DTS 201: Introduction to Data Science (3 Units C: LH 30; PH 45)

Fundamentals of Data Science. Methodology of extracting knowledge from big datasets as well as various tools and platforms for Data Science. What is Data and why is it important? Basic classification of Data (Structured, semi-structured and unstructured data), Scope of Data Science, Steps of Data Science Process: Data collection, Pre-processing, training, and testing. Rudiments of data visualisations; Distributions, Probability, and Simulations; Predictions and Models. Use cases in various domains such as Image, Natural Language, Audio and Video. Basic introduction to knowledge extraction: Data mining, Business Intelligence & Knowledge management, Introduction to Big Data integration and intelligence, Introduction to Data Analytics, Introduction to programming.

Lab work: Practical experiments on data science process steps in simulated models. Practical application of the methods and tools used in data science for prediction models with some simulated exercises. Practical experiments on how to extract knowledge; how to mine valuable data from large set of data sets using data mining process and methods. Learn how to integrate business intelligence in big data along with some data analytics practical exercises. Simple exercises on R programming to enhance the coding knowledge acquired during theory class.

Learning Outcomes

At the end of the course, the students should be able to:

1. demonstrate the principles of working with data across distributions, sizes and ranges;

2. explain from first principles the operations that power data-driven utilities that have transformed the modern computing industry; and

3. demonstrate foundational technological processes that enable various data functions.

Text Book(s):

1. Jugal K. Kalita, Dhruba K. Bhattacharyya and Swarup Roy (2024). Fundamentals of Data Science: Theory and Practice. Academic Press, Elsevier.

Course Writer/Developer: Dr. Femi E. AYO, Department of Computer Sciences, Olabisi Onabanjo University, Ago-Iwoye.

E-mail: ayo.femi@oouagoiwoye.edu.ng

Lesson 1: Fundamentals of Data Science

Introduction

Consumer satisfaction is a fundamental performance indicator and a key element of an enterprise's success. The success of any enterprise relies on its understanding of customer expectations and needs, buying behaviors, and levels of satisfaction. Modern giant business houses analyze customer expectations and perceptions of the quality and value of products to make effective decisions regarding product launch and update, servicing, and marketing.

Due to the availability of fast internet technologies and low-cost storage devices, it has become convenient to capture voluminous amounts of consumer opinions and records of consumer activities. However, discovering meaningful consumer feedback from a sea of heterogeneous sources of reviews and activity records is just like finding a needle in a haystack. Data Science appears to be the savior in isolating novel, unknown, and meaningful information that helps proper decision making. Data Science is an interdisciplinary field that involves the extraction of meaningful insights from complex and structured as well as unstructured data.

Data Science is the study of methods for programming computers to process, analyze, and summarize data from various perspectives to gain revealing and impactful insights and solve a vast array of problems. It is able to answer questions that are difficult to address through simple database queries and reporting techniques. Data Science aims to address many of the same research questions as statistics and psychology, but with differences in emphasis. Data Science is primarily concerned with the development, accuracy, and effectiveness of the resulting computer systems. Statistics seek to understand the phenomena that generate the data, often with the goal of testing different hypotheses about the phenomena. Psychological studies aspire to understand the mechanisms underlying the behaviors exhibited by people such as concept learning, skill acquisition, and strategy change.

Google Maps is a brilliant product developed by Google, Inc., using Data Science to facilitate easy navigation. But how does it work? It collects location data continuously from different reliable heterogeneous sources, including GPS locations via mobile phones of millions of users who keep their location services on. It captures location, velocity, and itinerary-related data automatically. Efficient Data Science algorithms are applied to the collected data to predict traffic jams and road hazards, the shortest routes, and the time to reach the destination. Massive quantities of collected past, current, and near current traffic data help Google predict real-time traffic patterns.

As a new discipline, data science is based on two essential properties: the diversity of data and the universality of data research. In modern society, data pervades all walks of life. There are many types of data, including structured data and unstructured data, such as web pages, text, image, video and audio.

Data science mainly consists of two aspects: using data to study scientific problems and using scientific methods to study data. The former includes bioinformatics, astroinformatics, digital earth, etc.; and the latter includes statistics, machine learning, data mining, database, etc. These are all essential to data science and when put together, they form the overall picture of data science.

1.1 Data, information, and knowledge

To introduce the arena of Data Science, it is of utmost importance to understand the data processing stack. Data Science-related processing starts with a collection of raw data. Any facts about events that are unprocessed and unorganized are called data. Generally, data are received raw and hardly convey any meaning. Data, in their original form, are useless until processed

further to extract hidden meaning. Data can be (i) operational or transactional data such as customer orders, inventory levels, and financial transactions, (ii) nonoperational data, such as market-research data, customer demographics, and financial forecasting, (iii) heterogeneous data of different structures, types, formats such as MR images and clinical observations, and (iv) metadata, i.e., data about the data, such as logical database designs or data dictionary definitions.

Information is the outcome of processing raw data in a meaningful way to obtain summaries of interest. To extract information from data, one has to categorize, contextualize, and condense data. For example, information may indicate a trend in sales for a given period of time, or it may represent a buying pattern for customers in a certain place during a season. With rapid developments in computer and communication technologies, the transformation of data into information has become easier. In a true sense, Data Science digs into the raw data to explore hidden patterns and novel insights from the data.

Knowledge represents the human understanding of a subject matter, gained through systematic analysis and experience. Knowledge results from an integration of human perspectives and processes to derive meaningful conclusions. Some researchers define knowledge with reference to a subject matter from three perspectives, (i) understanding (know-why), (ii) cognition or recognition (know-what), and (iii) capacity to act (know-how). Knowledge in humans can be stored only in brains, not in any other media. The brain has the ability to interconnect it all together. Unlike human beings, computers are not capable of understanding what they process, and they cannot make independent decisions. Hence, computers are not artificial brains! While building knowledge, our brain is dependent on two sources, i.e., data and information. To understand the relationship between data and information, consider an example. If you take a photograph of your house, the raw image is an example of data. However, details of how the house looks in terms of attributes such as the number of stories, the colors of the walls, and its apparent size, constitute information. If you send your photograph via email or message to your friend, you are actually not sending your house or its description to your friend. From the photograph, it is up to your friend, how he/she perceives its appearance or looks. If it so happens that the image is corrupted or lost, still your original house will be retained. Hence, even if the information is destroyed, the data source remains.

The key concepts of data, information, and knowledge are often illustrated as a pyramid, where data are the starting point placed at the base of the pyramid (see Figure 1.1), and it ends in knowledge generation. If we collect knowledge from related concepts, domains, and processes further, it gives rise to wisdom. We skip discussions on wisdom as the concept is highly abstract and controversial and difficult to describe. Usually, the sizes of repositories to store data, information, and knowledge become smaller as we move upward in the pyramid, where data in their original form have lower importance than information and knowledge. It is worth mentioning that quality raw data lead to more significant information and knowledge generation. Hence, good-quality data collection is a stepping stone for effective information and knowledge mining.

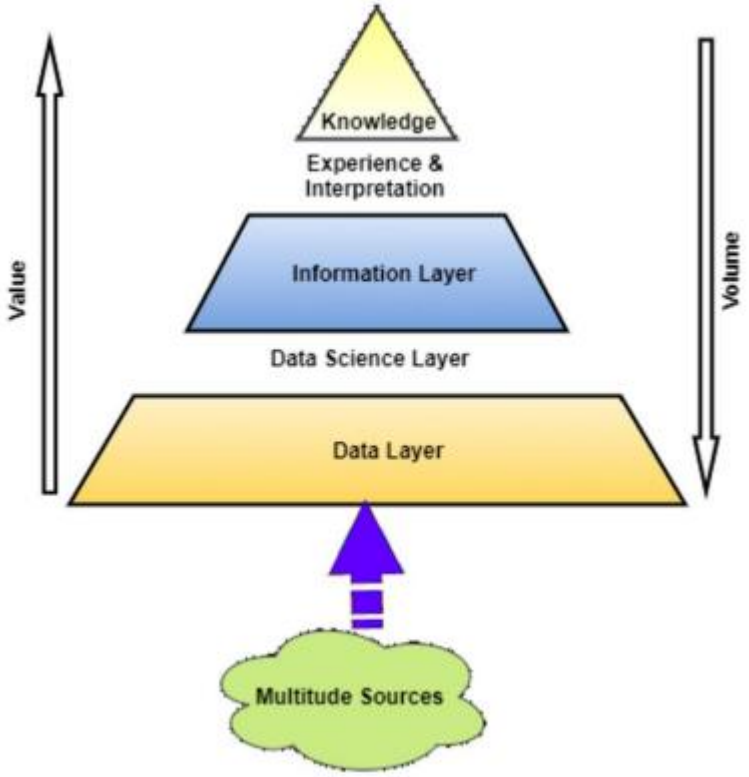


Figure 1.1 Data, Information, and Knowledge pyramid, and intermediate conversion layers. The directions of the arrowheads indicate increase in size and importance.

1.2 Data Science: the art of data exploration

Data Science is a multifaceted and multidisciplinary domain dedicated to extracting novel and relevant patterns hidden inside data. It encompasses mathematical and statistical models, efficient algorithms, high-performance computing systems, and systematic processes to dig inside structured or unstructured data to explore and extract nontrivial and actionable knowledge, ultimately being useful in the real world and having an impact.

The success of Data Science depends on many factors. The bulk of efforts have concentrated on developing effective exploratory algorithms. It usually involves mathematical theories and expensive computations to apply the theory to large-scale raw data.

1.2.1 Brief history

The dawn of the 21st century is known as the Age of Data. Data have become the new fuel for almost every organization as references to data have infiltrated the vernacular of various communities, both in industry and academia. Many data-driven applications have become amazingly successful, assisted by research in Data Science. Although Data Science has become a buzzword recently, its roots are more than half a century old. In 1962, John Wilder Tukey, a famous American mathematician published an article entitled, The Future Of Data Analysis that sought to establish a science focused on learning from data. After six years, another pioneer named Peter Naur, a Danish computer scientist introduced the term Datalogy as the science of data and of data processes, followed by the publication of a book in 1974, Concise Survey of Computer Methods, that defined the term Data Science as the science of dealing with data. Later, in 1977, The International Association for Statistical Computing (IASC) was founded with a plan for linking traditional statistical methodology, modern computer technology, and the knowledge of domain experts in order to convert data into information and knowledge. Tukey also published a major work entitled, Exploratory Data Analysis, that laid an emphasis

on hypothesis testing during data analysis, giving rise to the term data-driven discovery. Following this, the first Knowledge Discovery in Databases (KDD) workshop was organized in 1989, becoming the annual ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD).

Later, in 1996, Fayyad et al. introduced the term Data Mining, the application of specific algorithms for extracting patterns from data. By the dawn of the 2000s, many journals started recognizing the field and notable figures like William S. Cleveland, John Chambers, and Leo Breiman expanded boundaries of statistical modeling, envisioning a new epoch in statistics focused on Data Science.

The term Data Scientist was first introduced in 2008 by Dhanurjay Patil and Jeff Hammerbacher of LinkedIn and Facebook.

1.2.2 Methodology of extracting knowledge from big datasets

Data Science espouses a series of systematic steps for converting data into information in the form of patterns or decisions. Data Science has evolved by borrowing concepts from statistics, machine learning, artificial intelligence, and database systems to support the automatic discovery of interesting patterns in large datasets. A Data Science methodology of extracting knowledge from big datasets or data science pipeline is made of the following four major phases. An illustrative representation of a typical Data Science workflow is depicted in Figure 1.2.

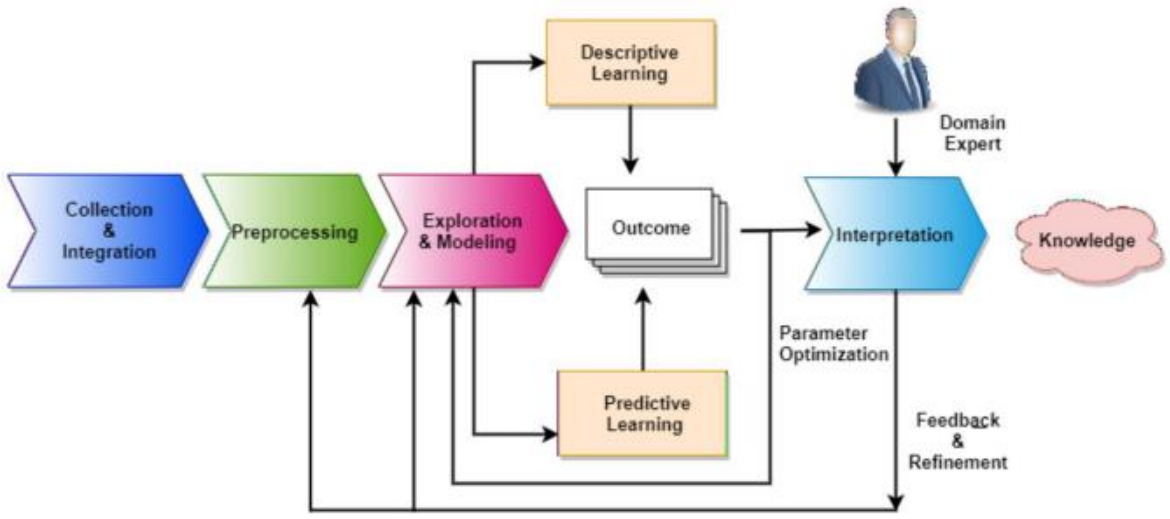


Figure 1.2 Major steps in Data Science pipeline for decision making and analysis.

1.2.2.1 Data collection and integration

Data are initially collected, and integrated if collection involves multiple sources. For any successful Data Science and -analysis activity, data collection is one of the most important steps. The quality of collected data carries great weight. If the collected samples are not sufficient to describe the overall system or process under study, downstream activities are likely to become useless despite employing sophisticated computing methods. The quality of the outcome is highly dependent on the quality of data collection.

It has been observed that dependence on a single source of data is always precarious. Integration of multifaceted and multimodal data may offer better results than working with a single source of information. In fact, information from one source may complement those from other sources when one source of data is not sufficient to understand a system or process well.

However, the integration of multisource data itself is a challenging task and needs due attention. Integration should be deliberate rather than random mixing to deliver better results.

1.2.2.2 Data preparation

Raw data collected from input sources are not always suitable for downstream exploration. The presence of noise and missing values, and the prevalence of nonuniform data structures and standards may negatively affect final decision making. Hence, it is of utmost importance to prepare the raw data before downstream processing. Preprocessing also filters uninformative or possibly misleading values such as outliers.

1.2.2.3 Learning-model construction

Different machine learning models are suitable for learning different types of data patterns. Iterative learning via refinement is often more successful in understanding data distributions. A plethora of models are available to a data scientist and choices must be made judiciously. Models are usually used to explain the data or extract relevant patterns to describe the data or predict associations.

1.2.2.4 Knowledge interpretation and presentation

Finally, results need to be interpreted and explained by domain experts. Each step of analysis may trigger corrections or refinements that are applied to the preceding steps.

1.2.3 Multidisciplinary science

Data Science is a multidisciplinary domain of study to extract, identify, and analyze novel knowledge from raw data by applying computing and statistical tools, together with domain experts for the interpretation of outcomes. It involves mathematical and statistical tools for effective data analysis and modeling, pattern recognition and machine learning to assist in decision making, data and text mining for hidden pattern extraction, and database technologies for effective large data storage and management (Figure 1.3). Due to the complex nature of the data elements and their relationships, most often, understanding the data itself is challenging. Before understanding the underlying distribution of data elements, it may not be very fruitful to apply any statistical or computational tools for knowledge extraction. Visualization may play a large role in deciphering the interrelationships among the data elements, thereby helping decide the appropriate computational models or tool for subsequent data analysis. The presence of noise in the data may be discovered and eliminated by looking into distribution plots. However, it is a well-known fact that visualizing multidimensional data itself is challenging and needs special attention. With the availability of low-cost data-generation devices and fast communication technologies, “Big Data” or vast amount of data have become ubiquitous. Dealing with Big Data for Data Science needs high-performance computing platforms. The science and engineering of parallel and distributed computing are important discipline that need to be integrated into the Data Science ecosystem. Recently, it has become convenient to integrate parallel computing due to the wide availability of relatively inexpensive Graphical Processing Units (GPU). Last but not least, knowledge of and expertise in the domain in which Data Science approaches are applied play major roles during problem formulation and interpretation of solutions.

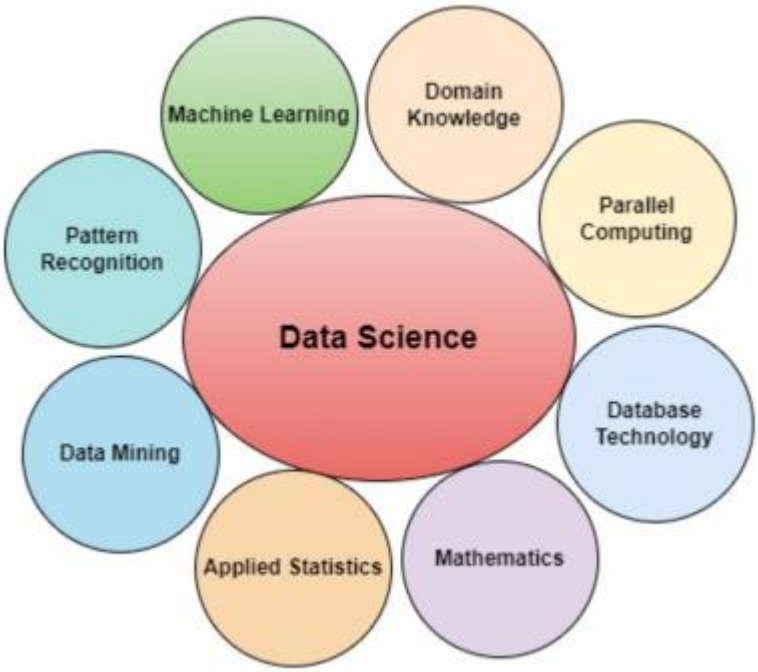


Figure 1.3 Data Science joins hands with a variety of other disciplines.

1.3 What is not Data Science?

In recent years, the term Data Science has become a buzzword in the world of business and intelligent computing. As usual, high demand and popularity invite misinterpretation and hype. It is important to be aware of the terms that are used as well as misused in the context of Data Science.

Machine Learning is not a branch of Data Science. It provides the technology or the tools to facilitate smart decision making using the software. Data Science uses Machine Learning as a tool for autonomous pattern analysis and decision making.

There is a prevalent fallacy that techniques of Data Science are applicable to only a very large amounts of data or so-called Big Data. This is not true, as even smaller amounts of data can also be analyzed usefully. The quality of the data in hand and the completeness of the data are always important. It is true that a Machine Learning system is likely to be able to extract more accurate knowledge when large amounts of relevant data are used to draw intuitions about the underlying patterns.

It is true that statistical techniques play a great role in effective data analysis. Statistics complements and enhances Data Science for efficient and effective analysis of large collections of data. Statistics use mathematical models to infer and analyze data patterns by studying data distributions of collected samples from the past. However, Data Science cannot be considered as completely dependent on statistics alone. Statistics is used mostly to describe past data, whereas Data Science performs predictive learning for actionable decision making. A number of nonparametric Data Science learning models help understand the data very well without knowing the underlying data distributions.

Last but not least, people often give more importance to scripting languages, such as Python or R, and tools available for ready use rather than understanding the theory of Data Science. Of course, knowledge of tools greatly helps in developing intelligent systems quickly and effectively. Without understanding the models and formalisms, quite often many users concentrate on using the readily and freely available preimplemented models. Knowing only

prescripted or programmed tools is not sufficient to have a good overall understanding of Data Science so that existing tools can be adapted and used efficiently to solve complex problems. Proficiency in data-analysis tools without deeper knowledge of data analysis does not make for a good data scientist.

1.4 Data Science tasks

Data Science-related activities are broadly classified into predictive and descriptive tasks. The former deals with novel inferences based on acquired knowledge and the latter describes the inherent patterns hidden inside data. With the rise in business-analysis applications, the span of Data Science tasks has extended further into two related tasks, namely diagnostic and prescriptive. Somewhat simplistic, but differentiating, views of the four tasks can be obtained by asking four different questions: “What is likely to happen?” (Predictive), “What is happening?” (Descriptive), “Why is it happening?” (Diagnostic) and “What do I need to do?” (Prescriptive), respectively.

1.4.1 Predictive Data Science

Predictive tasks apply supervised Machine Learning to predict the future by learning from past experiences. Examples of predictive analysis are classification, regression, and deviation detection. Some predictive techniques are presented below. **Classification** attempts to assign a given instance into one of several prespecified classes based on behaviors and correlations of collected and labeled samples with the target class. A classifier is designed based on patterns in existing data samples (training data). The trained model is then used for inferring the class of unknown samples. The overall objective of any good classification technique is to learn from the training samples and to build accurate descriptions for each class. For example, spam filtering separates incoming emails into safe and suspicious emails based on the signatures or attributes of the email. Similar to classification, **prediction** techniques infer the future state based on experiences from the past. The prime difference between classification and prediction models is in the type of outcome they produce. Classification assigns each sample to one of several prespecified classes. In contrast, prediction outcomes are continuous valued as prediction scores. The creation of predictive models is similar to classification. The prediction of the next day’s or next week’s weather or temperatures is a classic example of a prediction task based on observations of the patterns of weather for the last several years in addition to current conditions. **Time-series data analysis** predicts future trends in time-series data to find regularities, similar sequences or subsequences, sequential patterns, periodicities, trends, and deviations. For example, predicting trends in the stock values for a company based on its stock history, business situation, competitor performance, and current market.

1.4.2 Descriptive Data Science

Descriptive analysis is also termed exploratory data analysis. Unlike predictive models, descriptive models avoid inference, but analyze historical or past data at hand and present the data in a more interpretable way such that it may better convey the underlying configuration of the data elements. Leveraging powerful visualization techniques usually enhances descriptive analysis for better interpretation of data. A descriptive task may act as the starting point to prepare data for further downstream analysis. Clustering, association, and sequence mining are some descriptive analysis techniques.

Clustering is an unsupervised technique that groups data into meaningful clusters such that data within a cluster possess high similarity, but low similarity with data in other clusters. A similarity or distance measure is needed to decide the inclusion of a data sample in a particular cluster. The basic difference between classification and clustering is that classification assumes prior knowledge of class labels assigned to data examples, whereas clustering does not make

such an assumption. For example, clustering can be used to separate credit-card holders into groups, such that each group can be treated with a different strategy. **Association-rule mining** finds interesting associations among data elements. The market-basket analysis is the initial application where association rule mining was extensively used. Such market-basket analysis deals with studying the buying behaviors of customers. It can be used by business houses in creating better marketing strategies, logistics plans, inventory management, and business promotions. **Summarization** is an effective data-abstraction technique for the concise and comprehensive representation of raw data. With summarization, data may become better interpretable because irrelevant details are removed. Text summarization is a popular technique for generating a short, coherent description of a large text document.

1.4.3 Diagnostic Data Science

The diagnostic analysis builds on the outcomes of descriptive analysis to investigate the root causes of a problem. It includes processes such as data discovery, data mining, drilling down, and computing data correlations to identify potential sources of data anomalies. Probability theory, regression analysis, and time-series analysis are key tools for diagnostic analysis. For example, Amazon can drill down the sales and profit numbers to various product categories to identify why sales have gone down drastically in a certain span of time in certain markets.

1.4.4 Prescriptive Data Science

This is the most recent addition to analysis tasks. A prescriptive model suggests guidelines or a follow-up course of action to achieve a goal. It uses an understanding of what has happened, why it has happened, and what might happen to suggest the best possible alternatives. Unlike predictive analysis, prescriptive analysis does not need to be perfect but suggests the “best” possible way toward the future. Google Maps is an excellent prescriptive model that suggests the best route from the current location to the destination on the fly, considering traffic conditions and the shortest route.

1.5 Data Science objectives

Data Science aims to achieve four basic objectives: (i) to extract interesting patterns in data without human intervention, (ii) to predict the most likely future outcomes based on past data, (iii) to create actionable knowledge to achieve certain goals, and (iv) to focus on how to handle voluminous data in achieving the previous three objectives.

1.5.1 Hidden knowledge discovery

To discover hidden knowledge, Data Science builds models. A model is built using an algorithm that operates on a dataset. It is a computational structure that learned from collected and pre-processed data and used to solve data analytics tasks with possibly previously unseen data. To facilitate the automatic extraction of hidden patterns, the data scientist executes appropriate data-mining models. Data Science models are typically used to explore the types of data for which they are built. Many types of models can be adapted to new data.

1.5.2 Prediction of likely outcomes

One way Data Science can help predict likely outcomes is by generating rules, which are in antecedent-consequent form. Data scientists use statistical and machine-learning models to analyze data and identify patterns, trends, and relationships that can be used to predict future outcomes. The goal is to use data to gain insights and make accurate predictions that can help businesses and organizations make better decisions. The ability to make accurate predictions is critical for businesses and organizations to stay competitive and make informed decisions. Data Science plays a vital role in enabling organizations to leverage their data to gain insights

and make better predictions about future outcomes. Identifying potential fraudulent activities by analyzing patterns and anomalies in financial transactions or predicting future sales based on historical sales data, customer demographics, and other factors are some examples of prediction outcomes.

1.5.3 Grouping

Another major objective of Data Science is to obtain natural groupings in data to extract interesting patterns. An example is finding a group of employees in an organization that is likely to benefit from certain types of investments based on income, age, years of work, temperament, and desired investment goals.

1.5.4 Actionable information

Data Science can handle voluminous data, popularly termed Big Data (as alluded to earlier), and can extract relevant information, which can be of direct help in the decision-making process. For example, a bank may use a predictive model to identify groups of clients with a net worth above a predefined threshold such that they are likely to be receptive to investing large amounts of money in certain high-risk, high-reward business initiatives being proposed.

1.6 Applications of Data Science

Due to low-cost high-performance computing devices, superfast internet technologies, and ample cloud storage, it is now possible for business houses to rely on intelligent decision making based on large amounts of data. There is hardly any reputable organization in the world that is not leveraging Data Science for smart decision making. In a nutshell, Data Science is concerned with two broad endeavors. The first is the effort to make devices or systems smart and the other involves deciphering data generated by natural or engineered systems and learning from them. A few of the many potential application domains are introduced next.

• **Healthcare:** Recent advances in Data Science are a blessing to healthcare and allied sectors. Data Science has positively and extensively impacted upon these application areas, in turn helping mankind significantly. Health informatics and smart biomedical devices are pushing medical and health sciences to the next level. Precision medicine is likely to be a game changer in extending the human life span. Computer-vision and biomedical technologies are making quick and accurate disease diagnosis ubiquitous. Even a handheld smartphone may be able to continuously monitor the health metrics of a person and generate smart early alarms when things go wrong.

• **Computational Biology:** With the availability of high-throughput omics data, it is now possible to understand the genetic causes of many terminal diseases. Computational biology and bioinformatics mine massive amounts of genomic and proteomic data to better understand the causes of diseases. Once the causes are identified with precision, it becomes possible to develop appropriate drug molecules for treatment. On average, traditional drug development requires more than 14 years of effort, which can now be reduced drastically due to effective Data Science techniques. Precision medicine is the future of drug technology, customizing drugs for the individual and avoiding the one-size-fits-all approach that does not always work.

• **Business:** The rise of Data Science was originally intended to benefit business sectors. With the need for business intelligence, the use of data analytics has gained momentum. Almost every business venture is investing significant resources in smart business decision making with Data Science. Analyzing customer purchasing behavior is a great challenge, and important for improving revenue and profit. Integration of heterogeneous data is an effective way to promote sales of products. Data-analysis experts apply statistical and visualization tools to

understand the moods, desires, and wants of customers and suggest effective ways for business and product promotion and plans for progress and expansion.

• **Smart Devices**: A mobile device is no longer just a communication device, but rather a miniature multipurpose smart tool that enhances the lifestyle of the common man or woman. Apps installed in smart devices like smartphones or smart tablets can be used to understand a person well in regard to their choices, preferences, likes, and dislikes. Technology is becoming so personalized that installed apps in such devices will be able to predict a user's actions in advance and make appropriate recommendations. In the near future, smart devices will be able to monitor user health status, recommend doctors, book appointments, place orders for medicine, and remind users of medicine schedules. The Internet-of-Things (IoT) and speaking smart devices make our life easier. For example, a home automation system can monitor and control an entire home remotely with the help of a smartphone, starting from kitchen to home security.

• **Transportation:** Another important application area is smart transportation. Data Science is actively involved in making it possible to take impressive steps toward safe and secure driving. The driverless car will be a big leap into the future in the automobile sector. The ability to analyze fuel-consumption patterns and driving styles and monitor vehicle status makes it possible to create optimized individual driving experiences, spurring new designs for cars by manufacturers. A transportation company like Uber uses Data Science for price optimization and offers better riding experiences to customers.

In addition to the above, the list of applications is huge and is growing every day. There are other industry sectors like banking, finance, manufacturing, e-commerce, internet, gaming, and education that also use Data Science extensively.

Lesson 2: Data, sources, and generation

2.1 Introduction

The modern world generates enormous amounts of data almost every day. According to one source, 2.5 quintillion bytes of data are being created every day. It is expected that by the end of 2023, more than 100 zettabytes of data will need to be stored and handled across the world. Such massive amounts of data are generated primarily due to widespread access to the World Wide Web and social-media platforms. Facebook generates 4 petabytes of data per day. A single person can create up to 1.7 MB of data every second. Google processes over 40 000 searches every second, or 3.5 billion searches daily. Through video platforms such as Zoom, 3.3 trillion minutes are spent in meetings yearly. One hour of a meeting can create data ranging from 800 MB to 2.47 GB if recorded in video.

It is of utmost importance to dig into such large volumes of data to extract hidden novel and nontrivial patterns in the form of knowledge. Often, this process of knowledge discovery is termed Knowledge Discovery in Databases (KDD), and the intermediate step toward KDD that requires extraction of hidden patterns is popularly called Data Mining. Mining such a large volume of data necessitates efficient storage and processing. Many types, shapes, and formats of data must be ultimately stored in binary form to be processed by computing devices.

Below, we discuss the basic data types or attributes used for storage and processing.

2.2 Data attributes

The study and use of programming languages or database-management systems require dealing with data types or data attributes. A data attribute defines a data element using a single descriptor with specific characteristics. In statistics, it is also called a measurement scale. The way of handling each attribute varies with its nature. Hence, it is a necessary as well as important to understand attribute types (Figure 2.1) before processing. Below, we discuss major attribute types or variables with examples to facilitate understanding.

2.2.1 Qualitative

As the name suggests, qualitative attributes are data types that cannot be measured using standard measuring units; rather, they can be observed, compared, and classified.

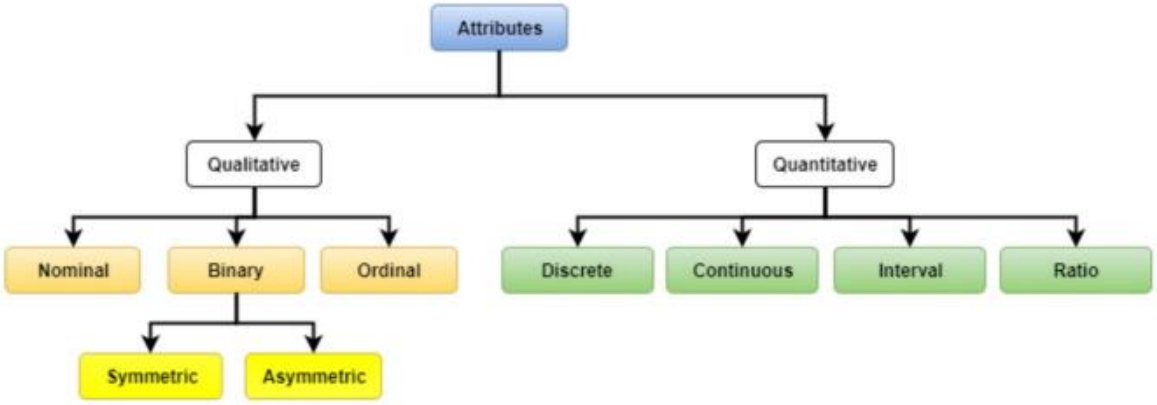


Figure 2.1 Attribute types in databases.

2.2.1.1 Nominal

A nominal attribute represents a specific symbol or the name of an entity; it is also called a categorical attribute. It is a qualitative attribute that cannot be ordered or ranked. No meaningful relationship can be established among the values. Only the logical or "is equal" operator can be applied among the values. Categorical values cannot be dealt with directly for various Data Science tasks. For example, neural networks are not built to handle categorical data straightforwardly. A few examples of nominal attributes are given in Table 2.1.

Table 2.1 Examples of nominal attributes and values.

Attribute	Values
Color	Red, Blue, Green
City	Moscow, Kolkata, New York
Name	John, Harry, Ron

2.2.1.2 Binary

A binary attribute holds discrete values with exactly two mutually exclusive states, either 1 (true) or 0 (false). A person is either infected by a virus or not infected. Both states never occur at the same time.

A binary attribute is said to be a symmetric binary attribute if both states possess equal importance. In such a case, it hardly matters which attribute value, such as 0 or 1, to assign. On the other hand, an asymmetric binary attribute does not give equal importance to both states. Table 2.2 gives a few examples of binary attributes.

Table 2.2 Examples of binary attribute types.

Attribute	Values	Type
Gender	Male, Female	Symmetric
Status	Present, Absent	Asymmetric

2.2.1.3 Ordinal

Attribute values that can be ordered or ranked are called ordinal attributes. It is worth mentioning that magnitude-wise, the difference between ordered values may not be known or may not even be significant. The ordering exhibits the importance of the attribute value but does not indicate how important. For example, the allowable values for an attribute may be Good, Better, and Best. One may rank the values based on their importance without being clear about how Good differs from Better or Best. A few more examples are listed in Table 2.3.

Table 2.3 Examples of ordinal attribute and values.

Attribute	Values
Grade	A+, A, B+, B, C+, C
Position	Asst. Prof., Assoc. Prof., Prof.
Qualification	UG, PG, PhD

2.2.2 Quantitative

Unlike Qualitative attributes, Quantitative attributes are measurable quantities. The values are quantified using straightforward numbers. Hence, they are also called numerical attributes. Statistical tools are easily applicable to such types of attributes for analysis. They can also be represented by data-visualization tools like pie charts, line and bar graphs, scatter plots, etc. The values may be countable or noncountable. Accordingly, they are of the two following types: Table 2.4 shows some quantitative attributes.

2.2.2.1 Discrete

Countable quantitative attributes are also called discrete attributes. Usually, if the plural of the attribute name may be prefixed with “the number of,” it can be considered a discrete value attribute. Examples of discrete attributes includes the number of persons, cars, buildings, population of a city, etc. They are denoted with whole numbers or integer values and can’t have fraction.

2.2.2.2 Continuous

Continuous values are noncountable and measurable quantities with infinite possible states. They are represented with fractional or floating-point values. Attributes such as Weight, Height, Price, Blood Pressure, Temperature, etc., hold continuous values.

Whether discrete or continuous type, quantitative attributes can be of the following two types.

2.2.2.3 Interval

Like ordinal attributes, quantitative attributes (both discrete and continuous) can be ordered. The distinction comes from whether differences or intervals among the ordered values are known or not. Interval values do not have any correct references or true zero points. The zero point is the value at which no quantity or amount of the attribute is measured. In other words, it is the point on the scale that represents the absence of the attribute. Values can be added and subtracted from the list of attribute values, but individual values cannot be multiplied or divided. For example, temperature intervals (Table 2.4), such as 10◦C to 20◦C, represents a

range of values within which temperature can vary. The temperatures for two consecutive days may differ by a certain degree, but we cannot say there is “no temperature” (zero-point).

2.2.2.4 Ratio

Ratio is another quantitative attribute representing a relative or comparative quantity relating to two or more values. Unlike intervals, ratios have fixed zero points. A fixed zero point is a reference point on a fixed scale that does not vary based on the measured attribute. In other words, it is a point on the scale representing a constant value, regardless of the quantity or amount of the attribute being measured. Ratio-scale attributes can also be ordered, but with a perfect zero. However, no negative value is allowed in ratio-scale attributes. If a value is ratio scaled, it implies a multiple (or ratio) of another value. Parametric measures such as mean, median, mode, and quantile range are considered ratio-scale attributes. Family Dependents are considered Ratio data because they possess all the properties of interval data and, in addition, have a meaningful zero point. As given in Table 2.4, Family Dependents are considered Ratio data because they possess all the properties of interval data and, in addition, have a meaningful zero point. In this case, a Family Dependents value of 0 indicates the absence of family dependents, which represents a true absence and not just an arbitrary point on a scale. We can say that having 3 dependents is 50% more than having 2 dependents, or having 2 dependents is twice as many as having 1 dependent. Additionally, we can perform meaningful ordering of the values and apply mathematical operations like addition and multiplication on Family Dependents values, making them Ratio data.

Table 2.4 Quantitative attributes.

Attribute	Values	Type
Population	5000, 7680, 2500	Discrete
Price	3400.55, 6600.9	Continuous
Temperature	10◦C–20◦C, 20◦C–30◦C	Interval
Family Dependents	0, 2, 3	Ratio

How can data be stored effectively? This depends on the nature and source of data generation. Appropriate treatment is necessary to store the data in any data repository and retrieve data from the repository. Accordingly, appropriate mechanisms of data retrieval and mining need to be devised for the same. Available data-storage formats are discussed next.

2.3 Basic classification of data (Data-storage formats)

A data structure is a specific way to organize and store data in a computer so that the data may be accessed and changed effectively. It is a grouping of data values of various types that preserves the logical dependencies among the data elements and the functions or operations that apply to the stored data for retrieval, access, and update.

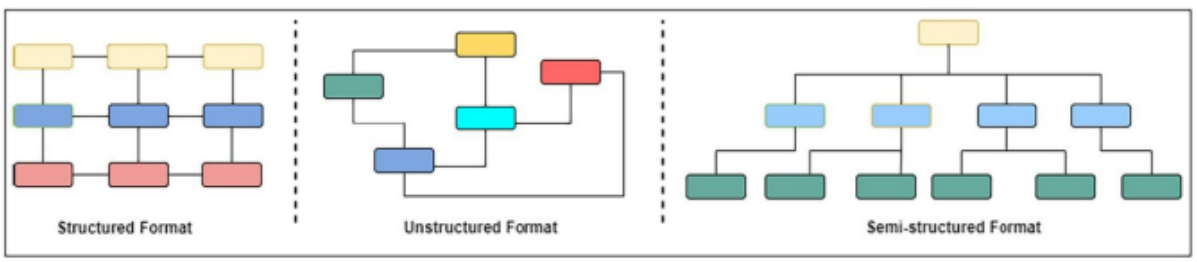


Figure 2.2 Three data formats. A node represents an entity or entity set, and an edge shows logical dependency. The color of a node indicates the characteristics of the entity. Similar color depicts the same entity or the same label for data representation.

The nature of the data dictates how data must be stored in a repository or database. Data may be stored in the following formats.

2.3.1 Structured data

Structured data can be either quantitative or qualitative. Structured data refers to data that have been organized into a structured format, such as a table or a spreadsheet, with a clear schema or data model that defines the relationships between different data elements. Structured data can be easily analyzed since such data follow a predefined data model. A data model, such as relational, object-oriented, or hierarchical, defines how data can be stored, processed, and accessed. For instance, in the relational data model, data are usually stored in row–column or tabular format. Excel files and relational databases are common examples of structured data. Structured data can easily be arranged in a specific order for effective searching. Every data field is distinct and can be accessed independently or in conjunction with information from other fields. In the illustrated structured format shown in Figure 2.2, each row depicts a collection of attributes or field values that belong to a particular record. A node contains a specific value for a specific record. Nodes are connected since the nodes belong to a single record with related attributes. As it is feasible to aggregate data from many areas in the database swiftly, structured data is incredibly powerful. It is one of the most common forms of data storage. Popular database-management systems (DBMS) follow a structured storage format for quick and efficient data access.

2.3.2 Unstructured data

Unlike structured data, unstructured data do not follow any specific format or adhere to a predefined model. Data or entities are associated arbitrarily due to logical association or dependency among them. As shown in Figure 2.2, such data can be viewed as a graph or network of data attributes (nodes). The lack of uniform structure leads to anomalies and ambiguities that make it more challenging to process such data using conventional programming platforms. Unstructured data are usually qualitative, including text, video, audio, images, reports, emails, social-media posts, graph data, and more. NoSQL databases nowadays provide a popular platform to store unstructured data. Storage and processing of unstructured data are growing needs as most of the current data-generation sources do not adhere to any rigid framework. In comparison to structured data, unstructured data formats are highly scalable, leading to the need for large or Big Data storage and processing.

2.3.3 Semistructured data

Another form of data arrangement is between structured and unstructured formats. Semistructured data represent a form of unstructured data that is loosely or partially structured. Although such data do not conform to specific data models as such, the containing data can be categorized with the help of tags or other markers incorporating certain semantic elements. As shown in Figure 2.3, attribute categorization may enforce a hierarchical arrangement composed of records and fields within the data. This format is also known as self-describing. A majority of the large-data sources produce semistructured data. Unlike the other two formats, it is relatively less complex and easy to analyze. Semistructured data analysis has become popular from a Big-Data perspective. Popular data-storage and -transaction formats such as JSON and XML are built for communication with semistructured data. To understand this better, we consider email as an example of semistructured data arrangement. Data components may be categorized as Sender, Recipient, Subject, Date, etc., and stored hierarchically.

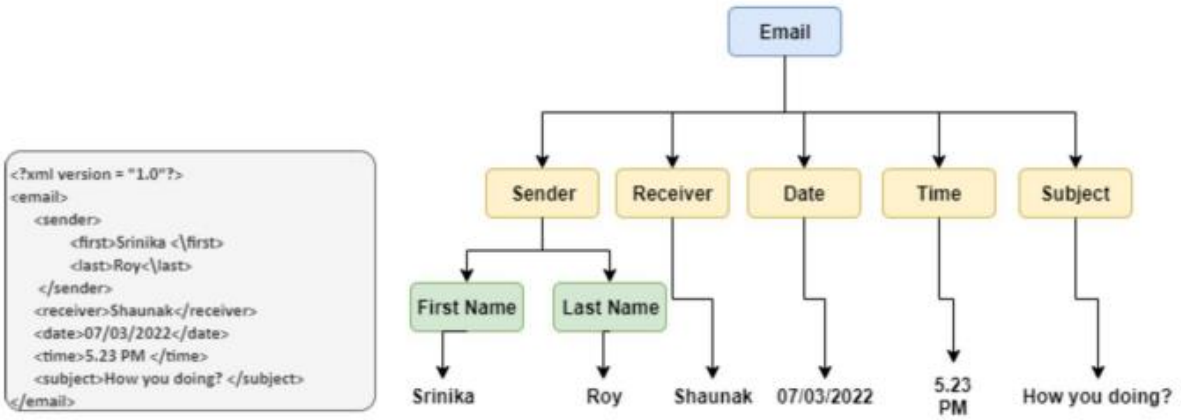


Figure 2.3 Snippet of XML code and semistructured arrangement of an email entity.

2.4 Data sources

The location from where the data are sourced is called the data source. The database, which may be kept on a local hard drive or a distant server, is the main data source for database-management systems. A computer program may use a file, a data sheet, a spreadsheet, an XML file, or even hard-coded data into the program itself as its data source. Sources are of two types, considering how data are collected or generated. A simple illustration (Figure 2.4) may be helpful to show how they are related to each other.

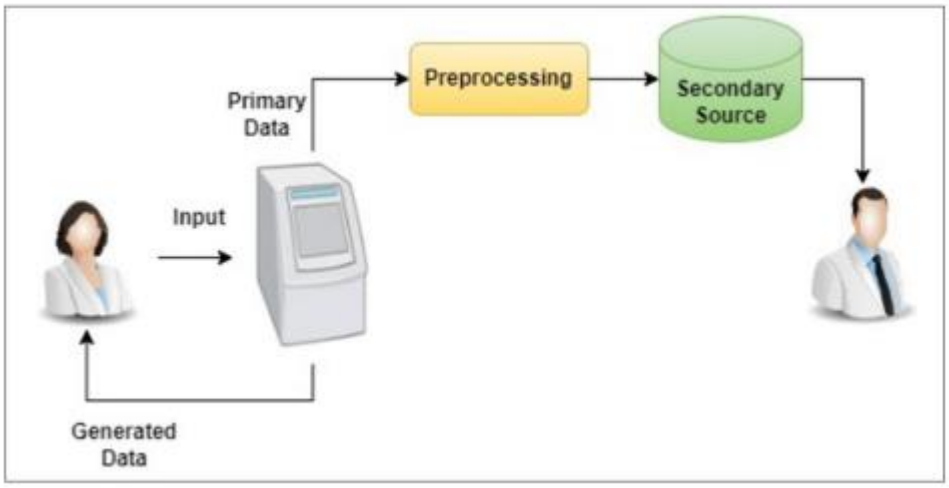


Figure 2.4 Transition from Primary to Secondary Data.

2.4.1 Primary sources

First-hand data are generated directly by data-generation sources and stored without major alteration or preprocessing. They can also be called original or raw data. Original and first-hand collection increases the reliability and authenticity of data. However, reliability further depends on how the data are produced and how reliable the source is. Primary data sources help retrieve hidden and novel facts. Generating primary data is an expensive and time-consuming activity. Once analyzed, primary data can be further processed to produce secondary data.

For instance, data collected directly by a researcher through a social survey or directly from IoT devices are primary data. Similarly, raw nucleotide sequences generated and recorded using a Next-Generation Sequencer are also primary data.

2.4.2 Secondary sources

Secondary data sources are repositories where the data have been derived from primary sources and are stored after curation. Usually, scientists generate the data and make them available for public use. The data may be passed through several rounds of experiments and statistical analysis. Although not generated directly, a secondary source ensures reliability and validity if it has tested the data. Secondary data are good for quick and cost- effective validation of a new system before deploying it or testing with original data. The suitability of available secondary data depends on the experimental purpose at hand. For instance, someone may want to validate her newly designed IoT intrusion detection system (IDS). Assume further that various computer-network attack examples are available but no examples of IoT attacks. In such a case, validating the IoT IDS with such data may not be appropriate. Either she needs to generate primary data or use further preprocessed available secondary data to fit.

2.4.3 Popular data sources

Many secondary data sources are publicly and privately available for Data Science research. The list is large; however, this section highlights a few popular and significant data repositories below.

UCI ML Repository: The machine-learning repository at the University of California, Irvine (UCI), is a great place to find open-source and cost-free datasets for machine learning experiments, covering a wide range of domains from biology to particle physics. It houses 622 datasets for machine-learning research. The datasets contain attributes that are of Categorical, Integer, and Real types. The datasets have been curated and cleaned. The repository includes only tabular data, primarily for classification tasks. This is limiting for anyone interested in natural language, computer vision, and other types of data. Data sizes are small and not suitable for Big-Data analysis.

Kaggle: This platform, recently acquired by Google, publishes datasets for machine learning experiments. It also offers GPU-integrated notebooks to solve Data Science challenges. At present, it houses approximately 77k datasets. Kaggle fosters advancements in machine learning through open community competitions. The datasets on Kaggle are divided into three categories: general-public datasets, private-competition datasets, and public-competition datasets. Except for the private datasets, all others are free to download. A majority of datasets are based on real-life use cases.

Awesome Public Datasets: This is a public repository of datasets covering 30 topics and tasks in diverse domains hosted in GitHub. The large datasets can be used to perform Big Data-related tasks. The quality and uniformity of the datasets are high.

NCBI: This is an open-access large collection of databases for exploratory molecular biology data analysis provided by the National Center for Biotechnology Information (NCBI) in the USA. Databases are grouped into six categories: Literature, Health, Genomes, Genes, Proteins, and Chemicals. It is a rich source of sequences, annotations, metadata, and other data related to genes and genomes.

SNAP: The Stanford Network Analysis Platform (SNAP) offers large network datasets. It covers large graphs derived from 80+ categories of domains, such as social networks, citation networks, web graphs, online communities, and online reviews. Libraries implementing more than 140 graph algorithms are available for download in SNAP. These algorithms can effectively process the characteristics and metadata of nodes and edges, manipulate massive graphs, compute structural properties, produce regular and random graphs, and generate nodes and edges.

MNIST: The Modified National Institute of Standards and Technology (MNIST) dataset is a large collection of hand-written digits, popular among deep-learning beginners and the image-processing community. The MNIST database contains 60,000 training images and 10,000 testing images. Extended MNIST (EMNIST) is a more refined database with 28×28 pixel-sized images. EMNIST is a dataset with balanced and unbalanced classes. MNIST is a very popular dataset for machine-learning experiments.

VoxCeleb Speech Corpus: This is a large collection of audiovisual data for speaker recognition. It includes more than one million real-world utterances from more than 6000 speakers. The dataset is available in two parts, VoxCeleb1 and VoxCeleb2. In VoxCeleb1, there are over 100,000 utterances for 1251 celebrities; in VoxCeleb2, there are over 1 million utterances for more than 6000 celebrities taken from YouTube videos. Different development and test sets are segregated with different speakers.

Google Dataset Search: Google has developed a dedicated dataset search engine, Google Dataset Search, that helps researchers search and download publicly available databases. Google claims that its Dataset Search engine has indexed about 25 million datasets, and one may obtain useful information about them. To locate these datasets in the search results, Google Dataset Search leverages schema.org and other metadata standards.

2.4.4 Homogeneous vs. heterogeneous data sources

It is important to consult multiple reliable data sources for effective data analytics and smart decision making. However, reliability is difficult to ensure for data obtained from a single arbitrary source. In addition, data obtained from a single source may be incomplete, biased, and/or unbalanced. The alternative is to amalgamate similar data from multiple sources and make decisions based on majority agreement.

Considering multiple datasets, if they are uniform and similar to one another with reference to data sources, data types, and formats, the sources are called homogeneous data sources. Homogeneous data sources can be combined easily without much additional processing. For example, Walmart may want to analyze countrywide sales patterns. It needs to integrate its transaction databases across different stores located in different places. Since various stores use a common database structure, it is easy to integrate large collection transactions for analysis.

On the contrary, heterogeneous data sources are sources with high variability of data types and formats. It is difficult to integrate heterogeneous data sources without preprocessing necessary to transform the datasets to a uniform format. This may sometimes lead to a loss of information while integrating. For instance, devices on the Internet of Things (IoT) frequently produce heterogeneous data. The heterogeneity of IoT data sources is one of the biggest challenges for data processing. This is due to the variety of data acquisition devices and the lack of common data-acquisition, -structuring, and -storage protocols. Four different types of data heterogeneity have been identified.

• **Syntactic heterogeneity:** This appears when data sources are created using different programming languages with varying structures.

• **Conceptual heterogeneity:** This is also known as semantic or logical heterogeneity. It represents the differences that arise while modeling the data in various sources within the same domain of interest.

• **Terminological heterogeneity:** Different data sources referring to the same entities with different names lead to such heterogeneity.

• **Semiotic heterogeneity:** This is also known as pragmatic heterogeneity due to varying interpretations of the same entity by different users.

2.5 Data generation

Researchers and data scientists frequently encounter circumstances where they either lack suitable real-world (primary or secondary) data or may be unable to use such data due to closed access, privacy issues, and/or litigation potential. The alternative solution in such cases is to use artificially generated datasets produced using synthetic-data generation. Synthetic-data generation is the process of creating artificial data as a replacement for real-world data, either manually or automatically, using computer simulations or algorithms. If the original data are unavailable, "fake" data may be created entirely from scratch or seeded with a real dataset. A realistic and statistically well-distributed synthetic dataset that works well with the target Data Science algorithms is desired while replacing real data to make the algorithms learn and behave as close to the real system as possible.

Concepts such as data augmentation and randomization are related to creating synthetic data. However, conceptually, there is a difference between them. Data augmentation involves adding slightly altered copies of already-existing dataset samples. Using data augmentation, a dataset can be expanded with nearly identical samples but with slight differences in data characteristics. This is useful when adequate data samples are scarce. For example, adding new face images into a face database by altering the orientation, brightness, and shape of the faces augments the face database. In contrast, data randomization does not produce new data elements; it simply changes slightly the items already present in the dataset. With randomization, data attributes or features may be altered in some samples in the dataset. Randomization helps protect sensitive data by making it harder for unauthorized persons to identify or infer personal information from the data. By randomizing the order of the data, any patterns or relationships that could be used to identify individuals are disrupted. For example, a clinical dataset of cancer patients may be randomized by altering a few selected features or attributes of the real data in order to hide the patient's personal details.

On the other hand, synthetic data allow us to create entirely new data elements similar to real data. However, such data may not accurately represent the real data in its entirety. In essence, synthetic data duplicate some patterns that may already exist in reality, introducing such properties without explicitly representing them.

The advantages of using synthetic data come from cost and time effectiveness, better data-labeling accuracy because the labels are already established, scalability (it is simple to produce enormous amounts of simulated data), and controlled data distribution and variability. Data samples can be generated for exceptional scenarios where such events do not occur regularly in the real world, however high the possibility of occurrence in the near future. Usually, real-data samples are mimicked to generate additional data samples with similar distribution and nature. An appropriate mathematical or statistical model is built by observing the real data for this purpose.

2.5.1 Types of synthetic data

The intention behind synthetic-data generation is to provide alternatives to real data when real data are unavailable or have privacy, intellectual property, or other similar issues. Based on the amount of artificiality or syntheticity in the generated data, they can be classified into two types:

• **Fully Synthetic:** Data that are fully synthetic and have no direct relationship to actual data. Even though all the necessary characteristics of real data are present, it is likely that there is no real individual or example that corresponds to a generated data example. When creating fully

synthetic data, the data generator often finds the density function of the features in the real data and estimates the appropriate parameters for data generation. The bootstrap method and multiple approaches to imputation are common ways of producing fully synthetic data.

• **Partially Synthetic:** All information from the real data is retained in partially synthetic data, with the exception of sensitive information. Only selected sensitive feature values are replaced with synthetic values in such data. Most real values are likely to be retained in the carefully curated synthetic-data collection because they are extracted from the real data. Multiple imputation methods and model-based procedures are used for producing partially synthetic data.

2.5.2 Data-generation steps

A basic data generation step may follow five major steps. The typical workflow of the data generation pipeline is shown in Figure 2.5.

1. Determine Objective: The first step is to understand and establish the objectives for the planned synthetic-data generation and the techniques to be used later for data analysis. Further, one needs to understand organizational, legal, and other restrictions because data privacy is currently a major concern. For instance, if one needs clinical data for AIDS patients for analysis, confidentiality is a serious issue.

2. Select Generator: Next, one needs to select a data-generation model. For simulation, appropriate technical knowledge and adequate infrastructure are necessary. For example, generating artificial IoT traffic data using Monte Carlo simulation requires knowledge of various model hyperparameters to be tuned.

3. Collect Sample Data: Most synthetic data generators need real-data samples to learn the underlying probability distribution. The availability of carefully collected real-data samples determines the quality of the generated synthetic data.

4. Train Model: The selected data-generation model needs to be trained using collected real-data samples by tuning the hyperparameters. The target is to make the model understand well the sample data and create data close to real-data samples.

5. Generate Data: Once trained, synthetic datasets of any size can be generated using the trained model.

6. Evaluate the Data: Assessing the quality of generated synthetic data is important to ensure usability. The best way to evaluate this is to feed the data to data-analysis algorithms, learn a trained model, and test its quality with real-data samples. If the system performs according to expectation, the generated data can be used or archived. In the case of underperformance, the error may be fed back to the synthetic-data-generation model for further tuning.

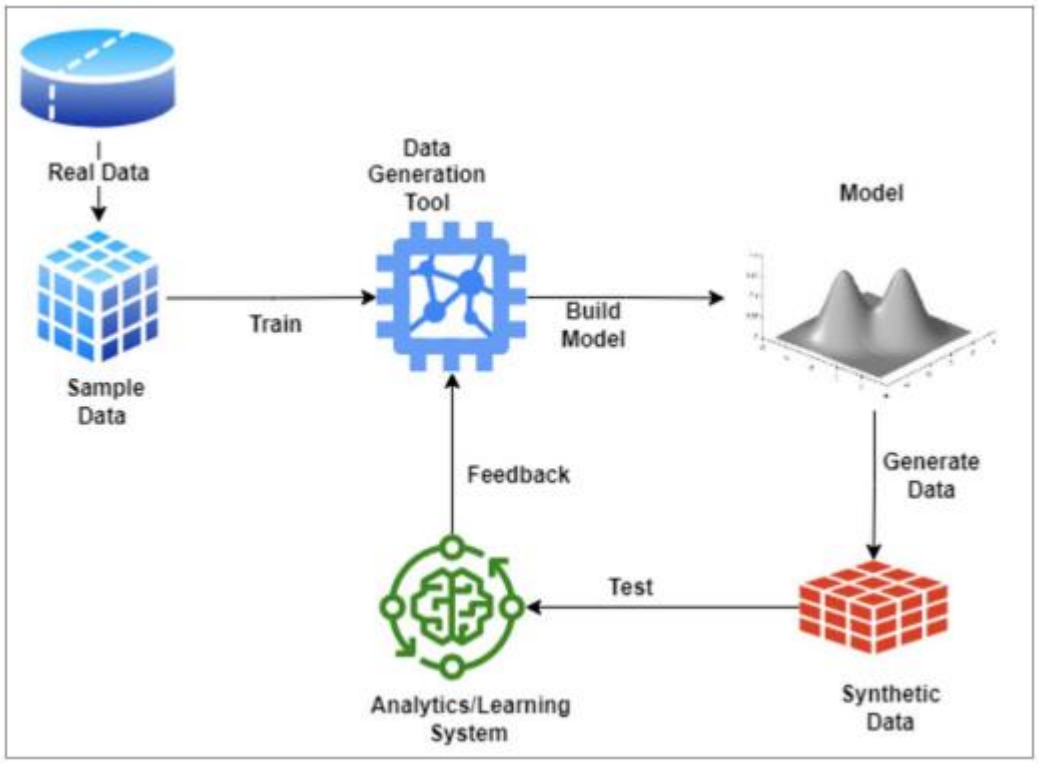


Figure 2.5 Synthetic-Data-Generation Workflow.

2.5.3 Generation methods

To generate synthetic data, the following techniques may be used.

• **Statistical Distribution:** The idea behind using a statistical distribution for synthetic data generation is first to learn the underlying probability distribution for real-data examples. The quality of synthetic data to be produced depends on sampling. If sampling is biased, naturally, the outcome may be ill-distributed and may not represent real data. There may be scenarios where no data samples are available. In such scenarios, expert judgment and experience matter greatly. A data scientist is able to generate random samples by following a probability distribution such as Gaussian, or Chi-square distribution.

• **Agent Modeling:** First, an agent data generator is modeled to behave as if in real scenarios. Next, it uses a modeled agent for random fake-data generation. It may perform curve fitting to fit real data to a known distribution. Later, a suitable simulation method like Monte Carlo can be used for synthetic-data generation. A machine-learning model like a decision tree can also be used as an alternative to fit the observed distribution. However, a proper selection of the machine-learning model is important as every model has its strengths and limitations for a good fit to the data distribution. A hybrid approach may also be adapted for the same. A part of the synthetic data may be produced using a statistical distribution approach, and the rest can be generated using agent modeling.

• **Deep Neural Networks:** In recent years, deep neural-network-based techniques have become popular and effective for generating bulk and close-to-real data. Advanced machine-learning approaches, including deep models, can learn relevant features for a variety of data. Neural networks are especially well suited for creating synthetic data through trial-and-error methods. They can learn to duplicate the data and generalize to create a representation that may have characteristics of real data. Two popular deep neural architectures include Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN).

VAEs are members of the autoencoder family and represent an unsupervised technique. They are highly effective generative models that are able to learn the underlying distribution of the original data. The primary objective of a VAE is to guarantee that input and output data remain strikingly identical. GAN is also an unsupervised technique that belongs to a generative family. It is an interesting model with two important components, a discriminator and a generator that work in an adversarial fashion. The generator is responsible for creating synthetic data, whereas the job of the discriminator is to detect the generated data as fake. By detecting synthetic data as fake, the generator gradually improves itself so that the synthetic-data examples become close to real-data examples, and undetectable by the discriminator as fake.

2.5.4 Tools for data generation

A plethora of synthetic data generators is available. They may be primarily categorized into two types: ready-to-use software and data-generator libraries. On the one hand, software tools are easy to use with limited or no knowledge of coding, but on the other hand, good, ready-to-use tools are usually not free. Moreover, other than parameter setting, no external control may be possible during data generation when using such a tool. In the case of software libraries, one needs the knowledge of coding so as to obtain better usability, customization, and control of synthetic-data generation. Discussed below are a few software tools or packages for generating data in different domains.

2.5.4.1 Software tools

A few examples of data-generation tools used in various domains where Data Science is commonly used are listed below.

• **Banking:** A good number of ready-to-use software tools are available to generate data for the banking, insurance, and fintech sectors. A few notable ones are Hazy, Datomize, Mostly.AI, and Facteus. The tools generate raw banking data for fintech industries. They provide alternative ways to generate banking customer data because real organizations in this sector have severe restrictions in terms of privacy and confidentiality in data storage and sharing. Synthetic data are generated by avoiding situations that pose the risk of fraud, as identified when gathering actual consumer data.

• **Healthcare:** Availability of patient data enables data scientists to design personalized care plans. Due to issues of privacy, security, and ownership, access to healthcare data is highly restricted. MDClone is a software tool that provides a systematic approach to democratizing the availability of healthcare data for research, analytics, and synthesis while learning to code protecting sensitive data. It creates fake patient data based on real statistical characteristics of patients. Similarly, SyntheticMass8 aims to statistically recreate data reflecting the actual population in terms of demographics, disease burdens, vaccinations, medical visits, and social determinants.

• **Computer Vision:** CVEDIA offers a synthetic-data-generation tool for computer vision. CVEDIA's SynCity, allows users to create synthetic environments, generate synthetic data, and simulate scenarios for various use cases such as autonomous vehicles, robotics, and augmented and virtual reality. Rendered.AI creates synthetic datasets for the satellite, autonomous vehicle, robotics, and healthcare industries. For example, using it one can recreate satellite-like images.

2.5.4.2 Python libraries

The Python language has become popular for Data Science research and development, and several Python-based data-generation libraries are freely available. It is important to choose the right Python library for the type of data that needs to be generated.

• **Scikit-Learn:** sklearn is a popular library that can be used to generate fake data for various Data Science experiments. It can generate data for multiclass classification, clustering, regression analysis, and many other tasks. It can also create data examples similar to data samples drawn from real data.

• **Numpy:** This is another library that can be used to generate random data with various probabilistic distributions. Numpy and sklearn need to be used together for data generation and benchmarking activities.

• **Pydbgen:** This is a lightweight library to generate random data examples and export in popular data formats such as Pandas data frame, SQLite table, or MS Excel.

• **Sympy:** This library can be used to create datasets with complex symbolic expressions for regression and classification tasks.

• **Synthetic Data Vault (SDV):** This library can produce anonymous datasets based on the characteristics of real data. The key capabilities are modeling relational, time-series, single tables, and benchmarking data. The need for a sizable real dataset to train SDV models is a drawback that stands out.

Listed in Table 2.5 is a summary of various Python libraries for data generation. Like Python, **R** is also a commonly used programming language in Data Science. bindata, charlatan, fabricatR, fakeR, GenOrd, MultiOrd, and SimMultiCorrData are example of packages in **R** that can generate fake data.

Table 2.5 Python Synthetic-Data-Generation Libraries and capabilities.

Library	Activity
TimeseriesGenerator, SDV	Time-series data generator
Mesa	Framework for building, analyzing, and visualizing agent-based models
Zpy, Blender	Computer vision and 3D creation suites
Faker, Pydbgen, Mimesis	Fake database generation
DataSynthesizer	Simulates a given dataset

2.6 Summary

To start studying Data Science and learn about the use and practice of Data Science, it is of utmost importance to understand well the nature of data, data-storage formats, and sources of data availability. This session has presented a careful discussion of these topics. Real data are not always available for use due to various reasons. The solution is to artificially generate data that mimic real-life data. Various data-generation methods, basic steps of generation, and popular tools and libraries have also been discussed. The quality of data decides the outcome of data analytics. Quality data collection, preparation, and generation is an important, yet challenging task. The next session is dedicated to data preparation processes.

Lesson 3: Data preparation

3.1 Introduction

The quality of input data largely determines the effectiveness of Data Science tasks. If the quality of data is not adequate, even a brilliant exploratory or predictive Data Science technique might fail to produce reliable results. Data quality refers to the state of the data in terms of completeness, correctness, and consistency. Assessing and ensuring data quality by removing errors and data inconsistencies can improve the input-data quality and makes it fit for the intended purpose.

The availability of low-cost computing resources and high-speed data transmission has led to explosive data growth in many data-intensive computing domains. High throughput experimental processes have resulted in the production of massive amounts of data, particularly in the biomedical field where experiments generate a wide variety of data such as mRNA, miRNA, gene expression, and protein–protein interaction data. Other sources of bulk data generation include the stock market, social networks, the Internet of Things (IoT), sensor networks, and business transactions. However, the effective storage and transmission of such data present a significant challenge. To address this challenge, efficient and scalable exploratory Data Science techniques are emerging as important tools for knowledge discovery from heterogeneous, multimodal data sources. Despite the plethora of data-exploration and machine-learning methods proposed in recent decades, the quality of input data remains a critical factor in producing reliable results. The heterogeneity of data due to different acquisition techniques and standards, as well as the presence of nonuniform devices in geographically distributed data repositories, make the task of producing high-quality data difficult in many cases.

In real-world scenarios, data are frequently of low quality and may not be suitable for use in advanced Data Science methods. Furthermore, these data may be incomplete or contain erroneous or misleading elements, known as outliers. Additionally, the same set of data can be represented in various formats, and values may have been normalized differently for different purposes. To use exploratory or predictive Data Science methods, it is necessary to adequately prepare the data beforehand through data preprocessing, which comprises several essential steps, including **Data Cleaning**, **Data Reduction**, **Data Transformation**, and **Data Integration**. Each of these steps is independent, equally crucial, and can be performed separately. The data and the tasks to be executed determine which steps need to be taken and when, and a domain expert may need to intervene to determine the appropriate steps for a particular dataset. It is also necessary to avoid using the steps as black boxes without understanding their underlying processes. A schematic overview of the overall workflow of preprocessing steps is illustrated in Figure 3.1.

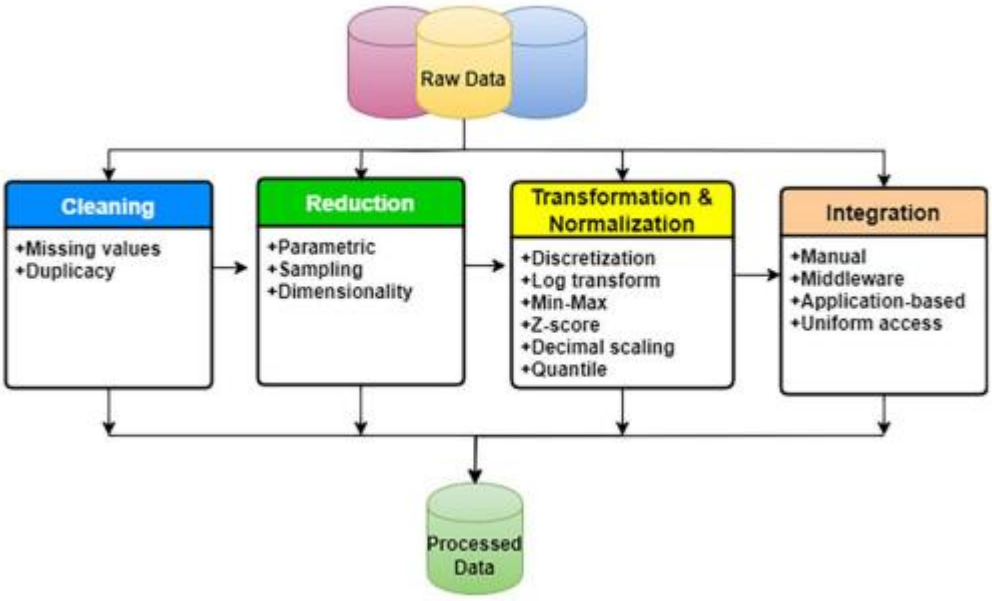


Figure 3.1 Data preprocessing steps and activities. The raw data may be single or multisourced, uniform or heterogeneous format.

Next, we discuss the major data-preprocessing steps in detail.

3.2 Data cleaning

The quality of data plays a significant role in determining the quality of the resulting output when using any algorithm. Creating high-quality data involves a step called data cleaning. This step involves removing noise, estimating missing values, and correcting background information before normalizing the data. Data cleaning typically addresses two primary issues: missing values and data duplication. The latter issue can increase computation time without providing any additional value to the final result.

3.2.1 Handling missing values

Dealing with missing values is a crucial challenge in data preprocessing. Missing values can have negative effects on the distribution of the dataset, introduce bias in results, or cause descriptive and predictive Data Science algorithms to fail. Thus, they need to be handled with care to ensure optimal performance of the methods. Missing data may arise due to various technical reasons. Missing values in data can occur due to several reasons. For example, issues with the measuring equipment, such as malfunctioning or physical limitations, can result in certain values not being recorded. Similarly, missing values can occur during data transportation if there are interruptions in the communication media. In some cases, missing values may not have been addressed during the data-collection phase because they were considered insignificant. Additionally, some values may have been removed due to inconsistencies with other recorded data.

Different types of missing data can be classified into several categories:

• **Missing Completely at Random (MCAR):** This refers to the missing data, in which the absence of observation is unrelated to its actual value.

• **Missing at Random (MAR):** When the probability of a missing observation is dependent on other aspects of the observed data, but not on the value of the missing observation itself.

• **Missing Not at Random (MNAR):** This is the missing data that do not fit into either of the aforementioned two categories. In this type of missing value, the likelihood of a missing observation is correlated with its value.

To illustrate the different types of missing data values, let us consider the example of two variables, a and b, represented by vectors A and B, respectively. While all the values in vector B are recorded, there are some missing values in vector A. A missing value in A is considered MCAR if its probability of being missing is not dependent on the values in both A and B. Conversely, if the likelihood of a missing value in A depends on the values in B but not on the values in A, it is called MAR.

There are several ways to address missing data, and ignoring it altogether is one of them. However, this approach can have a detrimental impact on the results if a large proportion of the dataset is missing. In the case of MNAR missing values, parameter estimation is necessary to model the missing data. To handle missing data, four broad classes of approaches are available.

3.2.1.1 Ignoring and discarding data

One way to handle missing values is to ignore them, which is only appropriate for the MCAR situation. However, using this method for other types of missing data can introduce bias into the data and ultimately impact the results. There are two ways to discard missing data: complete-case analysis and attribute discarding. Complete-case analysis is the most common method and involves removing all instances with missing data and conducting further analysis only on the remaining data.

The second method involves determining the extent of missing data with respect to the number of instances and the relevance of the attributes in the dataset. Based on these considerations, instances that are not widespread or whose relevance to the dataset is not significant can be discarded. However, if an attribute with missing values is highly relevant, it must be retained in the dataset.

3.2.1.2 Parameter estimation

This method is appropriate for MNAR missing data and requires parameter estimation using statistical or supervised learning approaches such as probabilistic, KNN, or SVD models. To estimate the parameters of a model, the maximum-likelihood approach can be used, which involves a variant of the Expectation-Maximization algorithm. The most common method for this approach is called BPCA, which involves three elementary steps: principal-components regression, Bayesian estimation, and an iterative Expectation-Maximization (EM) algorithm. This method is frequently used for biological data cleaning.

3.2.1.3 Imputation

Imputation refers to the actions taken to address missing data in a dataset. The objective of imputation is to replace missing data with estimated values by utilizing information from measured values to infer population parameters. Imputation may not provide the best prediction for the missing values, but it can enhance the completeness of the dataset. Imputation approaches can be categorized into three broad classes: Single-value imputation, Hot-deck and cold-deck methods, and Multiple imputations.

1. Single-value imputation: This is a method of replacing a missing value with a single estimated value, which is assumed to be very similar to the value that would have been observed if the data example were complete. There are various ways to substitute a single value for the missing value in a data example. Some of these methods include:

(i) Mean imputation: Replacing a missing attribute value in a data example with the mean of the existing values of the same attribute over all data examples is a common approach. However, this method can result in a poorer estimation of the standard deviation for the attribute in the resulting dataset because it reduces the diversity of the data.

(ii) Regression imputation: Using a regression model to predict the missing value of an attribute in a data example based on the actual values of the attribute in other data examples is another method for imputing missing values. This approach preserves the variability in the dataset and avoids introducing bias in the results. Unlike mean imputation, which underestimates the correlation between attributes due to the loss of variability, this method tends to overestimate the correlation.

(iii) Least-squares regression (LS impute): A special form of a regression model is used for imputing missing values, which involves developing an equation for a function that minimizes the sum of squared errors for the attribute's existing values across data examples from the model.

(iv) Local least squares (LLS impute): This imputation method is similar to LS impute but involves a prior step that is known as KNN impute. KNN impute first identifies the K nearest neighbors that have high correlations, and then represents a target missing value in a data example using a linear combination of values in similar data examples.

(v) K-Nearest-Neighbor imputation (KNN impute): The missing value for a particular attribute is estimated by minimizing the distance between the observed and predicted values using the known values of other examples. The K-nearest neighbors to the example with missing data are identified, and the final imputed value is estimated based on these neighbors.

2. Hot-deck and cold-deck methods: Hot-deck imputation is a method of filling in missing data on variables by using observed values from respondents within the same survey dataset. It is also known as Last Observation Carried Forward (LOCF) and can be divided into random and deterministic hot-deck prediction, depending on the donor with which the similarity is computed. This technique has the advantage of being free from parameter estimation and prediction ambiguities, and it uses only logical and credible values obtained from the donor pool set. In contrast, cold-deck imputation selects donor examples from an external dataset to replace a missing attribute value in a data example.

3. Multiple imputations: To determine the missing attribute value, multiple imputation involves a sequence of computations. It replaces each missing value with a set of plausible values, which are then analyzed to determine the final predicted value. This technique assumes a monotonic missing pattern, where if one value for an attribute is missing in a sequence of values, all subsequent values are also missing. Two prevalent methods used to determine the set of plausible values are:

• **Regression method:** A regression model is constructed by utilizing the existing attribute values for attributes with missing values, and this process is carried out following the monotonic property. Subsequently, a new model is formed based on this regression model, and the process is repeated sequentially for each attribute that has missing values.

• **Propensity-score method:** In order to address missing values in a dataset, propensity score imputation is used to calculate the conditional probability of assigning a certain value to a missing attribute, given a set of observed values. This method involves calculating a propensity score for each attribute with missing values, indicating the probability of the value being

missing. Grouped data are then subjected to Bayesian bootstrap imputation based on the propensity score to obtain a set of values to replace the missing values.

3.2.2 Duplicate-data detection

The problem of storage shortage due to duplicate-data values in instances or attributes is known as data redundancy. The conventional method for addressing this issue involves identifying similar value chunks and removing duplicates from them. However, this approach is time intensive, and alternative techniques have been developed to handle redundant data.

3.2.2.1 Knowledge-based methods

One approach to address data duplication is to utilize domain-specific knowledge from knowledge bases in data-cleaning tasks. Intelliclean is a tool that implements this approach by standardizing abbreviations to ensure consistency across records. For example, if one record abbreviates the word street as St., and another abbreviates it as Str, while another record uses the full word, all three records are standardized to the same abbreviation. Then, domain-specific rules are applied to identify and merge duplicates, and to flag any other anomalies. This technique allows for more efficient and accurate duplication elimination, as compared to traditional methods that rely on identifying similar values and removing duplicates from them.

3.2.2.2 ETL method

The ETL method, which stands for extraction, transformation, and loading, has become the most widely used approach for eliminating duplicates. This method involves three steps, namely extraction, transformation, and loading. It allows for two types of duplicate elimination: instance-level and schema-level processing. Instance-level processing aims to clean errors within the data itself, such as typos or misspellings. On the other hand, schema-level processing typically involves transforming the database into a new schema or data warehouse.

3.3 Data reduction

Large-scale data mining can be time consuming and resource intensive, making it impractical and infeasible. Using the entire dataset is not always necessary and may not contribute significantly to the quality of the results compared to using a smaller version of the data. Data-reduction techniques can reduce the size of the data by reducing its volume or the number of attributes without compromising the integrity of the data. There are several approaches available for data reduction, and an ideal method should be efficient and produce similar results to those obtained with the full dataset. Some of these methods are discussed briefly below.

3.3.1 Parametric data reduction

Parametric data-reduction techniques aim to reduce the volume of original data by using a more compact representation of the data. This is achieved by fitting a parametric model based on the distribution of the data and estimating the optimal values of the parameters required to represent that model. This approach involves storing only the model parameters and the outliers. Regression and nonlinear models are two popular examples of parametric data-reduction techniques.

On the other hand, nonparametric approaches do not rely on any specific model. Instead, they summarize the data using sample statistics, which will be discussed in more detail below.

3.3.2 Sampling

Dealing with a large dataset in its entirety can be a costly and time-consuming task. To avoid this, a smaller subset of representative data, consisting of k instances, is selected from a larger

dataset of G instances, where k is less than or equal to G. This process is known as sampling. Sampling can be divided into the following categories:

• **Simple random sampling without replacement:** In this approach, a sample of size k is selected from a large dataset of size G based on a probability of selection (p), and the selected samples are not returned to the dataset. This is known as sampling without replacement, which can result in a nonzero covariance between the chosen samples, making the computations more complex. However, if the dataset contains a large number of instances, the covariance is likely to be negligible.

• **Simple random sampling with replacement:** In this type of sampling, the chosen sample is returned to the dataset, allowing for the possibility of duplicated data in the sample. This results in a zero covariance between two selected samples. When dealing with skewed data distributions, simple random sampling with replacement often yields poor results for any data-analysis technique. In such cases, sampling with or without replacement makes little difference. However, sampling without replacement typically produces more precise estimates than sampling with replacement. For unbalanced datasets with uneven sample distribution, adaptive sampling methods like stratified sampling and cluster sampling are more likely to improve performance.

• **Stratified sampling:** To perform stratified sampling, the original dataset is divided into subgroups or strata, from which a sample is generated using simple random sampling. On the other hand, in cluster sampling, the original dataset is divided into subgroups or clusters, and a random selection of n clusters is taken from the total N clusters. Then, the sample is generated by selecting elements from these selected clusters.

3.3.3 Dimensionality reduction

The process of dimensionality reduction involves eliminating attributes that are unnecessary or insignificant. Having irrelevant attributes in a dataset can make it difficult to extract valuable information and can also lead to erroneous analysis, often referred to as the Curse of Dimensionality. Data sets with a high number of dimensions are also likely to be sparse. This can cause issues with machine-learning techniques such as clustering, which rely on data density and distance, and can lead to inaccurate results in the case of sparse data. To enhance the performance of machine-learning algorithms with high-dimensional data sets, reducing dimensionality is crucial. The reduction process should preserve the information content of the original data to the greatest extent possible and ensure that the use of the reduced dataset does not affect the final result. One way to achieve dimensionality reduction is by selecting only the relevant attributes and ignoring the irrelevant ones.

One widely used technique for dimensionality reduction is Principal-Component Analysis (PCA), as proposed by Hotelling. PCA is a statistical method that utilizes an orthogonal transformation to map a set of observations of possibly interrelated variables to a set of values of linearly independent variables known as principal components. It looks for k orthogonal vectors of n dimensions ($k < n$) from the original data that best represent it. This process projects the original data onto linear subspaces, resulting in a low-dimensional approximation of the original data and, consequently, dimensionality reduction.

3.4 Data transformation

To analyze data that are collected from diverse sources, it is common to find them in various formats due to the differences in experimental setups, conditions, and variables ofinterest during data collection. Transformation of the data is often necessary and effective for analysis. Transformation involves changing the format or structure of data from one to another to align better with the assumptions of statistical inference procedures or to enhance interpretability.

The transformation function y = f (x) is used to convert the data from one domain to another. Several techniques for data transformation are explained below.

3.4.1 Discretization

Discretization transforms the domain of values for each quantitative attribute into discrete intervals. Using discrete values during data processing offers a number of benefits, such as:

• Storing discrete data requires less memory compared to storing continuous data.

• Discrete data are likely to be simpler to visualize and understand.

• Discrete data are known to improve the accuracy of performance in some systems.

The following two concepts are commonly used when discussing discretization or interval generation:

• **Cutpoints:** The process of discretization involves identifying cutpoints, which are specific values within the range of a continuous attribute. These cutpoints divide the range of continuous values into successive intervals or bins. For instance, a continuous range such as $< a \ldots b >$ may be divided into intervals like [a,c] and (c, b], where c represents a cutpoint.

• **Arity:** Arity refers to the number of intervals generated for an attribute through discretization. It is determined by adding one to the number of cutpoints used in the process of discretization for that attribute. A high arity can lead to longer learning processes, while a very low arity may result in reduced predictive accuracy.

Any discretization method follows the enumerated steps below:

1. Arrange the continuous values in ascending or descending order for each attribute that needs to be discretized.

2. Determine the cutpoints for each of these attributes based on a certain criterion or method.

3. Generate intervals by dividing the range of values for each attribute using the computed cutpoints.

4. Map each value to its corresponding interval and represent it with a discrete value or category.

Below are a few measures that can be used by a discretizer to evaluate cutpoints and group different continuous values into separate discretized intervals:

• **Equal Width:** This method divides the range of values into equal intervals, each with the same width. The width of each interval is calculated as the difference between the maximum and minimum value of the attribute divided by the desired number of intervals.

• **Equal Frequency:** This method divides the range of values into intervals so that each interval contains the same number of observations. This method is based on the frequency of occurrence of values in the dataset.

• **Entropy-based:** This method is based on information theory and evaluates the quality of a cutpoint based on the reduction of entropy achieved by dividing the data into two intervals. The cutpoint that maximizes the reduction of entropy is selected.

• **Chi-square:** This method evaluates the quality of a cutpoint based on the difference in the distribution of values in the two intervals obtained by splitting the data. The cutpoint that maximizes the chi-square statistic is selected.

The choice of measure depends on the nature of the data and the objective of the analysis.

There are various techniques proposed in the literature for discretization, which are broadly classified into two categories: Supervised and Unsupervised techniques. Supervised techniques use class levels for discretization, whereas unsupervised techniques do not. Below are some of the techniques in each category.

3.4.1.1 Supervised discretization

This type of discretization technique uses the class labels of the original dataset to convert continuous data into discrete ranges. There are two types of supervised discretization techniques.

1. Entropy-Based Discretization Method: The entropy measure is utilized in this approach to determine the boundaries for discretization. Chiu et al. introduced a hierarchical technique that maximizes the Shannon entropy across the discretized space. The technique initiates with k partitions and applies hill climbing to refine the partitions, using the same entropy measure to obtain more precise intervals.

2. Chi-Square-Based Discretization: This technique of discretization employs the Chi-Square test to assess the probability of similarity between data in different intervals. Initially, each unique value of the attribute is in its own interval, and the x^2 value is calculated for each interval. The intervals with the smallest x^2 values are merged, and the process is repeated until no further satisfactory merging is possible based on the x^2 values.

3.4.1.2 Unsupervised discretization

The unsupervised discretization technique does not rely on any class information to determine the boundaries. Below are some examples of this type of discretization technique:

1. Average and Midranged value discretization: This is a binary discretization technique that uses the average value of the data to divide the continuous variable into two intervals. Values below the average are assigned to one interval, and values above the average are assigned to another interval. This technique does not use any class information to determine the boundaries.

To apply the average discretization method on a vector A = [23.73, 5.45, 3.03, 10.17, 5.05], the first step is to calculate the average score, $\bar{A}$ == 9.88. Based on this average, it discretizes the value using following equation.

$$D_i = \begin{cases} 1, & if\ A(i) >= \bar{A} \\ 0, & otherwise \end{cases}$$

In our example, $\bar{A}$ = 9.486, therefore, the discretized vector using average-value discretization is D=[1 0 0 1 0].

The midrange discretization method uses the middle value or midrange of an attribute in the dataset to determine the class boundary. To obtain the midrange value of a vector of values, the equation M = (H + U)/2 is used, where H is the maximum value and U is the minimum value in the vector. Discretization of values is carried out by assigning them to a class based on whether they fall below or above the midrange value.

$$D_i = \begin{cases} 1, & if\ A(i) >= M \\ 0, & otherwise \end{cases}$$

For the vector A, we obtain M as 13.38. Therefore, the corresponding discretized vector of values D = [1 0 0 0 0]. This approach is not of much use as it lacks robustness since outliers change the resulting vector significantly.

2. Equal-Width Discretization: This technique involves dividing the data into a specific number of intervals or bins, where the boundaries for each bin are determined by the range of the attribute being discretized. The range is divided into equal parts, each corresponding to one of the intervals, and the boundaries for each interval are determined by the maximum (H) and minimum values (U) of the attribute.

This approach divides the data into k equal intervals. The upper and lower bounds of the intervals are decided by the difference between the maximum and minimum values for the attribute of the dataset. The number of intervals, k, is specified by the user, and the lower and upper boundaries, pr and pr+1 of the class are obtained from the data (in particular, using the H and U values) in the vector according to the equation

$$p_{r+1} = p_r + (H - U)/k.$$

For our example, if k = 3, the group division occurs as given below

$$D_i = \begin{cases} 1, & \text{if } \mathrm{p_{r0}} <= \mathrm{A(i)} < \mathrm{p_{r1}} \\ 2, & \text{if } \mathrm{p_{r1}} <= \mathrm{A(i)} < \mathrm{p_{r2}} \\ 3, & \text{if } \mathrm{p_{r2}} <= \mathrm{A(i)} <= \mathrm{p_{r3.}} \end{cases}$$

Therefore the corresponding discretized vector is D = [3 1 1 2 1].

3. K-means Discretization: This technique divides the values of the variable into k intervals such that adjacent values are assigned to the same interval. The number of intervals k is specified by the user, and the size of each interval is determined by the range of the data divided by k. Values are then assigned to the appropriate interval based on their position within the range. For k = 2, discretization can be performed as follows.

$$D_i = \begin{cases} 1, & \text{if } \mathrm{A_{i+1}} - \mathrm{A_i} <= (\mathrm{H} - \mathrm{U})/\mathrm{k} \\ 0, & \text{otherwise.} \end{cases}$$

For our example, if k = 2, the discretized vector D = [3 1 1 2 1].

3.5 Data normalization

The term “normalization” is frequently used interchangeably with “transformation”. Normalization refers to the process of transforming data so that the resulting distribution is approximately normal. The following paragraphs cover several techniques for normalization.

3.5.1 Min–max normalization

Given an attribute value, *x*, from the original dataset, it is transformed to a new value, x', through a mapping process.

$$x' = \frac{x - min}{max - min},$$

where min and max are the minimum and maximum values of the attribute in the dataset.

3.5.2 Z-score normalization

The transformation of variable values is determined by the mean and standard deviation of the variable values in the dataset. For instance, if X is a variable with values x1, x2, ..., xn, the transformation can be achieved using the following formula:

$$x_i' = \frac{x_i - \bar{X}}{std_{dev}(\bar{X})}$$

where x_i' is the Z-score value of x_i, $\bar{X}$ is the mean of the values of X, and $\mathrm{std}_{\mathrm{dev}}$ is the standard deviation given by

$$std_{dev}(\bar{X}) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})^2}$$

3.5.3 Decimal-scaling normalization

Decimal scaling is a normalization technique that involves shifting the decimal point of an attribute's value. The number of decimal points shifted depends on the maximum absolute value of the attribute. To normalize a value v of an attribute to v' , we divide v by 10^{j} , where j is the smallest integer such that max(v') < 1.

For example, if the values of an attribute range from -523 to 237, the maximum absolute value is 523. To normalize using decimal scaling, we would divide each value by 1000 (i.e., j = 3) so that -523 normalizes to -0.523 and 237 normalizes to 0.237.

3.5.4 Quantile normalization

Quantile normalization is a method used to make multiple distributions statistically identical. To normalize a column (which represents an attribute), each value in the column is ranked based on its position from lowest to highest. The column values are then rearranged in ascending order so that each column is in order from the lowest to the highest value. The average value for each row is then calculated using the reordered values. The average value calculated in the first row is the lowest value (rank 1) from every column, the second average value is the second-lowest (rank 2), and so on. Finally, the original values are replaced with the average values based on their assigned ranks. This results in each column having the same distribution of values.

Quantile normalization assumes that there are no global differences in the distribution of data in each column. However, if these assumptions are violated, it is not clear how to proceed with normalization. To address this issue, a recent technique called smooth quantile normalization (qsmooth) has been proposed. Qsmooth is a generalization of quantile normalization that allows for differences in the distribution between groups by relaxing some of the assumptions made by quantile normalization.

3.5.5 Logarithmic normalization

In biomedical and psychosocial research, log-normalization is a popular technique used to address data skewness and heteroskedasticity. The purpose of log-normalization is to normalize the distribution of continuous data and make it more similar to a normal distribution, which reduces or eliminates input-data skewness. This technique involves a simple scaling normalization called the log transformation, which replaces each value x with its logarithm, denoted by x' = log(x). The choice of the log base depends on the specific problem being addressed. For instance, the base 2 logarithm, denoted by $\log_2$, is commonly used for expression data normalization in microarray experiments in computational biology. Log-normalization is easy to apply and has proven to be beneficial for statistical modeling.

3.6 Data integration

If data are obtained from only a single source, it is often difficult to make a reliable decision. The use of multiple sources of data is preferred. Multisource data may be stored in various storage repositories such as databases, flat files, and multidimensional data cubes. Often, the data structures and storage formats for the sources are different. To process, one needs to bring all data to a common format. Data integration is the process of analyzing data obtained from various sources, and obtaining a common format that can be used across all sources. Such a format usually makes more accurate predictions.

Data integration can be achieved at three different levels: data level, processing level, and decision level. Combining data from various sources and implementing a universal query system for all types of data involved is required to integrate data at the data level. This step is the most time consuming, as it requires considering various assumptions and experimental setup information before combining the data. The second level of data integration involves understanding and interpreting the datasets to identify associated correlations between the various datasets involved. The third level of data integration is at the decision level, where specific procedures are used to deal with different participating datasets and obtain individual results. These individual results are then mapped through a consensus process called "Ensembling".

Data integration involves processing steps like cleansing, sorting, enrichment, and other processes before storage. The above preprocessing steps may be applied before or after porting the data to the destination storage, termed as ETL (Extract, Transform, Load) or ELT (Extract, Load, Transform). When data from different sources are ported to common destination entirely through the process of ETL/ELT, it is called Tight Coupling. Instead of porting, if the integration happens on-the-fly through query-based data selection in the source storage for a consolidated view of the data, it is called Loose Coupling. Following are a couple of techniques for data integration.

3.6.1 Consolidation

Data consolidation is a tight coupling approach for data integration. It brings data from different sources to the centralized destination pool or data store. The centralized or consolidated storage is then used for downstream data-analytic tasks. An issue with the consolidation process is the latency in collecting updated data from multiple sources of storage to update the destination storage. Latency should be lower for more recent data in the centralized storage. Although the entire process of consolidation is expensive, with the advancement of integration technologies, it is now possible for near real-time upgradation.

3.6.2 Federation

Unlike consolidation, federation does not involve any physical movement of data. Rather, a virtual data storage is used to unify multiple data sources using a variety of data models. It is a type of loose coupling integration. It is an on-demand process of integration based on user query. When, a user issues a query, data are fetched from heterogeneous data sources by splitting the query into versions understandable by the candidate source storage. Consolidation of data sources in advance is avoided in a federation data-integration model.

3.7 Steps of Data Science Process

3.7.1 Data collection

Data are initially collected, and integrated if collection involves multiple sources. For any successful Data Science and -analysis activity, data collection is one of the most important

steps. The quality of collected data carries great weight. If the collected samples are not sufficient to describe the overall system or process under study, downstream activities are likely to become useless despite employing sophisticated computing methods. The quality of the outcome is highly dependent on the quality of data collection.

3.7.2 Pre-processing

Raw data collected from input sources are not always suitable for downstream exploration. The presence of noise and missing values, and the prevalence of nonuniform data structures and standards may negatively affect final decision making. Hence, it is of utmost importance to prepare the raw data before downstream processing. Preprocessing also filters uninformative or possibly misleading values such as outliers.

3.7.3 Training

The preprocessed data are divided into training and testing sets. The training set is used to build a machine learning model for prediction or classification tasks. The training data contain the attributes-class in the dataset. A machine learning model is normally trained on the instances containing these attributes and class. Different machine learning models are suitable for learning different types of data patterns. Iterative learning via refinement is often more successful in understanding data distributions. A plethora of models are available to a data scientist and choices must be made judiciously. Models are usually used to explain the data or extract relevant patterns to describe the data or predict associations

3.7.4 Testing

The testing dataset is used to evaluate the performance of the trained machine learning model. The testing set can be randomly selected from the original dataset to contain only instances with no class label. The machine learning model is expected to predict the class label of any instances based on the previous training. The accuracy and error of the prediction using the testing data are normally measured.

3.8 Statistics

Statistics is concerned with scientific methods for collecting, organizing, summarizing, presenting, and analyzing data as well as with drawing valid conclusions and making reasonable decisions on the basis of such analysis. The field of statistics is concerned with the collection, description, and interpretation of data (data are numbers obtained through measurement). In the field of statistics, the term “statistic” denotes a measurement taken on a sample (as opposed to a population). In general conversation, “statistics” also refers to facts and figures.

Statistics and probability are essential as these branches of science lead the basic foundation of all data science domains, i.e., artificial intelligence, machine learning algorithms, and deep learning. Statistics and probability are foundations to execute the concepts of trending technologies. Mathematics is embedded in every aspect of our lives, i.e., from shapes, patterns, and colors to the count of petals in flowers.

Sample

In collecting data concerning the characteristics of a group of individuals or objects, such as the heights and weights of students in a university or the numbers of defective and nondefective bolts produced in a factory on a given day, it is often impossible or impractical to observe the entire group, especially if it is large. Instead of examining the entire group, called the population, or universe, one examines a small part of the group, called a sample.

Population

A population can be finite or infinite. For example, the population consisting of all bolts produced in a factory on a given day is finite, whereas the population consisting of all possible outcomes (heads, tails) in successive tosses of a coin is infinite.

Inductive/ Inferential statistics

If a sample is representative of a population, important conclusions about the population can often be inferred from analysis of the sample. The phase of statistics dealing with conditions under which such inference is valid is called inductive statistics, or statistical inference. Because such inference cannot be absolutely certain, the language of probability is often used in stating conclusions.

Deductive/ Descriptive statistics

The phase of statistics that seeks only to describe and analyze a given group without drawing any conclusions or inferences about a larger group is called descriptive, or deductive, statistics.

Graphs

A graph is a pictorial presentation of the relationship between variables. Many types of graphs are employed in statistics, depending on the nature of the data involved and the purpose for which the graph is intended. Among these are bar graphs, pie graphs, pictographs, etc. These graphs are sometimes referred to as charts or diagrams.

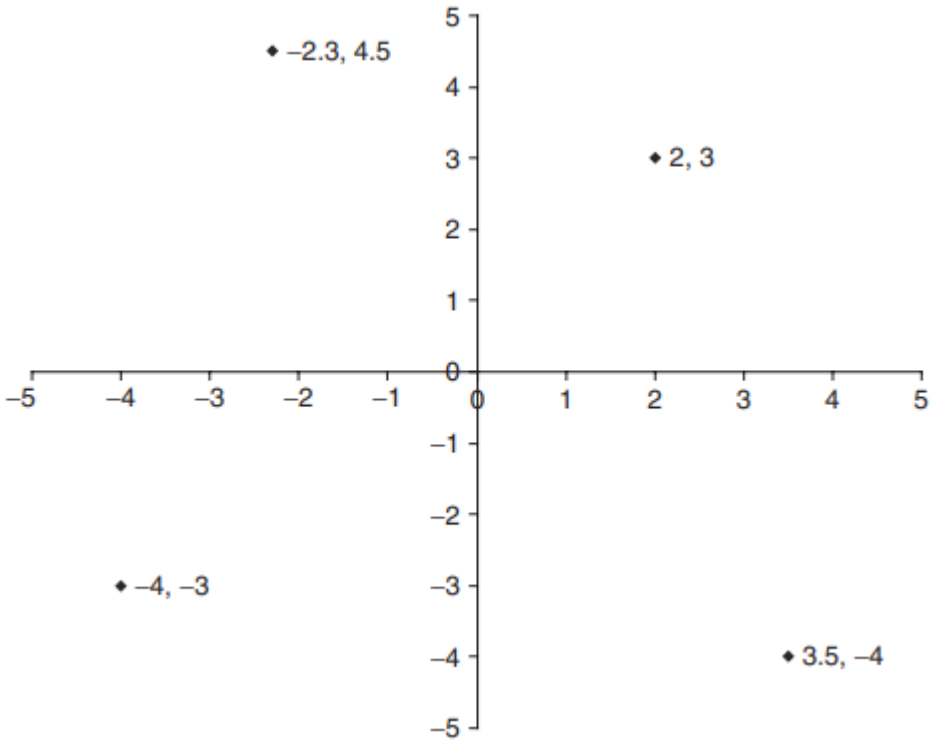


Figure 3.2 An EXCEL plot of points in the four quadrants.

3.8.1 Distributions

A statistical distribution describes the numbers of times each possible outcome occurs in a sample. If you have 10 test scores with 5 possible outcomes of A, B, C, D, or F, a statistical distribution describes the relative number of times an A,B,C,D or F occurs. For example, 2 A's, 4 B's, 4 C's, 0 D's, 0 F's.

Raw Data

Raw data are collected data that have not been organized numerically. An example is the set of heights of 100 male students obtained from an alphabetical listing of university records.

Arrays

An array is an arrangement of raw numerical data in ascending or descending order of magnitude. The difference between the largest and smallest numbers is called the range of the data. For example, if the largest height of 100 male students is 74 inches (in) and the smallest height is 60 in, the range is 74 - 60 = 14 in.

Frequency Distributions

When summarizing large masses of raw data, it is often useful to distribute the data into classes, or categories, and to determine the number of individuals belonging to each class, called the class frequency. A tabular arrangement of data by classes together with the corresponding class frequencies is called a frequency distribution, or frequency table. Table 3.1 is a frequency distribution of heights (recorded to the nearest inch) of 100 male students at XYZ University.

Table 3.1 Heights of 100 male students at XYZ University

Height (in)	Number of Students
60–62	5
63–65	18
66–68	42
69–71	27
72–74	8
	Total 100

The first class (or category), for example, consists of heights from 60 to 62 in and is indicated by the range symbol 60–62. Since five students have heights belonging to this class, the corresponding class frequency is 5.

Data organized and summarized as in the above frequency distribution are often called grouped data. Although the grouping process generally destroys much of the original detail of the data, an important advantage is gained in the clear ‘‘overall’’ picture that is obtained and in the vital relationships that are thereby made evident.

Class Intervals and Class Limits

A symbol defining a class, such as 60–62 in Table 3.1, is called a class interval. The end numbers, 60 and 62, are called class limits; the smaller number (60) is the lower class limit, and the larger number (62) is the upper class limit. The terms class and class interval are often used interchangeably, although the class interval is actually a symbol for the class.

A class interval that, at least theoretically, has either no upper class limit or no lower class limit indicated is called an open class interval. For example, referring to age groups of individuals, the class interval ‘‘65 years and over’’ is an open class interval.

Class Boundaries

If heights are recorded to the nearest inch, the class interval 60–62 theoretically includes all measurements from 59.5000 to 62.5000 in. These numbers, indicated briefly by the exact

numbers 59.5 and 62.5, are called class boundaries, or true class limits; the smaller number (59.5) is the lower class boundary, and the larger number (62.5) is the upper class boundary.

In practice, the class boundaries are obtained by adding the upper limit of one class interval to the lower limit of the next-higher class interval and dividing by 2.

Sometimes, class boundaries are used to symbolize classes. For example, the various classes in the first column of Table 3.1 could be indicated by 59.5–62.5, 62.5–65.5, etc. To avoid ambiguity in using such notation, class boundaries should not coincide with actual observations. Thus, if an observation were 62.5, it would not be possible to decide whether it belonged to the class interval 59.5–62.5 or 62.5–65.5.

The Size, or Width, of a Class Interval

The size, or width, of a class interval is the difference between the lower and upper class boundaries and is also referred to as the class width, class size, or class length. If all class intervals of a frequency distribution have equal widths, this common width is denoted by c. In such case c is equal to the difference between two successive lower class limits or two successive upper class limits. For the data of Table 3.1, for example, the class interval is c = 62:5 - 59:5 = 65:5 - 62:5 = 3.

The Class Mark

The class mark is the midpoint of the class interval and is obtained by adding the lower and upper class limits and dividing by 2. Thus, the class mark of the interval 60–62 is (60 + 62)/2 = 61. The class mark is also called the class midpoint.

For purposes of further mathematical analysis, all observations belonging to a given class interval are assumed to coincide with the class mark. Thus, all heights in the class interval 60–62 in are considered to be 61 in.

3.8.1.1 Histograms and Frequency Polygons

Histograms and frequency polygons are two graphic representations of frequency distributions.

1. A histogram or frequency histogram, consists of a set of rectangles having (a) bases on a horizontal axis (the X axis), with centers at the class marks and lengths equal to the class interval sizes, and (b) areas proportional to the class frequencies.

2. A frequency polygon is a line graph of the class frequencies plotted against class marks. It can be obtained by connecting the midpoints of the tops of the rectangles in the histogram.

The histogram and frequency polygon corresponding to the frequency distribution of heights in Table 3.1 are shown in Figures 3.3 and 3.4.

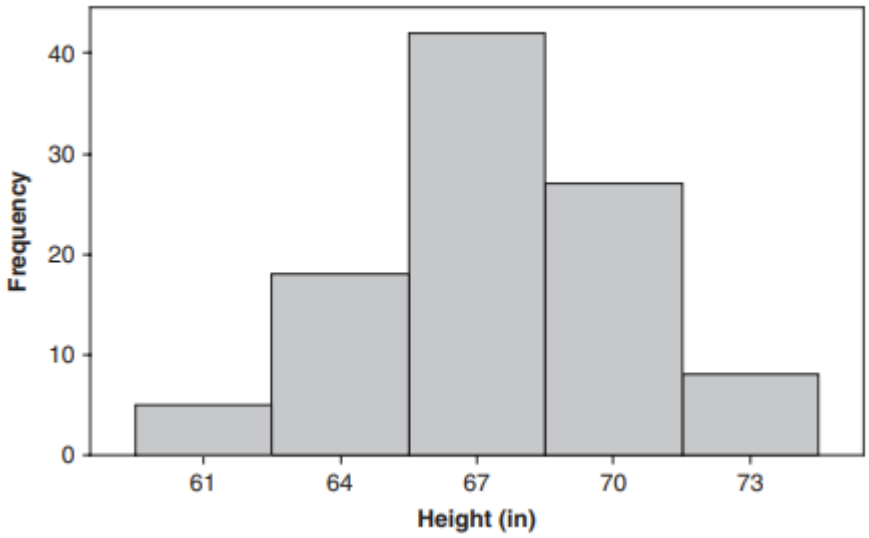


Figure 3.3. A histogram showing the class midpoints and frequencies.

3.8.1.2 Relative-Frequency Distributions

The relative frequency of a class is the frequency of the class divided by the total frequency of all classes and is generally expressed as a percentage. For example, the relative frequency of the class 66–68 in Table 3.1 is 42/100 =42%. The sum of the relative frequencies of all classes is clearly 1, or 100%.

If the frequencies in Table 3.1 are replaced with the corresponding relative frequencies, the resulting table is called a relative-frequency distribution, percentage distribution, or relative-frequency table.

Graphic representation of relative-frequency distributions can be obtained from the histogram or frequency polygon simply by changing the vertical scale from frequency to relative frequency, keeping exactly the same diagram. The resulting graphs are called relative-frequency histograms (or percentage histograms) and relative-frequency polygons (or percentage polygons), respectively.

3.8.1.3 Cumulative-Frequency Distributions and Ogives

The total frequency of all values less than the upper class boundary of a given class interval is called the cumulative frequency up to and including that class interval. For example, the cumulative frequency up to and including the class interval 66–68 in Table 3.1 is 5 +18 + 42 = 65, signifying that 65 students have heights less than 68.5 in.

A table presenting such cumulative frequencies is called a cumulative-frequency distribution, cumulative-frequency table, or briefly a cumulative distribution. A graph showing the cumulative frequency less than any upper class boundary plotted against the upper class boundary is called a cumulative-frequency polygon or ogive.

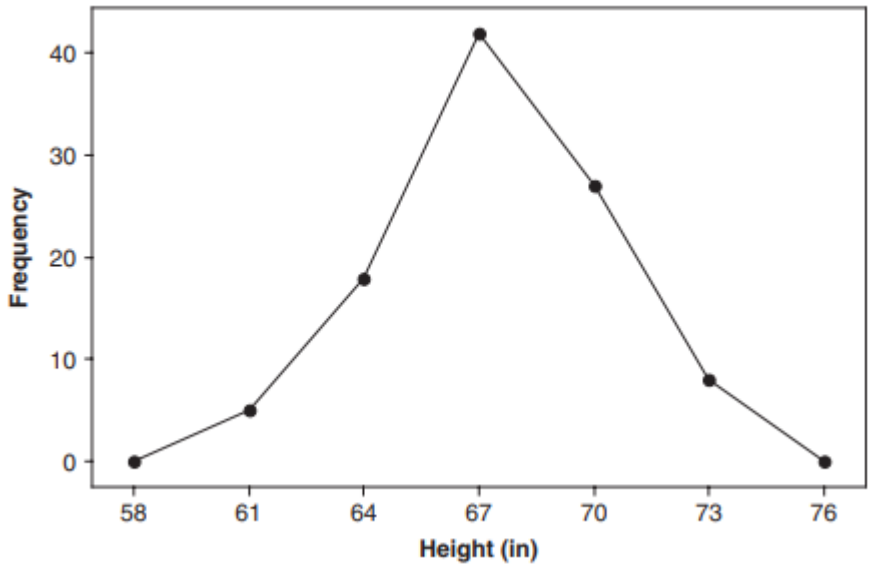


Figure 3.4 A frequency polygon of the student heights.

3.8.1.4 Relative Cumulative-Frequency Distributions and Percentage Ogives

The relative cumulative frequency, or percentage cumulative frequency, is the cumulative frequency divided by the total frequency. For example, the relative cumulative frequency of heights less than 68.5 in is 65/100 = 65%, signifying that 65% of the students have heights less than 68.5 in. If the relative cumulative frequencies are used in place of cumulative frequencies, the results are called relative cumulative-frequency distributions (or percentage cumulative distributions) and relative cumulative frequency polygons (or percentage ogives), respectively.

3.8.2 Measures of Central Tendency

Suppose we have a sample with the following 4 observations: 4, 1, 4, 3.

Mean - the sum of a set of numbers divided by the number of observations.

Mean = $\frac{4+1+4+3}{4} = \frac{12}{4} = 3$

Median - the middle point of a set of numbers (for odd numbered samples).

Median = 1,3,4,4 or $\frac{3+4}{4} = \frac{7}{2} = 3.5$

the mean of the middle two points (for even samples).

Mode - the most frequently occurring number.

Mode = 4 (4 occurs most).

The mean, median and mode are called measures of central tendency.

3.8.3 Measures of Variation

Range - the maximum value minus the minimum value in a set of numbers.

Range = 4-1 = 3.

Standard Deviation - the average distance a data point is away from the mean.

Standard deviation = $\frac{|4-3|+|1-3|+|4-3|+|3-3|}{4} = \frac{1+2+1+0}{4} = \frac{4}{4} = 1$

Standard deviation computes the difference between each data point and the mean. Take the absolute value of each difference. Sum the absolute values. Divide this sum by the number of data points. Median: first arrange data points in increasing order.

Mean, Median, Mode, Range, and Standard Deviations are measurements in a sample (statistics) and can also be used to make inferences on a population.

3.8.4 Data Distribution (visualization) in Graphs

- **Bar graphs** use bars to compare frequencies of possible data values.

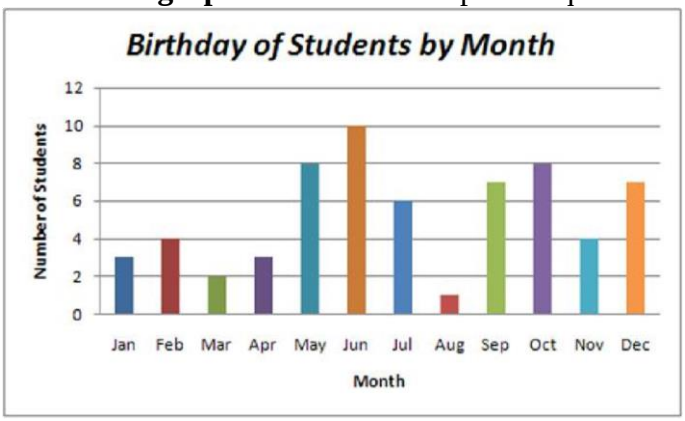


Figure 3.5: Bar graph

- **Double bar graphs** use two sets of bars to compare frequencies of data values between two levels of data.

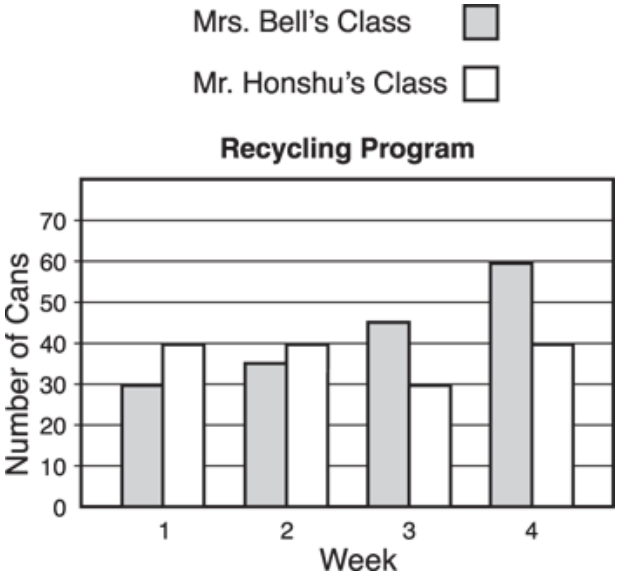


Figure 3.6: Double Bar graph

- **Histograms** use bars to show how frequently data occur within equal spaces within an interval.

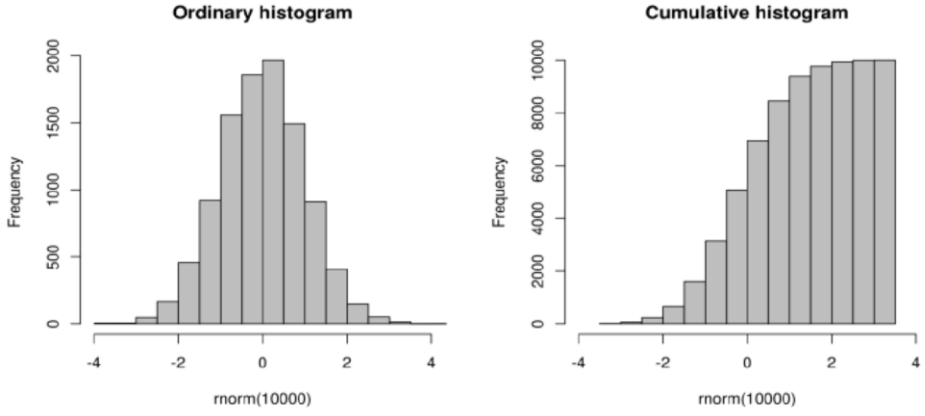


Figure 3.7: Histogram

- **Pie Charts** use portion of a circle to show contributions of data values.

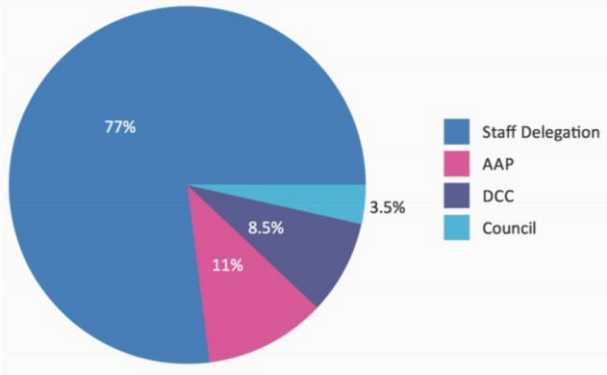


Figure 3.8: Histogram

3.9 Statistical Models in Simulation

Introduction

In modeling real-world phenomena, there are few situations where the actions of the entities within the system under study can be predicted completely. The world the model-builder sees is probabilistic rather than deterministic. There are many causes of variation. The time it takes a repairperson to fix a broken machine is a function of the complexity of the breakdown, whether the repairperson brought the proper replacement parts and tools to the site, whether another repairperson asks for assistance during the course of the repair, whether the machine operator receives a lesson in preventive maintenance, and so on. To the model-builder, these variations appear to occur by chance and cannot be predicted. However, some statistical model might well describe the time to make a repair.

Review of Terminology and Concepts

1. **Discrete random variables**. Let X be a random variable. If the number of possible values of X is finite, or countably finite, X is called a discrete random variable. The possible values of X may be listed as x_1, x_2 ... In the finite case, the list terminates; in the countably infinite case, the list continues indefinitely.

Example 1

The number of jobs arriving each week at a job shop is observed. The random variable of interest is X, where

X = number of jobs arriving each week

The possible values of X are given by the range space of X, which is denoted by R_x. Here R_x= {0, 1, 2, ... } .

Let X be a discrete random variable. With each possible outcome x_i in R_x, a number p(x_i) = P(X = x_i) gives the probability that the random variable equals the value of x_i. The numbers p(x_i), i = 1, 2, . . . , must satisfy the following two conditions:

1. $p(x_i) \geq 0$, for all i
2. $\sum_{i=1}^{\infty} p(x_i) = 1$

The collection of pairs (x_i, $p(x_i)$), i = 1, 2, . . . is called the probability distribution of X, and $p(x_i)$ is called the probability mass function (pmf) of X.

Example 2

Consider the experiment of tossing a single die. Define X as the number of spots on the up face of the die after a toss. Then R_x = {1, 2, 3, 4, 5, 6}. Assume the die is loaded so that the

probability that a given face lands up is proportional to the number of spots showing. The discrete probability distribution for this random experiment is given by

x_i	1	2	3	4	5	6
$p(x_i)$	1/21	2/21	3/21	4/21	5/21	6/21

The conditions stated earlier are satisfied- that is, $p(x_i) \geq 0$ for i = 1, 2,. . ., 6 and $\sum_{i=1}^{\infty} p(x_i) = 1/21 + 2/21 + \ldots + 6/21 = 1$.

2. **Continuous random variables.** If the range space R_x of the random variable X is an interval or a collection of intervals, x_i is called a continuous random variable. For a continuous random variable X, the probability that X lies in the interval [a, b] is given by

$$P(a \leq X \leq b) = \int_a^b f(x)dx \quad (1.1)$$

The function $f(x)$ is called the Probability Density Function (PDF) of the random variable X. The pdf satisfies the following conditions:

1. $f(x) \geq 0$ for all x in R_x
2. $\int_{R_x} f(x)dx = 1$
3. $f(x) = 0$ if x is not in R_x

As a result of Equation (1.1), for any specified value x_0, P(X = x_0) = 0, because

$$\int_{x_0}^{x_0} f(x)dx = 0$$

Example 3
The life of a device used to inspect cracks in aircraft wings is given by X, a continuous random variable assuming all values in the range $x \geq 0$. The pdf of the lifetime, in years, is as follows:

$$f(x) = \begin{cases} 1/2e^{-x/2}, x \geq 0 \\ 0, \quad \text{otherwise} \end{cases}$$

What is the probability that the life of the device is between 2 and 3 years?

Solution

$$P(2 \leq X \leq 3) = \frac{1}{2}\int_2^3 e^{-x/2}\,dx$$
$$= \frac{1}{2}\left[e^{-x/2} \div -1/2\right]_2^3$$
$$= -\frac{1}{2} \times \frac{2}{1}\left[e^{-x/2}\right]_2^3$$
$$= \left[-e^{-x/2}\right]_2^3$$

$$= -e^{-3/2} - (-e^{-2/2})$$
$$= -e^{-3/2} + e^{-1} = -0.223 + 0.368 = 0.145$$

3. **Cumulative distribution function**. The Cumulative Distribution Function (CDF), denoted by F(x), measures the probability that the random variable X assumes a value less than or equal to x, that is, F(x) = $P(X \leq x)$.

If X is discrete, then

$$F(x) = \sum_{\substack{all \\ x_i \le x}} p(x_i)$$

If X is continuous, then

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

Some properties of the cdf are listed here:

1. F is a nondecreasing function. If $a < b$, then $F(a) \le F(b)$.
2. $lim_{x\to\infty} F(x) = 1$
3. $lim_{x\to-\infty} F(x) = 0$

All probability questions about X can be answered in terms of the cdf. For example,

$P(a < X \le b) = F(b) - F(a)$ for all $a < b$

Example 4
The cdf for the device described in Example 3 is given by:

$$F(x) = \frac{1}{2}\int_0^x e^{-t/2}\,dt$$
$$= \frac{1}{2}\left[e^{-t/2} \div -1/2\right]_0^x$$
$$= -\frac{1}{\cancel{2}} \times \frac{\cancel{2}}{1}\left[e^{-t/2}\right]_0^x$$
$$= \left[-e^{-t/2}\right]_0^x$$

$$= -e^{-x/2} - (-e^{-0/2})$$
$$= -e^{-x/2} - (-e^{-0})$$
$$= -e^{-x/2} + e^{-0}$$
$$= 1 - e^{-x/2}$$

The probability that the device will last for less than 2 years is given by

$$P(0 \le X \le 2) = F(2) - F(0) = F(2) = 1 - e^{-1} = 0.632$$

The probability that the life of the device is between 2 and 3 years is calculated as

$$P(2 \le X \le 3) = F(3) - F(2) = \left(1 - e^{-3/2}\right) - (1 - e^{-1})$$
$$= -e^{-3/2} + e^{-1} = -0.223 + 0.368 = 0.145$$

4. **Expectation**. An important concept in probability theory is that of the expectation of a random variable. If X is a random variable, the expectation value of X, denoted by E(X), for discrete and continuous variables is defined as follows:

$$E(X) = \sum_{all\ i} x_i p(x_i) \qquad \text{if X is discrete}$$

and

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx \qquad \text{if X is continous}$$

The expected value E(X) of a random variable X is also referred to as the mean, μ, or the first moment of X. The quantity E(X^n), $n \geq 1$, is called the nth moment of X, and is computed as follows:

$$E(X^n) = \sum_{all\ i} x_i^n p(x_i) \qquad \text{if X is discrete}$$

and

$$E(X^n) = \int_{-\infty}^{\infty} x^n f(x) dx \qquad \text{if X is continuous}$$

The variance of a random variable, X, denoted by V(X) or var(X) or σ^2, is defined by

$$V(X) = E[(X - E[X])^2]$$

A useful identity in computing V(X) is given by

$$V(X) = E(X^2) - [E(X)]^2$$

The mean E(X) is a measure of the central tendency of a random variable. The variance of X measures the expected value of the squared difference between the random variable and its mean. Thus, the variance, V(X), is a measure of the spread or variation of the possible values of X around the mean E(X). The standard deviation, σ, is defined to be the square root of the variance, σ^2. The mean, E(X), and the standard deviation, $\sigma = \sqrt{V(X)}$, are expressed in the same units.

Example 5

The mean and variance of the die-tossing experiment (described in Example 2 are computed as follows:

$$E(X) = 1\left(\frac{1}{21}\right) + 2\left(\frac{2}{21}\right) + \cdots + 6\left(\frac{6}{21}\right) = \frac{91}{21} = 4.33$$

To compute V(X), first compute $E(X^2)$ as follows:

$$E(X^2) = 1^2\left(\frac{1}{21}\right) + 2^2\left(\frac{2}{21}\right) + \cdots + 6^2\left(\frac{6}{21}\right) = 21$$

Thus,

$$V(X) = 21 - \left(\frac{91}{21}\right)^2 = 21 - 18.78 = 2.22$$

and

$$\sigma = \sqrt{V(X)} = 1.49$$

Example 6

The mean and variance of the life of the device described in Example 3 are computed as follows:

$$E(X) = \frac{1}{2}\int_0^\infty xe^{-x/2}\,dx = -xe^{-x/2}|_0^\infty + \int_0^\infty e^{-x/2}dx$$

$$= 0 + \frac{1}{1/2}e^{-x/2}|_0^\infty = 2 \text{ years}$$

To compute V(X), first compute $E(X^2)$ as follows:

$$E(X^2) = \frac{1}{2}\int_0^\infty x^2 e^{-x/2}\,dx$$

Thus,

$$E(X^2) = -x^2e^{-x/2}|_0^\infty + 2\int_0^\infty xe^{-x/2}\,dx = 8$$

giving

$$V(X) = 8 - 2^2 = 4 \text{ years}$$

and

$$\sigma = \sqrt{V(X)} = 2 \text{ years}$$

With a mean life of 2 years and a standard deviation of 2 years, most analysts would conclude that actual lifetimes, X, have a fairly large variability.

5. **The mode**. The mode is used in describing several statistical models that appear in this chapter. In the discrete case, the mode is the value of the random variable that occurs most frequently. In the continuous case, the mode is the value at which the pdf is maximized. The mode might not be unique; if the modal value occurs at two values of the random variable, the distribution is said to be bimodal.

3.8 Summary

To deal with the incomplete and imperfect nature of real-life data, data-preparation steps are necessary. These steps are crucial to handle different forms of incompleteness and imperfection, including missing values, noise, inconsistencies, and the curse of dimensionality. However, the nature of real data is not static and data distributions vary over time, making it challenging to apply state-of-the-art preprocessing techniques. This challenge is particularly pronounced when data are rapidly produced. Therefore, the Big Data community is interested in conducting an empirical study to assess the suitability and efficiency of preprocessing techniques for handling dynamic data. We use Statistical distributions to: investigate how a change in one variable relates to a change in a second variable, represent situations with numbers, tables, graphs, and verbal descriptions, understand measurable attributes of objects and their units, systems, and processes of measurement, identify relationships among attributes of entities or systems and their association.

Lesson 4: Machine learning

4.1 Introduction

Data Science aims to identify novel patterns in large collections of data. The discovered patterns often help in quick and smart decision making. For instance, being a frequent user of any email service such as Gmail, one receives numerous important and unimportant emails daily. To deal with a large volume of incoming emails, one may opt to use a facility that filters unwanted emails automatically as Spam without explicit human intervention. Various marketing and social-media-related intended and unintended communications that are pushed can be screened automatically to keep the inbox clean. Automated email recognition and categorization guarantee efficient time utilization for users who need to apportion time to other activities. Likewise, in healthcare, smart MRI analysis can detect brain tumors automatically, or a smartwatch can analyze health status in real time and raise an alarm in advance to warn about possible health issues.

In conventional programming, programs make decisions based on algorithms. However, it has been found impossible to write a general program to recognize credit-card fraud based on rules obtained by analyzing how someone uses the card. Similarly, just by analyzing network traffic based on rules or algorithms written a priori, it is impossible to recognize whether it is normal or abnormal. This is because the number and/or complexity of rules or algorithms are context dependent and become bulky. We need a program that automatically learns contextual rules or parameters (say, individualized thresholds) for recognizing fraud, network attacks, or spam emails. The program itself attempts to identify patterns from past instances or experiences for the environment. In other words, the data are used to identify the relevant patterns in what is often known as data-driven decision making. Machine Learning is a subdomain of Artificial Intelligence (AI) that deals with developing algorithms that let a program automatically learn inherent patterns in the given data. Mathematically, it produces an implicit or explicit function that is used for recognition or prediction in the future. During the learning phase, the algorithm enhances its own performance on a particular activity as it is exposed to more data or experiences without being explicitly programmed.

4.2 Machine Learning paradigms

Depending on the type of experience data the underlying learning model uses, machine learning approaches can be categorized into three types: supervised, unsupervised, and semisupervised. Let us briefly introduce each of these three learning approaches.

4.2.1 Supervised learning

A set of instances, along with their labels called class labels, are provided to train a learning model. Note that the instances may or may not be given as drawings but in terms of some attributes or features whose values may or may not be numeric. The algorithm uses the labeled training data to learn patterns and relationships between input features and target outputs, enabling it to predict the labels of new, unseen test instances with the knowledge gained from the labeled training data. During testing, labels of the test instances are kept hidden from the learning algorithm to match how accurately the algorithm can predict the labels based on its prior learning.

To start the learning process, the raw rows or instances gathered or captured are preprocessed and usually split into three subsets, training, validation, and testing, as shown in Figure 4.1. The training set is used for training the model, and the validation set is used for evaluating the performance of the trained model to improve the trained model. The ultimate purpose of a supervised-learning method is to predict a class instance with the best possible accuracy for an unlabeled test instance. The validation set is used throughout the model's training phase to

evaluate the model's performance and fine tune the model's parameters. On the other hand, the test set is utilized once the model has been fully trained to evaluate the model's performance on data that has never been seen before. Once it achieves satisfactory performance, the learned model prepares to predict the category of unseen objects. There is always scope for fine tuning the model's training by tuning the parameters of the machine-learning algorithm or by restarting the whole process with different combinations of steps. Figure 4.1 presents the overall process of a supervised-learning method. There are five major steps: (i) Capturing or gathering raw data, (ii) Preprocessing to make the data ready for machine-learning algorithms, (iii) Splitting the preprocessed data into three parts: training, validation, and testing, (iv) Learning the classification model from the training data and fine tuning on the validation data, and finally (v) Evaluating the trained model on the testing set, to check the performance of the method. If the performance is not satisfactory in terms of accuracy, further parameter tuning may be performed. In some situations, reperforming all or some of the prior steps with different combinations of alternatives may improve the prediction performance. Once the model performs satisfactorily, it may be deployed for real-world application. However, a good prediction system should be able to adapt to dynamic real-world scenarios by updating itself.

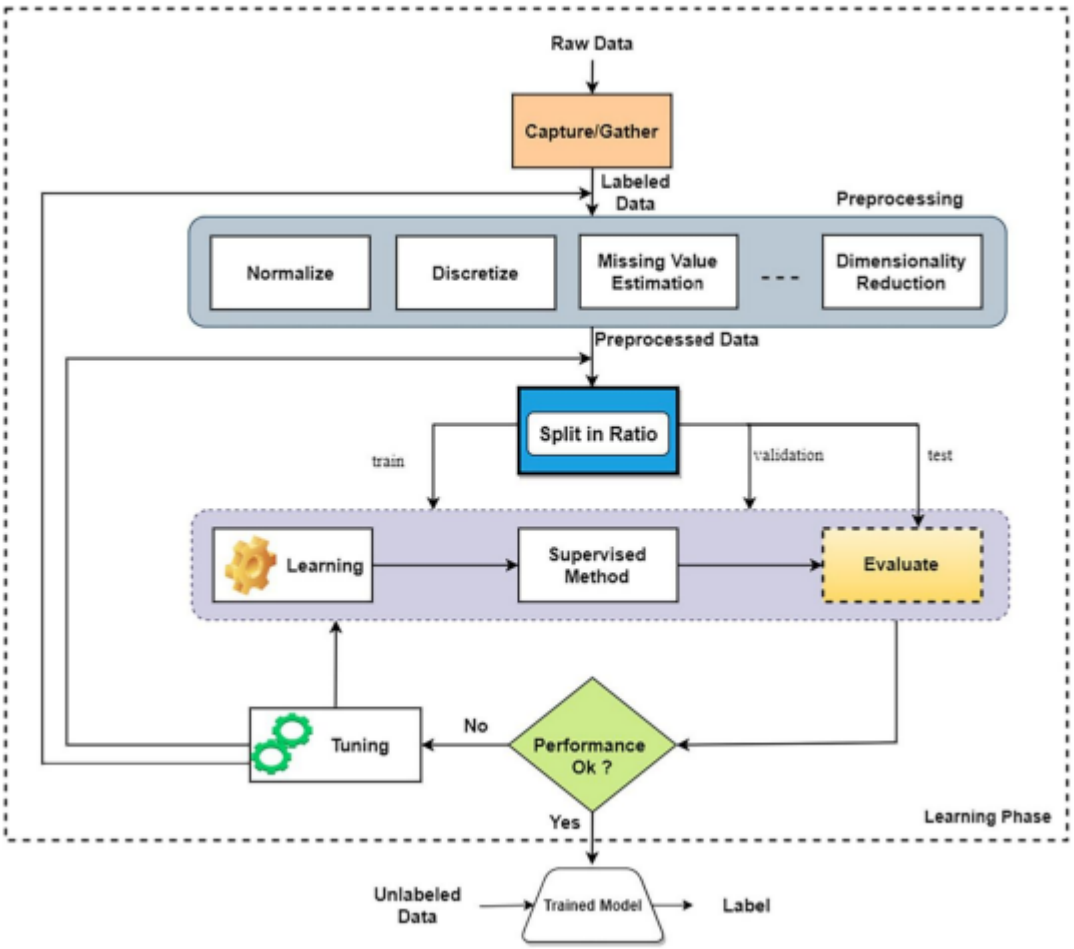


Figure 4.1. Typical workflow of a supervised-learning paradigm

4.2.2 Unsupervised learning

Unlike supervised learning, no prior knowledge (in terms of labeled data) is available during unsupervised learning. An unsupervised method attempts to learn patterns from the features that can be used to group or cluster unlabeled data instances in some natural or useful way without the intervention of human expert(s). It is also known as exploratory data analysis. More precisely, it involves a machine-learning algorithm that learns patterns from an unlabeled dataset in the absence of corresponding output or target variables.

The algorithm explores the inherent relationships and similarities within the data, aiming to discover clusters, patterns, or representations that can provide valuable insights into the underlying structure of the data. Unlike supervised learning, where labeled data guides the learning process, unsupervised learning relies solely on the input data to autonomously identify meaningful patterns and extract valuable information.

The input to an unsupervised method is a set of objects of various shapes, usually given in terms of some descriptive features—without any labels (the pictures may be given in some cases). The task is to group similar objects so that the intragroup similarity among the members of a group is high, whereas, for a pair of instances from two different groups, the similarity is significantly less. It may lead to grouping such as (i) shapes with three corners, (ii) shapes with four corners, (iii) shapes with five corners, and (iv) shapes with no corners, or the groups may be unexpected. For example, if similarly shaped objects had different colors, the objects may get grouped by color and not by shape. In designing an unsupervised method, proximity measures play a crucial role. The precision of cluster analysis given by an unsupervised method depends on the effectiveness (or expressiveness) of the proximity measure used. The success of such measures is also highly influenced by the data types, dimensionality, number of instances, and purpose of use (e.g., whether to capture trend or proximity). Input-order independence, effective noise handling, border object handling, the ability to recognize clusters of any shape and low dependence on algorithmic parameters are desired qualities of such methods.

4.2.3 Semisupervised learning

Obtaining many labeled instances for all the classes is difficult in real life. When only a limited number of labeled training instances are available, how to develop a prediction model that ensures the best possible accuracy remains a challenging issue. Another issue with supervised learning is that it requires hand labeling a large number of training instances by domain experts, incurring large costs. Unsupervised learning groups instances, but the obtained groups may not be consistent with actual classes present in the data, and hence may not be very useful. To address these issues, semisupervised learning has been found to be useful. Semisupervised learning holds an intermediate position between supervised and unsupervised learning. Such approaches attempt to learn how to predict the class labels for unlabeled instances using labeled and unlabeled training instances. Usually, the semisupervised approach is useful when unlabeled instances are more numerous than labeled instances. A small set of labeled instances are used to train any supervised model and, later, the same trained model is used to label a large pool of unlabeled instances.

In one iteration, labeled objects with high confidence scores are added to the training input for continued learning and model improvement. The process is iterated until as many unlabeled instances as possible can be utilized for the overall performance improvement of the model. Although the idea of semisupervised learning is effective, when the amount of labeled data is extremely limited, there is a high probability that the model will overfit the training data and generate inaccurate labels. This can result in the entire model being highly erroneous. Therefore, it becomes crucial to establish a confidence threshold to determine which labeled instances should be included in the retraining process.

4.3 Evaluating a classifier

A crucial part of machine learning is evaluating the performance of the learning algorithm. We discuss here how supervised machine-learning algorithms, particularly classifiers, are evaluated.

Assessing the performance of classifiers is a fundamental aspect of machine learning. Once we have trained a classification algorithm and performed classification tasks on examples the classifier has not been trained, we need to know how good our classifier is. Evaluation of classification is also important to compare various classifiers we may have as candidates for a task at hand. It involves quantifying how well a classifier distinguishes between different classes in a dataset. Researchers and practitioners can gain insights into a classifier's performance by employing a range of evaluation metrics. This evaluation process aids in making informed decisions about model selection, parameter tuning, and understanding the classifier's behavior under different scenarios.

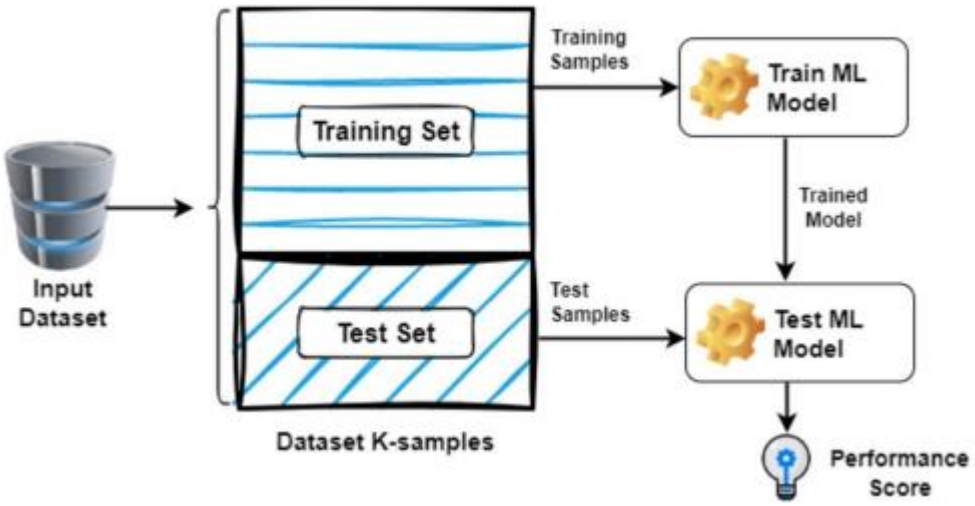


Figure 4.2. A typical Training and Testing workflow of a classifier on the split dataset.

4.3.1 Evaluation steps

First, we need to train our classifier. To train a classifier, we need labeled data or a dataset. For many problems we want to solve, datasets are available and can be found on the web for download. For other tasks, we may need to collect data ourselves. Quite frequently, we may be able to test our classifier on several datasets that are available for download from a website like the UCI Machine Learning Repository or Kaggle. Both sites contain hundreds of datasets that researchers from around the world have collected, performed experiments with, and then submitted so that others can perform experiments and compare results in a consistent manner. Figure 4.2 illustrates the situation. In many datasets, the training and test sets or splits are clearly identified. In such cases, one trains the model on the training set and tests the trained model on the test set, and reports the results. In many machine-learning competitions, only the training set is given. The competition organizers keep the testing set hidden and run the (trained) model uploaded by the participants to obtain results on the testing set and rank participants. Since everyone tests the same way, machine-learning algorithms developed by different individuals or groups can be easily compared.

Performance evaluation should be carried out during the validation and testing phases of a machine-learning model development. These two phases serve distinct purposes in the model-development process.

4.3.1.1 Validation

After the model has been trained using the training dataset, the validation step begins. A separate validation dataset (also known as a development or holdout dataset) is used during this phase to fine tune hyperparameters, select the optimum model architecture, and prevent overfitting (discussed below). Hyperparameters are the external parameters of the machine-

learning model that control how a learning model learns from data. Hyperparameter tuning refers to choosing the ideal settings for hyperparameters to improve a model's performance and capacity to work accurately for unseen data samples. Typically, tuning is carried out by evaluating different combinations on a validation dataset. During the validation phase, performance evaluation assists in making decisions that improve the model's capability to handle new, previously unknown data. Effective performance evaluation ensures that machine-learning models are dependable, resilient, and suitable for real-world scenarios.

4.3.1.2 Testing

The final assessment occurs during the testing phase when the model has been tuned through validation. The test set should be different from both the training and validation sets. The test set acts as an unbiased measure of the model's performance in the real world. The goal of the trained model testing is to determine how well the model will function on entirely fresh, untested data. Measurements made using the testing dataset clearly show the model's actual performance and capacity for new instances. Note that once a model has been trained on the training data and improved iteratively on the validation data, the final trained model is obtained. The final trained model is then tested once on the testing data.

4.3.1.3 K-fold crossvalidation

Often, a dataset is available without any identified training and testing subsets. In such cases, the usual approach is to randomly divide the dataset into K equal groups or parts or folds. In crossvalidation, training and testing sets are not prespecified. The entire dataset is divided randomly into K parts. One part is set aside for testing, and the other K − 1 parts are used for training. This is one execution. For this execution, results are recorded. Experiments are executed considering each of the K parts or folds as a testing set and the rest as a training set. Thus, K experiments are executed. The results from all the executions are averaged and reported. How experiments are performed in crossvalidation is shown in Figure 4.3. The purpose of performing crossvalidation is to remove accidental regularities that may occur in a dataset by coincidence and gain some feel for how a machine-learning algorithm is likely to behave with unseen examples, or in other words, gauge its predictive capability. Since we randomly pick the examples in the folds, the assumption is that any random or accidental associations or correlations among data items are at least partially removed from consideration.

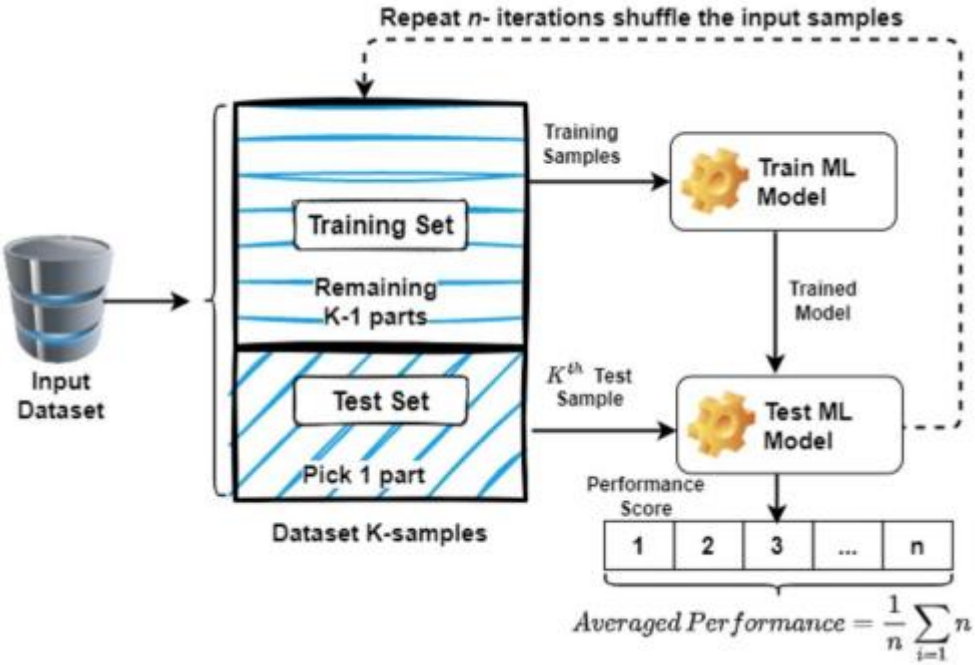


Figure 4.3. K-crossvalidation framework for training and testing.

4.3.2 Handling unbalanced classes

Frequently in a dataset, the examples from the various classes may be unbalanced. For example, suppose that a dataset has examples from two classes c1 and c2, and if 95% of the examples are from one class, say c1, and the classifier does not learn anything and simply says that every example is from class c1, we obtain 95% accuracy. When we work with such unbalanced datasets, we need to take extra care to make sure the results are valid. There are several ways to handle such a situation. One approach is to oversample the smaller class. In other words, when we randomly pick examples to go into our folds, we pick examples from the smaller class with replacement, i.e., we pick the same example several times. Another approach is to create synthetic examples. In this case, for the smaller class, we generate new examples by making small random changes to some of the attribute values in the real examples. When we do so, we must make sure that the new examples created after perturbation are valid examples. We also should know that certain algorithms perform reasonably well on unbalanced datasets compared to others. For example, decision-tree approaches may work well directly on an unbalanced dataset without data augmentation. There are advanced artificial neural techniques such as Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN) that can be used to generate realistic artificial examples for data augmentation, especially when dealing with images.

4.3.3 Model generalization

Building models in machine learning is not just about achieving impressive results on training data. It is about creating models that work well on never encountered, real-world data. The ability of a trained model to perform accurately on new, unseen data that it has not encountered during the training phase is referred to as generalization. When a model generalizes successfully, it means that it has discovered in the training data significant patterns and correlations that it can use to interpret new, previously unexplored data. A model will not perform well on new data if it merely memorizes the training data without grasping the underlying patterns in the general population of data examples, not just the examples that have been sampled for training. Therefore, generalization is crucial. A memorizing model merely recalls the training instances it has seen without truly understanding the underlying principles. Instead of generalizing from the data to understand the underlying relationships and trends, a memorizing model essentially memorizes the individual instances in the training data. It should be clear now why we need different training and test sets. If we train on test data, we have no idea whether the learning model is correctly generalizing or simply memorizing the training examples. However, achieving generalization for a learning algorithm is hindered by two major pitfalls, Underfitting and Overfitting.

4.3.3.1 Underfitting

A machine-learning algorithm is considered to be underfitting when it fails to understand the fundamental patterns or relationships within the data. It performs poorly on both the training data and fresh, previously unseen data because it struggles to grasp the underlying patterns and intricacies present in the data. This typically occurs when there are insufficient training data. Even when we attempt to develop a linear model with insufficient nonlinear data, underfitting results. To understand this better, let us consider a two-class classification problem to classify circles and squares, as shown in Figure 4.4(a). The model is so simple that it tries to create a linear boundary or straight line to separate samples of the two classes. This leads to poor performance during both training and testing. In these situations, the machine-learning model

will likely produce a number of incorrect predictions since its learned rules are too simple and naive to be applied to such sparse data. More data can be used to prevent underfitting, while feature selection can be used to reduce the number of features. As is clear, the simplicity of the fitted model may also be a significant reason for underfitting.

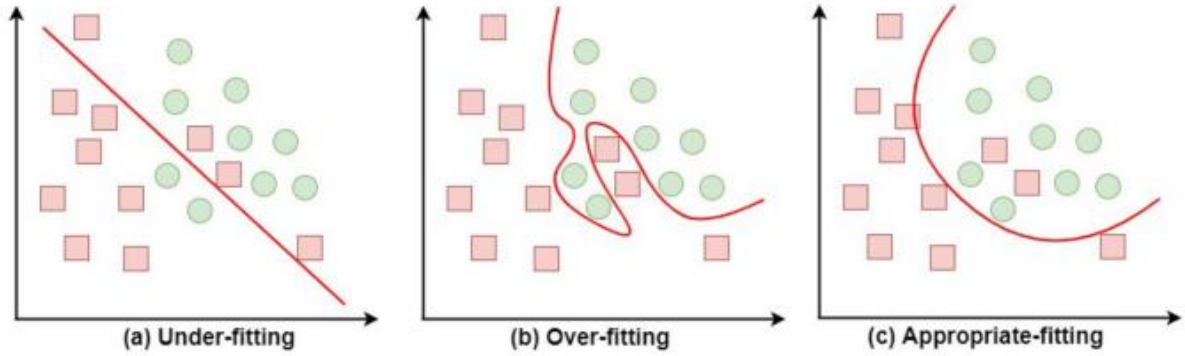


Figure 4.4. For a binary class problem, three trained classifiers create different separation boundaries during training based on training examples. The separation plane in (a) shows underfitting, (b) overfitting when the fitted function is very complex, and (c) appropriate fitting when the fitted function is somewhere in between, leading to better generalization.

4.3.3.2 Overfitting

When a model achieves excellent prediction performance during training but fails to produce reliable predictions on test data, it is said to be Overfitted. Overfitting occurs when a model, trained with an abundance of data, learns not only the meaningful patterns but also the noise and errors present in the dataset. Consequently, during testing, the model displays a high level of variability. Since the model captured patterns corresponding to excess details and noise, the model struggles to properly classify unseen data. Overfitting is particularly noticeable in nonparametric and nonlinear machine-learning algorithms, as they possess greater flexibility to construct models from the dataset. This can inadvertently result in models that diverge from reality. For example, during training for our two-class classification problem, the model creates a separation curve or boundary that perfectly separates all the training instances (Figure 4.4(b)) into true classes. This means the model achieves zero error during training. Such a complex boundary reduces the model's generalization capability, as the separation boundary is created with the involvement of noisy samples or random fluctuations in the training examples. As a result, when this overfitted model encounters new or unseen data, it might struggle to classify accurately because it has incorporated too many characteristic details of a flawed training set. Thus, if the learned model has a very high capacity, it may also overfit. An example of a high-capacity model may be an elaborate artificial neural network with a very large number of weights or parameters.

When hyperparameters are tuned using training data, this might result in overfitting, reducing the generalization capability of the model by making it too specifically customized to the training data's nuances. To put it another way, optimizing hyperparameters purely based on how much they enhance performance on the training set may result in a model that underperforms on fresh data. To overcome this issue, it is common practice to split the available data into three sets: training, validation, and testing. The training set is used to learn new parameters, the validation set aids in hyperparameter tuning, and the testing set evaluates the generalization capability of the model. Data splitting is one of the measures to control overfitting. Using K-fold crossvalidation creates K distinct training and testing sets, making generalization a little more achievable.

In addition to data splitting, early stopping is another measure to counter overfitting. The training is stopped early if the model's performance degrades during validation. This prevents the model from continuing to learn intricate details of the training data that may not generalize well to new data. Regularization is another popular method to control overfitting. These techniques introduce penalties to discourage the model from adopting extreme parameter values.

4.3.3.3 Accurate fittings

The ideal scenario is to have the model with appropriate fitting (Figure 4.4(c)), such that the model performs relatively well during both training and testing. Achieving accurate fitting involves creating a model that can effectively capture the underlying patterns and relationships within the data, both in the training phase and when applied to new, unseen data. To achieve accurate fittings, we must choose a model that is neither highly complex (overfitting) nor too simplistic (underfitting). It is critical to strike the correct balance between these two extremes. Complex models can capture intricate patterns in the training data, but they may have difficulty generalizing, while overly simple models may overlook significant trends. Intuitively, when we try to deal with a nonlinearly separable classification problem, a relatively simple nonlinear boundary (Figure 4.4(c)) may achieve better generalization. A simple nonlinear boundary focuses on capturing the major trends and general shapes of the data distribution without getting caught up in every tiny fluctuation. This prevents the model from becoming too specialized to the training data and helps it generalize better to unseen instances. Hence, allowing a certain degree of misclassification during training is often beneficial for achieving balanced and accurate fitting. While it might seem counterintuitive, striving for zero misclassification on the training data can lead to overfitting.

4.3.4 Evaluation metrics

We can use several metrics to quantify the performance of a classifier. The commonly used metrics are Accuracy, Precision, Recall, and F-measure. We provide the definitions below. Assume that there are only two classes, i.e., it is a binary classifier. To understand the terminology below, consider the first class as the Positive class (e.g., a class called DOG) and the second class as the Negative class (e.g., a class called NOT-DOG).

• True Positives (TP): The set of positive examples in the test set that are classified as positive by the trained program. This is the set of true positives, i.e., positives classified as positive. Let TP be the size of the set of true positives.

• False Negatives (FN): The set of positive examples in the test set that are classified as negative by the trained program. This is the set of false negatives, i.e., positives classified as negative. FN is the number of elements in the set of false negatives.

• True Negatives (TN): The set of negative examples in the test set that are classified as negative by the trained program. This is the set of true negatives, i.e., negatives classified as negative. TN is the size of the set of true negatives.

• False Positives (FP): The set of negative examples in the test set that are classified as positive by the trained program. This is the set of false positives, i.e., negatives classified as positive. FP is the size of the set of false positives.

As an example, if the Positive class corresponds to DOG and the Negative class corresponds to NOT-DOG, TP is the number of actual examples of dogs in the test set that are classified (or predicted) as dogs by a trained classifier. TN is the number of actual test examples of non-dogs that are classified as non-dogs by the trained classifier. FP is the number of tested non-dogs that are classified as dogs, and FN is the number of tested dogs that are classified as non-dogs.

The trained classifier performs correctly in the case of true positives and true negatives, and it performs incorrectly in the case of false positives and false negatives.

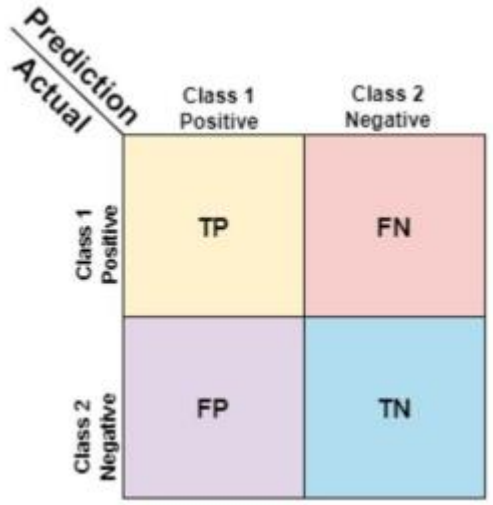


Figure 4.5. A confusion matrix displays numeric results of a classification experiment in one place.

4.3.4.1 Confusion matrix

In the binary confusion matrix (Figure 4.5), the first row contains numbers corresponding to the actual Positive class, and the second row contains numbers corresponding to the Negative class. The first column contains numbers corresponding to the predicted (or, classified by the trained classifier) Positive class, and the second column contains numbers corresponding to the predicted Negative class. Combining the row and column designations, the < 1, 1 > cell contains the number of actual positive test examples predicted as positive (TP). The < 1, 2 > cell contains the number of actual positive test examples classified as negative (FN)—these are error cases. Similarly, the < 2, 1 > cell contains the number of actual negative test examples classified as positive (FP)—these are also error cases. The < 2, 2 > cell contains actual negative test examples classified as negative (TN).

4.3.4.2 Accuracy

Accuracy is defined as the percentage of testing examples classified correctly by a classifier. In other words, for binary classification:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}} \qquad (4.1)$$

The numerator is the total number of examples classified correctly, whether positive examples or negative. The denominator is the total number of examples in the testing dataset. As alluded to earlier, it may not always be a good metric, especially if the classes are unbalanced. If accuracy is 100%, all the examples have been classified correctly. However, as noted earlier, if the classes are very unbalanced, a high accuracy may not indicate that the results are actually good.

Thus, in addition to or in lieu of accuracy, it is common to calculate other metrics for the evaluation of classifiers. Precision, Recall, and F-measure are a few of the many popular metrics. We discuss them below.

4.3.4.3 Precision and recall

Kent et al. were the first to use the concepts of Recall and Precision, albeit the term precision did not come into usage until later in the context of information retrieval. We can compute

precision and recall for each of the two classes, Positive and Negative separately, assuming it is a case of binary classification. The definitions are given below for the Positive (+) class:

$$\text{Precision}_+ = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.2}$$

$$\text{Recall}_+ = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.3}$$

Precision_+ gives a measure of precision for the Positive class. Precision is measured by dividing the number of test examples that are correctly classified as positive by the classifier by the number of test examples that are classified as positive (either correctly or wrongly) by the classifier. Precision for a class c is the proportion of all test examples that have been classified as belonging to class c that are true members of class c.

Recall measures the proportion of test examples belonging to a certain class that are classified correctly as belonging to the class. The formula given above computes recall for the positive class. Assume we have a total of 10 examples of the Positive class in all of the test set. Of these, our trained classifier classifies 8 as belonging to the Positive class. Then, our recall for the Positive class is 8/10=0.8 or 80%.

For a binary classifier, it is normally sufficient to provide the values of Precision_+ and Recall_+, and simply call them Precision and Recall. However, it is possible to compute precision and recall for the Negative class also:

$$\text{Precision}_- = \frac{\text{TN}}{\text{TN} + \text{FN}} \tag{4.4}$$

$$\text{Recall}_- = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{4.5}$$

4.3.4.4 F-measure

The F-measure (also known as the F1-score) is a popular performance metric that was introduced by Van Rijsbergen. It combines both precision and recall into a single score, providing a balanced evaluation of the model's performance. The F-measure for a class is the harmonic mean of the recall and precision for the class. The harmonic mean is used since the two metrics being combined are in opposition to each other—when one goes up, the other goes down. The F-measure is defined as follows.

$$\text{F} = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.6}$$

The F1-score is a number between 0 and 1, with 1 representing perfect precision and recall and 0 indicating poor performance. A higher F1-score reflects a better balance of precision and recall, as well as a good trade-off between false positives and false negatives. The F1- score is very helpful when working with datasets that are unbalanced and have one class outnumbering the other in terms of instance count. It is a useful indicator for gauging the general effectiveness of a binary classification model since it considers both false positives and false negatives, allowing for a more thorough evaluation.

Lesson 5: Regression

5.1 Introduction

Often, we are given a dataset for supervised learning, where each example is described in terms of a number of features, and the label associated with an example is numeric. We can think of the features as independent variables and the label as a dependent variable. For example, a

single independent variable can be a person's monthly income, and the dependent variable can be the amount of money the person spends on entertainment per month. The person is described in terms of one feature, income; the person's label is the amount of money spent on monthly entertainment. In this case, the training dataset consists of a number of examples where each person is described only in terms of monthly income. Corresponding to each person there is a label, which is the person's entertainment expense per month. Usually, the example is written as x and the label as y. If there are m examples in the training set, we refer to the ith example as $x^{(i)}$ and its corresponding label as $y^{(i)}$.

The training examples can be more complex if each person is described in terms of a number of features such as the person's age, income per month, gender, number of years of education, marital status, and the amount of money in the bank. In this case, the example is a vector x, and the label is still y. In a training set, the ith example can be referred to as $x^{(i)}$, and the corresponding label is $y^{(i)}$.

5.2 Regression

In regression, given a training dataset like the ones discussed above, a machine-learning program learns a function or a model that describes the trend in the values of the dependent variable (i.e., the label) in terms of the values of the independent variables, i.e., the feature values. The goal is to be able to predict the value of the dependent variable when a new unseen example is presented to the learned model. The regression function or model learned should be such that it generalizes well from the data, which is likely to have errors. In other words, the model learned should not overfit the training data, i.e., it should not just remember the training data or learn incidental patterns in the training data, but should predict values for unseen examples, and do so well without making too many errors.

Table 5.1 shows a dataset from 1968, obtained from a University of Florida website1 that presents the concentration of LSD in a student's bodily tissue and the score on a math exam.

Table 5.1 LSD level in tissue and performance on a math exam.

LSD	Math
78.93	1.17
58.20	2.97
67.47	3.26
37.47	4.69
45.65	5.83
32.92	6.00
29.97	6.41

5.2.1 Linear least-squares regression

Consider the first dataset given above. Suppose the goal is to fit a straight line that describes the relationship between the amount of drug in the tissue of an exam taker and the score received on a math test. A scatter plot for the dataset is shown in Figure 5.1(a). Viewing the scatter plot shows a potentially straight-line relationship going from the top left corner to the bottom right corner. A regression line may look like the one shown in Figure 5.1(b). However, there are many possible lines we can draw as the linear fit to the data. Which line should we draw?

In the Drugs and Grades dataset, the goal is to fit a line defined by the equation $y = mx + c$ to the data points given, since each data example is specified as a single scalar x. The assumption in regression or machine learning, in general, is that the data are not perfect, because the data

have flaws due to reasons such as observation errors, record-keeping errors, and transcription errors. Thus, when we fit a regression line, each point may actually be not sitting exactly on the line. Thus, the fit of each point to the regression line may have an error in it. Let the error associated with the fit of the ith point be $\epsilon^{(i)}$.

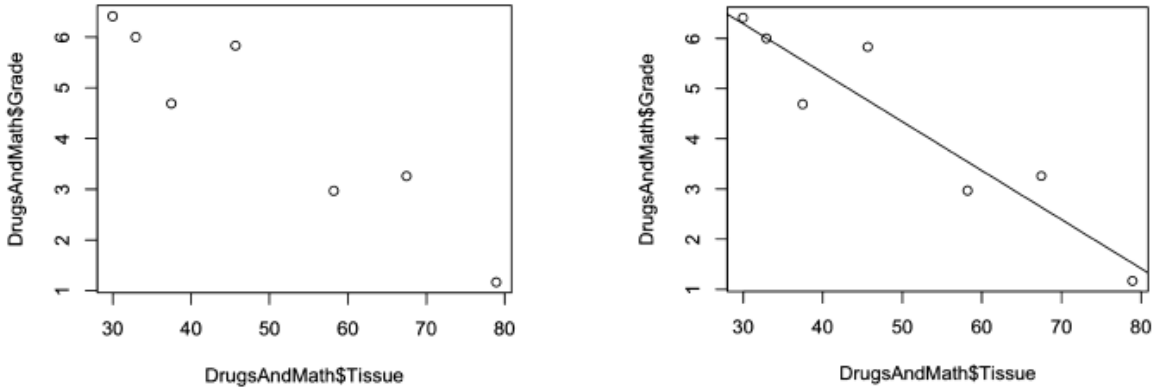


Figure 5.1 Scatter Plot and Linear-Regression Line for the Drugs and Grades dataset.

Since the dataset has n points in it, the total cumulative error of fit of a regression line to a training dataset can be written as

$$E = \sum_{i=1}^{n} \epsilon^{(i)} \qquad (5.1)$$

if all the errors across the dataset are simply added. Thus, we can write an objective function to obtain the regression line as

Find line y = mx + c such that it minimizes the cumulative error $E = \sum_{i=1}^{n} \epsilon^{(i)}$.

However, the problem with this approach is that some of the errors are positive and some of the errors are negative, and since negative errors cancel positive errors, the regression line that minimizes the direct sum of errors may actually turn out to be a bad fit.

An alternative may be to compute the cumulative error as

$$E = \sum_{i=1}^{n} \left|\epsilon^{(i)}\right|, \qquad (5.2)$$

where |.| is the absolute value. This representation of cumulative error is good, but absolute values are usually difficult to deal with in mathematics. As a result, an alternative approach that is commonly used is that the cumulative error is the sum of the squares of individual errors, given by

$$E = \sum_{i=1}^{n} \left\{\epsilon^{(i)}\right\}^2 \qquad (5.3)$$

Since squares of both positive and negative errors are positive, the total error does not vanish unless there is no cumulative error at all. If there is error, depending on the magnitude of the individual errors (less than 1 or more than 1), the error can be squashed or magnified. The modified objective function to obtain the regression line becomes

Find line y = mx + c such that it minimizes the cumulative error $E = \sum_{i=1}^{n}\{\epsilon^{(i)}\}^2$

Since the error $\epsilon^{(i)} = y^{(i)} - mx^{(i)} - c$, we can write the cumulative error expression as

$$E = \sum_{i=1}^{n}\{y^{(i)} - mx^{(i)} - c\}^2 . \quad (5.4)$$

In this expression, the $x^{(i)}$ and $y^{(i)}$ values are known from the training dataset. The values for m and c that minimize E need to be obtained. To find the equation of the line that minimizes this cumulative error in eq. (5.4), we obtain its partial derivatives with respect to the two "variables" m and c, set them to 0 and solve for the values of m and c.

The solutions are:

$$m = \frac{n\left(\sum_{i=1}^{n}\{x^{(i)}y^{(i)}\}\right) - \left(\sum_{i=1}^{n}\{x^{(i)}\}\right)\left(\sum_{i=1}^{n}y^{(i)}\right)}{n(\sum_{i=1}^{n}\{x^{(i)}\}^2) - (\sum_{i=1}^{n}\{x^{(i)}\})^2} \quad (5.5)$$

$$c = \frac{1}{n}\sum_{i=1}^{n}y^{(i)} - m\frac{1}{n}\sum_{i=1}^{n}x^{(i)}$$

$$= \bar{y} - m\bar{x} \quad (5.6)$$

In Eq. (5.6), $\bar{y}$ and $\bar{x}$ are means.

We can also perform linear least-squares regression using a wide variety of tools. If the dataset is two-dimensional, it is a good idea to perform a scatter plot of the data first. A scatter plot for the Drugs and Grades dataset is seen in Figure 5.1. A linear regression fit obtained using the programming language R can be seen in Figure 5.1. The equation of the line given by R is

$$= -0.09743473x + 9.21308462. \quad (5.7)$$

5.3 Evaluating linear regression

There are various metrics that are used to evaluate how well a trained model fits the data. A few of them are discussed below

5.3.1 Coefficient of determination R^2

A common way to determine how well a linear regression model fits the data is by computing the coefficient of determination, R^2. If $\hat{y}^{(i)}$ is the predicted value for $x^{(i)}$ using the regression line, whereas $y^{(i)}$ is the actual value of the dependent variable, the coefficient of determination is given as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{\sum_{i=1}^{n}(y^{(i)} - \bar{y})^2} \quad (5.8)$$

If the predicted values are close to the actual values, R^2 should be close to 1. If the predictions are not close to the actual values, $R^2 \approx 0$. The value of R^2 is always between 0 and 1, and a value close to 1 is better, although if we are comparing two fitted models, one that has a higher R^2 value is not necessarily better.

The R^2 metric describes the proportion of variation in the predicted variable explained by the regression model. The R^2 value is 0.8778 for the Drugs and Grades dataset, signifying that 87.78% of the variation in the math grade is captured by the model.

5.3.2 Standard error of regression and F-statistic

The Sum of Squared Errors, SSE, is defined as

$$SSE = \sum_{i=1}^{n} \left(\hat{y}^{(i)} - \bar{y}\right)^2 \tag{5.9}$$

The Mean-Squared Error, MSR, is defined as

$$MSR = \frac{SSE}{n-q}, \tag{5.10}$$

where q is the number of coefficients in the model. q = 2 for linear regression in two dimensions. Thus, for linear regression

$$MSR = \frac{SSE}{n-2}. \tag{5.11}$$

This definition reflects the fact that to begin with, we have n variables or degrees of freedom in the description of the data. Since there are 2 coefficients, two degrees of freedom are covered by the linear regression line.

The Standard Error for linear regression is defined as

$$StdError = \sqrt{MSE}, \tag{5.12}$$

where MSE is the mean of the squared errors. Smaller values of StdError are better. A value closer to 0 is good for StdError.

F-statistic is defined as

$$\text{F} - \text{statistic} = \frac{MSR}{MSE}. \tag{5.13}$$

A higher value of F-statistic is better than a lower value.

5.3.3 Is the model statistically significant?

It is customary to compute what is called a p-value when performing many statistical computations to indicate if the results obtained are statistically significant, i.e., they have any (statistical) merit. That is, whether the results can be trusted and are meaningful, and can really be used. P-values are computed in different ways for different situations.

When we compute a p-value, there is always an associated Null Hypothesis and an Alternate Hypothesis. For linear regression, the Null Hypothesis is that the coefficients associated with the independent variables (i.e., features) are all 0. In other words, the Null Hypothesis says there is no relationship of any significance between the dependent and independent variables. The Alternate Hypothesis is that the coefficients are not equal to 0, i.e., there is actually a linear relationship between the dependent variable and the independent variable(s).

To compute the p-value, we have to assume a value for what is called a t-value in turn. A larger t-value, written simply as t, indicates that it is less likely that the coefficient is not equal to 0 purely by chance. Hence, a higher t-value is better.

The p-value is defined as probability(> |t|), i.e., the probability that the t-value computed is high or higher than the observed value when the Null Hypothesis is true. Hence, if probability(> |t|) is low, the coefficients are significant (significantly different from 0). If probability(> |t|) is high, the coefficients are not significant.

In practice, we can assume a significance level for p-value as something like 0.05. If the computed p-value is less than the significance level (here, 0.05), we can reject the Null Hypothesis that the coefficients are 0. In other words, the linear regression model we computed is meaningful and can be used for predictive purposes.

5.4 Overfitting in regression

Overfitting happens when there are coincidental or accidental patterns in the training data. These accidental patterns could arise due to the presence of noise in the data, or because the size of the training dataset is small, or can happen just randomly.

One of the main purposes of a machine-learning algorithm is to be able to generalize, i.e., look past such accidental or random patterns and be able to perform well when dealing with previously unseen data. In the case of regression, it would mean that we would like to be able to predict well the value of the dependent variable given the values of the independent variables. A regressor that fits very well to the data, i.e., has a very low sum of squared errors, may in fact, be a bad generalizer; we say that such a regressor has overfit the data. A regressor that does not fit at all well to the data, i.e., has a very high sum of squared errors, is also bad; we say that such a regressor underfits the data. The goal in machine learning, and regression, in particular, is to neither underfit nor overfit.

5.7 Reducing overfitting in regression: regularization

There are several approaches to reducing overfitting. A common method is called regularization.

When a function is fitted using ordinary least-squares linear regression, the coefficients are easily understood and as a result, the fit is easy to interpret. However, it is likely that some predictor variables not related to the dependent variable occur in the fitted equation, making it unnecessarily complex. Even if a predictor is unrelated to the dependent variable, the corresponding coefficients are unlikely to be zero in an ordinary least-squares fit. There may also be wild swings in the fitted curve if the degree of the polynomial is high, i.e., we fit a polynomial instead of a linear function.

Thus, a preferred solution to the regression problem may fit a linear (or maybe even higher order) function, but constrain the sizes of the coefficients in some ways. Such methods are called shrinkage methods. Shrinkage methods suppress the values of the coefficients, preventing the fitted equation from fitting too closely to the dataset (also, from having wild swings if it is a polynomial fit), i.e., providing stability to the predictions and a measure of generalization. Some shrinkage methods can also reduce certain coefficients to zero, performing a type of feature reduction or selection.

5.5 Summary

This lesson has provided a comprehensive overview of regression analysis and its applications, including techniques for evaluating and improving linear-regression models, and approaches to addressing overfitting in regression models. We begin by introducing regression analysis and explaining the basic concepts of linear regression. We then move on to discuss techniques for evaluating linear regression models, including the coefficient of determination R^2, standard

error of regression, and F-statistic. We also cover the importance of assessing the statistical significance of a model.

Lesson 6: Classification

6.1 Introduction

Machine learning begins with a collection of data examples of interest to explore. It applies a variety of algorithms to discover hidden regularities or patterns in the collection in the form of a model, and uses the learned model for the purposes at hand.

There are primarily two types of machine-learning algorithms: supervised and unsupervised. Supervised algorithms use a collection of data where each example has been examined and labeled by an "expert" in some way. For example, many supervised algorithms perform what is called classification. The idea of class is fundamental to classification. We think of a class as a group of objects that share some properties or attributes or features, and can be thought of as a "natural" category, relevant to a problem at hand. Each example is described in terms of a number of features. For example, if our examples are from the domain of vehicles, each example may have features such as height, length, width, weight, color, runs on gas or diesel or electricity, and the number of seats. The associated classes may be the names of the manufacturers of the vehicles, such as Ford, General Motors, and BMW. The purpose of supervised learning in this case is to learn the patterns from a dataset of examples where the manufacturers are known, so that the learned patterns can be used to "predict" or identify the manufacturers of a vehicle for which the manufacturer is unknown, based on the feature values. Thus, for supervised learning, we need a dataset of examples, where each example is described in terms of its features and also, its class.

Given a dataset where each example is described in terms of features and a class label, the task of supervised learning is to discover patterns to build a "model" for distinguishing among the classes involved. A model, in this case, is a way to put together a description of the found patterns in such a way that it can be used to tell apart unseen examples of the classes. The model may not be explicit, but stored implicitly. Each machine-learning algorithm has a learning bias, and the model formed depends on the learning bias.

Unsupervised learning also attempts to discover patterns in a dataset. Datasets used in unsupervised learning have examples like datasets used in supervised learning, where examples are described in terms of features. However, examples used in unsupervised learning lack the class label. A common form of unsupervised learning, called clustering, attempts to put examples in various groups, where examples inside a group are similar to each other as much as possible, and dissimilar to examples in other groups as much as possible. To be able to perform such grouping, we need to use what is called a similarity measure or similarity metric to compute how similar or dissimilar are two examples. On the flip side, we can use a distance measure or distance metric also. It may also be useful to measure the similarity or dissimilarity between two groups of examples.

Researchers and practitioners have developed a large number of algorithms for supervised and unsupervised learning. Among the most commonly used supervised learning or classification algorithms are nearest-neighbor algorithms, tree-based algorithms, support-vector machines, and neural networks. We discuss some of these algorithms in this session. It is also common place to use several classifiers that complement each other to perform a number of separate classifications, and then use a group decision in terms of voting or consensus, to make the final classification. Such classifiers are called ensemble classifiers.

Developing a classifier algorithm, training it on examples, and using it on unseen examples (also, called testing) are the main things we do in supervised learning. The purpose of using a supervised learning algorithm is to be able to train it on a dataset and use the trained model to classify unseen examples in a test setting or in the real world. To decide which algorithm is best suited for a certain supervised machine-learning task, we need to evaluate the performance of the algorithm in terms of metrics.

6.2 Nearest-neighbor classifiers

These are among the simplest of classifiers, but also at the same time quite effective. There is an adage in English "A man is known by the company he keeps." Imagine that we have a number of data examples for which we know the classes they belong to. That is, each example is described in terms of its features and also has a class label assigned to it. Now, suppose a new example comes along and we have to decide which class it belongs to. First, we need to situate the new unclassified example e among the already labeled examples considering the features we use to describe examples. For example, if each example is described in terms of just two numeric features, say x and y, we can place the new example e among the already classified examples in a 2D space described by two dimensions x and y, assuming the two features are orthogonal to each other. To assign a class to the new example, we can look at its neighbors. In general, we can look at k neighbors. If k is equal to one, we look at just one neighbor, the nearest neighbor n of e, and assign e the class that this neighbor n belongs to. Figure 6.1(a) shows a situation where the unknown example e is closest to a known example of the "circle" class. Thus, the unknown example is assigned to be in the circle class as well. If k is equal to 3, we look at 3 nearest neighbors, n1, n2, and n3, and assign e the class that a majority of these three belong to. Figure 6.1(b) shows an example where of the three nearest neighbors of the unknown example e, two belong to the circle class and one belongs to the "square" class. Thus, if we go by majority vote, the unknown example is assigned the circle class by a two-to-one vote of its three nearest neighbors. Similarly, for a more general value of k, the k nearest neighbors can, in some sense vote and assign a class to the new example e. This is the essence of how nearest-neighbor classifiers work.

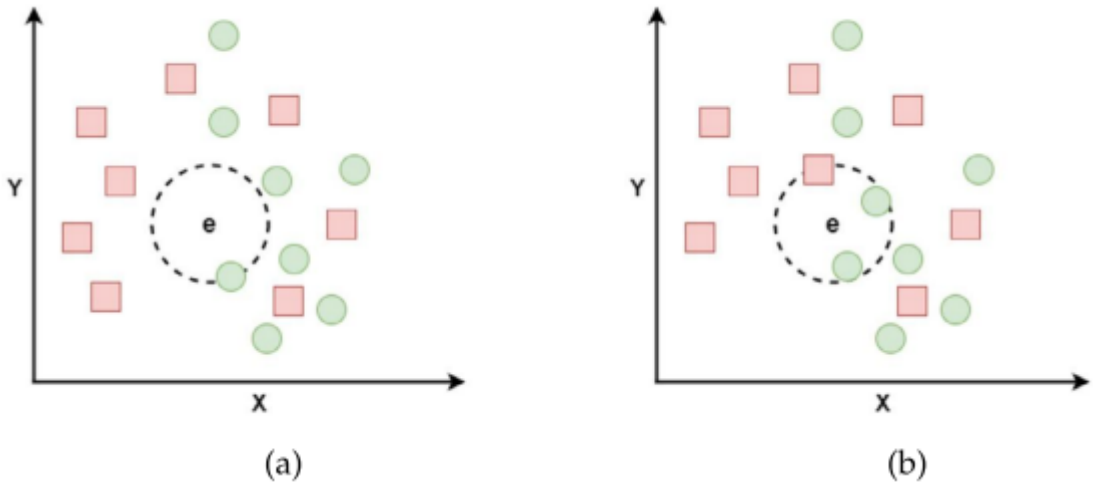


Figure 6.1. k-NN classifier, with k = 1 and k = 3.

Fig. 6.2 shows a graph of the results of experiments with the Iris dataset. The dataset was randomly divided into two parts: one with 60% of the 150 examples for training, and the rest of the examples for testing. Values of k = 1 through k = 15 were tried to determine what value works the best for this dataset. For each value of k, the figure shows the error in training as well as testing. The training-error values are shown in the dashed graph in blue. The test results

are in the graph with straight lines between two neighboring points. The figure shows that k = 6 or k = 7 produces the least test error and hence, any of these two values can be used.

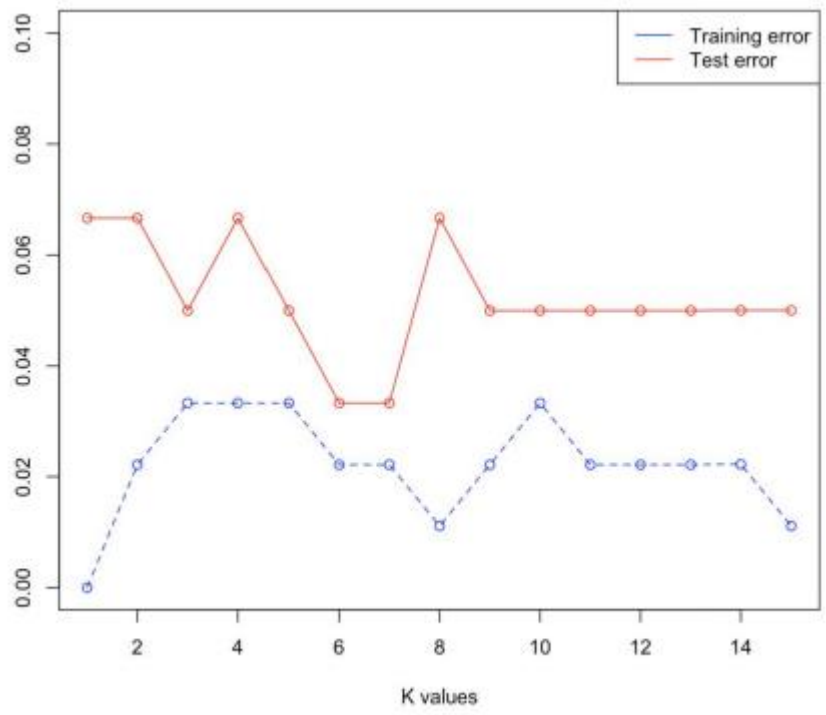


Figure 6.2. Running k-NN classifier with several values of k. The figure shows misclassification errors for training and testing. The top plot shows test error, and the lower one training error.

6.2.1 Storing classified examples

In the case of nearest-neighbor classifiers, we do not process the classified examples or the input training set at all. We simply record them in a manner that makes it efficient to look up neighbors of a new unknown example. This recording may involve indexing the examples by their feature values in terms of binary trees or more advanced data structures like K-D trees where there may be a large number of branches at each node. When a new example comes along, by traversing the index structure, we are able to find the appropriate number of neighbors quickly. The neighbors' classes are retrieved, and frequently, a majority vote is taken to produce a class assignment for the new example e. Using an efficient data structure is important if the number of data examples is large.

A K-D tree is a binary tree in which every node is a k-dimensional point, in our case a k-dimensional training example. An interior node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces. Points to the left of this hyperplane are represented by the left subtree of that node and points right of the hyperplane are represented by the right subtree. Because each training example has k features or dimensions, any of the dimensions may be chosen at an internal node for splitting. The dimension to split and the values to split at for the dimension can be chosen in many different ways depending on the algorithm to build and traverse the tree.

6.2.2 Distance measure

Thus, to find neighbors of an example, we have to traverse the index structure to obtain potential neighbors of example e. Next, we have to compute the pairwise distance or similarity

between e and the potential neighbors to find the k nearest neighbors. We must use a similarity or distance metric to implement a nearest-neighbor algorithm.

To measure the distance between two training examples, the most common measure used is Euclidean distance. Assume that a data example or point has m dimensions. If a and b are two data examples, we can write $a = \langle a_1 \dots a_m \rangle$ and $b = \langle b_1 \dots b_m \rangle$, where a_i and b_i are attribute values of the data points. The Euclidean distance between two examples is given as

$$d_{Euclidean}(a,b) = \sqrt{(a_1 - b_1)^2 + \cdots + (a_m - b_m)^2} = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2} \quad (6.1)$$

The Euclidean distance between two vectors is nonnegative, but potentially unbounded. Another common way to compute the distance between two data examples is to compute an angular distance between the two examples or points since each example is a vector. The traditional way is to compute the cosine of the angle between the two vectors representing the two points:

$$d_{Cosine}(a,b) = \cos(\theta_{ab}) = \frac{\sum_{i=1}^{m} a_i b_i}{\sqrt{\sum_{i=1}^{m} a_i^2 \sum_{i=1}^{m} b_i^2}}, \quad (6.2)$$

where θab is the angle between the two vectors a and b. The cosine distance is always between −1 and 1, with −1 meaning the two vectors are exactly opposite and 1 meaning the two vectors point in the same direction. Two data examples are considered closest when the cosine distance is 1 and farthest when the cosine distance is −1.

6.2.3 Voting to classify

Once k nearest neighbors have been identified, these neighbors vote to determine the class of the new and unclassified example e. Often, majority voting is used, but in some casesconsensus may be necessary. In majority voting, the class that occurs most frequently becomes the class of the new example. The class that a neighboring example belongs to is considered its vote. If two classes receive the same number of votes, it is usually broken randomly. In consensus voting, all of the k neighbors have to belong to the same class to assign a class to the new example. If this is not the case, the new example remains unclassified.

6.3 Decision trees

Often, as human, when we make decisions regarding classification, we use a tree-like structure to do so. The nodes of such a tree have queries, and the branches lead to different paths based on answers to the queries. We start from the top node, which poses a query, and based on the answer, we take a path down the tree, and we answer another query at the second node, and take another path down the tree based on the answer. We traverse the tree in such a manner to a leaf node where we know to which class the example belongs.

Decision trees are built using supervised learning. In other words, we build a decision tree from a number of labeled examples given to us.

6.3.1 Building a decision tree

The task of building a decision tree involves deciding which one among the descriptive features should be used to construct the first query. A query involves a feature or attribute and a value for the attribute, along with a comparison operator. The first question to ask is which feature among the ones available should be the feature for the question, i.e., on what basis should it be

selected? The next question is what value of the feature is relevant to ask the question and what the comparison operator should be. The comparison operator is usually = for text or a categorical attribute, and >, < or ≥, or ≤ for a numerical attribute.

The objective is to build a compact decision tree that is consistent with all the examples from which it is built. The process is called training the decision tree. If the goal is to build the most compact tree, the process becomes exponential in time since the number of possible trees that can be built from n examples having k features with each feature taking a number of values exponentials in number. Thus, in practice, there is no time to create all possible trees and choose the most compact or optimal tree. Therefore, the approach to build a decision tree is necessarily heuristic and expedited, deciding on a feature to use in a query in a greedy fashion, a decision that cannot be undone once made. Thus, it is quite likely that a decision tree built by one of the commonly used methods is only locally optimal and is not the globally optimal one. However, even such locally optimal trees often happen to be quite good classifiers.

6.3.2 Entropy for query construction

The choice of a feature or attribute to construct a query at a certain level of the tree is usually made using the concept of entropy (and, sometimes another similar concept called the Gini Index). Entropy measures the amount of chaos or dissimilarity in a dataset, or how mixed up or inhomogeneous a dataset is. Homogeneity is measured with respect to the class(es) the examples in the dataset belong to. For example, if a dataset has n examples, and all examples belong to the same class c, the entropy of the dataset is 0 (see Figure 6.3(a)). In this case, there are 10 examples in the dataset included inside the big dashed circle, and each data example is a small rectangle, and thus the dataset is homogeneous. If a dataset has n examples, and $\frac{n}{2}$ examples belong to class c_1 and $\frac{n}{2}$ examples belong to another class c_2, it can be said that the dataset is completely mixed up and its entropy is 1. See Figure 6.3(b), where the dataset of 10 contains 5 examples of the circle class and 5 from the rectangle class.

The entropy of a binary dataset D containing n examples, with n_1 examples from class c_1 and n_2 examples from class c_2 is

$$Entropy(D) = -\frac{n_1}{n} log \frac{n_1}{n} - \frac{n_2}{n} log \frac{n_2}{n} \quad (6.3)$$

$$= -p_1 log p_1 - p_2 log p_2 \quad (6.4)$$

$$= \sum_{i=1}^{2} -p_i log p_i \quad (6.4)$$

where $n_1 + n_2 = n$, $p_1 = \frac{n_1}{n}$, and $p_2 = \frac{n_2}{n}$. p_1 is the probability of a data example being in class c_1, and p_2 is the probability of a data example being in class c_2. If the dataset contains examples from k classes $C = \{c_1 \dots c_k\}$, the entropy of the dataset is

$$Entropy(D) = \sum_{i=1}^{k} -p_i log p_i \quad (6.5)$$

where $p_i = \frac{n_i}{n}$ with n_i being the number of examples in class c_i in the dataset, with $\sum_{i=1}^{k} p_i = 1$ and $\sum_{i=1}^{k} n_i = n$. Thus, the entropy of a dataset considering the classes of examples in it lies between 0 and 1. If a dataset is not as mixed as being equally divided between two (or more) classes, the entropy value is less than one. Therefore, the entropy curve looks like the one given in Figure 6.3, assuming we have two classes to choose from. The X-axis shows the proportion of examples from one of the two classes in the dataset and the Y-axis shows the entropy of the

dataset. If there are two classes, the maximum value of entropy is 1 and the minimum value is 0. If there are k classes in the dataset, the maximum value of entropy is $\log_2$ n and the minimum value remains 0.

Figure 6.3. (a) A dataset with all examples from the same class, with a dataset entropy of 0. (b) A dataset with half the examples from one class and the other half from another, with an entropy of 1.

To select a feature to create a query, it is not necessary to look at the entropy of a dataset directly, but compute how the entropy will change when the different features are used to separate out the dataset into subsets. If a feature *f* is used to separate the dataset into disjoint subsets, one can compute the before-separation entropy of the entire set, and the after-separation weighted sum of entropies of the subsets, and then the change in entropy when the feature *f* is used for the process:

$$\Delta Entropy(D, f) = Entropy(D) - \sum_{i=1}^{nv(f)} \frac{|D_{fi}|}{|D|} Entropy(D_{fi}),$$

where D is the original dataset, nv(f) is the number of distinct values feature f can take, |D| is the number of examples in dataset D, and $|D_{fi}|$ is the number of total examples in the ith branch of the tree, i.e., the number of examples in D that have the ith unique value of f . If f can take k possible discrete values, there will be k branches, as shown in Figure 6.4(a). As a specific example, if Att_1 is the attribute chosen to use as a query, there will be three branches, as shown in Figure 6.4(b). If one chooses Att_2 as the query feature, there will be two branches, as shown in Figure 6.4(c).

The change in entropy can be computed for all the features of an example, and choose the one that produces the highest amount of entropy change. In other words, one chooses the feature that separates the mixed dataset into less mixed subsets of data. This is because the ultimate objective in building a decision tree is to obtain a tree where all the leaves have homogeneous subsets of data, i.e., each leaf node corresponds to a small number of examples corresponding to only one of the possible classes.

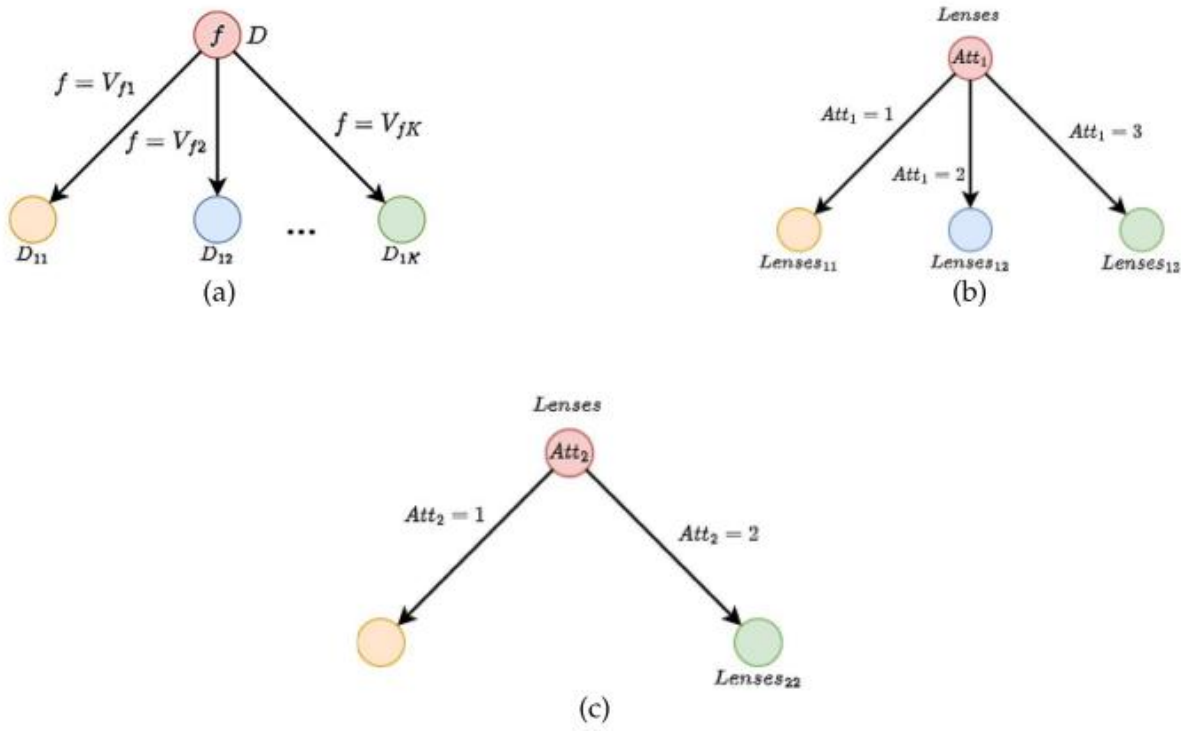


Figure 6.4. (a) Branching dataset D on feature f that has k possible values. (b) Branching dataset Lenses on feature Att1 that has 3 possible values. (b) Branching dataset Lenses on feature Att_2 that has 2 possible values.

6.3.3 Reducing overfitting in decision trees

One problem that is important to discuss is how to handle overfitting. The purpose of a machine-learning algorithm is to generalize from the limited number of examples by eliminating any accidental similarities that may exist so that the trained algorithm can be used on unseen examples or released to the public to use on whatever dataset they may have. Thus, a machine-learning algorithm must generalize to the right amount and try to avoid overfitting, which is difficult to do. In the context of decision trees, it is possible to build a large decision tree with many levels and many nodes to fit the training data exactly. Assume that a decision tree with 100 nodes has been built to fit the dataset exactly. However, it is likely that such a tree is too detailed and in addition to generalizing the information contained in the dataset, it also has built nodes to cater to accidental regularities that are present in this particular dataset only and may not be present in another similar dataset that may be collected for the same phenomena. Thus, it is possible that when put into action after training, a decision tree with 20 nodes will perform as well as or even better than a decision tree with 100 nodes.

Figure 6.5 shows a fairly detailed tree for the Lenses dataset with only 24 training examples. Perhaps a smaller tree may work as well, or even if it performs a little worse on the training data, it is likely to work better on unseen data that it needs to classify. In this specific tree, not only do the data fit the tree exactly, there is also a problem where one cannot decide what class it should be in one of the leaf nodes. How to build a decision tree with the right number of nodes so as not to overfit, but learn the requisite generalization involves pruning the decision tree. There are several ways to prune a decision tree, and the preferred way is to build a complete decision tree with as many nodes as necessary to produce homogeneous leaf nodes, but then remove as many nodes as possible so as to remove nodes created to cater to accidental regularities. One may simply pick and delete a random node and all nodes below it, and test the new reduced tree using the same dataset that was used to test the entire tree, and if the new reduced tree's performance is not worse than the original tree, keep the reduced tree. The process of pruning is repeated a number of times, and a node is removed only if performance

does not degrade or degrades just a little. It has been seen that reduction of 50–70% or even more nodes may keep the tree's performance almost as good as it was for the original tree. Therefore, it may be hypothesized that the nodes eliminated from the original tree covered only nonessential (maybe duplicated) and possibly accidental regularities and their removal actually not only does not degrade performance, but makes the performance more efficient (since fewer nodes need to be tested for an unseen example) and at the same time possibly a better performer on unseen examples. Thus, a pruned tree like the one shown in Figure 6.6 may work better than the tree in Figure 6.5 although the more complex tree has 6 nodes compared to just 3 nodes in the pruned tree.

Another way to prune converts the decision tree to a set of if-then rules. Each path from the root to a leaf node of a decision tree can be described by writing a corresponding if then rule. All the rules corresponding to all the paths can be written as a set of rules or a ruleset.

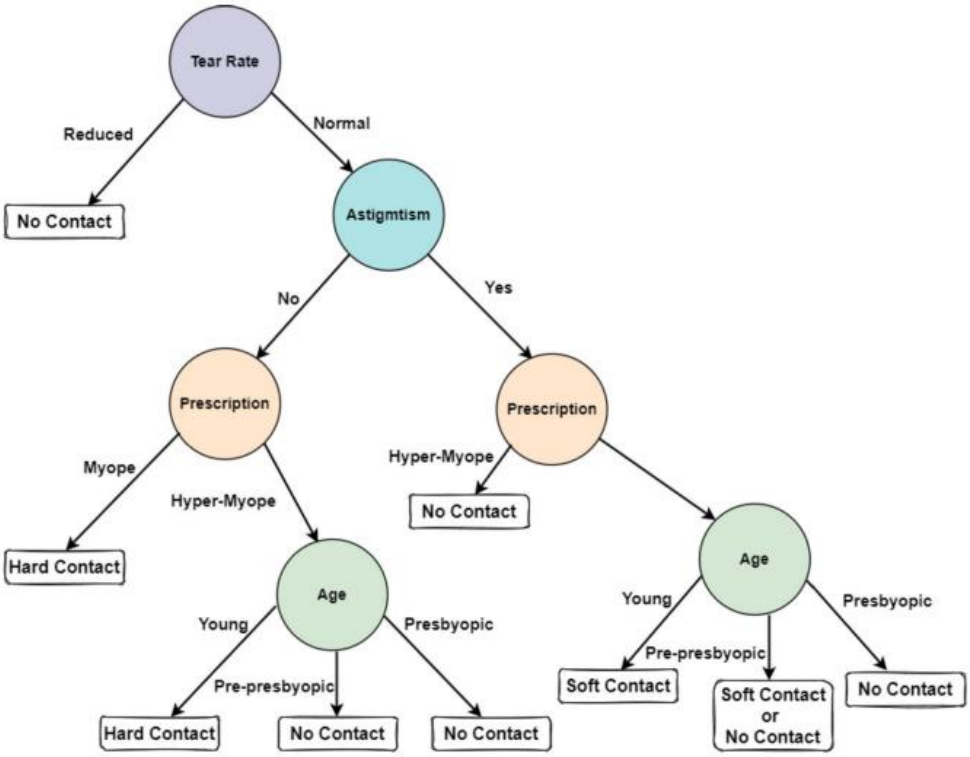


Figure 6.5. An example of a decision tree built from the Lenses dataset.

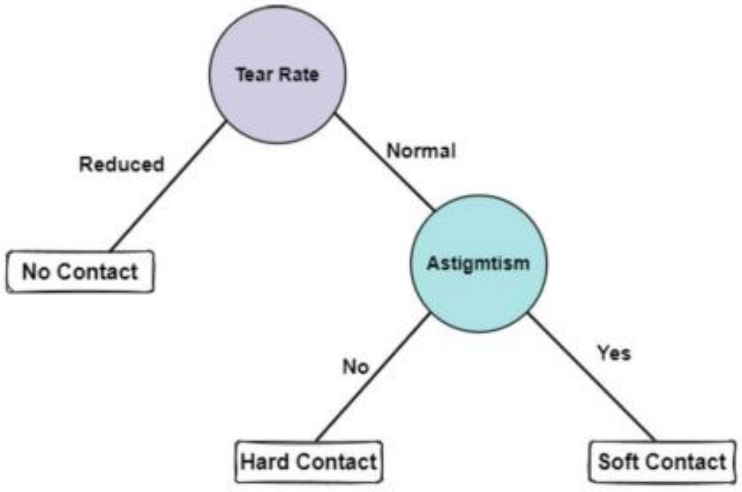


Figure 6.6. An example of a pruned decision tree built from the Lenses dataset.

Converting to a set of rules removes the distinction between nodes that occur at upper levels of the tree and those that occur below because all of a rule's conditions are considered to be at the same level. In addition, since a node, especially at an upper level occurs several times in the rules, the node's influence becomes distributed and as a result, the effect of pruning a node is distributed, since a node now occurs in several rules. Once there is a set of rules, pruning simply picks a random rule and picks a random antecedent and removes the antecedent. If after removing the antecedent, the ruleset's performance does not decrease, the reduced rule is kept in the ruleset instead of the original rule. If the ruleset's performance decreases that specific antecedent is not removed, but another one is tried for removal and the process is repeated. In general, it is possible to remove many antecedents and even complete rules and obtain a simplified ruleset.

Lesson 7: Artificial neural networks

7.1 Introduction

The nominal objective in the efforts to create Artificial Neural Networks (ANNs) is to build effective and efficient learning machines that are inspired by the computational structures in the human brain, in order to perform tasks that most human beings seem to perform effortlessly and hence achieve a modicum of so-called artificial intelligence. A just-born human baby is endowed with an impressive and elaborate brain structure that humans have acquired through millions of years of evolution. However, this initial brain learns new things from infancy to adulthood and also grows in size to a degree with age as it learns. Following this remarkable engineering achievement of evolution, scientists and engineers have attempted to construct artificial intelligence by building structures or organizations (called architectures) of artificial neurons that can learn from data or experience. In the recent past, such artificial neuronal architectures that can learn have achieved excellent performance in many computational tasks; this can be thought of as endowing artificial intelligence on machines.

The use of artificial neural networks offers several useful properties and capabilities:

1. Nonlinearity - An artificial neuron as a basic unit can be a linear- or nonlinear-processing element, but the entire ANN is highly nonlinear. It is a special kind of nonlinearity in the sense that it is distributed throughout the network. This characteristic is especially important, for ANN models the inherently nonlinear real-world mechanisms responsible for generating data for learning.

2. Learning from examples - An ANN modifies its interconnection weights by applying a set of training or learning samples. The final effects of a learning process are tuned parameters of a network (the parameters are distributed through the main components of the established model), and they represent implicitly stored knowledge for the problem at hand.

3. Adaptivity: An ANN has a built-in capability to adapt its interconnection weights to changes in the surrounding environment. In particular, an ANN trained to operate in a specific environment can be easily retrained to deal with changes in its environmental conditions. Moreover, when it is operating in a nonstationary environment, an ANN can be designed to adopt its parameters in real time.

4. Evidential Response: In the context of data classification, an ANN can be designed to provide information not only about which particular class to select for a given sample, but also about confidence in the decision made. This later information may be used to reject ambiguous data, should they arise, and thereby improve the classification performance or performances of the other tasks modeled by the network.

5. Fault Tolerance: An ANN has the potential to be inherently fault-tolerant, or capable of robust computation. Its performances do not degrade significantly under adverse operating conditions such as disconnection of neurons, and noisy or missing data. There is some empirical evidence for robust computation, but usually it is uncontrolled.

6. Uniformity of Analysis and Design: Basically, artificial neural networks enjoy universality as information processors. The same principles, notation, and the same steps in methodology are used in all domains involving application of artificial neural networks.

To explain a classification of different types of ANNs and their basic principles it is necessary to introduce an elementary component of every ANN. This simple processing unit is called an artificial neuron.

7.2 Model of ANN

An artificial neuron is an information-processing unit that is fundamental to the operation of an ANN. The block diagram (Figure 7.1), which is a model of an artificial neuron shows that it consists of three basic elements:

1. A set of connecting links from different inputs x_i (or synapses), each of which is characterized by a weight or strength w_{ki}. The first index refers to the neuron in question and the second index refers to the input of the synapse to which the weight refers. In general, the weights of an artificial neuron may lie in a range that includes negative as well as positive values.

2. An adder for summing the input signals x_i weighted by the respective synaptic strengths w_{ki}. The operation described here constitutes a linear combiner.

3. An activation function f for limiting the amplitude of the output y_k of a neuron.

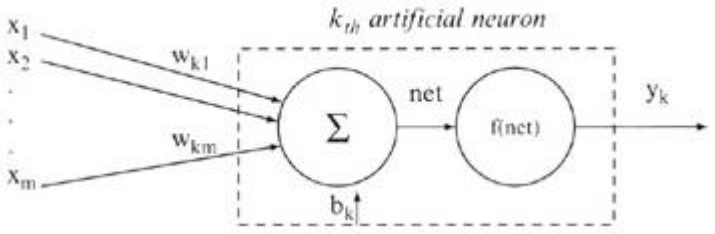


Figure 7.1. Model of an artificial neuron

The model of the neuron given in Figure 7.1 also includes an externally applied bias, denoted by b_k. The bias has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative.

In mathematical terms, an artificial neuron is an abstract model of a natural neuron, and its processing capabilities are formalized using the following notation. First, there are several inputs x_i, i = 1, …, m. Each input x_i is multiplied by the corresponding weight w_{ki} where k is the index of a given neuron in an ANN. The weights simulate the biological synaptic strengths in a natural neuron. The weighted sum of products $x_i w_{ki}$, for i = 1, …, m is usually denoted as *net* in the ANN literature:

$$net_k = x_1 w_{k1} + x_2 w_{k2} + \cdots x_m w_{km} + b_k = \sum_{i=1}^{m} x_i w_{ki}$$

The same sum can be expressed in vector notation as a scalar product of two m-dimensional vectors:

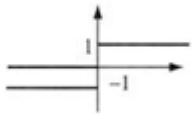
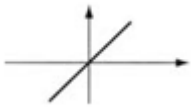
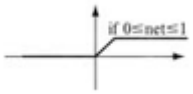
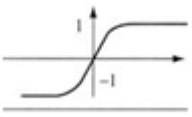
$$net_k = X.W$$

Finally, an artificial neuron computes the output y_k as a certain function of net_k value:

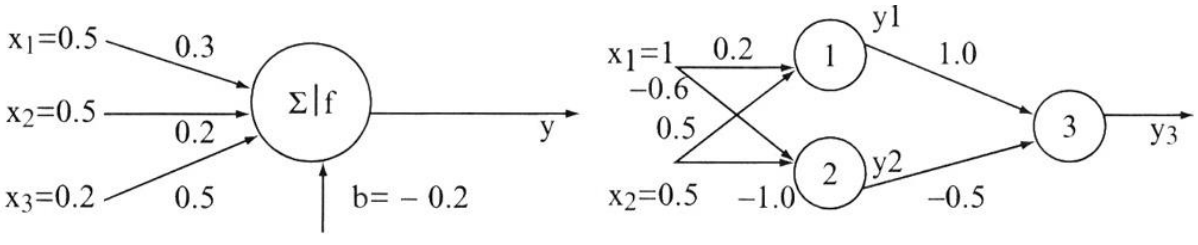
$$y_k = f(net_k)$$

The function f is called the activation function. Various forms of activation functions can be defined. Some commonly used activation functions are given in Table 7.1.

Table 7.1. A neuron's common activation functions

Activation Function	Input/output Relation	Graph
Hard Limit	$y = \begin{cases} 1 & \text{if net} \geq 0 \\ 0 & \text{if net} < 0 \end{cases}$	
Symmetrical Hard Limit	$y = \begin{cases} 1 & \text{if net} \geq 0 \\ -1 & \text{if net} < 0 \end{cases}$	
Linear	Y = net	
Saturating Linear	$y = \begin{cases} 1 & if\ net > 1 \\ net & \\ 0 & if\ net < 0 \end{cases}$	
Symmetric Saturating Linear	$y = \begin{cases} 1 & if\ net > 1 \\ net & if\ -1 \leq net \leq 1 \\ -1 & if\ net < -1 \end{cases}$	
Log-Sigmoid	$y = 1/(1 + e^{-net})$	
Hyperbolic Tangent Sigmoid	$y = (e^{net} - e^{-net})/(e^{net} + e^{-net})$	

For example, for the neuron with three inputs and one output, the corresponding input values, weight factors, and bias are given in Figure 7.2. It is necessary to find the output y for different activation functions such as Symmetrical Hard Limit, Saturating Linear, and Log-Sigmoid.



a) A single node b) Three interconnected nodes

Figure 7.2: Examples of artificial neurons and their interconnections

1. Symmetrical Hard Limit

   net =0.5.0.3+0.5.0.2+0.2.0.5+(-0.2).1= 0.15

   y = f(net) = f(0.15) = 1

2. Saturating Linear

   net =0.5.0.3+0.5.0.2+0.2.0.5+(-0.2).1= 0.15

   y = f(net)=f(0.15)=0.15

3. Log-Sigmoid

   net =0.5.0.3+0.5.0.2+0.2.0.5+(-0.2).1= 0.15

   y = f(net)=f(0.15)=1/(1+$e^{-0.15}$)=0.54

The basic principles of computation for one node may be extended for an artificial neural network with several nodes even if they are in different layers, as given in Figure 7.2b. Suppose that for the given configuration of three nodes all bias values are equal to 0 and activation functions for all nodes are symmetric saturating linear. What is the final output Y_3 from the node 3?

The processing of input data is layered. In the first step, the neural network performs the computation for nodes 1 and 2 that are in the first layer:

net_1=1.0.2+0.5.0.5=0.45⟹y_1=f(0.45)=0.45

net_2=1.(-0.6)+0.5.(-1)=-1.1⟹y_2=f(-1.1)=-1

Outputs y_1 and y_2 from the first layer nodes are inputs for node 3 in the second layer:

Net_3= y_1. 1 + y_2 .(-0.5)= 0.45.1 + (-1). (-0.5) = 0.95 ⟹y_3=f(0.95)=0.95

As we can see from the previous examples, the processing steps at the node level are very simple. In highly connected networks of artificial neurons, computational tasks are multiplied with an increase in the number of nodes. The complexity of processing depends on the ANN architecture.

Architectures of artificial neural networks

The architecture of an artificial neural network is defined by the characteristics of a node and the characteristics of the node's connectivity in the network. The basic characteristics of a single node have been given in a previous section and in this section the parameters of connectivity will be introduced. Typically, network architecture is specified. by the number of inputs to the network, the number of outputs, the total number of elementary nodes that are usually equal processing elements for the entire network, and their organization and interconnections. Neural networks are generally classified into two categories on the basis of the type of interconnections: feedforward and recurrent.

Feedforward

The network is feedforward if the processing propagates from the input side to the output side unanimously, without any loops or feedbacks. In a layered representation of the feedforward neural network, there are no links between nodes in the same layer; outputs of nodes in a specific layer are always connected as inputs to nodes in succeeding layers. This representation is preferred because of its modularity, i.e., nodes in the same layer have the same functionality or generate the same level of abstraction about input vectors.

Recurrent

If there is a feedback link that forms a circular path in a network (usually with a delay element as a synchronization component), then the network is recurrent. Examples of ANNs belonging to both classes are given in Figure 7.3.

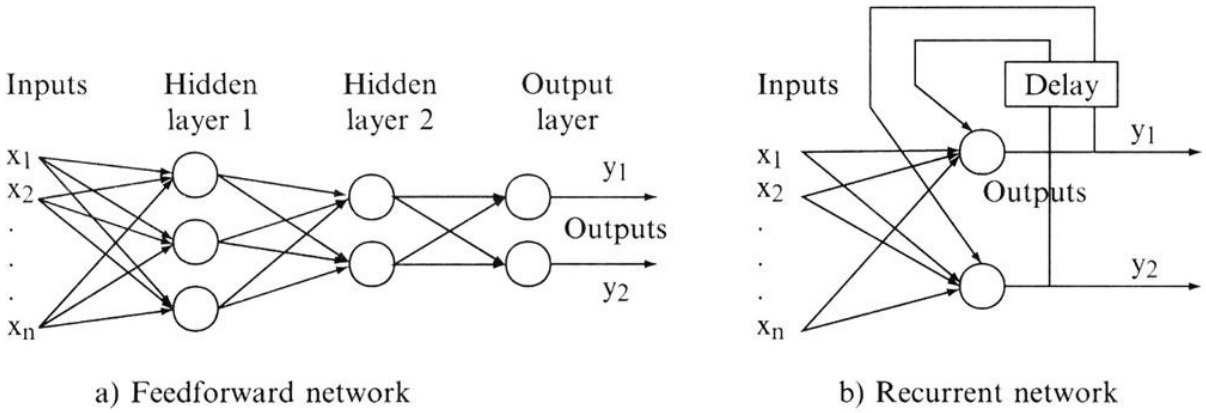


Figure 7.3: Typical architectures of artificial neural networks

Although many neural-network models have been proposed in both classes, the multilayer feedforward network with a backpropagation-Iearning mechanism, is the most widely used model in terms of practical applications. Probably over 90% of commercial and industrial applications are based on this model.

Why multilayered networks?

The basic conclusion is that single-layered ANNs are a convenient modeling tool only for relatively simple problems that are based on linear models. For most real-world problems, where models are highly nonlinear, multilayered networks are better and maybe the only solution.

Learning process

A major task for an ANN is to learn a model of the world (environment) in which it is embedded and to maintain the model sufficiently consistent with the real world so as to achieve the specified goals of the concerned application.

The property that is of primary significance for an artificial neural network is the ability of the network to learn from its environment based on real-life examples, and to improve its performance through that learning process. An ANN learns about its environment through an interactive process of adjustments applied to its connection weights. Ideally, the network becomes more knowledgeable about its environment after each iteration in the learning process. It is very difficult to agree on a precise definition of the term learning. In the context of artificial neural networks, one possible definition of inductive learning is:

Definition: Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameters change.

A prescribed set of well-defined rules for the solution of a learning problem is called a learning algorithm. Basically, learning algorithms differ from each other in the way in which the adjustment of the weights is formulated. Another factor to be considered in the learning process is the manner in which an ANN architecture (nodes and connections) is built.

Every data sample for ANN training (learning) consists of the input vector X(n) and the corresponding output d(n).

Processing the input vector X(n), a neuron k produces the output that is denoted by $y_k(n)$:

$$y_k = f(\sum_{i=1}^{m} x_i w_{ki})$$

It represents the only output of this simple network, and it is compared to a desired response or target output $d_k(n)$ given in the sample. An error $e_k(n)$ produced at the output is by definition

$$e_k(n) = d_k(n) - y_k(n)$$

The error signal produced actuates a control mechanism of the learning algorithm, the purpose of which is to apply a sequence of corrective adjustments to the input weights of a neuron. The corrective adjustments are designed to make the output signal $y_k(n)$ come closer to the desired response $d_k(n)$ in a step-by-step manner. This objective is achieved by minimizing a cost function E(n), which is the instantaneous value of error energy, defined for this simple example in terms of the error $e_k(n)$:

$$E(n) = (1/2)e_k^2(n)$$

The learning process based on a minimization of the cost function is referred to as error-correction learning. In particular, minimization of E(n) leads to a learning rule commonly referred to as the delta rule or Widrow-Hoff rule. Let $w_{kj}(n)$ denote the value of the weight factor for neuron k excited by input $x_j(n)$ at time step n. According to the delta rule, the adjustment $\Delta w_{kj}(n)$ is defined by

$$\Delta w_{kj}(n) = \eta . e_k(n) . x_j(n)$$

where η is a positive constant that determines the rate of learning. Therefore, the delta rule may be stated as: The adjustment made to a weight factor of an input neuron connection is proportional to the product of the error signal and the input value of the connection in question.

Having computed the adjustment $\Delta w_{kj}(n)$, the updated value of synaptic weight is determined by

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

In effect, $w_{kj}(n)$ and $w_{kj}(n+1)$ may be viewed as the old and new values of synaptic weight w_{kj}, respectively. From Figure 7.4 we recognize that error-correction learning is an example of a closed-loop feedback system. One of those parameters of particular interest is the learning-rate η. This parameter has to be carefully selected to ensure that the stability of convergence of the iterative-learning process is achieved. Therefore, in practice, this parameter plays a key role in determining the performance of error-correction learning.

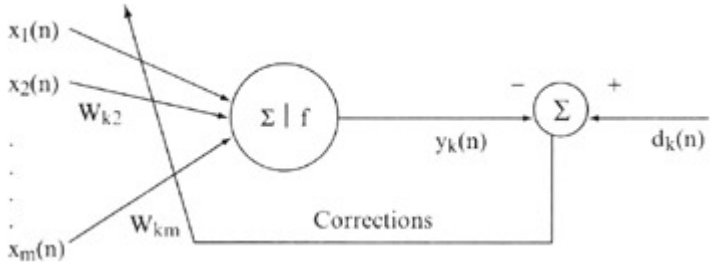
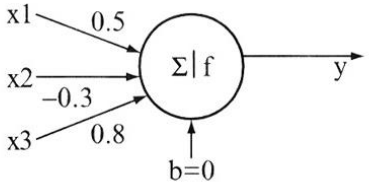


Figure 7.4: Error-correction learning performed through weights adjustments

Let us analyze one simple example of the learning process performed on a single artificial neuron in Figure 7.5a, with a set of the three training (or learning) examples given in Figure 7.5b.



a) Artificial neuron with the feedback

n (sample)	x1	x2	x3	d
1	1	1	0.5	0.7
2	−1	0.7	−0.5	0.2
3	0.3	0.3	−0.3	0.5

b) Training data set for a learning process

Figure 7.5: Initialization of the error correction-learning process for a single neuron

The process of adjusting the weight factors for a given neuron will be performed with the learning rate η = 0.1. The bias value for the neuron is equal 0, and the activation function is linear. The first iteration of a learning process, and only for the first training example, is performed with the following steps:

net(1) = 0.5 . 1 + (-0.3). 1 + 0.8. 0.5 = 0.6

y(1) = f(net(1))= f(0.6) = 0.6

e(1) = d(1) –y(1) = 0.7 – 0.6 = 0.1

$\Delta w_1(1) = 0.1.\ 0.1.\ 1 = 0.01 \Longrightarrow w_1(2) = w_1(1) + \Delta w_1(1) = 0.5 + 0.01 = 0.51$

$\Delta w_2(1) = 0.1.\ 0.1.\ 1 = 0.01 \Longrightarrow w_2(2) = w_2(1) + \Delta w_2(1) = -0.3 + 0.01 = -0.29$

$\Delta w_3(1) = 0.1.\ 0.1.\ 0.5 = 0.005 \Longrightarrow w_3(2) = w_3(1) + \Delta w_3(1) = 0.8 + 0.005 = 0.805$

7.3 Perceptron learning

The approach to learning a function by an ANN is loosely inspired by how (long-term) learning happens in the human brain. The human brain comes with weights of connections among neurons that have been learned through millions of years of evolution, unlike the random initialization of ANNs. The weights in a newly born baby's brain are modified slowly through learning as the individual is exposed to new experiences in life. Similarly, the weights in a neural network are the parameters that need to be learned from data. There may be hundreds of thousands or millions of such parameters.

The perceptron algorithm is a simple but powerful algorithm for binary classification tasks. While it can only classify linearly separable data, it can still be effective for many real-world problems. It involves the following steps:

1. Initialize the weights: The weights of the perceptron are initialized randomly within a range such as -1 to 1. The number of weights is equal to the number of input features.

2. Calculate the weighted sum: For each data point, calculate the weighted sum of the input features by multiplying each feature by its corresponding weight.

3. Apply the activation function: Apply an activation function (e.g., a step function or sigmoid function) to the weighted sum to produce the predicted output.

4. Compare the predicted output to the actual output: Compare the predicted output to the actual output of the data point to calculate the error.

5. Update the weights: Use the error to update the weights using a learning rate. The learning rate controls how much the weights are updated with each iteration of the training process.

6. Repeat steps: Iterate over all the data points and repeat steps 2–5 until the algorithm converges to a set of weights that accurately predict the output classes.

Lesson 8: Feature selection

8.1 Introduction

A feature or an attribute is an individual measurable property or characteristic of a phenomenon. A measurable property can be quantified in many ways, resulting in different types of values. For example, a quantity can be represented as nominal, ordinal, ratio, or interval. For any task of interest, we can usually identify a number of features to describe a data example suitable for the application domain. Ideally, as a whole, the features used to describe data examples should give us valuable insights into the phenomenon at hand. For example, in classification, data examples are separated into different categories or classes based on distinguishing feature values they contain. Similarly, network traffic examples can be classified either as an attack or normal using values of network-traffic features or attributes. However, if there are redundant and/or irrelevant features, they complicate the data-analysis process. If several features convey the same essential information, the analysis algorithms may become confused regarding how much information each similar feature should contribute to the process. An irrelevant feature also muddies the process by sending the analysis on the wrong path, unless we can determine that it is irrelevant and ignore it from consideration.

Reducing the number of features to consider is useful in Data Science. Often, when data are collected for a phenomenon or a dataset is given for analysis, the number of features is high. Using every feature may make data analysis slower, especially when dealing with datasets with many examples. If the number of features can be reduced, the analysis will likely be more efficient. Feature selection and feature extraction are used as dimensionality-reduction methods. Feature selection, also known as an attribute, or variable subset selection, is used in machine learning or statistics for the selection of a subset of relevant features so that efficient models can be constructed to describe the data. Two important issues to keep in mind when performing feature selection are (i) minimum redundancy, which requires that the selected features are as independent of one another as possible, and (ii) maximum relevance, which requires that each selected feature plays a role in the classification task. In addition, data scientists use feature selection for dimensionality reduction and data minimization for learning, improving predictive accuracy and increasing the comprehensibility of models.

Artificial Neural Networks (ANNs), in particular, Deep-Learning methods such as Convolutional Neural Networks (CNNs), are excellent feature extractors on their own. ANNs accept raw-data examples, such as pixels representing images, and extract features of increasing complexity from lower to higher layers. In a CNN, the highest layer in the convolutional part of the network extracts the highest-level features. These features are used by the dense part of the network (also called a multilayer perceptron or MLP) above that to perform classification. When we discuss feature selection or feature extraction in this chapter, our focus is not on ANNs or CNNs that perform feature extraction as an integral part of the learning process. This chapter deals with situations where data examples are given regarding several features. We use specialized feature-selection (extraction) algorithms to extract or select features that are used to send the "reduced" examples to downstream algorithms for tasks such as classification and clustering. These algorithms are, in some sense, akin to the convolutional part of a CNN. In particular, this chapter discusses feature selection in depth. Data scientists who develop cost-effective solutions for real-life problems often need to use such algorithms.

8.1.1 Feature extraction vs. feature selection

A feature-selection method selects a subset of relevant features from the original feature set. In contrast, a feature-extraction method creates or extracts features based on combinations or transformations of the original feature set. Both approaches are used to overcome the curse of dimensionality in machine learning, and statistical data analysis. The curse of dimensionality refers to the phenomenon that when the number of dimensions or features is large in a data-analysis problem, it is difficult to distinguish among the data examples since the space of examples becomes large and sparse. Thus, to solve a data analysis problem effectively, at a basic level, we must be able to perform computations to find which examples are similar and which ones are dissimilar. The main objective of feature selection is to identify m most informative features out of the n original features, where $m < n$. A feature-selection algorithm may be simple and greedy, using a statistical measure to find correlations among the features themselves (unsupervised approach), or the features or the response variable (supervised approach), or maybe more elaborate combining search techniques to discover a new subset of features from the original feature set, followed by evaluation of the selected subset of features using a score computed by an evaluation measure. A simple algorithm may test each possible subset of features finding the one that minimizes the error rate or maximizes the classification accuracy or any other downstream task.

Feature extraction differs from feature selection. Feature extraction attempts to reduce the dimensionality of the original feature set by creating and computing new features to ease subsequent processing. In contrast, a feature-selection method reduces the dimensionality by selecting a most relevant and nonredundant subset of features to achieve the best possible accuracy. In addition, feature extraction attempts to create new features by applying various transformation techniques to the original raw features. In contrast, a feature-selection technique attempts to select a subset of relevant and nonredundant features by applying a suitable statistical or information-theoretic measure with reference to a user-defined threshold to filter out unwanted features or following a more complex algorithm.

Figure 8.1 demonstrates both the dimensionality-reduction processes, which can be linear and nonlinear projections. The selection of an optimal subset of relevant features can help improve classification accuracy as well as classification efficiency. Feature-selection algorithms have been applied to many Data Science problems in many application domains.

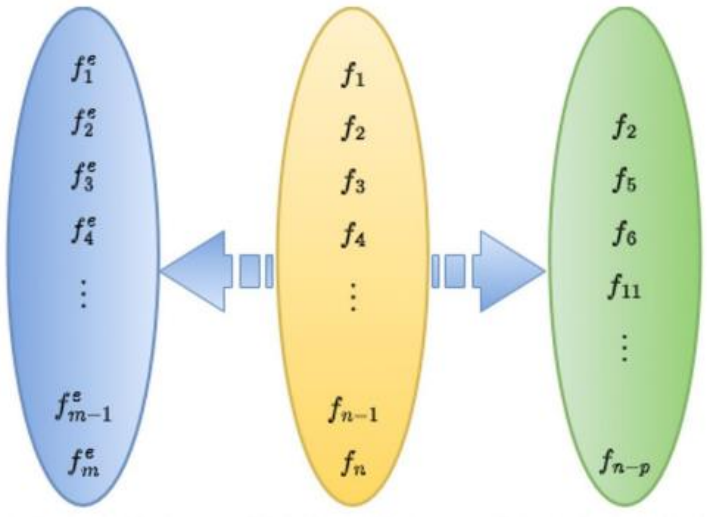


Figure 8.1. Feature Extraction vs. Selection. Feature selection extracts m = n − p features from the original set of n features. Feature extraction creates m new features by combining the original features.

8.2 Steps in feature selection

This section discusses steps in a generic feature-selection approach, whose components can be instantiated in various ways, simple to complex. As already mentioned, the objective of feature selection is to select a subset of features that are relevant to the task at hand. Different algorithms vary in the specifics of the steps. Some algorithms may short-circuit the steps by combining or eliminating some feature. Usually, the downstream task is classification. In general, feature selection involves four major steps:

i) Generation of feature subsets;

ii) Evaluation of the feature subsets that have been generated using various criteria;

iii) Determination of the goodness of a feature subset based on the values of evaluation criteria; and

iv) Finally, validation of the selected feature subset on a downstream task.

Figure 8.2 shows these steps with additional details. We discuss each of these steps next.

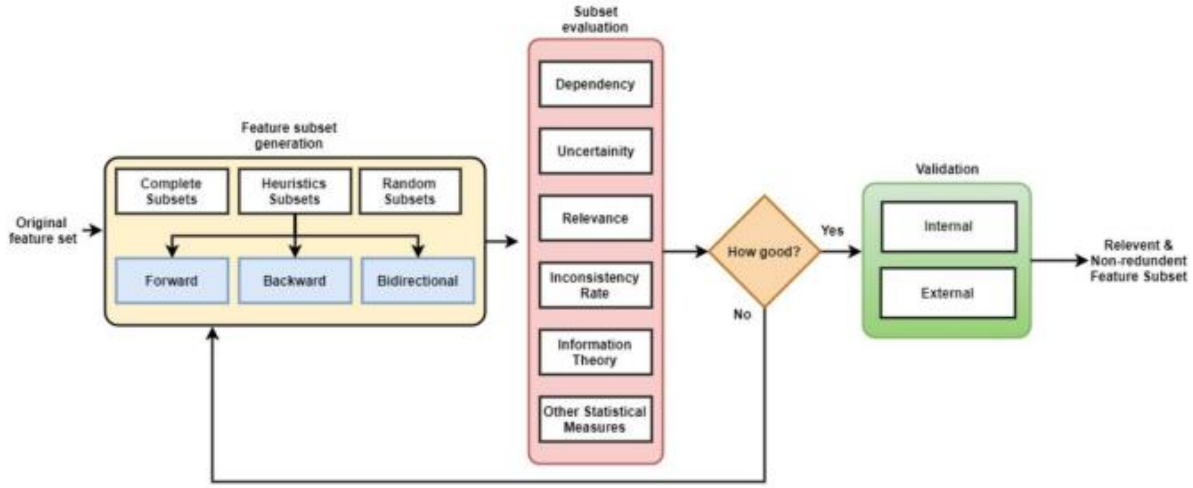


Figure 8.2. The summarize the major steps in Feature Selection along with different alternatives for each step.

8.2.1 Generation of feature subsets

This step is responsible for generating many candidate subsets of features from the raw feature set. One can take several approaches to generate candidate subsets from a dataset that has n features. These include an exhaustive search over all possible subsets that can be generated, performing a heuristic search over the set of all possible subsets, or performing a random search. A simple approach may create the desired subset of features by choosing features to add to or subtract from a working set of features using a statistical measure. Such an approach consists of only one candidate subset at a time.

8.2.1.1 Feature-subset generation using exhaustive search

Given a set of features, this approach generates all possible subsets of features. Thus for a dataset with n features, it will generate 2n unique subsets. Obviously, for a high dimensional dataset, the number of possible subsets is exponentially large and consequently, the cost of the evaluation will also be very high. As a result, this approach is impractical for higher-dimensional datasets. However, this is the only approach that guarantees the selection of the absolute best possible subset of features. In practice, it is likely that Feature Selection takes

shortcuts, making greedy choices, and adding or subtracting one feature at a time to a working set, instead of generating subsets in a more elaborate manner.

8.2.1.2 Feature-subset generation using heuristic search

An exhaustive search of feature subsets is exponential in computational complexity and therefore, infeasible if the number featured in the original dataset is larger than a relatively small number, say 15, since 2^{15} = 32, 768, resulting in the generation of a large number of features followed by subsequent evaluation. In practice, datasets with tens or even hundreds of features or more are common in real life.

A heuristic is a rule of thumb. It usually involves a quick and dirty computation to come to an informed but greedy or "hasty" relevant decision regarding a problem at hand, usually producing good results, while not guaranteed to do so. For example, assume that we have a heuristic that first computes the correlation between each individual feature and the label, which is the dependent variable. The heuristic then chooses the feature that shows the most positive correlation with the label. The assumption is that the features and the label are all numeric, and hence it is a simple regression problem. This is a heuristic because the entire set of numeric features may relate to the numeric label in various ways as a whole, but the heuristic looks at each feature in isolation assuming independence among features, and scores it. It makes a decision regarding the current most influential feature quickly, without looking at complex relationships that may exist between the features and the label. In addition, once a decision is made, often it cannot be undone.

Depending on how the heuristic is used as features are selected or deselected to produce the "optimal" subsect results in three commonly used heuristic approaches: (i) forward feature selection, (ii) backward feature selection, and (iii) bidirectional feature selection. In the forward heuristic approach, the approach starts with an empty feature set and successively adds highest-scoring features (typically, one by one) using the heuristic, whereas in the backward heuristic approach, it starts with the entire feature set and successively, eliminates lowest scoring features (typically one by one) from the whole set using the heuristic criterion. Each of these heuristic approaches has its own advantages and limitations. The bidirectional approach searches for the "optimal" subset of features using a combination of forward and backward searches to produce the result more efficiently. There are different ways to implement a bidirectional search. One simple way is to start with, say the empty set of features and add one or more iteratively, but as a feature is added, the current subset of features is analyzed to see if any of the current features in it should be eliminated. It can be performed in the reverse fashion also, by starting with the entire set of features and eliminating features one or more at a time, but every time a feature is eliminated, the approach looks to see if there are features that can be added. There are other more complex ways to achieve bidirectional feature selection. However, like the forward and backward heuristic approaches, the bidirectional approach also does not give a guarantee to select the best possible feature subset. The quality of a feature subset given by this approach depends on the (i) effectiveness of the heuristic used for selection and (ii) criterion used for its evaluation.

In practice, there may be no explicit subset generation at all. Instead, the algorithm may start from an empty set of features and add a new feature at a time, growing the subsect incrementally.

8.2.1.3 Random feature-subset generation

This approach, unlike the approaches discussed already, attempts to select feature subsets at random and evaluate their effectiveness. On average, this approach may be more time efficient, but it also does not give a guaranteed selection of the best possible feature subset. Random

feature selection is used in building an ensemble of decision trees known as Random Forests, where a number of trees are built by sampling features and examples from a featured dataset. It has also been seen that the use of what is called stratified random sampling involves the division of the feature space into smaller subgroups (known as strata), and then sampling from the subgroups, may be helpful in finding better feature subsets in less time. However, additional computation such as clustering is needed to find the subgroups.

8.2.2 Feature-subset evaluation

This step deals with the evaluation of the feature subsets generated in the previous step using an appropriate measure or metric. Depending on the nature of the computation, evaluation metrics are categorized into two types: independent metrics and dependent metrics.

An independent metric evaluates the goodness of a feature subset based on the intrinsic properties of the selected features. On the other hand, to evaluate the quality of a feature subset, a dependent metric analyzes the results generated by the feature subset when used on a downstream task. Based on the approach used for the evaluation of the results, a dependent criterion can be external or internal in nature. In external dependent metric-based evaluation, the quality of results is evaluated using prior knowledge, whereas, in internal dependent metric-based evaluation, prior knowledge is either not used or is not available. Rather, it considers the intrinsic properties such as cohesiveness or homogeneity in the results or separability among the instances.

Often, a simple approach to feature selection may not evaluate feature subsets in a formal way. Instead, the approach may start from an empty set, evaluate the individual features in a greedy fashion, and decide which feature to add to the subset to enhance it in a greedy fashion, without evaluating the entire feature set at the time.

8.2.3 Feature-selection methods

Based on the approaches used for the selection of the features, the methods are categorized into four types: (a) filter methods, (b) wrapper methods, (c) embedded methods, and (d) hybrid methods. In addition, there are two other approaches, namely incremental and ensemble feature selection, that are used to handle dynamic datasets, and to obtain the best possible feature subset.

8.2.3.1 Filter approach

The filter approach selects a subset of features with the help of an evaluation function, without using a machine-learning algorithm. Most filtering algorithms use statistical measures such as correlation and information gain. A complex filter approach may even use particle-swarm optimization, ant-colony optimization, simulated annealing, and genetic algorithms.

Depending on the data types of the feature values, it is necessary to use different statistical methods to compute how the features are similar. For example, if the original feavalues are all numeric, Pearson's or Spearman's correlation coefficients can be used. In case of categorical features, either χ^2 (Chi) or mutual information can be used to rank the features. While feature set is a combination of both numeric and categorical features, there are other appropriate methods that can be used. Below, we discuss Pearson correlation and mutual information-based feature section methods.

8.2.3.1.1 Pearson correlation-based feature selection

Pearson's correlation coefficient takes two vectors and produces a number between −1 to 1 to indicate the degree of correlation between the two. A value close to 0, whether positive or negative, implies a weak correlation between the two vectors, with a value of 0 implying no

correlation. A value close to 1 indicates a very strong positive correlation, indicating that when one feature's value changes in a certain way, the other feature's value changes in a very similar way. Values above 0.5 indicate strong positive correlations, and a value between 0 and 0.3 is considered a weak positive correlation. A value close to −1 indicates a negative correlation, indicating the values of the two features change in opposite ways. Often, a high positive correlation is used to determine if one feature can be dropped, although some approaches use a negative correlation as well. If we want to use either a positive or negative correlation, we can use the absolute value of the correlation. Low correlation can also be used to find a set of nonredundant features.

Suppose we want to determine if two features f_1 and f_2 are correlated, given values of the two features for the examples in a dataset. To start, we create two vectors, $\vec{f_1}$ containing the values of feature f_1 for all examples in the dataset one by one, and, $\vec{f_2}$ containing values of the feature f_2 for the entire set of examples in the same order. Assuming there are N examples in the dataset, Pearson's correlation coefficient between the two vectors is given by:

$$r_{Pearson} = \frac{N\sum_{i=1}^{N}(f_{i1}f_{i2}) - (\sum_{i=1}^{N} f_{i1})(\sum_{i=1}^{N} f_{i2})}{\left[\sqrt{N\sum_{i=1}^{N} f_{i1}^2 - (\sum_{i=1}^{N} f_{i1})^2}\right]\left[\sqrt{N\sum_{i=1}^{N} f_{i2}^2 - (\sum_{i=1}^{N} f_{i2})^2}\right]} \qquad (8.1)$$

It assumes that both feature vectors are quantitative, the values of the features are normally distributed, there are no outliers, and the relationship between the values of the two feature vectors is linear.

Supervised feature selection can be performed when the dataset is labeled, as if for classification or regression. For Pearson correlation to be used, the label must be numeric, i.e., the downstream task is regression.

In addition to correlation between pairs of features, we compute correlation between each feature and the label y. We obtain the vector of feature values $\vec{f_i}$ for feature f_i, and the vector of label values $\vec{y}$ for the label y.

A simple feature-selection algorithm may be described at high level in the following way; details have to be filled in to create the actual algorithm. The algorithm starts with an empty selected-working-set of features. It also has an empty excluded-working-set of features. It also has a yet-to-be-considered-set of features; this set's initial value is the entire set of features. From the yet-to-be-considered-set, features are chosen to be included in either the selected working set or the excluded working set. The algorithm finds a new feature from the yet-to-be-considered-set to add to the selected working set; this feature is most highly correlated to the label among the features in the yet-to-be-considered-set. Next, the algorithm looks for features in the yet-to-be-considered-set that are most highly correlated to the features of the selected working set, on average; one of these features is added to the excluded working set. The algorithm repeats the process of selecting a feature from the yet-to-be-considered-set to include in the selected working set and putting one in the excluded working set as many times as necessary. The algorithm stops when the yet-to-be-considered set becomes empty.

8.2.3.1.2 Mutual information in feature selection

Knowing the value of one feature for a data example may sometimes not give any clue about what the value of the label (or, another feature) may be, but sometimes it may to a small extent, or even to a large extent. This idea is quite similar to the correlation between the values of a feature and the label, or between the values of two features, as we have discussed above.

Information Theory deals with a similar idea in terms of what is called information content. The problem can be cast in terms of information contained in the following way: How much information about the label y (or feature fj) can be gleaned if we have information about feature fi. This idea of the amount of information obtained about the value of one random variable (here, label y or feature fj) from the value of another random variable (here, feature fi) is quantified in terms of mutual information between the two random variables.

The amount of mutual information between a feature fi and the label y, assuming both are discrete is given as

$$MI(f_i, y) = \sum_{f_{ij}} \sum_{y_k} p(f_{ij}, y_k) \, log \frac{p(f_{ij}, y_k)}{p(f_{ij}), p(y_k)}, \qquad (8.2)$$

where fij is the jth possible value for feature fi and yk is the kth possible value of the label y. In the formula above, in the log computation, the numerator is the probability of a feature value fij occurring together with the label value yk or the joint probability distribution, whereas the denominator contains the product of the probability of the feature value fij and the label value yk occurring independently. Thus, the ratio gives us an estimate of how much more likely the two values occur together if there is some dependency between the two compared to if they were independent of each other. It is a measure of the dependency between fij and yk. Taking log of this ratio still gives us a measure of the same dependency; it only changes the scale of the ratio. The formula above adds the values for all fij and yk pairs to give us a measure of how much information the feature contains about the label across all values of both.

We can also compute MI (fi, fj) to find the mutual information between two features fi and fj. Similar to our discussion on Pearson-correlation-based feature selection earlier, mutual information-based feature selection can be either supervised or unsupervised. Feature–feature and feature–label mutual information values can be combined in various ways to develop more complex algorithms for feature selection. Mutual information and correlation values may also be combined in various ways in sophisticated algorithms.

Mutual information can be computed for continuous values as well. If both feature fi and label y are continuous numerical values, the equation for mutual information given above can be changed as follows:

$$MI(f_i, y) = \int_{f_{ij}} \int_{y_k} p(f_{ij}, y_k) log \frac{p(f_{ij}, y_k)}{p(f_{ij}), p(y_k)} df_i dy. \qquad (8.3)$$

In this case, we see that we have to perform double integration over all values of the feature fi and all values of label y. Since the values are continuous, we have to assume probability density functions for the feature as well as the label. For example, we can assume that both have values that are distributed normally, and hence, use the formula for a Gaussian probability distribution. To specify a Gaussian probability distribution, we need to know the mean and standard deviation for it. The two distributions will have different means and standard deviations.

Mutual information can also be computed for all pairs of feature pairs, and can be used to develop complex algorithms along with the mutual information values between individual features and the label. Note that mutual information can be computed between a discrete variable and a continuous-valued variable also. For the discrete variable, we perform summation over all possible values, whereas for a continuous-valued variable, we perform the integration. Thus, it is possible to develop complex feature-selection algorithms using mutual information to cover datasets with a variety of features.

8.2.3.2 Wrapper approach

Unlike filter feature selection which relies on statistical measures to assess the relevance of features, wrapper feature selection uses machine learning algorithms to evaluate feature subsets based on their impact on model performance. The wrapper approach uses a learning algorithm to evaluate the accuracy produced by the use of the selected features in classification. Wrapper methods can give high classification accuracy for particular classifiers, but generally, they have high computational complexity. Figure 8.3 illustrates the working mechanism of filter and wrapper approaches. Kwak and Cho (2002) developed an algorithm called MIFS-U to overcome the limitations of MIFS to obtain better mutual information between input features and output classes than MIFS. Peng et al. (2005) introduced a mutual information-based feature-selection method called mRMR (Max-Relevance and Min-Redundancy) that minimizes redundancy among features and maximizes dependency between a feature subset and a class label. The method consists of two stages. In the first stage, the method incrementally locates a range of successive feature subsets where a consistently low classification rate is obtained. The subset with the minimum error rate is used as a candidate feature subset in stage 2 to compact further using a forward selection and backward elimination-based wrapper method. Estevez et al. (2009) proposed a mutual information-based feature-selection method as a measure of relevance and redundancy among features. Vignolo et al. (2013) introduced a novel feature-selection method based on multiobjective evolutionary wrappers using a genetic algorithm.

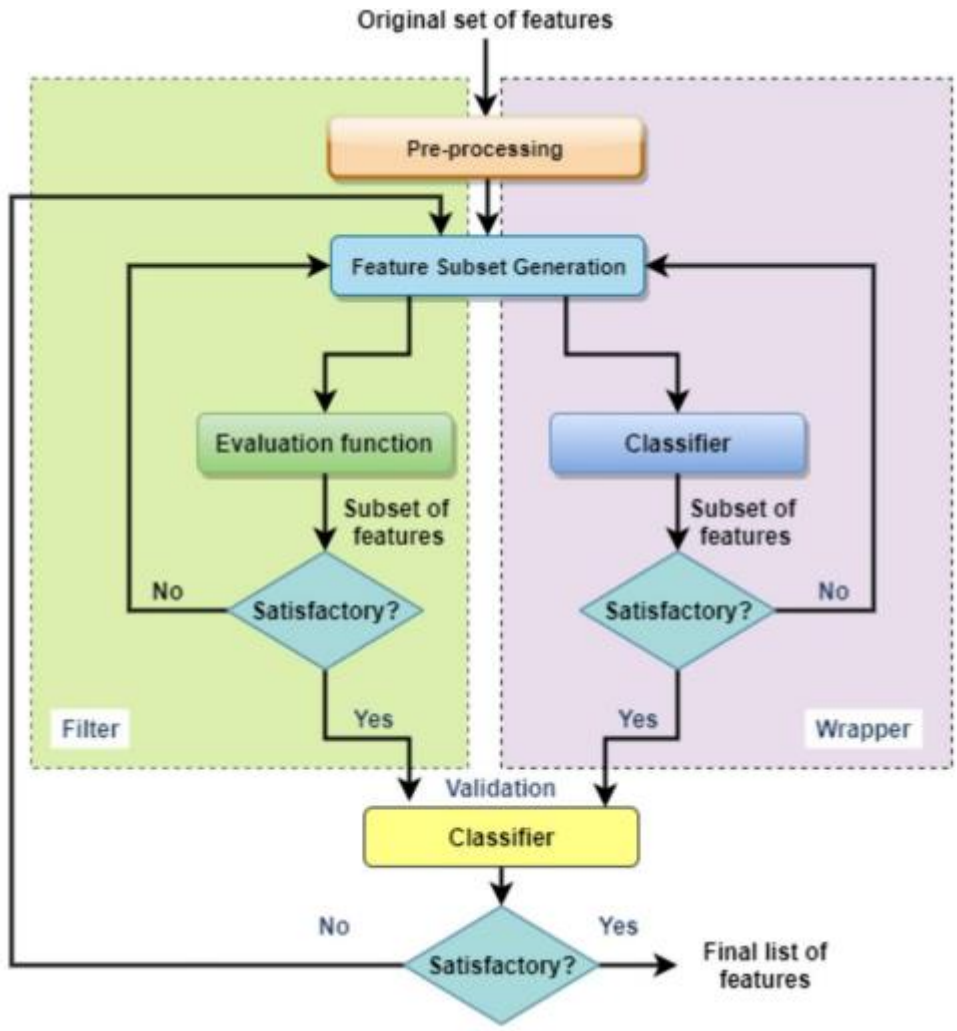


Figure 8.3. Basic steps of Filter and Wrapper feature selection methods.

8.2.3.3 Embedded approach

Embedded feature selection is a technique that integrates feature selection directly into the training process of a machine learning model. Instead of judging feature significance separately, embedded approaches learn the most essential features automatically during model training. By selecting the most useful features for the specific job at hand, this technique can increase both model performance and computational efficiency while reducing overfitting. Lasso regression and tree-based algorithms such as Random Forests are examples of embedded approaches. A conceptual framework of this approach is presented in Figure 8.4. Although the embedding of a prediction model is helpful in selecting highly ranked features, the bias (if any) of the model may lead to nonselection or the dropping of some important feature(s).

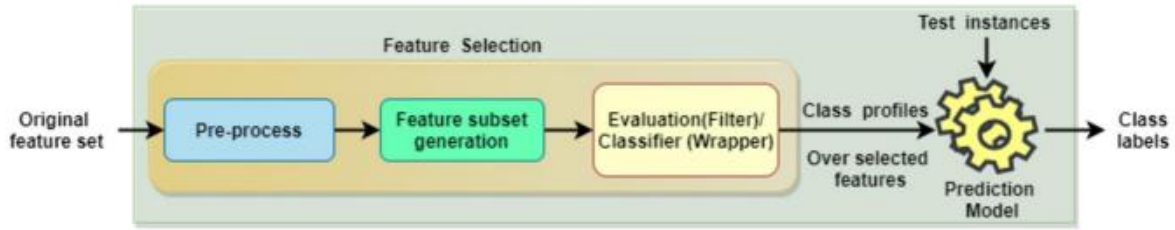


Figure 8.4. Typical pipeline of embedded feature selection

8.2.3.4 Hybrid approach

The hybrid approach is a combination of both filter- and wrapper-based methods. The filter approach selects a candidate feature set from the original feature set and the candidate feature set is refined by the wrapper approach. It exploits the advantages of these two approaches. Hybrid feature selection combines other feature-selection approaches that can improve the performance of a classifier. A hybrid method that combines filter and wrapper approaches tries to reduce the wrapper's complexity and at the same time improves the filter's predictive accuracy. It is a two-step feature-selection method where the output of the filter method is used as input to the wrapper method to boost the overall performance of a classifier. The filter step chooses the high-ranked features using an objective function, and the wrapper method reevaluates all possible subsets of features using a machine-learning model. Hybrid feature selection is useful for the classification of high-dimensional data with an informative subset of features. The most important characteristic of the hybrid method is that it can overcome the common issues of the filter and wrapper methods. Hybrid methods can also handle data instances that have diverse distribution patterns. Most hybrid FS methods have been developed to select subsets of informative features from high-dimensional voluminous data. Alejandro et al. (2019) developed a hybrid FS method for the classification of botnet-based IoT network traffic. The method uses the Pearson Correlation Coefficient in the filter step and the k-NN classifier in the wrapper step. Their method gives high detection accuracy in classifying benign and malignant instances. Liu et al. (2018) proposed an effective rule-based hybrid feature selection method. The method known as HGAWE uses a hybrid Genetic Algorithm (GA) with wrapper-embedded approaches for feature selection. The main objective of HGAWE is to improve learning performance and enhance the searching performance to identify the relevant feature subsets. The population of GA solutions was fine-tuned by selecting some signature features using an embedded method and constructing the learning model based on efficient gradient-regularization approaches. The population of GA solutions is induced by a wrapper method using heuristic search strategies. Many hybrid evolutionary algorithms have also been developed for effective feature-subset selection.

In addition to these four basic approaches of feature selection, a data scientist prefers to use two more effective approaches, i.e., ensemble approach and incremental approach, especially for handling dynamic datasets and achieving the best possible feature subset.

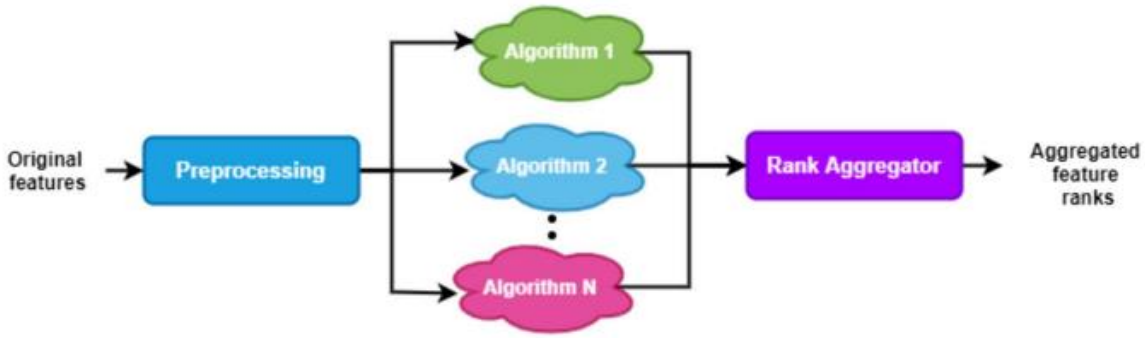


Figure 8.5. Ensemble Feature Selection.

8.2.3.5 Ensemble approach

The ensemble feature-selection approach uses a diverse set of feature-selection algorithms and combines them to achieve the best possible accuracy. The basic concept behind an ensemble approach is that a team's decision is usually better than the decision of an individual. Ideally, an ensemble approach should be designed to overcome the poor performance of any individual algorithm. A high-quality ensemble is often one in which the base algorithms are from diverse families and are independent of each other. This is necessary because the errors made by base algorithms should be uncorrelated so that any possible miscalculation by any one or a few of them is outweighed by the better performance of the other algorithms. The goal of constructing the ensemble is to obtain a composite model from several simpler submodels to obtain high overall accuracy. A combination function is used to aggregate the decisions made by the individual algorithms to obtain an aggregated list of ranked features. Figure 8.5 illustrates the process of ensemble feature selection. For rank aggregation, a good combination function should try to eliminate the bias of individual ranker algorithms and generate the best possible ranked list of features.

8.2.3.6 Incremental feature selection

An incremental feature-selection algorithm computes an optimal subset of features as new data examples become available dynamically. In other words, the data examples are not available at the same time as we have assumed in the prior discussions on feature selection. Incremental feature selection computes an optimal subset of features that is relevant and nonredundant based on information obtained from the currently available instances, and when a new instance or a number of new instances arrive, it updates the feature set dynamically without processing the earlier instances again. The purpose of incremental feature selection is to avoid repetitive feature selection as new examples come onboard. For example, incremental feature selection is important in applications dealing with gene expression, network intrusion detection, and text categorization where real-time classification needs to be performed on the data instances. Incremental feature selection is useful for dynamic datasets whose character changes over a period of time. For such datasets, traditional feature-selection methods are inefficient for knowledge discovery as the same learning algorithm needs to be run repeatedly.

An incremental feature-selection algorithm was proposed by Liu & Setiono. (1998) using a probabilistic approach. The algorithm is an incremental version of a filter algorithm called Las Vegas Filter (LVF). The main feature of a Las Vegas algorithm is that it must eventually

produce the proper solution. In the context of feature selection, this means that the algorithm must create a minimal subset of features that optimizes some parameter, such as classification accuracy, which was designed to ensure that features selected from the reduced data should not generate inconsistencies more than those generated optimally from the whole dataset. However, in the original LVF algorithm, if the dataset size becomes small, the number of inconsistency checks becomes small as well. As a result, the set of features selected from a reduced dataset may not be suitable for the whole dataset. The incremental version of LVF overcomes this problem without sacrificing the quality of feature subsets in terms of the number of features and their relevance.

8.3 Principal-component analysis for feature reduction

Feature Selection attempts to find an optimal subset of the entire feature set whereas Feature Reduction or Dimensionality Reduction obtains a set of newly created features where each feature is a linear combination of the original features, and the total number of new features is smaller than the number of original features.

One of the most common feature or dimensionality reduction techniques is Principal Component Analysis (PCA). Given a dataset with n features, PCA can be used to obtain k features with $k < n$ or even $k \ll n$ such that the new set of features expresses the variance in the original dataset well. In other words, the new features are almost as expressive as the original features, although some loss in expressiveness is expected. Another way to look at PCA is that it is able to transform the original features that may be correlated, expressing similar information, to obtain a smaller set of uncorrelated features. PCA is also able to rank the newly transformed features in terms of expressiveness, where a higher-ranked feature explains the variations in the dataset better than a lower-ranked feature. Thus, of the k resulting transformed features, the highest ranked one explains most of the variance (how the data values are spread out) in the dataset; the next ranked feature explains the next highest amount of variance in the dataset, and so on. Therefore, it is possible to use PCA to obtain a small set of k ranked features, often much smaller, that explain p% of variation in the data, with p being a high number like 90, 95, or 99.

To motivate what PCA accomplishes, let us look at a two-dimensional dataset given in Figure 8.6(a). The dataset is given in terms of an original feature set {f1, f2}. To reduce the number of features, the first step is to extract a principal component, somewhat similar to a least-squares linear regression, but a little different. In the least-squares linear regression, we obtain the line that gives the trend in the dataset with the smallest amount of sum of squared errors, whereas, PC line, also known as the first principal component, is the line that represents the direction of the maximum variance in the data. A PC is a linear combination of the original variables in a dataset. This line is obtained by finding the direction in which the projected points have the maximum variance, and then rotating the coordinate system so that this direction becomes the first coordinate axis.

To perform PCA, we project the points to the PC line we draw. In other words, we drop perpendiculars to the line we are trying to find (see Figure 8.6(b)); this line is also called a principal-component line. The projected points are also shown on the PC line in Figure 8.6(b). Once the points have been projected to the PC line, each point can be thought of simply as its magnitude along the PC line, which becomes a new axis. Among all possible lines we can draw, the one chosen as the PC line minimizes the variance of the projected set of points, considering the magnitude of the points along the new PC axis. The new feature, given by PCA-1, can be used as the single (only) transformed feature for the dataset. The PC line can be thought of as an axis representing a new composite feature or dimension, and the original points can now be redescribed by their values on this new axis. The PC is also called the first principal

component for the original dataset. This principal component accounts for most of the variations in the dataset.

Since the original dataset is two dimensional, it is possible to obtain a second principal component. The second principal component is orthogonal or perpendicular to the first principal component and is shown in Figure 8.7 as PCA-2. By looking at the projections of the original points on the line representing the direction of the second principal component, we see that the distances from the new origin, which is the intersection of the two principal components PCA-1 and PCA-2, are small, and thus the second principal component does not explain as much variation in the original dataset as the first principal component. As a result, if we choose to, we can ignore the second principal component as the second transformed feature without losing a significant amount of explanatory information.

In the example in Figures 8.6 and 8.7, we start with a dataset in two dimensions, obtain two principal components, and decide to ignore the second, thus effectively reducing the number of dimensions in the dataset. However, in a realistic dataset, the number of features is likely to be higher, possibly in the tens, hundreds or more. In such cases, we can use PCA to obtain a sequence of ranked principal components in descending order of informativeness or explaining capacity for variances in the data. Unless there are features in the original dataset that are completely collinear (i.e., two features whose values are multiples of each other), we can obtain n ranked principal components from the original n features. However, it is likely that the first k features, $k \ll n$, explain most of the variances in the dataset, and we do not have to produce any principal components beyond the first k. For example, it is possible that given an original feature set with 100 distinct features, only 5 or so features may be able to explain 99% of the variance in the dataset, thus achieving feature reduction from 100 to 5, a reduction of 95% in the feature-space size with loss of only 1% in explainability.

The prime task of PCA is to compute a principal component. A number of such PCs can be extracted (possibly up to the number of independent variables) from a dataset. We first need to standardize the data by subtracting the mean from each variable and dividing by the standard deviation. This ensures that each variable has equal weight in calculating the principal components. Next, we need to calculate the covariance matrix of the standardized data. The covariance matrix describes the relationships between the variables in the data set and is used to determine the directions of the principal components. The eigenvectors of the covariance matrix represent the directions of the principal components, and the eigenvalues represent the amount of variance explained by each principal component. To select the principal components with the highest variance, we need to sort the eigenvectors in descending order based on their corresponding eigenvalues. The top k eigenvectors are selected to explain the desired amount of variance in the data. Finally, we project the data onto the new coordinate system defined by the selected principal components. This transforms the original dataset into a new space with reduced dimensionality.

As new principal components are computed one by one, it must be noted that each principal component is orthogonal to every prior principal component. Thus if we have n, n = 100, say, original features, we need to imagine an n-dimensional space, which of course, cannot be visualized, and imagine constructing a new principal component, which is a linear combination of the original features, that is orthogonal to all other principal components constructed so far.

We have described the concept of principal components conceptually, outlining the process of computation in sequence. The efficient mathematical process of computing the principal components for a given dataset involves the use of linear algebra, primarily the concepts of eigenvalues and eigenvectors. These are not discussed here, and are left for the readers to pursue on their own.

After performing PCA, data examples need to be described in terms of the smaller number of transformed features. It can be observed that in the reduced and transformed feature space, originally similar data examples become more similar, and originally unlike examples become more unlike. It helps to remember that the new features are ordered from most informative to least. When features that do not show meaningful variations are removed, it is quite likely that it removes noise in the examples. Since data examples, expressed in terms of the new features, are shorter vectors, the new feature values make downstream computation faster.

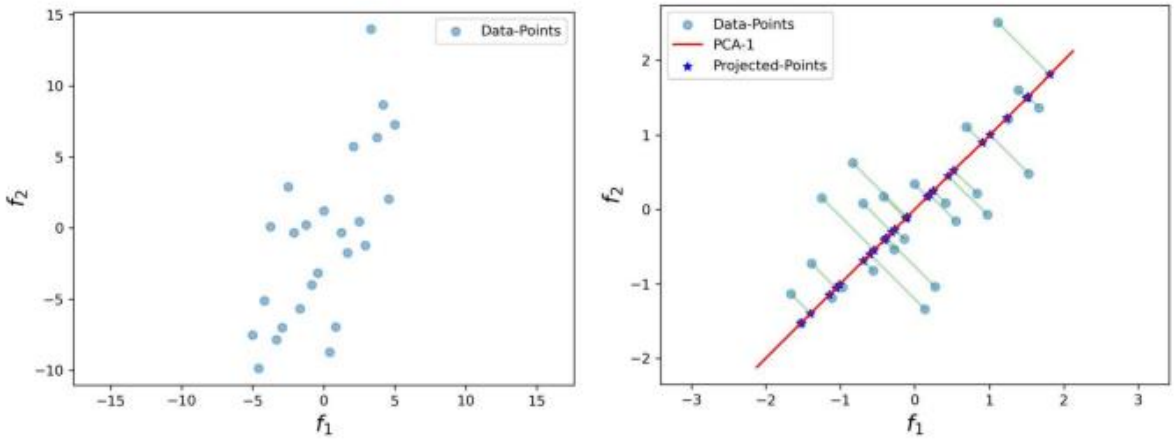


Figure 8.6 (a). A scatter plot of the original points in 2D. (b) A principal-component line is obtained by minimizing the variance in the set of values of the projected points. The principal-component line can also be used to reduce the two-dimensional data points into one dimension. The projection of the data points onto the principal component line are shown as blue solid circle (mid gray in print version).

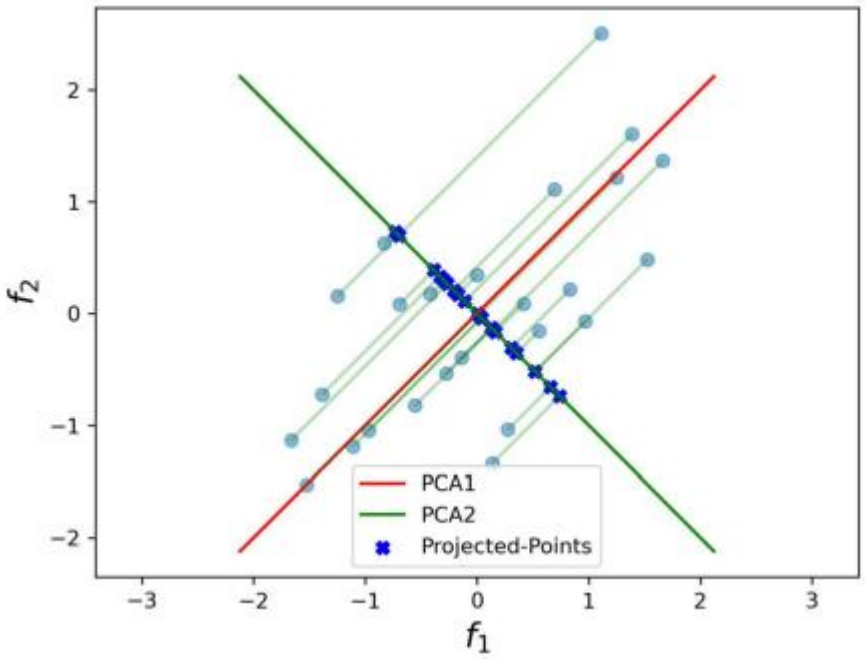


Figure 8.7. The line perpendicular to the principal-component 1 (PCA 1) line serves as the second principal component (PCA 2) for the dataset. It is necessary to drop perpendiculars to project the examples to the second principal component. We see that the spread of the projected points (shown in blue; mid gray in print version) is much tighter, and variances in values of the projected points are much smaller than the spread and variance along the first principal component in Figure 8.6(b).

8.4 Summary

Feature selection plays a crucial role in Data Science. The use of an appropriate feature selection method can reduce the cost of computation and resource requirements significantly. This chapter has discussed the basics of feature selection. It has also presented the steps in feature selection, various approaches for feature-subset generation and their evaluation, and finally, the metrics used to validate the selected features. The chapter has also briefly presented a few recent feature-selection approaches such as incremental feature selection and ensemble feature selection. The chapter ends by introducing Principal Component Analysis (PCA), the most common method used for feature or dimensionality reduction.

Lesson 9: Cluster analysis

9.1 Introduction

Classification and regression perform the task of prediction based on prior experience. In the real world of data-driven decision making, discovering underlying data patterns and interrelationships among the data elements is a crucial analytical task besides prediction. For instance, a marketing strategist of a retail corporation might like to understand the customers' purchasing patterns for individualized marketing. Accordingly, an organization may offer attractive price discounts or recommend specific products to a group of customers based on an analysis of their buying habits. More specifically, cluster analysis may help separate groups of customers based on purchase histories, addresses, and locations of favorite stores. Google uses cluster analysis for generalization, data compression, and privacy preservation of YouTube videos. YouTube-video metadata are generalized through clustering to promote and recommend less popular videos with hit videos. Data compression helps reduce YouTube data storage by grouping data for similar videos with a single cluster identifier. YouTube-user clustering can hide the user identity by grouping and tagging the search-history details with cluster identifiers corresponding to similar search histories.

Besides organizing the data into meaningful groups, cluster analysis may also help label extensive unlabeled datasets when few representative labeled samples are available for predictive learning, popularly known as semisupervised learning.

9.2 What is cluster analysis?

Let us distinguish the task of clustering from that of classification. When we group labeled data instances into prespecified classes (or groups), it is called classification. In the absence of labels, if we try to group the instances based on a notion of similarity or proximity, it is called clustering. It is a data-driven approach, where data items themselves decide their groupings based on certain features or attributes. Unlike other predictive machine learning activities, no parameter learning is involved in clustering. It is also known as data segmentation.

Cluster analysis is an unsupervised learning method for exploring interrelationships among a collection of data instances by organizing them into similar clusters. The objective of clustering is to group the elements so that elements within a group have strong intragroup similarity and weak intergroup similarity with other groups; i.e., individual clusters are largely independent. In turn, a good cluster is well separated from other clusters. The intragroup similarity is measured as an average of pairwise similarity between the instances within a single cluster. A high intragroup similarity among the instances in the same cluster indicates strong dependency among the instances.

9.2.1 Outliers

Outlier detection (also known as anomaly detection) is an important activity when solving real-life application problems. For example, a piece of malware or an instance of fraud can be treated as an outlier as its characteristics deviate from usual software or user activity, respectively.

The concept of an exclusive cluster is not the last word in real-life examples. For instance, social-media users may be interested in cricket and film. In molecular biology, it has been observed that certain groups of genes participate in several DNA-metabolism processes such as replication, repair, and transcription. Out of 1628 proteins in a hand curated yeast-complex dataset, 207 proteins participate in more than one complex. It may not be possible to describe all these complexes using disjoint or exclusive clustering. Hence, it is not always justified to group the entities exclusively into only one particular group following the classical definition of a cluster. The above definition should be referred to as exclusive clustering. To describe the overlapping or shared participation in multiple groups, as discussed above, we need a new definition of clustering, which can be called nonexclusive or overlapping.

9.3 Proximity measures

Proximity (similarity) or the distance measure is the key mathematical concept required for quantifying the degree of likeness between two data points or a set of data points or between two probability distributions. The proximity measure used determines how data samples are related to each other. The specific measure used depends on the downstream activity following clustering. Generally, it is measured in a particular range of continuous values where a smaller distance score represents a higher degree of similarity and vice versa.

1. Euclidean distance (Ed): The Euclidean distance represents the distance between two data points in the geometric space. For two scalar values, the distance between two points can be computed by obtaining the absolute value of their difference. To compute the distance between two two-dimensional data points, the underlying intuition involves the application of Pythagoras's theorem in the Cartesian plane:

$$E_d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2. Manhattan distance (Md): The Manhattan distance, often called the City-Block Distance or Taxi-cab distance, obtains the sum of the absolute differences between the respective dimensions when the input vector points are placed in such a way that one can only move in right-angled directions on a uniform grid. n is the number of dimensions or attributes

$$M_d(\vec{x}, \vec{y}) = \sum_{i=1}^{n} |\vec{x}_i - \vec{y}_i|$$

3. Cosine similarity (Cs): Cosine similarity refers to the cosine of the angle between two vectors. The measure is more concerned with the orientation of the two points in the space than it is with the exact distance from one to the other. It measures the level of similarity between two vectors via an inner product if they were normalized to both have length one:

$$C_s(\vec{x}, \vec{y}) = \frac{\vec{x}.\vec{y}}{\|\vec{x}\|\|\vec{y}\|}$$

4. Dot-Product similarity (DPs): In contrast to cosine similarity, the dot-product-based similarity incorporates the magnitude of the input vectors. The dot-product score increases as the input vector lengths increase. It indicates how two vectors are aligned in the space considering the directions they point:

$$DP_s(\vec{x}, \vec{y}) = |\vec{x}||\vec{y}| \cos(\theta).$$

9.3.2 Statistical measures

These measures employ statistical correlation techniques to evaluate the similarity between two vectors. If x and y are the two data instances (input vectors), then the correlation represents the measure of similarity where a value greater than 0 (positive correlation) represents the high similarity, and a value less than 0 represents the dissimilarity (negative correlation).

1. Pearson correlation coefficient (PCC): The Pearson correlation coefficient is a measure that quantifies the strength of the linear correlation between two vectors or data instances. It does so by computing the ratio between the covariance of the two sets of data and the product of their standard deviations:

$$PCC(\vec{x}, \vec{y}) = \frac{\sum_{i=1}^{n}(\vec{x}_i - \bar{x})(\vec{y}_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(\vec{x}_i - \bar{x})^2(\vec{y}_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ are the mean of $\vec{x}$ and $\vec{y}$, respectively.

2. Spearman rank correlation coefficient (SRCC): The Spearman rank correlation measures the rank correlation between two data instances or vectors by assessing their relationship using a monotonic function regardless of their linear relationship. It is high when observations have a similar rank:

$$SRCC(\vec{x}, \vec{y}) = \frac{n\sum \vec{x}_i\vec{y}_i - \sum \vec{x}_i \sum \vec{y}_i}{\sqrt{n\sum \vec{x}_i^2 - (\sum \vec{x}_i^2)}\sqrt{n\sum \vec{y}_i^2 - (\sum \vec{y}_i^2)}}$$

3. Kendall rank correlation coefficient (KRCC): The Kendall rank correlation coefficient computes the correlation between two vectors or data samples. Intuitively, it measures the similarity of sampled data when ordered by each vector. The association is based on the number of concordant (match the rank order) and discordant (mismatch the rank order) pairs, adjusted by the number of ties in ranks. The Kendall rank correlation coefficient, commonly known as the Kendall τb coefficient can be defined as:

$$KRCC(\vec{x}, \vec{y}) = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}$$

where $n_0 = n(n-1)/2$, n the length of the vector $\vec{x}$ or $\vec{y}$.

$n_1 = \sum_i t_i(t_i - 1)/2$,

$n_2 = \sum_j u_j(u_j - 1)/2$,

$n_c =$ number of concordant pairs,

$n_d =$ number of discordant pairs,

$t_i =$ number of tied values in the ith group of ties for 1st vector,

$u_j =$ number of tied values in the jth group of ties for 2nd vector.

9.4 Summary

Clustering is the most popularly used unsupervised learning technique. It has wide applications in real-life data analysis. We discussed a number of prominent clustering techniques, proximity measures, and how to validate the quality of clusters. New clustering algorithms are still being proposed to handle varying types of data. The techniques meant for moderate-sized data become ineffective and useless for large datasets. The dynamic nature of data also makes cluster analysis more challenging. There is a great need for new versatile clustering techniques that can work with small or large datasets with attributes of different data types, and datasets that may be static or generated dynamically with different generation speeds, including real-time or almost real-time generation.

Lesson 10: Ensemble learning

10.1 Introduction

The outcome of a single learning model or framework may not always be conclusive due to the relative merits and demerits of each framework. Single-model learning can be limited in its ability to achieve high accuracy and robustness in predictions for several reasons:

1. Overfitting: A single model may learn to fit the training data too closely and fail to generalize well to new, unseen data.

2. Underfitting: A model may not have enough complexity to capture the underlying patterns in the data, resulting in poor performance.

3. Biases and errors: A model may have biases and errors that are inherent in its design or training data, which can lead to inaccurate predictions.

Ensemble Learning is a powerful alternative learning technique that combines the outcomes of multiple models to improve the accuracy and robustness of the final outcome. In recent years, ensemble learning has gained increasing popularity due to its ability to achieve state-of-the-art performance on a wide range of tasks, including classification, regression, and clustering.

Ensemble learning takes into account the opinions of a number of learners to obtain the final decision. This ensures that individual drawbacks of learners are overcome and overall performance is improved. Based on whether existing knowledge is used or not, ensemble-learning algorithms are categorized as supervised and unsupervised. Another category of ensemble learning is metaensemble learning, where an ensemble of ensembles is built to further avoid biases of individual ensembles.

10.1.1 What is ensemble learning?

An ensemble-learning technique for a given task consists of a diverse set of learning algorithms, whose findings are combined in some manner to improve the performance of an individual learner. Ensemble learning builds on traditional machine learning. Conventionally, only one model is used to learn the hidden characteristics in a given dataset; the end performance of such a model may often be average. The primary notion in ensemble learning is that a group of decision makers is more reliable than a single decision maker. In the literature, such grouping of decision makers is known by different names such as Blending, Ensemble of Classifiers, Committee of Experts, Perturb and Combine, among many others.

The general idea behind such techniques is that the decision taken by a committee of models is likely to be better than the decision taken by a single model. This is because each model in the committee may have learned some distinct inherent properties in the data that other models

may have missed. Hence, each model may contribute some unique knowledge, impacting the overall performance positively. Therefore, by way of explanation, ensemble learning can be thought of as an opportunity to improve the decision-making process by choosing a diverse group of superior to average learners rather than an individual learner who may not give the best possible performance for a given task. This usually indicates that the generalizing power of a group of models is usually better than the generalizing power of an individual model.

10.1.2 Building an ensemble

An ensemble can be constructed in several ways at different levels. The end goal of an ensemble is to enhance the performance of the decision-making process so that the overall outcome is improved. Below, we discuss some popular approaches to construct an ensemble.

(A) Constructing different training sets: From a given pool of sample training data, several random training subsample sets are formed. A learning algorithm is then trained on each of these training sets to obtain different classifiers. Techniques such as resampling and reweighting are used to obtain sets of random training subsamples.

(a) Drawing samples randomly from a dataset in a repetitive manner.

(b) Reweighting of training examples is performed in creating an ensemble of decision trees called boosting trees.

(B) Random Forests sample features of data examples randomly, in addition to randomly sampling data examples with replacement.

(C) An ensemble can be built from a number of Convolutional Neural Networks (CNNs) such that the CNN models have different numbers of layers, and/or different numbers of filters and/or different hyperparameter values.

10.1.3 Categories of ensemble learning

Depending on how the individual models learn, ensemble-learning techniques can be of five types, as shown in Figure 10.1. Note that all individual learners use the same learning approach.

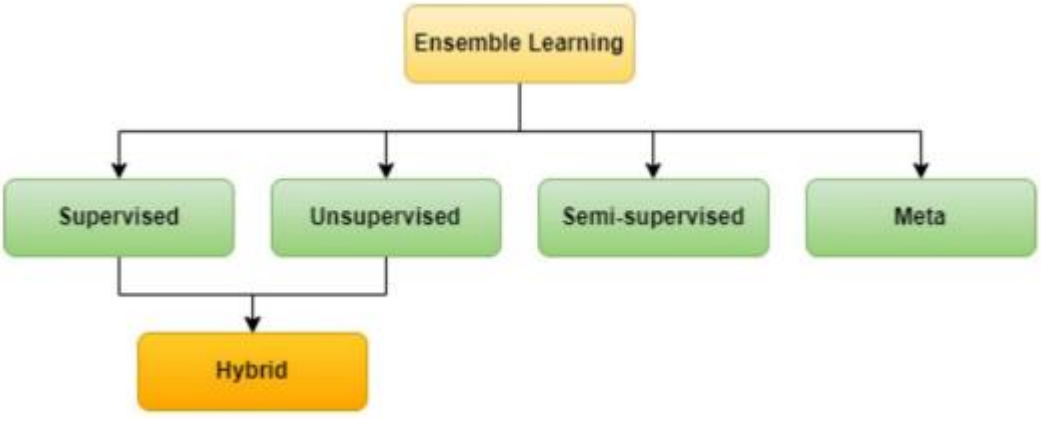


Figure 10.1. Categorization of various ensemble-learning techniques.

10.2 Ensemble-learning framework

A typical ensemble-learning framework performs a multilevel task. The top levels include a selection of base-learning models, followed by consensus decision making based on the outcomes of the base models. The base learners may be any classifier or clustering models depending on the nature of the task, like supervised or unsupervised ensembling. It is important to note that before using any learning technique, the original data may undergo various preprocessing methods such as normalization, missing-value estimation, discretization, feature

selection, or other necessary data transformations. The performance of a learning technique also largely depends on the preprocessing methods used. A typical ensemble-learning framework for classification or clustering can be broken down into several steps:

• **Data Preparation:** The first step is to prepare the data for modeling, which involves cleaning, preprocessing, and transforming the data as needed. This step may include tasks such as feature selection, dimensionality reduction, and normalization.

• **Model Selection:** The next step is to select a set of base models to be used in the ensemble. These models can be diverse in terms of the algorithms used, hyperparameters, or subsets of the data used for training. Common algorithms used in ensemble learning include decision trees, random forests, support-vector machines, and neural networks.

• **Ensemble Generation:** Once the base models are selected, the next step is to generate an ensemble by training each of the base models on either the entire dataset or a different subset of the data, or using different algorithms or hyperparameters. This can be done using techniques such as bagging, boosting, or stacking.

• **Model Combination:** The final step is to combine the predictions of the base models into a single, final prediction. This can be done using techniques such as averaging, voting, or weighted voting. For clustering, a consensus clustering approach can be used to combine the results of individual clustering models into a final clustering.

• **Evaluation:** The ensemble is evaluated on a validation set to determine its performance. Common performance metrics for classification include accuracy, precision, recall, and F1-score, while for clustering, metrics like the silhouette score, Dunn index, or Davies–Bouldin index can be used.

• **Deployment:** Once the ensemble is evaluated and its performance is satisfactory, it can be deployed to make predictions on new, unseen data.

Ensemble learning can be accomplished at different levels, such as attribute (feature) level, or decision level based on the outputs given by the individual machine-learning technique. For example, at the feature level, the feature ranks given by the individual feature-selection algorithms can be combined using combination rules to obtain an aggregated list of the best possible relevant features. The same learning models can be trained on random full-featured subsets of a dataset or versions of the same dataset with sampled subsets of features. The learning models are then tested on previously unseen samples. The final prediction of the models can be combined using different combination rules. The main goal is to increase the generalization capability of an ensemble model such that the final predictions are highly accurate. Figure 10.2 illustrates a generic framework for ensemble learning.

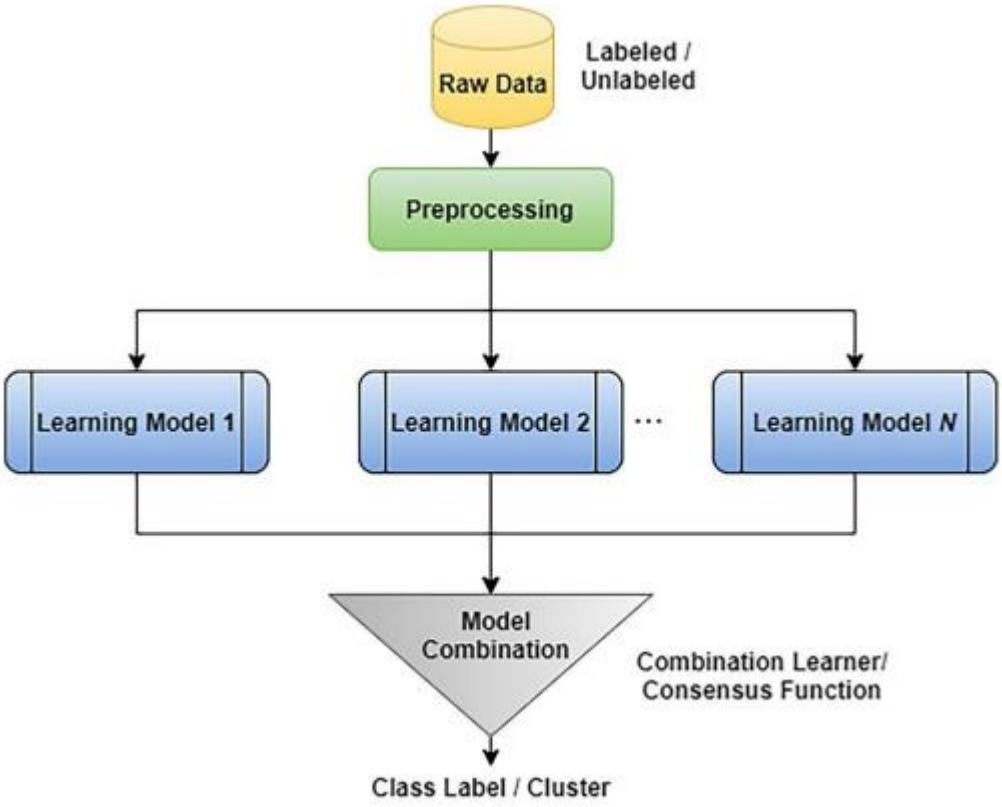


Figure 10.2. Typical ensemble-learning framework for supervised or unsupervised decision making. The learning models are either selected classifiers or clustering methods depending on the task. Combination learners are used during classification, while consensus is used during clustering.

10.2.1 Base learners

Several individual learners participate in forming an ensemble. Such learners are usually from distinct families and are called base learners. In a supervised framework, these base learners take as input a set of labeled samples (training data) and learn significant characteristics from the data to classify the samples with appropriate labels. The output of the base learners is combined to obtain an aggregated prediction, the ensemble's output. It is worth noting that the output of the individual classifier models can be either class labels or continuous outputs, or even class probabilities. The combination function is chosen depending on the output given by the base learners. For example, majority voting and weighted majority voting work when combining class labels, whereas the sum or mean rule work when combining continuous outputs.

In ensemble clustering, base-clustering algorithms are used to partition the data into clusters. These base-clustering algorithms can be different clustering algorithms or the same algorithm with different parameters or random seeds. Each base-clustering algorithm provides a different clustering solution.

The output of the base-clustering algorithms is then combined to form an ensemble clustering. Various methods can be used to combine the base-clustering algorithms, such as consensus clustering or cluster-level ensemble methods.

Similar to base learners in classification, the diversity and complementary nature of the base-clustering algorithms are important for the effectiveness of ensemble clustering. Ensemble clustering can improve clustering performance and robustness by combining the strengths of

different clustering algorithms and providing a more stable clustering solution that is less sensitive to the choice of initial parameters or random seeds.

10.2.2 Combination learners

Combination learning refers to the process of combining multiple models or learners to improve the overall performance of a machine-learning system. The goal of combination learning is to leverage the strengths of multiple models to compensate for their individual weaknesses, leading to a more accurate and robust prediction or classification.

A combination learner is a type of machine-learning algorithm that combines the outputs of multiple base learners, which are usually simpler or weaker models, to form a more powerful or complex model. The combination learner can be used in both supervised and unsupervised learning problems.

In supervised learning, combination learners are typically used for ensemble methods, such as random forests, boosting, or bagging, which combine the predictions of multiple base classifiers to improve classification accuracy or reduce overfitting. The base classifiers can be different types of algorithms, such as decision trees, support-vector machines, or neural networks.

In unsupervised learning, combination learners are often used for clustering or anomaly detection problems, where the goal is to identify hidden patterns or anomalies in the data. Ensemble-clustering methods, such as consensus clustering or cluster-level ensemble methods, can be used to combine the results of multiple base-clustering algorithms to improve the clustering performance and robustness.

A combination function tries to aggregate the outputs of learning models participating in the ensemble process. The outputs to be combined can be in class label form or continuous outputs. Depending on the type of output of the learning model, the combination function must be chosen.

10.2.2.1 Class-label combination

To combine the class-label outputs of learning models, two simple strategies that can be used are Majority voting and Weighted majority voting, as discussed below.

1. Majority voting: According to this technique, the ensemble chooses the class label with the most votes. That is, all the learning models have equal opportunities to choose their respective class labels (these are counted as votes), and the label with the most votes is chosen as the winner by the ensemble. For example, let us say we have ten learning models, and six learning models say that a particular instance, say X1 belongs to class A and four models say X1 belongs to class B. Then, the final output will be class A. This technique can have three different cases:

(a) Case 1: All learning models agree unanimously without any conflicts in predicting a single class label. This is a consensus voting decision.

(b) Case 2: At least one more than half the number of learning models agree to predict a class label. This is a majority voting decision.

(c) Case 3: The class label with the highest number of votes is chosen as the winner.

This is also called a winner-take-all voting decision. Although very popular, the majority has limitations because it may so happen that certain learning models are more suitable for a given task compared to others. In that case, weights must be assigned to the learning models.

2. Weighted majority voting: According to this technique, weights are assigned to the learning models when predicting a class label for a given task. This is because, out of all the participating learning models in the ensemble, some models may be more suitable compared to others. In this case, the suitable members of the ensemble are assigned higher weights than others.

10.2.2.2 Consensus clustering

Consensus clustering is an ensemble-clustering technique that combines multiple clustering solutions to produce a more robust and accurate clustering result. It does this by calculating the level of agreement or similarity between the individual clustering solutions and combining them to maximize the agreement between the solutions. The consensus function is used to measure the agreement between two clustering solutions. Given two clustering solutions, A and B, the consensus function c(A,B) measures the degree of similarity between A and B. A high value of c(A,B)indicates a high degree of similarity between A and B, while a low value indicates dissimilarity. Several consensus functions can be used, including the Jaccard coefficient, adjusted Rand index, and normalized mutual information.

Once the consensus function is defined, the consensus matrix is constructed by calculating the pairwise consensus values between all pairs of clustering solutions. The consensus matrix C is an n×n matrix, where n is the number of clustering solutions. The diagonal elements of C are set to one, as the consensus between a clustering solution and itself is always perfect. The offdiagonal elements of C are the consensus values between pairs of clustering solutions.

The consensus matrix is calculated by applying the consensus function to all pairs of clustering solutions. For example, the consensus value between A and B is 0.33, as only one cluster is the same between A and B ($\{o_3\}$). We can calculate the complete consensus matrix, C, as follows.

$$\begin{pmatrix} 1.00 & 0.33 & 0.00 \\ 0.33 & 1.00 & 0.33 \\ 0.00 & 0.33 & 1.00 \end{pmatrix}$$

Once the consensus matrix is obtained, various clustering algorithms can be applied to obtain a consensus clustering solution. For example, hierarchical clustering can produce a dendrogram that groups together clustering solutions with high consensus values.

Consensus clustering can be helpful in many applications with uncertainty or noise in the data. Combining multiple clustering solutions can produce a more robust and accurate clustering result that is less sensitive to the choice of clustering algorithm or parameter settings.

10.3 Supervised ensemble learning

Supervised ensemble learning uses existing knowledge to classify new instances into respective classes (categories). Let us assume a set of input instances $x_1, x_2, x_3, \ldots, x_n$ such that each $x_i \in$ X. Each x_i is associated with y and is characterized by a feature set F such that $f_1, f_2, f_3, \ldots, f_n \in$ F. X and y are related through a function f in such a way that y = f (X). The x_is are now fed to the learning algorithm with their respective ys and are called training instances. This process yields a prediction model.

Individual classifier models, however, sometimes may not be successful in truly learning the feature space as they may miss some local regions. Hence, they will constantly misclassify instances from such regions. That is why ensemble learning takes the decision of a set of base models to reach a conclusion.

10.3.1 Requirements for base-classifier selection

There are two primary requirements on how to select the base classifiers (learners).

1. Diverse nature: Classifiers based on a similar learning technique will tend to make similar or the same kind of mistakes. In other words, if one of the classifiers overlooks an inherent pattern in the data, there is a high probability that a similar classifier will also overlook that pattern. Such patterns may correspond to a distinguishing or interesting characteristic, in which case the classifiers will have poor performance. This is why, when creating an ensemble, the classifiers are usually chosen from a diverse set of families. This also makes sure that the outputs of individual models are less correlated with each other, and hence, overall performance is improved.

2. Accuracy in the classifier performance: The chosen base learners should be high performers in terms of accuracy individually or at least perform better than a random learner. If there are C classes in a dataset, a random classifier will have an average accuracy. On the other hand, a combination of weak learners, each performing worse than a random learner, may not give the desired quality outcomes.

10.3.2 Ensemble methods

Over the years, several ensemble-construction approaches have been developed. Some prominent ones are described next.

(a) Bagging: From the original dataset, samples are chosen at random with replacement to obtain different versions of the same dataset. This is known as a bootstrap technique and the samples are called bootstrap samples. A learning algorithm is then trained with these different sets of samples in parallel to obtain classifier models that assign class labels to the samples. The combination scheme used is majority voting, that is, if the majority of the base learners predict a sample to be of a particular class, then the corresponding label is assigned to it. Figure 10.3 illustrates an example of Bagging.

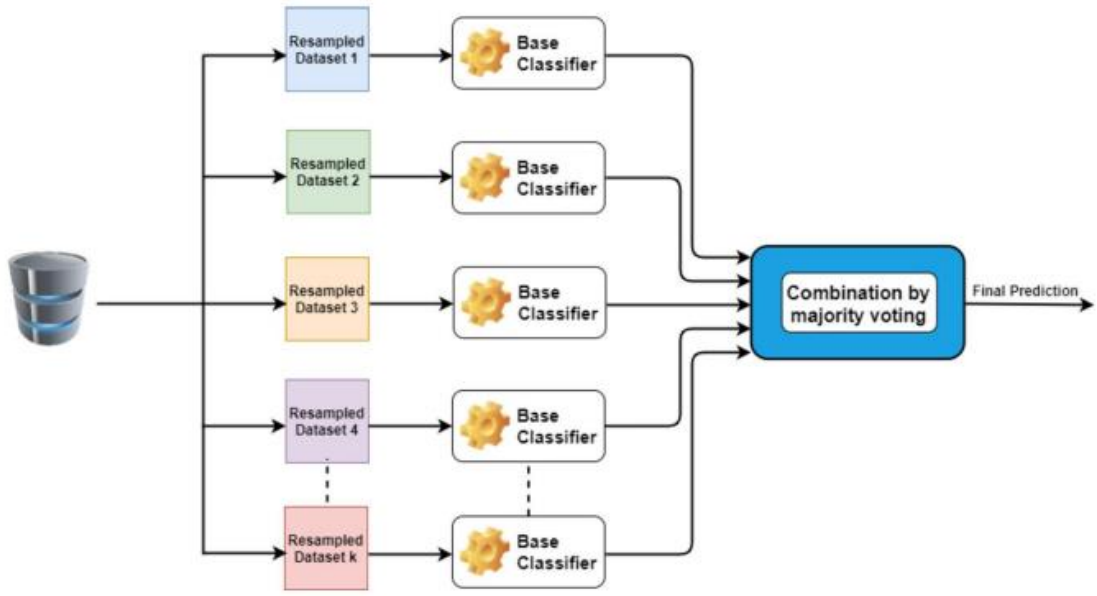


Figure 10.3. Working principle of Bagging approach.

(b) Boosting: Boosting is an ensemble approach that works sequentially. It focuses on the misclassified instances in every iteration. In the first iteration, all the instances are assigned equal weights. In subsequent iterations, the misclassified samples of the previous iteration are given more weight, or are said to be boosted. Such a mechanism works well with weak learners as several weak learners can be combined to solve some difficult tasks. For boosting to work, weak learners must perform better than random learners. Adaboost is a popular boosting model

with which we could use other learning models such as Decision trees or Naive Bayes as a base learner. Figure 10.4 illustrates an example of Boosting.

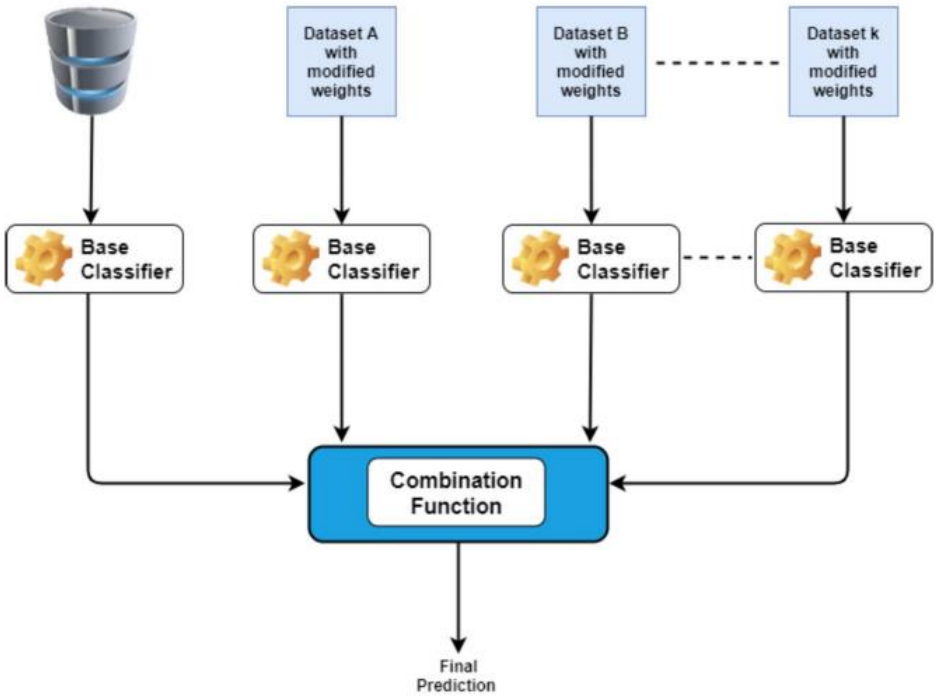


Figure 10.4. Typical Boosting steps

(c) Stacked Generalization: This is an ensemble approach where learning takes place in two levels. The layer-1 base classifiers are trained with a level-0 bootstrapped dataset. Their outputs are used as input by the next-level metalearner. As the name suggests, one layer of the dataset and classifier is stacked over another layer of the dataset and metaclassifier. It is called a metaclassifier because it learns from the behavior of a set of classifiers above it. Figure 10.5 illustrates an example of Stacked Generalization.

10.4 Unsupervised ensemble learning

Unsupervised ensemble learning works by generating a set of clustering groups using clustering algorithms and combining these groups to produce a consensus output. These ensemble methods have better performance in terms of accuracy, robustness, and stability when compared to single clustering algorithms because they can make full use of the information provided by the various clustering algorithms. Figure 10.6 shows an unsupervised ensemble-learning framework.

Different clustering models are combined to create a consensus cluster model that is superior to the individual clustering models. The resultant consensus is less sensitive to outliers and noise and is more stable and robust. There are two primary stages for creating clustering ensembles, generation and integration. In the first stage, generation, the given dataset is used to train different clustering models and their respective clustered groups are obtained. In the second stage, these clustered groups are integrated into a single consensus clustering to obtain the final result.

For generating the clustering ensemble, different mechanisms may be employed, including resampled instances or varying feature subspaces, different base-clustering models, different hyperparameters to the clustering models, or data projected onto different subspaces. The hardest is the integration stage, i.e., integrating the base-clustering models to obtain a consensus. For integration, the most common technique is the voting approach, where for a

given instance, a vote is taken from each base-clustering model, and the instance is finally assigned to the cluster that receives the highest votes.

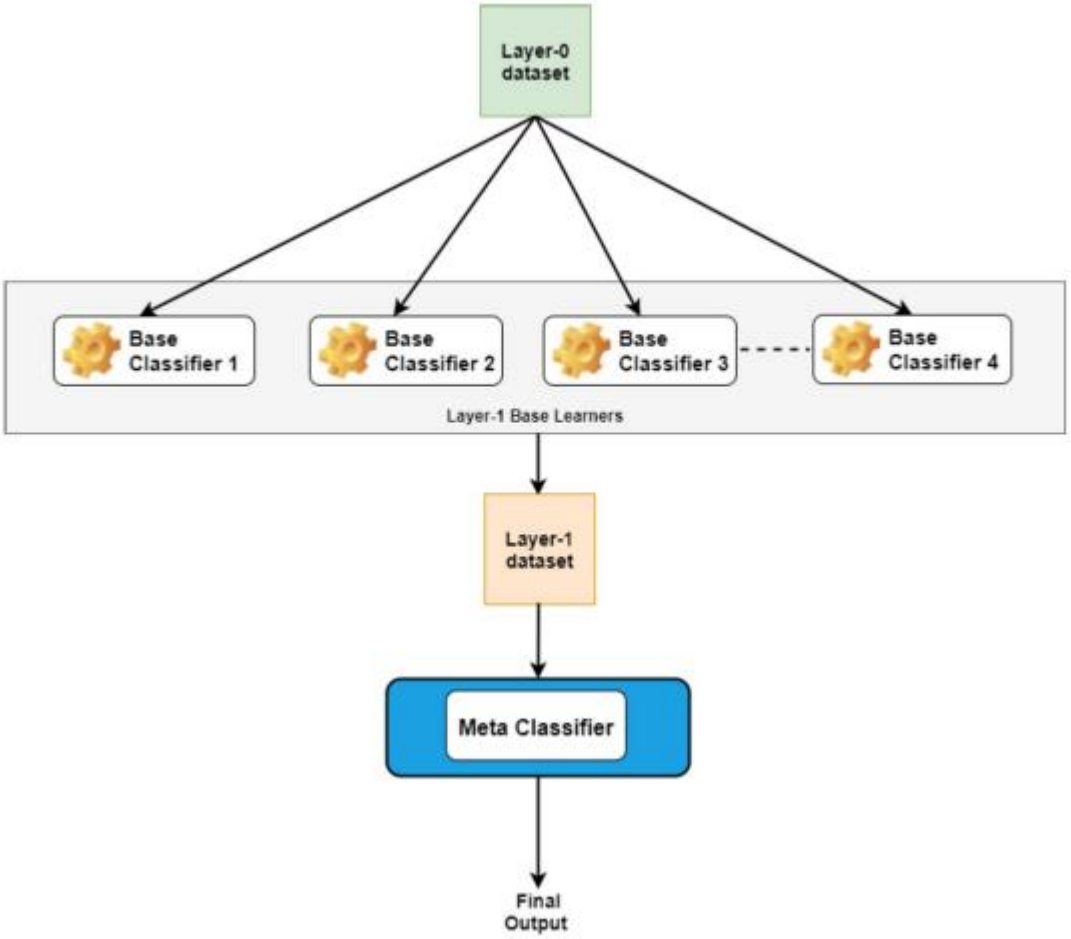


Figure 10.5. Stacking-based approach for ensembling.

10.5 Semisupervised ensemble learning

In recent years, semisupervised ensemble learning has garnered tremendous attention. In semisupervised ensemble learning, the training set is expanded to incorporate previously unlabeled instances. New patterns and informative data can be introduced to the existing distribution. The main issue in many application domains is the unavailability of labeled data. Often only a small quantity of labeled data is available, it being the main motivation for proposing most semisupervised methods in various domains. It is worth noting that some works in the literature deal with only positive and unlabeled instances although the results are not always good. Nevertheless, whenever the amount of labeled data is insufficient, extensive experimental results demonstrate that semisupervised ensemble learning performs better than the supervised ensemble-learning techniques. Figure 10.7 describes a semisupervised ensemble-learning framework. Below, we discuss a few popular methods.

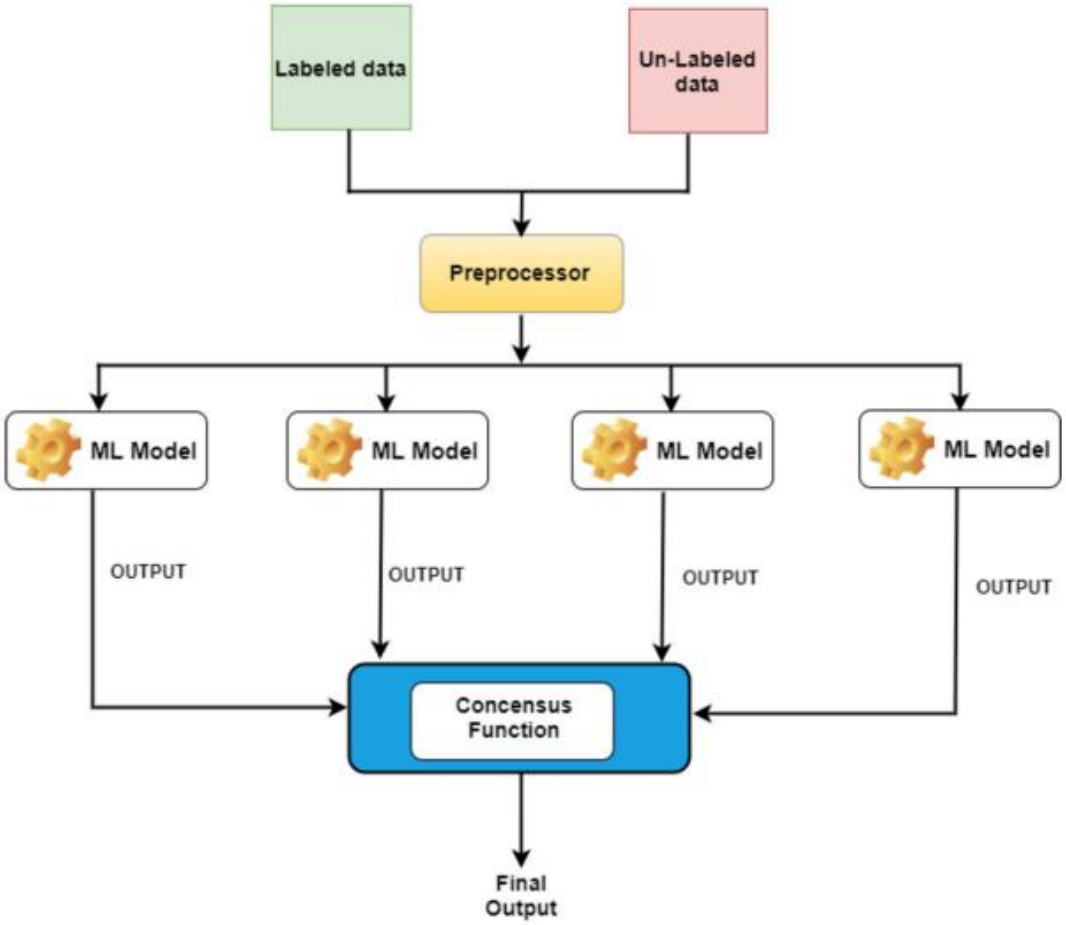


Figure 10.7. Semisupervised Ensemble-Learning Framework.

10.6 Summary

Ensemble learning is a relatively well-studied area in machine learning. Ensembles are flexible in structure and composition. An ensemble can be combined with other approaches in a seamless manner with some effort. It can be used not only with traditional machine-learning methods like supervised and unsupervised learning but also with more advanced concepts like deep learning and transfer learning. An ensemble for supervised learning can be constructed at different levels, for example at the sample level, or feature level or even the output level. Most of the research in the literature is focused on building ensembles at the feature level and at the output level. For unsupervised ensemble learning, two main stages are considered; first, the generation of clustering models and then the integration stage. In the literature, the construction of both stages has been given equal importance. Semisupervised learning, however, is a little different from the other two as labeled data are scarcely available for many tasks. It can be thought of as a more data-centric approach compared to the other two.

Lesson 11: Big Data analysis

11.1 Introduction

The term “Big Data” can be used to refer to data that are too big in size, too fast in generation, too varied in nature, and too complex in context for existing tools to handle and process. Storage, analysis, and visualization of such data are difficult because of size, complex structure, and heterogeneity. With rapidly evolving technologies, data generation sources have also been evolving. Today, data are generated from online transactions, emails, social media, sensors, mobile phones, and many other varied sources. Almost every device we use can connect through the Internet, and produce and record data such as details of device maintenance,

location, and usage including data pertaining to personal health and bill payments. The worldwide explosion of digital data can be surmised from the fact that the total amount of data created till 2003 was 5 exabytes (1018 bytes), which expanded to 2.72 zettabytes (1021 bytes) in 2012, and further increased to 8 zettabytes of data by 2015. In 2022, 2.5 quintillion bytes of data were generated daily, approximately 90% of the total data created in the prior two years. Today's online platforms generate a huge amount of data every minute. For example, 41.6 million messages per minute on WhatsApp, 1.3 million video calls per minute across applications, 404,000 hours of Netflix streaming per minute, 494,000 Instagram and Facebook posts per minute, 2.4 million Google searches per minute and 3 million emails sent per second all contribute to the growing bulk of digital data. Storage of this massive amount of data in databases is a major issue. Open-source database-management systems such as MySQL and Postgres lack scalability. Cloud storage that transfers the problem to someone else may be an option, but storing huge amounts of valuable data at distant service providers is subject to uncertainties of network availability, power outage, or other disruptions. All these data, generated incrementally in a geographically distributed manner, need to be processed fast, which is impossible for traditional data warehouses and their limited processing power.

Big Data analysis is concerned with the process of extracting valuable insights from large and complex datasets. The process involves several steps: **collection**, **processing**, **storage**, and **analysis**. In recent years, the ability to perform analysis of massive amounts of data has emerged as a crucial necessity for businesses and organizations. It allows them to gain new insights into customer behavior, market trends, and operational performance. One key challenge in Big Data analysis is dealing with the sheer volume of data that needs to be processed. Traditional data-analysis methods are insufficient to handle such large datasets, which can contain millions or even billions of records. As a result, new tools and techniques have been developed to process and analyze large amounts of data more efficiently. One such approach is Hadoop, an open-source software framework that provides a distributed file system and tools for processing large datasets. Hadoop allows analysts to break down a large dataset into smaller, more manageable chunks, which can be processed in parallel across many computers. This approach can greatly reduce the time and resources required for data analysis. Another important technique in Big Data analysis is machine learning. This involves using algorithms to automatically identify patterns and insights in large datasets. Machine learning can be used for various tasks, including predictive modeling, anomaly detection, and natural language processing. As Big Data analysis continues to evolve, it is likely that we will see new tools and techniques emerge to help process and analyze data even more efficiently. With the increasing availability of large datasets and the growing demand for insights from businesses and organizations, Big Data analysis is set to play a crucial role in the future of data-driven decision making.

This session discusses Big Data and its analysis. It explores the challenges involved in processing and analyzing Big Data and the tools and techniques developed to address these challenges. Specifically, we delve into the Hadoop and Spark frameworks and how they enable analysts to process and analyze large datasets efficiently. By the end of this session, the students should have a solid understanding of the key concepts and techniques involved in Big Data analysis and how they can be applied to real-world business problems.

11.2 Characteristics of Big Data

Big Data are characterized by five major attributes: variety, velocity, volume, veracity, and value, termed the 5Vs. (See Figure 11.1).

1. Variety refers to the varied sources from where data are collected. The data are generally of three types: structured, semistructured, and unstructured. Structured data are tagged, can be

easily sorted, and stored in a data warehouse. Unstructured data are random and difficult to analyze, and semistructured data do not have any specific fields but are tagged. The type of data must be known as the storage and analysis complexities vary.

2. Velocity refers to the increase in speed at which data are created, processed, stored, and analyzed. It also refers to the speed at which data are generated and moved around among databases. It is ideally necessary to process the bulk of data during the limited time during streaming, to maximize utilization and value. For example, Google processes videos of different sizes uploaded on YouTube, or emails sent in almost real-time.

3. Volume refers to the size of the data to be handled by an organization, which is increasing every day and has reached terabytes and petabytes for large organizations. This is among the first problems faced while handling Big Data. Organizations need to procure ample capacity to store and process huge volumes of data to gain maximum benefit.

4. Veracity refers to the reliability of the source from where the data are accumulated. The quality of the data must be of the utmost importance while simultaneously dealing with the large volume, velocity, and variety of data. The data must be correct and properly tagged. Different data sources are likely to have varied quality, and this must be taken into account while ensuring accuracy.

5. Value refers to utilizing the accumulated data such that the required information is extracted from it and used to the maximum value. Accessing and storing data become useless if they cannot be processed.

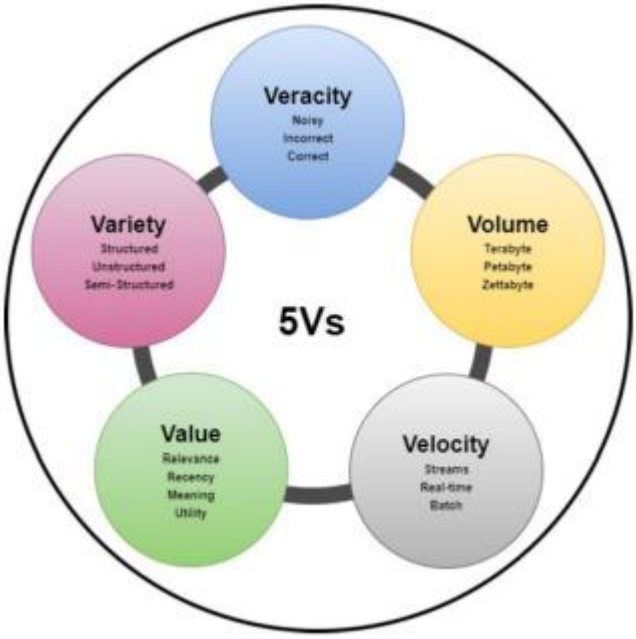


Figure 11.1 The five major challenges, denoted as 5V, that characterize Big Data. Every V is associated with key attributes shown.

11.3 Types of Big Data

As discussed in Lesson 2 and also mentioned earlier in this lesson, datasets that constitute Big Data may be structured, unstructured, and semistructured.

1. Structured data: Data items are properly tagged and placed in fixed fields within a record. All the data values within a certain field have the same set of properties, and thus it is easy to understand and process. Storing the data within fixed records enhances the integrity of the data; enables efficient and effective querying and analysis; and facilitates the accumulation of new data and transmission of data. Structured data can be considered relational data, to be handled

using relational databases and SQL (structured query language). For example, customer data from e-commerce websites are structured data.

2. Unstructured data: Data items do not follow any specific structure or rules and do not conform to the syntax of a definite type of record. The arrangement of the data is unplanned and cannot be handled without a prior preprocessing step. For example, text, image, and video data are unstructured as they might have many components that cannot be categorized into specific groups in a straightforward manner.

3. Semistructured data: Data components are not bound by any rigid schema for storage and handling. The datasets are not organized like relational data but have certain features that help categorize the data. Such data cannot be handled using SQL; thus, NoSQL or a similar data-serialization language may be used to exchange semistructured data across systems. Data-serialization languages like XML, JSON, and YML are used to process semistructured data. For example, the metadata associated with images and videos are semistructured data.

11.4 Big Data analysis problems

Issues related to Big Data can be viewed from three perspectives – data, processing, and management.

1. Data: Constituents of Big Data are characterized by high dimensionality and large sample sizes. Organizations collect these data from both internal and external sources. The volume, speed, quality, sensitivity, structure, and diversity of the data are major concerns. The volume of data is the main concern for storage and processing. Due to rapidly evolving cost-effective storage technologies, organizations can often handle data storage. In addition, outsourcing the storage of nonsensitive data to resourceful cloud-based providers can help store huge volumes of data. The inflow and outflow speeds of different types of data are different. The speed at which a text file is sent is much faster than the speed at which an audio file is sent. In addition to completeness and accuracy, quality must be maintained while working with Big Data. For example, missing values due to unavailability or undetectability may reduce the data quality. The sensitivity of the data also determines the data-handling requirements. Sensitive information, such as people's personal data, and organizations' confidential information requires encryption and secured storage. The structure of the data is required to decide the type of processing requirements. Structured data have a high degree of organization and are suitable for direct machine processing, whereas unstructured data require a preprocessing step before using conventional tools. The huge volume of Big Data can be very diverse, with different data types and varied sources. Data can be text, multimedia, or location data requiring different approaches to storage and processing. The diversity of the data adds enormously to the complexity.

2. Processing: Data processing is a major concern after collecting a large volume of data. The data need to be processed to extract maximal value. Availability of algorithms, scalability, actionable knowledge, and timelines are aspects of processing that need to be addressed. Suitable algorithms for processing the data are required to complete the required tasks with efficient use of space and time. Even with significant progress in technology, algorithms that can handle huge volumes of diverse data with ease are scarce. Scalable processing power is required to handle the ever-expanding amounts of data to meet demands. The knowledge derived from the collected data after processing is actionable knowledge. Different organizations need to extract different information from the same datasets, thus requiring different processing techniques. Timelines refer to the time required for processing the data. Offline and online modes of processing require different processing times. Online processing can be real-time and starts as soon as data acquisition starts, whereas offline processing may

be delayed. Real-time processing of Big Data requires higher computing power than offline mode.

3. Management: Challenges in managing the data abound during both acquisition and processing. Infrastructure, resource optimization, storage management, and database management are aspects of the management of Big Data. Efficient collection of the data, adequate storage capacities and computing power, and availability of high-speed Internet are all aspects of infrastructure challenges in handling Big Data. Optimizing resources, such as computing and communication equipment, manpower, time, and financial investments are also matters of concern. The large volume of data imposes huge demands on storage. As the volume of data increases each passing year, storage technologies need to progress at the same pace to accommodate the growing data. Storage management requires fast internal memories, with necessary storage capacity, and efficient backup options such as hard drives. Advanced database technologies are required for efficient processing and management of the data. Traditional databases that run on specialized single-rack high-performance hardware cannot meet the growing demands of Big Data. Data storage and management require both SQL and NoSQL database systems to handle the different data types.

11.5 Big Data analytics techniques

Big Data analytics refers to the process of extracting insights and knowledge from large and complex datasets. Approaches for Big Data analytics are constantly evolving, although there are a variety of existing techniques that can be used for analysis and interpretation. Some of the most common techniques used in Big Data analytics include the following:

• **Machine Learning:** Machine learning refers to the use of algorithms and statistical models to enable computer systems to learn from data without explicit programming. Machine learning is a powerful technique that can be used to extract insights from Big Data, such as identifying patterns and trends, making predictions, and detecting anomalies.

• **Data Mining:** Data mining is the process of analyzing large datasets to identify patterns and relationships. Data-mining techniques can be used to uncover hidden insights and identify trends. This can be useful in applications such as market-basket analysis, where patterns in customer purchasing behavior can be used to improve product recommendations and increase sales.

• **Predictive Modeling:** Predictive modeling involves using statistical techniques to make predictions about future events based on historical data. Predictive modeling can be used in a variety of applications, such as fraud detection, risk management, and customer segmentation.

• **Visualization:** Data visualization refers to the creation of graphical representations of data. Visualization techniques can be used to simplify complex data and enable analysts to quickly identify patterns and trends. This can be useful in applications such as dashboard reporting, where real-time data is displayed in an easy-to-understand format.

11.6 Big Data analytics platforms

The analysis of Big Data is supported by several platforms used to handle the growing demand for processing. Peer-to-peer networks and Apache Hadoop are two of the most well-known Big Data analytics platforms.

1. Peer-to-peer networks: A peer-to-peer network connects millions of machines. It is a decentralized and distributed network architecture where the nodes, called peers, provide and consume resources. The scheme used in the communication and exchange of data in a peer-to-peer network is called Message Passing Interface (MPI). The number of nodes in the network

can be scaled to millions. Broadcasting messages in a peerto-peer network is inexpensive, following the spanning-tree method to send messages, choosing any arbitrary node as the root. A major drawback is that the aggregation of data is expensive. MPI is a widely accepted scheme. A major feature of MPI is state preserving, whereby a process can live as long as the system runs and need not be read again and again. MPI is suitable for iterative processing and uses a hierarchical master– slave model. The slave machines can have dynamic resource allocation when there is a large volume of data to process. MPI has methods to send and receive messages, broadcast messages over all nodes, and methods to allow processes to synchronize. Although MPI is good for developing algorithms for Big Data analytics, it has no mechanism for fault tolerance. Users must implement their own fault-tolerance mechanisms.

2. Apache Hadoop: Hadoop is an open-source framework for storing and processing huge volumes of data. Hadoop can scale up to thousands of nodes and is highly fault tolerant. It comprises two main components – Distributed File System (HDFS) and Hadoop YARN. HDFS is a distributed file system used to store data across clusters of commodity hardware and also provides high availability and high fault tolerance. Hadoop YARN is used for resource management and to schedule jobs across the cluster. The programming model used in Hadoop is allied with MapReduce. The process followed by MapReduce includes dividing the task into two parts, called mappers and reducers. The job of mappers is to read the data from the HDFS file system and process it to generate intermediate results for the reducers. Reducers aggregate the intermediate results and generate the final result, which is written back to the HDFS. Several mappers and reducers are run across different nodes in the cluster to perform a job, helping perform parallel data processing. MapReduce wrappers are available to help develop source code to process the data. Wrappers such as Apache Pig and Hive provide an environment to programmers for developing code without handling complex MapReduce coding. DryadLINQ is a programming environment that provides flexibility for coding using C#. A major drawback of MapReduce is that it cannot handle iterative algorithms. Mappers must read the same data repeatedly, and each iteration must be written to the disk to pass them on to the next iteration. A new mapper and a new reducer are initialized for every new iteration. Thus the disk has to be accessed frequently, making it a bottleneck for performance. A scalable and distributed database called HBase supports structured data for large database tables. It is a nonrelational database (NoSQL) and runs on HDFS. It allows transactional functions like updates, inserts, and deletions. It also allows fast real-time read–write access and has fault-tolerant storage capability. Hadoop also includes a scripting language called Apache PIG that enables users to write MapReduce transformations like summarizing, joining, and sorting. PIG also enables parallel processing to handle large datasets. Another important module of Hadoop is Cassandra, which supports an open-source distributed management system. It is decentralized, highly scalable, and has the capability of handling large volumes of data across multiple servers.

3. Spark: Spark is a data-analysis paradigm that has been developed as an alternative to Hadoop. Spark has the ability to perform in-memory computations and can overcome disk I/O limitations that were observed in Hadoop. Data can be cached in memory, thus eliminating repetitive disk access for iterative tasks. Spark can run on Hadoop YARN and can read data from HDFS. It supports Java, Scala, and Python and has been tested to be up to 100x faster than Hadoop MapReduce.

4. High-Performance Computing (HPC) clusters: An HPC cluster is a supercomputer with thousands of cores that offers a parallel computing platform. An HPC cluster provides powerful hardware that is optimized for speed and throughput. The high-end hardware is also fault tolerant, making hardware failures extremely rare. The communication scheme used for HPC is MPI, discussed earlier.

5. Multicore CPU: A multicore CPU has dozens of processing cores that share memory, and have only one disk. The CPUs have internal parallelism and there is a single motherboard that houses multiple CPUs, improving parallel computing. Parallelism is achieved by multithreading. Each task is broken down into threads and each thread is executed in parallel on different CPU cores.

6. Graphics Processing Unit (GPU): A GPU is a specialized hardware designed primarily for graphical operations such as image and video editing and accelerating graphics related processing. A GPU has a large number of processing cores, much higher than a multicore CPU. It has a parallel architecture and a high throughput memory. In the past few years, GPU performance has increased significantly compared to CPUs. The CUDA framework has helped developers with GPU programming without the need to access the hardware. Machine-learning algorithms implemented on GPU using CUDA run much faster as compared to multicore CPU. GPUs are extremely popular due to their high speed and parallel processing.

11.7 Big Data analytics architecture

The five major issues to be considered in developing any analytics architecture for Big Data are (a) scalability, (b) fault tolerance, (c) iterative computing, (d) complex data dependencies, and (e) heterogeneity. Several novel efforts have been made in developing appropriate architectures to support cost-effective Big Data analytics. Three popular architectures are described next.

11.7.1 MapReduce architecture

The MapReduce architecture was developed by Google to achieve data parallelism. It exploits multiple compute nodes to perform a given task on different data subsets in parallel. Following this data-parallel execution approach, several successful efforts have been made commercially as well as noncommercially. Apache Hadoop is one such commercially successful open-source MapReduce implementation. The MapReduce approach includes two types of nodes: master and compute nodes. The master node is responsible for the tasks of configuration and control during the process of execution of a task. Figure 11.2 depicts the schematic view of the MapReduce architecture.

Compute nodes are involved in executing a task iteratively following two major steps of operations: (i) map and (ii) reduce. Each of these two steps includes three distinct states: (a) accept input, (b) perform computation, and (c) generate output. Between two consecutive steps, synchronization is necessary. The local memory is cleared during synchronization and writes onto the global memory are performed. A basic difference between the master and the compute nodes is that the master is empowered to (i) read–write onto the global memory and (ii) communicate with the nodes without any time restriction, whereas the compute nodes can use the global memory for read–write during the synchronization only. These two distinct abilities of the two types of nodes are indicated in the figure with different types of directed lines (thin and thick). The tasks of data distribution, partial results generation, and storage are completed with the compute nodes during the map operations.

During reduce operations, the intermediate results are combined to generate the final results that are written onto global memory. In case there is a necessity for further processing of the partial results, the steps are iterated. This architecture gives an excellent performance especially when (i) data are voluminous and (ii) the scope for parallelism is high. However, if the computational dependencies among the data are high, the architecture fails to perform well. Further, the higher iterative computational requirements often lead to high I/O overhead, leading to poor performance. Efforts have been made to address these issues of the MapReduce architecture. However, a fault-tolerant architecture that can handle such iterative computational

requirements for voluminous heterogeneous data with complex dependencies with cost-effective I/O operations is still a need of the hour.

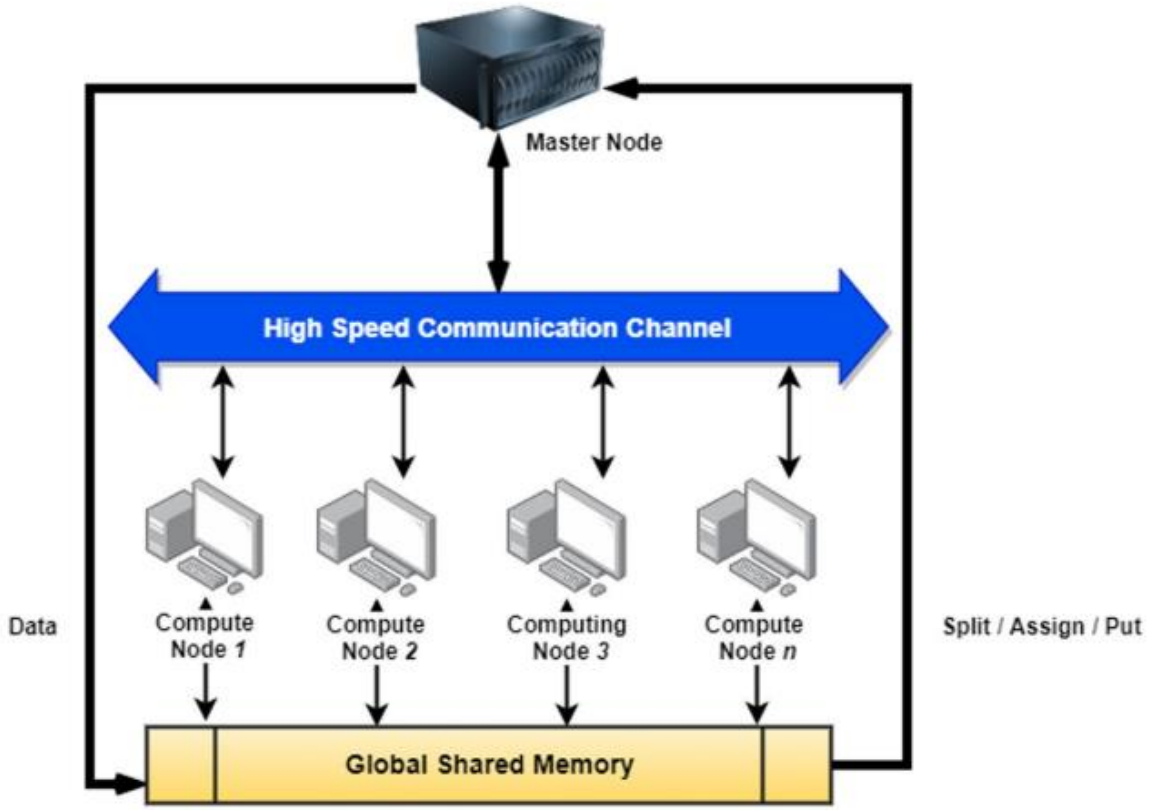


Figure 11.2. A MapReduce architecture with n computing nodes and a master node that is connected through a fast communication network and has access to a shared global memory.

11.7.2 Fault-tolerant graph architecture

To address the issue of handling iterative processing with complex data dependencies using fault-tolerant approaches, graph-based architectures are often suitable. Such a fault tolerant architecture associated with a global shared memory performs the computation in a synchronous manner like MapReduce. It divides the computation among the compute nodes in a heterogeneous manner by assigning dissimilar (or particular) tasks to the nodes. A graph-based architecture of this type includes two components, (a) a directed graph with a set of compute nodes and (b) a distributed global shared memory. Figure 11.3 depicts a schematic view of the architecture. The dependencies among the compute nodes are indicated by the directed dashed lines. A fast communication medium is used to improve communication with the nodes to achieve an improved overall performance.

GraphLab not only offers fault tolerance while supporting iterative computation using unreliable networks such as the Internet, but also can handle data with high dependency. Each node initially reads the instances from the shared repository and executes a compute task using its own and neighboring nodes' data. Results are gathered and merged, and written back onto the shared global repository for use in the subsequent cycle of execution. If a node fails during a cycle of operation, the corresponding compute task is repeated and as a consequence, one cycle is lost by the dependent node(s). Such loss of a cycle definitely causes degradation in efficiency, although, it guarantees fault tolerance. This architecture has a provision for replacing a node if it fails permanently. Additionally, it has other advantages such as (i) it can execute a problem with iterations and complex dependencies and (ii) it performs scalable distributed computation. However, two major disadvantages of GraphLab are: (a) high disk I/O

overhead and (b) nonoptimized performance. Several variants of GraphLab have also been introduced, including Pregel, Giraph, GraphX, and Hema.

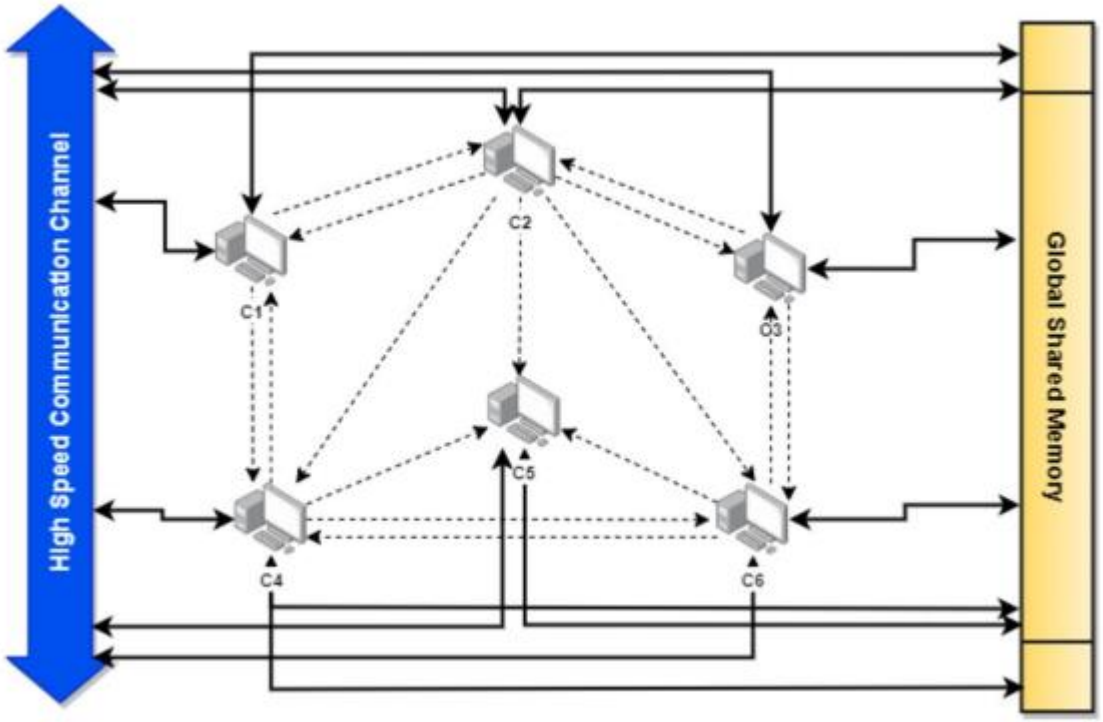


Figure 11.3 Fault-Tolerant Graph Architecture.

11.7.3 Streaming-graph architecture

The streaming-graph architecture was introduced to address the issue of high I/O overhead of previous architectures while handling streaming data. Although a good number of solutions have been introduced and consequently several packages such as Spark Streaming have been introduced, most of these attempts handle stream data in batches. These solutions convert stream data internally to batches prior to downstream processing. However, stream data require in-memory processing for high bandwidth. This architecture (see Figure 11.4) attempts to handle such stream data with the high processing speed, reliability, and high bandwidth.

The major characteristics of this architecture are as follows:

1. Unlike the previous architectures, it does not use a global shared memory.

2. It executes the operations in an asynchronous manner, allowing synchronous data flow only during merge operations.

3. It facilitates in-memory data processing at any scale rather than storing the data on a disk. It saves costs significantly as well as ensures high throughput.

4. This architecture also does not support fault tolerance, a major limitation. The failure of any node leads to restarting the whole process from the beginning, another serious drawback of this architecture.

11.8 Tools and systems for Big Data analytics

One major application area of Big Data analytics is Bioinformatics. Many efforts in this evolving area of research support the fast construction of coexpression and regulatory networks, salient module identification, detection of complexes from growing protein– protein

interaction data, fast analysis of massive DNA, RNA, and protein-sequence datasets, and fast querying on incremental and heterogeneous disease networks. Extensive research in Bioinformatics has given birth to a number of tools and systems to support downstream analysis of gene-expression data. In addition to the development of tools and systems, there are efforts to provide adequate cloud-based bioinformatics platforms to support integrative analysis. Some such example platforms are: Galaxy and Cloud-BLAST.

11.8.1 Bioinformatics tools

Microarrays are classical gene-expression data types commonly used for hidden knowledge extraction using statistical and machine-learning techniques. A large number of software tools have been introduced by computational biologists to perform analyses of microarray data. However, with the exponential growth of such data both in terms of volume and dimensionality, the time required to process disease queries on heterogeneous data to find interesting patterns (relevant complexes, modules, or biomarker genes) is extremely high.

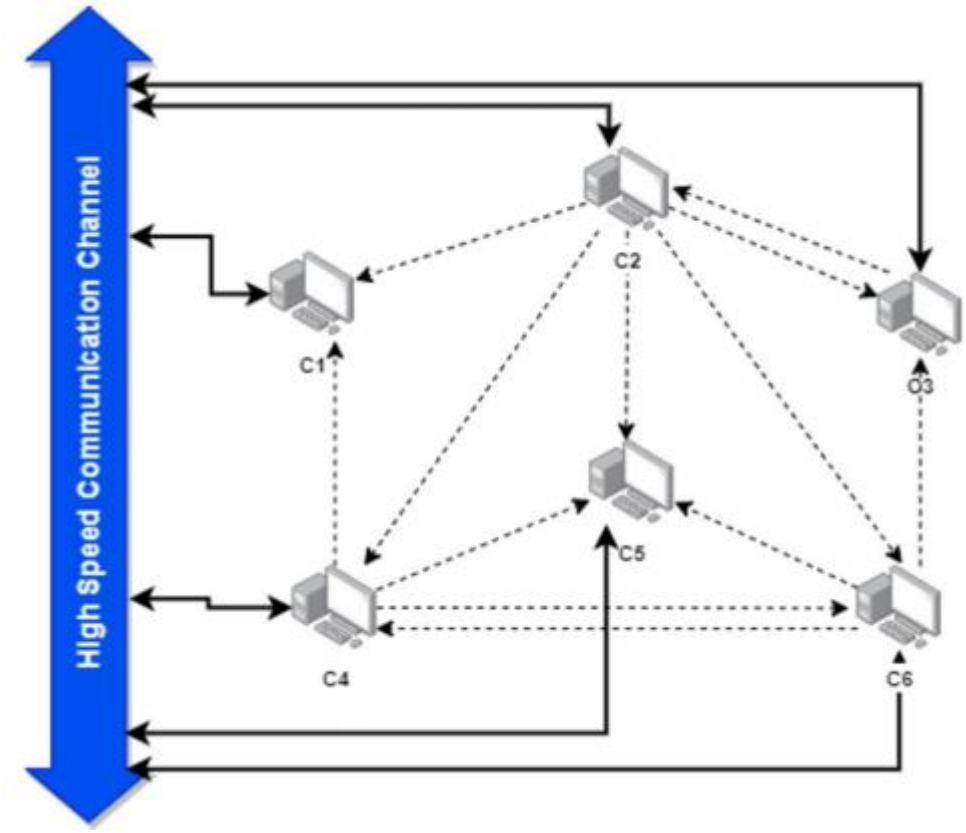


Figure 11.4 Streaming-graph architecture without any global memory.

One commonly used tool for such analysis is Beeline, which can handle Big Data by exploiting parallel computations and adaptive filtering (to minimize data size). Another tool named caCORRECT performs quality assurance by removing artifactual noises from high-throughput microarray data. This tool helps improve the integrity and quality of microarray gene-expression and reproduced data for validation with a universal quality score. OmniBiomarker is a popular web-based tool that helps find differentially expressed genes towards the identification of biomarkers from high-throughput gene-expression data using knowledge-driven algorithms. This tool is capable of identifying reproducible biomarkers with high stability.

The construction of a gene regulatory network using large gene-expression datasets (microarray or RNAseq) and subsequent analysis for the identification of crucial genes for

critical diseases is a task of high computational requirement. A tool that enables network-based analysis using voluminous gene-expression datasets helpful to computational biologists and bioinformaticians. FastGCN is one such tool that facilitates large-scale gene-network analysis using GPU-based distributed computing. Several other tools also support complex network analysis. UGET (UCLA Gene Expression Tool) is useful in extracting interesting disease associations by performing coexpression analysis. A popular tool that facilitates large-scale weighted coexpression network analysis is WGCNA. It works in a distributed computing environment. NETRA is a comprehensive single-stop web platform that can perform inference, visualization, network analysis, and benchmarking of regulatory networks.

Finding interesting complexes or groups from large-scale PPI (protein–protein interactions) networks is a difficult computational task. In a traditional uniprocessor computing environment, the successful execution of such a data-intensive (involving millions of interactions) task often requires weeks. However, to perform such computational tasks quickly, user-friendly tools have been developed. Such tools extract interesting complexes, isolated as well as overlapped from large PPI networks. ClusterONE, MCODE, and NeMo, are some popular tools that can operate in a standalone mode as well as Cytoscape Plugins. On the other hand, PathBLAST is a useful tool to align PPI networks efficiently.

Tools have been developed to perform analytics on large-scale sequence data using the Hadoop MapReduce platform. BioPig and SeqPig are two such scalable tools for sequence analysis that can be ported directly to Hadoop infrastructures. Crossbow is another effective tool that in turn uses an extremely fast and memory-efficient short-read aligner called Bowtie. SoapSNP is a scalable genotyper, which supports sequence analysis on large-scale whole genome-sequence datasets using cloud or Hadoop cluster platforms. There are several other scalable cloud-based tools such as Stormbow, CloVR, and Rainbow.

Pathway analysis is essential for validation of bioinformatics research findings. A faster and more accurate pathway analysis tool may be able to help identify pathway(s) for an identified gene during validation. Available pathway analysis tools include, GO-Elite to provide description of a particular gene or metabolite, PathVisio to support pathway analysis and visualization, directPA to identify pathways by performing analysis over high-dimensional spaces, Pathway Processor to analyze large expression datasets regarding metabolic pathways, Pathway-PDT to perform analysis using raw genotypes in general nuclear families, and Pathview to support integration of data based on pathway. A common disadvantage of these tools is that none can perform in a distributed computing environment to provide scalability.

11.8.2 Computer-vision tools

Computer-vision tools are developed to perform various computer-vision tasks like image processing, video analysis, object detection, etc. These tools are also included with various machine-learning and deep-learning models that can be applied on the computer vision data. OpenCV is a library for real-time computer-vision tasks. Similarly, Pytorch and GluonCV are two libraries that are built on top of two different frameworks that support computer-vision tasks. NVIDIA CUDA-X is a GPU-accelerated library to perform computer-vision tasks with high performance. Similarly, Insight Toolkit (ITK) is another crossplatform open-source library that includes a set of functions to perform image-processing tasks.

11.8.3 Natural language processing tools

Natural Language Processing tools analyze and process large amounts of natural language data. These tools are used for NLP tasks like POS tagging and NER (Named Entity Recognition). Spacy is one such open-source toolkit. Fairseq is an open-source sequence modeling toolkit used for translation, summarization, language modeling, and other text generation tasks.

Gensim is an open-source Python library for topic modeling, document indexing, and similarity retrieval with large corpora. Flair is an open-source library built on the Pytorch framework, and includes state-of-the-art natural language processing (NLP) models. Another library is HuggingFace Transformer, which is backed by JAX, Pytorch, and Tensorflow. It contains various transformer models for NLP tasks. CoreNLP is an open-source NLP tool in Java. A web-based tool Voyant is used for textual analysis.

11.8.4 Network-security tools

Network-Security tools are used to analyze large amount of data collected from different sources. NuPIC and Loom are two network-security tools used for anomaly detection. XPack is a machine-learning-based security tool that provides monitoring, alerting, reporting of security incidents. Splunk handles and analyzes large amounts of data from real-time sources to detect and prevent threats. QRadar and CISCO's Stealthwatch are two data-driven malware analysis tools that continuously monitor network traffic and generate real-time alerts in case of any anomaly.

As seen in the above discussions, there are already a number of Big Data analytics tools to support specific problems in real-life domains. Although some of these tools are good enough in extracting relevant, interesting, and nontrivial knowledge from massive amounts of specialized data, we believe that there is a lack of generic tools that can help comprehensive analytical research over a wide range of problems.

11.9 Challenges in handling Big Data

Although researchers have been successful in developing platforms to handle Big Data in many real-life application domains, there are several research challenges that need to be addressed.

1. I/O bottleneck: Rapid growth of mobile-network connection speeds has made it difficult to handle data well in real time. The annual growth rate reported for the period (2014–2019) reveals that there was a significant increase from 1683 kbps to 4 Mbps during this period. Presently, 4G speeds are up to 1000 Mbps, much higher than that reported in 2014. 5G has been introduced as well. To handle such high-velocity data in real time, the major bottleneck happens to be the speed of I/O operations. Slower I/O operations limit the desired computational throughput.

2. High-speed network accelerator: With continued advances in networking technology, network accelerators have become extremely fast, contributing to the growth of generated data. Handling of such growing voluminous data is challenging, and it lags behind data growth.

3. Heterogeneous semistructured data: The nature of exploding data is mostly heterogeneous and semi- or unstructured. Traditional approaches such as data warehousing are often inadequate in handling such highly varied data, growing with high velocity. As such approaches attempt to organize the data in terms of a set of defined schemas, it is often impossible in the case of unstructured heterogeneous data. Thus, development of methods that enable storage and update of unstructured heterogeneous data in real time remains a challenge.

4. Increasing use of hand-held devices: The increasing use of hand-held and the IoT devices and sensors has been contributing significantly in the growth of Big Data. Such data are voluminous not only in the number of instances but also in dimensionality. It is necessary to develop adequate methods and techniques to handle such growing data in real time.

11.10 Summary

In this session, we introduced the topic of Big Data along with the characteristics, types, and problems that large amounts of data introduce. We also discussed several analytics techniques

and three major architectures for handling Big Data, and their pros and cons from the implementation perspective. Further, we presented a number of tools that have evolved in the recent past to support Big Data analytics. We summarize the chapter contents as follows:

• In recent times, there has been an exponential growth of data. Advancing technological developments are generating 2.5 quintillion bytes of data every day, which is almost 90% of the data generated in the entire prior two years.

• The five major characteristics of Big Data are: volume, velocity, veracity, variety, and value.

• Big Data can be structured, unstructured, or semistructured. To analyze such data, three major perspectives are necessary: data, processing, and management.

• Several Big Data analytics platforms have been introduced. The most commonly used platforms are: (i) peer-to-peer networks and (ii) Apache Hadoop.

• The three major architectures necessary to handle Big Data are: (i) MapReduce, (ii) Graph-based fault tolerance, (iii) streaming graph. These architectures have been developed to satisfy five major requirements, (i) scalability, (ii) fault tolerance, (iii) iterative computation, (iv) complex data dependencies, and (v) data heterogeneity.

• In the past decade, a large number of tools have been developed to facilitate Big Data analytics. These tools are designed to support analysis and/or visualization of massive amounts of data with the complex need for extraction of hidden knowledge in a user-friendly and cost-effective manner.

Lesson 12: Data Science in practice

Data Science is being adopted to solve an exponentially growing number of real-life problems in diverse domains. The use of Data Science is expanding across disciplines where bulk data are available and there is a need to extract hidden or unknown facts. For smart decision making and intelligent support-system development, machine learning, and more recently deep learning have become indispensable due to the ability for accurate prediction.

12.1 Need of Data Science in the real world

In today's technology-driven world, Data Science enables smart problem solving, sometimes it even becomes a life saver. Government agencies and business organizations can use Data Science to make people's lives comfortable. Properly applied Data Science may help resolve many problems within an organization, in turn making life easier and more productive for individuals. Data Science helps people make the right decisions in advance and build effective strategies. For example, to travel to an unknown destination, one can depend on the Google Map application that guides in real time which roadways to follow. You can check in advance the weather conditions at the destination for the next five hours or days using a weather-prediction system.

With the assistance of Data Science-enhanced software technologies, progress in healthcare has accelerated, leveraging simultaneous advancements in actual medical science. It is now possible to investigate the root causes of many terminal diseases quickly, and hypothesize remedies for the same. During COVID-19, the world handled the pandemic rapidly, leveraging Data Science. Starting from characterizing the SARS-CoV2 virus, its origin, evolution, spreading pattern, and drug and vaccine development, Data Science played a major role in effective and rapid solution development. With the evolution of superfast communication and dissemination technologies, the world has become more dependent on network science and

engineering. Ubiquitous technologies such as IoT, sensor, and wireless networks have positively impacted on Data Science. Alongside this, networks and networked computers also have become more vulnerable to security threats. Intruders have also become smart and often adopt the attack patterns to fool preinstalled detectors. Such zero-day attacks are a growing threat. Effective machine learning is important to address such challenges in a timely manner. Recently, Google has become successful in attracting more users by developing easy-to-use techniques. For example, Google facilitates voice-over command through natural language processing (NLP). Amazon's Alexa is the popular name for a smart and artificially intelligent software tool where a machine is interacting with users and assisting in human-like interactions. Speech synthesis and NLP work together to make flexible and natural interactions a reality where users can avoid rigid command structures to work with a machine by typing on the keyboard, or clicking mouse buttons, or by speaking.

Data Science and associated analytics can play a critical role in achieving sustainable development goals (SDGs) by providing insights into complex systems, identifying trends and patterns, and developing models to predict future outcomes. Data Science and analytics provide powerful tools that can support sustainable development by providing insights, enabling evidence-based decision making, identifying inequalities, predicting the impacts of climate change, and tracking financial flows. By using Data Science to inform policy and decision making, we can achieve SDGs and create a more sustainable future for all.

1. Data Science can help monitor and measure progress towards SDGs. By collecting, analyzing, and interpreting data from various sources, we can assess progress towards the SDGs and identify areas that require attention.

2. Data Science can enable the tracking of financial flows to support sustainable development. By using data analytics, we can monitor the flow of resources and identify areas where investments are needed to achieve SDGs. By using data-driven approaches, we can identify the most effective policies and interventions for achieving SDGs and evaluate their impact over time.

3. Data Science can help identify and address inequality and exclusion. By analyzing data on poverty, health, education, and other social indicators, we can identify vulnerable groups and develop targeted interventions to reduce inequalities.

4. Data Science can aid in understanding complex environmental systems and predicting the impacts of climate change. By analyzing environmental data, we can develop models to predict future outcomes and develop strategies to mitigate the effects of climate change.

5. Data Science can enable the tracking of financial flows to support sustainable development. By using data analytics, we can monitor the flow of resources and identify areas where investments are needed to achieve SDGs.

12.2 Hands-on Data Science with Python

Demonstrating the various steps involved in Data Science using Python1 is helpful for several reasons. First, Python is a popular and widely used programming language in the Data Science community, and it offers a rich set of libraries and tools that are specifically designed for data analysis and manipulation. Therefore, showcasing Data Science techniques using Python can help learners and practitioners become more familiar with these powerful tools and improve their data-analysis skills. Python is easy to learn and understand, even for individuals without a strong background in programming. This accessibility makes it an ideal language for demonstrating Data Science steps to a broad audience, including students, researchers, and industry professionals. Furthermore, Python offers a wide range of visualization and data-presentation tools, which can be used to create meaningful and impactful representations of

complex datasets. By demonstrating Data Science techniques using Python, learners and practitioners can develop skills in data visualization and communication, which are essential for effectively disseminating insights to stakeholders and decision makers.

Demonstrating Data Science techniques using Python can help learners and practitioners stay up-to-date with the latest advancements in the field. Python is a dynamic language that is constantly evolving, and new libraries and tools are regularly developed to improve data-analysis capabilities. By showcasing Data Science steps using Python, learners and practitioners can stay informed about the latest developments and trends in the field and retool their skills and knowledge accordingly.

In this section, we demonstrate a step-by-step guide through a small end-to-end Data Science pipeline for a sample project using Python. We assume that the reader has a basic idea of coding in a Python environment. First, it is necessary to check the Python version that is installed in our system.

```
[1]: !python -version
```

Python 3.9.12

13.2.1 Necessary Python libraries

We use a set of libraries (already installed) for our Data Science project. First, we import these libraries with their required functions for our downstream activity.

```
[2]: import warnings   # suppresses the library warnings
     warnings.filterwarnings("ignore")

     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     from sklearn.preprocessing import LabelEncoder
     from sklearn.feature_selection import SelectKBest
     from sklearn.feature_selection import chi2
     from sklearn.model_selection import train_test_split
     from sklearn.model_selection import cross_val_score, StratifiedKFold
```

```
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.metrics import accuracy_score, confusion_matrix, auc, roc_curve
from sklearn.cluster import KMeans, AgglomerativeClustering
import scipy.cluster.hierarchy as shc

import tensorflow as tf
import keras
from scikeras.wrappers import KerasClassifier
```

12.2.2 Loading the dataset

We load the dataset using the Pandas dataframe. The dataset of interest can be downloaded from the UCI Machine Learning Repository, which contains information on 303 patients with

55 features. Out of 303 instances, the dataset contains the status of 216 patients with coronary artery disease (CAD) and 88 patients with normal status. We further use the Pandas library functions to explore the dataset and understand it indepth.

[3]:
```
data = pd.read_excel(open('Z-Alizadeh sani dataset.xlsx', 'rb'),
↪sheet_name='Sheet 1 - Table 1')
```

12.2.3 A quick look at the dataset

[4]:
```
data.head() # displays the first 5 rows of the dataset
```

[4]:

	Age	Weight	Length	Sex	BMI	DM	HTN	Current Smoker	EX-Smoker
0	53	90	175	Male	29.387755	0	1	1	0
1	67	70	157	Fmale	28.398718	0	1	0	0
2	54	54	164	Male	20.077335	0	0	1	0
3	66	67	158	Fmale	26.838648	0	1	0	0
4	50	87	153	Fmale	37.165193	0	1	0	0

	FH	...	K	Na	WBC	Lymph	Neut	PLT	EF-TTE	Region RWMA	VHD	\
0	0	...	4.7	141	5700	39	52	261	50	0	N	
1	0	...	4.7	156	7700	38	55	165	40	4	N	
2	0	...	4.7	139	7400	38	60	230	40	2	mild	
3	0	...	4.4	142	13000	18	72	742	55	0	Severe	
4	0	...	4.0	140	9200	55	39	274	50	0	Severe	

	Cath
0	Cad
1	Cad
2	Cad
3	Normal
4	Normal

[5 rows x 56 columns]

12.2.4 Checking dataset header

[5]:
```
data.columns # displays the column label of the dataset
```

[5]:
```
Index(['Age', 'Weight', 'Length', 'Sex', 'BMI', 'DM', 'HTN', 'Current Smoker',
       'EX-Smoker', 'FH', 'Obesity', 'CRF', 'CVA', 'Airway disease',
       'Thyroid Disease', 'CHF', 'DLP', 'BP', 'PR', 'Edema',
       'Weak Peripheral Pulse', 'Lung rales', 'Systolic Murmur',
       'Diastolic Murmur', 'Typical Chest Pain', 'Dyspnea', 'Function Class',
       'Atypical', 'Nonanginal', 'Exertional CP', 'LowTH Ang', 'Q Wave',
       'St Elevation', 'St Depression', 'Tinversion', 'LVH',
       'Poor R Progression', 'BBB', 'FBS', 'CR', 'TG', 'LDL', 'HDL', 'BUN',
       'ESR', 'HB', 'K', 'Na', 'WBC', 'Lymph', 'Neut', 'PLT', 'EF-TTE',
       'Region RWMA', 'VHD', 'Cath'],
      dtype='object')
```

12.2.5 Dimensions of the dataset

```
[6]: data.shape # shows the number of rows and columns in the dataset
```

```
[6]: (303, 56)
```

12.3 Dataset preprocessing

Next, we look into the dataset, to find if there exist any nonnumeric columns. We ignore such feature columns of the dataset.

12.3.1 Detecting nonnumeric columns

```
[7]: cols = data.columns
num_cols = data._get_numeric_data().columns
categorical_cols = list(set(cols) - set(num_cols))
print("Categorical Columns : ", categorical_cols)
```

```
Categorical Columns :  ['Poor R Progression', 'Dyspnea', 'Sex', 'LVH', 'BBB',
'Atypical', 'Weak Peripheral Pulse', 'CHF', 'Thyroid Disease', 'Airway disease',
'Obesity', 'Cath', 'CVA', 'LowTH Ang', 'Diastolic Murmur', 'Exertional CP',
'Systolic Murmur', 'Lung rales', 'VHD', 'Nonanginal', 'CRF', 'DLP']
```

12.3.2 Encoding nonnumeric columns

```
[8]: label_encoder = LabelEncoder()

for i in categorical_cols:
    data[i] = label_encoder.fit_transform(data[i])
print("Encoded data")
data.head()
```

```
Encoded data
```

[8]:

	Age	Weight	Length	Sex	BMI	DM	HTN	Current Smoker	EX-Smoker	\
0	53	90	175	1	29.387755	0	1	1	0	
1	67	70	157	0	28.398718	0	1	0	0	
2	54	54	164	1	20.077335	0	0	1	0	
3	66	67	158	0	26.838648	0	1	0	0	
4	50	87	153	0	37.165193	0	1	0	0	

	FH	...	K	Na	WBC	Lymph	Neut	PLT	EF-TTE	Region RWMA	VHD	Cath
0	0	...	4.7	141	5700	39	52	261	50	0	1	0
1	0	...	4.7	156	7700	38	55	165	40	4	1	0
2	0	...	4.7	139	7400	38	60	230	40	2	3	0
3	0	...	4.4	142	13000	18	72	742	55	0	2	1
4	0	...	4.0	140	9200	55	39	274	50	0	2	1

[5 rows x 56 columns]

12.3.3 Detecting missing values

```
[9]: any(X.isnull().sum())
```

```
[9]: False
```

12.3.4 Checking the class distribution

```
[10]: print(data.groupby('Cath').size()) # class distribution
```

```
Cath
0    216
1     87
dtype: int64
```

12.3.5 Separating independent and dependent variables

```
[11]: X = data.iloc[:,0:54] # Input feature variables
      y = data.iloc[:,-1]   # Class label
```

12.4 Feature selection and normalization

For predictive modeling, we perform the feature selection on input variables in order to use the most important (selective) variables for our task. We select the top-40 features for our task.

```
[12]: # Selecting the top-k features
      BestFeatures = SelectKBest(score_func=chi2, k=40)
      fit = BestFeatures.fit(X,y)
```

```
[13]: df_scores = pd.DataFrame(fit.scores_)
      df_columns = pd.DataFrame(X.columns)
```

```
[14]: f_Scores = pd.concat([df_columns,df_scores],axis=1)    # feature scores
      f_Scores.columns = ['Specification','Score']
```

```
[15]: top_features = f_Scores.nlargest(40,'Score')
```

[16]:
```
feat = {}
for i, (v1,v2) in enumerate(zip(top_features['Specification'],
 ↪top_features['Score'])):
    feat[v1] = v2

print(feat)
```

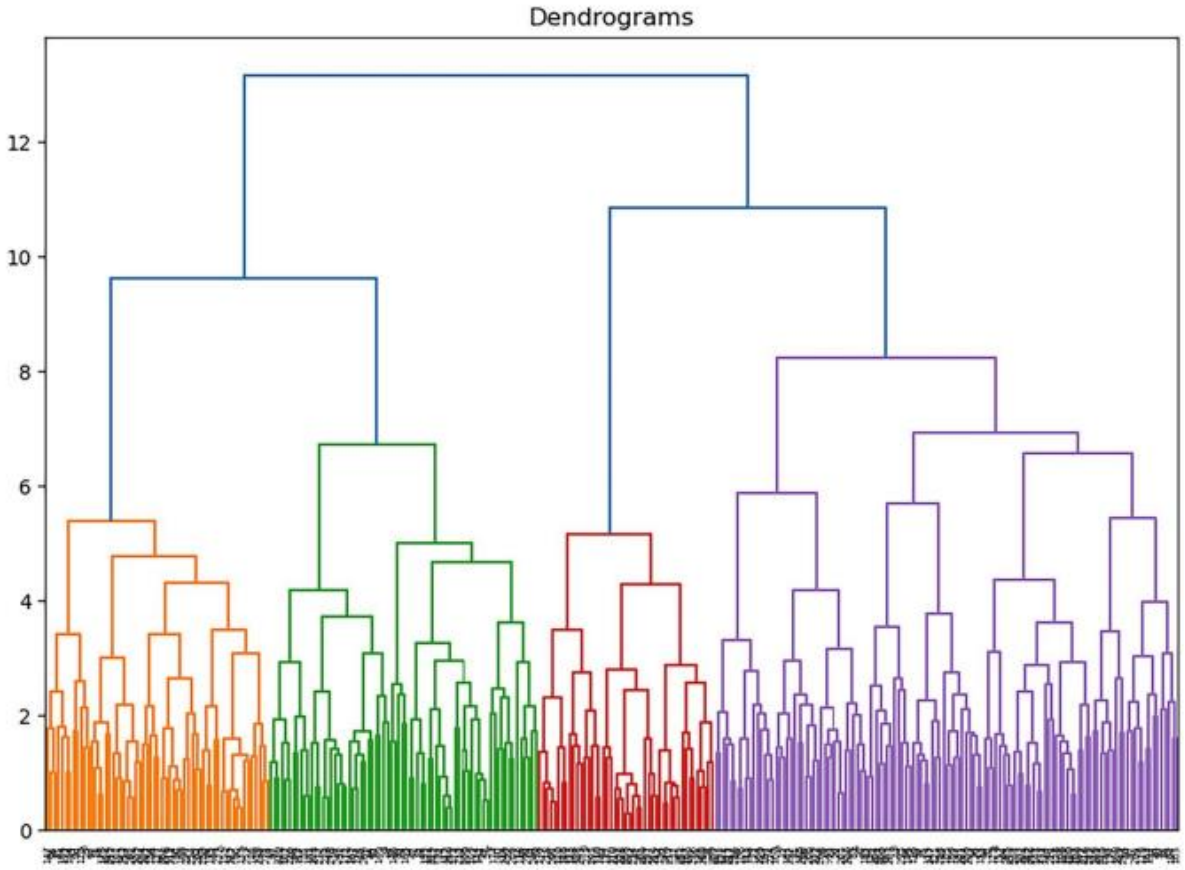
```
{'WBC': 1167.316183337504, 'TG': 381.0171907573125, 'FBS': 290.38280449892636,
'ESR': 125.49382868553845, 'Age': 70.67536529000986, 'Region RWMA':
62.344116837857655, 'BP': 47.262173693860326, 'PLT': 45.376214084848584,
'Typical Chest Pain': 40.97898560882161, 'Atypical': 36.32810118238372, 'EF-
TTE': 27.90445792173911, 'Nonanginal': 21.575700431034484, 'Lymph':
14.93885035810506, 'DM': 13.622818220519367, 'Tinversion': 11.957221157939546,
'HTN': 10.267987863610093, 'PR': 9.048198654830475, 'Neut': 8.015657437627791,
'BUN': 6.6005254923295755, 'Q Wave': 6.444444444444445, 'Diastolic Murmur':
6.3338122605364, 'St Elevation': 5.638888888888889, 'St Depression':
4.8392490961092225, 'Function Class': 4.570597682087642, 'Poor R Progression':
3.625, 'Dyspnea': 2.649543232115285, 'Weight': 2.625385886318694, 'CRF':
2.416666666666666, 'Airway disease': 2.0691396725879483, 'Weak Peripheral
Pulse': 2.0138888888888893, 'LDL': 2.0028214668383395, 'HDL': 1.517846316747344,
'Current Smoker': 1.2966687952320137, 'BMI': 1.1383731806674868, 'Edema':
0.8507343550446993, 'LowTH Ang': 0.8055555555555556, 'LVH': 0.74176245210728,
'Thyroid Disease': 0.6841817186644772, 'Lung rales': 0.5960031347962381, 'Sex':
0.5708050330895151}
```

[17]:
```
X = X[top_features['Specification']] # putting top-40 features in X
```

12.4.1 Correlation among features

Additionally, we check the correlation of the selected features among themselves. The diagonal column shows very high correlation of a variable with itself.

[18]:
```
# Checking the pairwise correlation of the top features columns with y-label

df = pd.concat([X, y], axis=1)

correlation = df.drop(['Cath'], axis=1).corr()
ax = sns.heatmap(correlation, annot=False, annot_kws={'size': 25})
ax.figure.axes[-1].yaxis.label.set_size(18)
```



12.4.2 Normalizing the columns

```
[19]: for column in X.columns:
          X[column] = (X[column] - X[column].min()) / (X[column].max() - X[column].
      ↪min())
```

12.4.3 Viewing normalized columns

```
[20]: X.head() # Top five rows after normalization
```

```
[20]:        WBC        TG       FBS       ESR       Age  Region RWMA   BP  \
      0  0.139860  0.210267  0.082840  0.067416  0.410714          0.0  0.2
      1  0.279720  0.268509  0.053254  0.280899  0.660714          1.0  0.5
      2  0.258741  0.065153  0.068047  0.101124  0.428571          0.5  0.1
      3  0.650350  0.025666  0.047337  0.842697  0.642857          0.0  0.1
      4  0.384615  0.131293  0.124260  0.292135  0.357143          0.0  0.2

             PLT  Typical Chest Pain  Atypical  ...       LDL       HDL  \
```

```
0  0.329149                 0.0       0.0  ...  0.640187  0.148265
1  0.195258                 1.0       0.0  ...  0.481308  0.211356
2  0.285914                 1.0       0.0  ...  0.242991  0.305994
3  1.000000                 0.0       0.0  ...  0.172897  0.116719
4  0.347280                 0.0       0.0  ...  0.429907  0.358570

   Current Smoker       BMI  Edema  LowTH Ang  LVH  Thyroid Disease  \
0             1.0  0.494721    0.0        0.0  0.0              0.0
1             0.0  0.451314    1.0        0.0  0.0              0.0
2             1.0  0.086105    0.0        0.0  0.0              0.0
3             0.0  0.382846    0.0        0.0  0.0              0.0
4             0.0  0.836058    0.0        0.0  0.0              0.0

   Lung rales  Sex
0         0.0  1.0
1         0.0  0.0
2         0.0  1.0
3         0.0  0.0
4         0.0  0.0

[5 rows x 40 columns]
```

Following the required transformation of the dataset for the downstream activity, we analyze the dataset before applying a few supervised and unsupervised modeling techniques. We employ a few classification and clustering algorithms to analyze predictive outcomes and observe various patterns that reside within the input data. We perform the classification task with some popular algorithms.

12.5 Classification

Classification is a supervised activity. We split the transformed dataset into requisite subsets for training and evaluation.

12.5.1 Splitting the dataset

[21]:
```
# Dataset splitting into 80% train and 20% test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=1)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(242, 40) (242,)
(61, 40) (61,)
```

12.5.2 Logistic regression

[22]:
```
logreg = LogisticRegression(solver='liblinear')

# Using 10-fold cross-validation
kfold = StratifiedKFold(n_splits=10)
logreg_cv_results = cross_val_score(logreg, X_train, y_train, cv=kfold,␣
 ↪scoring='accuracy')

print("Logistic Regression accuracy: ", logreg_cv_results.mean())
```

```
Logistic Regression accuracy:  0.8513333333333334
```

12.5.3 Support-vector machine

[23]:
```
svm_rbf = svm.SVC(kernel='rbf')


# Using 10-fold cross-validation
kfold = StratifiedKFold(n_splits=10)
svm_cv_results = cross_val_score(svm_rbf, X_train, y_train, cv=kfold,␣
 ↪scoring='accuracy')

print("SVM accuracy: ", svm_cv_results.mean())
```

```
SVM accuracy:  0.8758333333333332
```

12.5.4 Artificial neural network (ANN)

[24]:
```
def model_creation():
    model = keras.Sequential([
    keras.layers.Flatten(input_shape=(40,)),
    keras.layers.Dense(32, activation=tf.nn.relu),
    keras.layers.Dense(16, activation=tf.nn.relu),
    keras.layers.Dense(1, activation=tf.nn.sigmoid),
    ])

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

    return model

model = KerasClassifier(model=model_creation, epochs=15, batch_size=5, verbose=0)

# Using 10-fold cross-validation
kfold = StratifiedKFold(n_splits=10)
ann_cv_results = cross_val_score(model, X_train, y_train, cv=kfold)
print("ANN accuracy: ", ann_cv_results.mean())
```

```
ANN accuracy:  0.8846666666666666
```

12.5.5 Predictions on test data with a high-performing ANN model

[25]:
```
# Fitting X_train and y_train for prediction on the test data

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

12.5.6 Evaluating the model

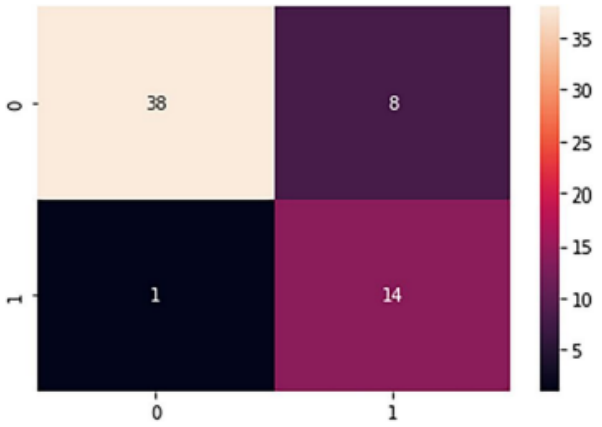
[26]:
```
# Test accuracy

print("Test Accuracy for ANN : ", accuracy_score(y_test, y_pred))
```

```
Test Accuracy for ANN :  0.8524590163934426
```

12.5.7 Performance measurement

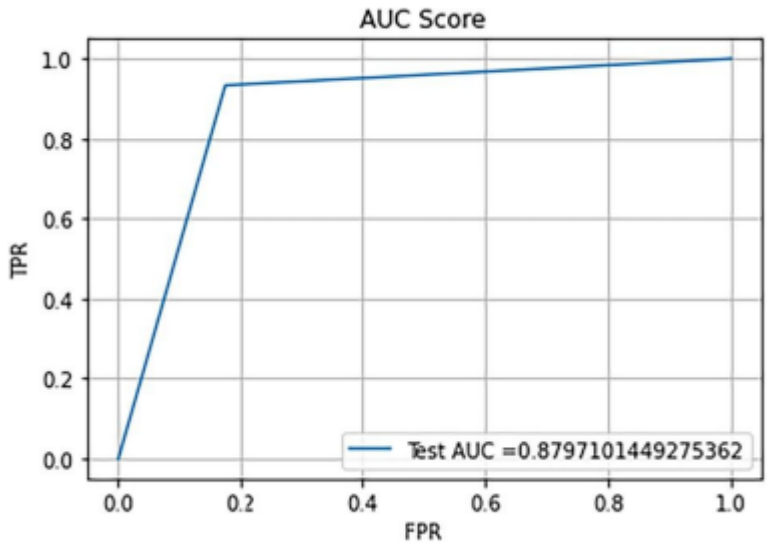
[27]:
```
# Confusion Matrix

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d')
```

[27]: <AxesSubplot:>



12.5.8 Curve plotting

```
[28]: # ROC Curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred)

plt.plot(fpr, tpr, label="Test AUC ="+str(auc(fpr, tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC Score")
plt.grid()
plt.show()
```



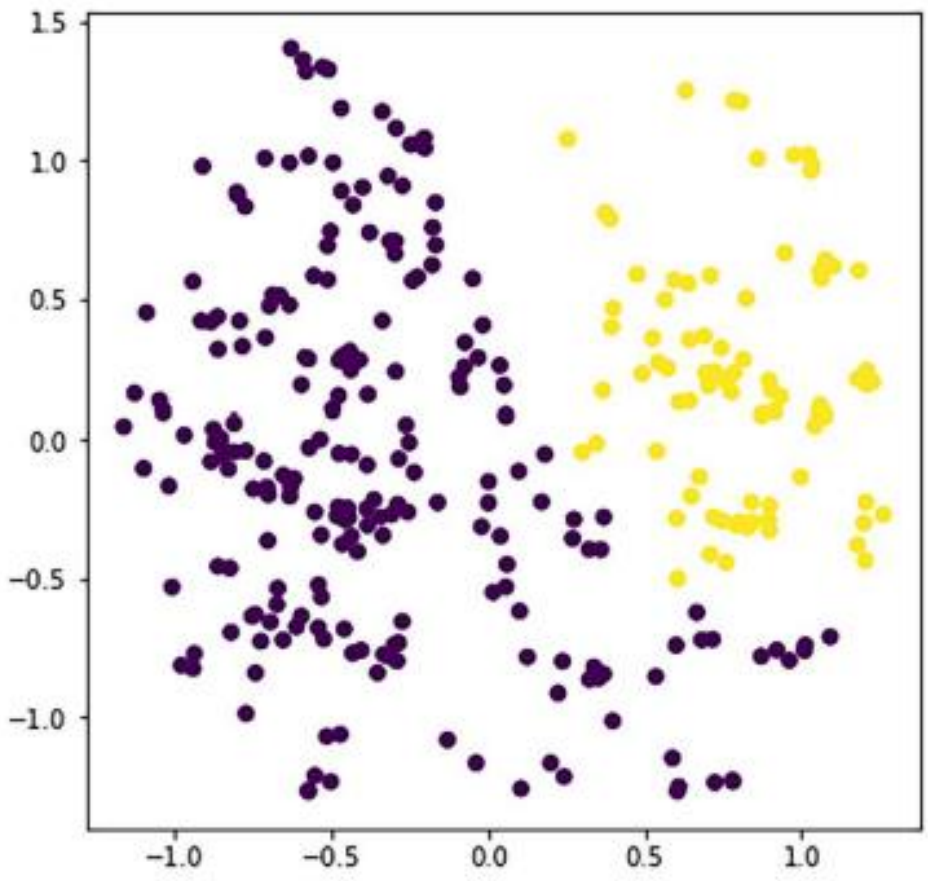
Next, we perform clustering, which is an unsupervised activity by employing a couple of algorithms to understand the grouped behavior of the data points that share similar properties or similar behavior.

12.6 Clustering

First, we use the K-means algorithm. As a preliminary activity for K-means, we use the elbow method to search for optimal clusters given the input dataset. More precisely, we compute inertia, which is the sum of squared distances of the samples nearest to the cluster center divided by the count of clusters (the fewer the better).

12.6.1 K-means—using the elbow method

```
[29]: inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init =␣
 ↪10, random_state = 16)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)
plt.plot(range(1, 11), inertia)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



12.6.2 Fitting the data

[30]:
```
# Using Cluster Count = 2

kmeans = KMeans(n_clusters=2,random_state=16)
kmeans.fit(X)
```

[30]:
```
KMeans(n_clusters=2, random_state=16)
```

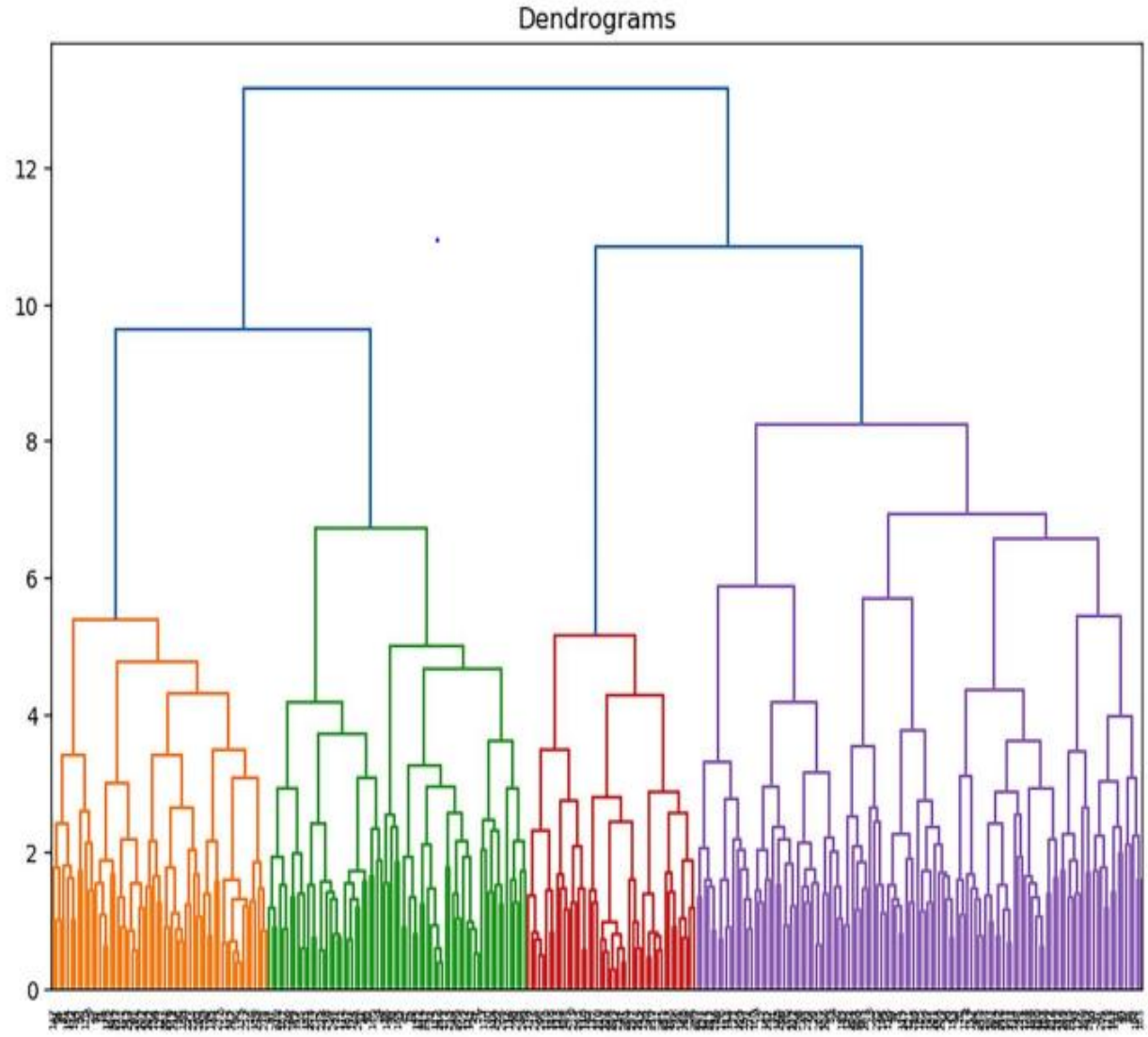
12.6.3 Validation with labels

[31]:
```
# Finding how many samples were correctly sampled

labels = kmeans.labels_
correct_labels = sum(y == labels)

print("Correctly sampled %d out of %d." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Correctly sampled 70 out of 303.
Accuracy score: 0.23
```

12.6.4 Agglomerative clustering—using dendrograms to find optimal clusters

[32]:
```
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(X, method='ward'))
```



12.6.5 Finding optimal clusters

[33]:
```
unique_colors = set(dend['color_list'])

optimal_clusters = len(unique_colors) - 1
print(optimal_clusters)
```

```
4
```

12.6.6 Fitting the data

[34]:
```
# Agglomerative Clustering

ag = AgglomerativeClustering(n_clusters=optimal_clusters, affinity='euclidean',␣
 ↪linkage='ward')
ag.fit_predict(X)
```

[34]:
```
array([2, 1, 3, 0, 0, 3, 3, 1, 0, 0, 3, 1, 1, 0, 0, 0, 3, 2, 1, 1, 2, 1,
       1, 0, 1, 3, 0, 2, 1, 1, 3, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0,
       0, 3, 0, 1, 0, 0, 0, 1, 1, 3, 0, 1, 0, 1, 2, 1, 2, 3, 2, 0, 0, 1,
       1, 3, 0, 1, 0, 3, 0, 0, 1, 1, 0, 0, 3, 0, 1, 3, 0, 0, 1, 3, 0, 1,
       1, 1, 3, 2, 0, 3, 3, 3, 3, 0, 3, 3, 0, 3, 2, 1, 3, 1, 0, 0, 0, 1,
       2, 0, 1, 0, 1, 1, 0, 1, 1, 3, 2, 0, 0, 2, 0, 3, 3, 3, 2, 0, 2, 1,
       3, 3, 1, 3, 0, 2, 0, 0, 1, 1, 0, 0, 3, 0, 1, 3, 2, 1, 3, 0, 0, 0,
       3, 3, 0, 0, 0, 3, 0, 1, 3, 0, 0, 0, 0, 1, 2, 0, 2, 1, 1, 1, 1, 3,
       2, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0, 0, 2, 3, 2, 1, 2, 0, 2, 1, 0, 0,
       0, 2, 0, 0, 3, 0, 3, 0, 1, 0, 0, 1, 2, 0, 0, 3, 3, 1, 0, 0, 2, 2,
       0, 0, 0, 2, 0, 0, 1, 3, 1, 1, 1, 1, 0, 0, 2, 3, 0, 3, 2, 0, 1, 2,
       3, 1, 0, 2, 1, 0, 1, 3, 1, 2, 3, 3, 3, 0, 2, 2, 0, 3, 0, 2, 3, 0,
       0, 0, 0, 3, 2, 1, 2, 3, 0, 0, 1, 1, 2, 1, 1, 1, 2, 0, 3, 0, 0, 2,
       0, 2, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 3, 0, 0, 0, 1])
```

12.6.7 Validation with labels

[35]:
```
labels = ag.labels_
correct_labels = sum(y == labels)

print("Correctly sampled %d out of %d." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Correctly sampled 76 out of 303.
Accuracy score: 0.25
```

12.7 Summary

The chapter focuses on the practice of Data Science. It also emphasizes hands-on learning. The chapter provides step-by-step tutorials that guide readers through the process of implementing Data Science techniques using Python. These tutorials cover a wide range of topics, including data cleaning and preprocessing, feature engineering, model selection and evaluation, and data visualization using Python, covering popular libraries such as NumPy, Pandas, Scikit-learn, and Matplotlib. The chapter provides practical examples of how to preprocess data, perform feature selection, and train and evaluate machine learning model.