variables, such as bar charts for categorical predictors and scatterplots for continuous predictors.

The graphs depict the relationship between the independent variables with the dependent variables. The box plot (visual on the left) shows the relationship between teaching method and score. The scatter plot (visual on the right) shows the relationship between height and weight (Fig. 5.5).

5.7 How to Identify Independent Variables and Dependent Variables?

Independent variables cause changes in another variable. The researchers and analysts control the values of the independent variables. They are controlled or manipulated variables. Experiments often refer to them as factors or experimental factors. In areas such as medicine, they might be risk factors.

Treatment and control groups are always independent variables. Dependent variables get changed due to other variables. Dependent variables are the outcomes we look for in ML Models.

5.8 Industrial and Modern-Day Applications in Machine Learning

Fraud Detection: The dependent variable classifies transactions as "fraudulent" or "legitimate". Transaction data, customer behavior, location, and historical patterns serve as independent variables that are used to identify fraudulent activities.

Sentiment Analysis: In sentiment analysis, the dependent variable classifies text as "positive," "negative," or "neutral." Independent variables include text data, word frequency, and sentiment indicators, helping models predict sentiment.

Credit Scoring: Credit scoring's dependent variable assesses creditworthiness with credit scores. Independent variables involve financial data, income, credit history, and demographics, enabling risk assessment.

Churn Prediction: Churn prediction's dependent variable identifies customers as "churn" or "no churn." Independent variables encompass behavior, transactions, demographics, and usage data, guiding retention efforts.

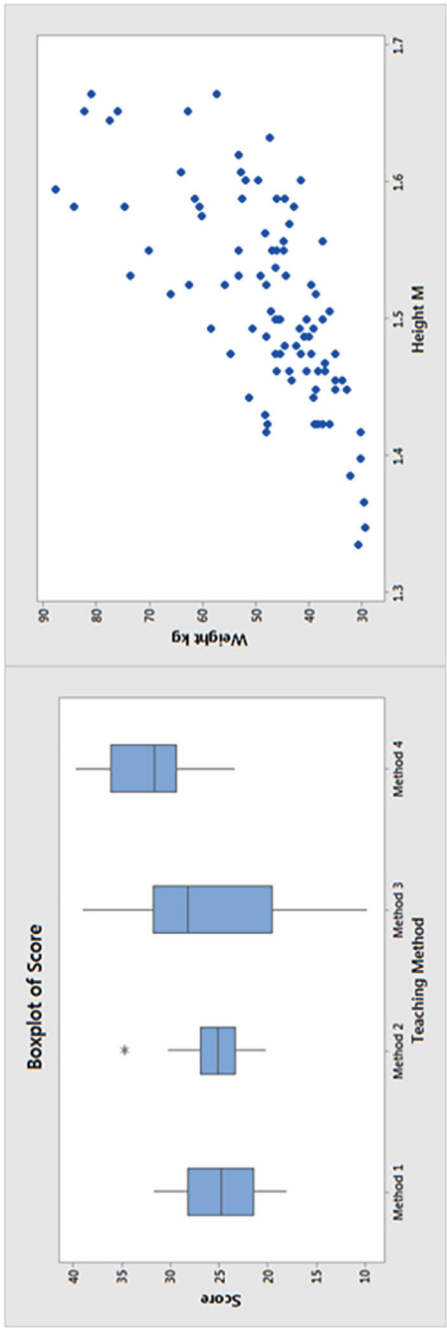


Fig. 5.5 Box plot and scatter plot

Table 5.1 Dependent and independent variables

Area	Dependent variable	Independent variables			
Fraud detection	Transaction class (fraudulent/ legitimate)	Transaction amount, time	Customer purchase history, account age	Geographical location, IP address	Historical transaction patterns, merchant category
Sentiment analysis	Text sentiment (positive/negative/ neutral)	Text data, N-gram analysis	Sentiment score, contextual polarity	Language tone, keyword analysis	
Credit scoring	Credit score (creditworthiness)	Account balance, monthly expenditure	Annual income, employment status	Repayment history, current debts	Credit utilization ratio, length of credit history
Churn prediction	Customer status (churn/no churn)	Account usage frequency, service usage patterns	Customer support interactions, plan changes	Demographics (age, gender)	Product holdings, billing cycles
Stock price prediction	Future stock price (numerical value)	Stock trading volume, closing prices	Market trends, trading indicators	Financial news sentiment, inflation rates	

Stock Price Prediction: The dependent variable forecasts future stock prices (numerical values). Independent variables encompass historical stock data, market sentiment, economic indicators, aiding investment decisions.

Table 5.1 summarizes the above examples for easy reference.

5.8.1 Data Types in Machine Learning

The availability and nature of data are indeed critical factors when constructing machine learning models and data-driven real-world systems (Jena and Jena 2019). Data can come in various forms, including structured, unstructured, semi-structured, and metadata, each with its own characteristics and challenges.

5.8.2 Structured Data

Structured data has a well-defined and organized format, following a standard data model with a consistent order. It is highly organized and easily accessible. Structured data is typically stored in a tabular format, like tables in relational databases or Excel. Examples of structured data include information like names, dates, addresses, credit card numbers, stock prices, geolocation coordinates, and more. These data types are easily organized into rows and columns.

5.8.3 Unstructured Data

Unstructured data lacks a predefined format or organization. It is much more challenging to capture, process, and analyze because it often contains textual and multimedia content. Unstructured data can include sensor data, emails, blog entries, word processing documents, PDF files, audio recordings, videos, images, presentations, web pages, and a wide range of other business documents. Deep learning algorithms are more suited for handling unstructured data. Deep learning is considered as a more advanced form of machine learning. Deep learning is covered in the latter sections of this book.

5.8.4 Semi-structured Data

Semi-structured data is not stored in a relational database but possesses certain organizational properties that make it more manageable for analysis. Examples of semi-structured data formats include HTML, XML, JSON documents, and data stored in NoSQL databases. While not as rigidly structured as tabular data, they have some level of organization.

5.8.5 Metadata

Metadata is not data in the traditional sense but rather "data about data." It provides information that describes and gives context to the actual data, making it more valuable to users. Metadata for a document might include information such as the author's name, file size, date of creation, keywords used to categorize the document, and other attributes that help users understand and manage the data.

For the purpose of machine learning, data can also be classified as shown in Fig. 5.6 (Alina 2018).

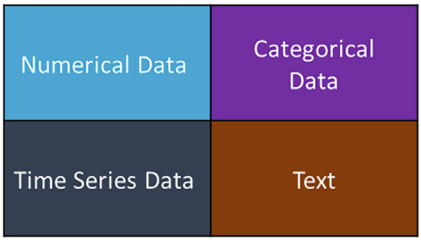


Fig. 5.6 Classification of data

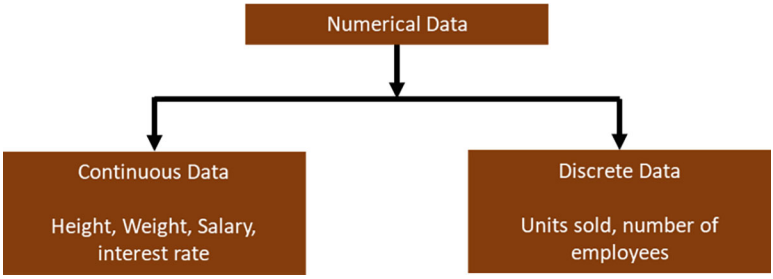


Fig. 5.7 Classification of numerical data

Numerical data is any data where data points are numbers. They can be continuous or discrete. Unlike discrete data, continuous data allows for infinitely fine subdivisions. Height can be represented as 169 cm or 1.69 m. On the contrary, number of employees can only be a finite number like 145 (Fig. 5.7).

Categorical data represents discrete categories or labels and is encoded as integers in the machine learning process. Encoding is necessary because machine learning models work only with numeric data. Thus, machine learning models require special handling, like one-hot encoding, to work effectively with categorical features. There are two types of categorical data, namely ordinal and nominal data (Fig. 5.8).

Ordinal data is a type of data where the order matters but the difference between values does not have a consistent meaning. For example, if you have a survey with responses like 'Strongly Agree,' 'Agree,' 'Neutral,' 'Disagree,' and 'Strongly Disagree,' this is ordinal data. Here, the order of the responses is important, but the difference between 'Strongly Agree' and 'Agree' might not be the same as the difference between 'Neutral' and 'Disagree.'

Nominal data, on the other hand, is about categories with no specific order. For instance, if you are categorizing people by their favorite color (like red, blue, green), this is nominal data. The colors are just different categories and there's no order or ranking among them. Each color is just a different, separate group without any implied sequence. Table 5.2 summarizes the different types of data across industries.

In addition to ordinal and nominal data, an important type to handle non-numeric data is binary. Binary has two options namely yes or no, go versus no-go, good or

Fig. 5.8 One-hot encoding

bad. Table 5.2 provides examples for different types of data across industries, which are relevant for machine learning.

Text data consists of textual content, such as sentences or documents. Natural language processing (NLP) techniques are used to process and analyze text data in machine learning, enabling tasks like sentiment analysis and text classification.

Time series data is a sequence of data points collected at successive time intervals. Machine learning models for time series forecasting and analysis consider the

Table 5.2 Types of data

Industry	Discrete	Continuous	Binary	Nominal	Ordinal
Finance	Number of transactions per day	Stock prices over time	Loan default (yes/no)	Types of currencies (USD, EUR, GBP)	Credit rating (poor, fair, good, excellent)
Healthcare	Number of patients per clinic	Patients' weight or blood sugar levels	Disease presence (positive/negative)	Blood types (A, B, AB, O)	Pain severity scale (no pain, mild, moderate, severe)
Retail	Items sold per day	Time spent on a shopping website	Purchase made (yes/no)	Product categories (electronics, clothing, furniture)	Customer satisfaction rating (1–5 stars)
Education	Number of students in a class	Time spent on homework	Pass/fail in an exam	Type of degree (arts, science, commerce)	Grade level (freshman, sophomore, junior, senior)
Marketing	Responses to a survey	Duration of a marketing campaign	Clicked ad (yes/no)	Campaign types (email, social media, TV)	Interest level (not interested, somewhat interested, very interested)
Real estate	Number of properties sold	Property sizes in square feet	Property available (yes/no)	Types of buildings (residential, commercial, industrial)	Condition of property (poor, fair, good, excellent)
Transportation	Number of daily flights	Distance traveled by a vehicle	On-time arrival (yes/no)	Types of vehicles (car, bus, train, plane)	Seating class (economy, business, first class)
Technology	Number of users accessing an app	Battery life duration	System error occurred (yes/no)	Operating system types (Windows, macOS, Linux)	User proficiency level (beginner, intermediate, advanced)

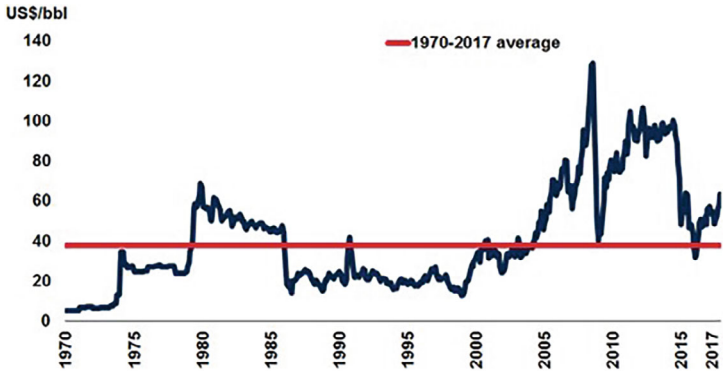


Fig. 5.9 Time series data of oil prices

temporal order of the data to make predictions or uncover patterns. Figure 5.9 image shows the average price of oil from 1970 to 2017 (source: World Bank).

References

Alina Z (2018) Data types from a machine learning perspective, towards data science

Dhombres J (2005) The function concept: a historical perspective. Springer

Jena SK, Jena AK (2019) Machine learning: algorithms, real-world applications and research directions. CRC Press

Obulesu O, Mahendra M, ThrilokReddy M (2018) Machine learning techniques and tools: a survey. In: Proceedings of the 2018 international conference on inventive research in computing applications (ICIRCA)

Tall DO (1992) Functions: how they have changed through history. In: Tall DO (eds) Mathematical concepts and methods in the mathematical sciences. Kluwer Academic Publishers, pp 1–28

Chapter 6
Descriptive Statistics

Descriptive Statistics: The Art of Telling Stories Through Numbers

Abstract Building upon the understanding of variables and data types from the previous chapter, this chapter introduces readers to descriptive statistics, a crucial step in summarizing and interpreting data effectively before applying machine learning models. Readers will explore how descriptive statistics have evolved, becoming foundational tools in extracting meaningful insights from data. Measures of central tendency – mean, median, and mode – are introduced clearly, highlighting their strengths and limitations in representing typical data points. Additionally, readers will learn about measures of dispersion such as range, variance, and standard deviation, which reveal the variability inherent in datasets. The chapter further explains skewness and kurtosis, enabling learners to assess the shape and characteristics of data distributions. Important techniques for detecting outliers are discussed, helping readers improve data quality before modeling. Concepts like percentiles and quartiles are introduced to deepen readers' skills in data segmentation. Lastly, an insightful discussion explains why averages can sometimes be misleading, emphasizing the importance of selecting appropriate statistical measures to accurately interpret real-world data. Readers will also understand how descriptive statistics significantly influence the performance and accuracy of ML models by ensuring data integrity – an essential aspect often overlooked in practice.

Keywords Descriptive statistics · Measures of central tendency · Measures of dispersion · Mean · Median · Mode · Standard deviation · Percentiles · Quartile · Skewness · Kurtosis · Misleading averages · Outliers · Interquartile range · IQR · Shapes of distribution · *Z* score · Normal distribution · Skewed distribution · Uniform distribution · Bimodal distribution · Multimodal distribution · Pareto distribution · Exponential distribution

In machine learning, it is important that you understand the data before developing the model. Only if one is aware of the intricacies and properties of the data can they choose the right model, make informed decisions, and get better and accurate results.

---

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_6

This is where descriptive statistics come into play.

6.1 An Introduction to Descriptive Statistics

Descriptive statistics is a branch of statistics that is dedicated to summarizing and presenting data in a clear and concise manner. Its primary objective is to describe and analyze the fundamental features and properties of a dataset without making broader inferences or conclusions about a larger population. Descriptive statistics offers a comprehensive overview of the data, enabling machine learning professionals and analysts to uncover insights, detect patterns, and grasp the distribution of values within the dataset.

Descriptive statistics also includes the use of visual representations such as charts, graphs, and tables to aid in the visualization and interpretation of the data. Common graphical tools include histograms, bar charts, pie charts, scatter plots, and box plots. Descriptive statistics provide a deeper understanding of the dataset leading to an informed decision-making process.

6.2 Evolution of Descriptive Statistics

The earliest known examples of descriptive statistics come from ancient Egypt and Mesopotamia, where they were used to track agricultural production and population growth. For example, the Ebers Papyrus (c. 1550 BC) contains a table of data on the average lifespan of people in different regions of Egypt.

During the middle ages, descriptive statistics were used by merchants and traders to track their profits and losses, and by governments to collect taxes and raise armies. For example, the Domesday Book (1086), a survey of all land and property in England, contains a wealth of descriptive statistics on population, land ownership, and livestock.

There was a renewed interest in descriptive statistics due to the rise of scientific inquiry during the Renaissance period. The Italian physician Girolamo Cardano (1501–1576) wrote a book on probability theory, which included a discussion of descriptive statistics.

Probability theory was further developed in the eighteenth century, leading to the development of new statistical methods that allowed researchers to make inferences about populations based on samples. For instance, the English mathematician Thomas Bayes (1702–1761) developed Bayes' theorem, which is a key concept in Bayesian statistics (Cambridge University Press 2009).

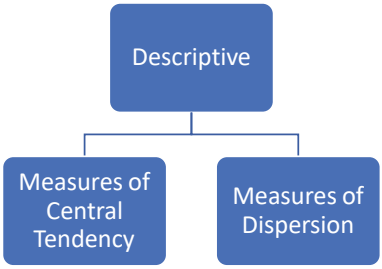
New statistical methods were developed in the nineteenth century, such as correlation and regression analysis, which allowed researchers to identify relationships

between variables. The Belgian astronomer and statistician Adolphe Quetelet (1796–1874) developed the correlation coefficient, which is a measure of the strength of the relationship between two variables (Galton 1889).

New statistical software was developed and computer-aided statistics became popular during the twentieth century, making it possible to analyze large and complex datasets. For example, the Statistical Package for the Social Sciences (SPSS) was released in 1968 (Pearson 1896; Fisher 1925).

There has been growing interest in descriptive statistics over centuries. In today's world, it serves as a vital tool to analyze complex data types in fields like machine learning, data analytics, social media analysis among various others to create appealing visualizations of data.

Descriptive statistics describes and summarizes data to provide insights into its main features and characteristics through two key measures—measures of central tendency and measures of dispersion or variance.



6.3 Measures of Central Tendency

'Central tendency' summarizes a dataset with a single value representing its center. Three primary measures are used.

Mean: It is the sum of observations divided by the total number of observations. It is also defined as average which is the sum divided by count. Mean is susceptible to outliers, as extreme values can skew it.

Median: It is the middle value in a dataset and is thus moderately resistant to outliers and skewed data. For an odd-sized dataset, it is the central element; for an even-sized dataset, it is the average of the two central elements.

Mode: The most frequently occurring value. A dataset may have no mode if all values are unique or more than one mode if multiple values occur with the same highest frequency. Mode is suitable for categorical variables (Fig. 6.1).

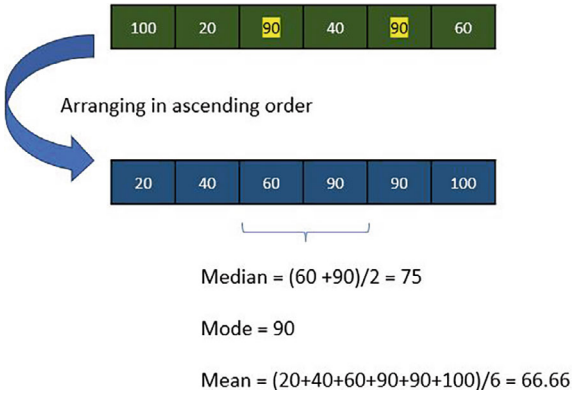


Fig. 6.1 Understanding mean, median, and mode

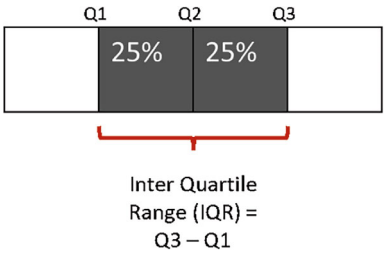


Fig. 6.2 Quartiles

6.4 Measures of Dispersion

Measures of dispersion, also known as spread of the data, describe how similar or varied are the set of observations. The most popular variability measures are the range, interquartile range (IQR), variance, and standard deviation.

Range: The range describes the difference between the maximum and the minimum points in your data. The bigger the range the more spread out is the data.

IQR: The interquartile range (IQR) is a measure of statistical dispersion between upper (75th) quartiles, i.e., Q3 and lower (25th) quartiles, i.e., Q1 (Fig. 6.2).

6.5 Percentiles and Quartiles

Percentile is a ranking mechanism and differs from percentage. Concept of percentile is used to rank students in competitive examinations like GRE, GMAT, CAT. The Pth percentile in the ordered set is that value below which lies $P\%$ (P percent) of the observations in the set. The position of the Pth percentile is given by $(n + 1)\,P/100$, where n is the number of observations in the set.

Quartiles are special names to percentiles.

Q1 = 25th percentile
Q2 = 50th percentile = median
Q3 = 75th percentile

In a dataset containing (6, 5, 2, 1, 4, 3, 7, 9, 8), the values of the quartiles are as follows:

Q1 (1st Quartile): 3, which means that 25% of the data is less than or equal to this value.
Q2 (2nd Quartile or Median): 5
Q3 (3rd Quartile): 7, indicating that 75% of the data is less than or equal to this value.
Q4 (4th Quartile or Maximum): 9. Fourth quartile is essentially the maximum value.

Variance: It is the average squared deviation from mean. The variance is computed by finding the difference between every data point and the mean, squaring them, summing them up, and then taking their average.

Standard Deviation: It is a statistical measure that quantifies the amount of variation or dispersion in a set of data points. Standard deviation is simply the square root of the variance (Fig. 6.3).

The above visual indicates how individual data points deviate from the mean (average) of the data. A measure to describe this deviation is standard deviation.

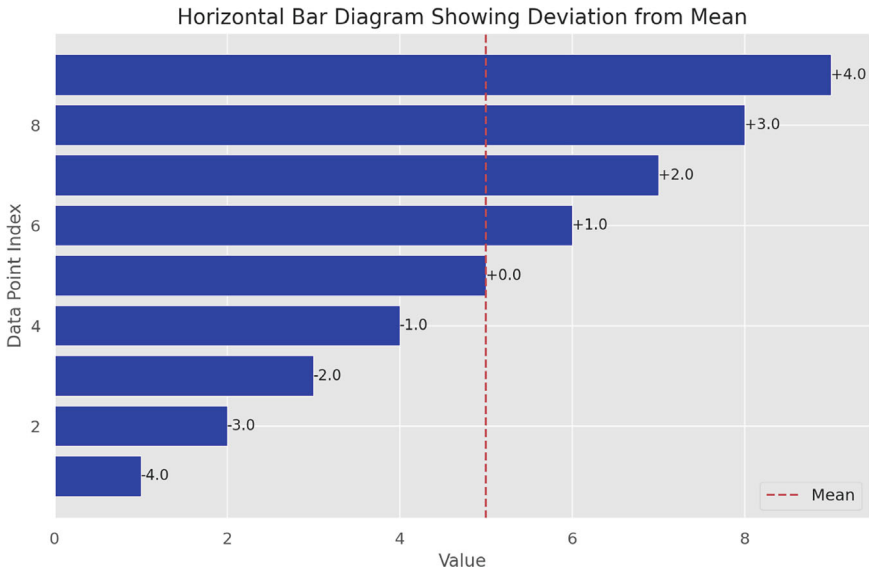


Fig. 6.3 Standard deviation

A higher standard deviation suggests greater variability, while a lower one indicates more consistency or less spread points in the data (Fig. 6.4).

Coefficient of Variation (CV): It is a dimensionless relative measure of dispersion that is defined as the ratio of the standard deviation to the mean. If there are datasets that have different units then the best way to draw a comparison between them is by using the coefficient of variation. The concept of population and sample is covered in the next chapter (Fig. 6.5).

For example, if stock A has an average annual return of 8% with a standard deviation (a measure of volatility) of 2%, and another stock B has an average annual return of 12% with a standard deviation of 4%, the CV can be used to determine which stock offers a better risk-to-return ratio. Despite the higher average return, the second stock B has a CV of 33% compared to the first's 25%, indicating that the first stock A offers a more desirable balance of risk to reward. Higher CV value indicates higher volatility. Lower volatility suggests a more stable investment outlook.

Skewness: Skewness characterizes the degree of asymmetry of a distribution around its mean (Fig. 6.6).

Skewness is calculated using the formula. This is also known as quartile skewness coefficient or Bowley's skewness coefficient.

$$\text{Skewness} = \frac{(Q_3 - Q_2) - (Q_2 - Q_1)}{(Q_3 - Q_1)}$$

Fig. 6.4 Formula for variance and standard deviation)

$$Variance = \sigma^2 = \frac{\sum (x - \mu)^2}{n}$$

$$Standard\ Deviation = \sigma = \sqrt{\frac{\sum (x - \mu)^2}{n}}$$

$$\mu = \text{Mean}$$

Fig. 6.5 Coefficient of variation and standard deviation

$$CV = \frac{\sigma}{\mu} * 100$$

$$CV\ for\ stock\ A = \frac{2}{8} * 100 = 25\%$$

$$CV\ for\ stock\ B = \frac{4}{12} * 100 = 33\%$$

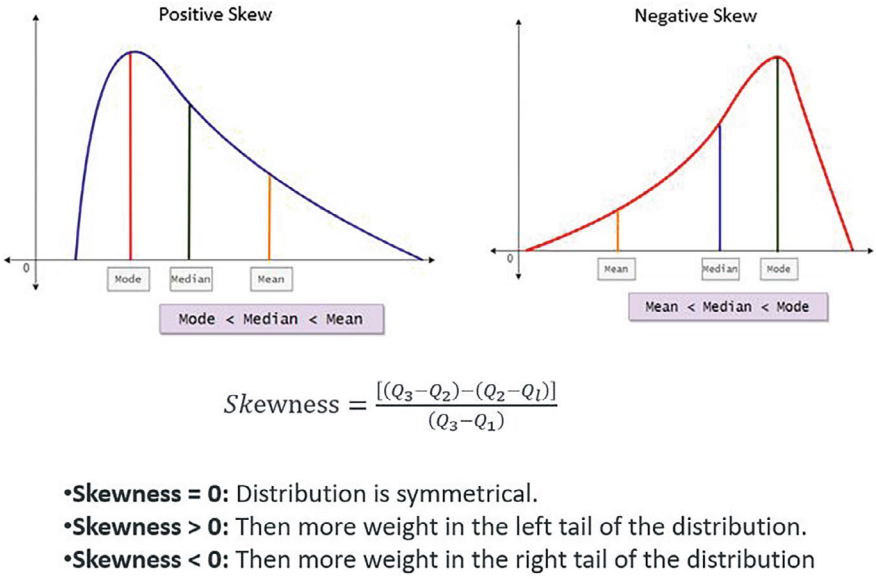


Fig. 6.6 Skewness

The resulting values range from − 1 to + 1, where a negative value indicates a negative skewness, a positive value indicates a positive skewness and a value of zero indicates a symmetric distribution.

The concept of skewness is mainly used to understand the distribution of the data and steps taken to normalize the data for further building of machine learning models.

In case of negatively skewed data, Mean < Median < Mode. This indicates, more data points are to the right of the curve where the data has very high values in large numbers.

In case of positively skewed data, Mode < Median < Mean. This means that more data points are to the left of the curve where the data has very low values in large numbers.

Kurtosis: Kurtosis characterizes the symmetric distribution through the relative peaks or flatness of the curve. There are three types of kurtosis.

$$\text{Kurtosis} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \tilde{x}}{\text{s}}\right)^4$$

where n is the number of observations, x_i is the individual observation, $\tilde{x}$ is the mean, and s is the standard deviation (Fig. 6.7).

- Platykurtic (relatively flat)—Kurtosis value < 3.
- Mesokurtic (normal)—Kurtosis value of 3.
- Leptokurtic (relatively peaked)—Kurtosis value > 3 (Fig. 6.7).

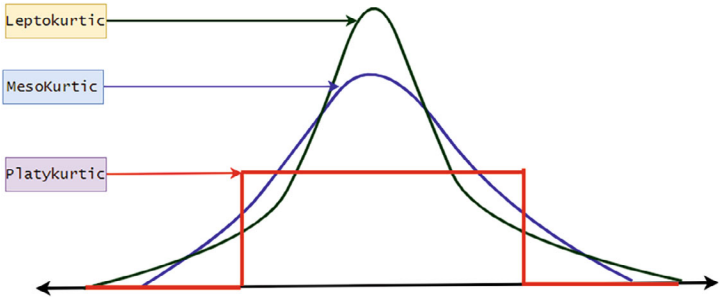


Fig. 6.7 Kurtosis

While skewness measures the asymmetry of the data distribution, kurtosis measures the "tailedness" of the distribution. While skewness focuses on the direction and extent of asymmetry, kurtosis is concerned with the extremity of data points—or outliers. Both contribute to understanding the normality of the data, as a normal distribution has a skewness of zero and a kurtosis of three (mesokurtic). Deviations from these values suggest a departure from normality, which can affect statistical analyses and interpretations.

6.6 The Four Moments of a Distribution

The shape of the distribution is best understood by the study of the four measures—mean, variance, skewness, and kurtosis. These four measures are called the four moments of a distribution. The term "moments" in statistics is derived from the concept of moments in physics. In physics, a moment is a quantitative measure of the shape of a set of points.

First Moment—Mean: The first moment is the mean or average of the data. It provides a measure of the central location of the distribution.

Second Moment—Variance: The second moment is the variance. It measures the spread or dispersion of the data points around the mean. The square root of the variance is the standard deviation.

Third Moment—Skewness: The third moment is skewness. It measures the asymmetry of the distribution. A distribution with a longer or fatter tail on the right side of the mean has positive skewness, while one with a longer or fatter tail on the left has negative skewness.

Fourth Moment—Kurtosis: The fourth moment is kurtosis. It measures the "tailedness" of the distribution, indicating the presence of outliers. Higher kurtosis means more of the variance is due to infrequent extreme deviations, as opposed to frequent modestly sized deviations.

Table 6.1 Income in two regions

Metric	Region A (incomes in $)	Region B (incomes in $)
	40,000	10,000
	50,000	20,000
	60,000	70,000
	70,000	100,000
	80,000	200,000
Average income	60,000	80,000
Income range	40,000–80,000	10,000–200,000

6.7 Why Averages Can Be Misleading in Some Cases?

Let us consider the income data for two regions, Region A and Region B. In Region A, incomes range from $40,000 to $80,000. The average income here is $60,000. In Region B, incomes are more spread out, from $10,000 to $200,000, with an average income of $80,000 (Table 6.1).

By looking at the average incomes, you might think that people in Region B earn more than those in Region A. Is the conclusion really true? The average doesn't show how the incomes are spread. In Region B, some people earn very little, and some earn a lot, which is a big difference not seen in the average. That is why averages can be misleading.

An analysis must include measures of both central tendency and dispersion. They give a full picture. The income range shows the spread. Region A has a smaller range, meaning incomes are more similar. Region B has a big range, showing a big gap between low and high incomes. So, studying both the average and the range helps us to understand not just what the typical income is, but also how incomes vary. This gives us a clearer and more complete view of the economic situation in each region.

6.8 Descriptive Statistics in Machine Learning

Machine learning engineers and data scientists commonly use machine learning and techniques to solve modern-day problems. They apply descriptive statistical analysis to gain insights, including range, skewness, median, and mode. These insights drive both supervised and unsupervised discretization methods.

Descriptive statistics are used in machine learning in a variety of ways:

Understanding the data: Descriptive statistics can be used to understand the basic properties of the data, such as the central tendency, variability, and distribution of the data. This information can be used to identify outliers, clean the data, and select the appropriate machine learning model.

Feature engineering: Descriptive statistics can be used to create new features from the existing data. For example, the mean, median, and standard deviation of a feature can be used to create new features that describe the distribution of the data that improves the performance of a model.
Model evaluation: Descriptive statistics can be used to evaluate the performance of machine learning models. For example, the accuracy, precision, recall, and $F1$-score (about which we will see in detail subsequently) can be used to measure the performance of a classification model.
Model selection: Descriptive statistics can be used to select the best machine learning model for a given dataset.

6.8.1 Examples of a Few Use Cases in Today's World

A fraud detection model may use descriptive statistics to identify transactions that are likely to be fraudulent. The model may look for transactions that are much larger or smaller than the average transaction, or transactions that are made in unusual locations.

A recommendation system may use descriptive statistics to identify products that a user is likely to be interested in. The system may look at the user's past purchase history and recommend products that are like the products they have purchased in the past.

A medical diagnosis model may use descriptive statistics to identify patients who are likely to have a certain disease. The model may look at the patient's medical test results and compare them to the average test results for patients with the disease.

6.9 Outliers in Machine Learning

Outliers in machine learning are data points that are significantly different from the rest of the dataset. They can be caused by a variety of factors, such as measurement errors, data entry errors, or natural variations in the data.

Outliers can have a negative impact on machine learning models. For example, if a model is trained on a dataset with outliers, it may learn to focus on those outliers rather than the underlying patterns in the data. This can lead to the model making inaccurate predictions on new data.

A few examples of outliers:

- A house with a price that is much higher or lower than the other houses in the dataset.
- A customer with a purchase history that is very different from the other customers in the dataset.
- A sensor reading that is much higher or lower than the other sensor readings in the dataset.

- A medical test result that is much higher or lower than the other medical test results in the dataset.

It is important to detect and remove outliers from data before training a machine learning model. This can be done using a variety of methods, such as statistical methods, clustering algorithms, and anomaly detection algorithms.

6.10 Methods to Detect Outliers

6.10.1 Standard Deviation Method

The standard deviation method assumes that the data is normally distributed. Let us denote the standard deviation of the distribution by σ, and the mean by μ.

This approach involves setting the lower limit to three standard deviations below the mean ($\mu - 3 * \sigma$), and the upper limit to three standard deviations above the mean ($\mu + 3 * \sigma$). Any data point that falls outside this range is detected as an outlier.

We can accomplish this programmatically using the below code.

```
lower_limit = df.mean() - 3*df.std()
upper_limit = df.mean() + 3*df.std()
print(lower_limit)
print(upper_limit)
df_filtered = df[(df['score'] > lower_limit) & (df['score'] <
upper_limit)]
print(df_filtered)
```

6.10.2 Z-Score Method

The z-score method is similar to the standard deviation method. For a normal distribution with mean μ and standard deviation σ, the z-score for a value x in the dataset is given by (Fig. 6.8)

$$z = (x - \mu)/\sigma.$$

From the above equation, we have the following:

- When $x = \mu$, the value of z-score is 0.
- When $x = \mu \pm 1$, $\mu \pm 2$, or $\mu \pm 3$, the z-score is $\pm$ 1, $\pm$ 2, or $\pm$ 3, respectively (Fig. 6.8).

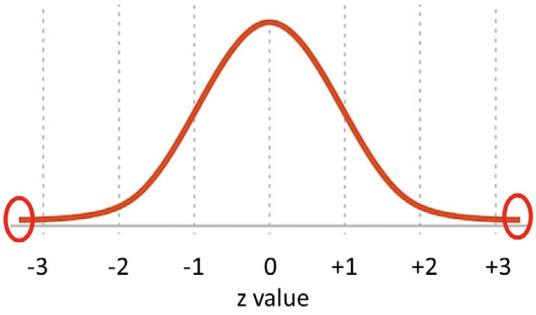


Fig. 6.8 Outliers based on Z value beyond $\pm$ 3

This technique is equivalent to the scores based on standard deviation we had seen earlier. Under this transformation, all data points that lie below the lower limit, $\mu - 3 * \sigma$, now map to points that are less than $-$ 3 on the z-score scale.

Similarly, all points that lie above the upper limit, $\mu + 3 * \sigma$ map to a value above 3 on the z-score scale. Points that lie outside this range are considered outliers.

We can accomplish this programmatically using the below code.

```
df['z_score'] = (df['score'] - df['score'].mean()) / df['score'].std()
df.head()
```

6.11 Interquartile Range (IQR) Method

The IQR method is a nonparametric method for outlier detection. It does not require the data to be normally distributed. We first calculate the first quartile (Q1) and third quartile (Q3) of the data. The IQR is then calculated as the difference between Q3 and Q1. Upper and lower bounds for outliers are defined as Q1 $-$ 1.5 * IQR and Q3 $+$ 1.5 * IQR. Points that lie outside this range are considered outliers.

6.12 Percentile Method

In this method, we define a custom range that accommodates all data points that lie anywhere between 0.5 and 99.5 percentile of the dataset. Points that lie outside this range are considered outliers.

6.13 Methods to Treat Outliners

Removal: Outliers can be removed from the dataset altogether. This is a simple and effective approach, but it can lead to a loss of data.

Transformation: Outliers can be moderated by capping, flooring, or applying logarithmic transformations to make them less extreme.

Imputation: Outliers can be replaced with more representative values. This can be done by using the mean, median, or mode of the data, or by using a more sophisticated imputation technique.

Weighting: Outliers can be given a lower weight in the training process. This reduces their influence on the model.

Robust methods: Robust machine learning algorithms are less sensitive to outliers. These algorithms can be used to train models that are more resistant to outliers.

6.14 Shapes of Distributions

The shape of a distribution in machine learning can have a significant impact on the performance of a model. For instance, if a model is trained on a dataset with a normal distribution, it may perform poorly on new data that is not normally distributed.

Normal Distribution (Gaussian Distribution): In a normal distribution, data is symmetrically distributed around a central mean, forming a bell-shaped curve. This distribution is characterized by a single peak at the mean, and data points are equally distributed on both sides. In a normal distribution, the values of mean, median, and mode are nearly the same. Most of the distributions in the real world do not follow a normal distribution (Fig. 6.9).

Skewed Distribution: As discussed earlier, skewed distributions are asymmetrical, with data points clustering more on one side of the mean than the other. There are two main types: positively skewed and negatively skewed (Fig. 6.10).

Uniform Distribution: In a uniform distribution, all values within a range have roughly the same frequency or probability. It forms a rectangular shape, indicating that each outcome is equally likely (Fig. 6.11).

Bimodal Distribution: Bimodal distributions have two distinct peaks, suggesting that the data can be divided into two separate groups or modes. Each mode represents a different set of characteristics within the data (Fig. 6.12).

Multimodal Distribution: Like bimodal, multimodal distributions have multiple peaks, indicating that the data can be divided into more than two distinct groups or modes (Fig. 6.13).

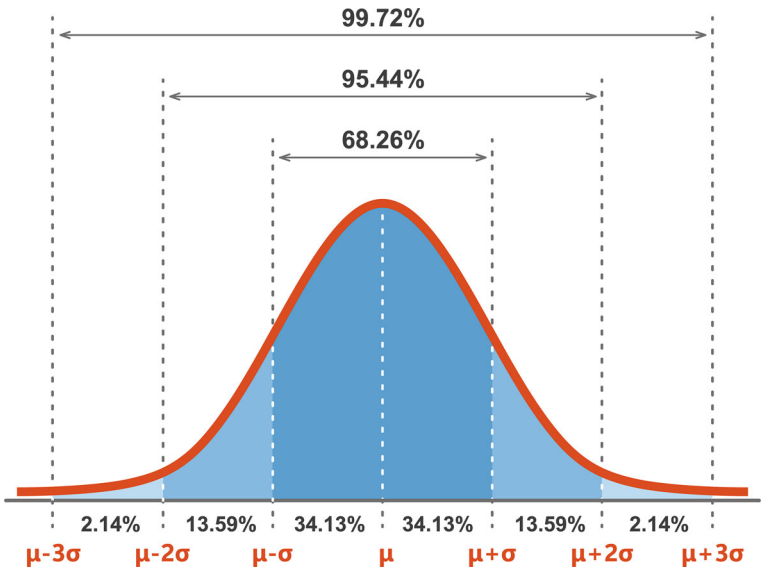


Fig. 6.9 Normal distribution

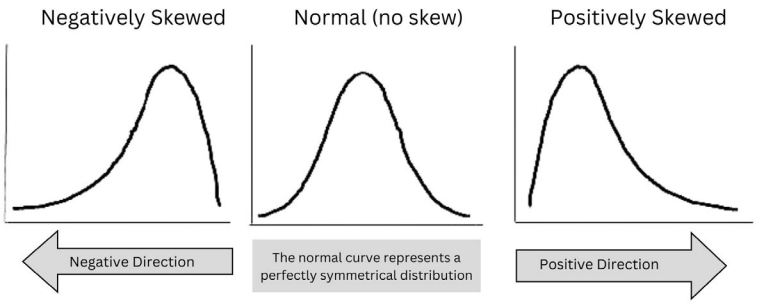


Fig. 6.10 Positive and negative skew

Exponential Distribution: An exponential distribution is characterized by a rapid drop-off in data values after the peak, creating a right-skewed distribution. It often represents the time between events in a Poisson process (Fig. 6.14).

Pareto Distribution: The Pareto distribution, also known as the 80–20 rule, is characterized by a small number of high-frequency values called the 'vital few' and many low-frequency values called the 'trivial many.' (Fig. 6.15)

The parameter alpha (α), also known as the "shape parameter" or "tail index," determines the thickness of the tail of the distribution. It is a measure of how "heavy" or "light" the tail of the distribution is, which in turn affects the concentration of values.

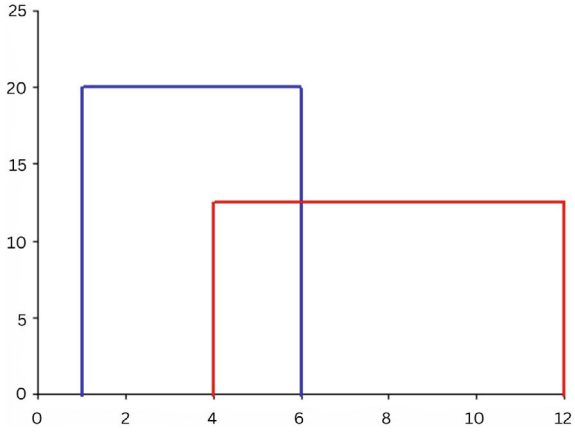


Fig. 6.11 Uniform distribution

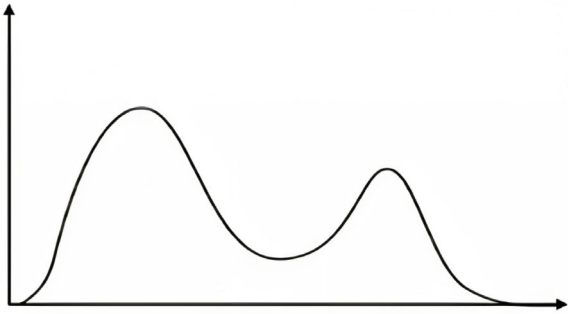


Fig. 6.12 Bimodal distribution

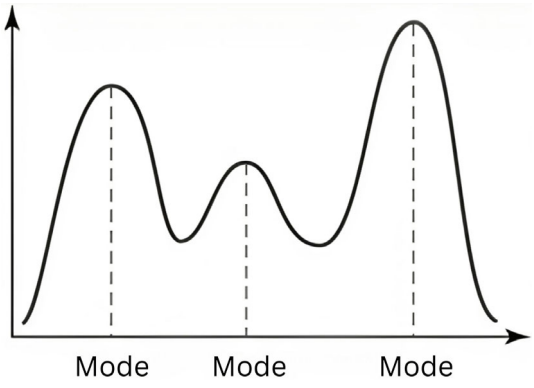


Fig. 6.13 Multimodal distribution

Gaining insights into the distribution's shape plays a vital role in machine learning by guiding the selection of a suitable model. It also aids in outlier detection and evaluates the data's integrity, thereby helping us to make better assumptions and decisions.

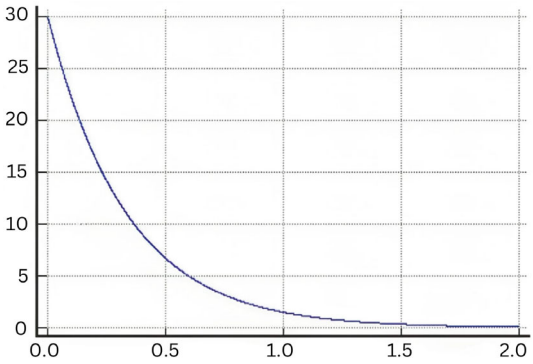


Fig. 6.14 Exponential distribution

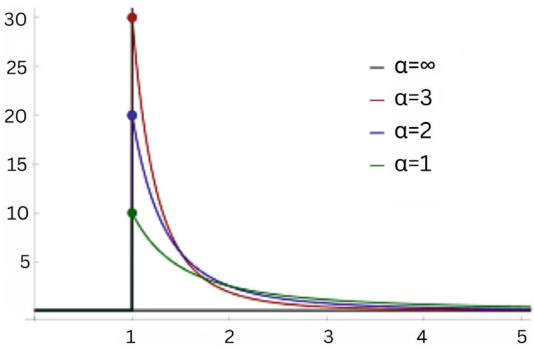


Fig. 6.15 Pareto distribution

References

Essai philosophique sur les probabilités. Cambridge University Press (2009)
Fisher RA (1925) Statistical methods for research workers, 3rd edn. Oliver and Boyd
Galton F (1889) Natural inheritance, 2nd edn. Macmillan
Pearson K (1896) On the theory of correlation. Philos Trans R Soc Lond Ser A, Containing Papers of a Mathematical or Physical Character 187:243–307

Chapter 7
Inferential Statistics

Inferential Stats: Beyond the Data's Surface to the Depths of What Could Be

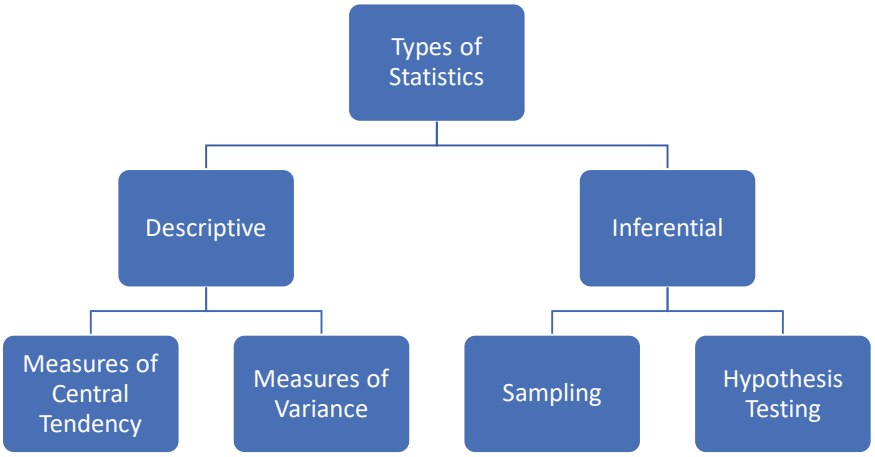
Abstract Building upon the descriptive statistics discussed in the previous chapter, this chapter introduces readers to inferential statistics, a critical component for generalizing findings from sample data to larger populations. Readers will explore the foundational concepts of sampling, recognizing that the accuracy of machine learning models greatly depends on selecting appropriate sampling strategies. Hypothesis testing, an essential method in validating model assumptions and data-driven decisions, is presented clearly. Various types of hypothesis tests, including the t-test, F-test, ANOVA, and Chi-square tests, are covered in detail. Readers will also gain clarity on distinguishing confidence levels from confidence intervals, essential for interpreting model reliability. The chapter thoroughly explains ANOVA and its variations, enabling readers to select the right statistical approach for comparing multiple groups. By mastering these inferential techniques, learners can ensure robust conclusions in machine learning tasks. Ultimately, this chapter underscores that effective sampling and hypothesis testing significantly enhance predictive accuracy and overall reliability of machine learning models.

Keywords Inferential statistics · Sampling · Hypothesis testing · Null hypothesis · Alternate hypothesis · Z test · t test · ANOVA · F test

In the previous chapter, we learnt about descriptive statistics. Inferential statistics completes our understanding of the data that begun with descriptive statistics.

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_7



Machine learning and statistics usually deal with voluminous amount of data. It is often tough for machine learning practitioners and engineers to analyze the whole dataset. Instead, they select a part of the data, study it, and make observations or inferences on the overall dataset.

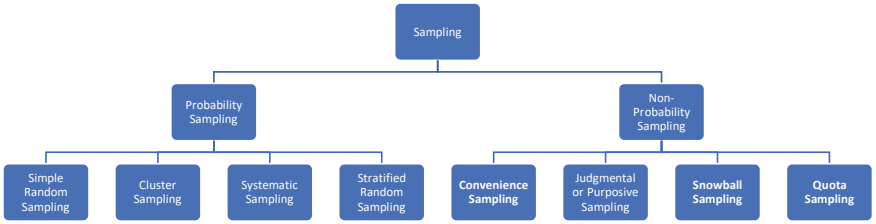
7.1 What Is a Sampling?

Sampling is a process in statistical analysis where machine learning professionals take a predetermined number of observations from a larger population or a large dataset. Sampling allows statisticians and researchers to conduct studies about a large group by using a small portion of the population or the dataset. The method of sampling depends on the type of analysis being performed, but it may include simple random sampling or systematic sampling. Sampling is commonly used in statistics, data analytics, machine learning, psychology, and the financial industry.

7.2 Types of Sampling

Probability Sampling: It is a method in which a statistician or a researcher utilizes specific criteria and randomly selects individuals from a population. In this approach, every member of the population has an equal chance of being included in the sample based on the established criteria.

Non-probability Sampling: This method, on the other hand, involves the practitioner selecting individuals for the study without a predetermined, systematic process. This method lacks a fixed or predefined selection procedure, leading to uneven opportunities for all population elements to be part of the sample.



We will now explore each of these types with relevant examples.

7.2.1 Types of Probability Sampling (Taherdoost 2016; Etikan and Bala 2017)

Simple random sampling stands out as a highly efficient probability sampling technique that conserves both time and resources. It is a dependable approach for gathering information because it involves the random selection of every member of a population, purely by chance. Everyone within the population has an identical likelihood of being selected for inclusion in the sample.

Imagine a university with 1000 undergraduate students. If the university administration wants to gather feedback on campus facilities, they could opt for a simple random sampling method by assigning each student a unique number and using a random number generator to select participants. In this scenario, every one of the 1000 students has an equal chance of being chosen for the survey.

Cluster sampling is a technique in which the population is divided into distinct clusters or groups that represent the larger population. These clusters are selected for inclusion in the sample based on certain demographic criteria pertaining to the experiment. This method simplifies the process of drawing meaningful conclusions from collected data.

Consider a multinational corporation aiming to assess employee satisfaction across its various offices worldwide. Instead of surveying every employee, they can use cluster sampling by categorizing their offices into clusters based on regions, such as North America, Europe, Asia, and so on. By randomly selecting and surveying a subset of offices within each region, they can efficiently gather feedback and gain valuable insights into employee satisfaction on a global scale. This approach streamlines data collection and analysis while providing a representative view of the entire organization.

Systematic sampling is a method employed by statisticians and researchers to select sample members from a population at fixed intervals. It involves determining a starting point for the sample and then consistently choosing individuals at regular intervals within a predetermined range. This approach is characterized by its efficiency and simplicity, making it a relatively time-saving sampling technique.

A manufacturing plant produces 5000 electronic components per day. The officials decide to perform systematic sampling to check the quality of the components. They

begin by examining every 100th component that comes off the production line. This systematic approach allows them to assess the quality of a representative sample of components (Total components/Sample Size = 5000/50 = 100) efficiently and ensure that their products meet the required standards.

Stratified random sampling is a method of sampling that divides the population into smaller groups, called strata, and then randomly selects a sample from each stratum. This method ensures that the sample is representative of the entire population, even if the population is made up of different groups with different characteristics.

If a school wants to survey students about their satisfaction with the cafeteria food, the school could divide the students into strata based on their grade level and then randomly select a sample of students from each grade level. This would ensure that the survey results are representative of the satisfaction levels of students of all grade levels.

7.2.2 Uses of Probability Sampling

Reducing Sample Bias: Probability sampling minimizes sample bias, ensuring that the selected sample accurately reflects the population.

Representing Diverse Populations: It is essential for obtaining a well-rounded dataset when dealing with diverse populations, preventing data skew toward specific demographics.

Creating Accurate Samples: Probability sampling helps analysts and researchers to construct precise and well-defined samples, leading to more reliable data collection.

Statistical Inference: Probability sampling forms the basis for valid statistical inferences, enabling analysts to generalize findings to a larger population.

Longitudinal Studies: It is valuable for maintaining sample representativeness over time in longitudinal research, facilitating accurate tracking of changes or trends within a population.

7.2.3 Types of Non-probability Sampling (Taherdoost 2016; Etikan and Bala 2017)

Convenience sampling relies on the accessibility of subjects, often chosen for its ease and practicality. Analysts select individuals based on proximity rather than representativeness. This method is used when time and resources are limited, making it convenient for data collection. Startups and NGOs distributing leaflets at a mall entrance to random passers-by to promote upcoming events or causes would be an example of convenience sampling.

In judgmental or purposive sampling, analysts use their judgment and purposefully select sample members based on specific criteria relevant to the study's objectives. This method is particularly useful when the research aims to gather insights from individuals who meet specific qualifications. A typical example would be selecting individuals who have a specific medical condition for an in-depth interview about their experiences with the condition.

Snowball sampling is employed when subjects are hard to locate, especially in sensitive or hidden populations. Researchers start with a few known subjects and use referrals to identify and interview others. It is commonly used in situations where the target population is challenging to reach or reluctant to participate. While conducting research on HIV/AIDS, researchers would first contact a few individuals associated with the cause or the affected community who, in turn, may refer researchers to more potential participants.

Quota sampling involves selecting sample members based on predetermined characteristics or attributes. The goal is to create a sample that mirrors the population in terms of specific criteria. This method is relatively quick for collecting samples and is employed when resources for random sampling are limited. Surveying a city's population to ensure that the sample includes the same proportion of age groups (e.g., 20% young adults, 30% middle-aged, and 50% seniors) as the overall population would be an example of quota sampling.

7.2.4 Uses of Non-probability Sampling

Non-probability sampling serves as a valuable tool for generating hypotheses in situations with limited prior information about a population, as it enables data collection from diverse respondents, unveiling potential patterns and trends. Widely embraced in exploratory research endeavors, non-probability sampling expedites data collection from a small, manageable number of respondents, facilitating a deeper understanding of new topics or phenomena. Non-probability sampling is commonly employed in pilot studies due to its ability to swiftly gather data from diverse respondents, making it an ideal choice for assessing the feasibility of larger research projects. When facing budget or time limitations, non-probability sampling proves to be a practical solution, enabling data collection without the resource-intensive demands of probability sampling methods.

7.3 Sampling in Machine Learning

Sampling is a powerful tool that can be used to improve the performance and efficiency of machine learning models. By understanding how sampling works and how it can be used in machine learning, analysts and practitioners can build better models. Sampling is used in machine learning for a variety of purposes.

Model training: Sampling can be used to train machine learning models on large datasets. A sample approach is more efficient because it is often impractical or impossible to train a model on the entire dataset. Instead, a sample of the dataset can be used to train the model.

Model evaluation: Sampling can be used to evaluate the performance of machine learning models on held-out data. This is important because it allows the model to be evaluated on data that it has not seen before.

Hyperparameter tuning: Sampling can be used to tune the hyperparameters of machine learning models. Hyperparameters are parameters that control the behavior of the model, but they are not learned from the data. By sampling different values for the hyperparameters, it is possible to find the values that result in the best-performing model.

7.3.1 Application of Sampling in Machine Learning

Customer Churn Prediction: A machine learning professional uses a sample from a large customer dataset to train a machine learning model for predicting customer churn, as the complete dataset is too large for efficient training.

Fraud Detection Model Evaluation: A software engineer assesses a fraud detection model's performance by dividing a historical transaction dataset into training and test sets, training the model on the former, and evaluating it on the latter to gauge its effectiveness.

Hyperparameter Tuning for Image Classification: A data scientist takes a sample of an image dataset to fine-tune the hyperparameters of a machine learning model designed for image classification, thus ensuring optimal model performance.

7.4 Introduction to Hypothesis Testing

Hypothesis testing is a statistical method used to determine whether there is enough evidence to support a claim about a population. The process of hypothesis testing involves making two statements about a population: the null hypothesis and the alternative hypothesis.

Null Hypothesis: The null hypothesis is a statement about the population that is assumed to be true unless there is enough evidence to reject it. The null hypothesis is often stated in the form of "there is no difference" or "there is no effect."

Alternative Hypothesis: The alternative hypothesis is a statement about the population that is the opposite of the null hypothesis. The alternative hypothesis is what the analyst is trying to prove.

The goal of hypothesis testing is to determine whether the data collected from a sample is sufficient to reject the null hypothesis. If the data is sufficient to reject the null hypothesis, then the analyst can conclude that the alternative hypothesis is true.

7.4.1 Evolution of Hypothesis Testing

The evolution of hypothesis testing can be traced back to the early 1800s, but it was not until the twentieth century that it became a formal and rigorous scientific method. In 1900, Karl Pearson published his work on the chi-square goodness-of-fit test, which is still widely used today. Pearson also developed other important statistical methods, such as the correlation coefficient and the regression coefficient.

Another important figure in the progress of hypothesis testing was Ronald Fisher. In 1925, Fisher published his book Statistical Methods for Research Workers, which popularized the use of hypothesis testing in science. Fisher also developed the concept of the *p*-value, which is a measure of the statistical significance of a result.

In the 1930s, Jerzy Neyman and Egon Pearson further developed hypothesis testing by introducing the Neyman–Pearson framework. This framework emphasized the importance of distinguishing between Type I and Type II errors (false positive and false negative respectively) and provided a rigorous approach for hypothesis testing by considering both the null and alternative hypotheses simultaneously.

Neyman and Pearson proposed a two-step process for hypothesis testing which is in vogue even today:

1. Formulate a null hypothesis and an alternative hypothesis. The null hypothesis is the hypothesis that is being tested.
2. The alternative hypothesis is the hypothesis that is accepted if the null hypothesis is rejected.

As part of hypothesis testing, we collect data and calculate a test statistic. The test statistic is a measure of how different the data is, from what would be expected, under the null hypothesis. The critical value is a threshold value that determines whether the null hypothesis is rejected. If the test statistic is greater than the critical value, the null hypothesis is rejected.

Fisher's work also popularized the concept of significance levels, denoted as alpha (α). Researchers began to use significance levels to set a threshold for the probability of Type I errors (false positives) when conducting hypothesis tests.

In the mid-twentieth century, Bayesian statistics offered an alternative approach to hypothesis testing. Bayesian methods consider prior beliefs or information about the parameters being tested and update these beliefs using observed data to compute posterior probabilities.

In recent years, the replication crisis, and debates about the misuse of *p*-values have prompted a re-evaluation of hypothesis testing practices. Researchers and statisticians have advocated for more transparent reporting, pre-registration of studies, and the use of effect sizes and confidence intervals in addition to *p*-values.

Contemporary hypothesis testing involves a more nuanced understanding of the limitations and potential pitfalls associated with p-values. Analysts and researchers often combine hypothesis testing with other statistical techniques, such as Bayesian analysis, to provide a more comprehensive picture of the evidence supporting or refuting a hypothesis.

7.4.2 Steps in Hypothesis Testing

Hypothesis testing involves a series of steps designed to evaluate a hypothesis about a population parameter based on sample data.

1. Define the Null and Alternative Hypotheses:
   (a) Null Hypothesis (H_0): A statement of no effect or no difference, indicating the status quo or a baseline that the test seeks to challenge.
   (b) Alternative Hypothesis (H_a or H_1): A statement that contradicts the null hypothesis, indicating the presence of an effect or a difference.
2. Choose the Significance Level (α):
   (a) This is the probability of rejecting the null hypothesis when it is actually true, typically set at 0.05, 0.01, or another value depending on the context and discipline.
3. Select the Appropriate Test Statistic:
   (a) The choice of test statistic depends on the nature of the data and the hypothesis being tested, such as a *t*-test for comparing means or a chi-square test for categorical data.
4. Determine the Sampling Distribution:
   (a) Understand the distribution that the test statistic follows under the assumption that the null hypothesis is true, which is crucial for calculating the p-value or setting critical values.
5. Collect Data and Calculate the Test Statistic:
   (a) Gather the sample data and perform the necessary calculations to find the value of the test statistic.
6. Compute the *p*-value or Compare to Critical Value:
   (a) *p*-value Approach: Calculate the *p*-value, which is the probability of observing a test statistic as extreme as, or more extreme than, the observed value under the null hypothesis.
   (b) Critical Value Approach: Determine the critical value(s) from the sampling distribution at the chosen significance level and compare the test statistic to this value.

7. Make a Decision:
   (a) If using the p-value: Reject the null hypothesis if the p-value is less than or equal to the significance level ($p \leq \alpha$).
   (b) If using critical values: Reject the null hypothesis if the test statistic falls into the critical region defined by the critical value(s).
8. Draw a Conclusion:
   (a) Interpret the results in the context of the original research question, stating whether there is enough evidence to reject the null hypothesis in favor of the alternative hypothesis.

7.4.3 Our Goal in Hypothesis Testing

In hypothesis testing, our goal isn't to prove a hypothesis definitively but rather to evaluate evidence against it. We aim to either support or refute an alternative hypothesis. Supporting a hypothesis numerous times does not confirm its absolute truth; a single instance of rejection among many can invalidate it. Therefore, our objective is often to reject the null hypothesis, which indirectly lends support to the alternative hypothesis (Fig. 7.1).

This curve shows what we might expect to see if our assumption (the null hypothesis) is true. The middle, where the curve is highest, is where sample averages usually fall if our assumption is right. There is a high chance that the average of our samples will be in this middle part, which wouldn't make us doubt our assumption.

The ends of the curve, called the tails, are where it is unusual to find our sample averages if our assumption is correct. Finding an average in these tail areas is rare and can make us suspect our assumption might not be true. We decide to accept or

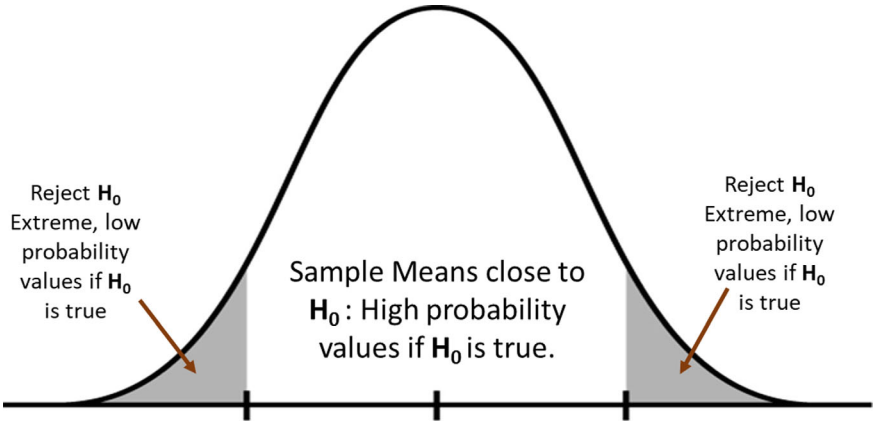


Fig. 7.1 Hypothesis testing

reject our assumption based on whether our sample average falls within these rare areas.

We set a specific cutoff point for this before we start, often at a confidence level of 5% or 0.05. This means we would expect our sample average to fall within this middle area 95% of the time if our assumption is correct, and outside of it in the tails just 5% of the time.

7.4.4 Confidence Level Versus Confidence Interval

A discussion on hypothesis testing often involves understanding the confidence interval. However, a confidence interval is sometimes confused with confidence level, though they serve interconnected purposes in the realm of statistics. The confidence level is a measure of reliability expressed as a percentage, such as 95%, and indicates the probability that the confidence interval will capture the true population parameter if we were to repeat the study multiple times. It is like saying we're 95% sure the net we cast into the sea of data will catch the true value we're after.

For example, if we're testing a new medication and we calculate that the average improvement in patient health scores lies between 10 and 20 points, with a 95% confidence level, this range is the confidence interval. It suggests that we are 95% confident the true average improvement, if we could measure it for the entire population, would fall within this 10-to-20-point range. If we conducted the study 100 times, we would expect the confidence interval to contain the true population mean in 95 out of 100 cases. This does not mean that there is a 95% chance that any given interval contains the population mean; rather, the confidence level pertains to the long-run proportion of such intervals that would contain the parameter if we repeated the study over and over again under the same conditions.

7.4.5 Important Terms in Hypothesis Testing

Important terms in hypothesis testing have been summarized for easy reference.

Fundamental Concepts

- **Population**: The entire group of individuals or data under study.
- **Sample**: A finite subset of the population, consisting of individuals or data points.
- **Sample Size**: The total number of individuals or data points in a sample. Minimum sample required is 30.
- **Parameters and Statistics:**
  - **Parameters**: Statistical constants describing a population.
  - **Statistics**: Statistical constants derived from a sample.

Sampling Errors and Standard Error

- **Standard Error (SE)**: The standard deviation of the sampling distribution of a statistic.
- **Errors in Sampling:**
  - **Type I Error**: Incorrectly rejecting the null hypothesis when it is true.
  - **Type II Error**: Incorrectly accepting the null hypothesis when it is false.

Hypothesis Testing

- **Test of Significance**: A method to determine if the deviation between sample statistics and population parameters, or between two-sample statistics, is significant.
- **Null Hypothesis (H_0)**: A statement suggesting no difference or effect, usually the hypothesis of no difference.
- **Alternative Hypothesis (H_a or H_1)**: A hypothesis that contradicts the null hypothesis, suggesting some difference or effect. Alternate hypothesis is what we are trying to prove.
- **Critical Region**: The region in the sample space that leads to the rejection of the null hypothesis.
- **Acceptance Region**: The region that leads to the acceptance of the null hypothesis.
- **Level of Significance (α)**: The probability that a statistic falls into the critical region, indicating the risk of a Type I error.

Types of Samples

- **Small Sample:** A sample with 30 or fewer individuals or data points.
- **Large Sample:** A sample with more than 30 individuals or data points.

7.4.6 One-Tailed and Two-Tailed Tests in Hypothesis Testing

One-tailed tests and two-tailed tests are two types of hypothesis tests that are used to determine whether there is enough evidence to support a claim about a population (Fig. 7.2).

One-tailed tests are used to test directional hypotheses, which are hypotheses that state that a population parameter is either greater than or less than a certain value. For example, a one-tailed test could be used to test the hypothesis that the average height of men is greater than 6 feet.

Two-tailed tests are used to test non-directional hypotheses, which are hypotheses that state that a population parameter is different from a certain value, but do not specify whether it is greater or less than that value. For example, a two-tailed test could be used to test the hypothesis that the average height of men is different from 6 feet (Table 7.1).

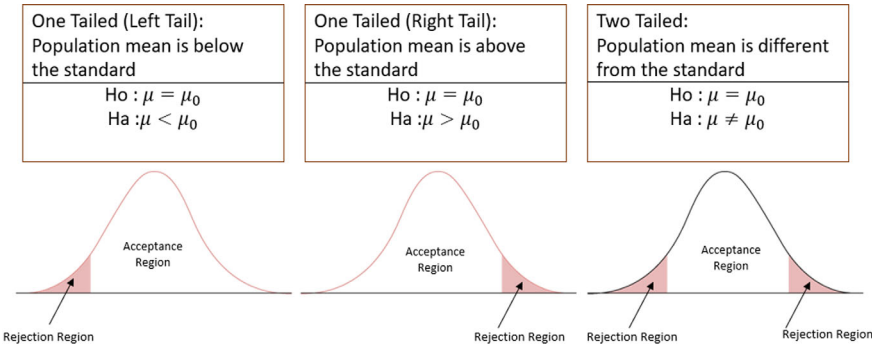


Fig. 7.2 One-tailed and two-tailed tests

Table 7.1 One-tailed and two-tailed tests

Characteristic	One-tailed test	Two-tailed test
Direction of the hypothesis	Directional	Non-directional
Type of test statistic	One-tailed test statistic	Two-tailed test statistic
p-value	Greater than or equal to 0.05 to reject the null hypothesis	Less than or equal to 0.05 to reject the null hypothesis

7.4.7 Types of Hypothesis Testing

Depending upon the type of data available and the size, different types of hypothesis testing are used to determine whether the null hypothesis can be rejected or not. *Z*-test, *t*-test, *F*-test, chi-squared test, and ANOVA are all statistical tests that are used to test hypotheses about populations.

Z-Test

The *Z*-test compares the average of a sample to a known value, like the average of the whole population. You use a *Z*-test when you know the population's average and standard deviation, and you have a large sample. If the *Z*-test gives you a result that's far from the expected value, it suggests that the sample provides enough evidence against the null hypothesis.

Let us understand the working of *Z*-test with an example.

A factory claims its light bulbs last 1200 h on average. You suspect that they last less than the claimed performance. You test 100 bulbs, and they last an average of 1150 h with a standard deviation of 100 h.

- Null Hypothesis (H_0): The bulbs last 1200 h on average.
- Alternative Hypothesis (H_1): The bulbs last less than 1200 h on average.

The *Z*-score is calculated as

$$\frac{x-\mu}{\left(\frac{\sigma}{\sqrt{n}}\right)} = -0.5$$

In the above formula, x is the sample mean (1150 h), μ is the population mean (1200 h), σ is the standard deviation (100 h), and n is the sample size (100). If the Z-score falls below a certain critical value, we reject H_0. The critical value in a Z-test is determined by referring to a Z-table, which provides the cumulative probability of a standard normal distribution. For a 95% confidence level, critical value is -1.645, which is obtained from z-table.

In this book, we will adopt a programmatic approach to calculate the probability value and make a decision regarding the hypothesis, instead of manually calculating the test statistic and then referring to a critical table to determine the p-value.

- Import the libraries

```
import numpy as np
from scipy import stats
```

- Sample data

```
sample_mean = 1150
sample_size = 100
sample_std = 100
```

- Population mean

```
population_mean = 1200
```

- Calculate the Z-score

```
z_score = (sample_mean - population_mean) / (sample_std /
np.sqrt(sample_size))
```

- Get the p-value

```
p_value = stats.norm.sf(abs(z_score))
```

- Print the Z-score and p-value

```
print("Z-score:", z_score)
print("P-value:", p_value)
```

- Output of the program

```
Z-score: -5.0
P-value: 2.866515718791933e-07
```

Inference

Since the *p*-value is very small (less than 0.05), we reject the null hypothesis. The evidence suggests that the light bulbs do not last 1200 h on average and support the alternative hypothesis that they last less than 1200 h.

t-Test

The *t*-test is similar to the *Z*-test but is used when you don't know the population's standard deviation and you have a smaller sample. It compares the average of your sample to the population average or to another sample. There are different types of *t*-tests, like one-sample, two-sample, and paired, each appropriate for different scenarios. The *t*-test tells you if the differences you see are likely not just by chance.

The different types of *t*-test are summarized in Table 7.2.

Let us understand the working of *t*-test with an example.

You want to know if a new teaching method is more effective. A class of 25 students using the new method scores an average of 78 on a test, while the typical score is 75. The standard deviation is 10.

- Null Hypothesis (H_0): The new method does not change the scores.
- Alternative Hypothesis (H_1): The new method improves the scores.

The *t*-score is calculated similarly to the *Z*-score, but with the sample standard deviation since the population standard deviation is unknown.

$$t = \frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}} = 1.5$$

where

Table 7.2 Types of *t*-test

Type of *t*-test	Purpose	Design	Example use case
One-sample *t*-test	To compare the mean of a single sample to a known value (often a theoretical mean)	Involves one group and compares the sample mean to a known or theoretical population mean	Comparing the average height of a class to the national average
Two-sample independent *t*-test	To compare the means of two independent groups	Involves two groups that are independent of each other. Each group provides a separate data sample	Comparing the average test scores of students from two different schools
Paired *t*-test (dependent *t*-test)	To compare means from the same group at different times or under different conditions	Involves one group measured twice, typically before and after a treatment or at two different times	Measuring students' test performance before and after a specific training program

- $\bar{x}$ is the sample mean
- μ is the population mean
- s is the sample standard deviation
- n is the sample size.

The calculated t-statistic is compared against a critical value from the t-distribution, which is determined by the desired level of significance (often α for a 95% confidence level) and the degrees of freedom. The degrees of freedom for a one-sample t-test are calculated as:

$\text{df} = n - 1$, where n is the sample size.

For the given example with 25 students, the degrees of freedom would be:

$\text{df} = 25 - 1 = 24$.

The critical t value is arrived at from the t distribution table for 24 degrees of freedom and 95% confidence level. If the calculated t-statistic is greater than the critical value (in absolute terms), you would reject the null hypothesis, suggesting that the average test score using the new method is significantly different from the typical score.

Computing t-score and p-value programmatically:

- Import the libraries

```
from scipy import stats
```

- Sample data

```
sample_mean = 78
sample_size = 25
sample_std = 10
```

- Population mean

```
population_mean = 75
```

- Calculate the t-score

```
t_score = (sample_mean - population_mean) / (sample_std /
np.sqrt(sample_size))
```

- Get the p-value

```
p_value = stats.t.sf(np.abs(t_score), df=sample_size-1) * 2
```

- Print the Z-score and p-value

```
print("T-score:", t_score)
print("P-value:", p_value)
```

- Output of the program

```
T-score: 1.5
P-value: 0.1466556460682007
```

Inference

Since the p-value is greater than 0.05, we do not reject the null hypothesis. There is not enough evidence to conclude that the new teaching method changes the scores significantly.

F Test

F-test is generally used to compare two populations' variances. The F-test can tell you if two populations are significantly different in terms of their variances. It's often used in scenarios like comparing the variability of two manufacturing processes or testing the assumption of homoscedasticity (equal variances) in regression analysis.

F-test involves calculating the ratio of the variances of two groups.

$$F = \text{Variance of sample } 1/\text{Variance of sample } 2$$

The test statistic follows an F-distribution under the null hypothesis that the variances are equal. The F-value is then used to decide whether the observed differences in variances are statistically significant.

Let us say you are conducting an experiment to compare the effectiveness of two different fertilizers on plant growth. You apply Fertilizer 1 to one group of plants and Fertilizer 2 to another group. After a certain period, you measure the growth of the plants in each group. You notice some differences in the growth patterns and want to know if these differences are statistically significant. To ascertain this, you decide to use an F-test to compare the variances in plant growth between the two groups.

- Null Hypothesis (H_0): The variances in plant growth are the same for both Fertilizer 1 and Fertilizer 2. In other words, any observed difference in plant growth variance between the two fertilizers is due to random chance.
- Alternative Hypothesis (H_1): The variances in plant growth are different for the two fertilizers. This means that one fertilizer leads to more variable (or consistent) plant growth than the other.

We have the following sample variances for plant growth with two different fertilizers.

Fertilizer 1: Variance = 12.
Fertilizer 2: Variance = 20.

Let us compute t-score and p-value programmatically to evaluate the hypotheses.

- Import the libraries

```
from scipy import stats
```

- Sample data

```
variance1 = 12
variance2 = 20
```

- Calculate the F statistic

```
f_score = variance2 / variance1
```

- Degrees of freedom for each sample (assuming equal sample sizes, example 30 each)

```
dfn = 30 - 1  # degrees of freedom for the numerator
dfd = 30 - 1  # degrees of freedom for the denominator
```

- Get the p-value

```
p_value = stats.f.sf(f_score, dfn, dfd)
```

- Print the Z-score and p-value

```
print("F-score:", f_score)
print("P-value:", p_value)
```

- Output of the program

```
F-score: 1.6666666666666667
P-value: 0.0874869519341921
```

Inference

Since the p-value is greater than 0.05, we do not reject the null hypothesis.

7.5 ANOVA (Analysis of Variance)

ANOVA, which stands for Analysis of Variance, is used when you are comparing the means of more than two groups. While it does use the F-distribution for its calculations, its primary goal is to determine whether there are any statistically significant differences among the means of the groups (Fig. 7.3).

ANOVA also calculates an F statistic, but it does so by dividing the variance between group means by the variance within the groups. This F-value is used to test the null hypothesis that all group means are equal.

$$F = \text{Variance between groups}/\text{Variance within group}.$$

The test statistic is an F-value calculated as the ratio of the mean square among groups to the mean square within groups. A high F-value indicates that there is more

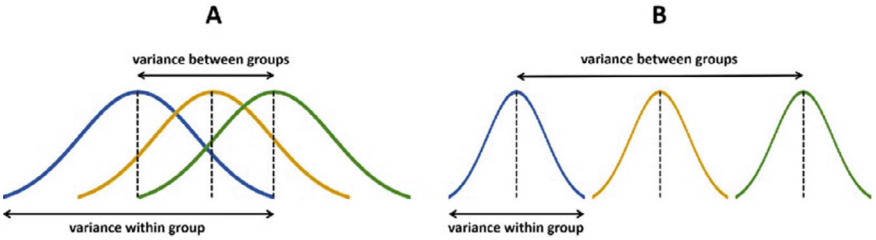


Fig. 7.3 ANOVA

variation among the groups than within them, suggesting significant differences in group means.

ANOVA is the tool to use if you are testing the effectiveness of different types of fertilizers or different teaching methods across several groups. It checks if there is a significant difference among the group means. Basically, ANOVA tells you whether the differences in results across the groups are probably not just by chance.

The different types of ANOVA are summarized in Table 7.3.

Table 7.3 Types of ANOVA

Type of ANOVA	Purpose	Design	Example
One-way ANOVA	To test the effect of one independent variable on a dependent variable	One categorical independent variable with at least three levels and one continuous dependent variable	Testing the effect of different diets on blood pressure
Two-way ANOVA	To examine the interaction between two independent variables on a dependent variable	Two categorical independent variables, each with multiple levels, and one continuous dependent variable	Impact of diet and exercise regimen on cholesterol levels
Repeated measures ANOVA	Used when the same subjects are used for each treatment (within-subjects design)	Participants are measured multiple times under different conditions or over time	Testing cognitive skills of students before, during, and after training
Multivariate ANOVA (MANOVA)	To test the effect of one or more independent variables on two or more dependent variables	Similar to ANOVA but with multiple dependent variables	Effect of teaching methods on students' math and science performance
Mixed-design ANOVA	Used when there are both between-subjects and within-subjects factors	Combination of repeated measures and independent groups designs	Effect of a new curriculum over time on student performance

Table 7.4 ANOVA example

Serial number	Diet A	Diet B	Diet C
1	6.93	3.87	4.76
2	5.44	7.8	4.32
3	6.07	8.06	3.98
4	7.45	6.09	8.31
5	7.05	9.78	5.4
6	3.93	5.22	5.48
7	6.04	7.06	4.52
8	4.83	6.77	6.92
9	4.89	8.88	4.09
10	5.45	8.8	5.75
11	5.16	7.19	4.94
12	6.59	7.46	6.46
13	5.83	5.91	5.4
14	5.13	4.57	4.6
15	5.49	6.57	5.97
16	5.37	7.19	6.51
17	6.64	8.51	6.08
18	4.78	8.47	6.36
19	5.34	6.53	5.25
20	4.06	6.63	5.57

7.5.1 How an Analysis of Variance Is Used to Check Difference in Means

Let us use ANOVA to test the effectiveness of three different diets on weight loss. Table 7.4 summarizes the weight loss across the three diet plans.

Hypotheses

- Null Hypothesis (H0): All diets result in the same average weight loss.
- Alternative Hypothesis (H1): At least one diet leads to a different average weight loss.

Approach

First, calculate the overall mean weight loss across all groups. Then, calculate the "Between-Group Variance" and "Within-Group Variance".

The F statistic is calculated as F = Within-Group Variance/Between-Group Variance. Let us look at the manual way of computation now.

1. Calculate the Overall Mean (Grand Mean):

(a) This is the average of all the observations across all groups.
(b) It serves as the reference point to measure the variability between the groups.

2. Calculate the "Between-Group" Sum of Squares (SSB):

(a) This measures the total variability between the different groups.
(b) For each group, we take the difference between the group mean and the grand mean, square it, and then multiply by the number of observations in that group.
(c) The SSB is the sum of these values for all groups.

$$\text{SSB} = n_A(\overline{x}_A - \mu)^2 + n_B(\overline{x}_B - \mu)^2 + n_C(\overline{x}_C - \mu)^2 = 29.62.$$

(d) In the above formula, n_A, n_B, n_C are the number of observations in each group and $\overline{x}_A, \overline{x}_B, \overline{x}_C$ are the means of each group.

3. Calculate the "Within-Group" Sum of Squares (SSW):

(a) This measures the variability within each group.
(b) For each observation, we take the difference between the observation and its group mean, square it, and sum these for all observations within the group.
(c) The SSW is the sum of these values for all groups.

$$\Sigma\left(x_{A_i} - \overline{x}_A\right)^2 + \left(x_{B_i} - \overline{x}_B\right)^2 + \left(x_{C_i} - \overline{x}_C\right)^2 = 81.32.$$

(d) In the above formula, $x_{A_i} x_{B_i} x_{C_i}$ represent the individual observations in each group.

4. Calculate the Between-Group Mean Square (MSB):

(a) This is the average of the between-group sums of squares.
(b) It is obtained by dividing the SSB by the number of groups minus one, which represents the degrees of freedom between groups.

$$\text{MSB} = \frac{\text{SSB}}{k-1} = 14.83$$

In the above, k is the number of groups.

5. Calculate the Within-Group Mean Square (MSW):

(a) This is the average of the within-group sums of squares.
(b) It is obtained by dividing the SSW by the total number of observations minus the number of groups, representing the degrees of freedom within groups.

$$\text{MSW} = \frac{\text{SSW}}{N-k} = 1.43.$$

(c) In the above, N is the total number of observations across all the groups.

6. Calculate the F-value:
   (a) This is the ratio of the MSB to the MSW.

$$\text{MSW} = \frac{\text{MSB}}{\text{MSW}} = 10.39.$$

   (b) It represents how much the between-group variability exceeds the within-group variability.
   (c) A larger F-value indicates that the between-group variability is more likely to be significant, while a smaller F-value suggests that the observed between-group variability could be due to random chance within the groups.

7. Compare F-Value Against a Critical Value
   (a) The calculated F-value is compared against a critical value from the F-distribution table. The critical value depends on the level of significance (α) you've chosen for your test (often 0.05 for a 95% confidence level) and the degrees of freedom for both the numerator (between groups) and the denominator (within groups).
   (b) Numerator Degrees of Freedom (df1): This is one less than the number of groups, so for three diets, $\text{df1} = 3 - 1 = 2$.
   (c) Denominator Degrees of Freedom (df2): This is the total number of observations minus the number of groups, so for 60 total observations across three groups, $\text{df2} = 60 - 3 = 57$.
   (d) Look up the critical value in an F-distribution table where df1 corresponds to the row (representing the between-groups variability) and df2 corresponds to the column (representing the within-groups variability).
   (e) If the calculated F-value is greater than the critical value from the table, you reject the null hypothesis.

Computing F-statistic and p-value programmatically:

- Import the libraries

```
from scipy import stats
```

- Sample data

```
dietA = [6.93, 5.44, 6.07, 7.45, 7.05, 3.93, 6.04, 4.83, 4.89, 5.45,
         5.16, 6.59, 5.83, 5.13, 5.49, 5.37, 6.64, 4.78, 5.34, 4.06]

dietB = [3.87, 7.80, 8.06, 6.09, 9.78, 5.22, 7.06, 6.77, 8.88, 8.80,
         7.19, 7.46, 5.91, 4.57, 6.57, 7.19, 8.51, 8.47, 6.53, 6.63]

dietC = [4.76, 4.32, 3.98, 8.31, 5.40, 5.48, 4.52, 6.92, 4.09, 5.75,
         4.94, 6.46, 5.40, 4.60, 5.97, 6.51, 6.08, 6.36, 5.25, 5.57]
```

- Performa ANOVA

```
f_score, p_value = stats.f_oneway(dietA, dietB, dietC)
```

- Display the *F*-score and *p*-value

```
f_score, p_value
```

- Output of the program

```
(10.395363700451647, 0.00014157617049981005)
```

Inference

Since the *p*-value is less than 0.05, we reject the null hypothesis and conclude that the average weight loss of at least one diet group is significantly different from that of the others. The *F*-score of about 10.40 indicates there is a significant difference in the average weight loss among the three diet groups. The low *p*-value suggests that the observed differences in average weight loss are unlikely to have occurred by chance.

Chi-Square Test

The Chi-square test is a statistical method used to determine if there is a significant difference between expected and observed frequencies in one or more categories. It's commonly used in hypothesis testing, particularly in goodness of fit (to see if a sample matches the population) and test of independence (to check if two variables are independent).

$$\chi^2 = \sum \frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}}$$

Goodness-of-Fit Test

A biologist wants to determine if a population of fruit flies has the expected distribution of eye colors. The expected distribution, based on Mendelian genetics, might be that 75% have red eyes and 25% have white eyes. After observing 100 fruit flies, the biologist finds that 70 have red eyes and 30 have white eyes. Using a chi-square goodness-of-fit test, the biologist can determine whether this observed distribution significantly deviates from the expected 75/25 split.

Test of Independence

A market researcher wants to understand if there is a relationship between gender and preference for a new beverage product. They survey 200 people—100 men and 100 women—and ask if they like the beverage. The results show that 60 men and 40 women liked the beverage, while 40 men and 60 women did not. A chi-square test of

Table 7.5 Chi-square test

Contingency table		
	Red eyes	White eyes
Observed	70	30
Expected	75	25

independence can help the researcher determine whether the preference for the new beverage is independent of gender, or if there's a statistically significant association between the two variables (Table 7.5).

For Red Eyes

- Observed (O) = 70
- Expected (E) = 75

$$\chi^2 = \sum \frac{(70 - 75)^2}{75} = 0.33$$

For White Eyes

- Observed (O) = 30
- Expected (E) = 25

$$\chi^2 = \sum \frac{(30 - 25)^2}{25} = 1$$

Adding these together for the chi-square statistic:

$$\chi^2 = 0.33 + 1 = 1.33$$

Subsequent Steps

The chi-square value is compared against a critical value from the chi-square distribution table to determine whether the observed frequencies significantly differ from the expected frequencies. This comparison depends on the significance level (commonly denoted as α) and the degrees of freedom (df) associated with the test.

Degrees of Freedom (df): For a chi-square test, the degrees of freedom are typically calculated as df $= N - 1$, where N is the number of categories or groups. In the fruit flies example with two groups (Red Eyes and White Eyes), df $= 2 - 1 = 1$.

Significance Level (α): This is a threshold chosen by the researcher to determine the level of statistical significance. Common choices are 0.05 (5%) for a 95% confidence level or 0.01 (1%) for a 99% confidence level.

To decide whether the observed differences are statistically significant:

1. Choose a significance level (α), such as 0.05.

2. Find the critical value in a chi-square distribution table corresponding to df and α.
3. Compare calculated chi-square value to the critical value:
   (a) If the chi-square value is greater than the critical value, you reject the null hypothesis, indicating a statistically significant difference between observed and expected frequencies.
   (b) If the chi-square value is less or equal to the critical value, you do not reject the null hypothesis, indicating that any observed differences could be due to chance.

For the fruit flies example with df $= 1$ and a common $\alpha = 0.05$, you would look up the critical value for df $= 1$ at the 0.05 significance level in the chi-square table, which is approximately 3.841. Since our calculated chi-square value (1.33) is less than the critical value, we would not reject the null hypothesis, suggesting that the observed deviation from the expected frequencies might not be statistically significant at the 0.05 level. This calculation can be done programmatically as well.

7.6 Applications of Hypothesis Testing in Machine Learning

Hypothesis testing is a crucial component of machine learning, primarily in the context of evaluating models and making decisions based on data. It helps practitioners to make informed decisions, refine models, and validate assumptions using statistical methods. It also plays a vital role in ensuring the robustness, fairness, and effectiveness of machine learning applications.

Model Evaluation: Hypothesis testing is used to assess the performance of machine learning models. Researchers or practitioners formulate hypotheses about the model's effectiveness, such as whether it outperforms a baseline or another model. They then conduct hypothesis tests to determine if there is statistical evidence to support these hypotheses. Common evaluation metrics like accuracy, precision, recall, $F1$-score, and ROC-AUC can be used in hypothesis testing to compare models.

Feature Selection: In feature selection, researchers and machine learning professionals often use hypothesis tests to determine which features are most relevant for a particular machine learning task. By testing the hypothesis that a feature significantly contributes to predictive accuracy, unnecessary or redundant features can be eliminated, thereby simplifying the model, and potentially improving its performance.

Hyperparameter Tuning: Hypothesis testing can be employed to fine-tune hyperparameters of machine learning models. Researchers and analysts may hypothesize that adjusting certain hyperparameters will lead to better model performance. They

can conduct experiments with different hyperparameter settings and use statistical tests to determine if the changes result in statistically significant improvements.

Anomaly Detection: In anomaly detection tasks, machine learning models are used to identify unusual patterns or events in data. Hypothesis testing is employed to assess whether a particular observation is significantly different from the norm or constitutes an anomaly.

Fairness and Bias Assessment: Hypothesis testing can be used to evaluate whether a machine learning model exhibits bias with respect to specific demographic groups. Analysts may test the hypothesis that the model's predictions are distributed equally across groups or assess whether differences are statistically significant.

Statistical Significance: Machine learning researchers often use hypothesis testing to determine the statistical significance of experimental results. For example, in reinforcement learning, analysts might test the hypothesis that a new algorithm outperforms existing ones in terms of convergence time or overall reward.

References

Etikan I, Bala K (2017) Sampling and sampling methods. Biom Biostat Int J 5(6):215–217. https://doi.org/10.15406/bbij.2017.05.00149

Taherdoost H (2016) Sampling methods in research methodology; how to choose a sampling technique for research. Int J Acad Res Manage (IJARM) 5

Chapter 8
Libraries in Machine Learning

Abstract Building upon inferential statistics discussed earlier, this chapter sets the stage for understanding machine learning algorithms and their implementation. Readers are introduced to essential libraries in Python, focusing specifically on scikit-learn—one of the most popular and user-friendly libraries for ML tasks. Briefly exploring scikit-learn's core functionalities, readers will recognize how it simplifies complex modeling processes, from data preprocessing to algorithm deployment. Additionally, the chapter touches upon AutoML (Automated Machine Learning), highlighting its growing importance in streamlining model selection and optimization tasks. By familiarizing themselves with these powerful tools, learners will be well-equipped to efficiently handle real-world machine learning projects. Ultimately, this foundational knowledge ensures that readers can confidently transition into exploring algorithms, their concepts, and the associated programming aspects in the subsequent chapters.

Keywords Scikit learning · Auto ML

8.1 Libraries

In the world of programming and machine learning, libraries are incredibly important. A library in programming is a collection of pre-written code that developers can use to perform common tasks. Think of it as a toolbox where each tool serves a specific function, saving you the time and effort of creating tools from scratch.

The growth of libraries has significantly boosted programming efficiency. Instead of writing every single line of code, programmers can use libraries to handle routine tasks. This not only speeds up the development process but also reduces the chance of errors. Libraries often include documentation, which makes it easier for new developers to understand and use them effectively.

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_8

In the field of machine learning, libraries have been a game-changer. They have made machine learning more accessible, allowing more people to build and implement machine learning models without being experts in the underlying algorithms. This democratization has led to a surge in innovative applications of machine learning across various industries.

There are many examples of machine learning libraries, each designed for different aspects of the process. Some popular ones include:

- TensorFlow: Developed by Google, it's great for deep learning and neural network-based applications.
- PyTorch: Known for its flexibility and ease of use, it's another favorite for deep learning tasks.
- Pandas: Excellent for data manipulation and analysis, it's a must-have for data preprocessing in machine learning.
- NumPy: Provides support for large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions.
- Scikit-learn: Renowned for its simplicity and accessibility, it's a versatile library offering various algorithms for both supervised and unsupervised learning.

8.2 Scikit-Learn

Let us take a closer look at Scikit-learn. It is a Python library and one of the most popular tools for machine learning. It is known for its ease of use and wide range of machine learning algorithms, including classification, regression, clustering, and dimensionality reduction.

What makes Scikit-learn stand out is its consistent interface. Regardless of the type of algorithm you are using, the steps for using it are usually the same: import the algorithm, instantiate it, fit it to your data, and then use it to make predictions or transform data. This consistency makes it easier to experiment with different algorithms.

Scikit-learn is also well-documented and supported by a large community. Whether you're a beginner or an experienced programmer, you'll find a wealth of resources and community advice to help you along the way.

Moreover, Scikit-learn integrates seamlessly with other Python libraries, like Pandas and NumPy. This integration makes it a flexible tool that fits nicely into the broader Python ecosystem for data science and machine learning.

Libraries have thus become an integral part of programming, especially in the field of machine learning. They provide reusable code that saves time and effort, making it easier for developers to focus on solving the unique aspects of their problems.

Libraries like Scikit-learn, in particular, have lowered the barrier to entry, allowing more people to participate in the development and application of machine learning models. As the field continues to grow, these libraries will play a crucial role in shaping the future of how we interact with and utilize machine learning.

8.3 Toward Automated Machine Learning (Auto ML)

In the author's considered opinion, Scikit-learn is a starting point for auto ML because it is very straightforward to use. You can actually build a model that can predict outcomes with less than 20 lines of code. This is simple considering that making predictions is usually a complex task.

With Scikit-learn, you do not need to write a lot of code to make something work. It is like having a set of shortcuts that let you build a working prediction model quickly. This is a big help for people who are just starting with machine learning, as it makes the first steps easier.

Because you can do so much with so little code, Scikit-learn is a favorite starting tool for automated machine learning systems. These systems begin with the basics that Scikit-learn offers and then builds more complex things on top. So, Scikit-learn is not just for beginners. It is also the foundation that more advanced auto-ML systems are built upon, helping them to start strong and get to the finish line faster.

Chapter 9
Simple Linear Regression

Simple Linear Regression: Finding a Straight Path Through a World of Variables

Abstract Building on the foundational knowledge from previous chapters, this chapter introduces readers to Simple Linear Regression (SLR), one of the fundamental algorithms in machine learning. It begins by highlighting the evolution of regression techniques, emphasizing their importance in predictive modeling. Readers will understand how to evaluate the accuracy of regression algorithms using metrics such as R-squared and RMSE. The chapter demonstrates the practical implementation of SLR using Excel, providing a hands-on approach that makes the algorithm accessible. Importantly, readers will delve into the mathematics and linear algebra underpinning regression, reinforcing the critical yet often overlooked role linear algebra plays due to widespread reliance on libraries. The concepts of training versus testing datasets are clearly explained to highlight best practices in model validation. Readers also learn about Polynomial Regression, exploring its ability to model nonlinear relationships effectively. Lastly, the chapter addresses common pitfalls such as underfitting and overfitting, guiding learners on achieving the optimal model balance.

Keywords Regression · Simple linear regression · Accuracy · Mean squared error · Root mean squared error · Mean absolute error · *R*-squared · Correlation coefficient · Coefficient of determination · Adjusted *R*-squared · Trendline · Line of best fit · Slope · Intercept · Linear algebra in machine learning · Test versus train · Test-Train split · Random_state = 42 · Cross-validation · Leave-one-out cross-validation · *K*-fold cross-validation · Polynomial regression · Degrees of freedom · Underfit · Overfit · Optimal fit · Bias-variance tradeoff

Let us start with a summary of the algorithms before getting into simple linear regression.

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_9

Machine learning models can be broadly categorized into four groups, namely—supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. We have already discussed the definitions of these groups and their use cases.

9.1 Supervised Learning

Supervised learning is a type of machine learning where the model is trained on labeled data. In this approach, the input data includes explicit labels that define the desired output, providing the model with the correct answers for a set of given inputs. For instance, in image classification tasks, each image is tagged with a category label such as "apple" or "orange", which guides the 'learning' process.

The primary objective of supervised learning is to develop a function or mapping that accurately connects input data to its corresponding labels. By learning from the patterns, relationships, and decision boundaries identified in the training data, the algorithm becomes capable of generalizing this knowledge to predict or classify new, unseen data effectively. This process not only enhances the model's predictive accuracy but also enables it to navigate through complex datasets and to make informed decisions based on 'learned' insights.

We are going to look at the following supervised learning models (Nasteski 2017) in this book.

Linear regression is a predictive modeling technique used to estimate a continuous value, such as housing prices or product sales. It identifies the best-fit line through a dataset, allowing predictions based on variables like size, location, and other attributes. For instance, a linear regression model could be trained to forecast house prices using data on square footage, neighborhood quality, and age of the property.

Logistic regression is utilized to predict binary outcomes—whether an event occurs or not, such as customer churn or email classification as spam. This model estimates the probability of an outcome based on input features and classifies the outcome by the highest likelihood, transforming linear regression outputs through a logistic function.

Decision trees are versatile models used in both classification and regression. They work by progressively splitting the data into smaller subsets based on feature values, selecting splits that most effectively distinguish the classes. Predictions are made by tracing the path from the tree's root to the leaf node corresponding to the criteria of the input data.

A **random forest** is an ensemble model that improves upon single decision trees by aggregating the predictions of multiple trees trained on different parts of the dataset. This approach reduces the risk of overfitting and enhances prediction accuracy, especially in complex, high-dimensional datasets.

Support vector machines (SVMs) are powerful models used for classification and regression. SVMs work by finding the optimal hyperplane that separates data into classes with the maximum margin. This makes SVMs effective for complex classification tasks, such as distinguishing between images of cats and dogs using features derived from image pixels.

The **k-Nearest Neighbors** (kNN) algorithm predicts the class or value of a new data point by analyzing the 'k' closest points in the feature space. The output is determined by the majority vote or average of these neighboring points, making it a straightforward yet effective method for classification and regression.

Naive Bayes classifiers are based on Bayes' theorem, with the naive assumption of conditional independence between every pair of features, given the response variable. This model is particularly effective for classification tasks such as email filtering, where it predicts the likelihood of an email being spam based on the frequency of words typically seen in spam messages.

We will now focus on the first and the most used machine learning model—simple linear regression also referred to as linear regression.

9.2 Introduction to Simple Linear Regression (SLR)

Simple linear regression is a fundamental statistical method used to explore and quantify the relationship between two quantitative variables. This technique is primarily utilized for two main purposes.

1. **Determining the Strength of Association**: It assesses how strongly two variables are related. For example, it can measure the relationship between rainfall and soil erosion or between the square footage of a house and its price.
2. **Predicting Values**: It predicts the value of one variable based on the known value of the other variable. For instance, it can estimate the amount of soil erosion expected from a specific level of rainfall or determine the market price of a house based on its size in square feet.

Simple linear regression accomplishes these objectives by fitting a straight line through the data points, which represents the best approximation of the relationship between the dependent and independent variables. This linear model allows us to understand how the dependent variable changes in response to variations in the independent variable, providing a clear and quantifiable insight into the nature of their relationship.

9.3 Assumptions Made in Simple Linear Regression

Simple linear regression, as a parametric test, operates under several key assumptions that ensure the validity of its results.

One such assumption is **homogeneity of variance**, also known as homoscedasticity, which requires that the scatter of data points around the regression line remains relatively consistent across the range of the independent variable.

Another critical assumption is the **independence of observations**, which stipulates that each data point is uncorrelated with others and each occurrence is equally probable. These assumptions are fundamental to maintaining the integrity of the regression analysis.

Additionally, the assumptions of **normality** and **linearity** are pivotal for the effective application of simple linear regression. Normality assumes that the residuals of the data points are normally distributed around the mean, forming a bell-shaped and symmetrical curve. Linearity, on the other hand, assumes a linear relationship between the independent and dependent variables, implying that the best-fit line through the data points is straight.

If the data used does not meet these conditions, the result produced by a simple linear regression model might not be reliable.

9.4 Mathematics Behind Simple Linear Regression

The formula for simple linear regression is:

$$y = m * X + c + e,$$

where

- y is the predicted value of the dependent variable for any given value of the independent variable.
- m is the intercept, the predicted value of y when the x is 0.
- c is the slope, which tells us how much we expect y to change with respect to X.
- X is the independent variable, the variable that is influencing y.
- e is the error of the estimate, or how much variation there is, in our estimate of the regression coefficient.

Line of Best Fit: Linear regression finds the line of best fit through the data by searching for the regression coefficient (m) that minimizes the total error (e) of the model (Fig. 9.1).

The **intercept**, denoted as c, represents the expected value of the dependent variable when the independent variable is zero. Essentially, it is where the line of the regression model intersects the y-axis. This value is critical as it provides a baseline from which the effects of the independent variable are measured.

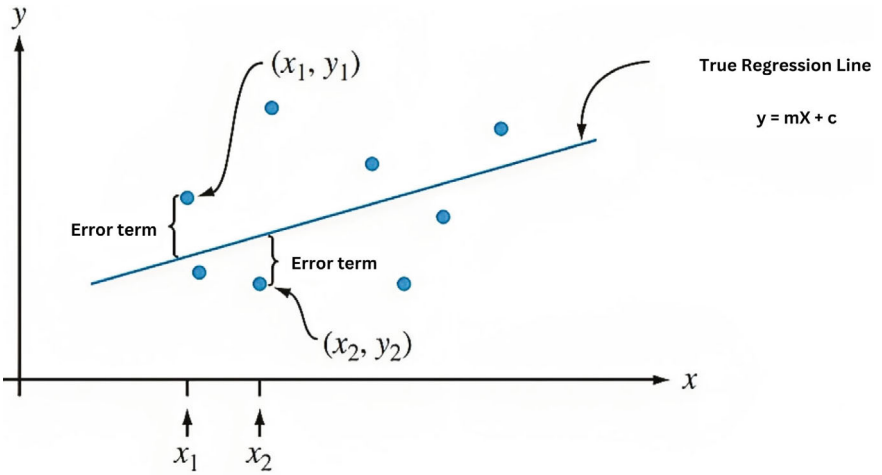


Fig. 9.1 Simple linear regression

The **regression coefficient**, represented as *m*, quantifies the expected change in the dependent variable for each unit increase in the independent variable. It illustrates the strength and direction of the relationship between the variables. A positive coefficient indicates a direct relationship, while a negative coefficient suggests an inverse relationship.

The **error of the estimate**, noted as *e*, reflects the degree of variation or uncertainty in our estimate of the regression coefficient. This error is inevitable due to the inherent complexities and multiple influencing factors in real-world scenarios. Understanding this error is crucial as it helps to gauge the reliability of the predictions made by the regression model.

9.5 Evolution of Simple Linear Regression

Simple linear regression (SLR) has been a fundamental statistical tool for centuries, with its development tracing back to notable historical figures. The earliest applications of SLR can be attributed to the eighteenth-century French mathematician Pierre-Simon Laplace, who utilized the technique to analyze the orbital motion of the planets. This early adoption set the stage for further refinement by the German mathematician Carl Friedrich Gauss in the early nineteenth century. Gauss introduced the least squares method, which remains the most prevalent method for fitting a linear regression model to data today.

By the late nineteenth century, the British statistician Francis Galton further advanced SLR by developing statistical inference methods. These methods enable analysts to draw conclusions about a population from sample data, enhancing the

applicability of SLR in various scientific fields. The twentieth century saw a significant transformation in SLR with the advent of computers, which facilitated the development of more sophisticated and accurate computational techniques for both fitting and analyzing SLR models.

In recent times, several innovative developments (Freedman 2009) have emerged in the field of SLR:

- **Robust Regression** methods offer resilience against outliers, improving the reliability of regression analysis, when traditional least squares methods fail.
- **Bayesian Regression** approach incorporates prior knowledge about the parameters within the regression model, blending historical data with new observations for more informed analysis.
- **Nonparametric Regression** do not make any assumptions about the distribution of the data, unlike traditional SLR methods. Since these methods do not assume a specific form of the relationship between variables, they bring more fluidly to the structure of the data.

9.6 Ways to Evaluate the Model (Measuring Accuracy)

There are several ways to evaluate a model and measure errors including mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), R-squared error (R^2) and adjusted R^2 score.

1. **Mean Squared Error (MSE)** is calculated by taking the sum of the squared differences between the predicted values and the actual values, then dividing this sum by the number of observations. This approach ensures that larger errors are penalized more heavily, as squaring the errors magnifies the impact of larger discrepancies more significantly than smaller ones.

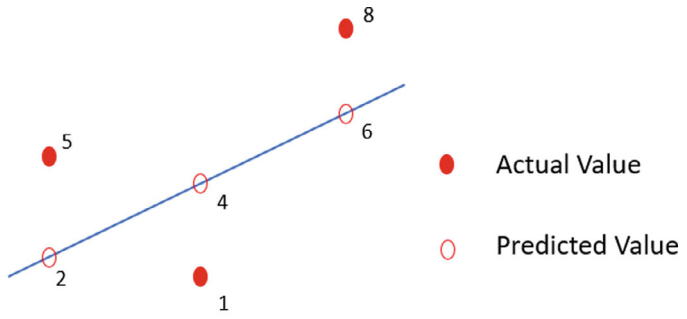
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where

y_i is the actual value,
$\hat{y}_i$ is the predicted value,
n is the number of test examples.

Sample Calculation (Fig. 9.2):

Squaring each error serves a dual purpose. It eliminates any negative signs, ensuring that errors in either direction are treated as equally detrimental, and it emphasizes larger errors more than smaller ones, which can be particularly impactful in assessing



$$\text{MSE} = \frac{1}{3} * (5-2)^2 + (1-4)^2 + (8-6)^2 = 7.3$$

Fig. 9.2 Mean squared error

model performance. A lower MSE value indicates a model that more accurately fits the data, reflecting better performance.

One key benefit of MSE is its differentiability, which makes it particularly useful as a loss function in optimization algorithms. This characteristic allows for the application of efficient mathematical methods to find the model parameters that minimize MSE, thus improving model accuracy.

However, MSE's sensitivity to outliers is a notable drawback. Since MSE squares differences, outliers can disproportionately affect the overall error metric, leading to potential overfitting to outlier observations and misrepresentations of the model's performance on typical data points. This makes MSE less robust in datasets where outliers are present or in scenarios where uniform accuracy across all data points is critical.

2. **Root Mean Squared Error (RMSE)** is the square root of the mean squared error.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}}$$

The root mean squared error (RMSE) is an effective performance metric for regression models because it is differentiable, making it a practical choice for use as a loss function in optimization algorithms. This property allows for the application of gradient-based methods to minimize RMSE, thereby refining model predictions. Additionally, RMSE is measured in the same units as the dependent variable (y), which enhances interpretability. For example, if the predicted values are in the hundreds, the RMSE will also be expressed in hundreds, making it straightforward for analysts and stakeholders to understand the magnitude of prediction errors relative to the scale of the data.

However, RMSE shares a critical limitation with the mean squared error (MSE), from which it is derived. It is sensitive to outliers. Because RMSE involves squaring the prediction errors before averaging them, extreme values or outliers can disproportionately influence the overall metric. This sensitivity to outliers can lead to misleadingly high error values and potentially degrade the model's performance if these outliers are not representative of the general data trend. Consequently, this impacts the model's ability to generalize from training data to unseen data effectively.

3. **Mean Absolute Error (MAE)** metric turns negative errors to positive errors. This is done so that negative errors do not decrease positive errors when summing up.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

One significant benefit of the mean absolute error (MAE) is its robustness to outliers compared to other metrics like MSE or RMSE. Since MAE calculates the average of the absolute differences between predicted values and actual values without squaring them, it does not overly penalize larger errors. This characteristic makes MAE particularly useful in scenarios where data may contain outliers or anomalies that are not representative of the general data trends, ensuring that these do not disproportionately influence the overall error metric.

However, a notable limitation of MAE is its lack of differentiability at zero, which can pose challenges in optimization algorithms, particularly those that rely on gradient-based methods. This occurs because the absolute value function has a kink at zero, making the gradient undefined at that point. As a result, while MAE is advantageous for its robustness to outliers, its application as a loss function in regression models can be restricted, requiring modifications or alternative approaches for minimizing error in certain analytical contexts.

4. The ***R*-squared (R^2)** is a statistical measure that quantifies the proportion of variance in the dependent variable that is predictable from the independent variable(s). It serves as an indicator of the adequacy of how well the regression model captures the observed variation in the data. *R*-squared values range from 0 to 1, where higher values suggest a better model fit. Specifically, a value of 1 indicates that the model perfectly predicts the dependent variable, with no variance unaccounted for.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

One of the primary advantages of using *R*-squared over other metrics like the mean squared error (MSE) or the root mean squared error (RMSE) is its scale-independence. This feature makes *R*-squared particularly useful for comparing the performance of models across different datasets or contexts where the scales of the dependent variables differ. Unlike MSE and RMSE, which can exhibit large variations when applied to data of different scales, *R*-squared provides a consistent basis

for model comparison, simplifying the evaluation of model effectiveness irrespective of the underlying data scale.

A key disadvantage is that *R*-squared will never decrease as additional variables are added to a model, regardless of whether those variables are relevant or not. This can lead to a misleadingly high *R*-squared value for models that are overfitted, as it will appear to improve simply by including more variables, even if those variables contribute little to the actual explanatory power of the model. This characteristic of *R*-squared can encourage the inclusion of superfluous variables, potentially complicating the model without genuine benefit. As a result, relying solely on *R*-squared to gauge model quality can be problematic, particularly in situations where parsimony and the relevance of predictors are crucial.

5. **Adjusted R^2** refines the traditional R^2 metric by introducing a penalty for the inclusion of irrelevant independent variables in the regression model. This adjustment is specifically designed to prevent the artificial inflation of the model's explanatory power that can occur with the addition of superfluous predictors.

$$R^2_{\text{adj}} = \left[\frac{\left(1 - R^2\right)(n-1)}{(n-k-1)} \right],$$

where k is the number of independent variables and n is the number of test samples.

By incorporating the number of predictors and the sample size into its calculation, adjusted R^2 provides a more accurate reflection of the true predictive quality of the model, particularly when comparing models with different numbers of variables. The above discussed methods can be used to measure the errors and thereby the accuracy of the machine learning model.

Adjusted *R*-squared offers a significant advantage over the traditional R^2 score by penalizing the indiscriminate addition of predictors that do not enhance the model's predictive ability. This feature makes it particularly valuable for assessing the potential overfitting of a model, ensuring that each included variable contributes meaningfully to the explanation of variability in the dependent variable.

However, a limitation of adjusted R^2 is that it does not fully account for model bias. Even with the adjustment, a model can still exhibit a relatively high R^2 score while having systemic biases in its predictions. Therefore, while adjusted R^2 can help mitigate the risk of overfitting, it should be used in conjunction with other diagnostic measures to fully assess the performance and appropriateness of a regression model.

Table 9.1 summarizes the different accuracy measures.

9.6.1 Implementation of SLR in Excel

- Dependent variable: Price

Table 9.1 Accuracy measures in regression

Accuracy measure	Purpose	Question it answers	Limitations
Mean absolute error (MAE)	To measure the average magnitude of errors in a set of predictions, without considering their direction	"What is the average absolute difference between the predicted and actual values?"	Does not indicate overprediction or underprediction Less sensitive to outliers compared to other measures
Mean squared error (MSE)	To measure the average of the squares of the errors or deviations, emphasizing larger errors more than smaller ones	"What is the average of the squared differences between the predicted and actual values?"	More sensitive to outliers than MAE Can give disproportionately high weight to larger errors
Root mean squared error (RMSE)	To measure the standard deviation of prediction errors, providing a measure of the magnitude of errors	"What is the standard deviation of the differences between the predicted and actual values?"	Overly sensitive to outliers Scale of the errors can be unclear due to squaring
R-squared (R^2)	To measure the proportion of the variance for the dependent variable that's explained by the independent variables	"How much of the variance in the dependent variable is explained by the independent variables?"	Does not indicate the adequacy of a model Can be artificially high for models with many predictors (overfitting)
Adjusted R-squared	To adjust the R-squared value for the number of predictors in a model, addressing the issue of model complexity	"How much of the variance in the dependent variable is explained by the independent variables, adjusted for the number of predictors?"	Can be misleading if the model is not specified correctly More complex to interpret than R-squared

- Independent variable: Area in square feet (Table 9.2)
- Select the scatter plot (Fig. 9.3)
- Once the scatter plot is selected, the following visual appears (Fig. 9.4).
- Right click any of the dots or the data points in the above visual (Fig. 9.5).
- Select the "Add Trendline" option as shown above to get the line of best fit (Fig. 9.6).
- Select format trendline option (Fig. 9.7).
- Select the display options for equation and R-squared value (Fig. 9.8).
- Equation for simple linear regression is displayed in the chart:
  - $y = 301.44x + 52434$
  - This could be written as Price = 301.44 * Area + 52434

Table 9.2 Example for simple linear regression

Area in square feet	Price
1000	300,000
1250	400,000
1500	500,000
1750	600,000
2000	700,000
2250	800,000
2500	850,000
2800	910,000
3000	950,000
3200	920,000

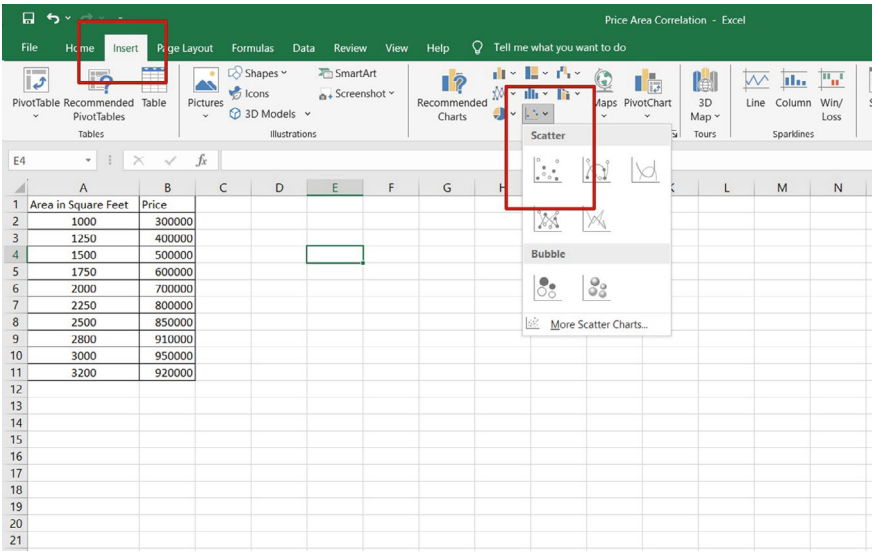


Fig. 9.3 SLR in Excel 1

  - Slope is 52434 and intercept is 301.44
  - Coefficient of determination (R^2) is 0.9537.

- Understanding slope (m) and intercept (c) (Fig. 9.9).
- The equation can be used to predict the price for a particular area
  - If area = 4000
  - Price = 301.44 * 4000 + 52,434 = 1,258,205.4.

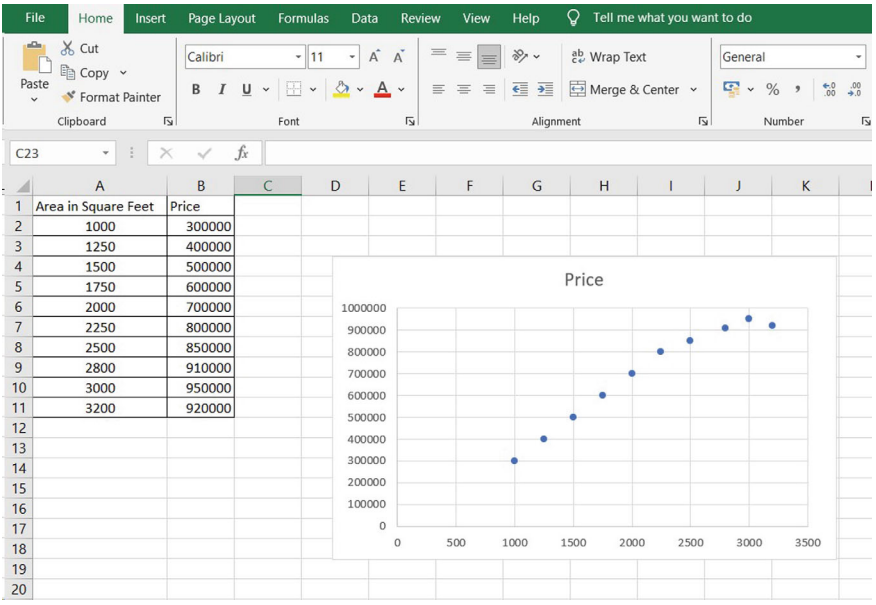


Fig. 9.4 SLR in Excel 2

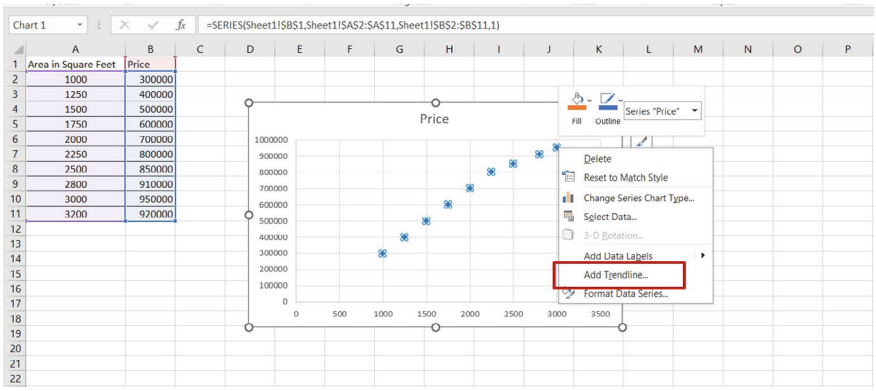


Fig. 9.5 SLR in Excel 3

- Computing the mean absolute error (Table 9.3).

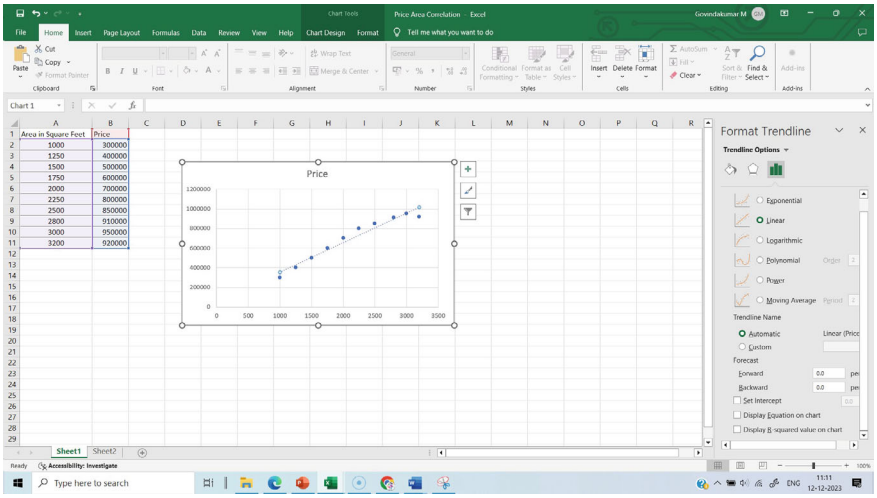


Fig. 9.6 SLR in Excel 4

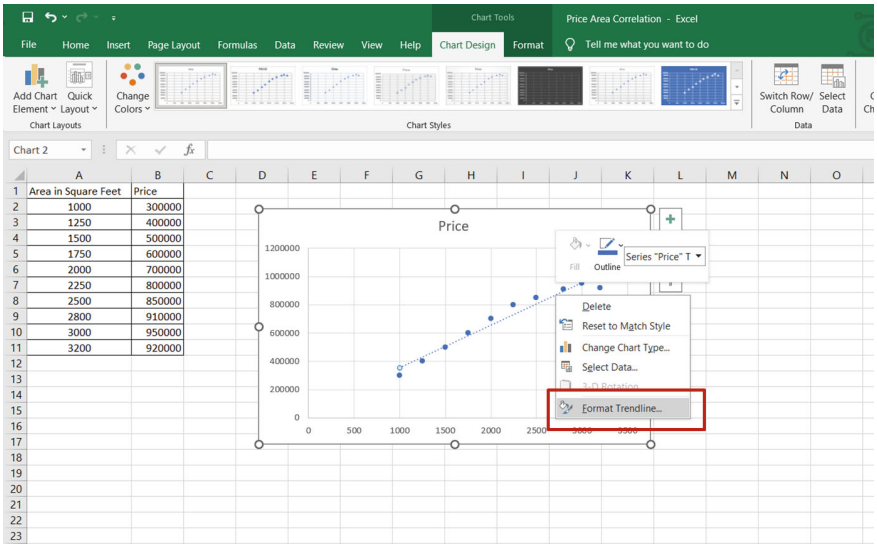


Fig. 9.7 SLR in Excel 5

9.7 Implementation of SLR Using Python for House Price Prediction (with Respect to Area)

- Import necessary modules

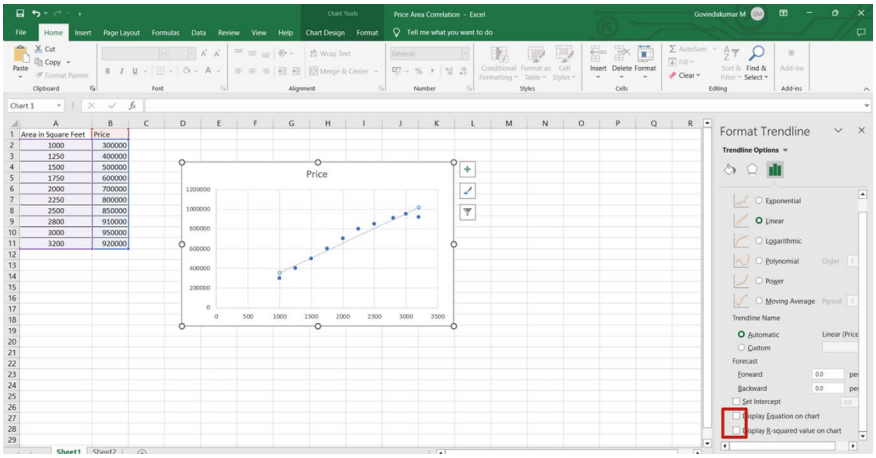


Fig. 9.8 SLR in Excel 6

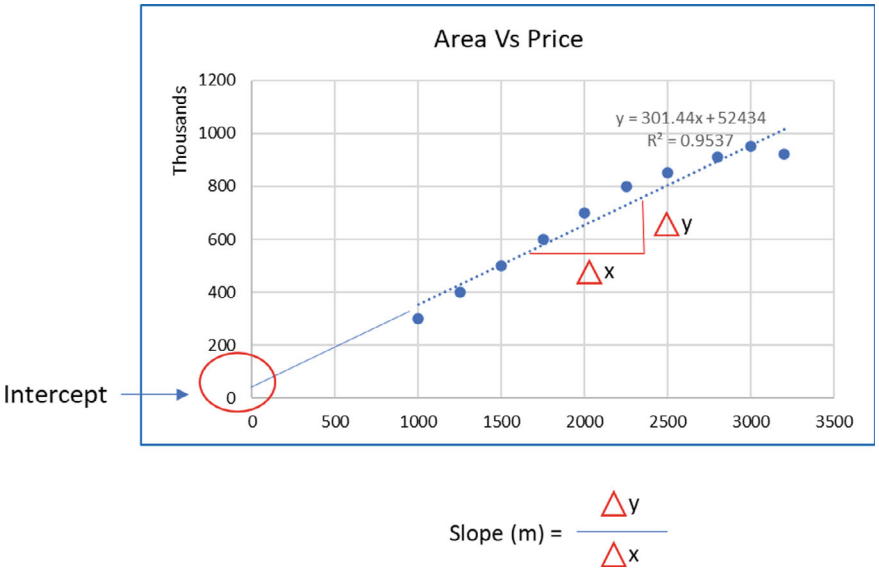


Fig. 9.9 SLR in Excel 7

```
    import numpy as np
    from sklearn.linear_model import LinearRegression
    import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, r2_score
```

- Load the data

Table 9.3 Calculation of mean absolute error

Area (sq. ft)	Actual price	Predicted price	Difference	Absolute value of the difference
1000	300,000	353,876.74	− 53,876.74	53,876.74
1250	400,000	429,237.47	− 29,237.47	29,237.47
1500	500,000	504,598.19	− 4,598.19	4,598.19
1750	600,000	579,958.92	20,041.09	20,041.09
2000	700,000	655,319.64	44,680.36	44,680.36
2250	800,000	730,680.36	69,319.64	69,319.64
2500	850,000	806,041.09	43,958.91	43,958.91
2800	910,000	896,474.05	13,525.95	13,525.95
3000	950,000	956,762.53	− 6,762.53	6,762.53
3200	920,000	1,017,051.11	− 97,051.11	97,051.11
			Average	38,305.199

```
area = np.array([1000, 1250, 1500, 1750, 2000, 2250, 2500, 2800, 3000, 3200]).reshape(-1, 1)
price = np.array([300000, 400000, 500000, 600000, 700000, 800000, 850000, 910000, 950000,
920000])
```

- Create a linear regression model

```
model = LinearRegression()
```

- Fit the model with the data

```
model.fit(area, price)
```

- Compute slope and intercept of the regression line

```
slope = model.coef_[0]
intercept = model.intercept_
```

- Making predictions

```
predicted_price = model.predict(area)
```

- Plotting the results

```
plt.scatter(area, price, color='blue', label='Actual Price')
plt.plot(area, predicted_price, color='red', label='Predicted Price')

plt.title('Linear Regression: Area vs Price')
plt.xlabel('Area in Square Feet')
plt.ylabel('Price')
plt.legend()
plt.show()
```

- Display the slope and intercept

```
print(f"The slope of the regression line is: {slope}")
print(f"The intercept of the regression line is: {intercept}")
```

- Make predictions using the model

```
predictions = model.predict(area)
```

- Calculate the coefficient of determination (*R*-squared)

```
r_squared = r2_score(price, predictions)
```

- Calculate the mean absolute error (MAE)

```
mae = mean_absolute_error(price, predictions)
```

- Calculate *r*, correlation coefficient. *R* is the square root of *R*-squared for simple linear regression

```
r = np.sqrt(r_squared)
```

- Display the results

```
print(f"Correlation coefficient (r): {r}")
print(f"Coefficient of determination (R^2): {r_squared}")
print(f"Mean Absolute Error (MAE): {mae}")
```

See Fig. 9.10.

The slope of the regression line is: 301.4428955734899
The intercept of the regression line is: 52,433.84690633393.

Correlation coefficient (*r*): 0.976561672113539
Coefficient of determination (R^2): 0.9536726994411912
Mean absolute error (MAE): 38,305.20909757886.

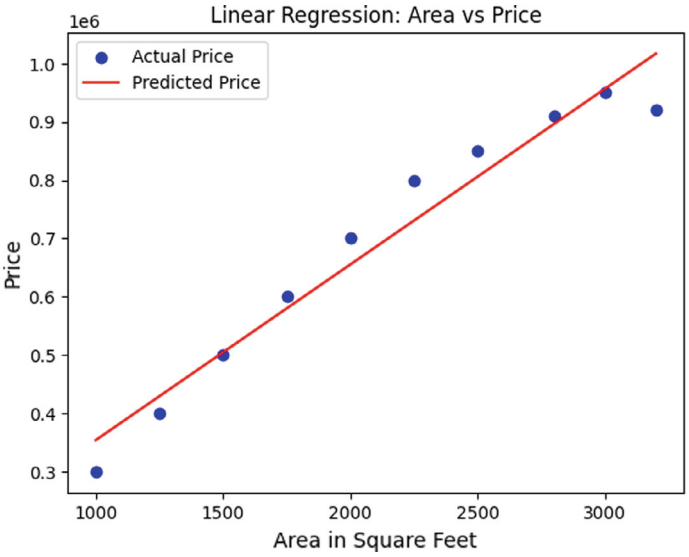


Fig. 9.10 Scatter plot for area versus price

9.8 Importance of *R* and *R*-Squared

While the coefficient of determination tells us how much of the dependent variable's variation our model accounts for, the coefficient of correlation tells us about the direction and strength of a relationship. Both coefficients are central to understanding and communicating the effectiveness of a simple linear regression model.

Table 9.4 summarizes the importance of these two measures using hypothetical scenarios, though the second and third scenarios are less likely in real world.

Table 9.4 Scenarios to explain *R* and *R*-squared

	Hypothetical *R* Value	Hypothetical R^2 Value	Interpretation of *R*	Interpretation of R^2
Scenario 1	0.9	0.81	A strong positive linear relationship (as area increases, price increases significantly)	81% of the variance in price is explained by the area
Scenario 2	− 0.7	0.49	A strong negative linear relationship (as area increases, price decreases significantly)	49% of the variance in price is explained by the area
Scenario 3	0.1	0.01	A very weak positive linear relationship (area has little effect on price)	Only 1% of the variance in price is explained by the area

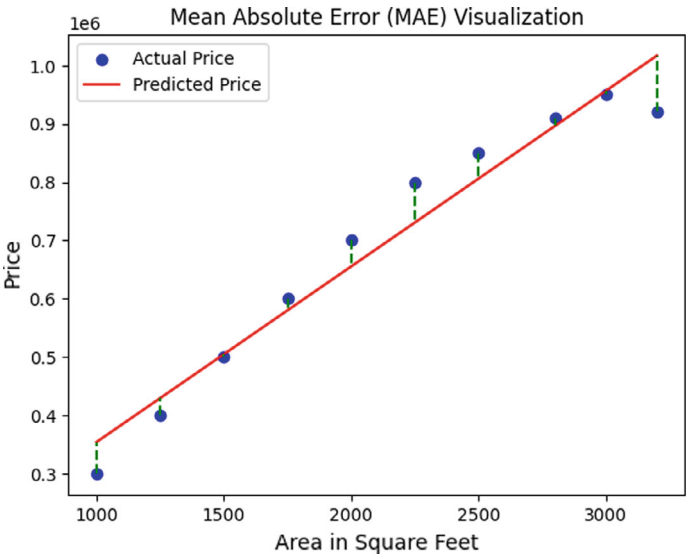


Fig. 9.11 Scatter plot showing MAE

Now, let us visualize the absolute error (Fig. 9.11).

- Calculate the absolute errors

```
errors = price - predictions
absolute_errors = np.abs(errors)
```

- Plot the actual prices and predicted prices

```
plt.scatter(area, price, color='blue', label='Actual Price')
plt.plot(area, predictions, color='red', label='Predicted Price')
```

- Plot the MAE for each data point

```
for i in range(len(area)):
    plt.plot([area[i], area[i]], [price[i], predictions[i]], color='green', linestyle='--')
```

```
plt.title('Mean Absolute Error (MAE) Visualization')
plt.xlabel('Area in Square Feet')
plt.ylabel('Price')
plt.legend()
plt.show()
```

9.8.1 Use Cases of Simple Linear Regression

In the realm of real estate, simple linear regression (SLR) is a powerful tool for predicting house prices based on attributes such as square footage. By establishing a linear relationship between the size of a property and its market value, SLR allows real estate analysts and potential buyers to estimate price trends effectively. Similarly, in the sales sector, SLR is instrumental in forecasting product sales by analyzing the relationship between advertising expenditures and sales volume. This can help businesses allocate their marketing budgets more efficiently, optimizing return on investment.

Additionally, SLR finds extensive applications in marketing, where it is used to gauge the effectiveness of various marketing campaigns. By correlating specific campaign attributes with sales outcomes, companies can identify which strategies yield the best results. In finance, SLR assists in predicting stock market trends and assessing loan risk by linking financial indicators to market behaviors or creditworthiness. In the field of medicine, SLR is employed to predict the likelihood of developing certain diseases or to evaluate the efficacy of new medications, providing valuable insights that can guide treatment and prevention strategies.

9.8.2 Limitations of Simple Linear Regression

SLR can only model linear relationships between variables. If the relationship between two variables is not linear, then SLR will not be able to accurately model it. SLR is sensitive to outliers. Outliers are data points that are very different from the rest of the data. If your data contains outliers, they can skew the results of your SLR analysis. SLR cannot account for all the factors that may influence the dependent variable. This means that there will always be some error in your predictions.

9.8.3 Examples of How These Limitations Might Manifest Themselves in Real Life

Simple linear regression (SLR) is an effective statistical tool for modeling relationships between two variables that exhibit a linear correlation. For instance, the relationship between height and weight generally follows a linear pattern, making SLR suitable for modeling these variables and yielding a good fit. Conversely, SLR may not be as effective when applied to relationships that are inherently non-linear, such as the relationship between income and happiness, where the connection is more complex and potentially influenced by various other factors.

Moreover, it's critical to recognize that correlation does not imply causation. For example, while there is a noticeable correlation between the consumption of junk food

and the incidence of heart attacks, it is overly simplistic and potentially misleading to conclude that junk food consumption directly causes heart attacks. Other underlying factors such as stress, lifestyle, and genetics also play significant roles.

Similarly, in real estate, using SLR to model the relationship between house price and square footage can be problematic if the dataset contains outliers, such as exceptionally large or small houses, which can distort the analysis. While SLR is a powerful and versatile tool across various disciplines for understanding relationships and making predictions, it is essential to apply it with an awareness of its limitations and the context of the data being analyzed.

9.9 Linear Algebra in Machine Learning

Linear algebra is the foundation upon which many machine learning algorithms are built, though it is often overshadowed by the convenience of high-level libraries like Scikit-learn. Linear algebra provides the mathematical framework required to understand and implement various algorithms. Despite the prevalence of high-level libraries like Scikit-learn that simplify machine learning implementations, a foundational knowledge of linear algebra remains critical. This branch of mathematics deals extensively with vectors and matrices, which are crucial for performing operations on datasets typically represented in these formats. Key concepts such as vector spaces, matrix multiplication, determinants, eigenvalues, and eigenvectors are integral to facilitating complex data transformations that underpin many machine learning processes.

Machine learning is deeply rooted in mathematics, with linear algebra playing a pivotal role across a broad spectrum of algorithms. Whether it is simple models like linear regression and logistic regression or more complex constructs like support vector machines and deep learning networks, linear algebra is indispensable. These algorithms leverage linear algebraic operations to manipulate data, optimize performance, and make predictions or classifications. For example, the transformation of input data into a suitable format, the optimization of learning parameters, and the architecture of neural networks in deep learning all rely fundamentally on linear algebra. Understanding these principles enriches the practitioner's ability to develop more effective machine learning models and enhances the overall grasp of the field's theoretical underpinnings.

9.9.1 The Overlooked Importance Due to High-Level Libraries: A Double-Edged Sword

With the advent of high-level libraries like Scikit-learn in Python, the intricacies of linear algebra often go unnoticed. These libraries have abstracted the complex mathematics, making it easier for practitioners to implement machine learning algorithms without understanding the underlying math.

This has significantly lowered the barrier to entry in the field, allowing even those who are less inclined toward mathematics to explore and learn machine learning.

9.9.2 Behind the Scenes of Scikit-Learn Library

In the previous sections, we learnt about simple linear regression using excel and libraries in Python. Now, let us explore how linear algebra is used to compute slope and intercept in simple linear regression.

Let us consider two data points to understand this.

$$(x_1, y_1) = (1, 2)\text{---Here } x_1 = 1 \text{ and } y_1 = 2$$
$$(x_2, y_2) = (2, 3)\text{---Here } x_2 = 2 \text{ and } y_2 = 3$$

We want to find the line that best fits these points, which can be represented as $y = mX + c$, where m is the slope and c is the y-intercept.

Solving for m and c using matrix computations involves the following steps.

Step 1: Set up the equations as a matrix equation $A \,.\, x = B$, where A is the matrix of coefficients, x is the vector of variables, and B is the vector of constants. The equations given are:

$$1m + 1c = 2$$
$$2m + 1c = 3$$

So the matrix equation ($A \,.\, x = B$) becomes:

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

So,

$$\begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}^{-1} * \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

Step 2: Find the determinant of matrix A:

$$\det(A) = (1)(1) - (2)(1) = 1 - 2 = -1.$$

Step 3: Find the adjugate of matrix A:

The adjugate of a 2×2 matrix is found by swapping the elements on the diagonal and changing the signs of the off-diagonal elements.

$$\text{adj(A)} = \begin{bmatrix} 1 & -1 \\ -2 & 1 \end{bmatrix}$$

Step 3: Calculate the inverse of matrix A using the determinant and adjugate.

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A) = \frac{1}{-1}\begin{bmatrix} 1 & -1 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 2 & -1 \end{bmatrix}$$

Step 4: Calculate the values of m and c through matrix multiplication.

$$\begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 2 & -1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

This gives us $m = 1$ and $c = 1$, meaning the best fit line is $y = x + 1$.

Let us do the above calculations programmatically now.

- Import the required library

```
import numpy as np
```

- Define the data points

```
X = np.array([[1, 1], [2, 1]])
Y = np.array([2, 3])
```

- Calculate transpose of X

```
X_transpose = X.T
```

- Calculate product of X transpose and X

```
X_transpose_X = np.dot(X_transpose, X) # np.dot is used for matrix multiplication
```

- Calculate inverse of the product

```
X_transpose_X_inv = np.linalg.inv(X_transpose_X)
```

- Calculate product of X transpose and Y

```
X_transpose_Y = np.dot(X_transpose, Y)
```

- Compute coefficients m and c

```
coefficients = np.dot(X_transpose_X_inv, X_transpose_Y)
m = coefficients[0]
c = coefficients[1]
```

- Print the results

```
print(f"The slope (m) is: {m}")
print(f"The y-intercept (c) is: {c}")
```

- Output of the program
  The slope (a) is: 1.0000000000000036
  The y-intercept (b) is: 0.9999999999999964.

9.10 Testing and Training in Machine Learning

Machine learning uses algorithms to analyze and learn from data. These algorithms or models find patterns, relationships, intrinsic characteristics of the datapoints and make decisions and predictions based on their understanding. The process of machine learning typically involves dividing the data into two key subsets: training data and testing data.

9.10.1 Training Data

Training data is fundamental in 'teaching' a machine learning model how to make predictions. This dataset allows the model to learn the relationship between input features and the target outcome. Typically, training data is more extensive than testing data because exposing the model to a broader array of examples enables it to identify and learn from meaningful patterns more effectively.

This process is analogous to how humans learn from experience; however, machine learning algorithms require a significantly larger volume of data to detect patterns and learn efficiently. As a model is provided with more relevant training data, its performance and accuracy in making predictions improve progressively.

9.10.2 Testing Data

After a machine learning model (which is basically a statistical equation) is developed using training data, it must be tested with new, unseen data to evaluate its performance. This subset is known as testing data, sometimes referred to as validation data. Test

data plays a crucial role in assessing the effectiveness of the model's training and fine-tuning the algorithm for enhanced results.

The model uses test data to make predictions, which are then compared against the actual outcomes to gauge metrics such as accuracy, precision, recall, and $F1$-score. This evaluation helps to ensure that the model can generalize its learning to new situations and perform reliably in real-world applications.

9.10.3 How Much Data Is Required to Train a Model?

The volume of data needed to effectively train a machine learning model is influenced by several key factors:

1. **Complexity of the Problem**: Simpler problems, such as those that can be addressed with linear regression, may require relatively modest amounts of data. In contrast, more complex challenges, like those tackled by neural networks, typically demand significantly larger datasets to capture the intricacies of the problem and enable the model to learn effectively.
2. **Complexity of the Learning Algorithm**: The sophistication of the algorithm also dictates data requirements. For instance, an image classification system, which often utilizes complex algorithms like convolutional neural networks, generally needs a larger dataset compared to simpler regression models. This is because complex models have more parameters to learn, requiring more examples to generalize well without overfitting.
3. **Quality of the Data**: The cleanliness and representativeness of the data play a crucial role as well. High-quality, representative data can reduce the amount of data needed for training because the model can learn the true underlying patterns more efficiently. Conversely, noisy or unrepresentative data necessitates larger datasets to discern genuine signals amidst the noise.
4. **Feature Dimensionality**: Models with a high number of input features (high-dimensional feature space) often require more data to prevent overfitting. Overfitting occurs when a model learns noise rather than the signal from the data. Techniques such as feature selection or dimensionality reduction can help to mitigate the risk of overfitting by simplifying the model's input space.

As with gaining experience, the adage 'the more data, the better the training' holds true, provided there is a sufficient subset of data reserved for testing and evaluating the model's performance. This ensures that the model not only learns well but also can apply its learning effectively to new, unseen data.

9.10.4 Evolution of Testing and Training

The evolution of testing and training methodologies in machine learning has been crucial to the advancement of the field. Initially, when machine learning was in its nascent stages, the approaches to training and testing models were quite rudimentary and ad hoc. Researchers would typically train models on available datasets and then test their performance on separate datasets. This method was straightforward but lacked rigor, especially as the algorithms and datasets grew in complexity and size.

As the need for more structured methods became evident, the train-test split emerged as a pivotal development. This approach involves dividing the data randomly into two distinct sets: one for training the model and the other for evaluating its performance. The train-test split method is particularly effective in preventing overfitting, ensuring that the model learns to generalize from the training data rather than merely memorizing it.

Further refinement led to the introduction of cross-validation, an enhancement over the train-test split. Cross-validation involves partitioning the data into multiple subsets and then cyclically using different subsets for training and testing the model. This technique reduces variance in performance assessment and provides a more accurate measure of a model's effectiveness.

In recent years, the field has continued to evolve with developments addressing specific challenges such as imbalanced datasets, missing data, and dynamically changing data environments. These advancements reflect the ongoing effort to refine machine learning training and testing practices, ensuring models are both robust and adaptable to real-world conditions.

9.10.5 Major Developments in Recent Years

As the field of machine learning advances, we can anticipate further innovations in testing and training methodologies. These developments will likely expand the capabilities and accessibility of machine learning, opening up new applications and improving existing ones across various industries. Some of the recent advances are discussed below.

Generative adversarial networks (GANs) are a powerful class of neural networks used for generating synthetic data that closely mimics real data. This capability is particularly valuable in improving the performance of machine learning models, especially in scenarios where real data is limited or sensitive. By augmenting datasets with high-quality synthetic data, GANs can help models to learn more generalizable patterns, thereby enhancing their performance on new, unseen data.

Feature engineering techniques are another crucial tool in the machine learning toolkit. By creating new features from existing data, these techniques can uncover additional insights and relationships that were not initially apparent. This can significantly enrich the model's learning environment, providing it with a deeper and

more nuanced understanding of the data, which in turn can lead to improved model performance.

In transfer learning, a model developed for one task is repurposed for a related but different task. This approach is efficient as it leverages learned features from large and diverse datasets. It reduces the time and computational resources required, compared to training a new model from scratch. It is particularly useful in domains where labeled data is scarce or expensive to obtain.

AutoML tools streamline the machine learning pipeline by automating the selection and tuning of algorithms. These tools are designed to achieve optimal performance on a given dataset. They make advanced machine learning techniques more accessible to non-experts and accelerate the development and deployment of models.

Explainable AI (XAI) focuses on making the decision-making processes of AI systems transparent and understandable to human users. By identifying key features and filtering out noise, XAI not only enhances the training process but also builds trust through clear explanations of how models make their predictions. Additionally, it enables continuous improvement of models through insights derived from human feedback.

9.10.6 Test-Train Split

Train-test split, thus, is a model validation procedure that allows you to simulate how a model would perform on new/unseen data. The steps are as follows:

1. **Arrange the Data**: Organize the data by dividing it into "Features" (inputs) and "Target" (output). The "Features" are the variables the model uses to learn, and the "Target" is the outcome it tries to predict.
2. **Split the Data**: Divide the dataset into two parts: a training dataset and a testing dataset. The proportion in which the data is split can vary depending on the specific needs of the project and the size of the data. Common splits include 70:30 or 80:20, where the larger portion is used for training and the smaller for testing.
3. **Train the Model**: Use the training dataset to train your machine learning model. This step involves the model learning to make predictions by understanding the relationships between the features and the target variable, based on the provided data.
4. **Test the Model**: Evaluate the performance of the model using the testing dataset. This step is critical as it provides insights into how the model will perform in real-world scenarios by making predictions on data it has never seen before (Fig. 9.12).

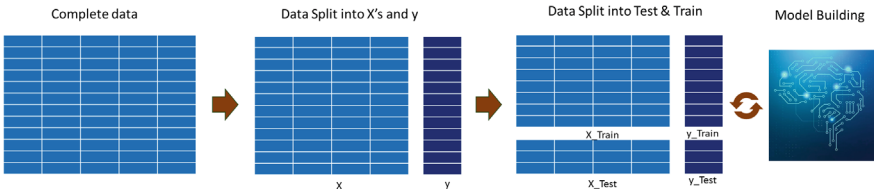


Fig. 9.12 Test versus train

9.10.7 Factors Influencing Test-Train Split

The train-test split, a critical aspect of model validation, is influenced by several factors that determine the amount of data needed for effective training and testing:

1. **Complexity of the Problem**: The inherent complexity of the problem influences the size of the dataset required. Typically, more complex problems, such as those involving multiple interactions between variables or non-linear relationships, necessitate larger datasets to capture the underlying patterns adequately.
2. **Model Complexity**: The complexity of the model, often determined by the number of parameters it includes, directly impacts data requirements. Models with a higher number of parameters (like deep neural networks) require substantial amounts of data to train effectively without overfitting, whereas simpler models might achieve optimal performance with considerably less data.
3. **Data Quality**: The quality of the dataset plays a crucial role. High-quality, clean, and well-preprocessed data can significantly enhance model training efficiency, allowing for better performance with fewer data points. Conversely, datasets with a lot of noise or errors typically require more extensive data to distil meaningful insights.
4. **Feature Dimensionality**: Models dealing with high-dimensional data (many features) often need more data to prevent overfitting. Techniques such as feature selection and dimensionality reduction can help mitigate this by reducing the number of input features and focusing on the most relevant information.
5. **Data Imbalance**: If the dataset has imbalanced classes, it may skew the model's ability to generalize. Strategies such as oversampling the minority class or using synthetic data generation can help to balance the dataset and provide a more effective training outcome.
6. **Desired Model Performance**: The level of performance you aim to achieve with your model also dictates the amount of data needed. Higher accuracy and precision require more nuanced understanding of the data, often necessitating larger datasets to fine-tune the model's predictions.
7. **Exploratory Data Analysis (EDA)**: Conducting a thorough exploratory data analysis can provide insights into the data's characteristics, helping to determine whether the dataset size is adequate to capture all relevant patterns and interactions.

8. **Cross-validation**: Implementing cross-validation techniques, such as k-fold cross-validation, helps in evaluating the model's ability to generalize across different subsets of data. This not only confirms the robustness of the model but also indicates whether the data volume is sufficient to achieve consistent and reliable performance across different data splits.

9.10.8 Using Train-Test Split in Python

Python has a lot of modules that can be used in machine learning. The one that is used for train-test split is called sklearn.model_selection.train_test_split(). We need to divide our data into features (X) and labels (y). The dataframe gets divided into x_train, x_test, y_train and y_test. x_train and y_train sets are used for training and fitting the model. The x_test and y_test sets are used for testing the model if it is predicting the right outputs/labels. It is suggested to keep our train sets larger than the test sets.

Syntax: *sklearn.model_selection.train_test_split().*

Parameters:

Arrays: Acceptable inputs include lists, NumPy arrays, SciPy sparse matrices, and Pandas dataframes.

test_size:

- Type: int or float (default: None).
- Description: Specifies the proportion of the dataset to include in the test split.
- If a float, it should be within the range 0.0 to 1.0 and denotes the fraction of the dataset to use as test data.
- If an int, it represents the absolute number of test samples.
- If None, the value is set to the complement of the train size or 0.25 if the train size is also None.

train_size:

Type: int or float (default: None).
Description: Specifies the proportion of the dataset to include in the train split. It can be a float representing a fraction of the dataset or an int representing the absolute number of train samples. If None, the remainder of the dataset not specified by test_size is used.

random_state:

- Type: int (default: None).
- Description: Controls the shuffling applied to the data before the split. Providing an integer value ensures repeatable outputs across multiple function calls.

shuffle:

Table 9.5 Test versus train split

Scenario	Test-train split ratio	Reference
Small to medium-sized datasets	80:20	Jiang et al. (2021)
Large datasets	60:40	Hanczar et al. (2010)
Datasets with a risk of overfitting	70:30	Bradley (1997)
Datasets where there is a need to balance evaluation on both the training and test sets	50:50	Saxena et al. (2020)

- Type: Boolean (default: True).
- Description: Determines whether or not to shuffle the data before splitting. If set to False, stratify must be None.

stratify:

- Type: array-like (default: None).
- Description: Provides class labels for data stratification. If not None, data is split in a stratified fashion using these as the class labels.

returns:

- Type: list.
- Description: Returns a list containing the split datasets.

9.10.9 Most Used Split Ratios

The following are some of the most used test-train split ratios in various scenarios (Table 9.5).

9.10.10 Example of Test-Train Split in House Price Prediction

A little earlier, we explored the basics of simple linear regression using a small dataset of just 10 data points. The 10 data points were manually entered for analysis. However, as the amount of data increases, manually inputting data becomes impractical and inefficient. To manage larger datasets effectively, we can utilize tools like Excel, which allow us to organize vast amounts of data efficiently in a spreadsheet format.

Using an Excel file not only streamlines the data management process but also facilitates the creation and refinement of a linear regression model. The fundamental concept remains unchanged: we aim to predict the price of a house based on its size. Yet, with the advantage of a more extensive dataset, our model can potentially achieve

greater accuracy and provide more reliable insights due to the richer information at its disposal. This approach enhances our ability to make well-informed predictions by leveraging the power of larger, more complex datasets.

- Import necessary modules

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
```

- Load the dataset from a csv file

```
df = pd.read_csv('/content/housing1.csv')
```

- Create a linear regression model

```
model = LinearRegression()
```

- Define the dependent (y) and independent (X) variables

```
X = df[['area']]
y = df['price']
```

- Split the data

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

random_state = 42 is used to set the seed for the random number generator. This means that the way the data is split into training and test sets will be the same every time you run the code. It ensures that the random process of dividing the data is repeatable. By setting it to the number 42, or any other number, you make sure that anyone else who runs this code with the same dataset will get the exact same training and test sets, which is useful for comparing results or for reproducing experiments.

But Why the Number 42?

The number 42 is often used in examples as a reference to Douglas Adams' "The Hitchhiker's Guide to the Galaxy," where 42 is "the Answer to the Ultimate Question of Life, the Universe, and Everything." In practice, any number could be used for the random_state (Fig. 9.13).

- Create the model

```
model = LinearRegression()
```

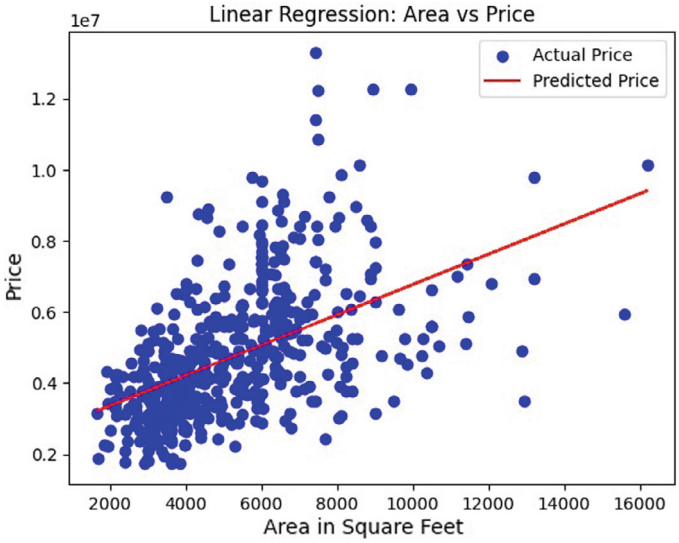


Fig. 9.13 Scatter plot for a larger dataset

- Fit the model

```
model.fit(X_train, y_train)
```

- Evaluate the model

```
train_error = mean_squared_error(y_train, model.predict(X_train))
test_error = mean_squared_error(y_test, model.predict(X_test))
```

- Plotting the results

```
plt.scatter(df['area'], df['price'], color='blue', label='Actual
Price')
plt.plot(X, model.predict(X), color='red', label='Predicted
Price')
plt.title('Linear Regression: Area vs Price')
plt.xlabel('Area in Square Feet')
plt.ylabel('Price')
plt.legend()
plt.show()
```

- Display the training and test errors

```
print(f"Training Error: {train_error}")
print(f"Test Error: {test_error}")
```

- Output of the program

```
Training Error: 2204738681379.341
Test Error: 3675286604768.185
```

Cross-Validation in Machine Learning (O'Hara-Wild 2019)

Cross-validation is a crucial technique in machine learning used to assess the performance of a model on unseen data. It involves splitting the training dataset into multiple subsets or "folds." The model is then trained on a combination of these folds and validated on the remaining fold. This process is repeated multiple times, with each fold serving as the validation set once. The average performance across all validation sets provides a more robust estimate of the model's effectiveness on new, unseen data.

There Are Two Main Types of Cross-Validation

In **K-fold cross-validation**, the training data is split into K folds. The model is then trained on $K - 1$ folds of the data and evaluated on the remaining fold. This process is repeated K times, and the performance of the model on the test folds is averaged to get an estimate of the model's performance on new data.

In **leave-one-out cross-validation**, the training data is split into K folds, where K is equal to the number of data points in the training set. The model is then trained on all the data except for one data point, and evaluated on the remaining data point. This process is repeated K times, and the performance of the model on the test data points is averaged to get an estimate of the model's performance on new data.

Benefits of using cross-validation:

- It offers a more precise estimate of the model's performance on unseen data.
- It helps to minimize the variance in the model's performance estimates.
- It aids in detecting overfitting.
- It assists in tuning the model's hyperparameters.

Drawbacks of using cross-validation:

- It can be computationally expensive, especially for large datasets.
- It can be difficult to interpret the results of cross-validation, especially for complex models.

Cross-validation is a more robust way to evaluate the performance, as it reduces the variance of the model's performance estimates. It is an improved version of test-train split.

We will apply cross-validation programmatically in a subsequent chapter.

9.11 Fitting a Polynomial Regression

Following our exploration of simple linear regression, let us look at polynomial regression, which is another vital concept in the field of machine learning and data analysis.

While simple linear regression is effective for linear relationships, it falls short in dealing with more complex, non-linear data patterns. This is where polynomial regression becomes particularly useful.

Polynomial regression is essentially an extension of linear regression. It allows us to model the relationship between the independent variable x and the dependent variable y as an nth degree polynomial.

This approach allows us to move beyond straight lines and capture curves, fitting a wider variety of data points more accurately.

The general form of a polynomial regression equation is:

$$y = a + bx + cx^2 + dx^3 + \cdots + nx^n + \varepsilon.$$

At its core, polynomial regression involves fitting a polynomial equation to a set of data points. This process involves several key steps:

(i) Choosing the Degree of the Polynomial: The first step is to decide the degree of the polynomial. The degree is simply how many powers of x are included in the equation. For example, a second-degree polynomial (quadratic) will include terms up to x^2, a third-degree (cubic) up to x^3, and so on. The higher the degree, the more curves (more number of changes) the line can have.

(ii) Constructing the Equation: Once the degree is chosen, the algorithm constructs the polynomial equation. This equation will have coefficients (like a, b, c, in the general form) that need to be determined.

(iii) Fitting the Model: The fitting process involves calculating the coefficients that result in the best fit curve for your data. 'Best fit' means the curve has the least amount of error in representing your data points. This is usually achieved through a method called 'least squares', which minimizes the sum of the squares of the differences between the observed and predicted values.

(iv) Predicting and Analyzing: With the coefficients determined, the polynomial equation can now be used to make predictions. For any given value of x, you can calculate the corresponding y using the equation. This is useful for both predicting future values and understanding the underlying trends in your data.

9.11.1 Why Polynomial Regression?

Polynomial regression is powerful because it can model data with both linear and non-linear relationships. This makes it more versatile than simple linear regression for real-world scenarios where data often shows complex trends and patterns. However,

it's important to choose the appropriate degree for the polynomial. Too low, and you might not capture the complexity of the data; too high, and you risk overfitting, where the model is too tailored to the specific dataset and may not perform well with new data.

9.11.2 *Implementation of Polynomial Regression in Excel for House Price Prediction (with Respect to Area)*

We are going to continue with the example introduced in the previous chapter. Instead of a linear regression, we will fit a polynomial regression for the same dataset of area and price.

- Instead of linear trendline, choose polynomial trendline (Fig. 9.14).
- We get the following output (Fig. 9.15)
- Let us enable display of equation and *R*-squared value (Fig. 9.16).

By using polynomial regression, *R*-squared value has improved from 95.37 to 99.39.

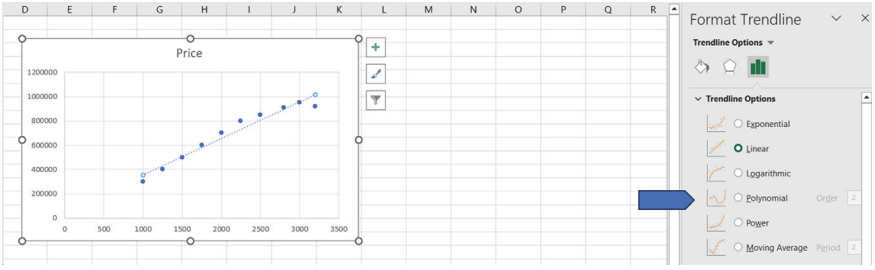


Fig. 9.14 Polynomial regression in Excel 1

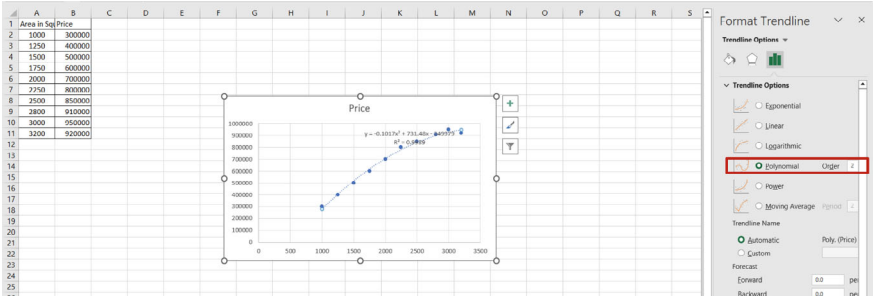


Fig. 9.15 Polynomial regression in Excel 2

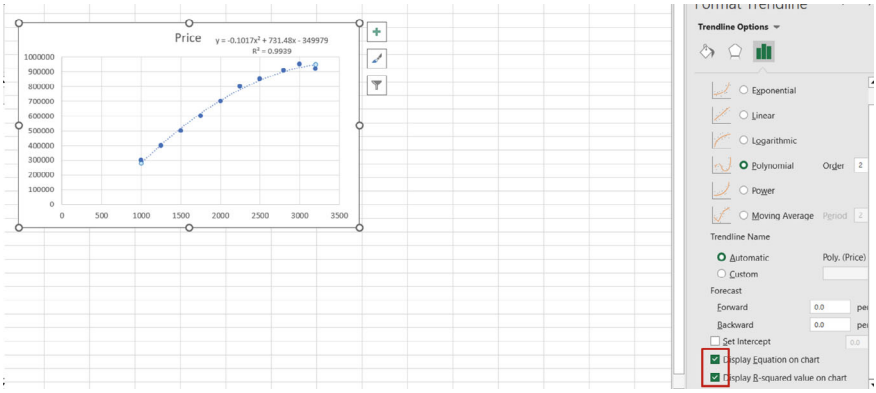


Fig. 9.16 Polynomial regression in Excel 3

9.12 Implementation of Polynomial Regression Using Python for House Price Prediction (with Respect to Area)

- Import necessary modules

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
```

- Define the data

```
area = np.array([1000, 1250, 1500, 1750, 2000, 2250, 2500, 2800, 3000, 3200]).reshape(-1, 1)
price = np.array([300000, 400000, 500000, 600000, 700000, 800000, 850000, 910000, 950000,
920000])
```

- Transform the feature matrix to include polynomial features

```
poly_features = PolynomialFeatures(degree=2)
area_poly = poly_features.fit_transform(area)
```

- Fit a polynomial regression model

```
model = LinearRegression()
model.fit(area_poly, price)
```

- Make predictions

```
predictions = model.predict(area_poly)
```

- Compute *R*-squared value

```
r_squared = r2_score(price, predictions)
```

- Print the *R*-squared value

```
print("R-squared value:", r_squared)
```

- Plot the original data and the polynomial regression curve

```
plt.scatter(area, price, label='Original Data')
plt.plot(area, predictions, color='red', label='Polynomial Regression')
plt.xlabel('Area')
plt.ylabel('Price')
plt.legend()
plt.show()
```

9.13 UnderFit Versus OverFit

In the world of machine learning, achieving the right balance in model accuracy is crucial. This balance is often disrupted by two common problems: underfitting and overfitting. Understanding these concepts is key to building effective models, and they are closely linked to the concepts of bias and variance, as well as their trade-off (Fig. 9.17).

9.13.1 Underfitting: Too Simple to Be Effective

Underfitting occurs when a model is too simple to capture the underlying complexity of the data. For example, fitting a straight line to data points that clearly form a curved

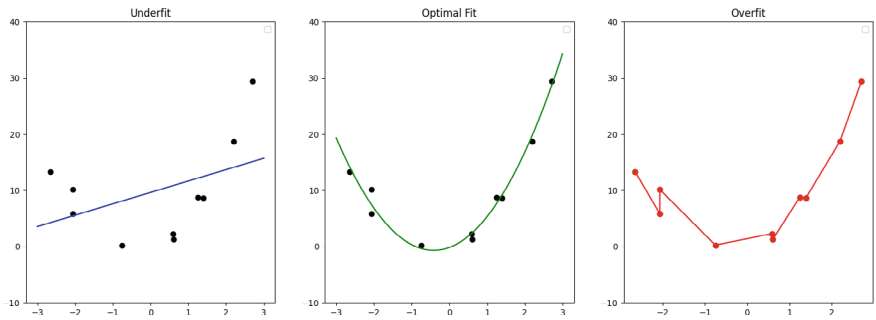


Fig. 9.17 Underfit, overfit, and optimal fit

pattern would result in underfitting. Such a model has high bias, which means it makes strong assumptions about the shape of the data and consequently misses important trends. This results in poor performance on both training and new, unseen data.

9.13.2 Overfitting: Too Complex and Overly Specific

Overfitting occurs when a model is too complex, capturing not only the underlying trend but also the noise in the data. If a model attempts to create a perfect curve passing through every single data point, including random fluctuations, it may perform exceptionally well on the training data but is likely to perform poorly on new, unseen data. Such a model exhibits high variance, indicating an oversensitivity to the specific details of the training data.

The Bias-Variance Trade-off (Dar et al. 2021; James et al. 2013)

The concepts of underfitting and overfitting are closely linked to the bias-variance trade-off. Bias refers to errors resulting from incorrect assumptions in the learning algorithm. High bias can cause an algorithm to overlook the relevant relationships between features and target outputs, leading to underfitting. Variance, on the other hand, refers to errors due to the model's sensitivity to fluctuations in the training set. High variance can cause overfitting, where the model captures random noise in the training data instead of the intended patterns.

9.13.3 An Example for Clarity

If you are trying to teach a child to recognize cats, and you only show them pictures of black cats, they might think all cats are black, leading to underfitting. If you show them pictures of black cats, tigers, and leopards, and tell them these are all common cats, they might start thinking any large feline is a cat, leading to overfitting. The goal is to provide a balanced learning experience that captures the essence of what makes a cat a cat, without including every possible variation or restricting the definition too much.

If a student preparing for a history exam studies only events from one specific decade, they may find themselves ill-equipped for an exam that covers multiple centuries and regions, exemplifying underfitting where their preparation was too narrow. On the other hand, if they over-prepare by memorizing every single detail from the textbook, including minor facts unlikely to be tested, this represents overfitting, where their focus on excessive detail can hinder their ability to recall more significant historical events during the exam. Ideally, the student should seek a balance in their study habits, ensuring a thorough understanding of major themes and key details without being overwhelmed by irrelevant information, mirroring the need for a well-tuned model that generalizes well to new data in machine learning.

9.13.4 Exploring Underfit and Overfit in Detail

Bias and variance are two aspects of model error. Underfitting, with high bias and low variance, arises when a model is too simplistic, missing important patterns in the data and consistently performing poorly. Overfitting, with low bias and high variance, occurs when a model is overly complex, capturing noise in the training data and thus performing well on training data but poorly on new data. Achieving a good model involves balancing these two to avoid both underfitting and overfitting.

Bias: The difference between the predicted value and the actual value is known as bias. A model that makes incorrect predictions about a dataset is considered a biased model. Such a model oversimplifies the target function to make it easier to learn.

Attributes of a High-Bias Model

(i) Underfitting: A model with high bias tends to underfit the data by oversimplifying the solution, often resulting in a linear function.
(ii) Oversimplification: The simplicity of a biased model prevents it from capturing complex features in the training data, making it ineffective for solving complex problems.
(iii) Low Training Accuracy: The inability of the high-bias model to effectively process training data leads to high training loss and low training accuracy.

Ways to Reduce the Bias in a Model

- Add more features from the data to increase the model's complexity.
- Increase the number of training iterations to allow the model to learn more complex patterns from the data.
- Utilize non-linear and non-parametric algorithms.
- Decrease regularization to enable the model to learn the training set more effectively and prevent underfitting.
- Consider using a new model architecture as a last resort if the above methods do not yield satisfactory results.

Variance: Variance refers to the extent of variability in the target function due to changes in the training data. A model with high variance considers almost every noise and fluctuation in the data, leading to overfitting.

Attributes of a High-Variance Model

(i) Overfitting: A high variance model tends to overfit the data by being overly complex. This results in a model that fits the training data exceptionally well, including noise and outliers, but performs poorly on unseen data.
(ii) Overcomplexity: Due to its complexity, a high-variance model captures not only the underlying patterns but also the noise in the training data. This makes it less generalizable and ineffective for accurately predicting new, unseen data.
(iii) High Training Accuracy, Low Test Accuracy: A high-variance model achieves high training accuracy due to its ability to capture all nuances, including noise,

in the training set. However, this overfitting leads to poor performance on test data, evidenced by a significant drop in test accuracy compared to training accuracy.

(iv) Sensitivity to Data Fluctuations: High-variance models are sensitive to minor fluctuations in the training data. Small changes in the training set can lead to significant changes in the model's predictions, indicating a lack of stability and robustness.

(v) Need for Large Datasets: These models often require large amounts of data to train effectively. Without sufficient data, they are prone to overfitting and struggle to generalize well to new data.

(vi) Complex Decision Boundaries: In classification problems, high-variance models tend to form very complex decision boundaries that twist and turn to accommodate the training data points, often at the expense of losing the broader, simpler trend.

(vii) Computationally Intensive: Due to their complexity, high-variance models typically require more computational resources for training and inference, making them less efficient in terms of computational cost.

Solution to High Variance

- Reducing the number of features in the dataset can help streamline the model's complexity.
- Increasing the size of the training dataset can help balance the complexity of more intricate models.
- Choosing algorithms with lower complexity can be an effective approach.
- Careful tuning of hyperparameters can help prevent overfitting.
- Increased regularization at different input levels can reduce model complexity and prevent overfitting.
- As a last resort, consider adopting a new model architecture if previous methods do not yield satisfactory results.

Bias-Variance Trade-off (Dar et al. 2021; James et al. 2013)

Bias and variance are complementary to each other; increasing one often results in the decrease of the other and vice versa. Finding the optimal balance between these two is known as the bias-variance trade-off.

A desirable algorithm should strike a balance, avoiding both underfitting and overfitting. The ultimate objective of all machine learning algorithms is to generate a model that minimizes both bias and variance simultaneously.

9.13.5 How to Check for Underfit or Overfit (Including k*-fold Split of Dataset) Programmatically*

- Dataset contains minutes of exercising and weight reduced.

- Import necessary modules

```
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
```

- Load the data

```
Minutes_of_Exercise =
np.array([30,35,40,45,50,55,60,65,70,75,80,85,90,95,100,105,110,115,
120,125]).reshape(-1, 1)
Weight_reduction = np.array([1.2, 1.4, 1.7, 2.1, 2.2, 2.3, 3, 2.8,
3.2, 3.4, 3.5, 4, 4.1, 4.7, 4.9, 4.5, 4.1, 4.4, 4.3, 4.1])
```

- Split the data

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

- Create the model

```
model = LinearRegression()
```

- Training the model

```
model.fit(X_train, y_train)
```

- Evaluate the model

```
train_error = mean_squared_error(y_train, model.predict(X_train))
test_error = mean_squared_error(y_test, model.predict(X_test))
```

- Perform cross-validation

```
cv_scores = cross_val_score(model, Minutes_of_Exercise,
Weight_reduction, cv=3)
```

- Display the training and test errors

```
print(f"Training Error: {train_error}")
print(f"Test Error: {test_error}")
print(f"Cross-validation Scores: {cv_scores}")
print(f"Mean CV Score: {np.mean(cv_scores)}")
```

- Plotting the results

```
plt.scatter(Minutes_of_Exercise, Weight_reduction, color='blue')
plt.title('Linear Regression: Exercise Vs Weight Reduction')
plt.xlabel('Minutes of Exercise')
plt.ylabel('Weight Reduction')
plt.legend()
plt.show()
```

- Output of the program

```
Training Error: 0.18491498265721618
Test Error: 0.17721990880360883
Cross-validation Scores: [ -1.72741617 0.23777339 -17.07500306]
Mean CV Score: -6.188215280851651
```

- Interpretation of the output

  Training Error (0.1849): This is the error the model makes on the training data, which it has seen and learned from. A lower training error generally indicates that the model is fitting well to the training data. In this case, the training error is reasonably low, suggesting that the model fits the training data to a certain extent.
  Test Error (0.1772): This is the error on the test data, which the model has not seen during training. It is used to gauge how well the model generalizes to new data. The test error is slightly lower than the training error, which is a good sign. It suggests that the model generalizes well and is not overfitting the training data.
  Cross-validation Scores: [− 1.7274, 0.2378, − 17.0750]: Cross-validation involves dividing the dataset into parts, training the model on some parts, and testing it on others. This helps in assessing the model's performance more robustly. The scores can vary depending on the model and the type of scoring used. In this case, the scores are quite varied: one is slightly positive, and the others are negative, with one being significantly negative.
  Mean CV Score (− 6.1882): This is the average of the cross-validation scores. A negative mean score, especially one that is significantly negative like in this case, suggests that the model might not be performing consistently across different subsets of the data.

In summary, this model seems to perform reasonably well on both the training and test datasets, as indicated by the relatively low errors. However, the varied and negative cross-validation scores raise concerns. (Such a performance is expected since the dataset has been created for educational purposes). The significant variation in the CV scores suggests that the model's performance is inconsistent across different subsets of the data. This inconsistency can be a sign of several issues, such as:

- The model might be overfitting specific parts of the data but underfitting others.

Table 9.6 Underfitting and overfitting

Aspect	Underfitting	Overfitting
Definition	The model is too simple to capture the underlying patterns in the data	The model is too complex and captures almost every noise in the data along with the underlying pattern
Model complexity	Low complexity The model lacks sufficient parameters or flexibility to learn from the data	High complexity The model has too many parameters or excess flexibility
Training performance	Performs poorly on the training data	Performs exceptionally well on the training data, often perfectly fitting it
Testing performance	Performs poorly on new, unseen data (test data)	Poor performance on new, unseen data due to memorizing rather than learning from the training data
Generalization	Poor generalization due to oversimplification (missing relevant details in the data)	Poor generalization, capturing noise instead of true patterns
Symptoms	– High bias – High errors on both training and testing data – Simplistic model assumptions	– High variance – Low error on training data but high error on testing data – Overly complex model (capturing noise)
Causes	– Too few features – Insufficient or irrelevant features – Excessive regularization	– Too many parameters – Limited training data – Insufficient regularization
Solutions	– Add relevant features – Reduce regularization	– Simplify the model – Collect more training data – Apply regularization – Use cross-validation

- The dataset might have high variability or outliers affecting the model's performance.
- The model or the features used might not be capturing the underlying patterns in the data effectively.

To improve the model, consider:

- Reviewing and preprocessing your data to ensure quality and consistency.
- Experimenting with different models or adjusting the current model's complexity.
- Examining the features used in the model to ensure they are relevant and informative.

Table 9.6 summarizes the differences between underfitting and overfitting and how to address them.

References

Bradley AP (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. Mach Learn 38(1):3–19

Dar Y, Muthukumar V, Baraniuk RG (2021) A farewell to the bias-variance tradeoff An overview of the theory of overparameterized machine learning. IEEE Signal Process Mag 38(1):164–181

Freedman DA (2009) The evolution of linear regression. Ann Stat 37(5):2862–2892

Hanczar et al (2010) Data splitting for machine learning with small sample sizes. Expert Syst Appl 37(4):3736–3744

James G, Witten D, Hastie T, Tibshirani R (2013) An introduction to statistical learning: with applications in R. Springer Science & Business Media

Jiang B, Liu Y, Zhang W, Wang H (2021) A critical look at the current train/test split in machine learning. ACM Trans Intell Syst Technol (TIST) 12(3):55:1–55:31

Nasteski V (2017) An overview of the supervised machine learning methods. Horizons Ser B 4(1):17–22

O'Hara-Wild M (2019) Cross-validation: a practical guide. Chapman & Hall/CRC

Saxena A, Kumar S, Kumar V (2020) Evolution in machine learning model selection and evaluation: a survey. ACM Comput Surv (CSUR) 52(6):89:1–89:49

Chapter 10
Multiple Linear Regression

In Multiple Linear Regression, Every Variable Adds a New Dimension to the Story

Abstract Building upon the foundation laid by Simple Linear Regression (SLR), this chapter introduces readers to Multiple Linear Regression (MLR)—a powerful technique enabling the modeling of relationships involving multiple independent variables. Readers will learn the key assumptions underlying MLR, gaining clarity on how meeting these assumptions significantly impacts model reliability. The chapter also provides a concise historical perspective on how regression techniques evolved, highlighting their growing complexity and capability. Practical use cases across various industries illustrate MLR's versatility in real-world applications. Readers will receive a detailed, programmatic walkthrough demonstrating the end-to-end implementation of MLR, enhancing their practical skills. Essential concepts critical to accuracy improvement, such as managing multicollinearity, detecting and handling outliers, addressing null values, and encoding categorical variables, are covered thoroughly. Additionally, advanced optimization methods, including hyperparameter tuning and grid search, are introduced clearly with hands-on coding examples. Ultimately, this chapter empowers readers to build robust, accurate multiple regression models, reinforcing best practices for real-world machine learning tasks.

Keywords Multiple linear regression · Multicollinearity · Variance inflation factor · VIF · Outliers · Encoding · Data transformation · One-hot encoding · Ordinal encoding · Label encoding · Binary encoding · Mean encoding · Null values · Regularization · L1 · L2 · Lasso · Ridge · Elastic Net · Normalization · Standardization · Robust scaling · Grid search · Hyperparameter · GridsearchCV

10.1 Toward Multiple Linear Regression

Simple linear regression is suitable when only one independent variable is used to predict a dependent variable, such as predicting house prices based solely on square footage. However, house prices can also be influenced by multiple factors, including

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_10

the number of rooms, available facilities, and geographic location. To accommodate multiple independent variables in predicting a single dependent variable, like the price of a house, multiple linear regression is employed. This approach estimates the relationship between several independent variables and one dependent variable, providing a more comprehensive analysis of factors that influence the outcome.

You can use multiple linear regression when you want to know:

- How strong the relationship is between two or more independent variables and one dependent variable (for example, how rainfall, temperature, and amount of fertilizer affect the crop growth).
- The value of the dependent variable at a certain value of the independent variables (for instance, the expected yield of a crop at certain levels of rainfall, temperature, and fertilizer addition).

10.1.1 Assumptions Made in Multiple Linear Regression

The assumptions of multiple linear regression are essentially the same as those in simple linear regression:

(i) Homogeneity of Variance (Homoscedasticity): This means that the error's size in our predictions does not significantly vary across different values of the independent variable.
(ii) Independence of Observations: The dataset consists of observations collected through statistically valid sampling methods, and there are no hidden relationships among variables.
(iii) Normality: The data is expected to adhere to a normal distribution. The distribution of the data points around the mean is bell-shaped and symmetrical.
(iv) Linearity: The best-fitting line through the data points is a straight line, as opposed to a curve or some other non-linear pattern.

In multiple linear regression, it is also crucial to avoid high correlation between two independent variables. Let us inspect a model that predicts the salary of a person that has highly correlated variables. For example, when two independent variables, such as "Years of Education" and "Highest Degree Attained," are highly correlated, it is advisable to include only one of them in the model to avoid multicollinearity. Multicollinearity occurs when correlated variables make it difficult to distinguish their individual impacts on the dependent variable.

If "Years of Education" and "Highest Degree Attained" are strongly correlated (e.g., coefficient $r^2 = 0.6$), it is wise to choose the variable that aligns better with your research question or has a stronger theoretical basis. This will ensure a more stable and interpretable regression model.

If the data used does not meet these conditions, the result produced by a linear regression model might not be reliable.

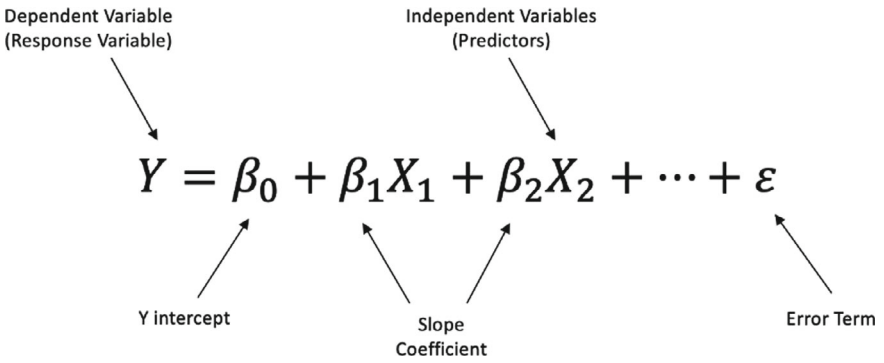


Fig. 10.1 Multiple linear regression equation

The formula for a multiple linear regression is (Fig. 10.1):

- Y is the predicted value of the dependent variable
- B_0 is the y-intercept (value of y when all other parameters are set to 0)
- B_1X_1 is the regression coefficient (B_1) of the first independent variable (X_1) (a.k.a. the effect that increasing the value of the independent variable has on the predicted y value)
- (Repeat the same for all the independent variables you are testing)
- B_nX_n is the regression coefficient of the last independent variable
- e = model error (a.k.a. how much variation there is in our estimate of y).

To Find the Best-Fit Line

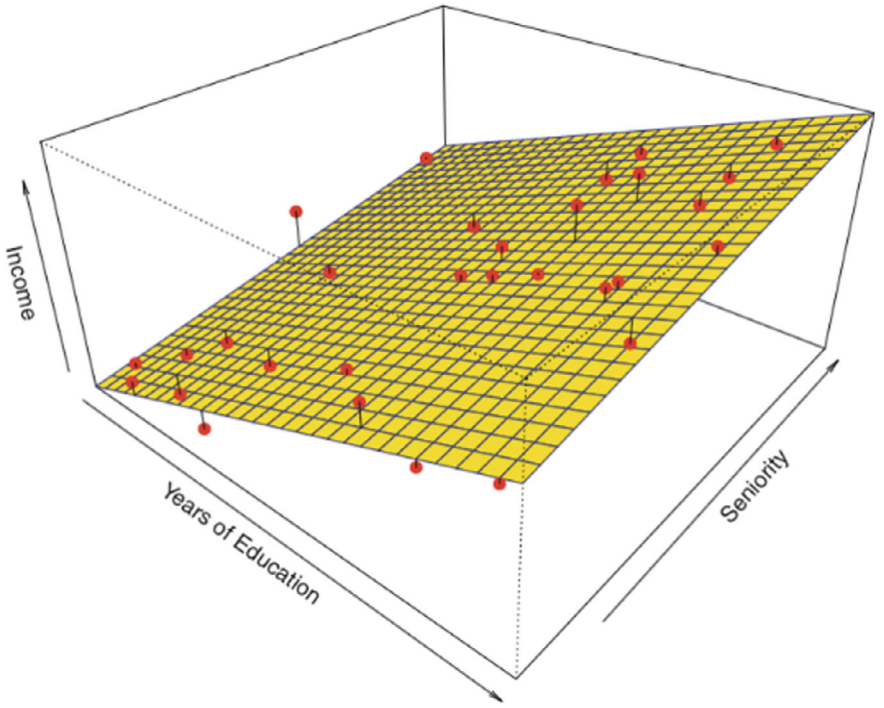
In multiple linear regression (MLR), the first step involves identifying regression coefficients that minimize the mean squared error (MSE), effectively determining the best-fit line that considers all independent variables together.

Following this, MLR computes the t-statistic for each regression coefficient. This t-statistic, part of the t-test, assesses whether each coefficient significantly differs from zero, thus evaluating the strength and significance of the relationship between each independent variable and the dependent variable. Alongside the t-statistic, the p-value for each coefficient is calculated to determine the likelihood that the observed relationship occurred by chance.

If the p-value associated with a coefficient falls below a predefined threshold (commonly set at 0.05), the null hypothesis—which asserts that there is no relationship between the independent and dependent variables—is rejected. A p-value below this threshold suggests that the relationship is statistically significant and unlikely to be due to random chance.

Example plot of predicting income from different factors (James et al. 2013).

Figure 10.2 visually shows the impact of two independent variables years of education and seniority on income.



Source: James et al. *Introduction to Statistical Learning* (Springer 2013)

Fig. 10.2 Multiple linear regression in 3D

Multiple linear regression thus finds the best-fit line for each independent variable by minimizing the overall error and testing the significance of each variable's contribution to the model.

10.1.2 Evolution of Multiple Linear Regression

The history of multiple linear regression (MLR) can be traced back to the early eighteenth century when the French mathematician Pierre-Simon Laplace developed a method for fitting linear models to data. However, it wasn't until the early twentieth century that MLR became widely used.

One of the earliest papers on MLR was published by Karl Pearson in 1908, where he derived the least squares method for fitting linear models to data. This method remains the most widely used technique for fitting MLR models today.

Ronald A. Fisher also significantly contributed to the development of MLR. In his book "Statistical Methods for Research Workers," published in 1925, Fisher

introduced the concept of the F-test for significance testing in MLR. The F-test is still widely used today to test the significance of the relationship between the response variable and the explanatory variables.

Since the early twentieth century, extensive research on MLR has led to the development of new methods for fitting MLR models, model selection, and interpreting MLR results.

10.1.3 Applications of MLR in Machine Learning

In the field of machine learning, multiple linear regression (MLR) is regarded as a supervised learning algorithm. With the rise of machine learning in the twentieth and twenty-first centuries, MLR has found a new role as a predictive modeling technique, integrated into various machine learning frameworks, thus becoming an essential part of predictive analytics. While advanced algorithms like random forests, gradient boosting, and neural networks have surpassed MLR in many predictive modeling tasks, MLR remains a useful tool for explanatory analysis and as a baseline model.

MLR is used to build **predictive models** for numerous tasks, such as predicting customer churn, forecasting sales, and diagnosing diseases. For example, researchers at Google AI have utilized MLR to develop models predicting the likelihood of a user clicking on an advertisement.

MLR helps to **identify the most important features** for predicting a response variable, reducing the complexity of machine learning models and improving their performance. For example, data scientists have employed MLR to create a method for selecting features to predict the risk of heart disease.

MLR is valuable for explaining **causal relationships** between variables, aiding in understanding how different factors impact a specific outcome.

Techniques such as ridge regression and Lasso regression were developed to handle multicollinearity and prevent overfitting in MLR. These regularization methods have become popular for enhancing model stability and accuracy.

10.1.4 Implementation of MLR Using Python for House Price Prediction

- Dependent variable: House Price
- Independent variables:
  - Area
  - Number of bedrooms
  - Number of bathrooms
  - Number of stories or floors
  - Number of parking

- Sample data of the dataset (Table 10.1)

- Import necessary modules

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
```

- Load the data from a CSV file

```
df = pd.read_csv('/content/housing_data.csv')
```

- Define the dependent variable (y) and independent variables (X)

```
X = df.drop('price', axis=1)
y = df['price']
```

- Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Create the multiple linear regression model

```
mlr_model = LinearRegression()
```

- Predict price on test data

```
y_pred = mlr_model.predict(X_test)
```

- Calculate the coefficient of determination (R^2)

```
r_squared = r2_score(y_test, y_pred).round(1)
```

- Calculate the mean absolute error and rounding off the value to two decimal points

Table. 10.1 Sample dataset for multiple linear regression

Price	Area	Bedrooms	Bathrooms	Stories	Parking
13,300,000	7420	4	2	3	2
12,250,000	8960	4	4	4	3
12,250,000	9960	3	2	2	2
12,215,000	7500	4	2	2	3
11,410,000	7420	4	1	2	2
10,850,000	7500	3	3	1	2

```
mae = mean_absolute_error(y_test, y_pred).round(2)
```

- Print *R* squared and MAE

```
print(f'R^2: {r_squared}')
print(f'MAE: {mae}')
```

- Result of the program
  R^2: 0.5
  MAE: 1127483.35

10.1.5 Use Cases of MLR

MLR is a powerful and versatile tool that can be used to solve a wide range of problems. It is a valuable tool for anyone who wants to understand and use data to make better decisions.

MLR can be utilized to build models that **predict future revenue** from customers. This information helps to develop targeted marketing campaigns aimed at retaining customers, thereby reducing churn.

MLR can **forecast sales** by analyzing historical data and other factors such as economic conditions and marketing efforts. This predictive insight allows businesses to make informed decisions about inventory levels, pricing, and marketing strategies.

MLR can develop models that **predict the outcomes of medical tests** based on a patient's symptoms and medical history. This capability enhances the accuracy and efficiency of diagnoses, leading to better patient care.

MLR can **identify risk factors for various diseases** or other events. This information is crucial for developing preventive measures and early detection programs, improving overall public health outcomes.

MLR can study the **relationships between multiple variables**, providing insights into complex systems. This understanding supports better decision-making in fields ranging from economics to environmental science.

10.1.6 Limitations of MLR

Multiple linear regression (MLR) is a valuable statistical tool but has inherent limitations due to its assumptions. These include:

(i) **Linearity**: MLR assumes a linear relationship between the response and explanatory variables, which can lead to inaccuracies if the relationship is non-linear.
(ii) **Normality**: It assumes normally distributed residuals. If this is not the case, the results may not be reliable.

(iii) **Homoscedasticity**: MLR requires constant variance across the residuals, without which the results could be skewed.
(iv) **Multicollinearity**: The technique assumes little to no correlation among explanatory variables. High correlation can compromise the model's accuracy.
(v) **Overfitting**: MLR can overfit the data, fitting the training set too closely and failing to generalize well to new data.

Being aware of these limitations is crucial when using MLR to model relationships between variables and taking measures to mitigate their effects.

10.2 Multicollinearity

Following our exploration of regression analysis, an important concept to understand in this context is multicollinearity.

Multicollinearity occurs when two or more independent variables in a regression model are highly correlated, allowing one variable to be accurately predicted from the others. This shared variance makes it challenging to isolate the individual impact of each variable on the dependent variable.

A common method to detect multicollinearity is through the variance inflation factor (VIF), which measures the extent of correlation between a predictor and the rest of the predictors in the model. VIF is computed by regressing each predictor against all other predictors in the model. The formula for VIF is:

$$\text{VIF} = \frac{1}{1 - \boldsymbol{R}_i^2}$$

where $\boldsymbol{R}_i^2$ is the coefficient of determination of a regressor of predictor i on all other predictors.

Interpreting VIF values

- A VIF of 1 means there is no correlation between the ith predictor and any other predictor.
- A VIF between 1 and 5 indicates moderate correlation, but is often considered acceptable.
- A VIF above 5 can signify problematic levels of multicollinearity, warranting further investigation or corrective measures.

10.2.1 Cutoff for Multicollinearity

While a VIF of 5 is commonly cited as a cutoff for concern, this can vary depending on the context and field of study. In some scenarios, a higher VIF may be tolerable, whereas in others, even a lower VIF could be problematic.

Consider a real estate pricing model where both the size of the house (in square feet) and the number of bedrooms are predictors. These two variables are likely to be correlated, as larger homes typically have more bedrooms. High multicollinearity in this case can make it challenging to discern how each variable independently affects house prices.

Multicollinearity doesn't reduce the predictive power or reliability of the model as a whole, but it does affect the reliability of the individual predictors. This can make it difficult to identify the true relationship between predictors and the outcome, leading to unreliable and unstable estimates of regression coefficients.

10.2.2 Addressing Multicollinearity Programmatically

Now, let us assess and address multicollinearity within the dataset. Our objective is to eliminate multicollinear dependencies and subsequently recompute the accuracy metrics.

- Import necessary modules

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

- Load the data from a CSV file

```
df = pd.read_csv('/content/housing_data.csv')
```

- Define the dependent variable (y) and independent variables (X)

```
X = df.drop('price', axis=1)
y = df['price']
```

- Calculate variance inflation factor (VIF) for each feature

```
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]
    print("VIF for each feature:\n", vif_data)
```

- Identify the feature with the highest VIF

```
max_vif_feature = vif_data.sort_values(by="VIF", ascending=False).iloc[0]['feature']
print("Removing feature due to high VIF:", max_vif_feature)
```

- Remove the feature with the highest VIF

```
X_new = X.drop(max_vif_feature, axis=1)
```

- Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Create the multiple linear regression model

```
mlr_model = LinearRegression()
```

- Fit the model with the training data

```
model.fit(X_train, y_train)
```

- Predict the prices on the test data

```
y_pred = model.predict(X_test)
```

- Calculate *R*-squared and MAE

```
r_squared = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
```

- Print *R* squared and MAE

```
print(f'R^2: {r_squared}')
print(f'MAE: {mae}')
```

- Result of the program (Fig. 10.3)

```
VIF for each feature:
      feature        VIF
0        area   6.737363
1    bedrooms  13.652525
2   bathrooms   9.155151
3     stories   6.683602
4     parking   1.901606
Removing feature due to high VIF: bedrooms
R^2: 0.5427029584600664
MAE: 1135357.3851648744
```

Fig. 10.3 Program output showing VIF

10.3 Outliers

In data analysis and machine learning, outliers are data points that deviate significantly from other observations, potentially due to variability in the data or experimental errors. These outliers can substantially affect the outcomes of data analysis and statistical modeling, including machine learning models.

For example, consider a dataset detailing the ages of people in a community where most individuals are between 20 and 60 years old. In such a scenario, an age of 95 would stand out as an outlier, as it drastically differs from most of the data points.

Outliers can be identified in various ways:

- **Statistical Methods**: One common method is the use of the interquartile range (IQR). Any data point that falls below the first quartile (Q1) minus 1.5 times the IQR or above the third quartile (Q3) plus 1.5 times the IQR is considered an outlier. IQR is Q3–Q1.
- **Visualization Tools**: Box plots and scatter plots can visually aid in identifying outliers.

Once identified, outliers can be managed in several ways:

- **Trimming/Removing**: Remove the outliers from the dataset.
- **Imputation**: Replace outliers with values that are more representative of the data, like the mean, median, or a value estimated from a model.
- **Transformation**: Apply a transformation to the data to reduce the effect of outliers, such as logarithmic or cube root transformations. Below table demonstrates how the effect of outliers is reduced when we apply transformation to data.

As you can see in Table 10.2, the range has reduced considerably, after the transformation.

Table 10.2 Outlier transformation

	Original data	Cube root transformation	Log transformation
	3	1.44	0.48
	5	1.71	0.7
	8	2	0.9
	14	2.41	1.15
	86	4.41	1.93
Range (Max–Min)	83	2.97	1.45

10.3.1 Why Outlier Management Is Necessary

Outliers can significantly skew the results of a data analysis:

- In regression models, outliers can have a large effect on the slope, intercept, and statistical tests.
- In clustering analyses, outliers can distort the groups formed.
- For classification problems, outliers can result in misclassification.
- Models can overfit if they try to accommodate the outliers, leading to poor performance on new, unseen data.

10.3.2 Addressing Outliers Programmatically

Now, let's assess and address outliers within the dataset. Our objective is to manage outliers and subsequently recompute the accuracy metrics.

- Import necessary modules

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

- Load the data from a CSV file

```
df = pd.read_csv('/content/housing_data.csv')
```

- Plotting the box plots for price and area columns

```
plt.figure(figsize=(12, 6))
```

- Box plot for price

```
plt.subplot(1, 2, 1)
sns.boxplot(x=df['price'])
plt.title('Box Plot of Price')
```

- Box plot for area

```
plt.subplot(1, 2, 2)
sns.boxplot(x=df['area'])
plt.title('Box Plot of Area')

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

- Defining the columns to check for outliers

```
outlier_columns = ['price', 'area']
```

- Calculate and print the percentage of outliers for 'price' and 'area' columns

```
for column in outlier_columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    outlier_condition = (df[column] < (Q1 - 1.5 * IQR)) | (df[column] > (Q3 + 1.5 * IQR))
    outliers_percentage = np.mean(outlier_condition) * 100
    print(f'Percentage of outliers in {column}:    {outliers_percentage:.2f}%')
```

- Transform the outliers using log transformation in price and area columns

```
for column in outlier_columns:
    df[column] = np.log1p(df[column])
```

- Define the dependent variable (y) and independent variables (X)

```
X = df.drop('price', axis=1)
y = df['price']
```

- Split data into training and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Create the multiple linear regression model

```
mlr_model = LinearRegression()
```

- Fit the model with the training data

```
model.fit(X_train, y_train)
```

- Predict the prices on the test data

```
y_pred = model.predict(X_test)
```

- Calculate *R*-squared

```
r_squared = r2_score(y_test, y_pred)
```

- Print *R* squared and MAE

```
print(f'R-squared after transformation: {r_squared}')
```

- Result of the program (Figs. 10.4 and 10.5)
  Percentage of outliers in price: 2.75%
  Percentage of outliers in area: 2.20%
  R-squared after transformation: 0.5595051997328596.

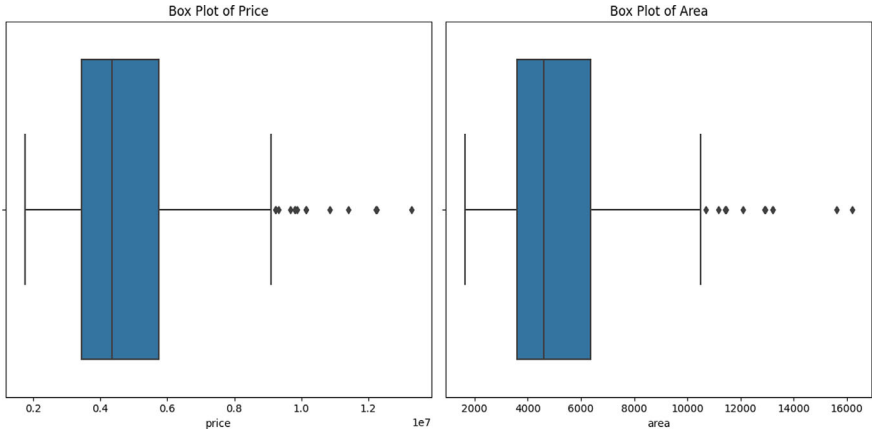


Fig. 10.4 Program output showing box plot of price

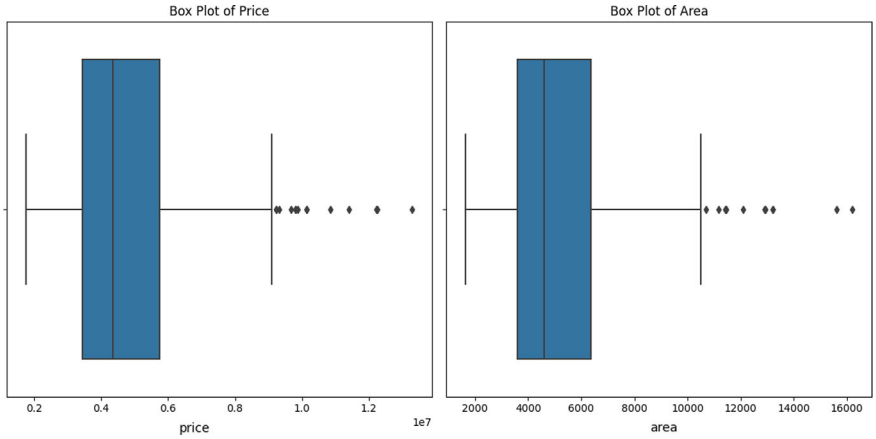


Fig. 10.5 Program output showing box plot of area

10.4 Encoding

Encoding is a critical process in data analysis and machine learning, particularly for handling categorical data. It involves transforming categorical variables, such as gender or country names, from text format into a numerical format. This transformation is essential because most machine learning algorithms require data to be in numerical form to perform mathematical operations effectively.

Categorical data are commonly found in many datasets and must be converted into numeric types to be utilized by machine learning models. Proper encoding allows these algorithms to process the data accurately and ensures that the categorical variables contribute appropriately to the model's performance. Without encoding, these models would be unable to interpret the categorical data, significantly hindering their predictive capabilities.

10.4.1 Different Types of Encoding

(i) **One-Hot Encoding**: This method generates a new binary column for each category in the original data. For example, if a column contains three categories, it will be converted into three binary columns, each representing one category with a 1 for presence and 0 for absence. Though straightforward, one-hot encoding can significantly increase the number of columns if there are many unique categories (Fig. 10.6).

(ii) **Label Encoding**: Each category is assigned a unique integer based on alphabetical ordering. Although simple, this method can introduce the issue of the model incorrectly interpreting these integers as having a natural order or ranking that does not actually exist (Fig. 10.7).

(iii) **Ordinal Encoding**: Used when the categorical variable has a natural ordering, this method assigns numbers to categories that reflect their inherent order or rank, making it suitable for ordinal data (Fig. 10.8).

(iv) **Frequency or Count Encoding**: Categories are replaced by their frequency or count in the dataset. This approach can be effective in managing variables with a large number of categories by reducing dimensionality while preserving some information about category prevalence.

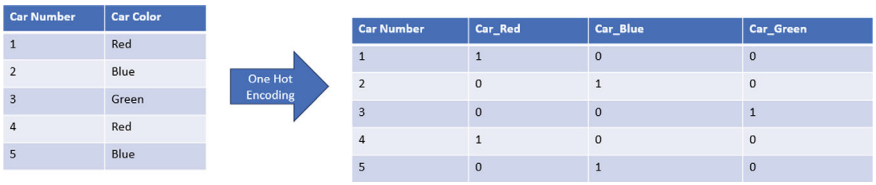


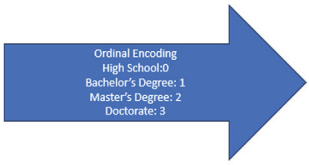
Fig. 10.6 One-hot encoding

Car Number	Car Color
1	Red
2	Blue
3	Green
4	Red
5	Blue

Car Number	Car Color Label
1	0
2	1
3	2
4	0
5	1

Fig. 10.7 Label encoding

Student Number	Education Level
1	High School
2	Bachelor's Degree
3	Master's Degree
4	Doctorate
5	High School



Student Number	Education Level Encoded
1	0
2	1
3	2
4	3
5	1

Fig. 10.8 Ordinal encoding

(v) **Binary Encoding**: A hybrid of hashing and one-hot encoding, this method converts categories into binary numbers and then splits these numbers into separate columns. It can be more efficient than one-hot encoding when dealing with numerous categories.

(vi) **Mean Encoding**: Each category is replaced with the average target value associated with that category. This technique is especially beneficial for features with high cardinality, as it directly incorporates the target variable's influence into the encoding.

Now let us apply encoding to house price dataset and compute the accuracy programmatically.

- Dataset (https://www.kaggle.com/datasets/yasserh/housing-prices-dataset) now has numeric and non-numeric variables (Table 10.3)

- Dependent variable: Price
- Numeric independent variables
  - Area of the house
  - Number of bedrooms
  - Number of bathrooms
  - Number of stories or floors.
- Non-numeric independent variables
  - Does the house have a guest room or not
  - Does the house have a basement or not
  - Does the house have water heater or not

Table 10.3 Part of the dataset for encoding

Price	Area	Bedrooms	Bathrooms	Stories	Mainroad	Guestroom	Basement	Hot-water heating	Air-conditioning	Parking	Pref. area	Furnishing status
13,300,000	7420	4	2	3	Yes	No	No	No	Yes	2	Yes	Furnished
12,250,000	8960	4	4	4	Yes	No	No	No	Yes	3	No	Furnished
12,250,000	9960	3	2	2	Yes	No	Yes	No	No	2	Yes	Semi-furnished
12,215,000	7500	4	2	2	Yes	No	Yes	No	Yes	3	Yes	Furnished
11,410,000	7420	4	1	2	Yes	Yes	Yes	No	Yes	2	No	Furnished
10,850,000	7500	3	3	1	Yes	No	Yes	No	Yes	2	Yes	Semi-furnished

- Does the house have air conditioner or not
- Does the house come with parking area or not
- Is the house located in preferred areas in the city or not
- What is the extent of furnishing available in the house.

- Import necessary modules

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

- Load the data from a CSV file

```
df = pd.read_csv('/content/housing.csv')
```

- Function to detect and encode non-numeric columns

```
def encode_non_numeric(df):
  for column in df.select_dtypes(include=['object']).columns:
    # If binary categorical data, use Label Encoding
    if df[column].nunique() == 2:
      df[column] = LabelEncoder().fit_transform(df[column])
    # If non-binary categorical data, use One Hot Encoding
    else:
      df = pd.get_dummies(df, columns=[column], drop_first=True)
  return df
```

- Encode non-numeric data

```
df_encoded = encode_non_numeric(df)
```

- Define the dependent variable (y) and independent variables (X)

```
X = df.drop('price', axis=1)
y = df['price']
```

- Split data into training and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Create the multiple linear regression model

```
mlr_model = LinearRegression()
```

- Fit the model with the training data

```
model.fit(X_train, y_train)
```

- Predict the prices on the test data

```
y_pred = model.predict(X_test)
```

- Calculate *R*-squared

```
r_squared = r2_score(y_test, y_pred)
```

- Print *R* squared

```
print(f'R-squared after transformation: {r_squared}')
```

- Result of the program
  R^2: 0.65292426

10.5 Null Values

Null values, frequently represented as not a number (NaN) or missing entries in datasets, pose a common challenge in machine learning. If not managed properly, they can significantly impair model performance.

Missing or unknown data points in a dataset, or null values, can arise from several sources such as errors during data collection, data transmission corruption, or instances where information was simply not recorded. Since most machine learning algorithms are not designed to handle null values inherently, it is crucial to identify and treat them effectively.

Leaving null values untreated can compromise the accuracy of models and lead to skewed analytical results. They may distort the distribution of data, resulting in unreliable outputs. Therefore, adequately handling null values is an essential step in the data preprocessing phase before training any model.

10.5.1 Methods of Handling Null Values

(i) **Deletion**: This method involves removing records with null values. Though this method is simple, this method can lead to loss of valuable data, which could hurt the model's performance, especially if the dataset is not large.

(ii) **Imputation with Mean/Median/Mode**: A common technique for handling missing values in numerical data is to replace them with the mean, median, or

mode of the respective column. This approach helps to maintain the overall data distribution.

(iii) **Predictive Imputation**: This method uses predictive modeling techniques to fill in null values, offering a potentially more accurate solution, especially when the missing data is not random.

(iv) **Assigning a Unique Category**: For categorical data, replacing null values with a new category, such as 'Unknown', can be effective, especially when the absence of data itself provides meaningful information.

(v) **Using Algorithms that Support Null Values**: Some machine learning algorithms, like decision trees, can intrinsically handle missing values, eliminating the need for imputation.

(vi) **K-Nearest Neighbors Imputation**: This technique fills in missing values using the K-Nearest neighbors approach, which identifies and uses the most similar data points.

(vii) **Last Observation Carried Forward (LOCF) or Next Observation Carried Backward (NOCB)**: These methods are useful in time series data, where missing values are replaced with the last or next observed values, respectively.

10.5.2 Addressing Null Values Programmatically

Leaving null values untreated can significantly affect a model's accuracy. Null values can introduce bias, leading to underfitting or overfitting of the model. Proper treatment ensures that the model learns from accurate and complete data, leading to better performance.

- Dataset under consideration contains null values as shown in Table 10.4.

- Import necessary modules

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
```

- Load the data from a CSV file

```
df = pd.read_csv('/content/housing_with_null.csv')
```

- Function to replace missing values

Table 10.4 Part of the dataset containing null values

Price	Area	Bedrooms	Bathrooms	Stories	Mainroad	Guestroom	Basement	Hotwaterheating	Airconditioning	Parking	Prefarea	Furnishingstatus
13,300,000	7420	4	2	3	Yes	No	No	No	Yes	2	Yes	Furnished
12,250,000	8960	4	4	4	Yes	No	No	No	Yes	3	No	
12,250,000		3	2	2		No	Yes	No	No	2	Yes	Semi-furnished
12,215,000	7500	4	2	2	Yes	No	yes	No	Yes	3	Yes	Furnished
11,410,000	7420	4	1	2	Yes	Yes	Yes	No		2	No	Furnished
10,850,000	7500	3	3	1	Yes	No	Yes	No	Yes	2	Yes	Semi-furnished

```
def replace_missing_values(df):
    for column in df.columns:
        if df[column].dtype == 'object':  # Non-numeric columns
            mode = df[column].mode()[0]
            df[column] = df[column].fillna(mode)
        else:  # Numeric columns
            median = df[column].median()
            df[column] = df[column].fillna(median)
    return df
```

- Replace missing values in the dataframe

```
df = replace_missing_values(df)
```

- Function to detect non-numeric columns

```
def encode_non_numeric(df):
    for column in df.select_dtypes(include=['object']).columns:
        # If binary categorical data, use Label Encoding
        if df[column].nunique() == 2:
            df[column] = LabelEncoder().fit_transform(df[column])
        # If non-binary categorical data, use One Hot Encoding
        else:
            df = pd.get_dummies(df, columns=[column], drop_first=True)
    return df
```

- Encode non-numeric data

```
df_encoded = encode_non_numeric(df)
```

- Define the dependent variable (y) and independent variables (X)

```
X = df_encoded.drop('price', axis=1)
y = df_encoded['price']
```

- Split data into training and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Create the multiple linear regression model

```
mlr_model = LinearRegression()
```

- Fit the model with the training data

```
mlr_model.fit(X_train, y_train)
```

- Predict the prices on the test data

```
y_pred = mlr_model.predict(X_test)
```

- Calculate *R*-squared

```
r_squared = r2_score(y_test, y_pred)
```

- Print *R* squared

```
print(f'R-squared after transformation: {r_squared}')
```

- Result of the program
  R^2:0.6054233923741501

10.6 Regularization

Regularization is a fundamental technique in machine learning designed to prevent overfitting, a scenario where a model not only learns the underlying patterns but also the noise in the data, which negatively impacts its performance on new, unseen data. By introducing a penalty to the loss function, regularization discourages overly complex models, promoting simpler ones that are more likely to generalize well.

$$\text{Regularization} = \text{Loss Function} + \text{Penalty}$$

There are several types of regularization.

10.6.1 L1 Regularization (Lasso)

This method adds a penalty equivalent to the absolute value of the coefficients. It encourages the model to drive some coefficients to zero, thus effectively removing certain features from the model. This feature elimination aspect makes L1 particularly useful for feature selection in models with many variables.

10.6.2 L2 Regularization (Ridge)

In this method, the penalty is the square of the magnitude of the coefficients. Unlike L1, L2 does not zero out coefficients but reduces their size. It is beneficial for dealing with multicollinearity and helps in shrinking the coefficients, thereby improving the model predictions while retaining all features.

10.6.3 Elastic Net

Combining the penalties of L1 and L2, Elastic Net is effective when dealing with datasets that have multiple correlated features. It can both eliminate irrelevant features (like L1) and manage multicollinearity (like L2), offering a versatile regularization choice.

Regularization is particularly crucial when dealing with complex models or datasets with numerous features, as it helps simplify the model, enhance efficiency, and reduce error risks on new data. Implementing regularization requires selecting the appropriate type (L1, L2, or Elastic Net) and tuning a hyperparameter that governs the penalty's strength. This tuning is typically done through experimentation to strike an optimal balance that minimizes overfitting while sustaining robust model performance.

The mean squared error, which is the loss function for the regression model, is optimized with the addition of penalty.

$$\text{MSE} = \frac{1}{n}\Sigma\left(y_{\text{predicted}} - y_{\text{original}}\right)^2.$$

L1 Lasso

$$\text{MSE} = \frac{1}{n}\Sigma\left(y_{\text{predicted}} - y_{\text{original}}\right)^2 + \lambda\sum_{i=1}^{n}|m_i|.$$

L2 Ridge

$$\text{MSE} = \frac{1}{n}\Sigma\left(y_{\text{predicted}} - y_{\text{original}}\right)^2 + \lambda\sum_{i=1}^{n}m_i^2.$$

10.6.4 Hyperparameter

The lambda ($\boldsymbol{\lambda}$) in the above is a regularization coefficient, which is also a hyperparameter. The absolute value of the slope is used in ridge and the square value of the slope is used in lasso. The hyperparameters are tuned using GridSearchCV, which is an optimization exercise where different values of the hyperparameters are assessed in an iterative fashion.

The difference between lasso and ridge is summarized in Table 10.5.

Table 10.5 Lasso and ridge

Feature	Lasso (L1 regularization)	Ridge (L2 regularization)
Penalty term	L1 adds the absolute values of the coefficients	L2 adds the squared values of the coefficients
Norm type	L1 norm	L2 norm
Effect on coefficients	Coefficients can shrink to zero, effectively performing feature selection	Coefficients are shrunk toward zero but not exactly to zero
Solution path	Non-smooth, can be discontinuous	Smooth and continuous
Handling multicollinearity	Can handle multicollinearity by setting some coefficients to zero	Does not naturally eliminate multicollinearity but minimizes its effect
Feature selection	Yes to a limited extent, due to zeroing out of coefficients	No. All features are kept but their influence is minimized
Bias-variance trade-off	Higher bias, potentially lower variance by eliminating variables	Lower bias, but can have higher variance if many variables are included
Model complexity	Produces simpler models with fewer variables	Tends to produce more complex models, with more variables included
Use case	When you have many features and expect only a few of them to be important	When you need to keep all features

10.6.5 Implementing Regularization to the House Price Dataset Programmatically

- Import necessary modules

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
```

- Load the data from a CSV file

```
df = pd.read_csv('/content/housing.csv')
```

- Encoding non-numeric data

```
for column in df.select_dtypes(include=['object']).columns:
    if df[column].nunique() == 2:
        df[column] = LabelEncoder().fit_transform(df[column])
    else:
        df = df.join(pd.get_dummies(df[column], prefix=column))
        df.drop(column, axis=1, inplace=True)
```

- Separate features (independent variables) and target variable (dependent variable)

```
X = df.drop('price', axis=1)
y = df['price']
```

- Feature scaling

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- Split the dataset into test and train datasets

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

- Linear regression (no regularization)

```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
r2_lr = r2_score(y_test, y_pred_lr)
```

- L1 regularization (Lasso)

```
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
y_pred_lasso = lasso_model.predict(X_test)
r2_lasso = r2_score(y_test, y_pred_lasso)
```

- L2 regularization (Ridge)

```
ridge_model = Ridge(alpha=0.1)
ridge_model.fit(X_train, y_train)
y_pred_ridge = ridge_model.predict(X_test)
r2_ridge = r2_score(y_test, y_pred_ridge)
```

- Elastic Net regularization

```
elastic_model = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_model.fit(X_train, y_train)
y_pred_elastic = elastic_model.predict(X_test)
r2_elastic = r2_score(y_test, y_pred_elastic)
```

- Print R squared values

```
print(f'R^2 (Linear Regression): {r2_lr}')
print(f'R^2 (Lasso): {r2_lasso}')
print(f'R^2 (Ridge): {r2_ridge}')
print(f'R^2 (Elastic Net): {r2_elastic}')
```

- Result of the program
  R^2 (Linear Regression): 0.6529242642153177
  R^2 (Lasso): 0.6529242267777824
  R^2 (Ridge): 0.6529136575217613
  R^2 (Elastic Net): 0.6500423648173266.

10.7 Scaling

In the above programmatic implementation of regularization, feature scaling was used. Feature scaling is the process of normalizing or standardizing the range of independent variables or features of data. Feature scaling is needed because many machine learning algorithms perform better when numerical input variables are scaled to a standard range. If one variable (say age) has values from 1 to 80 and another variable (say income) ranges from 20 to 500k, the scales of variables vary so widely that it interferes with the performance of the model. There are several methods to scale features to a uniform scale across features.

(i) **Normalization** scales features to a range of [0, 1]. Normalization is often done using the formula
(ii) $(x - \min)/(\max - \min)$, where min and max are the minimum and maximum values in a feature column.
(iii) **Standardization** scales features to have a mean of 0 and a standard deviation of 1. Standardization is accomplished using the formula $(x - \text{mean})/\text{standard deviation}$
(iv) **Robust Scaling** scales features using statistics that are robust to outliers. Robust scaling subtracts the median and divides by the interquartile range, thus reducing the influence of outliers.

10.7.1 Regularization with Grid Search

We saw earlier that regularization is used in machine learning to prevent models from overfitting, which occurs when a model becomes overly complex and closely fitted to the training data. To optimize regularization, hyperparameter tuning techniques like grid search are employed.

Before looking at the process, let us understand the key terms.

Grid Search: Imagine having numerous dials, each controlling a different aspect of your machine. Finding the optimal combination, akin to unlocking a complex lock, is a daunting task. Grid search systematically explores various settings of these dials to identify the most effective combination.

GridSearchCV: This tool from Python's Scikit-learn library integrates grid search with cross-validation. The "CV" stands for cross-validation, a technique to evaluate model performance. GridSearchCV not only tests different settings but also ensures they are robustly evaluated.

Parameters: These are configurations learned by the model during training, similar to ingredients in a recipe that influence the final dish.

Hyperparameters: These are the user-set configurations that dictate the model's overarching behavior, comparable to manual settings on a camera like brightness or zoom.

Hyperparameter Tuning: This involves adjusting the hyperparameters to find the most effective settings for the model, akin to fine-tuning camera settings to capture the perfect photograph. Grid search is one of the methods used for this purpose.

Alpha: Specifically related to regularization, alpha is a hyperparameter that controls the intensity of the regularization. Adjusting alpha modifies the balance between preventing overfitting and maintaining sensitivity to subtle data patterns.

10.8 Process of Grid Search

The process of grid search, which involves conducting an exhaustive search to identify the optimal settings for a machine learning model, includes the following steps.

(i) **Define the Parameter Grid**: Start by selecting the hyperparameters you want to tune and establishing a range of possible values for each. For instance, in a model utilizing regularization, you might test various strengths of regularization. These values create a 'grid' of potential combinations to explore.
(ii) **Choose a Model**: Select the type of machine learning model you wish to optimize, such as a linear regression model, a decision tree, or a neural network.
(iii) **Set up Cross-Validation**: Implement a cross-validation method to assess model performance. This involves dividing your dataset into several parts, or 'folds'. The model is trained on some folds and validated on others, ensuring it performs consistently across different subsets of your data.
(iv) **Apply Grid Search**: Employ a tool like GridSearchCV, within Python's Scikit-learn environment, to perform the grid search. This tool systematically trains your model with various hyperparameter combinations from your grid, each time evaluating the model's performance through cross-validation.

(v) **Evaluate the Results**: Grid search provides a performance score for each hyperparameter combination, indicating how well the model performed with those settings.
(vi) **Select the Best Combination**: After assessing all possible combinations, select the one that delivered the best performance. These are the optimal hyperparameter settings for your model.
(vii) **Final Model Training**: Train your model once more using the best hyperparameters on the entire dataset to fully prepare it for practical application.

Grid search is like methodically trying every key on a keyring to find the one that unlocks a door. It's a systematic approach to finding the optimal settings for a machine learning model, ensuring that you've thoroughly explored the possibilities to get the best performance.

10.8.1 Applying the Grid Search Process Programmatically to the House Price Dataset

- Import necessary modules

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.metrics import r2_score
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
```

- Load the data from a CSV file

```
df = pd.read_csv('/content/housing.csv')
```

- Encoding non-numeric data

```
for column in df.select_dtypes(include=['object']).columns:
  if df[column].nunique() == 2:
    df[column] = LabelEncoder().fit_transform(df[column])
  else:
    df = df.join(pd.get_dummies(df[column], prefix=column))
    df.drop(column, axis=1, inplace=True)
```

- Separate features (independent variables) and target variable (dependent variable)

```
X = df.drop('price', axis=1)
y = df['price']
```

- Feature scaling

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- Split the dataset into test and train datasets

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=
```

- Linear regression (no regularization)

```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
r2_lr = r2_score(y_test, y_pred_lr)
```

- L1 regularization (Lasso) with hyperparameter tuning

```
lasso_params = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
lasso_model = GridSearchCV(Lasso(), lasso_params, cv=5)
lasso_model.fit(X_train, y_train)
y_pred_lasso = lasso_model.predict(X_test)
r2_lasso = r2_score(y_test, y_pred_lasso)
```

- L2 regularization (ridge) with hyperparameter tuning

```
ridge_params = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
ridge_model = GridSearchCV(Ridge(), ridge_params, cv=5)
ridge_model.fit(X_train, y_train)
y_pred_ridge = ridge_model.predict(X_test)
r2_ridge = r2_score(y_test, y_pred_ridge)
```

- Elastic Net regularization with hyperparameter tuning

```
elastic_params = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100], 'l1_ratio': [0.2, 0.4, 0.6, 0.8]}
elastic_model = GridSearchCV(ElasticNet(), elastic_params, cv=5)
elastic_model.fit(X_train, y_train)
y_pred_elastic = elastic_model.predict(X_test)
r2_elastic = r2_score(y_test, y_pred_elastic)
```

- Print R squared values and best params

```
print(f'R^2 (Linear Regression): {r2_lr}')
print(f'R^2 (Lasso): {r2_lasso}, Best Params: {lasso_model.best_params_}')
print(f'R^2 (Ridge): {r2_ridge}, Best Params: {ridge_model.best_params_}')
print(f'R^2 (Elastic Net): {r2_elastic}, Best Params: {elastic_model.best_params_}')
```

- Result of the program

  R^2 (Linear Regression): 0.6529242642153177
  R^2 (Lasso): 0.6528856636032526, Best Params: {'alpha': 100}
  R^2 (Ridge): 0.6517359300483782, Best Params: {'alpha': 10}

R^2 (Elastic Net): 0.6478651645976948, Best Params: {'alpha': 0.1, 'l1_ratio': 0.2}

10.8.2 Understanding the Results of the Model

The *R*-squared values reported for the linear regression, Lasso, Ridge, and Elastic Net models provide a measure of how well each model's predictions fit the actual data. These values indicate that all models explain a similar proportion of the variance in the dependent variable.

The close *R*-squared values between the models, including the one without regularization (linear regression) and those with regularization (Lasso, Ridge, Elastic Net), suggest that the impact of regularization on this dataset might be minimal. The best parameters for Lasso and Ridge indicate relatively high 'alpha' values, which means stronger regularization is applied.

The warnings about the objective not converging imply that, despite the regularization, the models did not successfully find the optimal solution within the given computational constraints. This could mean that the number of iterations was too low, the data might not have been scaled appropriately, or the regularization strength (the 'alpha' value) was not set high enough to address the issues within the dataset that could be causing the optimization challenges.

In this context, the *R*-squared values after hyperparameter tuning give us a snapshot of the models' performance, but the convergence warnings suggest that these results may not be reliable. We might need to revisit the model training process, possibly scaling the features, adjusting the 'alpha' value, or allowing more iterations for the optimization algorithm to converge.

10.8.3 Other Hyperparameter Tuning Methods (Snoek et al. 2012; Li et al. 2017)

Bayesian optimization (BO) and hyperband are two popular hyperparameter tuning methods in machine learning.

Bayesian Optimization (BO) is a sequential model-based optimization algorithm that employs a Bayesian model to understand the relationship between hyperparameter values and model performance. It iteratively updates this model to make informed decisions about which hyperparameters to evaluate next, focusing on areas likely to yield the best results.

Hyperband is a resource allocation algorithm that efficiently utilizes a limited budget (e.g., time or computational resources) to tune hyperparameters. It works by dynamically allocating resources to promising hyperparameter configurations, thereby

quickly identifying the best-performing models while minimizing computational cost.

Bayesian Optimization with Hyperband (BOHB) combines the strengths of Bayesian optimization (BO) and hyperband to create an efficient and robust hyperparameter tuning method. BOHB uses the Bayesian optimization framework to model the relationship between hyperparameters and performance, guiding the search toward promising configurations. Simultaneously, it leverages hyperband's resource allocation strategy to efficiently manage computational resources, allowing for rapid and effective exploration of the hyperparameter space. This combination results in a method that balances exploration and exploitation, yielding high-performance models with reduced computational cost.

References

https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

James G, Witten D, Hastie T, Tibshirani R (2013) An introduction to statistical learning: with applications in R. Springer, New York

Li L, Jamieson K, DeSalvo G, Rostamizadeh A, Talwalkar A (2017) Hyperband: A novel bandit-based approach to hyperparameter optimization. J Mach Learn Res 18(18–558):1–51

Snoek J, Larochelle H, Adams RP (2012) Gaussian processes for Bayesian optimization of hyperparameters. In: Neural information processing systems (NIPS), pp 2851–2859

Chapter 11
Logistic Regression

Logistic Regression: Where Data Decides Between Yes or No

Abstract Following the comprehensive coverage of regression algorithms, this chapter introduces readers to classification algorithms, starting specifically with logistic regression—a foundational approach to solving classification problems in machine learning. Readers will clearly understand how logistic regression differs from linear regression by predicting categorical rather than continuous outcomes. The chapter discusses, in detail, various accuracy metrics such as confusion matrix, precision, recall, F1-score, ROC curve, and AUC, helping readers evaluate model performance effectively. Central concepts like cost functions and gradient descent are thoroughly explained, illustrating how logistic regression algorithms optimize their predictions. Readers also gain practical exposure through detailed, step-by-step programmatic examples, ensuring clarity on implementation and interpretation of logistic regression models. Real-world scenarios demonstrate the wide-ranging applications of classification tasks across industries. By the end of this chapter, readers will confidently understand, implement, and evaluate logistic regression models, building a solid foundation for exploring advanced classification algorithms in subsequent chapters.

Keywords Classifier · Logistic regression · Classification algorithm · Binary classifier · Multiclass classifier · Confusion matrix · Type 1 error · Type 2 error · True positive · False positive · True negative · False negative · Precision · Recall · Accuracy · $F1$-score · AUC · ROC curve · Log loss · Logistic function · Need for logistic regression · Accuracy in classification algorithm · Cost function · Gradient descent · Derivatives in machine learning

Simple and multiple linear regression focus on predicting numeric outcomes like estimating house or stock prices. What if our task is to discern whether emails are spam or legitimate, forecast a customer's likelihood of showing up for an appointment, or diagnose the presence of a disease in an individual?

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_11

In all these scenarios, the output comes to a 'Yes' or a 'No, a '0' or a '1', or other categorization. Such scenarios are managed by classification algorithms, and we will begin with a popular and widely used algorithm—logistic regression.

11.1 What Are Classification Algorithms?

Classification algorithms are a form of supervised learning that helps to categorize new observations into specific classes or groups based on the patterns learned from a labeled dataset. These classes can be binary, like "Yes or No," "0 or 1," "Spam or Not Spam," or multiclass like different genres of a movie.

In contrast to regression, where the goal is to predict a continuous value, classification deals with assigning data to discrete categories or labels. The output of a classification algorithm is a class or category, not a numeric value.

Classification relies on supervised learning, meaning that it requires input data with corresponding output labels. This labeled data is used to train the model, allowing it to make predictions on new, unlabeled data by assigning it to one of the defined categories.

In classification algorithm, a discrete output function (y) is mapped to input variable (x) as $y = f(X)$, where y is the categorical output and X is the independent variable.

The classification algorithm has classified the inputs into Class A and Class B. Data points in a class have features like each other and features that differentiate them from the other class (Fig. 11.1).

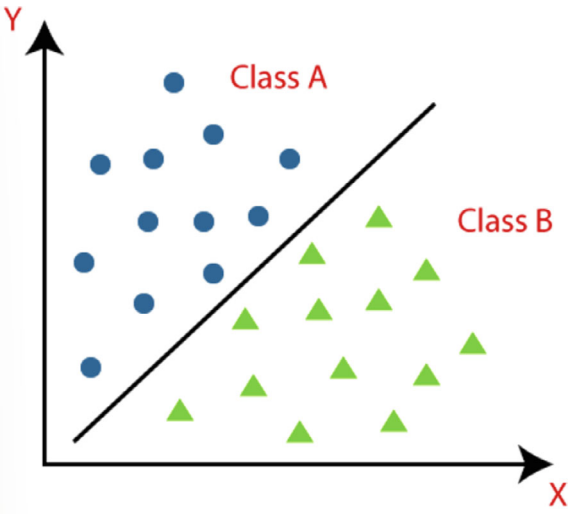


Fig. 11.1 Classification algorithm

The algorithm that implements classification is also called as a 'classifier.' There are two types of classifiers, as explained earlier.

Binary Classifier: If the classification problem has only two possible outcomes, then it is called as binary classifier.

Examples: YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.

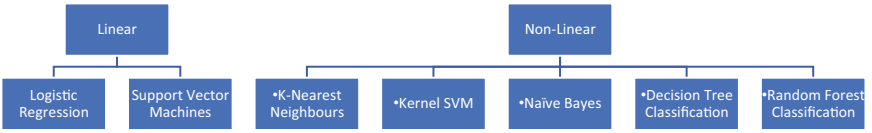
Multiclass Classifier: If a classification problem has more than two outcomes, then it is called as multiclass classifier.

Examples: Classifications of types of crops and classification of types of music.

Classification algorithms can be broadly classified into linear and non-linear as shown below.

Linear algorithms produce outcomes that change in a way that's directly proportional to the change in input. This means if you double the input, the output will double as well. They are usually simpler and easier to understand, implement, and analyze. Their behavior can often be represented by a straight line in a graph, hence the term 'linear'.

Nonlinear algorithms, on the other hand, deal with outcomes that don't change proportionally with changes in input. This can make them more complex and harder to predict. They are often more flexible and can model more complex relationships than linear algorithms. In a graph, their behavior is represented by curves or more complex shapes, not straight lines.



Another way to group the classification algorithms is based on the types of learners.

Lazy Learners: A lazy learner initially retains the training dataset and defers processing until it receives the test dataset. In such a model, classification is conducted based on the most closely corresponding data in the training set. This approach minimizes training time but requires more time to make predictions due to the on-the-fly analysis it performs during the testing phase.

Examples: K-NN algorithm, case-based reasoning.

Eager Learners: Eager learners construct a classification model by thoroughly analyzing the training dataset prior to the introduction of any test data. Contrary to lazy learners, eager learners invest more time in the learning phase to reduce the prediction time, thereby ensuring swift and efficient classification when new data is encountered.

Examples: Decision trees, Naïve Bayes, ANN.

11.2 Accuracy of Classification Algorithms

Similar to using multiple metrics to assess the accuracy of regression algorithms, multiple metrics are used to assess the classification algorithms.

1. Accuracy: Accuracy simply measures how often the classifier correctly predicts. We can define accuracy as the ratio of the number of correct predictions to the total number of predictions.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Accuracy is not always the best metric to consider in classification. In scenarios where data is imbalanced, such as spam email detection, credit card fraud detection, or medical diagnosis, relying solely on accuracy can be misleading.

For instance, if 99% of the cases belong to one class and only 1% to another, a model that always predicts the majority class can achieve high accuracy, but it is not necessarily effective at identifying the minority class.

2. Confusion Matrix: A confusion matrix is a tabular representation of the outcome of the classification algorithm. It provides a clear and concise representation of how well the model's predictions align with the actual class labels in a dataset.

The confusion matrix typically consists of four key values (Table 11.1):

- True Positive (TP)
  Actual value is positive and ML model also predicted a positive value.
- False Negative (FN)
  Actual value is positive, but ML model predicted a negative value.
- False Positive (FP)
  Actual value is negative, but ML model predicted a positive value.
- True Negative (TN)
  Actual value is negative and ML model also predicted a negative value.

Important Observations:

- Sum of all the actual positive values/cases = TP + FN.
- Sum of all the actual negative values/cases = FP + TN.
- Sum of all the positively predicted values/cases = TP + FP.
- Sum of all the negatively predicted values/cases = FN + TN.

Table 11.1 Confusion Matrix

		Actual values	
		Positive (1)	Negative (0)
Predicted values	Positive (1)	TP	FP
	Negative (0)	FN	TN

These are very useful for measuring the recall, precision, accuracy, and AUC-ROC curves.

The concept of TP, TN, FP, and FN can be confusing initially. Let us look at the below example to understand this clearly.

In the below example, true positive is when the test accurately diagnoses a patient with the disease. It's crucial for timely and effective treatment. Correctly identifying patients who don't have the disease (true negative) is just as important, ensuring that they don't undergo unnecessary treatments. At the same time, a false positive, where we diagnose a disease in a healthy patient, can lead to unnecessary stress and treatment. We should strive to minimize these occurrences. False negatives, where the test misses the disease in an affected patient, are particularly dangerous. They can delay essential treatment, worsening patient outcomes (Table 11.2).

Let us look at one example. In the below example, spam filter correctly identifying and isolating spam emails is true positive. Ensuring that non-spam emails rightly appear in the inbox is true negative. A high true negative rate means that the spam filters are effective, but they are not overbearing. When legitimate emails are mistakenly marked as spam, it results in false positives and disrupts communication. At the same time, a spam message slipping through into the inbox (false negative) not only annoys users but also poses security risks (Table 11.3).

3. Precision: It measures the proportion of correctly predicted positive observations to the total predicted positives. Precision is useful in the cases where false positive is more important than false negatives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

4. Recall (Sensitivity): It measures the proportion of actual positives that were correctly identified by the model. Recall is a useful metric in cases where false negative is more important than false positive.

Table 11.2 Accuracy in disease prediction

Actual positive	Actual negative
Predicted positive **True positive (TP)** The model correctly predicts the positive class The model predicts a disease and the patient actually has the disease	**Predicted negative** **False negative (FN)** The model incorrectly predicts the negative class when it is actually positive The model predicts no disease, but the patient actually has the disease
Predicted negative **False positive (FP)** The model incorrectly predicts the positive class when it is actually negative The model predicts a disease, but the patient is healthy	**Predicted positive** **True negative (TN)** The model correctly predicts the negative class The model predicts no disease and the patient is healthy

Table 11.3 Accuracy in spam email prediction

Actual spam	Actual not spam
Predicted spam **True positive (TP)** The email is correctly identified as spam A junk promotional email is correctly moved to the spam folder	**Predicted not spam** **False negative (FN)** The email is incorrectly marked as not spam, but it is actually spam A phishing email ends up in the inbox instead of the spam folder
Predicted not spam **False positive (FP)** The email is incorrectly identified as spam when it is actually not spam An important work email is mistakenly sent to the spam folder	**Predicted spam** **True negative (TN)** The email is correctly identified as not spam A regular email from a colleague lands in the inbox as it should

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

5. $F1$-Score: It is the harmonic mean of precision and recall.

$$F1\text{-Score} = 2 * \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$F1$-score could be an effective evaluation metric in the following cases:

- When FP and FN are equally costly.
- Adding more data does not effectively change the outcome.
- True negatives are high.

6. AUC-ROC: The Receiver Operator Characteristic (ROC) is a probability curve that plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold values and separates the 'signal' from the 'noise.'

The area under the curve (AUC) is the measure of the ability of a classifier to distinguish between classes. Greater the AUC, the better is the performance of the model at different threshold points between positive and negative classes.

This simply means that when AUC is equal to 1, the classifier can perfectly distinguish between all positive and negative class points. When AUC is equal to 0, the classifier would be predicting all negatives as positives and vice versa. When AUC is 0.5, the classifier is not able to distinguish between the positive and negative classes (Fig. 11.2).

In the above ROC graph, the X-axis value shows false positive rate (FPR), and Y-axis shows true positive rate (TPR). Higher the value of X means higher the number of false positives (FP) than true negatives (TN), while a higher Y-axis value indicates a higher number of TP than FN. So, the choice of the threshold depends on the ability to balance between FP and FN.

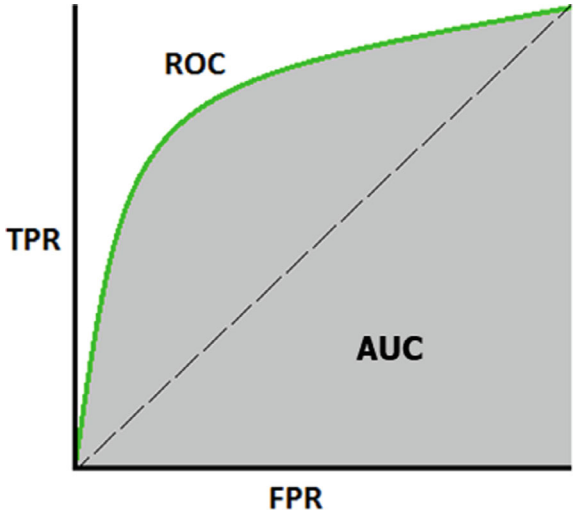


Fig. 11.2 ROC curve

7. Logistic Loss (Log Loss): Log loss, also known as logistic loss or cross-entropy loss, penalizes both types of errors, but especially those predictions that are confident and wrong. The closer the predicted probability is to the actual label, the lower the log loss. If the predicted probability diverges from the actual label, the log loss increases. Perfect predictions would have a log loss of 0

$$\log\ \text{loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - P_i)\right].$$

Metrics like accuracy, precision, recall are good ways to evaluate classification models for balanced datasets, but if the data is imbalanced, then other methods like ROC/AUC perform better in evaluating the model performance.

Table 11.4 summarizes the different accuracy metrics used in classification algorithms.

11.3 Introduction to Logistic Regression

Logistic regression is a statistical model that is used to predict the probability of a binary (e.g., yes/no, true/false) or multiclass outcome (e.g., genre of music) based on a set of independent variables. It is one of the most widely used machine learning algorithms and has been successfully applied in a wide range of domains.

Logistic regression is based on the logistic function, which is a non-linear function that maps a set of real numbers to a value between 0 and 1.

Table 11.4 Summary of accuracy measures in classification algorithms

Accuracy measure	Purpose	Question it answers	Limitations
Accuracy	To measure the proportion of correctly predicted instances out of all predictions	"What proportion of all predictions were correct?"	Can be misleading in imbalanced datasets
Precision	To measure the proportion of correctly predicted positive observations to the total predicted positives	"Of all instances predicted as positive, how many are actually positive?"	Does not take into account false negatives; not ideal when false negatives are costly
Recall (sensitivity)	To measure the proportion of actual positives that were correctly identified	"Of all the actual positives, how many did we correctly identify as positive?"	Does not consider false positives; critical when missing a positive is costly
$F1$-score	To balance the trade-off between precision and recall	"What is the balance between precision and recall for our model?"	Can be less informative than precision and recall in certain contexts
Specificity	To measure the proportion of actual negatives that were correctly identified	"Of all the actual negatives, how many did we correctly identify as negative?"	Does not consider false negatives; more relevant when avoiding false positives is crucial
Area under the ROC curve (AUC-ROC)	To measure the ability of a classifier to distinguish between classes	"How well can the model distinguish between the two classes?"	Can be overly optimistic in imbalanced datasets
Area under the precision–recall curve (AUC-PR)	To evaluate the classifier's performance in imbalanced datasets focusing on the positive class	"How well does the model perform in terms of precision and recall, particularly for the positive class?"	Less common and more difficult to interpret than AUC-ROC

To train a logistic regression model, we need to provide the model with a set of data that includes the independent variables and the binary or multiclass outcome. The model will then learn the relationship between the independent variables and the outcome and use this relationship to predict the probability of the outcome for new data points.

Once a logistic regression model has been trained, it can be used to make predictions about new data points. For example, we could use a logistic regression model to predict the probability of a patient having a certain disease based on their age, sex, and medical history. Or, we could use a logistic regression model to predict the probability of a customer making a purchase based on their browsing history and demographic data (Kleinbaum 2009).

11.3.1 Logistic Function

Logistic regression is expressed as a probability.

$$p = \frac{1}{1 + e^{-(mx+c)}}$$

The purpose of the logistic function is to take any input from the linear combination of the predictors, $mX + c$, which can range from negative infinity to positive infinity, and transform it into a valid probability value between 0 and 1.

This transformation is crucial because it allows the regression model to handle a binary dependent variable, which takes on two possible values, by predicting probabilities that are constrained to the [0, 1] interval, rather than predicting raw scores as linear regression does.

11.3.2 Need for Logistic Regression

Let us say we are trying to predict the malignancy of a tumor based on the size of the tumor (Fig. 11.3).

If a linear regression is used in this case, the line of best fit, which would be a straight line, will not cover all the data points. On the other hand, if a curved line

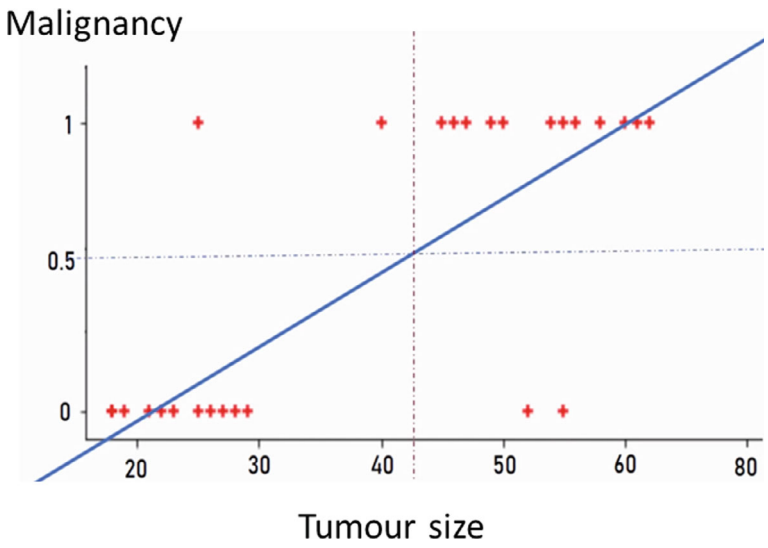


Fig. 11.3 Need for logistic regression 1

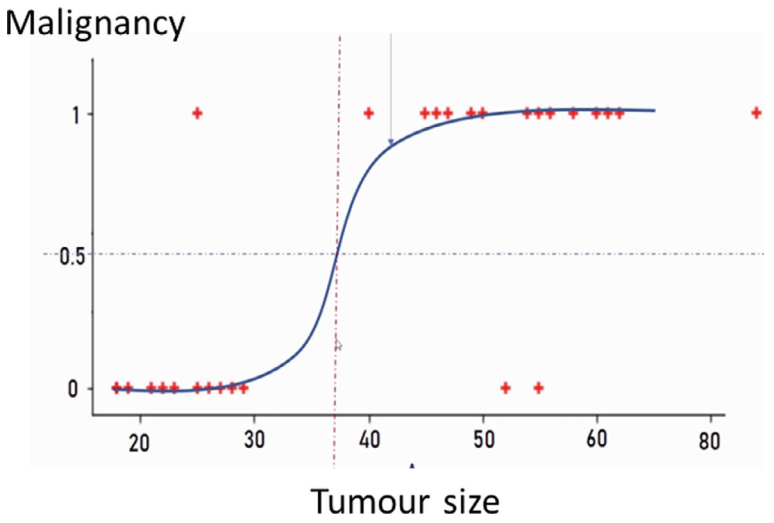


Fig. 11.4 Need for logistic regression 2

is used to cover the points, the curved line would cover most of the data points as shown in Fig. 11.4.

11.3.3 Linear Versus Logistic Regression Equation

In the linear model, the equation of the line of best fit is $y = mx + c$. In the logistic model, it is a sigmoid function to find probability p.

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve (Fig. 11.5).

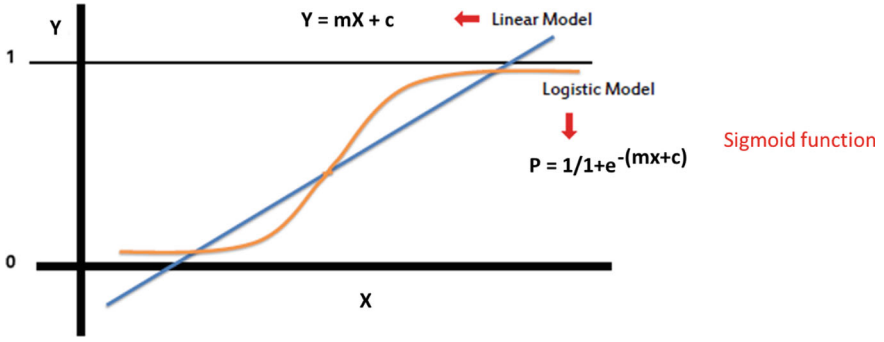


Fig. 11.5 Logistic regression

11.3.4 Cost Function and Gradient Descent

The primary goal of any machine learning algorithm is to minimize the cost function or reduce the inaccuracy. The cost function aggregates the errors (or losses) from all individual predictions to understand the algorithm's performance across the entire training dataset.

To understand the concept of cost function and gradient descent, let us consider the example of customer churn prediction based on three factors, namely average revenue (x_1), customer status as old or new (x_2), and past complaint history (x_3). The goal is to develop a model with high accuracy in predicting customer churn, incorporating these factors and their respective weights ($w_1, w_2, w_{3)}$ into the model equation. Weights can be viewed as the relative importance of each factor in predicting customer churn. The accuracy of the model is directly linked to the correct assessment of these weights (Fig. 11.6).

The determination of these weights is achieved through a process of initially selecting weights at random and evaluating the resulting accuracy. This iterative adjustment of weights continues until satisfactory accuracy is achieved. This optimization exercise forms the core of every machine learning algorithm, including logistic regression.

The method employed for adjusting weights based on accuracy assessments is known as gradient descent. This optimization algorithm aims to minimize the cost function by iteratively moving toward the steepest descent direction. The size of the adjustments made to the weights—whether marginal or significant—depends on the current level of accuracy and the distance from the desired outcome. This decision-making process is governed by the principles of gradient descent.

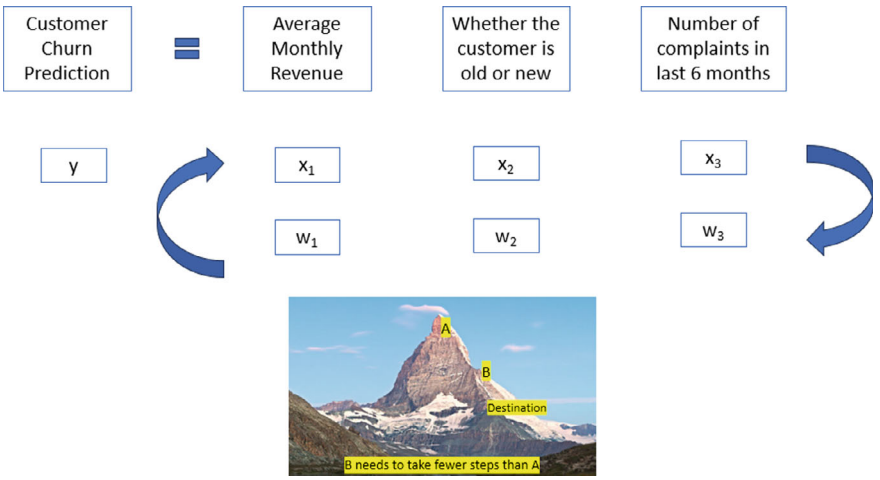


Fig. 11.6 Gradient descent with an example

In the context of gradient descent, the concept of slope (dy/dx) is crucial. For instance, the slope at point A is less steep than that at point B, indicating that reaching the minimum from the blue point (A) requires much smaller steps compared to the green point (B) (Fig. 11.7).

Gradient descent utilizes derivatives to determine the direction of weight adjustments, either increasing or decreasing, to optimize the objective function. By computing the derivative of a function, the direction for minimizing the function becomes clear.

The optimization process, thus, involves the following steps.

Initialization: Gradient descent begins with an initial set of parameters (coefficients), often chosen randomly or set to zero.

Compute Gradient: It calculates the gradient of the binary cross-entropy loss function with respect to each parameter. The gradient is a vector that contains the partial derivatives of the loss function with respect to each coefficient β, and it points in the direction of the steepest ascent on the loss surface.

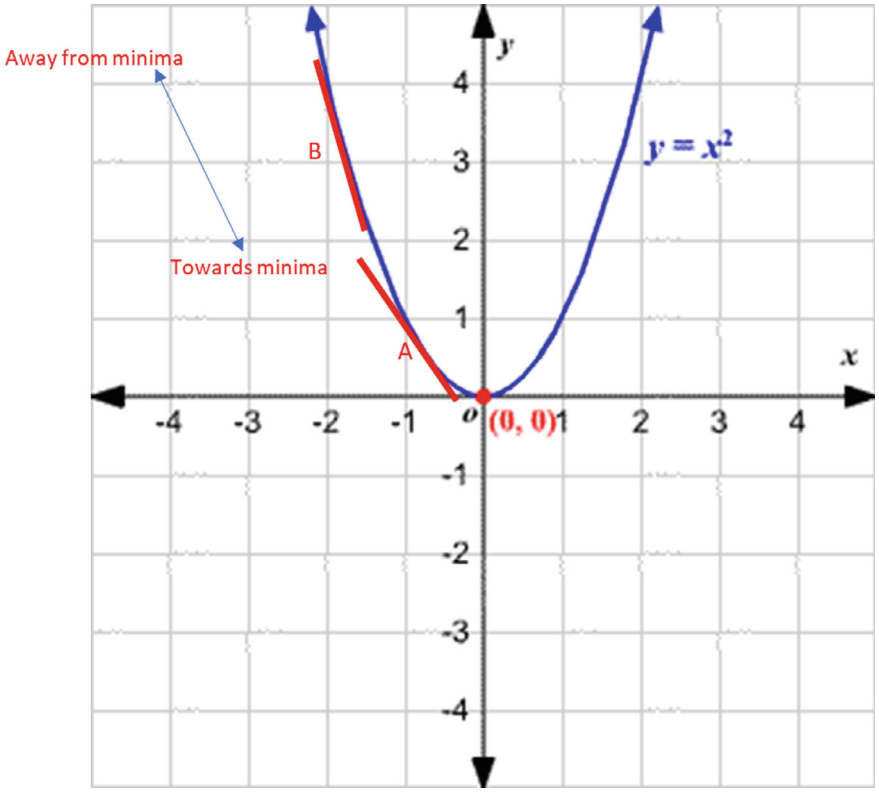


Fig. 11.7 Gradient descent graph

Update Parameters: To minimize the loss, we need to move against the gradient. Thus, each parameter is updated in the direction opposite to the gradient. This is done by subtracting the gradient of the loss function multiplied by the learning rate α from the current value of the parameter:

$$\beta := \beta - \alpha\{\partial J(\beta)/\partial\beta\}.$$

Let us understand the formula and notations.

The notation $\beta :=$ is used in algorithms, especially in the context of iterative methods such as gradient descent, to denote the assignment or update of the variable β. When you see $\beta := \beta - \alpha\{\partial J(\beta)/\partial\beta\}$, it means that the current value of β is being updated and replaced with a new value which is the result of the expression on the right-hand side.

In the context of gradient descent for logistic regression:

β represents the vector of coefficients or parameters of the model.
α is the learning rate, which controls the size of the step taken in the direction opposite to the gradient to reach a minimum.
$\partial J(\beta)/\partial\beta$ is the partial derivative of the cost function $J(\beta)$ with respect to the parameter β also known as the gradient.

So, when you perform the update $\beta := \beta - \alpha \cdot \{\, \partial J(\beta)/\partial\beta \,\}$, you're taking the current estimate of β, computing the gradient of the cost function at that point, and then adjusting β by taking a step proportional to the negative of the gradient, scaled by the learning rate. This process is intended to move the parameter β toward the values that minimize the cost function $J(\beta)$.

Iterate: This process is repeated iteratively until the change in the loss function is below a certain pre-defined threshold or for a pre-defined number of iterations. The idea is that with each iteration, the parameters are getting closer to the optimal values that minimize the loss function.
Convergence: Eventually, gradient descent will reach a point where the cost function stops decreasing significantly with each iteration. At this point, the algorithm has converged, and the parameters at this state are considered to be the optimal parameters for the model.

By using gradient descent in logistic regression, we aim to find the parameter values that give us the smallest possible binary cross-entropy loss, indicating that the predicted probabilities are as close as possible to the actual class labels. This optimization process is fundamental to the training of many machine learning algorithms, including logistic regression.

11.3.5 Evolution of Logistic Regression

The origins of logistic regression can be traced back to the early-nineteenth century, when the logistic function was first proposed by Pierre François Verhulst to model population growth. However, it was not until the mid-twentieth century that logistic regression was first used for statistical inference.

The first paper to apply logistic regression to a real-world problem was published in 1958 by Joseph Berkson. Berkson used logistic regression to predict the probability of death from heart disease based on a set of risk factors, such as age, blood pressure, and cholesterol levels.

In the 1960s and 1970s, logistic regression became increasingly popular among statisticians and researchers. This was due in part to the development of new computational methods that made it possible to fit logistic regression models to large datasets.

In the 1980s and 1990s, logistic regression continued to evolve, with new methods and techniques being developed. One of the most significant advances was the development of regularized logistic regression, which can help to prevent overfitting.

In recent years, logistic regression has been extended to new domains, such as computer vision and natural language processing. This has been made possible by the development of new machine learning algorithms, such as deep learning.

11.3.6 Programmatic Implementation of Logistic Regression

This example demonstrates the application of logistic regression to predict the categorization of house prices into three levels: high, medium, or low. While the dataset employed here is identical to the one used in multiple linear regression, there is a significant difference in the nature of the dependent variable. Instead of a numeric value representing the house price, a categorical variable is used.

Although the independent variables and the underlying scenario remain consistent with the previous model, the shift from a numeric to a categorical dependent variable fundamentally alters the model's output. Consequently, this change in the dependent variable necessitates a different approach in pattern recognition. The logistic regression model will identify distinct patterns, correlating the same set of independent variables to the categorical outcomes of high, medium, or low house prices, as opposed to the continuous numerical values used in linear regression. This variation underscores the adaptability of regression models to different types of data and predictive requirements.

- Import necessary modules

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,
classification_report, roc_auc_score, roc_curve, accuracy_score
from sklearn.preprocessing import OneHotEncoder, LabelEncoder,
StandardScaler
import matplotlib.pyplot as plt
```

- Load the data

```
df = pd.read_csv('/content/housing_classification.csv')
```

- Preprocess the data (eliminating NULL values)

```
df.fillna(df.median(), inplace=True)
df.fillna(df.mode().iloc[0], inplace=True)
```

- Encoding the non-numeric data

```
label_encoder = LabelEncoder()
one_hot_encoder = OneHotEncoder()
for column in df.select_dtypes(include=['object']).columns:
    if column != 'Pricing':  # Exclude the target variable
        if df[column].nunique() == 2:
            df[column] = label_encoder.fit_transform(df[column])
        else:
            df = df.join(pd.get_dummies(df[column],
prefix=column))
            df.drop(column, axis=1, inplace=True)
```

- Encoding the target variable

```
df['Pricing'] = label_encoder.fit_transform(df['Pricing'])
```

- Separate features (independent variables) and target variable (dependent variable)

```
X = df.drop('Pricing', axis=1)
y = df['Pricing']
```

- Split the dataset into test and train datasets

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
```

- Feature scaling

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- Fitting the logistic regression to the training set. Limited-memory Broyden–Fletcher–Goldfarb–Shanno' algorithm (LBGFS) is an advanced form of gradient descent. Unlike basic gradient descent, LBFGS uses an approximation to the second-order partial derivatives to guide its search for the minimum. This allows for more informed and potentially faster convergence.

```
model = LogisticRegression(multi_class='multinomial',
solver='lbfgs')
model.fit(X_train, y_train)
```

- Predicting the test results

```
y_pred = model.predict(X_test)
```

- Making the confusion matrix

```
cm = confusion_matrix(y_test, y_pred)
```

- Feature scaling

```
scaler = StandardScaler()
```

- Plotting the confusion matrix with labels

```
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_labels, yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

- Classification report

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

- Computing the ROC AUC

```
y_prob = model.predict_proba(X_test)
roc_auc = roc_auc_score(y_test, y_prob, multi_class='ovr')
print("\nROC AUC Score:", roc_auc)
```

- Plotting the ROC curve

```
fpr = {}
tpr = {}
thresh ={}
n_class = 3

for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_prob[:,i],
pos_label=i)

plt.plot(fpr[0], tpr[0], linestyle='--', color='orange',
label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--', color='green',
label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--', color='blue',
label='Class 2 vs Rest')
plt.title('Multiclass ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

- Calculating and printing the accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)
```

- Split the data into test data and train data (Figs. 11.8, 11.9 and 11.10)
  ROC AUC Score: 0.8803873656596428.

Interpreting the Results

There are three main components to interpret: the confusion matrix, the classification report, and the multiclass ROC curve.

Confusion Matrix:

- It shows how many predictions were correct and how many were errors.
- The rows represent the actual classes, and the columns represent the predicted classes.
- For the 'High' class, there were three correct predictions and four incorrect predictions.
- For the 'Low' class, all three predictions were correct.
- For the 'Mid' class, there were 87 correct predictions and $5 + 2 = 7$ incorrect predictions.
- The model seems to perform best at identifying 'Mid' prices.

Classification Report:

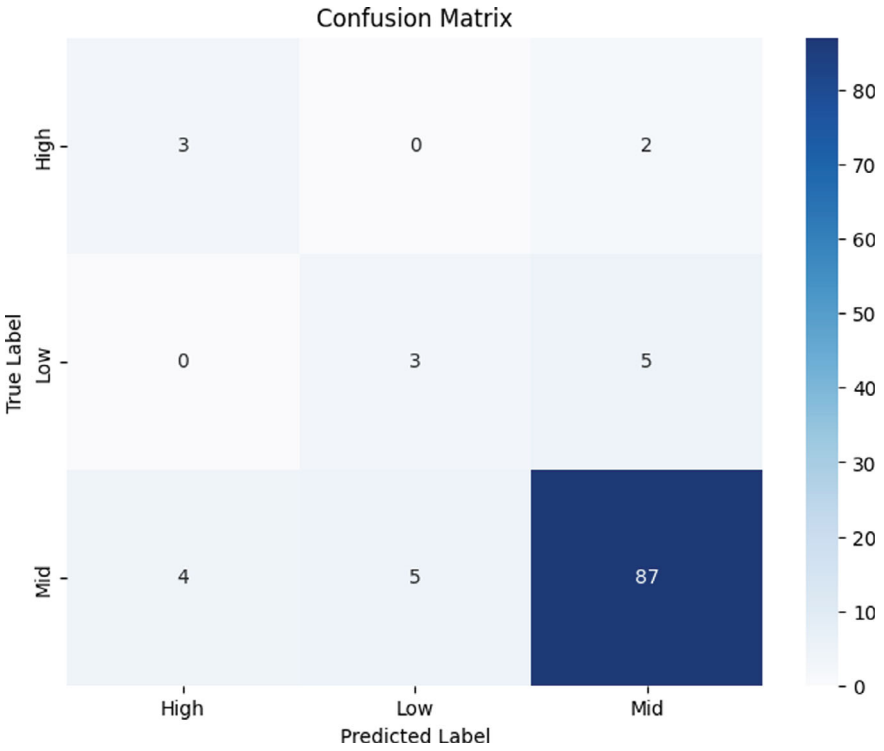


Fig. 11.8 Logistic regression program output showing confusion matrix

Classification Report:

	precision	recall	f1-score	support
0	0.43	0.60	0.50	5
1	0.38	0.38	0.38	8
2	0.93	0.91	0.92	96
accuracy			0.85	109
macro avg	0.58	0.63	0.60	109
weighted avg	0.86	0.85	0.86	109

Fig. 11.9 Logistic regression program output showing classification report

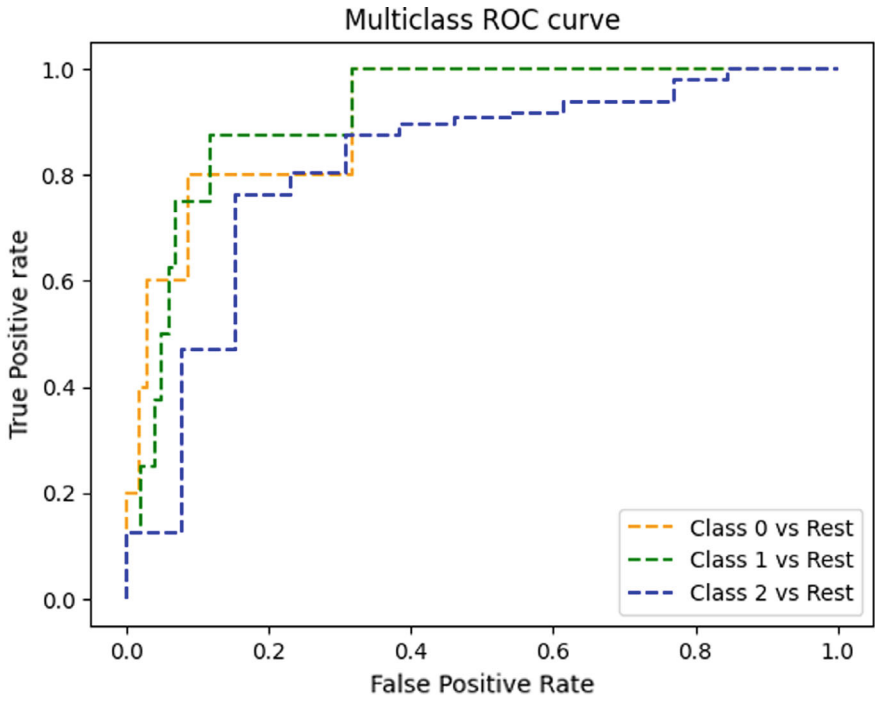


Fig. 11.10 Logistic regression program output showing ROC curve

- Precision is the ratio of true positives to the sum of true and false positives. It's how many selected items are relevant.
- Recall (or sensitivity) is the ratio of true positives to the sum of true positives and false negatives. It's how many relevant items are selected.
- $F1$-score is the harmonic mean of precision and recall. A higher score is better.
- Support is the number of actual occurrences of each class.
- The report shows that the model is best at predicting 'Mid' priced houses (with precision, recall, and $F1$-score of 0.93, 0.91, and 0.92, respectively).
- 'High' and 'Low' priced predictions are less accurate and reliable based on their lower scores.
- Overall accuracy is 0.85, meaning that the model correctly predicts the price category 85% of the time.

Multiclass ROC Curve:

- The receiver operating characteristic (ROC) curve is a graph showing the performance of a classification model at all classification thresholds.
- This curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

- The dotted lines represent the ROC curve for each class compared against all others (One-vs-Rest). The orange dashed line represents Class 0, the green dashed line corresponds to Class 1, and the blue dashed line represents Class 2.
- Ideally, the curve should push toward the top-left corner, indicating a higher true positive rate and a lower false positive rate. The closer the curves are to the left and top borders, the better the model's ability to distinguish between classes.
- The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. Hence, Class 2 is being predicted with high accuracy, whereas Classes 0 and 1 have less accurate predictions. This aligns with the precision and recall metrics from the classification report, where Class 2 had significantly higher values.

Summary

This logistic regression model is quite good at predicting medium-priced houses but not as good at predicting high or low-priced houses. The ROC curve indicates the model's effectiveness at distinguishing between classes, with a preference for the 'Mid' class.

Overall, while the model seems to have a decent accuracy, there is room for improvement, especially in correctly identifying the 'High' and 'Low' price categories.

11.3.7 Use Cases of Logistic Regression

- Estimating the likelihood of a patient being afflicted with a particular illness.
- Determining the probability of a customer completing a purchase.
- Assessing the probability of a loan applicant failing to meet loan obligations.
- Predicting the likelihood of a student successfully passing an examination.
- Estimating the probability of a previously convicted individual committing another offense.
- Predicting the likelihood of a user clicking on an advertisement.
- Estimating the probability of a product being returned by a customer.
- Assessing the probability of a customer discontinuing their association with a service or product.

11.3.8 Advantages of Logistic Regression

Logistic regression is easier to implement, interpret, and very efficient to train. It provides good accuracy for many simple data sets and it performs well when the dataset is linearly separable.

Due to its simple probabilistic interpretation, the training time of logistic regression algorithm comes out to be far less than most complex algorithms.

11.3.9 Limitations of Logistic Regression

Logistic regression assumes that the independent variables are linearly related to the outcome variable. This assumption may not be valid in all cases. Logistic regression does not perform well with non-linearly separable data.

Logistic regression is not as effective in handling high-dimensional data or complex relationships between the independent variables and the outcome variable.

Logistic regression can be overfitting, especially when there is a small amount of training data.

Despite its limitations, logistic regression remains to be one of the most widely used classification algorithms for addressing a wide range of real-world problems.

11.3.10 De-mystifying Gradient Descent

Gradient descent is a method used to find the best solution or answer in machine learning. It's like trying to find the lowest point in a valley by taking steps downhill.

This concept is important in machine learning because many problems involve finding the best settings or parameters for a model. Gradient descent helps by changing the parameters little by little to improve the model. In the example we saw earlier, the algorithm adjusted the weights or relative importance of the features.

Optimization is the process of finding the best or most efficient solution. Gradient descent is one way to do optimization. It's connected to the optimization process because it's a tool that helps to tweak and improve the model step by step.

The math behind gradient descent involves calculus, specifically derivatives. A derivative tells you how much a function changes when you change the inputs a little bit. In gradient descent, you use derivatives to find out which way is 'downhill,' and that tells you which way to adjust your parameters.

The gradient descent process:

- Start with Random Parameters: You begin with some random settings for your model.
- Calculate the Gradient: The gradient is the slope of the function you're trying to optimize, like the slope of a hill. You calculate it using derivatives.
- Update the Parameters: You change the parameters a bit in the direction that goes 'downhill,' based on the gradient.
- Repeat: You keep doing this over and over, each time moving the parameters a little bit toward the best solution.

Every time you update the parameters, you're hoping to get a model that's a little bit better. You stop when you can't find a better solution anymore or when the improvements are too small to matter.

11.3.11 Understanding Derivative

A derivative in mathematics is a way to show how a function's output changes as its input changes. It's like looking at a car's speedometer; the speedometer shows how quickly the car's position is changing over time.

For example, if you're driving a car and you press the accelerator, the speedometer's needle moves to show your speed is increasing. The speedometer is giving you a 'derivative' of your position with respect to time.

In machine learning, the 'weights' are adjusted. These weights determine how much each feature (like square footage or number of bedrooms) affects the prediction (like the price of a house).

Imagine you're holding a fishing line with a weight at the end, and you feel the pull of the weight in your hand. If you add more weight, the pull gets stronger. The derivative tells you exactly how much stronger the pull is when you add that extra weight.

So, when we adjust weights in logistic regression, we're trying to change them so that our model's predictions match the real outcomes. The derivative helps us understand if we should increase or decrease a weight to improve the model. If our prediction is too high, we need to 'let out some line' or decrease the weight. If it's too low, we need to 'reel in' or increase the weight.

In the context of gradient descent, the derivative points out the direction we should take to adjust weights to minimize the difference between our model's predictions and the actual outcomes. This process is repeated iteratively to make the predictions as accurate as possible.

11.4 Concept of Gradient Descent

You know how to come down to the base if you are currently at Point x, highlighted in the picture below. You know that you have to take Route 1. However, if you are a computer robot, how will you know how to come down. You can either take Route 1 or 2, although Route 2 will take more time than Route 1 (Fig. 11.11).

So the gradient descent algorithm helps us to make those decisions. As part of this process, the algorithm determines whether you are closer to your goal or not. If you are closer to your destination, you need to take only smaller steps. If you are away from your goal, you would need bigger steps. This concept of bigger versus smaller steps corresponds to the relative weights assigned to the factors.

How will the computer know whether it has to take smaller or bigger steps? That is, how will the computer know whether you are closer to the goal or not? The computer uses the concept of dy/dx, which is nothing but slope or gradient (Fig. 11.12).

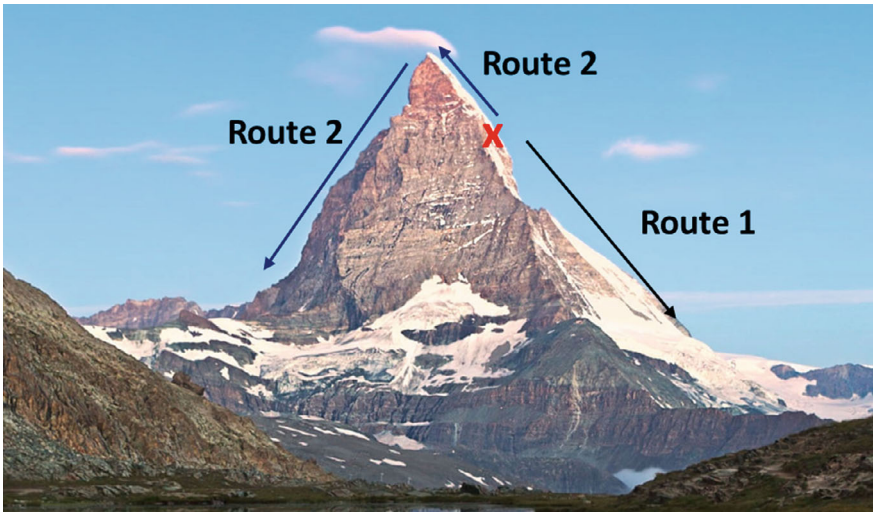


Fig. 11.11 Gradient descent 1

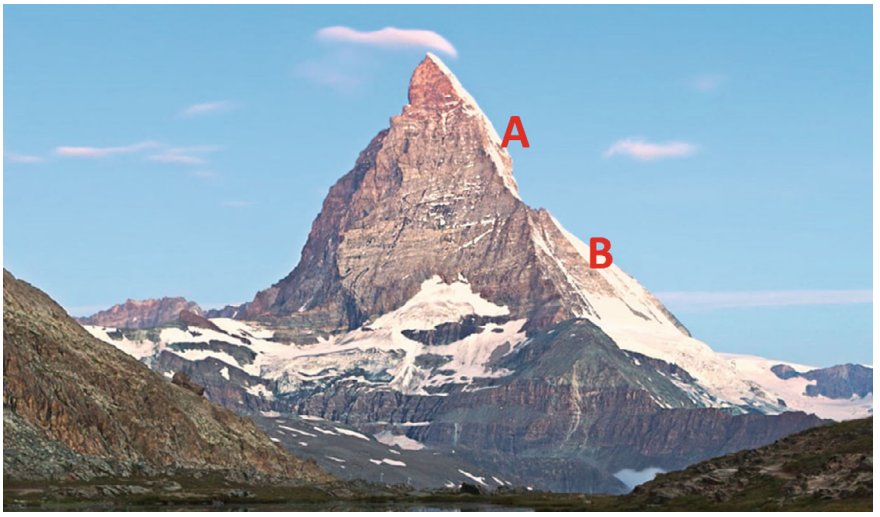


Fig. 11.12 Gradient descent 2

The slope at Point B is less steep than that at Point A, and this means that the computer robot will need to take fewer steps to reach the base from B than A. Machine learning uses derivatives in optimization exercises. Optimization algorithms like gradient descent use derivatives to decide whether to increase or decrease the relative weights. If we can compute the derivative of a function, we will know which direction to proceed. In our algorithms, we compute the negative of the gradient (or

gradient descent). This is nothing but the opposite of the gradient or negative value of dy/dx.

Reference

Kleinbaum DG (2009) Logistic regression: mathematical and statistical foundations. Springer

Chapter 12
Decision Trees, Bagging, and Boosting

Decision Trees: Branching Out to Find Answers Hidden in Data's Leaves

Abstract This chapter introduces readers to decision trees, powerful classification algorithms known for their intuitive, tree-like structure and ease of interpretation. The discussion begins with the historical evolution of decision trees, highlighting their rise from simple decision-support tools to sophisticated predictive models widely used in industries such as healthcare, finance, and marketing. Various practical use cases illustrate decision trees' versatility in solving complex problems by breaking decisions into logical branches. The concepts of bagging and boosting are explained, emphasizing how ensemble methods significantly enhance predictive accuracy. Readers will gain hands-on experience with detailed, programmatic implementation of decision trees, bagging, and boosting, along with clear interpretations of the generated results. A step-by-step demonstration of manually constructing decision trees further deepens readers' intuitive understanding of the underlying decision-making logic. By the end, learners will appreciate how decision trees and their advanced ensembles provide interpretable yet robust solutions for complex real-world machine learning tasks.

Keywords Decision tree · Bagging · Boosting · Random forest · XGBoost · AdaBoost · Root node · Leaf node · Decision node · Entropy · GINI · Information gain · Predictive maintenance

We often make a lot of complex decisions in our lives that require careful consideration. While algorithms can provide us with solutions, they often lack transparency in explaining how they arrived at those solutions. But decision trees are different. They serve as a visual roadmap, shedding light on the decision-making process.

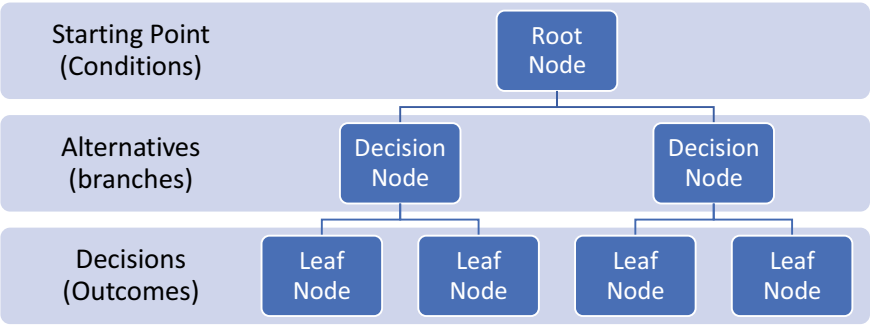
sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_12

12.1 Introduction to Decision Trees

A decision tree is a tree with branches. Each point where a branch splits, called a node, represents a feature or question. The branches are the possible answers or rules, and the ends of the branches, called leaves, give the final outcome of the algorithm. It is a versatile supervised machine learning algorithm, which is used for both classification and regression problems.

12.1.1 Elements and Important Terminologies of a Decision Tree



- Root Node: The starting point of the decision tree, representing the entire dataset and dividing it into more homogeneous sets.
- Leaf Node: The ultimate output node, where the tree cannot divide further.
- Splitting: The act of segmenting the decision or root node into subnodes based on specified conditions.
- Branch/Subtree: A tree that results from splitting a larger tree.
- Pruning: The process of eliminating undesired branches from the tree.
- Parent/Child Node: The initial node of the tree is the parent node, while all other nodes are considered child nodes.

12.1.2 Construction of a Decision Tree

Decision trees work by asking questions about your data. With each question, they split the data into groups that are different from each other. This process of splitting over and over is called recursive partitioning.

To select which feature to split the data on, the decision tree algorithm uses an Attribute Selection Measure (ASM). ASMs measure how well a feature can distinguish between different classes. The feature with the highest ASM score is used to split the data.

The process stops when the data is pure, meaning that all the data points in a group have the same class label.

The resulting decision tree can then be used to predict the class label of new data points by asking the same questions in the same order.

12.1.3 Attribute Selection Methods

1. Entropy: Entropy is a measure of the randomness or uncertainty in a dataset. In classification tasks, it measures how random the distribution of class labels is (Fig. 12.1).

The entropy of a dataset with K classes can be calculated as follows:

$$E(s) = \sum_{i=1}^{c} -p_i \log_2 p_i,$$

where s is the dataset, i is a particular class from K classes, and $p(i)$ is the proportion of data points in class i.

Entropy is 0 when the dataset is completely homogeneous, meaning that all data points belong to the same class. It is at its maximum value when the dataset is equally divided between all classes.

Entropy is used to evaluate the quality of a split in a decision tree. The goal is to select the attribute that minimizes the entropy of the resulting subsets, meaning that the subsets are more homogeneous with respect to the class labels.

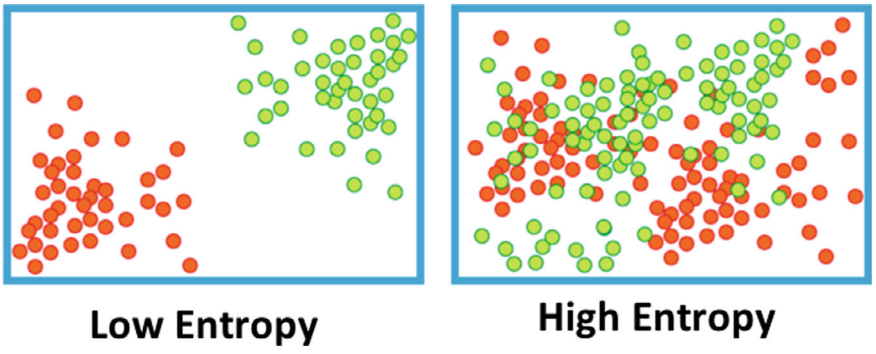


Fig. 12.1 Entropy

The attribute with the highest information gain is chosen as the splitting criterion (information gain is the reduction in entropy after splitting on that attribute). The process is repeated recursively to build the decision tree.

2. Gini Impurity: Gini impurity is a measure of how mixed a group of data points are. It ranges from 0 to 1, with 0 being the purest group (all data points belong to the same class) and 1 being the most mixed group (data points are evenly distributed among the classes). The term "GINI" doesn't have a commonly recognized acronym expansion. Instead, "Gini" typically refers to concepts named after the Italian statistician Corrado Gini.

$$1 - \sum \left(p(i)^2\right),$$

where p_i is the proportion of elements in the set that belongs to the ith category.

Decision tree algorithms use Gini impurity to evaluate the quality of a split. The goal is to select the split that minimizes the Gini impurity of the resulting subsets.

3. Information Gain: Information gain is a measure of how well a feature can split a dataset into more homogeneous subsets with respect to the class labels or target variable. It is calculated by subtracting the entropy of the dataset after the split from the entropy of the dataset before the split.

The higher the information gain of an attribute, the better it is at predicting the target variable. This is because a higher information gain indicates that the attribute can be used to split the dataset into more homogeneous subsets, which means that the decision tree will be able to make more accurate predictions.

Information gain is used in both classification and regression decision trees. In classification, entropy is used as a measure of impurity, while in regression, variance is used as a measure of impurity. However, the information gain calculation remains the same in both cases.

$$\text{Information Gain} = \text{Entropy (S)} - \left[\text{(Weighted Average)} * \text{Entropy (Each Feature)}\right].$$

12.1.4 Evolution of Decision Trees

One of the earliest decision tree algorithms was ID3, which was developed by J. Ross Quinlan in the 1980s. ID3 was a simple but effective algorithm that used information gain to select the best feature to split the dataset on at each node of the tree (Langley 2017).

Another important development in the field of decision trees was the introduction of Classification and Regression Trees (CARTs) by Leo Breiman in the 1980s. CART was a more sophisticated algorithm than ID3 that was able to handle both classification and regression tasks. CART also introduced several new features, such as

pruning and cross-validation, which helped to improve the performance of decision trees.

In the 1990s, Quinlan developed a new decision tree algorithm called C4.5. C4.5 was an improvement over ID3 in several ways, including its ability to handle missing values and continuous attributes. C4.5 is still one of the most popular decision tree algorithms in use today (Papagelis and Kalles 2006).

In recent years, there has been a renewed interest in decision trees, due to their ability to work well with large datasets and their interpretability. Decision trees are now being used in a wide variety of applications, including fraud detection, medical diagnosis, and natural language processing.

Working of Decision Trees

1. Commence the tree with the root node labeled as S, encompassing the complete dataset.
2. Employ the Attribute Selection Measure (ASM) to identify the most suitable attribute within the dataset.
3. Segment S into subsets, each containing the potential values for the chosen attribute.
4. Construct a decision tree node featuring the selected attribute.
5. Iteratively create new decision trees using the subsets generated in step 3. This process continues until a point is reached where further classification is not possible, marking that final node as a leaf node in the classification and regression tree algorithm.

12.1.5 Use Cases of Decision Trees

Classification: Decision trees are applicable for categorizing data points into different groups. They can be used to determine whether customers are likely to churn or not or to distinguish between images containing cats and dogs.

Regression: Decision trees are effective in predicting continuous values. They can be utilized to estimate house prices or forecast the number of customers visiting a store on a given day.

Recommendation Systems: Decision trees can be harnessed to construct recommendation systems, suggesting products to customers based on their previous purchases.

Fraud Detection: Decision trees are valuable tools for identifying fraudulent transactions, such as pinpointing potentially deceitful credit card activities.

Medical Diagnosis: Decision trees can play a crucial role in aiding medical diagnoses, helping doctors in the process of diagnosing patients with specific diseases.

Advantages

- Easy to Understand: Decision trees can be visualized, making them simple to interpret and explain, even for those without a background in data analysis.
- Handles Both Types of Data: They can manage both numerical and categorical data.
- Non-linear Relationships: Decision trees can capture non-linear relationships between features and the target variable without requiring data transformation.
- No Need for Data Preprocessing: They often require less data preprocessing, such as normalization or scaling.
- Identifies Important Features: The structure of the decision tree can help in identifying the most significant variables that affect the outcome.

Disadvantages

- Overfitting: Decision trees can easily overfit, especially with complex trees. They may capture noise in the data rather than the actual signal.
- Instability: Small variations in the data can result in a completely different tree being generated. This problem is mitigated by using ensemble methods like random forests.
- Biased Trees: If some classes dominate, decision trees can create biased trees. Balancing the dataset is necessary to avoid this.
- Difficulty with Complex Relationships: While they handle non-linear data well, they might struggle with complex relationships that other models can capture more effectively.
- Not Ideal for Continuous Variables: Decision trees are less effective at predicting outcomes for continuous variables compared to categorical ones.

Despite their shortcomings, decision trees remain a favored algorithm in machine learning due to their simplicity in understanding and interpretation. Moreover, they lay the groundwork for more sophisticated algorithms.

12.2 Bagging and Boosting: Enhancements to Decision Tree

12.2.1 Bagging

"Bagging" is an acronym for "Bootstrap Aggregating". The most famous bagging algorithm is random forest. The process of bagging involves combining multiple decision trees to arrive at the final outcome. Instead of utilizing all features to structure the trees, bagging process constructs multiple trees by randomly selecting the features. Bagging thus achieves the twin objectives of model stability and accuracy by lowering variance and minimizing the risk of overfitting (Li et al. 2023; Zhang and Rokach 2022).

Steps involved in bagging:

1. The original dataset is split into multiple subsets through a method known as bootstrapping, which involves random sampling with replacement.
2. A base model is then trained on each subset.
3. Since these subsets are independent, the models can be trained simultaneously, enhancing efficiency.
4. The final prediction is obtained by aggregating the predictions from all individual models.

12.2.2 Boosting

Boosting is a machine learning ensemble technique that aims to create a strong classifier from a number of weak classifiers. This method works by sequentially training weak classifiers, each focusing on the mistakes of the previous ones, and combining them to improve the overall model's accuracy and performance. Examples of boosting algorithms include adaptive boosting (AdaBoost) and gradient boosting.

Steps involved in boosting:

1. Initialization: The algorithm starts by assigning equal weights to all instances in the training dataset. These weights indicate the importance of each instance in the dataset.
2. Iterative Learning:
   (a) A weak classifier is trained on the data. A weak classifier is simply a model that performs slightly better than random guessing.
   (b) After training, the classifier's performance is evaluated based on the weighted instances. The errors are calculated by considering the weights of the instances.
   (c) Increase the weights of the instances that are misclassified and decrease the weights for those that are correctly classified. This step ensures that the next classifier focuses more on the instances that were previously misclassified.
3. Adding the Classifier: The weak classifier is then added to the ensemble of classifiers, with a weight assigned to it. This weight is typically based on the classifier's accuracy, with more accurate classifiers receiving higher weights.
4. Repeat: Steps 2 and 3 are repeated for a specified number of iterations or until a desired level of accuracy is achieved. With each iteration, the model becomes better at classifying the training data.
5. Final Model: Once all the iterations are completed, the boosting algorithm combines the weak classifiers into a single strong classifier. When making predictions, the final model considers the weighted vote of all the weak classifiers.

What Is an Ensemble?

You may keep hearing this term ensemble. What is an ensemble. An ensemble is a machine learning technique that combines multiple models to improve the overall

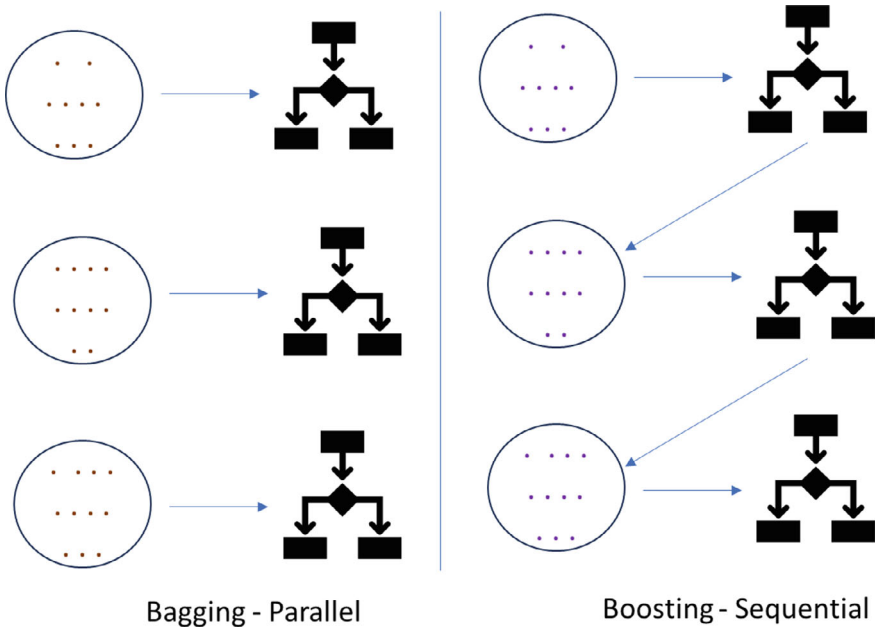


Fig. 12.2 Bagging and boosting

performance, accuracy, and robustness of predictions or classifications. The fundamental idea behind ensemble methods is that a group of weak models can come together to form a strong model. This approach leverages the diversity among the models to reduce errors that might arise from using a single model. Ensemble methods work by either averaging the predictions (in the case of regression) or by voting (in the case of classification) from multiple models. Ensemble methods are widely used due to their effectiveness in improving predictions and handling various data complexities that single models might not adequately address. Bagging and boosting are examples of ensemble (Fig. 12.2).

Table 12.1 summarized the differences between the two methods.

12.3 Demonstration of (Manual) Implementation of Decision Trees

Determination of root node is the crucial step in construction of decision tree, and the same shall be explained below using GINI impurity.

The scenario under consideration is employee churn or employee attrition (EA), which is influenced by factors such as job satisfaction, salary level, and years in the current role (Fig. 12.3).

Table 12.1 Bagging and boosting

Feature	Bagging	Boosting
Goal	To reduce variance and prevent overfitting	To reduce bias and variance, and to improve predictions
Method	Combines the results of multiple models built with different subsets of the same training set	Builds models sequentially, each one trying to correct the errors of the previous model
Weighting of classifiers	Each classifier gets equal weight when making the final decision	Classifiers are weighted according to their accuracy, with more accurate ones having more influence
Training data sampling	Uses bootstrapping (random sampling with replacement) to create subsets for training each model	Each new model is trained on a modified version of the dataset, emphasizing the previously misclassified instances
Classifier independence	Classifiers are trained independently in parallel	Classifiers are trained sequentially and are dependent on the performance of previous ones
Error type focus	Generally better for reducing overfitting in models that have high variance	Focuses on reducing errors by giving more attention to instances that were previously misclassified
Examples	Random forest is a classic example of bagging	AdaBoost and gradient boosting are examples of boosting
Complexity	Simpler to parallelize and often faster to train because models are independent	More computationally intensive and usually slower to train due to the sequential nature
Output combination	Typically uses simple voting (for classification) or averaging (for regression)	Weights the output of each classifier before combining them
Flexibility	Less flexible since each model is usually trained to its maximum extent	More flexible, often includes mechanisms to prevent overfitting

- Part of the dataset (Table 12.2).
- Summarize the data for each of the nodes as shown in Figs. 12.4, 12.5 and 12.6.
- GINI is computed for good and not good scenarios as shown in Fig. 12.7.
- GINI is computed for each of the nodes and summarized as shown in Fig. 12.8.

12.4 Implementation of Decision Trees Programmatically

- Scenario: Predictive maintenance in a manufacturing industry.
- Dependent Variable: Failure.

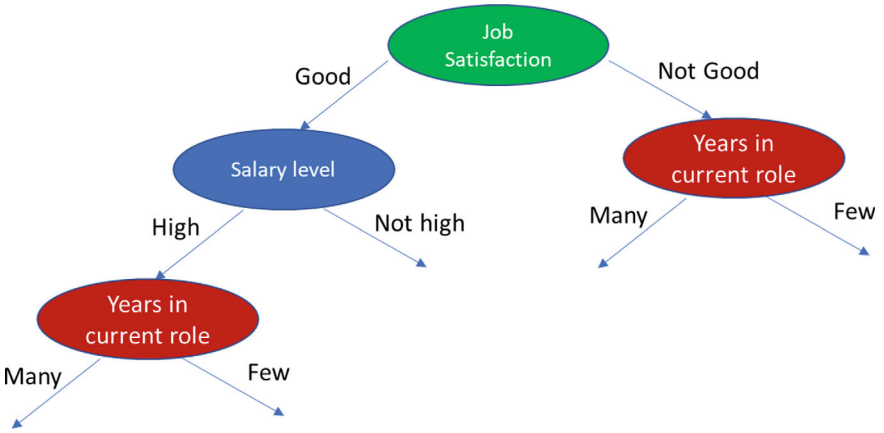


Fig. 12.3 Decision tree

Table 12.2 Part of the dataset used to demonstrate decision tree

Job satisfaction	Years in current role	Salary level	Will employee leave?
Not good	Few	Not high	No
Good	Many	High	Yes
Good	Many	Not high	No
Good	Few	Not high	Yes

Fig. 12.4 Decision tree for job satisfaction)

- Independent Variables: Vibration, temperature, hours of operation.
- Part of the dataset (Table 12.3).

- Import necessary modules

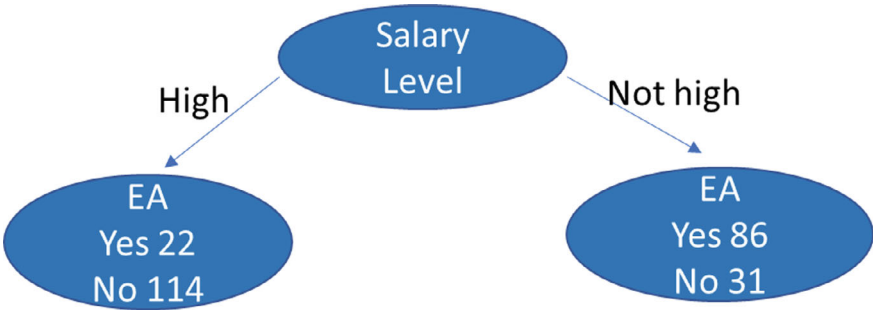


Fig. 12.5 Decision tree for salary level)

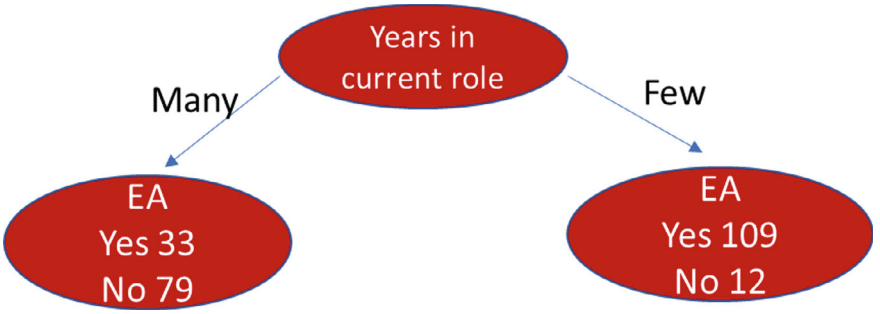


Fig. 12.6 Decision tree for tenure)

Fig. 12.7 Decision tree for job satisfaction

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import xgboost as xgb
import matplotlib.pyplot as plt
```

- Load the data

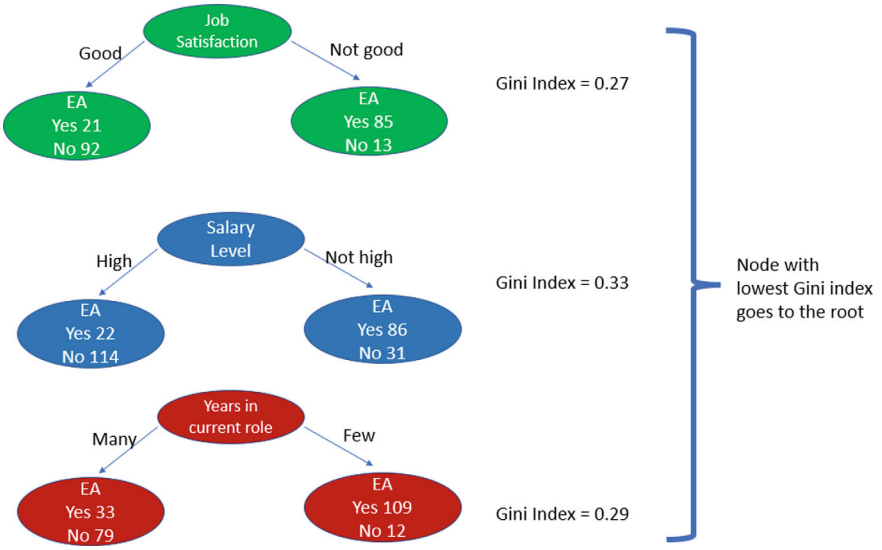


Fig. 12.8 Decision tree with Gini

Table 12.3 Part of the dataset for predictive maintenance

Vibration	Temperature	Hours_of_Operation	Failure
1.8	105.6	4005.1	0
0.4	108.9	4109.2	0
1.0	95.8	1032.5	1
2.2	101.0	5191.0	0
1.9	102.3	968.0	0
− 1.0	102.0	8897.6	0
1.0	105.4	618.4	0
− 0.2	81.8	4496.9	1

```
data = pd.read_csv('/content/predictive_maintenance_dataset.csv')
```

- Define dependent and independent variables

```
X = data.drop('Failure', axis=1)
y = data['Failure']
```

- Split the data into test and train datasets

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

- Initialize the models

```
dt = DecisionTreeClassifier(random_state=42)
rf = RandomForestClassifier(random_state=42)
xg_clf = xgb.XGBClassifier(use_label_encoder=False,
eval_metric='logloss')
```

- Train the models

```
dt.fit(X_train, y_train)
rf.fit(X_train, y_train)
xg_clf.fit(X_train, y_train)
```

- Make predictions

```
dt_predictions = dt.predict(X_test)
rf_predictions = rf.predict(X_test)
xg_predictions = xg_clf.predict(X_test)
```

- Calculate the accuracies

```
dt_accuracy = accuracy_score(y_test, dt_predictions)
rf_accuracy = accuracy_score(y_test, rf_predictions)
xg_accuracy = accuracy_score(y_test, xg_predictions)
```

- Print the accuracies

```
print('Decision Tree Accuracy:', dt_accuracy)
print('Random Forest Accuracy:', rf_accuracy)
print('XGBoost Accuracy:', xg_accuracy)
```

- Plot the decision tree

```
plt.figure(figsize=(20,10))
plot_tree(dt, filled=True, feature_names=X.columns,
class_names=['No Failure', 'Failure'])
plt.title('Decision Tree')
plt.show()
```

- Plot one tree from the random forest

```
plt.figure(figsize=(20,10))
plot_tree(rf.estimators_[0], filled=True,
feature_names=X.columns, class_names=['No Failure', 'Failure'])
plt.title('One Tree from the Random Forest')
plt.show()
```

- Plot one tree from XGBoost

```
plt.figure(figsize=(20,10))
xgb.plot_tree(xg_clf, num_trees=0)
plt.title('One Tree from XGBoost')
plt.show()
```

- Result of the program (Figs. 12.9, 12.10 and 12.11).
  Decision Tree Accuracy: 0.86.
  Random Forest Accuracy: 0.895.
  XGBoost Accuracy: 0.895.

Since the images are not viewable, let us restrict the depth of the trees in plotting the tree structure for decision tree and random forest. Please note that this is done merely to visualize the trees and that restricting the depth of the tree will reduce the accuracy. In XGBoost, the model's complexity comes from being an ensemble of decision trees. Unlike a single decision tree, XGBoost builds many trees in a

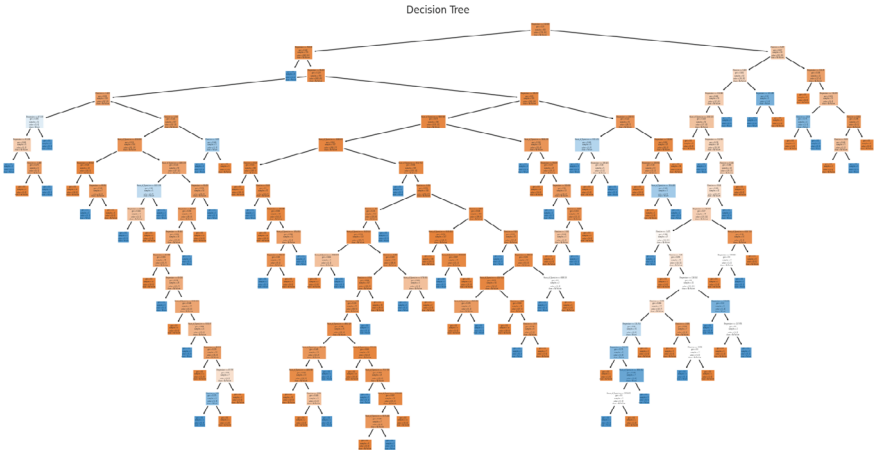


Fig. 12.9 Program output for decision tree

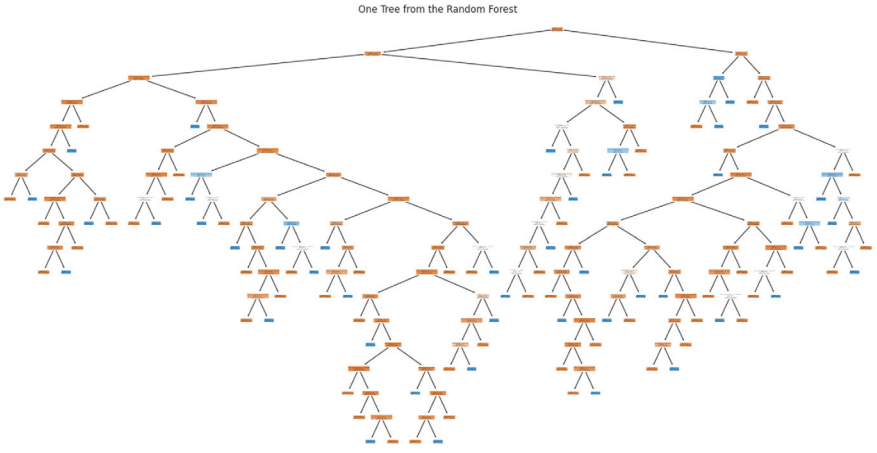


Fig. 12.10 Program output for random forest

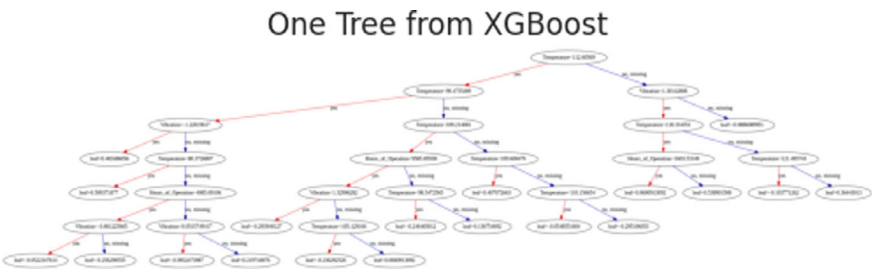


Fig. 12.11 Program output for XGBoost

sequential manner where each tree learns from the mistakes of the previous ones (a process known as boosting). This sequential building cannot be easily represented as a single restricted tree.

12.5 Implementation of Decision Trees with Restriction in the Depth of the Tree

- Scenario: Predictive maintenance in a manufacturing industry
- Import necessary modules

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

- Load the data

```
data = pd.read_csv('/content/predictive_maintenance_dataset.csv')
```

- Define dependent and independent variables

```
X = data.drop('Failure', axis=1)
y = data['Failure']
```

- Split the data into test and train datasets

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

- Initialize the models with a max. depth parameter

```
dt = DecisionTreeClassifier(random_state=42, max_depth=3)
rf = RandomForestClassifier(random_state=42, max_depth=3)
```

- Train the models

```
dt.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

- Plot the decision tree

```
print("Decision Tree:")
print(export_text(dt, feature_names=list(X.columns)))
```

- Plot one tree from the random forest

```
print("\nRandom Forest Tree 0:")
print(export_text(rf.estimators_[0],
feature names=list(X.columns)))
plt.title('One Tree from the Random Forest')
```

- Result of the program (Fig. 12.12).

This decision tree uses temperature and vibration features to determine two classes, representing a binary outcome like "functioning properly" versus "failure". The most important features are temperature followed by vibration, as these are the features used to make decisions at each node of the tree. The tree suggests that temperature is the most critical factor in predicting class 0, except in a specific range of temperatures where vibration also plays a key role.

The root node (the first decision point) tests the feature "Temperature". It checks if the temperature is less than or equal to 112.63. Based on this test, it splits the data into two branches.

The left branch of the root node handles cases where the temperature is less than or equal to 112.63. This branch further splits into two at the second level:

- If the temperature is less than or equal to 78.82, the tree predicts class 1.
- If the temperature is greater than 78.82, it further checks if it's less than or equal to 90.43. If so, it predicts class 0; if not, it also predicts class 0. This suggests that for all temperatures less than or equal to 112.63 but greater than 78.82, the model predicts class 0 regardless.

The right branch of the root node addresses cases where the temperature is greater than 112.63. It then considers "Vibration":

- If the vibration is less than or equal to 0.50, it further checks if it's less than or equal to 0.41. If so, it predicts class 0; if not, it predicts class 1.
- If the vibration is greater than 0.50, it then checks the temperature again. If the temperature is less than or equal to 115.39, it predicts class 0; if the temperature is greater than 115.39, it predicts class 0 (Fig. 12.13).

The root node is based on "Vibration", and it is the most significant feature for the first set of splits in this particular tree. Each path from the root to a leaf represents a rule derived from the tree.

```
Decision Tree:
|--- Temperature <= 112.63
|   |--- Temperature <= 78.82
|   |   |--- class: 1
|   |--- Temperature >  78.82
|   |   |--- Temperature <= 90.43
|   |   |   |--- class: 0
|   |   |--- Temperature >  90.43
|   |   |   |--- class: 0
|--- Temperature >  112.63
|   |--- Vibration <= 0.50
|   |   |--- Vibration <= 0.41
|   |   |   |--- class: 0
|   |   |--- Vibration >  0.41
|   |   |   |--- class: 1
|   |--- Vibration >  0.50
|   |   |--- Temperature <= 115.39
|   |   |   |--- class: 0
|   |   |--- Temperature >  115.39
|   |   |   |--- class: 0
```

Fig. 12.12 Program output showing a part of decision tree

Root Node: The first decision point at the top of the tree is based on "Vibration". The tree checks if the vibration is less than or equal to 0.30.

Branching:

- If the "Vibration" is less than or equal to 0.30, the next decision is based on "Temperature".
  - If the "Temperature" is less than or equal to 113.94, the tree looks at "Hours_ of_Operation". If it's less than or equal to 3498.53, the predicted class is 0. If it's more than 3498.53, the predicted class is also 0.
  - If the "Temperature" is greater than 113.94, the tree again checks "Hours_ of_Operation". If it's less than or equal to 6715.51, the predicted class is 1, otherwise, it's 0.

```
Random Forest Tree 0:
|--- Vibration <= 0.30
|   |--- Temperature <= 113.94
|   |   |--- Hours_of_Operation <= 3498.53
|   |   |   |--- class: 0.0
|   |   |--- Hours_of_Operation >  3498.53
|   |   |   |--- class: 0.0
|   |--- Temperature >  113.94
|   |   |--- Hours_of_Operation <= 6715.51
|   |   |   |--- class: 1.0
|   |   |--- Hours_of_Operation >  6715.51
|   |   |   |--- class: 0.0
|--- Vibration >  0.30
|   |--- Hours_of_Operation <= 1417.93
|   |   |--- Vibration <= 0.47
|   |   |   |--- class: 0.0
|   |   |--- Vibration >  0.47
|   |   |   |--- class: 0.0
|   |--- Hours_of_Operation >  1417.93
|   |   |--- Vibration <= 0.31
|   |   |   |--- class: 1.0
|   |   |--- Vibration >  0.31
|   |   |   |--- class: 0.0
```

Fig. 12.13 Program output showing a part of random forest

- If the "Vibration" is greater than 0.30, the tree splits decisions based on "Hours_of_Operation".
  - If "Hours_of_Operation" is less than or equal to 1417.93, the tree considers "Vibration" again. If "Vibration" is less than or equal to 0.47, the predicted class is 0. If "Vibration" is greater than 0.47, the predicted class is still 0.
  - If "Hours_of_Operation" is greater than 1417.93, the tree looks at "Vibration" one more time. If "Vibration" is less than or equal to 0.31, the predicted class is 1. If "Vibration" is greater than 0.31, the predicted class is 0.

Leaf Nodes: The ends of the branches (the final decisions) are the leaf nodes. These are the outcomes of the decision tree, indicating the predicted class based on the conditions above.

References

Langley P (2017) Advances in decision tree learning. Mach Learn 106(7):963–973
Li Y, Wang Z, Gao W, Zhang J, Rokach L (2023) Bagging and boosting for unsupervised anomaly detection. IEEE Trans Inf Forensics Secur 18(7):1902–1916
Papagelis L, Kalles D (2006) Evolutionary algorithms for decision tree induction: a survey. Appl Soft Comput 6(2):201–210
Zhang J, Rokach L (2022) Bagging and boosting: a review of ensemble learning techniques. IEEE Trans Pattern Anal Mach Intell 45(1):1–26

Chapter 13
Naïve Bayes

Naive Bayes: Simplifying Complexity with a Touch of Probability

Abstract In this chapter, readers explore the foundational concept of Bayes' theorem and its crucial role in classification through conditional probability. Beginning with a clear introduction to decisionmaking under uncertainty, the chapter thoroughly explains how Bayes' theorem leverages prior knowledge to compute conditional probabilities. Practical examples illustrate manual calculations step-by-step, enabling readers to understand precisely how conditional probabilities are derived and utilized in predictions. To reinforce this understanding, readers will learn how these manual computations are mirrored in programming constructs, demystifying processes typically hidden within libraries. Programmatic implementation of Bayesian classifiers is demonstrated, along with detailed interpretations of their outputs. Through hands-on examples, readers learn to intuitively interpret results from Bayesian methods and appreciate their practical utility. By uncovering the internal logic libraries employ for such calculations, readers develop deeper confidence and clarity in their modeling skills. Ultimately, the chapter emphasizes why mastering these concepts enhances readers' abilities to build accurate and explainable predictive models.

Keywords Naïve Bayes · Conditional probability · Bayes' theorem · Heatmap

The Naive Bayes algorithm is a straightforward yet powerful tool used in the field of machine learning for classification tasks—determining which category an object belongs to based on its features. Its simplicity and effectiveness make it popular, especially in text classification tasks such as spam detection or sentiment analysis. Bayes' theorem is named after the English statistician and philosopher Thomas Bayes (1701–1761).

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_13

13.1 What Is Naive Bayes?

Naive Bayes is based on Bayes' theorem, a fundamental theory in probability. Bayes' theorem helps in understanding the likelihood of an event based on prior knowledge of conditions that might be related to the event (https://plato.stanford.edu/entries/bayes-theorem/#1). The 'naive' part comes from the algorithm's assumption that all features it analyzes are independent of each other, which isn't always true in real-world data but simplifies the calculation.

13.1.1 Implementing Naive Bayes

Prepare the Data: Gather and clean your data. In text classification, this might involve converting text into a format the algorithm can work with, like a bag-of-words model where text is represented as a collection of individual words.

Calculate Probabilities: For each category, calculate the probability of each feature appearing. For example, in spam detection, this might be the probability of certain words appearing in spam emails.

Apply Bayes' Theorem: Use these probabilities in Bayes' theorem to calculate the probability of a data point belonging to each category. The category with the highest probability is typically chosen as the classification result.

Model Evaluation: Test the model's accuracy using a separate test dataset to ensure that it generalizes well to new data.

13.1.2 Understanding Bayes' Theorem

Bayes' theorem calculates the probability of an event based on prior knowledge of conditions that might be related to the event. In mathematical terms, it relates the conditional and marginal probabilities of random events.

Imagine a rare disease that affects 1 in every 1000 people. There's a test for this disease, but the test isn't perfect. If you have the disease, there's a 99% chance that the test will be positive (this is called the true positive rate). However, if you don't have the disease, there's still 5% chance the test will be positive (this is known as the false positive rate).

Now, let's say you take the test and it comes back positive. What are the chances you actually have the disease?

Here's where Bayes' Theorem comes in. It helps us update our initial beliefs (the 1 in 1000 chance) based on the new evidence (the positive test result).

The formula for Bayes' theorem is

$$P(A|B) = P(B|A) * P(A)/P(B),$$

where

- $P(A|B)$ is the probability of having the disease (A) given a positive test result (B).
- $P(B|A)$ is the probability of a positive test result given the disease, which is 99%.
- $P(A)$ is the initial probability of having the disease, which is 0.1% (1 in 1000).
- $P(B)$ is the overall probability of a positive test result. This includes both true positives (people who have the disease) and false positives (people who don't).

$P(B)$ is the probability of a positive test across all possibilities—those who have the disease and those who don't. So, it includes the true positives (1 in 1000 people with 99% detection) and the false positives (999 in 1000 people with a 5% false positive rate).

By plugging in the numbers, we can calculate the actual probability of having the disease given a positive test result. The result might be surprising—it's much lower than just taking the 99% accuracy at face value, mainly because the disease is so rare.

- Initial probabilities

```
P_A = 1/1000  # Probability of having the disease
P_not_A = 999/1000  # Probability of not having the disease
```

- Probabilities of test outcomes

```
P_B_given_A = 0.99  # Probability of a positive test given having
the disease (True Positive Rate)
P_B_given_not_A = 0.05  # Probability of a positive test given not
having the disease (False Positive Rate)
```

- Overall probability of a positive test result

```
P_B = (P_B_given_A * P_A) + (P_B_given_not_A * P_not_A)
```

- Applying Bayes' theorem to calculate $P(A|B)$: Probability of having the disease given a positive test result

```
P_A_given_B = (P_B_given_A * P_A) / P_B
P_A_given_B
```

- Result of the program
  0.019434628975265017

This example illustrates how Bayes' theorem allows us to adjust probabilities based on additional evidence, providing a more accurate picture than initial assumptions might suggest. It's an essential tool in fields like medical diagnostics, where understanding the true likelihood of conditions is critical.

Despite Its Usefulness, Naive Bayes Has Limitations

Assumption of Independence: The assumption that all features are independent can lead to inaccuracies since this isn't always the case in real-life data.

Data Scarcity: If a categorical variable has a category in the test dataset, which was not observed in the training dataset, the model will assign it a zero probability and be unable to make a prediction. This is often known as "Zero Frequency".

Simplicity: While its simplicity is a strength, it also means Naive Bayes can be outperformed by more complex models, especially when the independence assumption is violated.

Applications of Naive Bayes

Naive Bayes is particularly favored in text classification due to its efficiency and effectiveness with large datasets. Common applications include:

- Email spam filtering: Classifying emails as spam or not spam.
- Sentiment analysis: Determining whether sentiment expressed in text is positive, negative, or neutral.
- Document categorization: Classifying news articles into different topics.

13.1.3 Programmatic Application of Naïve Bayes Algorithm—Classification of News Item

- We will use the 20 Newsgroups dataset from Scikit-learn. This dataset is a collection of approximately 20,000 newsgroup documents, partitioned across 20 different newsgroups (Table 13.1).

- Import the libraries and the dataset

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

- Fetching the dataset

```
newsgroups = fetch_20newsgroups(subset='all')
data = newsgroups.data
targets = newsgroups.target
```

- Displaying the first document and the category

```
print("Sample document:\n", data[0])
print("Target category:", newsgroups.target_names[targets[0]])
```

Table 13.1 Details of the dataset used in Naïve Bayes

Serial number	Name of the group	Details of the news group
1	comp.graphics	Computer graphics
2	comp.os.ms-windows.misc	Microsoft Windows operating systems
3	comp.sys.ibm.pc.hardware	IBM personal computers and their hardware
4	comp.sys.mac.hardware	Apple Macintosh computers and their hardware
5	comp.windows.x	The X Window System (a windowing system for bitmap displays)
6	rec.autos	Automobiles
7	rec.motorcycles	Motorcycles
8	rec.sport.baseball	Baseball
9	rec.sport.hockey	Hockey
10	sci.crypt	Cryptography
11	sci.electronics	Electronics
12	sci.med	Medicine
13	sci.space	Space exploration
14	misc.forsale	Selling and buying various items
15	talk.politics.misc	Miscellaneous politics
16	talk.politics.guns	Firearms and related legislations
17	talk.politics.mideast	Politics of the Middle East
18	talk.religion.misc	Miscellaneous religion
19	alt.atheism	Atheism
20	soc.religion.christian	Christianity

- Creating a pipeline that combines a CountVectorizer with a naïve Bayes Classifier

  CountVectorizer is a feature extraction tool in natural language processing, typically used within the Python library scikit-learn. It converts a collection of text documents into a matrix of token counts.

```
model = make_pipeline(CountVectorizer(), MultinomialNB())
```

- Splitting the dataset into test and train dataset

```
X_train, X_test, y_train, y_test = train_test_split(data, targets,
test_size=0.25, random_state=0)
```

- Training the model

```
model.fit(X_train, y_train)
```

- Predicting the test set

```
predicted = model.predict(X_test)
```

- Evaluating the model

```
cm = confusion_matrix(y_test, predicted)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt='d',
xticklabels=newsgroups.target_names,
yticklabels=newsgroups.target_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print(classification_report(y_test, predicted,
target_names=newsgroups.target_names))
```

- Output of the program (Fig. 13.1)

13.2 Interpretation of the Output

The heatmap is a confusion matrix for a Naive Bayes classifier applied to the 20 newsgroups dataset. This heatmap provides a visual representation of where the Naive Bayes classifier is making correct predictions and where it's making errors. It helps us to better understand the classifier's performance across different categories.

Diagonal Values: The numbers along the diagonal from the top-left to the bottom-right represent correct classifications by the Naive Bayes model. For example, the model correctly predicted 214 instances of comp.graphics and 228 instances of sci.crypt. Higher numbers on the diagonal show where the model performed well.

Off-Diagonal Values: The numbers not on the diagonal are instances that were misclassified. For instance, 12 instances that were actually sci.electronics were predicted as comp.graphics by the model.

Row Analysis: Each row represents the true category of the data. For example, in the alt.atheism row, 187 instances were correctly identified as alt.atheism, but some instances were misclassified into other categories like comp.graphics and soc.religion.christian.

Column Analysis: Each column represents the predicted category by the model. If you look at the soc.religion.christian column, the model predicted several instances as soc.religion.christian that were actually from different true categories.

Color Intensity: The color intensity indicates the frequency of predictions. Darker colors show a higher number of instances, and lighter colors show fewer instances. Darker colors on the diagonal are good, as they show more correct predictions.

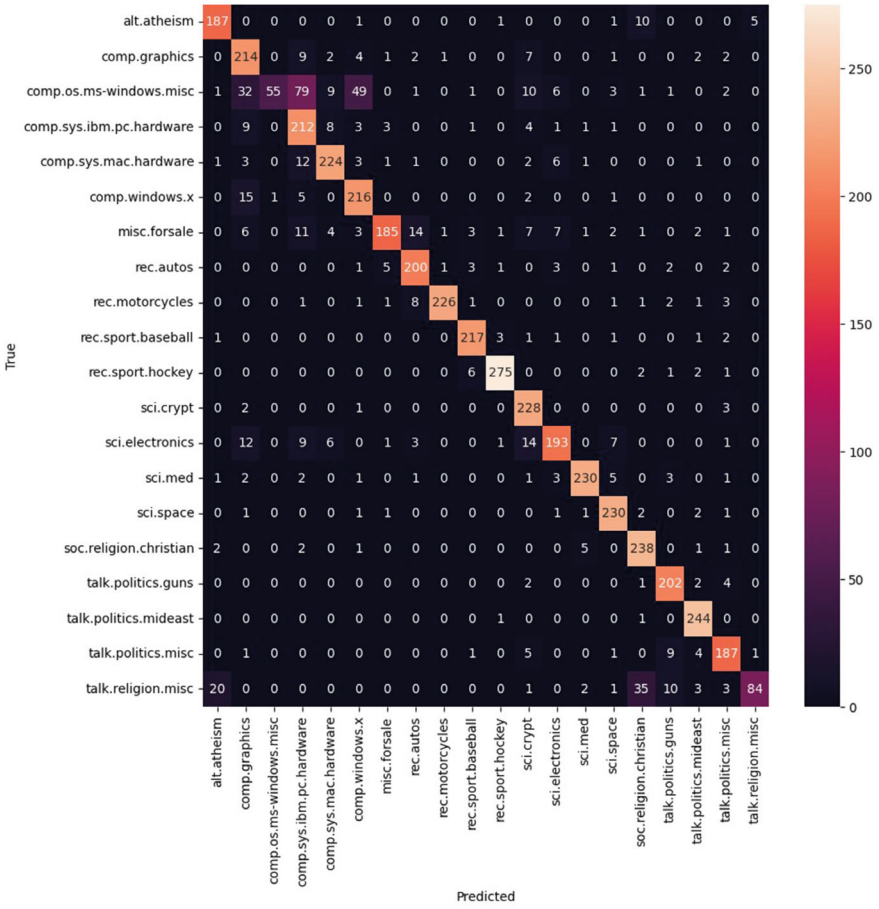


Fig. 13.1 Program output showing heatmap

Misclassification Patterns: By observing off-diagonal cells with darker colors, you can identify which categories are commonly confused by the model. For instance, talk.religion.misc seems to be often confused with alt.atheism and soc.religion.christian (Fig. 13.2).

Now, let us interpret the classification report.

Precision: Precision is the ratio of true positive predictions to the total positive predictions made for a category. For example, for alt.atheism, a precision of 0.88 means that 88% of the items predicted as alt.atheism were actually alt.atheism.

Recall: Recall, or sensitivity, is the ratio of true positive predictions to the actual number of positives. So, for alt.atheism, a recall of 0.91 means that the model correctly identified 91% of all actual alt.atheism instances.

Predicted

	precision	recall	f1-score	support
alt.atheism	0.88	0.91	0.89	205
comp.graphics	0.72	0.87	0.79	245
comp.os.ms-windows.misc	0.98	0.22	0.36	250
comp.sys.ibm.pc.hardware	0.62	0.87	0.72	243
comp.sys.mac.hardware	0.89	0.88	0.88	255
comp.windows.x	0.76	0.90	0.82	240
misc.forsale	0.93	0.74	0.83	249
rec.autos	0.87	0.91	0.89	219
rec.motorcycles	0.99	0.92	0.95	246
rec.sport.baseball	0.93	0.96	0.94	227
rec.sport.hockey	0.97	0.96	0.96	287
sci.crypt	0.80	0.97	0.88	234
sci.electronics	0.87	0.78	0.82	247
sci.med	0.95	0.92	0.94	250
sci.space	0.90	0.96	0.93	240
soc.religion.christian	0.82	0.95	0.88	250
talk.politics.guns	0.88	0.96	0.92	211
talk.politics.mideast	0.92	0.99	0.95	246
talk.politics.misc	0.87	0.89	0.88	209
talk.religion.misc	0.93	0.53	0.67	159
accuracy			0.86	4712
macro avg	0.87	0.86	0.85	4712
weighted avg	0.87	0.86	0.85	4712

Fig. 13.2 Program output showing different metrics)

$F1$-Score: The $F1$-score is the harmonic mean of precision and recall, giving a balance between the two. It's a single metric that summarizes the accuracy of the classifier for each category. An $F1$-score of 0.89 for alt.atheism is quite high, suggesting a good balance of precision and recall.

Support: Support indicates how many actual instances of each class are present in the specified dataset. For alt.atheism, there are 205 instances.

The confusion matrix provides a detailed breakdown of predictions for each class, showing exactly where the model is making mistakes. It shows the true positives along with false positives and false negatives for each category. The classification report summarizes this information into key metrics, giving you a more aggregated view of the model's performance.

Differences Between the Confusion Matrix and Classification Report:

The confusion matrix shows the number of predictions for each actual and predicted class pairing, while the classification report summarizes these into performance metrics for each class.

The confusion matrix is great for identifying which specific classes are being confused, while the classification report gives an overall statistical measure of performance.

Both are important—the confusion matrix for detailed analysis and understanding of model performance on a per-class basis and the classification report for an overall statistical summary.

Reference

Bayes' Theorem. https://plato.stanford.edu/entries/bayes-theorem/#1

Chapter 14
Support Vector Machine (SVM)

Support Vector Machines: Drawing the Line in Complex Data Dimensions

Abstract This topic will cover the concept, methodology, and applications of Support Vector Machines in machine learning.

Keywords Support vector machines (SVMs) · Linear SVMs · Non-linear SVMs

A support vector machine (SVM) is a type of machine learning model used for classification tasks (Cortes and Vapnik 1995). It finds the best line or boundary that separates data into classes.

The idea behind SVM is to create a line or a hyperplane that separates the data into groups. It was developed in the 1960s but gained popularity in the 1990s. The key feature of SVM is that it focuses on the data points that are hardest to tell apart, which are called support vectors.

Support vector machines (SVMs) can be categorized into different types or variants based on various factors, including the problem they address, the type of data they handle, and their specific characteristics.

Linear SVMs

Linear SVMs are used to classify data that can be separated into two classes by a straight line. They are often used for classification tasks with text data, such as spam filtering and sentiment analysis.

Non-linear SVMs

Non-linear SVMs employ the kernel trick to transform the data into a higher-dimensional space where it becomes linearly separable. Common kernels for non-linear SVM include polynomial, radial basis function (RBF), sigmoid, and custom kernels.

Advantages

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_14

- Effective in High Dimensions: SVM works well when your data has lots of features.
- Memory Efficient: It uses only a few key data points to create the dividing line, so it doesn't need much memory.
- Versatility: You can use different functions, called kernels, to draw boundaries, not just straight lines.

Limitations

- Not Good for Large Datasets: When you have a lot of data, SVM can get slow.
- Needs Careful Preprocessing: Data needs to be scaled and prepared well, or SVM might not work properly.
- Hard to Interpret: The way SVM makes decisions is not always easy to understand, especially with complex boundaries.

Use Cases

- Face Detection: SVM can classify parts of an image as a face or not a face.
- Text and Hypertext Categorization: SVMs help classify emails as spam or not spam.
- Classification of Images: They can recognize handwritten characters used by the postal service.
- SVMs are a powerful tool when you have clear margins of separation and can invest time in tuning them for your specific needs.

Key Principles

- Margin: The distance between the hyperplane and the nearest data points from each class is maximized.
- Support Vector: These are data points that are closest to the decision boundary and directly influence the position of the hyperplane.
- Kernel Trick: SVM can handle non-linear data by mapping it into a higher-dimensional space using kernel functions.
- C Parameter: It controls the trade-off between maximizing the margin and minimizing classification errors.

How Does SVM Work

Imagine you're playing a game where you have two types of marbles: red and blue. Your goal is to draw a line on the ground that separates the red marbles from the blue ones. In the simplest case, you can draw a straight line that keeps all the red marbles on one side and all the blue marbles on the other (Fig. 14.1).

Now, this line represents what an SVM does in a two-dimensional space (like the ground you're drawing the line on). The marbles are your data points, and the colors are your two classes.

If these marbles are spread out in such a way that a straight line can't perfectly separate them, an SVM looks for the "best" line that can do the job. "Best" means that the line should have the widest possible margin or distance between the red and blue

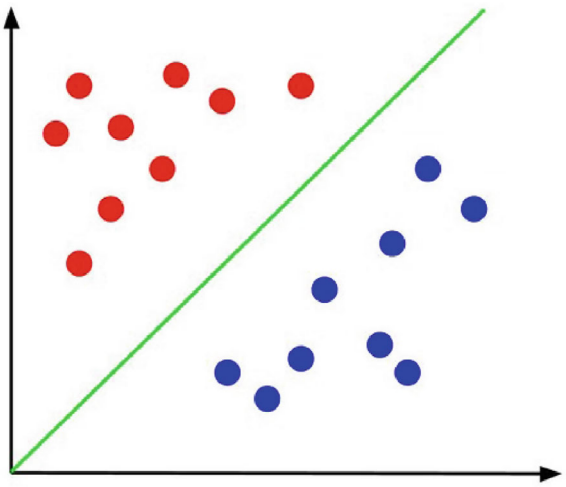


Fig. 14.1 Linear Support Vector Machine

marbles. The closest marbles to the line, the ones that define this margin, are called the support vectors because they "support" or define the position and orientation of the line.

Now, let's make it a bit more complex. Imagine some of the blue marbles are inside the red area and vice versa. In the real world, data is often like this—messy and overlapping. SVM can handle this by using something called a kernel trick to add another dimension. Imagine you can lift some of the marbles off the ground, creating a hill. Now, instead of a line, you can draw a curve (like a circle around the top of the hill) to separate the marbles. By adding this new dimension, SVM can handle more complex data separation even when it's not linearly separable on the flat ground.

Programmatic Application of SVM Algorithm—Diabetes Prediction

We will use the diabetes dataset that is available as a pre-installed dataset in scikit-learn library. It contains ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements for 442 diabetes patients.

- Import the libraries and the dataset

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
```

- Load the diabetes dataset

```
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

- Classify patients as having or not having diabetes based on the median value

```
y_class = np.where(y > np.median(y), 1, 0)
```

- Split the dataset into test and train dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y_class,
test_size=0.2, random_state=42)
```

- Standardize the features

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- Create the SVM model

```
svm_model = SVC(kernel='rbf', random_state=42)
```

- Train the SVM model

```
svm_model.fit(X_train, y_train)
```

- Predict using SVM model

```
y_pred = svm_model.predict(X_test)
```

- Calculate accuracy and display the classification report

```
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

- Output of the program

  See Fig. 14.2.

- Evaluating the output

```
Accuracy: 0.7415730337078652
              precision    recall  f1-score   support

           0       0.80      0.71      0.75        49
           1       0.69      0.78      0.73        40

    accuracy                           0.74        89
   macro avg       0.74      0.74      0.74        89
weighted avg       0.75      0.74      0.74        89
```

Fig. 14.2 Output of the program

The model performs reasonably well at predicting whether patients have diabetes or not. It's slightly better at figuring out who doesn't have diabetes than who does, because the scores that measure how accurate it is (precision and $F1$-score) are a little higher for the people without diabetes.

- Understanding the different measures

Accuracy: The model is about 74.15% accurate overall. This means that, on average, the model correctly predicts whether a patient has diabetes 74.15% of the time.

Precision (for class 0): The precision for predicting class 0 (likely no diabetes) is 0.80, meaning that when the model predicts a patient does not have diabetes, it is correct 80% of the time.

Precision (for class 1): The precision for predicting class 1 (likely diabetes) is 0.69, indicating that the model's prediction of diabetes is correct about 69% of the time.

Recall (for class 0): The recall for class 0 is 0.71, which means that of all the actual cases without diabetes, the model correctly identifies 71% of them.

Recall (for class 1): The recall for class 1 is 0.78, meaning that the model correctly identifies 78% of the actual cases with diabetes.

$F1$-Score: The $F1$-score is a balance between precision and recall. For class 0, the $F1$-score is 0.75, and for class 1, it's 0.73, indicating a relatively balanced precision and recall for both classes.

Support: This is the number of actual occurrences in each class. There are 49 instances of class 0 and 40 instances of class 1 in the test set.

Macro Avg: This is the average precision, recall, and $F1$-score for both classes without taking the support into account. It's also about 0.74, which means the model performs fairly uniformly across both classes.

Weighted Avg: This is similar to the macro average, but it takes the support into account. It's also about 0.74, indicating that the model's performance is consistent when adjusted for the number of instances in each class.

Reference

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20:273–297

Chapter 15
Unsupervised Machine Learning

Unsupervised Learning: Discovering Hidden Patterns Without a Guiding Hand

Abstract In this chapter, readers transition from supervised to unsupervised learning, opening up new possibilities in discovering hidden structures within unlabeled data. Building upon the supervised techniques previously explored, this chapter clarifies how unsupervised methods differ fundamentally by focusing on inherent patterns rather than labeled outcomes. Readers are introduced to the powerful concepts behind clustering and association rules, enabling algorithms to group data points and identify meaningful relationships without predefined categories. The chapter extensively covers clustering techniques, particularly emphasizing detailed explanations and programmatic implementations. Additionally, association rules, including market-basket analysis using methods like Apriori algorithm, are explored through clear examples. Comprehensive coverage of clustering methods like K-means further enhances practical understanding. Readers will appreciate step-by-step programmatic demonstrations accompanied by detailed interpretations of results. Ultimately, this chapter equips readers with the ability to leverage unsupervised learning techniques effectively, enriching their analytical toolkit for handling real-world, unlabeled datasets.

Keywords Unsupervised learning · Association rules · Apriori algorithm · Support · Confidence · Lift · Market basket analysis · Network diagram · K-means · Clustering · Segmentation

Unsupervised learning is a type of machine learning, in which the algorithms learn from data without being told the right answers. The goal is to discover patterns and structures in the data on its own. It's like giving a machine a puzzle without the picture on the box.

Why is unsupervised learning needed? It's useful because we often have lots of data, but we don't know what it means. For example, imagine having thousands of photos without labels. Unsupervised learning can help group similar photos, like all the beach photos together, without needing to know the labels in advance.

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_15

15.1 Different Types of Unsupervised Learning

Clustering is about putting similar data points together into groups. It looks for how these points are alike naturally. Examples of methods used for clustering include K-Means and hierarchical clustering.

Association is another way to analyze data by finding common patterns or rules in big chunks of data. It's like looking for habits or tendencies that show up a lot, like people who buy bread often buy milk too. Apriori algorithm and Eclat algorithm are methods used here.

Dimensionality reduction is about making the data simpler to understand by reducing the number of things you need to look at. It's like focusing on the important parts and not getting distracted by too much information. Imagine describing a busy street scene with just a few key words instead of a detailed picture. Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) are techniques used for this purpose.

Anomaly detection is used to spot the odd ones out in data, the points that don't fit in with the rest. It's like finding something unusual that stands out. Methods like isolation forest and one-class SVM help with this.

Finally, neural networks or deep learning uses complex networks that can learn and make sense of data on their own, helping to identify patterns that are too complicated for other methods. Techniques in this category include autoencoders and deep belief networks. These methods are good for recognizing complex patterns in data.

15.1.1 Differences Between Supervised and Unsupervised Learning

Unsupervised learning is very different from supervised learning. In supervised learning, the machine is like a student who learns with the help of a teacher. The teacher provides examples with correct answers (like photos labeled as 'beach' or 'mountain'), and the machine learns to recognize and categorize new examples based on these.

15.1.2 Limitations

Unsupervised learning is like exploring a new city without a map, finding patterns and landmarks on your own, while supervised learning is like exploring with a guide who knows the city well. Both have their own roles and challenges in the world of machine learning. Notable limitations are:

Less Accuracy: Because it doesn't have correct answers to learn from, it might not be as precise as supervised learning.

Harder to Evaluate: Without known answers, it's tough to tell how well the machine is doing.
Complexity: Understanding and interpreting the results can be more complex than in supervised learning.

15.2 Association Rules and Apriori Algorithm (Agrawal et al. 1993)

Association rules are a way to find out how items are related to each other in large datasets. Think of it like discovering that when someone buys a phone, they often buy a phone case too. These rules are used to uncover these kinds of relationships or patterns in data.

Association rules are common in market basket analysis. This is like understanding what people often buy together in a supermarket. For example, finding out that people who buy pasta also tend to buy tomato sauce. They're also used in other areas like health care, web usage mining, and recommendation systems.

There are different algorithms to find these rules, and one of the most famous is the Apriori algorithm. The Apriori algorithm is like a detective who first finds all the single items that appear frequently in the dataset. Then, it combines them to see which item sets appear together frequently. It's like first noticing a lot of people buy milk, and a lot buy bread, and then realizing many buy both together.

However, like all methods, association rules and the Apriori algorithm have limitations.

Time-consuming: Especially with large datasets, they can be slow. It's like going through a huge shopping list to find patterns.
Lots of Rules: Sometimes, they can produce too many rules, which can be overwhelming. It's like having too many clues in a mystery.
Not Always Useful: Some of the rules found might not be very useful or might state the obvious. Like finding out that people who buy swimsuits often go to the beach.

In simple terms, association rules and the Apriori algorithm help us find hidden patterns in large amounts of data, like a detective looking for clues in a case.

15.2.1 Example

Let's look at the example of 100 customer transactions to understand how often items are bought together. In these transactions, 30 customers bought bread, 20 customers bought butter, and 15 customers bought both bread and butter. Metrics like support, confidence, and lift are used to understand the association.

Table 15.1 Association rules

Metric	Computation	Interpretation
Support for bread	30/100 = 0.3 or 30%	Bread is purchased in 30% of all transactions
Support for butter	20/100 = 0.2 or 20%	Butter is purchased in 20% of all transactions
Support for bread and butter	15/100 = 0.15 or 15%	Bread and butter are bought together in 15% of all transactions
Confidence (bread → butter)	15/30 = 0.5 or 50%	When bread is bought, there is a 50% chance of butter being bought in the same transaction
Lift (bread → butter)	0.50/0.2 = 2.5	The likelihood of buying butter when bread is bought is 2.5 times higher than the likelihood of buying butter in general

Support for Bread: Thirty customers bought bread. This means 30 out of 100 transactions included bread. So, bread was in 30% of all transactions.

Support for Butter: Twenty customers bought butter, so butter was in 20% of all transactions.

Support for Bread and Butter: Fifteen customers bought both bread and butter. This tells us that in 15 out of 100 transactions, customers bought both items. So, bread and butter were bought together in 15% of all transactions.

Confidence (Bread → Butter): Out of the 30 customers who bought bread, 15 also bought butter. This means if a customer buys bread, there's a 50% chance that they will also buy butter.

Lift (Bread → Butter): The chance of bread and butter being bought together is 2.5 times higher than the chance of just butter being bought. This suggests that bread and butter are more likely to be purchased together than butter alone.

In simple terms, these numbers help a store understand how items are connected. For example, they might decide to place bread and butter closer together in the store because customers like to buy them at the same time.

Table 15.1 summarizes the computation.

15.2.2 Implementation of Association Rules and Apriori Algorithm Programmatically

- Import necessary modules

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
import networkx as nx
import matplotlib.pyplot as plt
```

- Transactions' data

```
transactions = [
  ["egg", "sugar", "cheese", "coffee"],
  ["egg", "sugar"],
  ["egg", "coffee", "tea"],
  ["egg", "sugar", "coffee", "tea"]
```

- Encoding the transactions data using one-hot encoding

```
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_onehot = pd.DataFrame(te_ary, columns=te.columns_)
```

- Applying the Apriori algorithm to find frequent itemsets

```
frequent_itemsets = apriori(df_onehot, min_support=0.1, use_colnames=True)
```

- Generating the association rules

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
```

- Display the top rules—sorted by lift

```
top_rules = rules.sort_values(by='lift', ascending=False).head()
```

- Calculate overall support, confidence, and lift for 'egg', 'sugar', 'cheese', and 'coffee'

```
overall_metrics = {}
for item in ['egg', 'sugar', 'cheese', 'coffee']:
  filtered_rules = rules[(rules['antecedents'].apply(lambda x: item in x)) |
                (rules['consequents'].apply(lambda x: item in x))]
  overall_metrics[item] = {
    'support': filtered_rules['support'].mean(),
    'confidence': filtered_rules['confidence'].mean(),
    'lift': filtered_rules['lift'].mean()
  }
```

- Create a network graph for visualization

```
G = nx.Graph()
for _, row in rules.iterrows():
    antecedents = list(row['antecedents'])
    consequents = list(row['consequents'])
    for a in antecedents:
        for c in consequents:
            G.add_edge(a, c)
```

- Draw the network graph

```
nx.draw(G, with_labels=True, node_color='lightblue', font_size=10, node_size=2000)
plt.show()
```

- Display the top rules

```
top_rules
```

- Output
  See Fig. 15.1.

The network diagram visualizes the relationships laid out in the below table. The size of the nodes (like 'cheese' and 'egg') represents the frequency of the item in the dataset, and the lines between them represent the association rules. The diagram makes it easier to see which items are often associated with one another. For example,

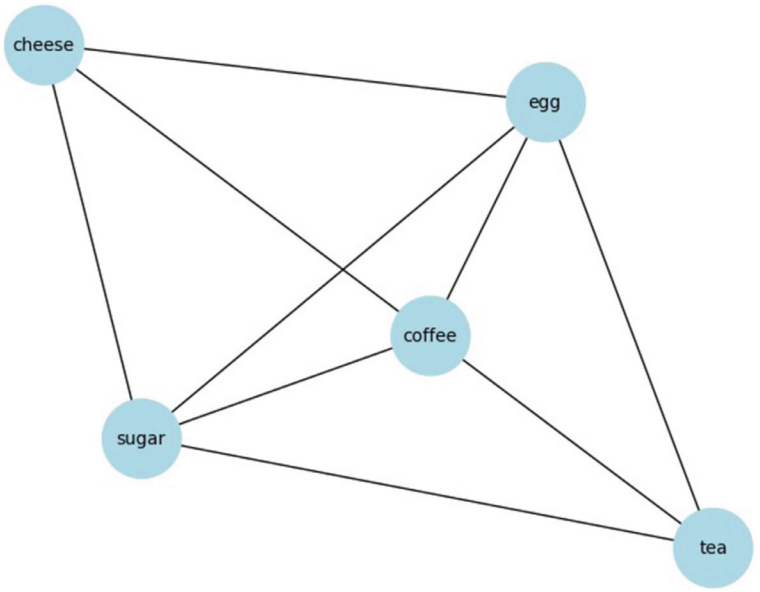


Fig. 15.1 Network Diagram

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
20	(cheese)	(coffee, sugar)	0.25	0.50	0.25	1.0	2.0	0.125	inf	0.666667
48	(coffee, sugar)	(egg, cheese)	0.50	0.25	0.25	0.5	2.0	0.125	1.5	1.000000
50	(egg, cheese)	(coffee, sugar)	0.25	0.50	0.25	1.0	2.0	0.125	inf	0.666667
51	(cheese)	(coffee, sugar, egg)	0.25	0.50	0.25	1.0	2.0	0.125	inf	0.666667
45	(coffee, sugar, egg)	(cheese)	0.50	0.25	0.25	0.5	2.0	0.125	1.5	1.000000

Fig. 15.2 Output of the program

'cheese' is connected to 'coffee' and 'sugar', indicating that these items are frequently purchased together (Fig. 15.2).

Let us understand the metrics in the above table.

Antecedents: These are items that are found in combination with other items. For instance, 'cheese' is an antecedent to 'coffee, sugar'.

Consequents: These are items that are often bought with the antecedents. For example, when 'coffee, sugar' is bought, 'egg, cheese' is often bought too.

Support: This is how often a rule is found in the dataset. A support of 0.25 for 'cheese' → 'coffee, sugar' means this combination appears in 25% of all transactions.

Confidence: This measures how often items in the consequent are found in transactions that contain the antecedents. For example, 'cheese' → 'coffee, sugar' has a confidence of 1, meaning that 'coffee, sugar' is always bought when 'cheese' is bought.

Lift: This tells us how much more often the antecedent and consequent occur together than we would expect if they were statistically independent. A lift of 2 for 'egg, cheese' → 'coffee, sugar' means the presence of 'egg, cheese' increases the likelihood of 'coffee, sugar' being bought by 2 times.

Leverage: This shows the difference in the probability of the antecedent and consequent appearing together and the probability of them appearing independently. Positive values indicate a positive association.

Conviction: This measures the reliance of the consequent on the antecedent. A higher value suggests a strong rule. An 'inf' (infinity) value, as seen in the table for rule 50 and 51, suggests that the consequent is highly dependent on the antecedent.

Zhang's Metric: This is another measure of the rule's reliability, with values closer to 1 indicating stronger association.

15.3 Clustering

Clustering is a way to group similar things together. Imagine you have a bunch of fruits and you sort them into piles of apples, bananas, and oranges based on their shape and color. That's what clustering does with data.

It's useful because it helps to organize data in a way that we can understand better. For example, a company might have information about their customers' shopping habits. Clustering can help the company see which customers buy similar things, which can then help with marketing or sales strategies.

15.3.1 Examples of Clustering

- Grouping customers by buying habits, as mentioned.
- Organizing different types of plants or animals based on their characteristics.
- Sorting news articles into topics like sports, politics, or entertainment.

15.3.2 Different Algorithms Used for Clustering

K-Means: This is like choosing a fixed number of spots (centroids) and then grouping data points based on which spot they're closest to.

Hierarchical: This is like building a family tree for the data, deciding who's related to whom, and how closely.

DBSCAN: This finds clusters based on how close data points are to their neighbors, without needing to know how many clusters there should be beforehand.

15.3.3 Limitations of Clustering

Choosing Parameters: For some algorithms like K-Means, you need to decide how many groups you want before you start, which can be tricky.

Varied Sizes and Densities: Clusters can come in all shapes and sizes, and some algorithms have a hard time if the clusters are too different from one another.

Noise and Outliers: If there are weird or unusual data points, they can throw off the clustering process.

In simple terms, clustering helps us find groups in data, which can be really useful for making sense of information and making decisions.

15.3.4 Applying K-Means Algorithms in Clustering

K-Means clustering is a straightforward way to organize data into groups. It's like sorting socks into piles of the same color.

15.3.5 Steps Involved in K-Means Clustering

- Choose the Number of Clusters (k): First, you decide how many groups (clusters) you want to make. This is the 'K' in K-Means. K is a hyperparameter.
- Select Centroids: The algorithm picks 'k' different points in the data at random. These points are the starting center points of the clusters, called centroids.
- Assign Points to the Nearest Centroid: Each piece of data is then put into the group of the closest centroid. It's like assigning each sock to the pile with the nearest matching color.
- Computing the Distance: To determine which cluster a data point belongs to, K-Means calculates the distance from each data point to each centroid. It's like measuring how far each sock is from a color pile. The most common way to measure this distance is by using something called 'Euclidean distance,' which is like drawing a straight line between two points. The data point is then assigned to the nearest centroid based on these distance measurements. This step is crucial because it directly affects the formation of clusters.
- Adjust Centroids: After all data points are assigned, the center point of each cluster is recalculated. This new point is the average of all the points in that cluster.
- Repeat Assigning and Adjusting: Steps 3 and 4 are repeated. Data points might move to different clusters as centroids adjust. This process goes on until the centroids don't move much anymore, meaning that the clusters are stable.

K-Means is useful because it's simple and fast. But it works best when clusters are round and roughly the same size. If clusters are stretched out or very different from each other, K-Means might not work as well.

Let us consider the customer scenario shown in Table 15.2.

The distance between the center and any of the other points is computed using the formula for Euclidean distance (Fig. 15.3).

Table 15.2 Clustering example

Customer	Spending	Visits/month
Customer 1	$200	3
Customer 2	$50	1
Customer 3	$300	4
Customer 4	$225	3
Customer 5	$75	2

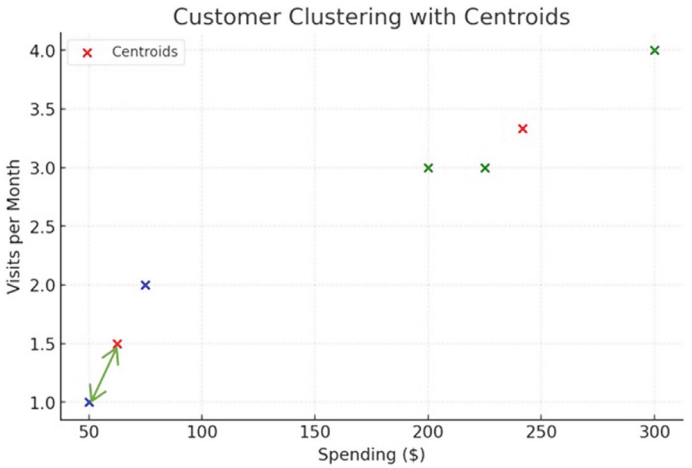


Fig. 15.3 Clustering

$$\text{Euclidean Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

15.3.6 Finding Clusters Programmatically

- Import necessary modules

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

- Load the dataset

```
df = pd.read_csv('/content/customers_data_for_clustering.csv')
```

- Select the features for clustering

```
features = df[['NumberOfVisits', 'Spending']]
```

- Using K-Means clustering with three clusters

```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(features)
```

- Adding the cluster labels to the original dataframe

```
df['Cluster'] = kmeans.labels_
```

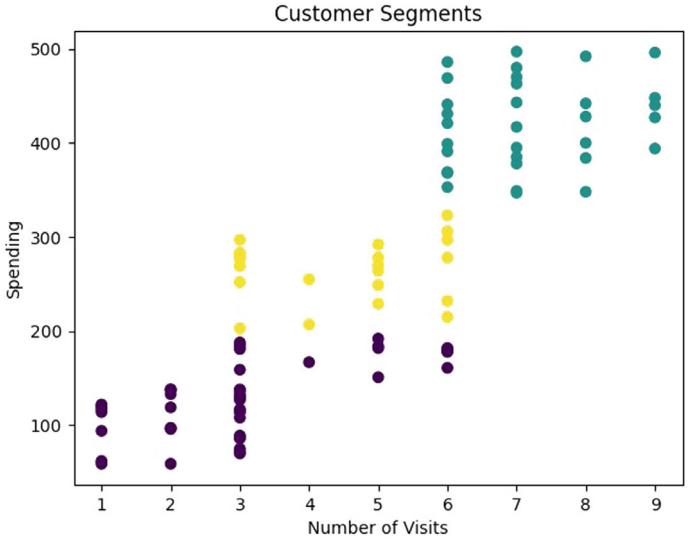


Fig. 15.4 Clustering – Customer Segmentation

- Plotting the clusters

```
plt.scatter(df['NumberOfVisits'], df['Spending'], c=df['Cluster'], cmap='viridis')
plt.xlabel('Number of Visits')
plt.ylabel('Spending')
plt.title('Customer Segments')
plt.show()
```

- Output of the program
  See Fig. 15.4.

The clustering output provides a visual way to understand the data and make informed decisions based on customer behavior patterns.

Axes: The X-axis represents the "Number of Visits" a customer makes, and the Y-axis represents the "Spending" of a customer. Each point on the plot represents a customer.

Clusters: The different colors of the points likely represent different clusters identified by the algorithm. Clusters group customers with similar visit and spending patterns.

Customer Segments: The clustering algorithm has segmented the customers into groups based on their behavior. For example:

- The purple cluster at the bottom left represents customers who visit infrequently and spend less.
- The yellow cluster in the middle represents customers with a moderate number of visits and spending.

Table. 15.3 Comparison of unsupervised learning

Type	Description	Key characteristics	Example methods
Clustering	Grouping data points into clusters based on similarity	• Finds structure in unlabeled data • Groups are defined by the intrinsic characteristics of the data	• K-Means • Hierarchical clustering • DBSCAN
Association	Finding rules or patterns that describe large portions of the data	• Discovers interesting relations between variables • Often used for market basket analysis	• Apriori algorithm • Eclat algorithm
Dimensionality reduction	Reducing the number of features while retaining important information	• Simplifies data without losing much information • Helps in noise reduction and data visualization	• Principal Component Analysis (PCA) • *t*-Distributed Stochastic Neighbor Embedding (*t*-SNE)
Anomaly detection	Identifying data points that deviate significantly from the majority of the data	• Useful in fraud detection, system health monitoring • Works by identifying outliers or unusual data points	• Isolation forest • One-class SVM
Neural networks/deep learning	Utilizing layered neural networks to learn data representations	• Can automatically discover the representations needed for feature detection or classification • Often used for complex pattern recognition	• Autoencoders • Deep belief networks

- The teal cluster on the right represents customers who visit frequently and tend to spend more.

Such clustering can help a business understand its customer base and tailor marketing or loyalty programs. For instance, the business might target the high-spending teal cluster with premium offers, while the purple cluster might be encouraged to visit more often with special deals.

Table 15.3 summarizes the different algorithms and techniques in unsupervised learning.

Reference

Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of Items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on management of data (SIGMOD ’93)

Chapter 16
Deep Learning

Transition from Machine Learning to Deep Learning

Abstract In this chapter, readers are introduced to deep learning, an advanced subset of machine learning that utilizes neural networks inspired by human brain architecture. Building upon foundational concepts from earlier chapters, readers learn how neural networks function by leveraging interconnected layers to identify complex patterns within large datasets. Key terms, such as activation functions—including sigmoid, ReLU, and softmax—are discussed clearly to illustrate their roles in neural network behavior. Readers gain insights into the distinct evolution and growing significance of Artificial Neural Networks (ANN), understanding their real-world applications. Specialized neural network structures, namely Convolutional Neural Networks (CNNs) for image analysis and Recurrent Neural Networks (RNN) for sequential data, are also explored. Advantages, limitations, and scenarios where each network type performs optimally are presented clearly. Ultimately, this chapter prepares readers to confidently approach complex real-world applications using deep learning algorithms. It is worth noting that RNNs are the foundational building blocks behind transformer-based architectures like transformers and GPT models, including ChatGPT.

Keywords Deep learning · Neural networks · Artificial neural networks · Convolutional neural networks · Recurrent neural networks · Feed-forward neural networks · Action functions · ReLU · Tanh · Softmax · Pooling · Fully connected layer · LSTM

This is the last chapter of the book. It offers information to help readers transition from machine learning to deep learning, highlighting the differences between the two. This chapter aims to make it easier for learners to understand how deep learning differs from machine learning. Please note, the content in this chapter is not included in the video lectures.

sn.pub/Lia1RD

G. Madhavan, *Mastering Machine Learning: From Basics to Advanced*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-97-9914-5_16

16.1 Introduction to Neural Networks

Neural networks are a subset of machine learning algorithms that are inspired by the structure and function of the human brain. Neural networks are made up of interconnected artificial neurons, which are mathematical models of biological neurons. Artificial neurons process information by passing signals to each other, and the strength of the signal between two neurons is determined by the weight of the connection between them.

Neural networks can be trained to perform a variety of tasks, such as image recognition, natural language processing, and machine translation. To train a neural network, you provide it with a dataset of examples of the task you want it to learn. The neural network then adjusts the weights of its connections until it can accurately predict the output for the given input (Fig. 16.1).

16.1.1 Evolution of Neural Networks

The name "Neural Network" was first coined by Warren McCulloch and Walter Pitts in their 1943 paper, "A Logical Calculus of the Ideas Immanent in Nervous Activity". In this paper, McCulloch and Pitts proposed a mathematical model of a neuron, which is the basic building block of the nervous system. They showed that networks of these artificial neurons could be used to perform logical operations (McCulloch and Pitts 1943).

In 1957, Frank Rosenblatt published a paper called "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In this paper, Rosenblatt developed a new type of neural network called the perceptron. The perceptron was a simple neural network that could learn to perform simple tasks, such as recognizing handwritten characters (Rosenblatt 1957).

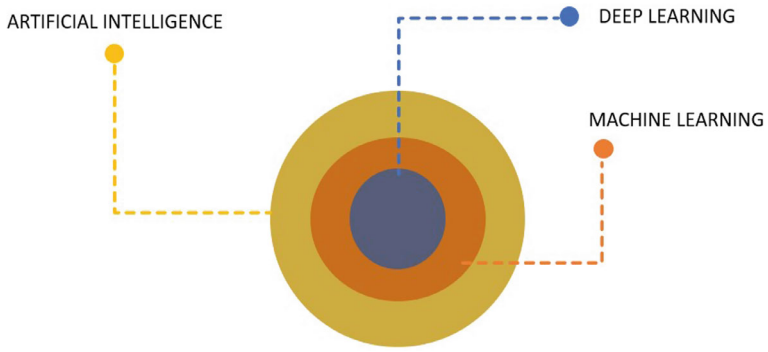


Fig. 16.1 Deep learning

Neural networks experienced a dry spell of research in the 1970s, largely due to a lack of funding. However, in 1982, Jon Hopfield presented his paper on recurrent neural networks, "Neural Networks and Boltzmann Machines," which sparked a renewed interest in the field (Hopfield 1982).

One of the key breakthroughs during this period was the resurgence of the back-propagation algorithm. Paul Werbos is often credited with the primary contribution to this development in his Ph.D. thesis rewrite, "Beyond Regression: New Tools for Prediction and Analysis." (Werbos 1974, 1994).

Backpropagation is a supervised learning algorithm that allows neural networks to learn from their mistakes. It works by calculating the gradient of the error function with respect to the weights of the network and then using this information to adjust the weights to minimize the error.

The development of backpropagation algorithm made it possible to train neural networks to solve complex problems, such as image recognition and natural language processing. As a result, neural networks have become one of the most powerful tools in machine learning today.

Most recently, more specific neural network projects are being generated for direct purposes. For example, Deep Blue, developed by IBM, conquered the chess world by pushing the ability of computers to handle complex calculations.

16.1.2 Working of Neural Networks

Neural networks emulate the fundamental processes of the human brain, inspired by how our brains interpret information. They excel in real-time tasks due to their ability to perform rapid computations and deliver swift responses.

Artificial neural networks consist of a vast number of interconnected processing elements, or Nodes, linked to one another through connection links. These links possess weights containing information about input signals. The weights are updated with each iteration and input, ultimately resulting in a trained neural network after processing all data instances from the training dataset.

This process, known as neural network training, equips these networks to address specific problems as defined in the problem statement. These problems include tasks like classification, pattern matching, and data clustering.

We use artificial neural networks because they learn very efficiently and adaptively. They have the capability to learn "how" to solve a specific problem from the training data it receives. After learning, it can be used to solve that specific problem very quickly and efficiently with high accuracy.

16.1.3 Types of Neural Networks

1. Feedforward Neural Networks (Artificial Neural Network—ANN): Feedforward neural networks are the simplest type of neural networks. They have a single input layer, a single output layer, and one or more hidden layers. The information flows in one direction only, from the input layer to the output layer through the hidden layers. Feedforward neural networks are often used for classification and regression tasks.

   Use Cases: Image classification, handwriting recognition, medical diagnosis, financial forecasting.
2. Convolutional Neural Networks (CNNs): CNNs are a type of neural network that is well-suited for image recognition tasks. CNNs have a special type of layer called a convolutional layer, which is designed to extract features from images. CNNs are also able to learn spatial relationships between pixels in an image. CNNs are widely used in image recognition, object detection, and image segmentation tasks.

   Use Cases: Image recognition, image segmentation, object detection
3. Recurrent Neural Networks (RNNs): RNNs are a type of neural network that is well-suited for processing sequential data, such as text or audio. RNNs have a feedback loop that allows them to remember information from previous inputs. This makes them well-suited for tasks such as machine translation, text summarization, and speech recognition (Fig. 16.2).
4. Long Short-Term Memory (LSTM) Networks: LSTM networks are a type of RNN that is specifically designed to learn long-term dependencies in sequential data. LSTM networks are widely used in natural language processing and machine translation tasks.
5. Generative Adversarial Networks (GANs): GANs are a type of neural network that consists of two competing networks—a generator network and a discriminator network. The generator network tries to generate synthetic data that is

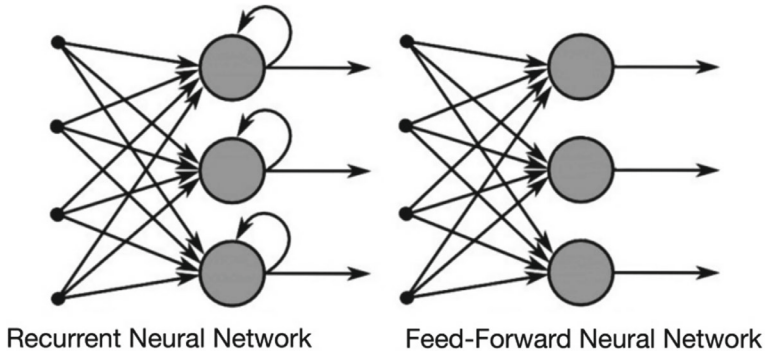


Fig. 16.2 Feedforward versus recurrent neural networks)

indistinguishable from real data, while the discriminator network tries to distinguish between synthetic and real data. GANs are widely used in image generation, text generation, and music generation tasks.

16.2 What Are Activation Functions

Activation functions are crucial components in neural networks that apply a non-linear transformation to the input signal, allowing the network to learn complex patterns in the data. They decide whether a neuron should be activated or not by calculating the weighted sum of its inputs and then applying a predefined function to this sum (Fig. 16.3).

16.2.1 Why We Need Activation Functions

The primary purpose of activation functions is to introduce non-linearity into the output of a neuron. This is essential because real-world data is complex and non-linear, and without non-linearity, a neural network would essentially operate as a linear regression model, limiting its ability to solve complex problems.

16.2.2 Types of Activation Functions

Activation functions can be broadly categorized into linear and non-linear types. Linear activation functions are rarely used in practice because they do not allow the network to model complex relationships. Non-linear functions, such as the Sigmoid, Tanh, Rectified Linear Unit (ReLU), and their variations, are commonly used because they enable the network to capture non-linearities in the data.

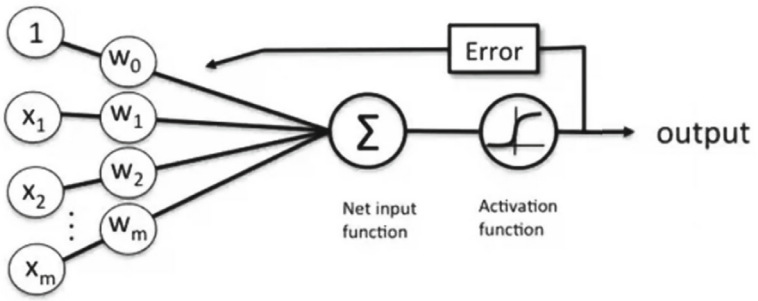


Fig. 16.3 Activation function

16.2.3 Role of Activation Functions Across Domains

The choice of activation function can significantly impact the performance of a neural network across different tasks and domains, such as image recognition, natural language processing, and more. Different functions have different mathematical properties, such as the range of output values and how they propagate gradients during backpropagation, which can affect the learning dynamics and the ability of the network to converge to a good solution.

16.2.4 Linear Activation Functions

A linear activation function applies a linear transformation to the input data. In the context of neural networks, it means that the output of the activation function is directly proportional to the input. The most general form of a linear activation function can be expressed as:

$$f(x) = ax + b,$$

where

- x is the input to the function.
- a is the slope or the coefficient.
- b is the bias or intercept.

The parameters a and b define the linearity of the function. In its simplest form, where $a = 1$ and $b = 0$, the linear activation function becomes $f(x) = x$, meaning that the output is identical to the input.

16.2.5 Characteristics of Linear Activation Functions

Unlike non-linear activation functions (e.g., ReLU, Sigmoid, Tanh), linear activation functions do not introduce non-linearity into the neural network. This means that no matter how many layers a neural network has, if all are equipped with linear activation functions, the entire network will still be equivalent to a single-layer network. This is because the composition of linear functions is itself a linear function.

Linear activation functions produce outputs that range from negative infinity to positive infinity. This unbounded nature can lead to issues in certain tasks where the output needs to be within a specific range, such as probability predictions in classification tasks.

Linear activation functions are typically used in regression problems or as the activation function in the output layer for problems where we want to predict a value

that can have a wide range of real numbers. However, they are less common in hidden layers of deep neural networks, where non-linearity is crucial for modeling complex patterns.

Limitations of Linear Activation Functions

Because they do not introduce non-linearity, networks with linear activation functions cannot model complex relationships in the data as effectively as those with non-linear activation functions.

In networks with only linear activations, gradient descent optimization becomes straightforward but limited. The network's ability to learn complex patterns and improve during training is significantly constrained compared to networks with non-linear activations.

Despite these limitations, linear activation functions have their place in neural network architectures, particularly in the output layer for certain types of regression tasks. However, for most deep learning tasks, especially those involving complex data patterns and classifications, non-linear activation functions are preferred.

16.2.6 Non-linear Functions

Non-linear activation functions are widely employed in neural networks because they enhance the model's ability to adapt to diverse datasets and distinguish between different outcomes effectively.

The non-linear output after the application of the activation function is given by:

$$y = \alpha(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b),$$

where α is the activation function.

16.2.7 Sigmoid Function

The sigmoid function is particularly useful in scenarios where we need to predict something that has a probability nature, such as whether an email is spam or not. It's mathematically represented as

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

This formula effectively squashes the input values to a range between 0 and 1, which is useful for binary classification problems. The advantages of the sigmoid function include its smooth gradient, which prevents sudden jumps in output values. However, it has notable disadvantages like the vanishing gradient problem, where

gradients become very small, impeding the network from learning effectively. Another issue is that its output is not zero-centered, which can make the optimization process less efficient.

16.2.8 Tanh Activation Functions

The tanh (hyperbolic tangent) function is another activation function, similar to sigmoid but with an output ranging from -1 to 1. Its mathematical expression is

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

This range makes the tanh function zero-centered, which generally leads to better performance in practice compared to the sigmoid function. However, tanh also suffers from the vanishing gradient problem, especially for inputs that are significantly large or small.

16.2.9 Rectified Linear Unit (ReLU)

ReLU's simplicity has made it a popular choice in many neural network architectures. The formula for ReLU is $f(x) = \max(0, x)$, meaning that it outputs the input directly if it's positive, else it will output zero. This simplicity allows for faster computation compared to other activation functions. ReLU helps alleviate the vanishing gradient problem, allowing models to learn faster and perform better, especially in deep networks. However, ReLU is not without its flaws. It can lead to what's known as the dying ReLU problem, where some neurons essentially die, meaning that they stop outputting anything other than zero. In such cases, these neurons stop participating in the data processing and learning, which can be a significant drawback.

16.2.9.1 Popular Types of ReLU

The standard ReLU function is the simplest type of ReLU activation function. It outputs its input if it is positive, and 0 otherwise (Table 16.1).

The leaky ReLU function is a modified version of the standard ReLU function that outputs a small negative value (α) instead of 0 for negative inputs. This helps to prevent the "dying neuron" problem, where neurons become deactivated and never learn. In practice, alpha is usually a small number, such as 0.01, though it can be tuned as a hyperparameter depending on the specific application or network architecture.

The parameterized ReLU function is a more general form of the leaky ReLU function that allows the negative slope to be learned by the network. This can help to

Table 16.1 Types of ReLU

Type	Equation	Negative input	Learning
Standard ReLU	$f(\mathrm{x}) = \max(0, x)$	0	No
Leaky ReLU	$f(\mathrm{x}) = \max(\alpha x, x)$	αx, $\alpha > 0$	No
Parameterized ReLU	$f(\mathrm{x}) = \max(\beta x, x)$	βx, β is learned	Yes
Exponential linear unit (ELU)	$f(\mathrm{x}) = x$ if $x \geq 0$, $\alpha(\exp(x) - 1)$ if $x < 0$	$\alpha(\exp(x) - 1)$, $\alpha > 0$	No

improve the performance of the network on certain tasks. β is a parameter that, like α in leaky ReLU, is used for the input x when x is less than zero. The key difference between PReLU and leaky ReLU is that in PReLU, beta is not a predetermined small constant. Instead, beta is a learnable parameter. This means that during the training of a neural network, beta is adjusted along with other parameters in the network (like weights and biases) through backpropagation and gradient descent methods.

The ELU function is a smooth ReLU activation function that has a negative slope for negative inputs. This helps to prevent the dying neuron problem and makes the ELU function more robust to noise. Unlike leaky ReLU or PReLU, ELU saturates to a negative value (approximately − alpha) for large negative inputs. This can help with reducing the "vanishing gradient problem".

16.2.10 Softmax Function

The softmax function is commonly used in the output layer of neural networks for multiclass classification. It converts the output into a probability distribution over predicted output classes. The formula for the softmax function is

$$f(x_i) = \mathrm{e}^{x_i} \Big/ \sum \mathrm{e}^{x}$$

for each output x_i. It's particularly useful in scenarios where we need to categorize inputs into multiple categories, like recognizing handwritten digits. While it's excellent for multiclass problems, it's less suitable for binary classification tasks where simpler functions like sigmoid can be more efficient.

16.2.11 Use Cases of Neural Networks

Neural networks identify objects in images like faces, cars, and signs, applied in self-driving cars, facial recognition, and image search engines. They can also understand and generate human language, used in machine translation, chatbots, and voice assistants. They can transcribe speech to text, utilized in dictation software, voice

Table 16.2 Types of activation functions

Activation function	Formula	Output range	Advantages	Disadvantages
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	0 to 1	Smooth gradient, bounded output	Vanishing gradient problem, not zero-centered
ReLU	$f(x) = \max(0, x)$	0 to ∞	Fast computation, reduces vanishing gradient problem	Dying ReLU problem (neurons may stop learning)
Tanh	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1.$	-1 to 1	Zero-centered, smooth gradient	Vanishing gradient problem
Softmax	$f(x_i) = e^{x_i} / \sum e^{x}$	0 to 1 (probabilities)	Useful for multiclass classification	Not suitable for binary classification

assistants, and call centers. They are effective in diagnosing diseases and suggesting treatments by analyzing medical images (X-rays and MRIs) and patient data patterns.

Neural networks can forecast stock prices and financial data, informing investment decisions for hedge funds and financial institutions. They are adept at recommending products, movies, and content to users, benefiting e-commerce, streaming services, and social media platforms.

Last but not the least, neural networks enable superhuman game playing, used in video game development and training game-playing robots.

Table 16.2 summarizes the different types of activation functions.

16.3 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a type of deep learning neural networks that are particularly well-suited for image processing and recognition. They are inspired by the structure and function of the visual cortex, which is the part of the human brain that is responsible for processing visual information.

Introduction to Convolutional Neural Networks

CNNs are made up of a series of layers, each of which performs a specific task. The first layer typically extracts basic features from the image, such as edges and corners. Subsequent layers extract more complex features, such as shapes and objects. The final layer of the CNN typically classifies the image into a specific category (Fig. 16.4).

Consider a simple CNN that is used to classify images of lions and cats. The first layer of the CNN might extract basic features from the image, such as edges and corners. The second layer might extract more complex features, such as shapes like

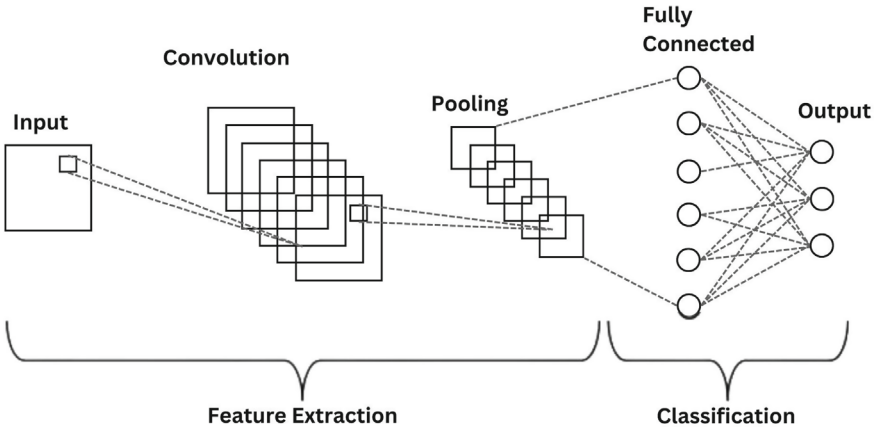


Fig. 16.4 Feature extraction and classification in CNN

eyes, noses, and mouths. The third and final layer might then classify the image as either a cat or a lion.

Let us now understand the three layers of CNN in detail (Fig. 16.5).

1. Convolutional layer.
2. Pooling layer.
3. Fully connected layer.

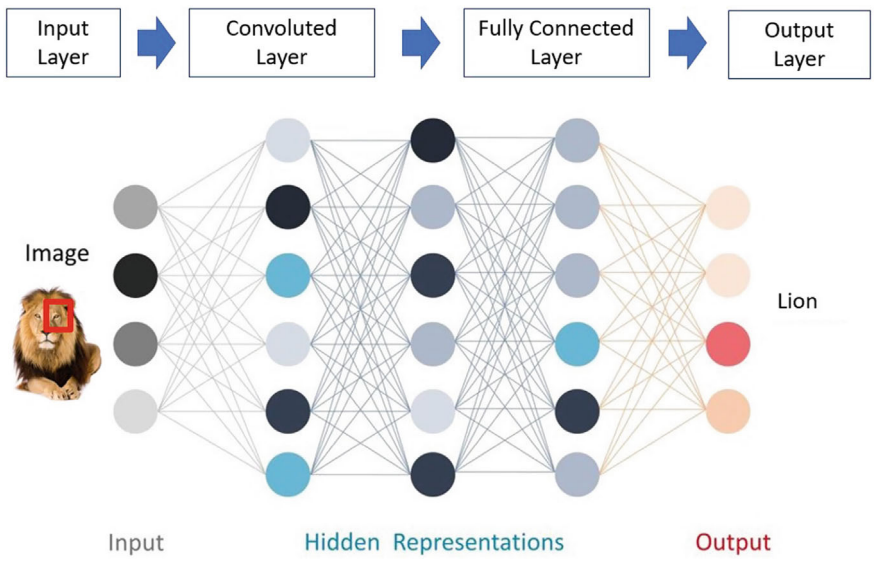


Fig. 16.5 CNN showing different layers

16.3.1 Layers of CNN

16.3.1.1 Convolutional Layer

A convolutional layer is a cornerstone of convolutional neural networks (CNNs), designed to process and transform input data in a way that emphasizes the extraction of spatial features. This layer utilizes a combination of input data, specialized filters, and the generation of feature maps to achieve its task. The structure and function of this layer are geared toward identifying patterns within the data, such as edges or textures in images, by maintaining the spatial relationships between pixels (Fig. 16.6).

The essence of the convolutional process lies in its method of applying filters to the input data. By moving small, square sections (kernels) across the entire input image, the layer performs element-wise multiplication between the kernel and the section of the image it covers. This method ensures that the convolution operation retains the spatial hierarchy of pixel arrangements, allowing the network to learn varying features based on the complexity and depth of the layer. The operation is akin to looking through a small window that slides across the image, capturing localized patterns.

Kernels within the convolutional layer are tasked with detecting specific types of features, such as edges or color gradients, by activating strongly in response to the presence of these features. As the kernel slides—or convolves—over the image, it produces a new matrix known as the convolved feature or feature map. This feature map represents a filtered version of the input image, emphasizing the detected features and diminishing the less relevant information. This selective emphasis allows CNNs to focus on the most informative parts of the input for further processing.

The convolution operation itself is a calculated blend of multiplication and addition. Each pixel value from the input image (first tensor) is multiplied by a corresponding weight from the kernel (second tensor), and these products are then summed together to form a single value in the output feature map. Through this meticulous process, CNNs are capable of transforming raw data into a form that highlights essential features, setting the stage for advanced analysis and interpretation in subsequent layers of the network. This methodology underscores the convolutional layer's

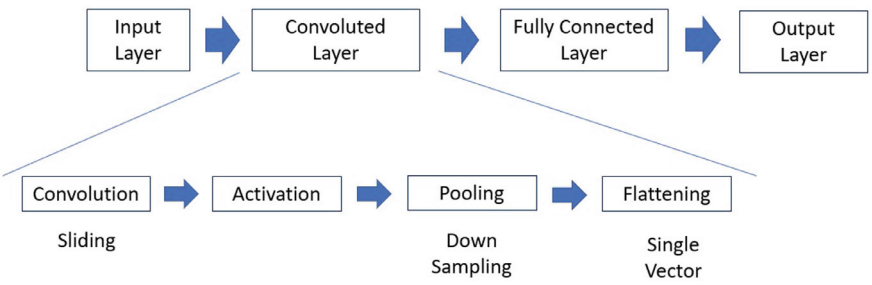


Fig. 16.6 Components of convoluted layer

Table 16.3 Sample 5 × 5 grid to explain convolution operation

10	20	30	15	18
40	50	60	32	37
70	80	90	43	41
25	35	45	58	85
55	65	75	76	86

Table 16.4 Sample 2 × 2 grid used as a filter in convolution operation

1	0
− 1	1

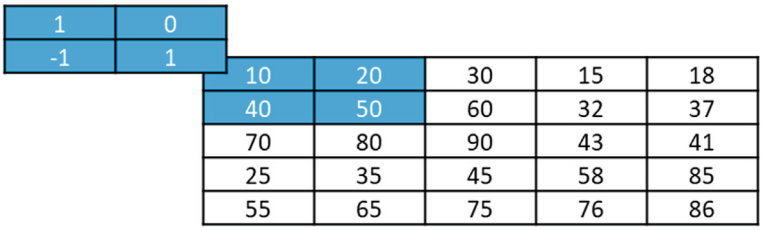


Fig. 16.7 Convolution operation

pivotal role in enabling deep learning models to understand and classify complex visual data effectively.

Understand the Math Behind the Convolution Operation

Let us assume an image with a 5 × 5 grid of pixels. Each pixel has a grayscale value (let's say from 0 to 255, where 0 is black and 255 is white) (Table 16.3).

We will use a 2 × 2 grid as a filter. Each value in the filter is a weight (Table 16.4).

Convolution Operation

The convolution involves sliding the filter over the image, usually starting from the top-left corner, and then moving it across the image. At each position, we multiply the values in the filter with the corresponding values in the image and sum them up. This sum becomes a single value in the output feature map.

Let's perform the convolution for the first position by placing the filter over the top-left corner of the image (Fig. 16.7).

Perform element-wise multiplication and sum:

$$(10 * 1) + (20 * 0) + (40 * -1) + (50 * 1) = 10 + 0 - 40 + 50 = 20.$$

This sum is placed in the corresponding position in the output feature map. So, the top-left value of our feature map is 20. We continue this process, sliding the filter right (and then down, row by row) and repeating the multiplication and summation. If our image is larger, or if we have padding (extra pixels around the image) or a stride (steps we move the filter each time), the process incorporates those elements, but the basic operation remains the same.

Result

The result is a new matrix, the feature map, which highlights the features that the filter detects in the original image. If our filter is designed to detect edges, for instance, areas with high values in the feature map might indicate the presence of an edge. This process is a core part of how CNNs process and learn from image data, allowing them to detect complex patterns and features in images.

The behavior of the convolutional layer is primarily governed by the following main hyperparameters:

Kernel Size: The size of the sliding window. Smaller kernel sizes are generally recommended, preferably odd values such as 1, 3, 5, and occasionally, rarely 7.
Stride: The number of pixels the kernel window will move during each step of convolution. Typically, the stride is set to 1 to ensure that no locations are missed in an image. However, the stride can be increased to reduce the input size.
Padding: The technique of adding zeros to the border of an image. Padding allows the kernel to fully filter every position of an input image, ensuring that even the edges are properly processed.
Number of Filters (Depth): The number of patterns or features that the layer will seek to identify. In other words, the depth governs the number of distinct characteristics or elements that the convolutional layer will focus on detecting.

To calculate the output size of the convoluted image, we can apply the following formula:

$$\text{output\_size} = 1 + (\text{input\_ size} - \text{kernel\_ size} + (2 * \text{padding})) \big/ \text{stride}.$$

Example:

- Kernel with dimensions of 3×3 pixels, a stride of 1, and no padding.
- output_size $= 1 + (6 - 3 + (2 * 0))/1 = 1 + (3/1) = 1 + 3 = 4$ (plugging values).
- Resulting convoluted image will have dimensions of 4×4 pixels.

When the input has more than one channel, the filter should have a matching number of channels. To calculate one output cell, perform convolution separately on each pair of matching channels, sum the results across all channels, and optionally add a bias term, producing the final output value.

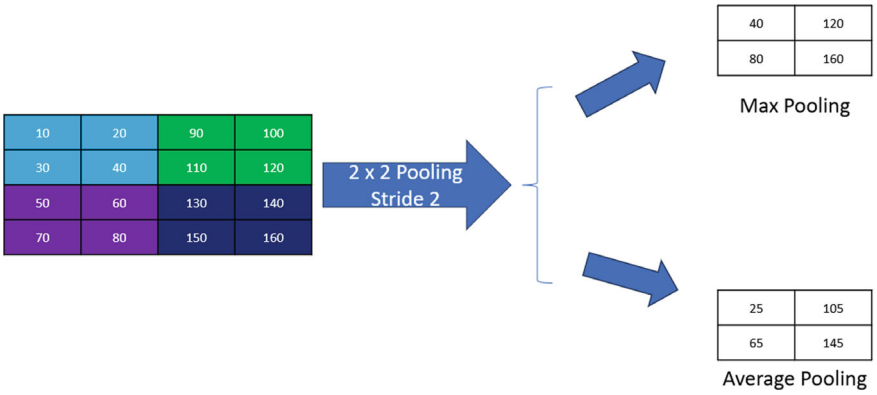


Fig. 16.8 Pooling operation in CNN

16.3.1.2 Pooling Layer

Pooling layers are a type of layer in convolutional neural networks (CNNs) that reduce the dimensionality of the input feature maps. This is done by applying a pooling operation to each feature map, which produces a new feature map with a smaller spatial dimension.

The two most common pooling operations are:

Max pooling: Selects the maximum value from a region of the input feature map.
Average pooling: Computes the average value from a region of the input feature map.

Pooling layers are used after convolutional layers in a CNN. This is because convolutional layers extract features from the input image, while pooling layers reduce the dimensionality of the feature maps, which makes the CNN more efficient to train and can also help to prevent overfitting (Fig. 16.8).

16.3.1.3 Fully Connected Layer

The fully connected layer is responsible for combining the high-level features learned by the convolutional and pooling layers into a single output. It does this by connecting each neuron in the fully connected layer to every neuron in the previous layer. The weights of these connections are learned during training, and they allow the fully connected layer to learn complex relationships between the different features.

The fully connected layer is typically the last layer in a CNN, but it can also be used in other parts of the network, such as after a pooling layer or between two convolutional layers (Fig. 16.9).

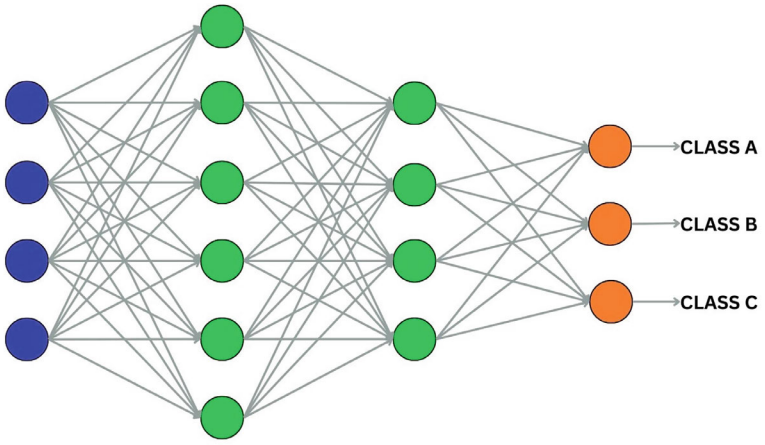


Fig. 16.9 Fully connected layer

16.3.2 Types of CNN

There are many different types of convolutional neural networks (CNNs), each with its own advantages and disadvantages. Some of the most common CNN architectures include:

AlexNet: AlexNet was one of the first CNNs to achieve state-of-the-art results on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). It is a relatively simple architecture, but it is still effective for many tasks (Krizhevsky et al. 2012).
VGGNet: VGGNet is a family of CNNs that are based on the AlexNet architecture. VGGNets are known for their depth and their ability to learn complex features (Krizhevsky et al. 2012).
GoogLeNet: GoogLeNet is a CNN that is designed to be more efficient than previous CNN architectures. It uses a technique called Inception modules to reduce the number of parameters and computations required for training (Szegedy et al. 2014).
ResNet: ResNets are a family of CNNs that use a technique called residual connections to improve the accuracy and efficiency of training. ResNets have achieved state-of-the-art results on a wide variety of computer vision tasks (He et al. 2015).
DenseNet: DenseNets are a family of CNNs that are based on the ResNet architecture. DenseNets use a technique called dense connections to further improve the accuracy and efficiency of training (Huang et al. 2016).

16.3.3 Evolution of CNN

CNN, or convolutional neural networks, have come a long way since their inception in the early 1980s. Today, they are one of the most powerful machine learning algorithms available, and they are used in a wide range of applications, including image recognition, natural language processing, and machine translation.

The evolution of CNNs can be traced back to the work of Yann LeCun, who developed the first successful CNN in 1989. LeCun's CNN was able to recognize handwritten digits with an accuracy that was unprecedented at the time (LeCun et al. 1989).

In the 1990s, CNNs fell out of favor due to the rise of support vector machines (SVMs). SVMs were able to achieve state-of-the-art results on a variety of tasks, and they were easier to train than CNNs.

However, in the early 2000s, CNNs began to make a comeback. This was due in part to the work of Geoffrey Hinton, who developed a new type of CNN called a deep belief network (DBN). DBNs were able to achieve better results on a variety of tasks than SVMs, and they were also easier to train.

In the late 2000s, CNNs began to be used for image recognition tasks. This was due in part to the work of AlexNet, which won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012. AlexNet was a CNN that was trained on a massive dataset of images, and it was able to achieve state-of-the-art results on a variety of image recognition tasks.

Since then, CNNs have continued to improve. In 2015, ResNet won the ILSVRC competition, and it achieved even better results than AlexNet. ResNet was a CNN that used a new type of residual block that made it easier to train deep networks.

In recent years, CNNs have been used for a wide range of tasks, including natural language processing, machine translation, and video recognition. CNNs have also been used to develop new types of applications, such as self-driving cars and facial recognition software.

The evolution of CNNs is a testament to the power of machine learning. CNNs have been able to achieve state-of-the-art results on a wide range of tasks, and they have revolutionized the way that we interact with computers.

16.3.4 Use Cases of CNN

CNNs are employed to recognize objects within images, including cars, pedestrians, and traffic signs. This technology finds utility in self-driving cars, facial recognition systems, and the analysis of medical images. They can be harnessed for the detection and tracking of objects in videos. This has practical applications in video surveillance systems, sports analytics, and the field of robotics.

CNNs are useful for segmenting images into different regions, distinguishing between foreground and background. This technology is used in medical image

analysis, object counting, and enhancing image editing capabilities. They are a valuable tool for tasks like text classification, sentiment analysis, and machine translation in the realm of natural language processing. They play a role in social media content filtering, spam detection, and powering customer service chatbots.

CNNs are instrumental in analyzing videos for purposes such as action detection, object tracking, and segmenting video scenes. These applications are prevalent in sports analytics, security systems, and advertising strategies.

16.4 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) represent a sophisticated class of neural networks engineered to recognize patterns in sequences of data. These models stand out for their ability to remember past information and use it to inform future predictions, making them ideal for applications in areas such as language translation, speech recognition, and forecasting events over time. Technologies like Apple's Siri and Google's Voice Search are notable examples of RNNs in action, showcasing their capability to process and interpret complex sequences of human speech.

16.4.1 Why We Need RNN—Overcoming the Limitations of Feed-Forward Network

The motivation behind the development of RNNs stems from the limitations inherent in traditional feed-forward neural networks. In a standard feed-forward architecture, information moves in a single direction: from input to output, passing through any hidden layers without looping back. This design lacks the mechanism to retain any memory of previous inputs, rendering it less effective for tasks that rely on understanding sequences or the temporal context of data. Key shortcomings include the inability to process sequential information inherently, only considering the current input without reference to its predecessors, and a lack of capability to maintain memory across inputs.

RNNs address these challenges by incorporating loops within their structure, allowing information to persist. This looping mechanism enables the network to hold onto information from previous inputs, thus providing a memory function that is crucial for sequential tasks. Each step in processing takes into account not just the current input but also the information gleaned from preceding steps. This is achieved by passing the output of a layer as an input to itself in future steps, effectively creating a memory of what has been learned previously (Graves et al. 2005) (Fig. 16.10).

RNNs thus are designed to process and make predictions on data where the sequence and context matter. By capturing temporal dependencies and remembering past inputs, RNNs offer a powerful tool for dealing with sequential data, such as

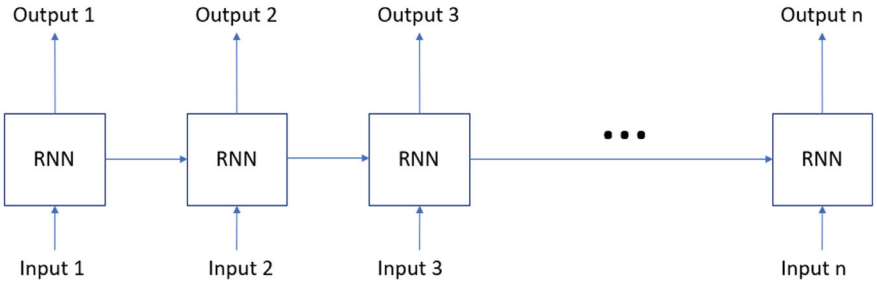
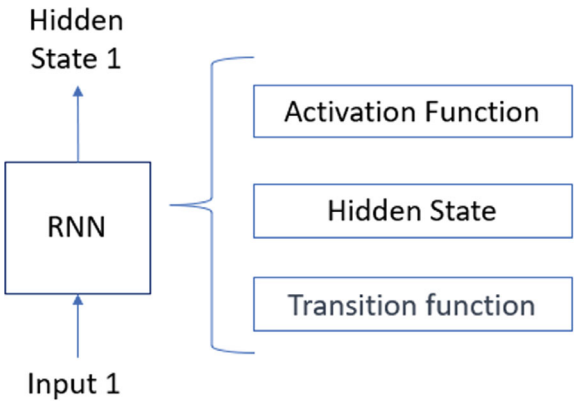


Fig. 16.10 Recurrent neural network

Fig. 16.11 Components of a RNN unit



predicting the next word in a sentence or understanding the trend in a time series. This ability to process sequences in their entirety, rather than in isolation, marks a significant advancement over traditional neural network models and underpins the creation and application of RNNs in various fields requiring nuanced understanding of sequential data.

Each unit of RNN will have an activation function, a hidden state, and a transition function (Fig. 16.11).

16.4.2 Hidden State

Each unit in the network has a hidden state that is updated based on both the current input and the previous hidden state. This makes the model capable of processing sequences of inputs—text, speech, or time series data. Hidden state is then used as input to compute the next hidden state and the network's output. Hidden state of an RNN can be thought of as a kind of memory that stores information from past inputs,

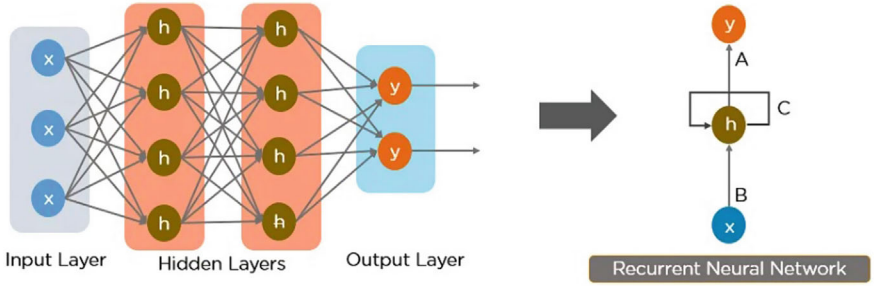


Fig. 16.12 Input, hidden, and output layers in a recurrent neural network

and the model uses this information to make predictions or decisions based on the current input.

Converting a Feed-Forward Neural Network into a Recurrent Neural Network (Goodfellow et al. 2016)

The nodes in different layers of the neural network are compressed to form a single layer of recurrent neural networks (Fig. 16.12).

Here, "*x*" is the input layer, "*h*" is the hidden layer, and "*y*" is the output layer. A, B, and C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at t) and $t - 1$. The output at any given time is fetched back to the network to improve on the output (Fig. 16.13).

16.4.3 Types of Recurrent Neural Networks

One to One: This type of RNN has a single input and a single output. It is also known as the Vanilla Neural Network and is used for general machine learning problems.

One to Many: This type of RNN has a single input and multiple outputs. An example is an image captioning model, which generates a caption for a given image.

Many to One: This type of RNN takes a sequence of inputs and generates a single output. An example is a sentiment analysis model, which classifies a given sentence as expressing positive or negative sentiment.

Many to Many: This type of RNN takes a sequence of inputs and generates a sequence of outputs. An example is a machine translation model, which translates a sentence from one language to another.

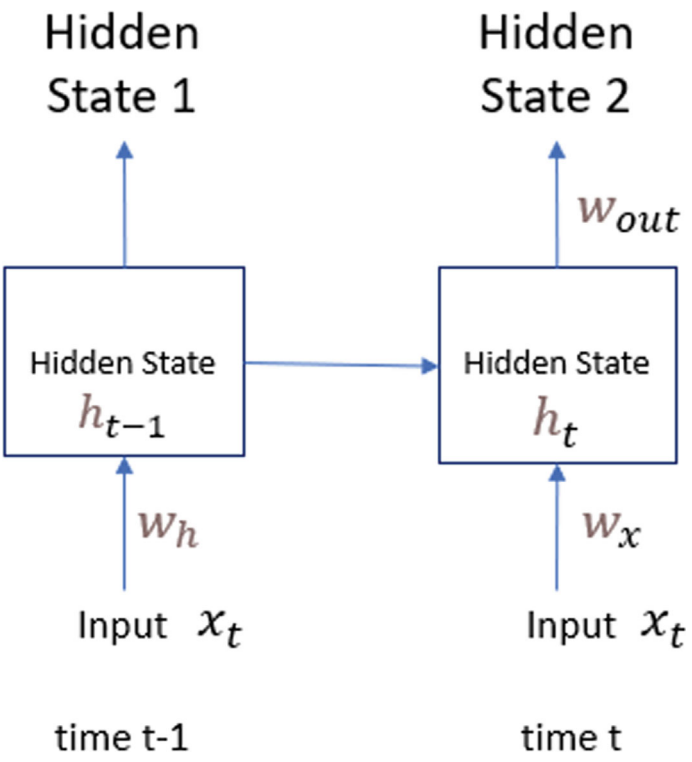


Fig. 16.13 Output in an RNN

16.4.4 Advantages of RNNs

RNNs bring several key advantages to the table, particularly when dealing with data that unfolds over time or sequences.

Unlike many other neural network architectures, RNNs can adeptly manage sequences that vary in length. This trait is invaluable for applications like understanding spoken words, parsing sentences, or analyzing trends in financial data, where the amount of information can change dramatically from one instance to the next.

One of the hallmark features of RNNs is their ability to remember information from previous inputs. This memory capability is crucial for understanding context or the sequence of events in tasks such as predicting the next word in a sentence or determining the sentiment of a paragraph.

By using the same parameters (weights and biases) at each step in processing a sequence, RNNs minimize the total number of parameters they need to learn. This not only simplifies the model but also helps it to generalize better from the training data to new, unseen data.

Thanks to non-linear activation functions, RNNs can learn intricate patterns beyond simple linear relationships. This allows them to tackle complex tasks that involve understanding the nuances and subtleties in data, such as distinguishing between different speakers' voices or interpreting the meaning behind a piece of text.

RNNs process data one step at a time in a sequential manner. This approach is not just computationally efficient; it also aligns naturally with how many types of data are structured, from spoken sentences to series of stock prices.

RNNs are incredibly versatile and can be tailored to a wide variety of tasks and data types. Whether it's text, audio, or even sequences of images, RNNs can be adapted to analyze and make predictions on the data.

In many fields, RNNs have achieved top-notch results, setting new benchmarks for accuracy in language translation, speech-to-text conversion, and more. Their unique structure makes them especially powerful for modeling sequences and predicting future events based on past data.

16.4.5 Two Issues of Standard RNNs (Pascanu et al. 2013)

Vanishing gradient issue and exploding gradient challenge complicate the training of RNNs, especially when dealing with sequences that require understanding information spread out over long intervals.

Vanishing Gradient Issue: In the context of standard RNNs, a notable challenge is the vanishing gradient problem. This issue arises when the gradients, which are essential for adjusting the network's parameters during training, shrink to very small values as they are propagated back through the network. Such diminutive gradients make it increasingly challenging for the RNN to adjust its parameters in a way that captures and learns from long-term dependencies present in the data.

Exploding Gradient Challenge: On the flip side, RNNs can also encounter the exploding gradient problem. This happens when the gradients become excessively large, leading to drastic updates to the network's parameters. This situation can result in an unstable training process, where the model's performance fluctuates wildly, making it hard to achieve or maintain good results.

16.4.6 Addressing the Issues of RNNs

To overcome the limitations of classic RNNs, several advanced architectures and techniques have been developed, each with unique features to tackle specific challenges. These advancements provide robust solutions to the inherent challenges of traditional RNNs, enhancing their applicability to a wider range of tasks that involve sequential data or require nuanced understanding of context over time.

Long Short-Term Memory (LSTM) Networks: LSTMs are designed to mitigate the vanishing gradient issue, incorporating a sophisticated system of gates (input, output, and forget gates) that regulate information flow. These gates enable LSTMs to retain or discard information selectively, making them adept at learning from long-term dependencies within the data. This selective memory feature is crucial for complex tasks where understanding historical context is essential.

Gated Recurrent Unit (GRU) Networks: Similar to LSTMs, GRUs offer a solution to the vanishing gradient problem but with a simpler architecture. They use two gates (reset and update gates) to manage information flow. These gates help the network decide which information to keep and which to discard as it processes each piece of input, providing a balance between memory retention and computational efficiency.

Bidirectional RNNs: Bi-RNNs enhance the model's understanding of context by processing data in both forward and backward directions. This dual-path processing allows the network to capture context from both the past and the future, enriching its understanding of the input sequence. Bidirectional RNNs are particularly useful in applications like speech recognition, where context from both directions can significantly influence interpretation.

Encoder–Decoder RNNs: This architecture is split into two parts: an encoder that condenses an input sequence into a context vector and a decoder that unfolds this vector into a new sequence. This setup is ideal for sequence-to-sequence tasks, such as translating text from one language to another, by providing a framework for handling sequences of variable lengths in both input and output.

Attention Mechanisms: To further refine the model's focus on relevant parts of the input sequence, attention mechanisms enable the network to weigh parts of the sequence differently. This approach helps the model to concentrate on the most informative parts of the input, improving its performance on tasks with long sequences or where specific segments of the data are more critical for the task at hand.

16.4.7 Evolution of RNNs

Early RNNs: The first RNNs were developed in the 1980s, but they were not widely used due to the vanishing gradient problem.

Long Short-Term Memory (LSTM) Networks: In 1997, Hochreiter and Schmid-Huber proposed a new type of RNN called the Long Short-Term Memory (LSTM) network. LSTMs can overcome the vanishing gradient problem by using a special gating mechanism to control the flow of information through the network (Graves et al. 2005).

Bidirectional RNNs: In 1997, Schuster and Paliwal proposed a type of RNN called the bidirectional RNN. Bidirectional RNNs can process input sequences in both the forward and backward directions, which allows them to capture information about the context of the sequence (Schuster and Paliwal 1997).

Gated Recurrent Unit (GRU) Networks: In 2014, Cho et al. proposed another type of RNN called the gated recurrent unit (GRU) network. GRUs are like LSTMs, but they are simpler and more efficient to train.

Encoder–Decoder RNNs: In 2014, encoder–decoder RNNs were found which are typically used for sequence-to-sequence tasks, such as machine translation.

Attention Mechanisms: In 2014, Bahdanau et al. proposed a technique called the attention mechanism. Attention mechanisms allow RNNs to focus on specific parts of the input sequence, which can improve the performance of RNNs on long-sequence tasks.

In recent years, there have been many advances in RNN technology such as transformers and GAN architectures.

16.4.8 Use Cases of RNNs

RNNs are used in NLP for tasks such as language modeling, machine translation, and text summarization. They are used in speech recognition to transcribe speech to text. RNNs are also helpful in time series analysis to forecast future values of a time series, such as stock prices or weather patterns.

The application of RNNs extends to image captioning and generating descriptions of images. Recent application of RNNs includes music generation. Many state-of-the-art technologies today including Google Translate, Amazon Alexa, Apple Siri, Netflix recommendations, and stock market predictions use RNNs.

References

Goodfellow I, Bengio Y, Courville A (2016) Feed-forward neural networks vs recurrent neural networks: a comparative review. In: Deep learning. MIT Press, pp 194–201

Graves A, SchmidHuber A, Hochreiter S (2005) Long short-term memory. Neural Comput 17(9):1853–1909

He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778

Hopfield JJ (1982) Neural networks and Boltzmann machines. Proc Natl Acad Sci 79(8):2554–2558

Huang G, Liu Z, van der Maaten L, Weinberger KQ (2016) Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4700–4708

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Adv Neural Inf Process Syst 25:1097–1105

LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to hand-written zip code recognition. Neural Comput 1(4):541–551

McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133

Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. In: Proceedings of the 30th international conference on machine learning (ICML), pp 1310–1318

Rosenblatt F (1957) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386–408

Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Neur Netw 8(1):148–153

Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2014) Going deeper with convolutions. arXiv preprint arXiv:1409.4842

Werbos PJ (1974) Beyond regression: new tools for prediction and analysis. Unpublished doctoral dissertation, Harvard University.

Werbos PJ (1994) The roots of backpropagation: from McCulloch and Pitts to Hopfield and Hinton (1986–1990). IEEE Control Syst Mag 14(5):30–44