Ameet V Joshi

Machine Learning and Artificial Intelligence

Springer

Machine Learning and Artificial Intelligence

Ameet V Joshi

Machine Learning and Artificial Intelligence

Springer

Ameet V Joshi
Microsoft (United States)
Redmond, WA, USA

ISBN 978-3-030-26621-9 ISBN 978-3-030-26622-6 (eBook)
https://doi.org/10.1007/978-3-030-26622-6

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To everyone who believes in human intelligence to be mature enough to coexist with machine intelligence and be benefitted and entertained by it. . .

Foreword

I first got exposed to computing about 35 years back. It was the time when the term "microcomputers" was being replaced by the term "personal computers (PC)." Within a space of 5 years, there was this seismic shift that happened during which computing as a superspecialized domain with a total of only a few thousand people across the world working with these magical machines in cooled clean rooms suddenly became readily accessible to hundreds of millions. It dawned on everyone that computing was going to impact every aspect of business and eventually personal lives too. With that came the scramble to learn computing—whether it be about the basics of what a computer is or how to use applications like word processing or spreadsheets or engineers in every part of the world swarming to learn computer architectures and programming, algorithms, and data structures. If you were not learning any of those to some level of competency, you were heavily disadvantaged in your professions—regardless of the profession.

Fast forward 15 years to about 1995. PCs had evolved and become powerful and had taken over business computing with client-server architectures, and most people in developed countries had PCs of their own at work and home. Then, the terms Internet, web, Mosaic, Netscape, browser, webserver, HTML, and HTTP suddenly swept the computing landscape. The seismic shift that happened then entirely reframed how data and information became democratized at a scale several orders of magnitude greater than anything that preceded in the history of humanity. The Internet led to the creation of the smartphones which in turn exponentially increased the scale and reach of the Internet. The computing architectures, especially around data representation, storage, and distribution, went through a massive revolution. With that came the scramble to learn "Internet"—whether it was people of all ages learning the basics of what the Internet is, how to use a browser, and how to communicate with people electronically to authoring content as web consumables (emails, pages, sites, blogs, posts, tweets, etc.). Engineers swarmed to learn Internet architectures, web and mobile application development, distributed data architectures, etc. If you were not learning something along this spectrum, you were not only heavily disadvantaged professionally, you were probably disadvantaged socially.

Fast forward another 15 years. Advances in the Internet and mobile technologies lead to explosive growth in creation and aggregation of data. The massive aggregation of data and the virtually infinite computational capacity that goes with it evolved to being physically distributed and yet logically unified. This became the "Cloud." Organizing and making sense of this scale of data were not only beyond human capabilities, it was even beyond the capabilities of traditional algorithms. Traditional algorithms couldn't even scale to the "searching" through this sea of data let alone make sense of it. Yet, the easy availability of such vast volumes of data finally makes possible the long-held dream of computers transcending from "processing information" to actually "creating intelligence" by making sense of the data. Decades of research in ML and AI became re-energized and have made progress in leaps and bounds with the easy access to massive quantities of data and computation in the cloud. Thus, we are officially now in the age of "AI." Just as has happened in the PC and the Internet revolutions, there is now a realization that AI is going to transform not just computing but society and humanity in unprecedented ways. It is beginning to dawn on people that applicability of AI and machine learning go far beyond mundane things like facial recognition in their Facebook feeds. It is also beyond the flashy, sci-fi scenarios like self-driving cars and digital assistants. Businesses of every kind—health care, manufacturing, hospitality, financial, etc.—are in the process of massive transformation with data and AI at the center of it. In all facets of life, digital information without intelligence will become pointless. The next decade will be transformed by AI. The Internet age is now set to transition to the age of AI.

However, the "swarm" towards understanding AI is just beginning. In that context, I am thrilled that Dr. Ameet Joshi is coming out with this book on Machine Learning and Artificial Intelligence. When a topic is of such broad human interest and impact, one can never have enough sources of knowledge and learning. Compared to the kind of learning content that was created in the PC era or the Internet era, machine learning and AI related content is still sparse and somewhat narrowly targeted. This is an area which is still mysterious even for the technically savvy folks. While there is a need to address the full spectrum of audiences, there is an urgent need to demystify the space for folks who are comfortable with the math and science without necessarily being computer scientists or engineers. These are folks who are willing and capable of getting a good handle on ML and AI. They are folks who understand data and math at fairly advanced levels; they possibly understand traditional computing and algorithms—but they have been left off the island populated by the "priests of data science." They are eager to roll up their sleeves, dive in, and understand what ML and AI from its foundations. ML and AI learning is ripe for democratization. Dr. Joshi's book hits the exact right sweet spot on this. Most people who read this may not become data scientists—but all of them will gain a perspective on how ML and AI work and will be able to take it to whatever domain of work they will be involved in. They will be able to see every application of AI, be it a digital assistant they converse with or a spookily accurate targeting of an advertisement or price fluctuations of airline ticket prices in a very different light and with a much deeper understanding. The shroud of mystery around

many things digital happening around us will seem more obvious and matter of fact. More importantly, I fully expect people who read this book to use that learning to look for applications in their future domains of work. For some set of readers, it may even intrigue them enough to become practitioners of ML and AI as a profession. There is definitely a huge demand for such professionals.

Dr. Joshi is a passionate practitioner himself. His passion shows through in this book. He has worked on broad set of ML and AI problems, and that practical experience has helped him identify and explain relatable scenarios and applications in this book. This is one of those books which likely is more than a onetime read. In all likelihood, the readers will want to revisit sections of the book to refresh their understanding or dig deeper in an area when they see real-world applications of the techniques described in this book. I think those are the best kinds of books to own. Even as an industry veteran, I enjoyed reading the book and will keep a copy handy.

Happy reading.

General Manager Vij Rajarajan
Microsoft
Redmond, Washington, USA

Preface

One of the greatest physicists of all time and Nobel Laureate Dr. Richard Feynman was once asked by his peer to explain a property of Fermi-Dirac statistics that was very recently discovered then. Feynman, quickly said,

> Not only I will explain it to you, but I will prepare a lecture on it for freshman level.

However, quite unusually, after few days, he came back and admitted,

> I could not do it. I just could not reduce the explanation to freshman level. That means we really don't understand it.

It was quite a bold remark coming from even Dr. Feynman. However, apart from the topic of Fermi-Dirac statistics itself, it alludes to a very deep thought about our understanding of things in general. Freshman level here essentially meant something that can be derived directly using the first principles in mathematics or physics. This thought has always made me conscious to try and explain everything that I claim to understand using first principles, try to explain everything conceptually and not only using elaborate set of equations.

The area of *artificial intelligence* and *machine learning* has exploded over the last decade. With the widespread popularity, the core concepts in the field have been sometimes diluted and sometimes reinterpreted. With such exponential growth in the area, the scope of the field has also grown in proportion. A newcomer in the area can quickly find the topic daunting and confusing. One can always start with searching the relevant topics on the web or just start with Wikipedia, but more often than not, every single topic opens a rabbit hole with more and more new and unknown concepts, and one can get lost very easily. Also, most of the concepts in machine learning are deeply rooted in mathematics and statistics. Without solid background in theoretical mathematics and statistics, the sophisticated derivations of the theorems and lemmas can make one feel confused and disinterested in the area.

I have made an attempt here to introduce most fundamental topics in machine learning and their applications to build artificially intelligent solutions with an intuitive and conceptual approach. There would be some mathematical guidance

used from time to time, without which the concepts would not be sufficiently clear, but I have tried to avoid complex derivations and proofs to make the content more accessible to readers who are not from strong mathematical background. In the process, as per Dr. Feynman, I also make sure I have understood them myself. As far as general mathematical and statistical requirements go, I would say that typical undergraduate level should suffice. Also, with proliferation and standardization of the machine learning libraries in open-source domain, one does not need to go that deep into mathematical understanding of the theory to be able to implement the state-of-the-art machine learning models, leading to the state-of-the-art intelligent solutions.

One of the main sources of confusion that arises when trying to solve a problem in the given application is the choice of algorithm. Typically, each algorithm presented here has originated from some specific problem, but the algorithm is typically not restricted to solving only that problem. However, choosing the right algorithm for the given problem is not trivial even for a doctoral fellow with strong mathematical background. In order to separate these two areas, I have divided these two areas into separate parts altogether. This will make the topics much easier to access for the reader.

I would recommend the reader to start with *Part I* and then choose *Part II* or *Part III*, depending on the needs. It will be ideal for a student to go sequentially through the book, while a newcomer to the area from professional background would be better suited to start with *Part III* to understand or focus on the precise application at hand and then delve into the details of the theory for the algorithms as needed in *Part II*. *Part IV* and *Part V* should follow afterwards. I have added sufficient references between the two parts to make this transition smooth.

In my mind, unless one can see the models in action on real data that one can see and plot, the understanding is not complete. Hence, following the details of algorithms and applications, I have added another part to cover the basic implementation of the models using free and open-source options. Completion of this part will enable the reader to tackle the real-world problems in AI with state-of-the-art ML techniques!

Redmond, WA, USA Ameet Joshi
March 2019

Acknowledgments

I would like to use this opportunity to acknowledge the people who had deep impact on the creation of this book. This, being the first ever book written by me, was quite a challenging task. There were quite a few times when I had almost given up. However, the continuous support and encouragement from my wife, Meghana, and sons, Dhroov and Sushaan, really helped me continue with the efforts and ultimately complete the book. I would also like to thank my father, Vijay, and brother, Mandar, for their feedback and support.

I would like to thank Mary James of Springer publications for the encouragement and support and understanding as I worked through the completion of the book.

The present book is an exercise in unification of the disparate areas in the field of machine learning to produce artificially intelligent experiences, and hence, the book essentially stands on the pillars of this knowledge that is created by the numerous extraordinary scientists and brilliant mathematicians over several decades in the past. So, I would like to thank them as well.

Last but not the least, I have used multiple quotes from some of the landmark movies produced over the last 60–70 years that depicted various scenarios of AI. These movies, like *2001: A Space Odyssey*, *Star Wars*, *WarGames*, *Terminator 2: Judgment Day*, and *The Matrix*, have showcased the power and scope of what AI can do and how it can change our lives. These cinematic interpretations express these extremely complex and technical concepts to laymen in highly influential manner and prepare them for what is coming in the future. Without them, the technological progress in this area would remain disconnected from people, so I would like to thank the creators of these movies.

Contents

Part I
Introduction

> This is your last chance. After this, there is no turning back. You take the blue pill - the story ends, you wake up in your bed and believe whatever you want to believe. You take the red pill - you stay in Wonderland and I show you how deep the rabbit-hole goes. The question is, which pill would you choose?
>
> —Morpheus, "The Matrix"

Part Synopsis

This part introduces the concepts of Artificial Intelligence (AI) and Machine Learning (ML) in the modern context. Various scenarios where these concepts are used will also be discussed. Going further, this part also discusses the understanding, representation and visualization aspects of data that form the foundation to the next topics. This will serve as a good starting point to delve into the details of the techniques, applications and implementations.

Chapter 1
Introduction to AI and ML

1.1 Introduction

It is an understatement to say that the field of Artificial Intelligence and Machine Learning has exploded in last decade. There are some rather valid reasons for the exponential growth and some not so much. As a result of this growth and popularity, there are also some of the most wrongly used words. Along with the enormous growth of the field it has also become one of the most confusing fields to understand the scope of. The very definitions of the terms AI and ML have also become diluted and vague.

We are not going to try and define these entities with word play, but rather are going to learn the context around the origins and scope of these entities. This will make their meaning apparent. The roots of these words originate from multiple disciplines and not just computer science. These disciplines include pure mathematics, electrical engineering, statistics, signal processing, and communications along with computer science to name the top few. I cannot imagine any other area that emerges as a conflation of such wide variety of disciplines. Along with the wide variety of origins, the field also finds applications in even greater number of industries ranging from high-tech applications like image processing, natural language processing to online shopping to nondestructive testing of nuclear power plants and so on.

In spite of being such multi-disciplinary, there are several things that we can conceptually learn collectively and obtain clarity of its scope. That is the primary objective behind this book. I want to make the field accessible to newcomers to the field in the form of graduate level students to engineers and professionals who are entering the field or even actively working in this field.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_1

1.2 What Is AI

Alan Turing defined Artificial Intelligence as follows: "If there is a machine behind a curtain and a human is interacting with it (by whatever means, e.g. audio or via typing etc.) and if the human feels like he/she is interacting with another human, then the machine is artificially intelligent." This is quite a unique way to define AI. It does not directly aim at the notion of intelligence, but rather focusses on human-like behavior. As a matter of fact, this objective is even broader in scope than mere intelligence. From this perspective, AI does not mean building an extraordinarily intelligent machine that can solve any problem in no time, but rather it means to build a machine that is capable of human-like behavior. However, just building machines that mimic humans does not sound very interesting. As per modern perspective, whenever we speak of AI, we mean machines that are capable of performing one or more of these tasks: understanding human language, performing mechanical tasks involving complex maneuvering, solving computer-based complex problems possibly involving large data in very short time and revert back with answers in human-like manner, etc.

The supercomputer depicted in movie 2001: A space odyssey, called HAL represents very closely to modern view of AI. It is a machine that is capable of processing large amount of data coming from various sources and generating insights and summary of it at extremely fast speed and is capable of conveying these results to humans in human-like interaction, e.g., voice conversation.

There are two aspects to AI as viewed from human-like behavior standpoint. One is where the machine is intelligent and is capable of communication with humans, but does not have any locomotive aspects. HAL is example of such AI. The other aspect involves having physical interactions with human-like locomotion capabilities, which refers to the field of robotics. For the purpose of this book, we are only going to deal with the first kind of AI.

1.3 What Is ML

The term "Machine Learning" or ML in short, was coined in 1959 by Arthur Samuel in the context of solving game of checkers by machine. The term refers to a computer program that can learn to produce a behavior that is not explicitly programmed by the author of the program. Rather it is capable of showing behavior that the author may be completely unaware of. This behavior is learned based on three factors: (1) Data that is consumed by the program, (2) A metric that quantifies the error or some form of distance between the current behavior and ideal behavior, and (3) A feedback mechanism that uses the quantified error to guide the program to produce better behavior in the subsequent events. As can be seen the second and third factors quickly make the concept abstract and emphasizes deep

mathematical roots of it. The methods in machine learning theory are essential in building artificially intelligent systems.

1.4 Organization of the Book

I have divided this book into six parts, as described below.

Parts of the Book

1. Introduction
2. Machine Learning
3. Building end to end pipelines
4. Artificial Intelligence
5. Implementations
6. Conclusion

1.4.1 Introduction

The first part of the book introduces the key concepts in the field as well as outlines the scope of the field. There are multiple aspects discussed in these chapters that may appear disconnected, but put together they are all extremely important in holistic understanding of the field and inspiring the confidence. This part also discusses the preliminary understanding and processing of the raw data as is typically observed. This sets the foundation for the reader to understand the subsequent parts.

1.4.2 Machine Learning

Second part of the book deals with the theoretical foundation of the machine learning. In this part we will study various algorithms, their origins and their applications as well. Machine learning has united a vast array of algorithms that find their origins in fields ranging from electrical engineering, signal processing, statistics, financial analysis, genetic sciences, and so on. All these algorithms are primarily developed from the principles of pure mathematics and statistics, in spite of being originated in fundamentally different areas. Along with the roots, they also have one more thing in common: the use of computers to automate complex computations. These computations ultimately lead to solving problems that seem so hard that one would believe that they are solved by some intelligent entity, or *Artificial Intelligence*.

1.4.3 Building End to End Pipelines

After learning about the theory of machine learning, one is *almost* ready to start solving problems in real world. However, the small gap that still remains is addressed in this part. Putting together theory and application is the primary objective to be learned from this part. Invariably the machine learning system built to solve a real life problem always involves multiple components being operated in sequence, creating a pipeline of operations. This part discusses all the essential components of such pipeline that takes the raw data at the input and produces interpretable results at the output.

1.4.4 Artificial Intelligence

This is likely the most interesting part of the book that directly deals with problems that machine learning has solved and we experience in our day-to-day life. These applications represent real problems that have been solved in variety of different industries, e.g., face recognition requirements for national security, detecting spam emails for e-commerce providers, deploying drones to carry out tactical jobs in areas where situations are hazardous for humans, etc. These applications need help from the rich collection of machine learning techniques that are described in the previous part. It is important to separate the applications and techniques, as there is a non-trivial overlap between those areas. There are some techniques specifically developed for certain types of applications, but there are some applications that can be benefitted from variety of different techniques, and the relationship between them is many to many. Only when we start looking at the field from these angles separately, lot of confusion starts to melt away and things start to make sense. The seamless solution of these problems and their delivery to millions of people is what creates the feel of artificial intelligence.

1.4.5 Implementations

The completion of studying parts 1 and 2, one can have sufficient confidence in analyzing a given problem in the areas of machine learning and artificial intelligence and come up with an outline of solution. However, this outline is not sufficient in actually implement the solution. In most cases, the theory is quite complex and solution of optimization problem is non-trivial. I am not going to delve into deep mathematical proofs and flowcharts that can be directly translated into computer programs for two reasons:

1. The content can quickly become too mathematical and will lose the interest of most of the target audience.

2. The complete theory behind all the techniques discussed here is so vast, that it is just not within the scope of a single book.

However, what I am going to target here is a conceptual understanding of the techniques, that is sufficient for a professional computer engineer, or a data analyst/scientist to grasp underlying key parameters of the problem.

Fortunately, the open source libraries in machine learning have matured to such level that, with such conceptual understanding of the techniques and applications, one can confidently implement reliable and robust artificially intelligent solutions without need to fully understand the underlying mathematics. This is the target of the part 3 of the book. This part will truly enable the reader to go ahead and build end to end solution of a complex practical problem and demonstrate an artificially intelligent system.

1.4.6 Conclusion

In this last part we summarize the learnings and discuss the next steps for the reader. After completion of this book the reader is well equipped to embark on solving real world problems that are still unsolved and make the world a better place.

Chapter 2
Essential Concepts in Artificial Intelligence and Machine Learning

2.1 Introduction

There are various concepts that are spread all over the place in terms of categorizing them with techniques or applications or implementations. However, they are at the heart of the entire machine learning theory and its applications and need special attention. We will cover these topics in this chapter. We will keep revising them in the book, but aggregating them here will make the reader more equipped to study the subsequent chapters.

2.2 Big Data and Not-So-Big Data

2.2.1 What Is Big Data

Big data processing has become a very interesting and exciting chapter in the field of AI and ML since the advent of cloud computing. Although there is no specific memory size above which the data is called big, the generally accepted definition is as follows: When the size of data is large enough such that it cannot be processed on a single machine, it is called as big data. Based on current (2018) generation of computers this equates to something roughly more than 10 GB. From there it can go to hundreds of petabytes (1 petabyte is 1000 terabytes and 1 terabyte is 1000 gigabytes) and more.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_2

2.2.2 Why Should We Treat Big Data Differently?

Although the big data is separated from the not-so-big-data by simple size restrictions, the handling of the data changes drastically when we move from not-so-big data to big data. For processing not-so-big data, one need not worry about the location where each bit of data is stored. As all the data can be loaded into memory of single machine and can be accessed without additional overhead, all the typical numeric or string operations are executed in predictable time. However, once we enter the realm of big data, all such bets are off. Instead of a single machine, we are dealing with a cluster of machines. Each machine has its own storage and computational resources. For all the numeric and string operations one must carefully split the data into different machines, such that all the operands in individual operation are available on the same machine. If not, there is additional overhead of moving the data across the machines and it typically a very computation intensive operation. Iterative operations are especially very computation intensive for big data. Thus when dealing with big data, one must also carefully design the storage along with the operations.

2.3 Types of Learning

The machine learning algorithms are broadly classified into three types:

1. Supervised learning algorithms
2. Unsupervised learning algorithms
3. Reinforcement learning algorithms

Let's look at each type in detail.

2.3.1 Supervised Learning

For simplicity, let's consider the ML system as a black box that produces some output when given some input. If we already have a historical data that contains the set of outputs for a set of inputs, then the learning based on this data is called as supervised learning. A classic example of supervised learning is classification. Say we have measured 4 different properties (sepal length, sepal width, petal length, and petal width) of 3 different types of flowers (Setosa, Versicolor, and Virginica) [3]. We have measurements for say 25 different examples of each type of flower. This data would then serve as training data where we have the inputs (the 4 measured properties) and corresponding outputs (the type of flower) available for training the model. Then a suitable ML model can be trained in supervised manner. Once the model is trained, we can then classify any flower (between the three known types) based on the sepal and petal measurements.

2.3.2 Unsupervised Learning

In unsupervised learning paradigm, the labelled data is not available. A classic example of unsupervised learning is "clustering." Consider the same example as described in the previous subsection, where we have measurements of the sepal and petal dimensions of three types of flowers. However, in this case, we don't have exact names of the flowers for each set of measurements. All we have is set of measurements. Also, we are told that these measurements belong to flowers of three different types. In such cases, one can use unsupervised learning techniques to automatically identify three clusters of measurements. However, as the labels are not known, all we can do is call each cluster as flower-type-1, flower-type-2, and flower-type-3. If a new set of measurement is given we can find the cluster to which they are closest and classify them into one of them.

2.3.3 Reinforcement Learning

Reinforcement learning is a special type of learning method that needs to be treated separately from supervised and unsupervised methods. Reinforcement learning involves feedback from the environment, hence it is not exactly unsupervised, however, it does not have a set of labelled samples available for training as well and hence it cannot be treated as supervised. In reinforcement learning methods, the system is continuously interacting with the environment in search of producing desired behavior and is getting feedback from the environment.

2.4 Machine Learning Methods Based on Time

Another way to slice the machine learning methods is to classify them based on the type of data that they deal with. The systems that take in static labelled data are called static learning methods. The systems that deal with data that is continuously changing with time are called dynamic methods. Each type of methods can be supervised, or unsupervised, however, reinforcement learning methods are always dynamic.

2.4.1 Static Learning

Static learning refers to learning on the data that is taken as a single snapshot and the properties of the data remain constant over time. Once the model is trained on the data (either using supervised or unsupervised learning) the trained model can be applied to similar data anytime in the future and the model will still be valid and will perform as expected. Typical examples of this would be image classification of different animals.

2.4.2 Dynamic Learning

This is also called as time series based learning. The data in this type of problems is time sensitive and keeps changing over time. Hence the model training is not a static process, but the model needs to be trained continuously (or after every reasonable time window) to remain effective. A typical example of such problems is weather forecasting, or stock market predictions. A model trained a year back will be completely useless to predict the weather for tomorrow, or predict the price of any stock for tomorrow. The fundamental difference in the two types is notion of state. In static models, the state of the model never changes, while in dynamic models the state of the model is a function of time and it keeps changing.

2.5 Dimensionality

Dimensionality is often a confusing concept when dealing with various data sets. From the physical standpoint, dimensions are space dimensions: length, width, and height. (Let's not go deeper into physics with time as fourth dimension for simplicity.) However, in any of the real life scenario we never encounter more than these three dimensions. However, it is very common to have tens if not hundreds and more dimensions when we deal with data for machine learning. In order to understand these high dimensions, we need to look at the fundamental property of dimensions. The space dimensions are defined such that each of the dimensions is perpendicular or orthogonal to other two. This property of orthogonality is essential to have unique representation of all the points in this 3-dimensional space. If the dimensions are not orthogonal to each other, then one can have multiple representations for same points in the space and whole mathematical calculations based on it will fail. For example if we setup the three coordinates as length, width, and height with some arbitrary origin (The precise location of origin only changes the value of coordinates, but does not affect the uniqueness property and hence any choice of origin is fine as long as it remains unchanged throughout the calculation.) The coordinates (0,0,0) mark the location of origin itself. The coordinates (1,1,1) will mark a point space that is 1 unit away from the origin in each of the dimensions and is unique. No other set of coordinates will mean the same location in space.

Now, let's extend this concept for higher dimensions. It is relatively easy to add more dimensions mathematically, but is very hard to visualize them spatially. If we add a fourth dimension, it needs to be orthogonal to all the previous three dimensions. In such 4-dimensional space the coordinates of the origin would be (0,0,0,0). The point (1,1,1) in 3-dimensional space may have coordinates (1,1,1,0) in 4-dimensional space. As long as orthogonality is assured, uniqueness of coordinates will be guaranteed. In the same way, we can have any arbitrarily large number of dimensions and all the mathematical calculations will still hold.

Consider the example of Iris data described in previous chapters. The input has 4 features: lengths and widths of sepals and petals. As all these 4 features are independent of each other, these 4 features can be considered as orthogonal. Thus when we were solving the problem with Iris data, we were actually dealing with 4 dimensional input space.

2.5.1 Curse of Dimensionality

Even if adding arbitrarily large number of dimensions is fine from mathematical standpoint, there is one problem. With increase in dimensions the density of the data gets diminished exponentially. For example if we have 1000 data points in training data, and data has 3 unique features. Let's say the value of all the features is within 1–10. Thus all these 1000 data points lie in a cube of size $10 \times 10 \times 10$. Thus the density is 1000/1000 or 1 sample per unit cube. If we had 5 unique features instead of 3, then quickly the density of the data drops to 0.01 sample per unit 5-dimensional cube. The density of the data is important, as higher the density of the data, better is the likelihood of finding a good model, and higher is the confidence in the accuracy of the model. If the density is very low, there would be very low confidence in the trained model using that data. Hence, although high dimensions are acceptable mathematically, one needs to be careful with the dimensionality in order to be able to develop a good ML model with high confidence.

2.6 Linearity and Nonlinearity

Concept of linearity and nonlinearity is applicable to both the data and the model that built on top of it. However, the concept of linearity differs in each context. Data is called as linear if the relationship between the input and output is linear. To put this simply, when the value of input increases, the value of output also increases and vice versa. Pure inverse relation is also called as linear and would follow the rule with reversal of sign for either input and output. Figure 2.1 shows various possible linear relationships between input and output.

Linear models have a slightly more involved definition. All the models that use linear equations to model the relationship between input and output are called as linear models. However, sometimes, by preconditioning the input or output a nonlinear relationship between the data can be converted into linear relationship and then the linear model can be applied on it. For example if input and output are related with exponential relationship as $y = 5e^x$. The data is clearly nonlinear. However, instead of building the model on original data, we can build a model after applying a log operation. This operation transforms the original nonlinear relationship into linear one as $\log y = 5x$. Then we build the linear model to predict $\log y$ instead of y, which can then be converted to y by taking exponent. There can also be

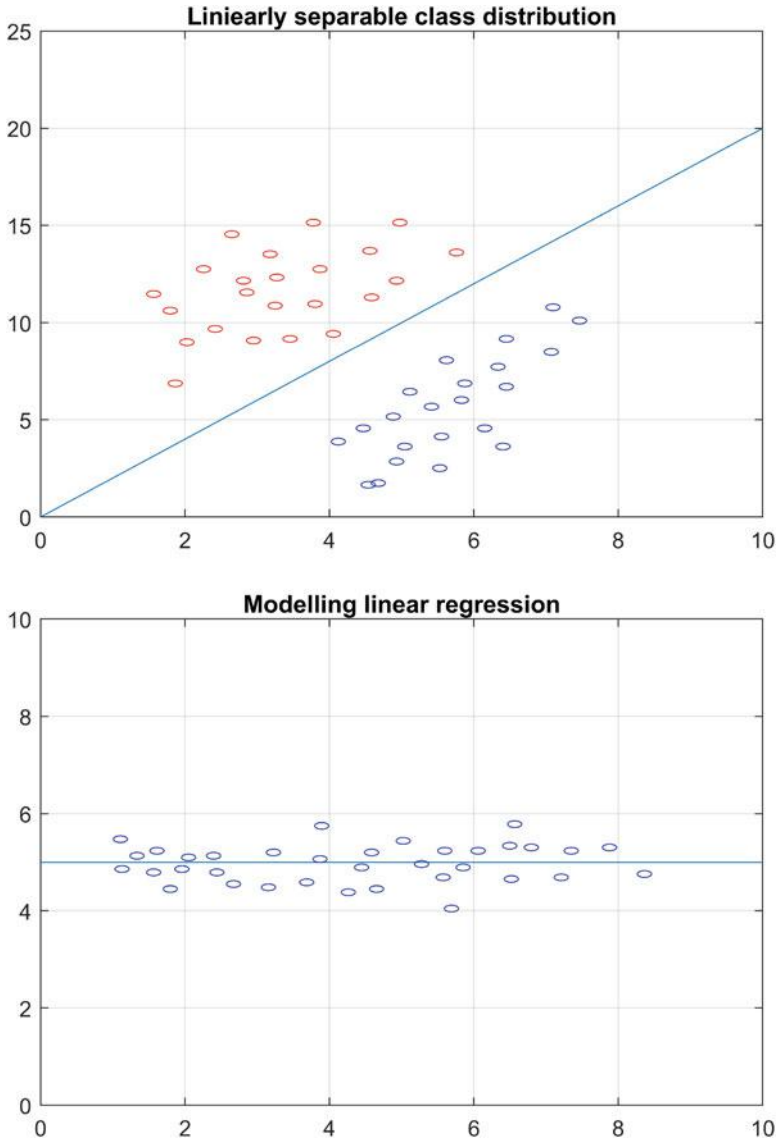


Fig. 2.1 Examples of linear relationships between input and output

cases where a problem can be broken down into multiple parts and linear model can be applied to each part to ultimately solve a nonlinear problem. Figures 2.2 and 2.3 show examples of converted linear and piecewise linear relationships, respectively. While in some cases the relationship is purely nonlinear and needs a proper nonlinear model to map it (Fig. 2.4). Figure 2.4 shows examples of pure nonlinear relationships.

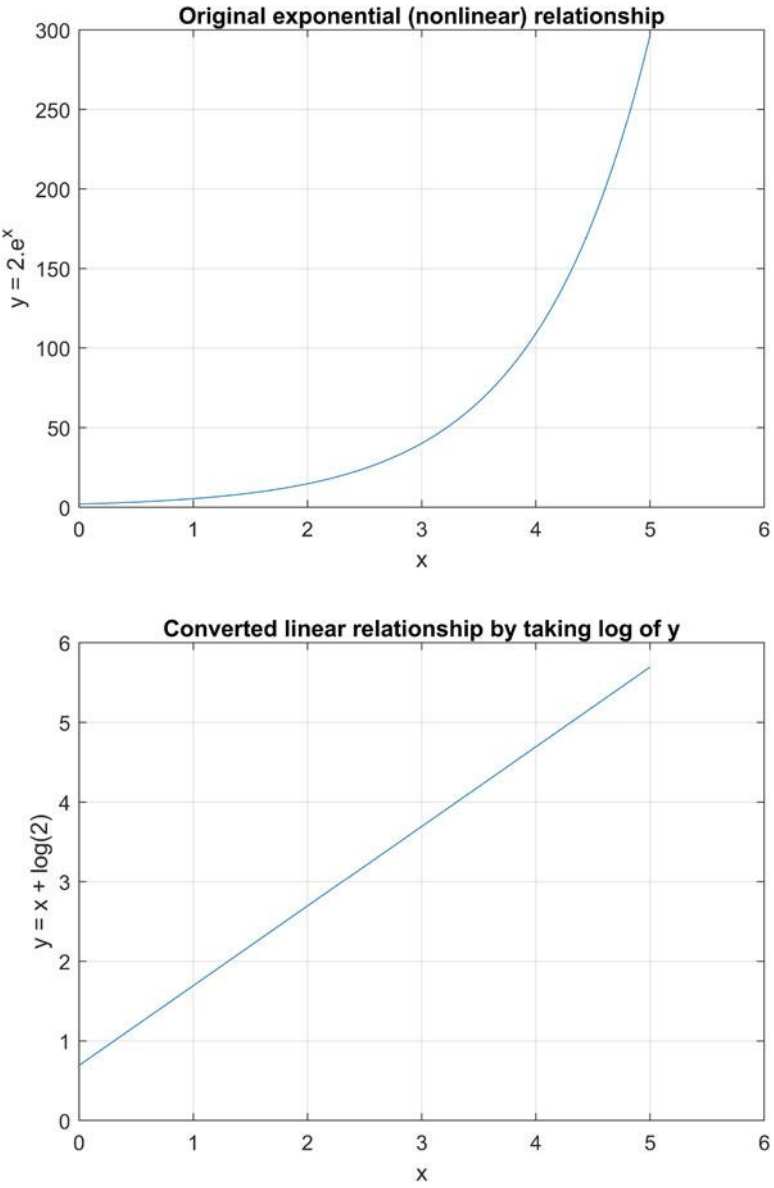


Fig. 2.2 Example of nonlinear relationship between input and output being converted into linear relationship by applying logarithm

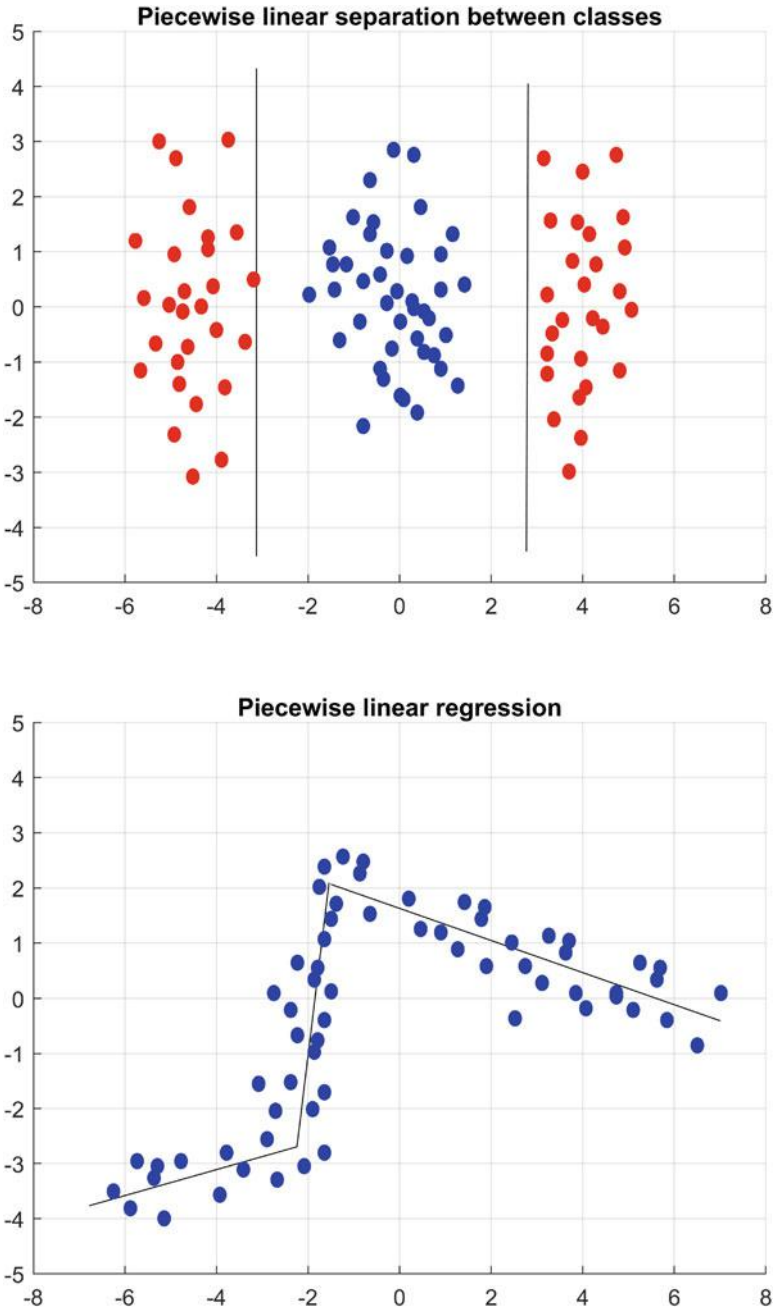


Fig. 2.3 Examples of piecewise linear relationships between input and output

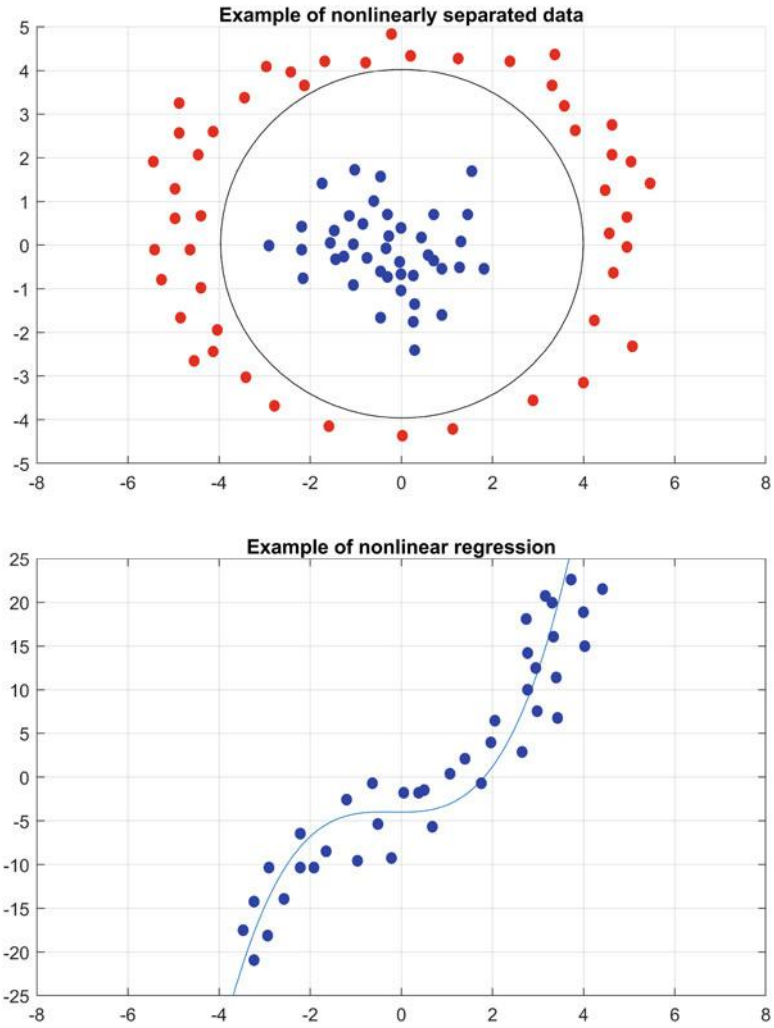


Fig. 2.4 Examples of pure nonlinear relationships between input and output

Linear models are the simplest to understand, build, and interpret. Our brain is highly tuned for linear models, as most of our experiences tend to have linear trends. Most times, what we call as intuitive behavior is mathematically a linear behavior. All the models in the theory of machine learning can handle linear data. Examples of purely linear models are linear regression, support vector machines without nonlinear kernels, etc. Nonlinear models inherently use some nonlinear functions to approximate the nonlinear characteristics of the data. Examples of nonlinear models include neural networks, decision trees, probabilistic models based on nonlinear distributions, etc.

In analyzing data for building the artificially intelligent system, determining the type of model to use is a critical starting step and knowledge of linearity of the relationship is a crucial component of this analysis.

2.7 Occam's Razor

In developing and applying machine learning models one always comes across multiple possible solutions and multiple possible approaches to get the answer. Many times there is no theoretical guideline as to which solution or which approach is better than the rest. In such cases the concept of *Occam's Razor*, which is also sometimes called as *principle of parsimony* can be effectively applied. The principle states,

Definition 2.1 Occam's Razor One should not make more assumptions than the minimum needed, or in other words, when there are multiple alternatives for a solution, the simplest approach is the best.

This principle is not quite a theorem and cannot be applied as a quantitative rule or equation. However, it stands a strong effective conceptual guideline when making such decisions in real life. It is also important to note that this rule creates a form of tradeoff, where on one side we have more information in the form of more complexity and on the other hand less information in the form of simplicity. One should not oversimplify the problem such that some of the core information is lost. Another derived aspect of Occam's razor is the simpler solutions tend to have more generalization capabilities.

2.8 No Free Lunch Theorem

Another interesting concept that is good to be aware of when designing a machine learning system comes from a paper by Wolpert and Macready [59], in the form of no free lunch theorem or NFL theorem in optimization. The theorem essentially states that

Definition 2.2 NFL Theorem If an algorithm performs better on certain class of problems, then it pays for it in the form of degraded performance in other classes of problems. In other words, you cannot have single optimal solution for all classes of problems.

This theorem needs to be taken more of a guideline than a law, as it is quite possible to have one well designed algorithm outperform some other not so well designed algorithm in all the possible classes of problems. However, in practical situations, we can infer from this theorem that we cannot have one solution for all the problems, and expect it to work well in all situations.

2.9 Law of Diminishing Returns

The law of diminishing returns is typically encountered in economics and business scenarios, where it states that adding more headcounts to complete a job starts to yield less and less returns with increase in existing number of headcount [9]. From the machine learning perspective, this law can be applied with respect to feature engineering. From the given set of data, one can only extract a certain number of features, after which the gains in performance start to diminish and the efforts are not worth the effect. In some ways it aligns with Occam's razor and adds more details.

2.10 Early Trends in Machine Learning

Before the machine learning started off commercially in true sense, there were few other systems that were already pushing the boundary if routine computation. One such notable application was *Expert Systems*.

2.10.1 Expert Systems

The definition by Alan Turin marks the beginning of the era where machine intelligence was recognized and with that field of AI was born. However, in the early days (all the way till 1980s), the field of Machine Intelligence or Machine Learning was limited to what were called as *Expert Systems* or *Knowledge based Systems*. One of the leading experts in the field of expert systems, Dr. Edward Feigenbaum, once defined as expert system as,

Definition 2.3 Expert Systems An intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for the solution [40].

Such systems were capable of replacing experts in certain areas. These machines were programmed to perform complex heuristic tasks based on elaborate logical operations. In spite of being able to replace the humans who are experts in the specific areas, these systems were not "intelligent" in the true sense, if we compare them with human intelligence. The reason being the system were "hard-coded" to solve only a specific type of problem and if there is need to solve a simpler but completely different problem, these system would quickly become completely useless. Nonetheless, these systems were quite popular and successful specifically in areas where repeated but highly accurate performance was needed, e.g., diagnosis, inspection, monitoring, control [40].

2.11 Conclusion

In this chapter we studied variety of different concepts that are used in the theory of machine learning and building artificially intelligent systems. These concepts arise from different contexts and different applications, but in this chapter we aggregated them at one place for reference. All these concepts need to be always at the back of mind when solving some real life problems and building artificially intelligent systems.

Chapter 3
Data Understanding, Representation, and Visualization

3.1 Introduction

Before starting with the theory of machine learning in the next part of the book, this chapter focusses on the understanding, representing, and visualizing the data. We will use the techniques described here multiple times throughout the book. These steps together can be called as data preprocessing.

With recent explosion of small devices that are connected to internet,[1] the amount of data that is being generated has increased exponentially. This data can be quite useful for generating variety of insights if handled properly, else it can only burden the systems handling it and slow down everything. The science that deals with general handling and organizing and then interpreting the data is called as data science. This is a fairly generic term and concepts of machine learning and artificial intelligence are part of it.

3.2 Understanding the Data

First step in building an application of artificial intelligence is to understand the data. The data in raw form can come from different sources, in different formats. Some data can be missing, some data can be mal-formatted, etc. It is the first task to get familiar with the data. Clean up the data as necessary.

The step of understanding the data can be broken down into three parts:

1. Understanding entities
2. Understanding attributes
3. Understanding data types

[1] Sometimes the network consisting of such devices is referred to as *internet of things* or *IOT*.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_3

In order to understand these concepts, let's us consider a data set called *Iris data* [3]. Iris data is one of the most widely used data set in the field of machine learning for its simplicity and its ability to illustrate many different aspects of machine learning at one place. Specifically, Iris data states a problem of multi-class classification of three different types of flowers, *Setosa*, *Versicolor*, and *Virginica*. The data set is ideal for learning basic machine learning application as it does not contain any missing values, and all the data is numerical. There are 4 features per sample, and there are 50 samples for each class totalling 150 samples. Here is a sample taken from the data.

3.2.1 Understanding Entities

In the field of data science or machine learning and artificial intelligence, entities represent groups of data separated based on conceptual themes and/or data acquisition methods. An entity typically represents a table in a database, or a flat file, e.g., comma separated variable (csv) file, or tab separated variable (tsv) file. Sometimes it is more efficient to represent the entities using a more structured formats like svmlight.[2] Each entity can contain multiple attributes. The raw data for each application can contain multiple such entities (Table 3.1).

In case of Iris data, we have only one such entity in the form of dimensions of sepals and petals of the flowers. However, if one is trying to solve this classification problem and finds that the data about sepals and petals alone is not sufficient, then he/she can add more information in the form of additional entities. For example more information about the flowers in the form of their colors, or smells or longevity of the trees that produce them, etc. can be added to improve the classification performance.

3.2.2 Understanding Attributes

Each attribute can be thought of as a column in the file or table. In case of Iris data, the attributes from the single given entity are *sepal length in cm*, *sepal width in cm*,

[2]The structured formats like *svmlight* are more useful in case of sparse data, as they add significant overhead when the data is fully populated. A sparse data is data with high dimensionality (typically in hundreds or more), but with many samples missing values for multiple attributes. In such cases, if the data is given as a fully populated matrix, it will take up a huge space in memory. The formats like svmlight employ a name-value pair approach to specify the name of attribute and its value in pair. The name-value pairs are given for only the attributes where value is present. Thus each sample can now have different number of pairs. The model needs to assume that for all the missing name-value pairs, the data is missing. In spite of added names in each sample, due to the sparsity of the data, the file is much smaller.

Table 3.1 Sample from Iris data set containing 3 classes and 4 attributes

Sepal-length	Sepal-width	Petal-length	Petal-width	Class label
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
4.8	3.4	1.9	0.2	Iris-setosa
5.0	3.0	1.6	0.2	Iris-setosa
5.0	3.4	1.6	0.4	Iris-setosa
5.2	3.5	1.5	0.2	Iris-setosa
5.2	3.4	1.4	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4.0	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
6.7	3.1	4.7	1.5	Iris-versicolor
6.3	2.3	4.4	1.3	Iris-versicolor
5.6	3.0	4.1	1.3	Iris-versicolor
5.5	2.5	4.0	1.3	Iris-versicolor
5.5	2.6	4.4	1.2	Iris-versicolor
6.3	3.3	6.0	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3.0	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica
6.5	3.0	5.8	2.2	Iris-virginica
6.7	3.0	5.2	2.3	Iris-virginica
6.3	2.5	5.0	1.9	Iris-virginica
6.5	3.0	5.2	2.0	Iris-virginica
6.2	3.4	5.4	2.3	Iris-virginica
5.9	3.0	5.1	1.8	Iris-virginica

petal length in cm, *petal width in cm*. If we had added additional entities like color, smell, etc., each of those entities would have their own attributes. It is important to note that in the current data, all the columns are all features, and there is no *ID* column. As there is only one entity, ID column is optional, as we can assign arbitrary unique ID to each row. However, when we have multiple entities we need to have an ID column for each entity along with the relationship between IDs of different entities. These IDs can then be used to join the entities to form the feature space.

3.2.3 Understanding Data Types

Attributes in each entity can be of various different types from the storage and processing perspective, e.g., string, integer valued, datetime, binary ("true"/"false", or "1"/"0"), etc. Sometimes the attributes can originate from completely different domains like an image or a sound file, etc. Each type needs to be handled separately for generating a feature vector that will be consumed by the machine learning algorithm. We will discuss the details of this processing in Chap. 15. As discussed before, we can also come across sparse data, in which case, some attributes will have missing values. This missing data is typically replaced with special characters, which should not be confused with any of the real values. In order to process data with missing values, one can either fill those missing values with some default values, or use an algorithm that can work with missing data.

In case of Iris data, all the attributes are real valued numerical and there is no missing data. However, if we add additional entities like color, it would have enumerative type string features like *green*, *orange*, etc.

3.3 Representation and Visualization of the Data

Even after we have understood the data with the three hierarchical levels, we still do not know how the data is distributed and how it related to the output or class label. This marks the last step in preprocessing the data. We live in 3-dimensional world, so any data that is up to 3 dimensional, we can plot it and visualize it. However, when there are more than 3-dimensions, it gets tricky. The Iris data, for example, also has 4 dimensions. There is no way we can plot the full information in each sample in a single plot that we can visualize. There are couple of options in such cases:

1. Draw multiple plots taking 2 or three dimensions at a time.
2. Reduce the dimensionality of the data and plot upto 3 dimensions

Drawing multiple plots is easy, but it splits the information and it becomes harder to understand how different dimensions interact with each other. Reducing dimensionality is typically preferred method. Most common methods used to reduce the dimensionality are:

1. Principal Component Analysis or PCA
2. Linear Discriminant Analysis or LDA

3.3.1 Principal Component Analysis

We can only visualize the data in maximum of 2 or 3 dimensions. However, it is common practice to have the dimensionality of the data in tens or even hundreds.

In such cases, we can employ the algorithms just fine, as the mathematics on which they are based scales perfectly fine for higher dimensions. However, if we want to actually have a look at the data to see the trends in the distribution or to see the classification or regression in action, it becomes impossible. One way to do this is to plot the data in pairs of dimensions. However, in such case, we only see partial information in any one plot and it is not always intuitive to understand the overall trends by looking at multiple separate plots. In many situations, the real dimensionality of data is much less than what the dimensionality in which the data is presented. For example, check the 3-dimensional data plotted in Fig. 3.1 and the same data plotted with different perspective in Fig. 3.2. As can be seen in the second perspective, the data is actually only 2 dimensional, if we can adjust the coordinates suitably. In other words, one can imagine all the data to be plotted on a single piece paper, which is only 2-dimensional and then holding it in skewed manner in a 3-dimensional space. If we can find exact coordinates (X', Y') of the paper's orientation as linear combination of X, Y, and Z coordinates of the 3-dimensional space, we can reduce the dimensionality of the data from 3 to 2. The above example is for illustrations only and in most real situations the lower dimensionality is not reflected in such obvious manner. Typically the data does have some information in all the dimensions, but the extent of the information in some dimensions can be very small compared to the information present in the other dimensions. For visualization purposes and also in real application purposes, it is quite acceptable to lose the information in those dimensions, without sacrificing any noticeable performance.

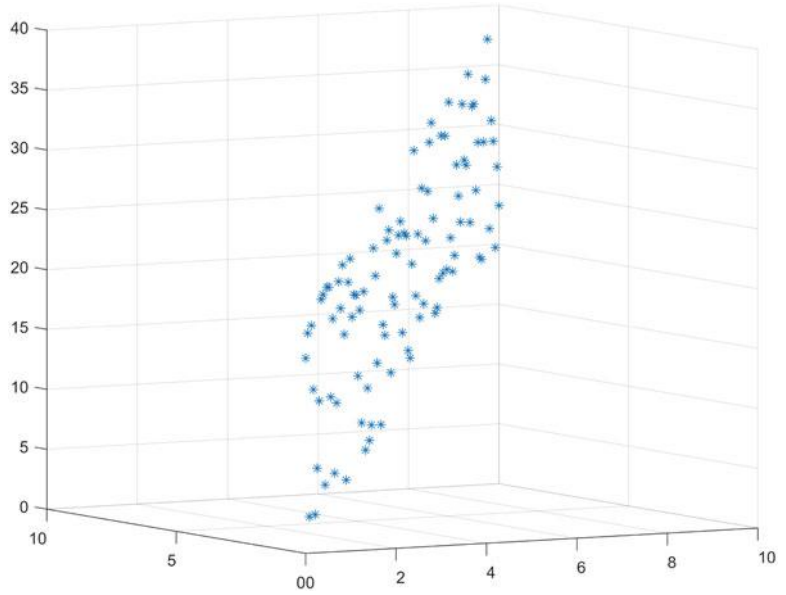


Fig. 3.1 3-dimensional data containing only 2-dimensional information with first perspective

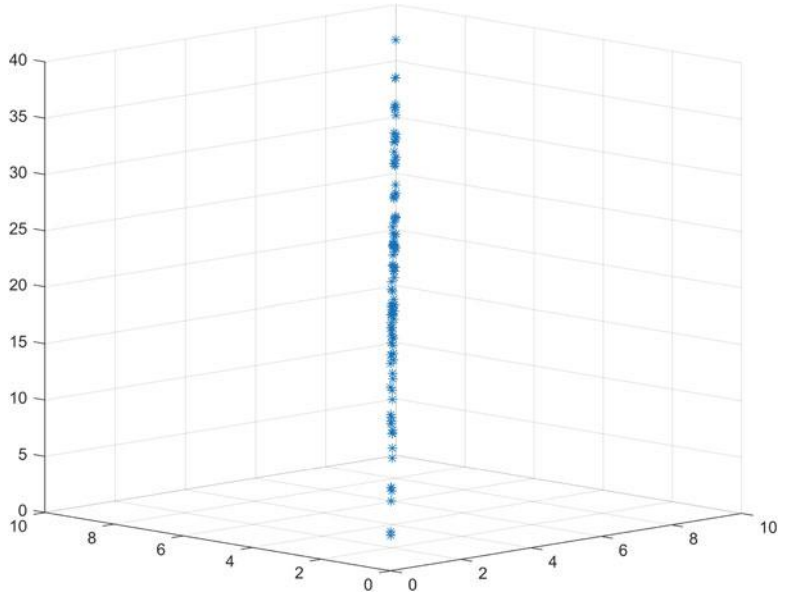


Fig. 3.2 3-dimensional data containing only 2-dimensional information with alternate perspective

As discussed in the previous chapter, the higher dimensionality increases the complexity of the problem exponentially, and such reduction in dimensionality for loss of some information is acceptable in most situations.

The mathematical process for finding the information spread across all the dimensions and then ranking the dimensions based on the information content is done using theory of principal component analysis or PCA. This theory is based on properties of matrices, specifically the process of *singular value decomposition (SVD)*. This process starts with first finding the dimension along which there is maximum spread or information content. If we have started with n-dimensions, then after first principal component is found the algorithm tries to find the next component of maximum spread in the remaining $n - 1$ dimensional space that is orthogonal to the first component. The process continues till we reach the last dimension. The process also gives coefficient for each of the principal components that represent the relative spread along that dimension. As all the components are found with deterministic and monotonically decreasing coefficient of spread, this representation is useful for any further analysis of the data. Thus this process is not restricted to only reducing the dimensions, but to find the optimal dimensions that represent the data. As can be seen in Fig. 3.3 (principal components are shown in red arrows), the original data and principal components are both 2-dimensional, but the representation of the data along principal components is different and preferable compared to original coordinates.

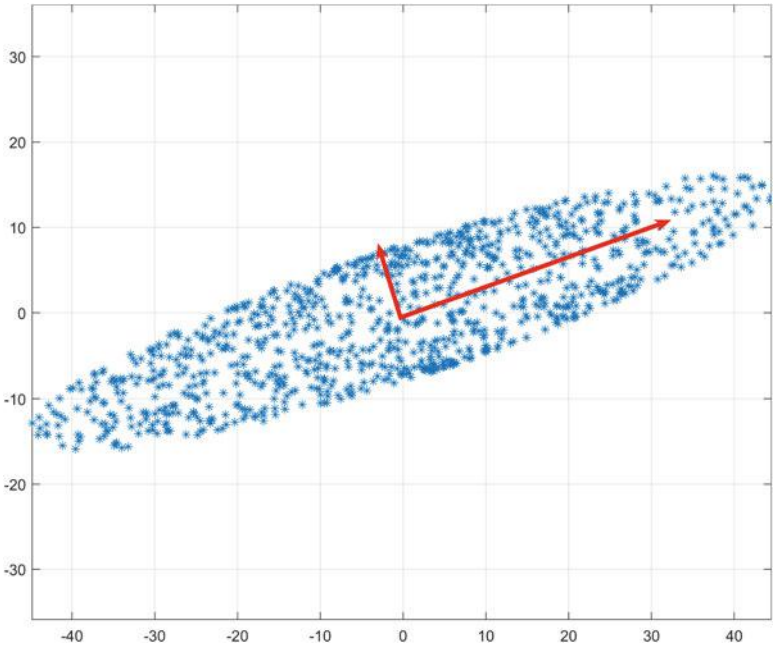


Fig. 3.3 2-dimensional data where PCA dimensionality is same but along different axes

3.3.2 Linear Discriminant Analysis

The way PCA tries to find the dimensions that maximize variance of the data, linear discriminant analysis or LDA tries to maximize the separation between the classes of data. Thus LDA can only effectively work when we are dealing with classification type of problem, and the data intrinsically contains multiple classes. Conceptually LDA tries to find the hyperplane in $c - 1$ dimensions to achieve: maximize the separation of means of the two classes and minimize the variance of each class. Here c is number of classes. Thus, while doing this classification it finds a representation of the data in $c - 1$ dimensions. For full mathematical details of theory one can refer to [5].

Figure 3.4 shows 3 dimensional data in one perspective, where the classes are not quite separable. Figure 3.5 shows another perspective of the data where classes are separable. LDA finds precisely this perspective as linear combination of the features and creates a 1-dimensional representation of the data on a straight line where the classes are best separated. As there are only two classes, the LDA representation is 1-dimensional. The LDA representation is independent of the original dimensionality of the data.

As can be seen, sometimes this representation is bit extreme and not quite useful in understanding the structure of the data and then PCA would be a preferred choice.

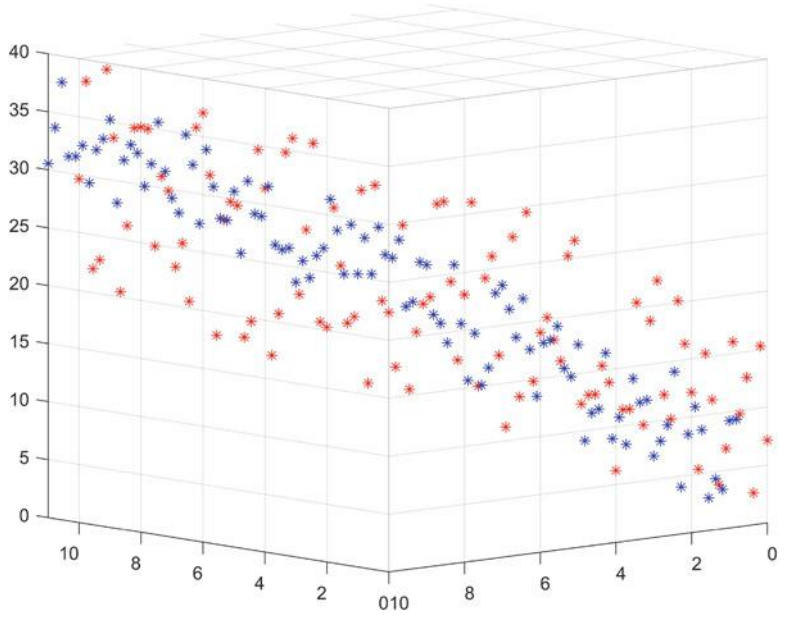


Fig. 3.4 3-dimensional data with 2 classes

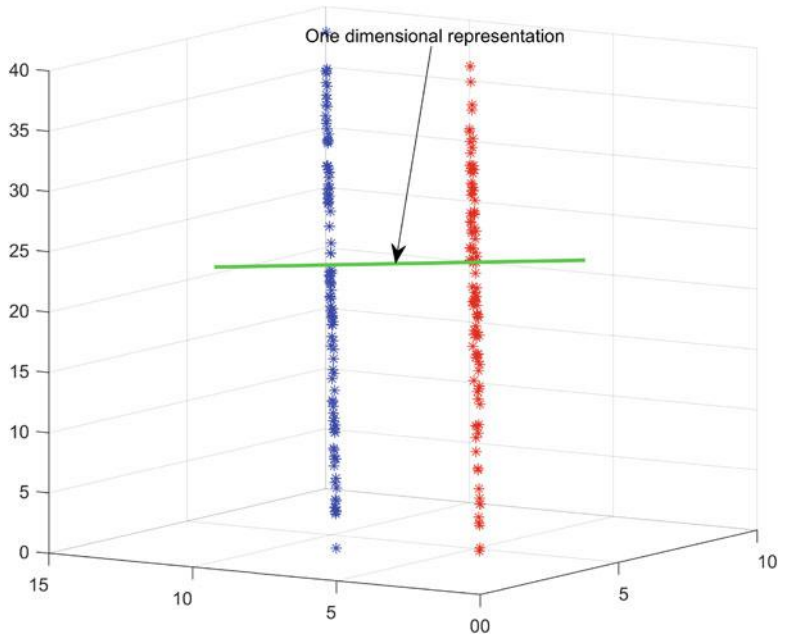


Fig. 3.5 3-dimensional data with 2 classes in another perspective, with LDA representation. LDA reduces the effective dimensionality of data into 1 dimension where the two classes can be best separated

3.4 Conclusion

In this chapter, we studied the different aspects of preparing the data for solving the machine learning problem. Each problem comes with a unique set of properties and quirks in the training data. All these custom abnormalities in the data need to be ironed out before it can be used to train a model. In this chapter we looked at different techniques to understand and visualize the data, clean the data, reduce the dimensionality as possible, and make the data easier to model in the subsequent steps in the machine learning pipeline.

Part II
Machine Learning

> The unknown future rolls toward us. I face it, for the first time, with a sense of hope. Because if a machine, a Terminator, can learn the value of human life, maybe we can too.
>
> —Sarah Connor, "Terminator 2: Judgement Day"

Part Synopsis

In this part we will focus on various techniques that form the foundation of all the static Machine Learning models. The static ML models address the class of problems where the data is static and there is no concept of time series. The model does not learn incrementally. If there is a new set of data, the entire learning process is repeated.

We will discuss these techniques from the conceptual and mathematical standpoint without going too much into the mathematical details and proofs. We will point the user with reference papers and books to find more details on theory whenever required.

Chapter 4
Linear Methods

4.1 Introduction

In general machine learning algorithms are divided into two types:

1. Supervised learning algorithms
2. Unsupervised learning algorithms

Supervised learning algorithms deal with problems that involve learning with guidance. In other words, the training data in supervised learning methods need labelled samples. For example for a classification problem, we need samples with class label, or for a regression problem we need samples with desired output value for each sample, etc. The underlying mathematical model then learns its parameters using the labelled samples and then it is ready to make predictions on samples that the model has not seen, also called as test samples. Most of the applications in machine learning involve some form of supervision and hence most of the chapters in this part of the book will focus on the different supervised learning methods.

Unsupervised learning deals with problems that involve data without labels. In some sense one can argue that this is not really a problem in *machine learning* as there is no knowledge to be learned from the past experiences. Unsupervised approaches try to find some structure or some form of trends in the training data. Some unsupervised algorithms try to understand the origin of the data itself. A common example of unsupervised learning is clustering.

In Chap. 2 we briefly talked about the linearity of the data and models. Linear models are the machine learning models that deal with linear data or nonlinear data that be somehow transformed into linear data using suitable transformations. Although these linear models are relatively simple, they illustrate fundamental concepts in machine learning theory and pave the way for more complex models. These linear models are the focus of this chapter.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_4

4.2 Linear and Generalized Linear Models

The models that operate on strictly linear data are called linear models, and the models that use some nonlinear transformation to map original nonlinear data to linear data and then process it are called as generalized linear models. The concept of linearity in case of supervised learning implies that the relationship between the input and output can be described using linear equations. For unsupervised learning, the concept of linearity implies that the distributions that we can impose on the given data are defined using linear equations. It is important to note that the notion of linearity does not imply any constraints on the dimensions. Hence we can have multivariate data that is strictly linear. In case of one-dimensional input and output, the equation of the relationship would define a straight line in two-dimensional space. In case of two-dimensional data with one-dimensional output, the equation would describe a two-dimensional plane in three-dimensional space and so on. In this chapter we will study all these variations of linear models.

4.3 Linear Regression

Linear regression is a classic example of strictly linear models. It is also called as polynomial fitting and is one of the simplest linear methods in machine learning. Let us consider a problem of linear regression where training data contains p samples. Input is n-dimensional as $(\mathbf{x}_i, i = 1, \ldots, p)$ and $\mathbf{x}_i \in \Re^n$. Output is single dimensional as $(y_i, i = 1, \ldots, p)$ and $y_i \in \Re$.

4.3.1 Defining the Problem

The method of linear regression defines the following relationship between input $\mathbf{x}_i$ and predicted output $\hat{y_i}$ in the form of linear equation as:

$$\hat{y_i} = \sum_{j=1}^{n} x_{ij}.w_j + w_0 \tag{4.1}$$

$\hat{y_i}$ is the predicted output when the actual output is y_i. $w_i, i = 1, \ldots, p$ are called as the weight parameters and w_0 is called as the bias. Evaluating these parameters is the objective of training. The same equation can also be written in matrix form as

$$\hat{\mathbf{y}} = \mathbf{X}^T.\mathbf{w} + w_0 \tag{4.2}$$

where $\mathbf{X} = [\mathbf{x}_i^T], i = 1, \ldots, p$ and $\mathbf{w} = [w_i], i = 1, \ldots, n$. The problem is to find the values of all weight parameters using the training data.

4.3.2 Solving the Problem

Most commonly used method to find the weight parameters is to minimize the mean square error between the predicted values and actual values. It is called as least squares method. When the error is distributed as Gaussian, this method yields an estimate called as maximum likelihood estimate or MLE. This is the best unbiased estimate one can find given the training data. The optimization problem can be defined as

$$\min \| y_i - \hat{y_i} \|^2 \tag{4.3}$$

Expanding the predicted value term, the full minimization problem to find the optimal weight vector $\mathbf{w^{lr}}$ can be written as

$$\mathbf{w^{lr}} = \arg\min_{w} \left\{ \sum_{i=i}^{p} \left(y_i - \sum_{j=1}^{n} x_{ij}.w_j - w_0 \right)^2 \right\} \tag{4.4}$$

This is a standard quadratic optimization problem and is widely studied in the literature. As the entire formulation is defined using linear equations, only linear relationships between input and output can be modelled. Figure 4.1 shows an example.

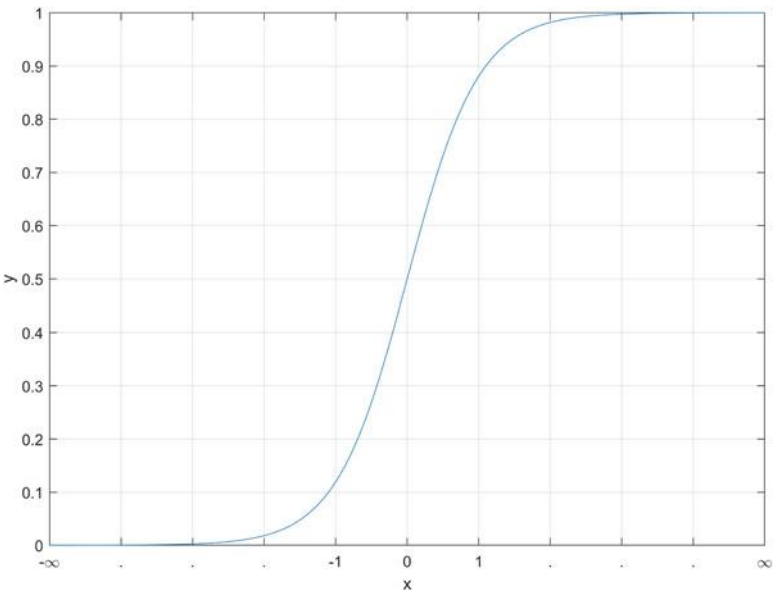


Fig. 4.1 Plot of logistic sigmoid function

4.4 Regularized Linear Regression

Although, in general, the solution obtained by solving Eq. 4.4 gives the best unbiased estimate, but in some specific cases, where it is known that the error distribution is not Gaussian or the optimization problem is highly sensitive to the noise in the data, above procedure can result in what is called as overfitting. In such cases, a mathematical technique called regularization is used.

4.4.1 Regularization

Regularization is a formal mathematical trickery that modifies the problem statement with additional constraints. The main idea behind the concept of regularization is to simplify the solution. The theory of regularization is typically attributed to Russian mathematician *Andrey Tikhonov*. In many cases the problems are what is referred to as *ill posed*. What it means is that the training data if used to full extent can produce a solution that is highly overfitted and possesses less generalization capabilities. Regularization tried to add additional constraints on the solution, thereby making sure the overfitting is avoided and the solution is more generalizable. The full mathematical theory of regularization is quite involved and interested reader can refer to [39].

Multiple regularization approaches are proposed in the literature and one can experiment with many. However, we will discuss two most commonly used ones. The approaches discussed below are also sometimes referred to as *shrinkage methods*, as they try to shrink the weight parameters close to zero.

4.4.2 Ridge Regression

In *Ridge Regression* approach, the minimization problem defined in Eq. 4.4 is constrained with

$$\sum_{j=1}^{n}(w_j)^2 \leq t \tag{4.5}$$

where t is a constraint parameter. Using Lagrangian approach,[1] the joint optimization problem can be written as

$$\mathbf{w}^{Ridge} = \arg\min_{w} \left\{ \sum_{i=1}^{p} \left(y_i - \sum_{j=1}^{n} x_{ij}.w_j - w_0 \right)^2 + \lambda \sum_{j=1}^{n}(w_j)^2 \right\} \tag{4.6}$$

λ is the standard Lagrangian multiplier.

[1] Lagrangian method is a commonly used method of integrating the regularization constraints into the optimization problem thereby creating a single optimization problem.

4.4.3 Lasso Regression

In *Lasso Regression* approach, the minimization problem defined in Eq. 4.4 is constrained with

$$\sum_{j=1}^{n} |w_j| \leq t \tag{4.7}$$

where t is a constraint parameter. Using Lagrangian approach, the joint optimization problem can be written as

$$\mathbf{w}^{Lasso} = \arg\min_{w} \left\{ \sum_{i=1}^{p} \left(y_i - \sum_{j=1}^{n} x_{ij}.w_j - w_0 \right)^2 + \lambda \sum_{j=1}^{n} |w_j| \right\} \tag{4.8}$$

4.5 Generalized Linear Models (GLM)

The *Generalized Linear Models* or *GLMs* represent generalization of linear models by expanding their scope to handle nonlinear data that can be converted into linear form using suitable transformations. The obvious drawback or limitation of linear regression is the assumption of linear relationship between input and output. In quite a few cases, the nonlinear relationship between input and output can be converted into linear relationship by adding an additional step of transforming one of the data (input or output) into another domain. The function that performs such transformation is called as basis function or link function. For example, logistic regression uses logistic function as basis function to transform the nonlinearity into linearity. Logistic function is a special case where it also maps the output between range of $[0-1]$, which is equivalent to a probability density function. Also, sometimes the response between input and output is monotonic, but not necessarily linear due to discontinuities. Such cases can also be converted into linear space with the use of specially constructed basis functions. We will discuss logistic regression to illustrate the concept of GLM.

4.5.1 Logistic Regression

The building block of logistic regression is the logistic sigmoid function $\sigma(x)$ and is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.9}$$

Logistic regression adds an exponential functional on top of linear regression to constrain the output $y_i \in [0, 1]$, rather than $y_i \in \Re$ as in linear regression. The relationship between input and predicted output for logistic regression can be given as

$$\hat{y_i} = \sigma\left(\sum_{j=1}^{n} x_{ij}.w_j + w_0\right) \tag{4.10}$$

As the output is constrained between [0, 1], it can be treated as a probabilistic measure. Also, due to symmetrical distribution of the output of logistic function between $-\infty - \infty$, it is also better suited for classification problems. Other than these differences, there is no fundamental difference between linear and logistic regressions. Although there is nonlinear sigmoid function present in the equation, it should not be mistaken for a nonlinear method of regression. The sigmoid function is applied after the linear mapping between input and output, and at heart this is still a variation of linear regression. The minimization problem that needs to be solved for logistic regression is a trivial update from the one defined in Eq. 4.1. Due to its validity in regression as well as classification problems, unlike the linear regression, logistic regression is the most commonly used approach in the field of machine learning as default first alternative.

4.6 *k*-Nearest Neighbor (KNN) Algorithm

The KNN algorithm is not exactly an example of a linear method, but it is one of the simplest algorithms in the field of machine learning, and is apt to discuss it here in the first chapter of this part. KNN is also a generic method that can be used as classifier or regressor. Unlike linear methods described before in this chapter, this algorithm does not assume any type equation or any type of functional relationship between the input and output.

4.6.1 Definition of KNN

In order to illustrate the concept of k-nearest neighbor algorithm, consider a case of 2-dimensional input data as shown in Fig. 4.2. The top plot in the figure shows the distribution of the data. Let there be some relationship between this input data and output data (not shown here). For the time being we can ignore that relationship. Let us consider that we are using the value of k as 3. As shown in bottom plot in the figure let there be a test sample located as shown by red dot. Then we find the 3 nearest neighbors of the test point from the training distribution as shown. Now, in order to predict the output value for the test point, all we need to do is find the value

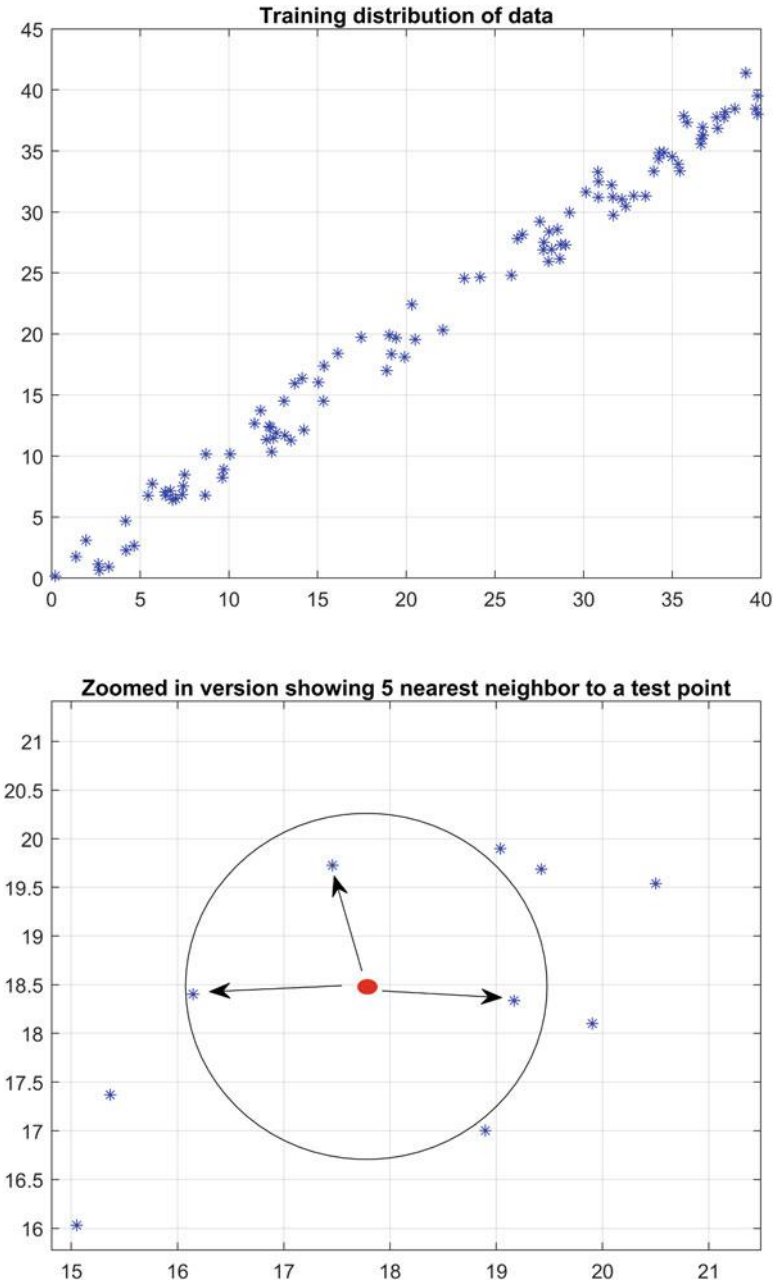


Fig. 4.2 Figure showing a distribution of input data and showing the concept of finding nearest neighbors

of the output for the 3 nearest neighbors and average that value. This can be written in equation form as

$$\hat{y} = \left(\sum_{i=1}^{k} y_i \right) / k \tag{4.11}$$

where y_i is the output value of the ith nearest neighbor. As can be seen this is one of the simplest way to define the input to output mapping. There is no need to assume any priory knowledge, or any need to perform any type of optimization. All you need to do is to keep all the training data in memory and find the nearest neighbors for each test point and predict the output.

This simplicity does come at a cost though. This lazy execution of the algorithm requires heavy memory footprint along with high computation to find the nearest neighbors for each test point. However, when the data is fairly densely populated, and computation requirements can be handled by the hardware, KNN produces good results in spite of being expressed with overly simplistic logic.

4.6.2 Classification and Regression

As the formula expressed in Eq. 4.11 can be applied to classification as well as regression problems, KNN can be applied to both types of problems without need to change anything in the architecture. Figure 4.2 showed as example of regression. Also, as KNN is a local method as opposed to global method, it can easily handle nonlinear relationships unlike the linear methods described above. Consider the two class nonlinear distribution as shown in Fig. 4.3. KNN can easily separate the two classes by creating the circular boundaries as shown based on the local neighborhood information expressed by Eq. 4.11.

4.6.3 Other Variations of KNN

As such, the KNN algorithm is completely described by Eq. 4.11. However, there exist some variations of the algorithm in the form of weighted KNN, where the value of each neighbors output is inversely weighted by its distance from the test point. In other variation, instead of using Euclidean distance, one can use Mahalanobis distance [28] to accommodate the variable variance of the data along different dimensions.

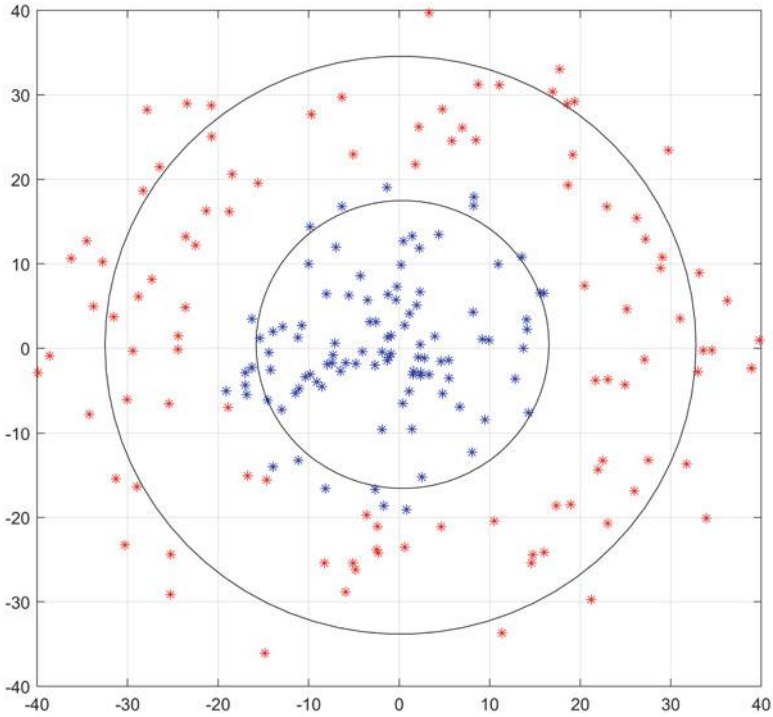


Fig. 4.3 Figure showing nonlinear distribution of the data

4.7 Conclusion

In this chapter we looked at some of the simple techniques to introduce the topic of machine learning algorithms. Linear methods form the basis of all the subsequent algorithms that we will study throughout the book. Generalized linear methods extend the scope of the linear methods to apply for some simple nonlinear cases as well as probabilistic methods. KNN is another simple technique that can be used to solve most basic problems in machine learning and also illustrates the use of local methods as opposed to global methods.

Chapter 5
Perceptron and Neural Networks

5.1 Introduction

Perceptron was introduced by Rosenblatt [44] as a generalized computation framework for solving linear problems. It was quite effective, one of a kind machine at the time and seemingly had unlimited potential. However soon some fundamental flaws were detected in the theory that limited scope of perceptron significantly. However, all these difficulties were overcome in time with addition of multiple layers in the architecture of the perceptron converting it into artificial neural networks and addition of nonlinear kernel functions like sigmoid. We will study the concept of perceptron and its evolution into modern artificial neural networks in this chapter. However, we will restrict the scope to small sized neural networks and will not delve into the deep networks. That will be studied later in separate chapter.

5.2 Perceptron

Geometrically a single layered perceptron with linear mapping represents a linear plane in n-dimensions. In n-dimensional space the input vector is represented as $(x_1, x_2, \ldots, x_n)$ or $\mathbf{x}$. The coefficients or weights in n-dimensions are represented as $(w_1, w_2, \ldots, w_n)$ or $\mathbf{w}$. The equation of perceptron in the n-dimensions is then written in vector form as

$$\mathbf{x}.\mathbf{w} = y \tag{5.1}$$

Figure 5.1 shows an example of an n-dimensional perceptron. This equation looks a lot like the linear regression equation that we studied in Chap. 4, which is essentially true, as perceptron represents a computational architecture to solve this problem.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_5

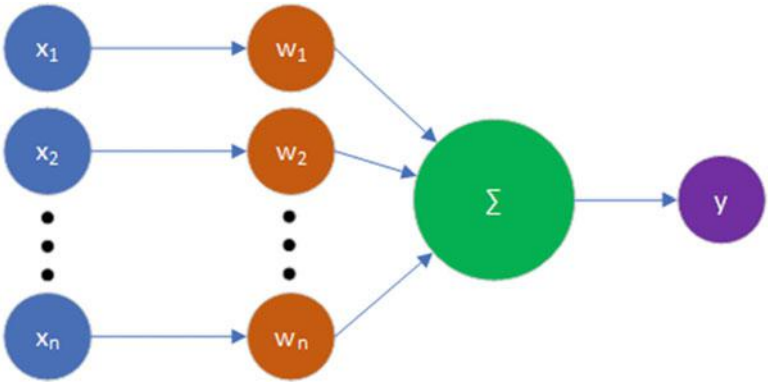


Fig. 5.1 Perceptron

5.3 Multilayered Perceptron or Artificial Neural Network

Multilayered perceptron (*MLP*) seems like a logical extension of the single layer architecture, where we use multiple layers instead of single. Figure 5.1 shows an illustration of a generic MLP with m layers. Let $n1$ be the number of nodes in layer 1, which is same as the input dimensionality. The subsequent layers have ni number of layers where $i = 2, \ldots, m$. The number of nodes in all the layers except the first one can have any arbitrary value as they are not dependent on the input or output dimensionality. Also, one other obvious difference between the single layer perceptron and MLP is the fully connectedness. Each of the internal nodes is now connected all the nodes in the subsequent layer. However, as long as we are using linear mapping as described above, single layer perceptron and multilayered perceptron are mathematically equivalent. In other words, having multiple layers does not really improve the capabilities of the model and it can be rigorously proved mathematically.

5.3.1 Feedforward Operation

The network shown in Fig. 5.2 also emphasizes another important aspect of MLP called as feedforward operation. The information that is entered from the input propagates through each layer towards the output. There is no feedback of the information from any layer backwards when the network is used for predicting the output in the form of regression or classification. This process closely resembles the operation of human brain.

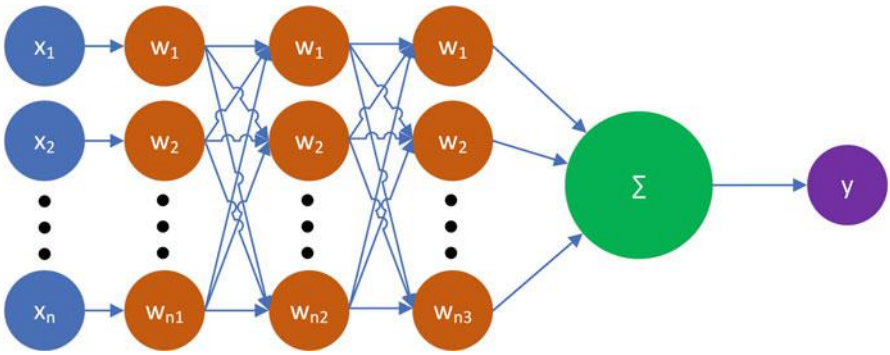


Fig. 5.2 Multilayered perceptron

5.3.2 Nonlinear MLP or Nonlinear ANN

The major improvement in the MLP architecture comes in the way of using nonlinear mapping. Instead of using simple dot product of the input and weights, a nonlinear function, called as activation function is used.

5.3.2.1 Activation Functions

Most simple activation function is a step function, also called as *sign* function as shown in Fig. 5.3. This activation function is suited for applications like binary classification. However, as this is not a continuous function is not suitable for most training algorithms as we will see in the next section.

The continuous version of step function is called as sigmoid function or logistic function as discussed in the previous chapter. Sometimes, a hyperbolic tan or *tanh* function is used, which has similar shape but its values range from $[-1, 1]$, instead of $[0 - 1]$ as in case of sigmoid function. Figure 5.4 shows the plot of *tanh* function.

5.3.3 Training MLP

During the training process, the weights of the network are learned from the labelled training data. Conceptually the process can be described as:

1. Present the input to the neural network.
2. All the weights of the network are assigned some default value.
3. The input is transformed into output by passing through each node or neuron in each layer.

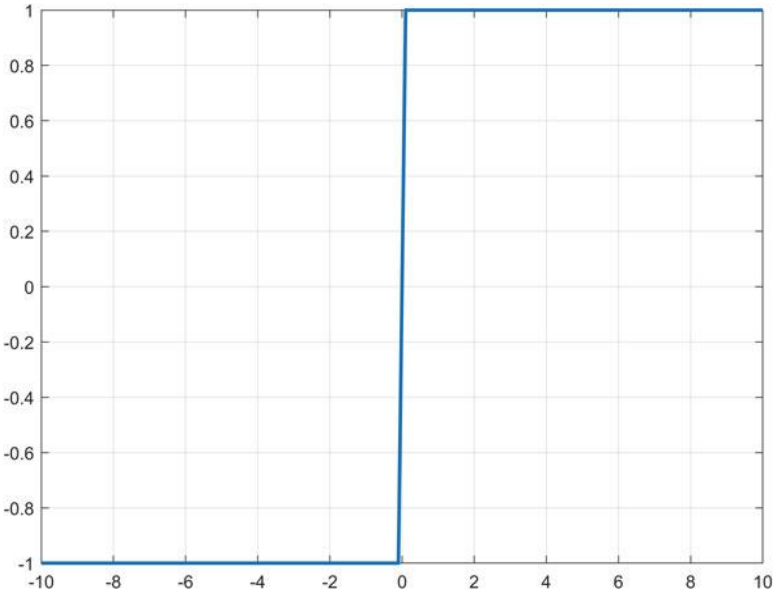


Fig. 5.3 Activation function *sign*

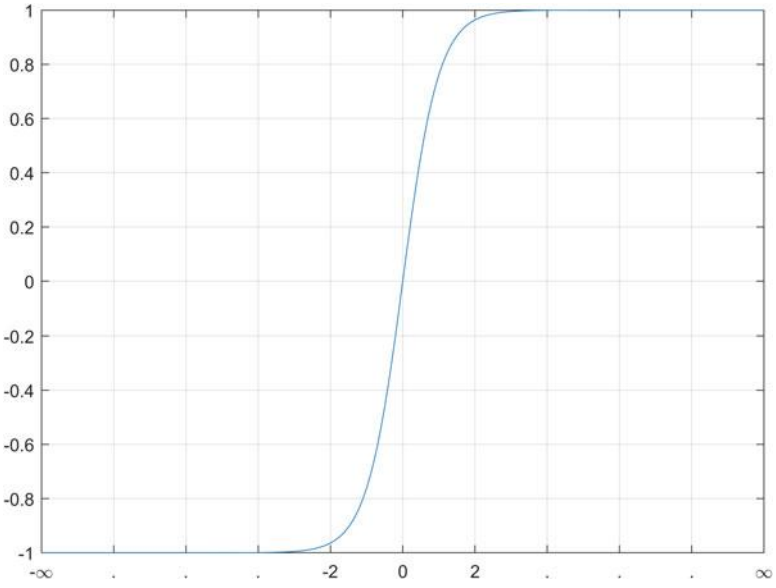


Fig. 5.4 Activation function *tanh*

4. The output generated by the network is then compared with the expected output or label.
5. The error between the prediction and label is then used to update the weights of each node.
6. The error is then propagated in backwards direction through every layer, to update the weights in each layer such that they minimize the error.

Cömert [45] summarizes various backpropagation training algorithms commonly used in the literature along with their relative performances. I am not going to go into the mathematical details of these algorithms here, as the theory quickly becomes quite advanced and can make the topic very hard to understand. Also, we will see in the implementation part of the book that with conceptual understanding of the training framework and open source libraries, one is sufficiently armed to apply these concepts on real problems.

Thus backpropagation algorithm for training and feedforward operation for prediction mark the two phases in the life of neural network. Backpropagation-based training needs to be done in two different methods.

1. Online or stochastic method
2. Batch method

5.3.3.1 Online or Stochastic Learning

In this method a single sample is sent as input to the network and based on the output error the weights are updated. The optimization method most commonly used to update the weights is called stochastic gradient descent or *SGD* method. The use of stochastic here implies that the samples are drawn randomly from the whole data set, rather than using them sequentially. The process can converge to desired accuracy level even before all the samples are used. It is important to understand that in stochastic learning process, single sample is used in each iteration and the learning path is more noisy. In some cases, rather than using a single sample, a mini-batch of samples is used. SGD is beneficial when the expected learning path can contain multiple local minima.

5.3.3.2 Batch Learning

In batch method the total data set is divided into a number of batches. Entire batch of samples is sent to the network before computing the error and updating the weights. After entire batch is processed, the weights are updated. Each batch process is called as one iteration. When all the samples are used once, it's considered as one epoch in the training process. Typically multiple epochs are used before the algorithm fully converges. As the batch learning uses a batch of samples in each iteration, it reduces the overall noise and learning path is cleaner. However, the process is lot

more computation heavy and needs more memory and computation resources. Batch learning is preferred when the learning path is expected to be relatively smooth.

5.3.4 *Hidden Layers*

The concept of hidden layers needs a little more explanation. As such they are not directly connected with inputs and outputs and there is no theory around how many such layers are optimal in given application. Each layer in MLP transforms the input to a new dimensional space. The hidden layers can have higher dimensionality than the actual input and thus they can transform the input into even higher dimensional space. Sometimes, if the distribution of input in its original space has some nonlinearities and is ill conditioned, the higher dimensional space can help improve the distribution and as a result improve the overall performance. These transformations also depend on the activation function used. Increasing dimensionality of hidden layer also makes the training process that much more complicated, and one needs to carefully trade between the added complexity and performance improvement. Also, how many such hidden layers should be used is another variable where there are no theoretical guidelines. Both these parameters are called as hyperparameters and one needs to do an open-ended exploration using a grid of possible values for them and then choose the combination that gives the best possible results within the constraints of the training resources.

5.4 Radial Basis Function Networks

Radial basis function networks *RBFN* or radial basis function neural networks *RBFNN* are a variation of the feedforward neural networks (we will call them as RBF networks to avoid confusion). Although their architecture as shown in Fig. 5.5 looks similar to MLP as described above, functionally they are more close to the support vector machines with radial kernel functions. The RBF networks are characterized by three layers, input layer, a single hidden layer, and output layer. The input and output layers are linear weighing functions, and the hidden layer has a radial basis activation function instead of sigmoid type activation function that is used in traditional MLP. The basis function is defined as

$$f_{RBF}(x) = e^{-\beta\|x-\mu\|^2)} \tag{5.2}$$

Above equation is defined for a scalar input, but without lack of generality it can be extended for multivariate inputs. μ is called as center and β represents the spread or variance of the radial basis function. It lies in the input space. Figure 5.6 shows the plot of the basis function. This plot is similar to Gaussian distribution.

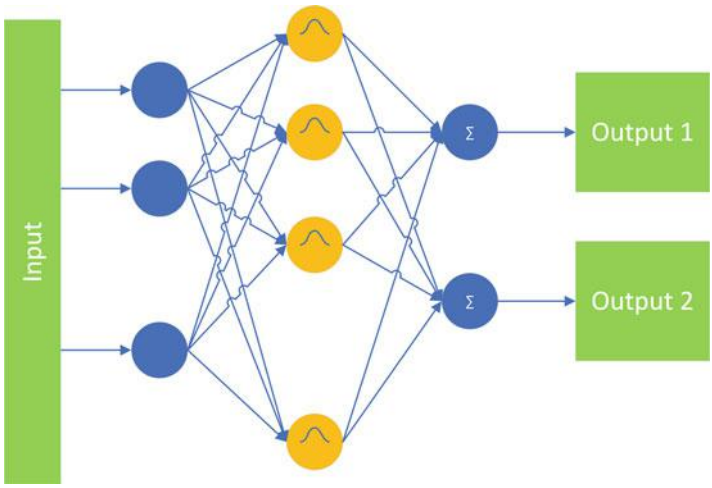


Fig. 5.5 Architecture of radial basis function neural network

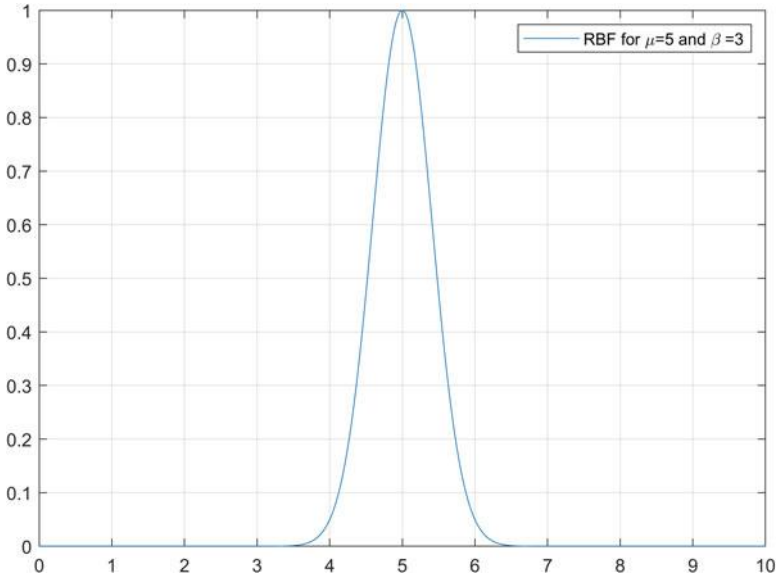


Fig. 5.6 Plot of radial basis function

5.4.1 Interpretation of RBF Networks

Aside from the mathematical definition, RBF networks have a very interesting interpretability that regular MLP does not have. Consider that the desired values of output form n number of clusters for the corresponding clusters in the input

space. Each node in the hidden layer can be thought of as a representative of each transformation from input cluster to output cluster. As can be seen from Fig. 5.6, the value of radial basis function reduces to 0 rather quickly as the distance between the input and the center of the radial basis function μ increases with respect to the spread β. Thus RBF network as a whole maps the input space to output space by linear combination of outputs generated by each hidden RBF node. It is important to choose these cluster centers carefully to make sure the input space is mapped uniformly and there are no gaps. The training algorithm is capable of finding the optimal centers, but number of clusters to use is a hyperparameter (in other words it needs to be tuned by exploration). If an input is presented to RBF network that is significantly different than the one used in training, the output of the network can be quite arbitrary. In other words the generalization performance of RBF networks in extrapolation situations is not good. However, if requirements for the RBF network are followed, it produces accurate predictions.

5.5 Overfitting and Regularization

Neural networks open up a feature-rich framework with practically unlimited scope to improve the performance for the given training data by increasing the complexity of the network. Complexity can be increased by manipulating various factors like

1. Increasing number of hidden layers
2. Increasing the nodes in hidden layers
3. Using complex activation functions
4. Increasing the training epochs

Such improvements in training performance with arbitrary increase in complexity typically lead to overfitting. Overfitting is a phenomenon where we try to model the training data so accurately that in essence we just memorize the training data rather than identifying the features and structure of it. Such memorization leads to significantly worse performance on unseen data. However determining the optimal threshold where the optimization should be stopped to keep the model generic enough is not trivial. Multiple approaches are proposed in the literature, e.g., *Optimal Brain Damage* [47] or *Optimal Brain Surgeon* [46].

5.5.1 L1 and L2 Regularization

Regularization approaches the problem using Lagrangian multiplier, where on top of minimizing the prediction error, we add another term in the optimization problem that restricts the complexity of the network with Lagrangian weighing factor λ.

Equations 5.3 and 5.4 show the updated cost function $C(x)$ use of L1 and L2 type of regularizations to reduce the overfitting.

$$C(x) = L(x) + \lambda \sum \|\mathbf{W}\| \tag{5.3}$$

$$C(x) = L(x) + \lambda \sum \|\mathbf{W}\|^2 \tag{5.4}$$

$L(x)$ is the loss function that is dependent on the error in prediction, while $\mathbf{W}$ stand for the vector of weights in the neural network. The L1 norm tries to minimize the sum of absolute values of the weights while the L2 norm tries to minimize the sum of squared values of the weights. Each type has some pros and cons. The L1 regularization requires less computation but is less sensitive to strong outliers, as well as it is prone to making all the weights zero. L2 regularization is overall a better metric and provides slow weight decay towards zero, but is more computation intensive.

5.5.2 Dropout Regularization

This is an interesting method and is only applicable to the case of neural networks, while the L1 and L2 regularization can be applied to any algorithm. In dropout regularization, the neural network is considered as an ensemble of neurons in sequence, and instead of using fully populated neural network, some neurons are randomly dropped from the path. The effect of each dropout on overall accuracy is considered, and after some iterations optimal set of neurons are selected in the final models. As this technique actually makes the model simpler rather than adding more complexity like L1 and L2 regularization techniques, this method is quite popular, specifically in case of more complex and deep neural networks that we will study in later chapters.

5.6 Conclusion

In this chapter, we studied the machine learning model based on simple neural network. We studied the concept of single perceptron and its evolution into full-fledged neural network. We also studied the variation of the neural networks using radial basis function kernels. In the end we studied the effect of overfitting and how to reduce it using regularization techniques.

Chapter 6
Decision Trees

6.1 Introduction

Decision trees represent conceptually and mathematically a different approach towards machine learning. The other approaches deal with the data that is strictly numerical and can increase and/or decrease monotonically. The equations that define these approaches cannot process a data that is not numerical, e.g., categorical or string type. However, the theory of decision trees does not rely on the data being numerical. While other approaches start by writing equations about the properties of data, decision trees start with drawing a tree-type structure such that at each node there is a decision to be made. At heart, decision trees are heuristic structures that can be built by a sequence of choices or comparisons that are made in certain order.

Let's take an example of classifying different species on earth. We can start with asking questions like: "Can they fly?". Based on the answer, we can split the whole gamut of species into two parts: ones that can fly and ones that can't. Then we go to the branch of species that cannot fly. We ask another question: "How many legs do they have?". Based on this answer we create multiple branches with answers like 2 legs, 4 legs, 6 legs, and so on. Similarly we can either ask same question for the flying species or we can ask a different question and continue splitting the species till we reach the leaf nodes such that there is only one single species there. This approach essentially summarizes the conceptual process of building a decision tree.

Although the above process describes the high level operation underlying the decision tree, the actual building process for a decision tree in a generalized setting is much more complex. The reason for complexity lies in answering the following: "how to choose which questions to ask and in which order?". One can always start asking random questions and ultimately still converge on the full solution, but when the data is large and high dimensional, this random or brute force approach can never be practical. There are multiple variations of the implementations of this concept that are widely used in the machine learning applications.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_6

6.2 Why Decision Trees?

Before going into the details of decision tree theory, let's understand why decision trees are so important. Here are the advantages of using decision tree algorithms for reference:

1. More human-like behavior.
2. Can work directly on non-numeric data, e.g., categorical.
3. Can work directly with missing data. As a result data cleaning step can be skipped.
4. Trained decision tree has high interpretability compared to abstract nature of trained models using other algorithms like neural networks, or SVM, etc.
5. Decision tree algorithms scale easily from linear data to nonlinear data without any change in core logic.
6. Decision trees can be used as non-parametric model, thus hyperparameter tuning becomes unnecessary.

6.2.1 Types of Decision Trees

Based on the application (classification or regression) there are some differences in how the trees are built, and consequently they are called classification decision trees and regression decision trees. However, we are going to treat the machine learning techniques based on applications in the next part of the book and in this chapter, we will focus on the fundamental concepts underlying the decision trees that are common between both.

6.3 Algorithms for Building Decision Trees

Most commonly used algorithms for building decision trees are:

- CART or Classification and Regression Tree
- ID3 or Iterative Dichotomiser
- CHAID or Chi-Squared Automatic Interaction Detector

CART or classification and regression tree is a generic term used for describing the process of building decision trees as described by Breiman–Friedman [39]. ID3 is a variation of CART methodology with slightly different use of optimization method. CHAID uses a significantly different procedure and we will study it separately.

The development of classification trees is slightly different but follows similar arguments as a regression tree. Let's consider a two-dimensional space defined by

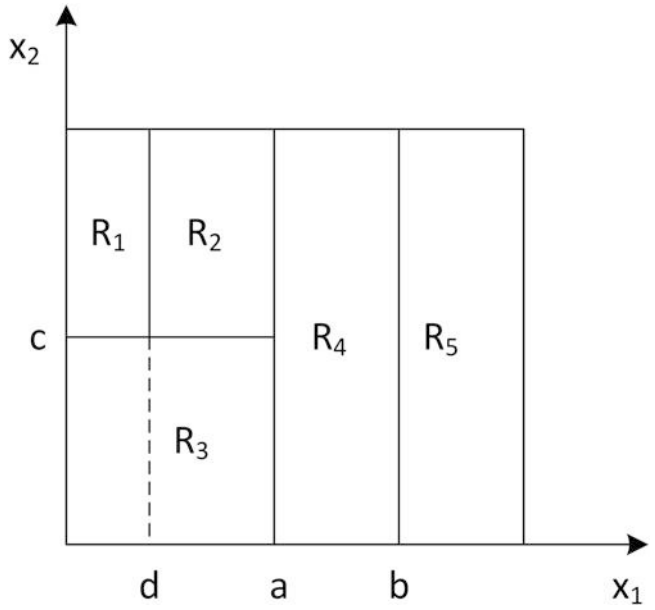


Fig. 6.1 Rectangular regions created by decision tree

axes (x_1, x_2). The space is divided into 5 regions $(R_1, R_2, R_3, R_4, R_5)$ as shown in figure, using a set of rules as defined in Fig. 6.1.

6.4 Regression Tree

Regression trees are the trees that are designed to predict the value of a function at given coordinates. Let us consider a set of N-dimensional input data $\{\mathbf{x_i}, i = 1, \ldots, p$ and $\mathbf{x_i} \subset \Re^n\}$. The corresponding outputs are $\{y_i, i = 1, \ldots, p$ and $y_i \subset \Re\}$. It is required that in case of regression trees the input and output data is numerical and not categorical. Once given this training data, it is the job of the algorithm to build the set of rules. How many rules should be used, what dimensions to use, when to terminate the tree are all the parameters that the algorithm needs to optimize based on the desired error rate.

Based on the example shown in Figs. 6.1 and 6.2, let the classes be regions R_1 to R_5 and the input data is two dimensional. In such case, the desired response of the decision tree is defined as

$$t(x) = r_k \forall x_i \in R_k \tag{6.1}$$

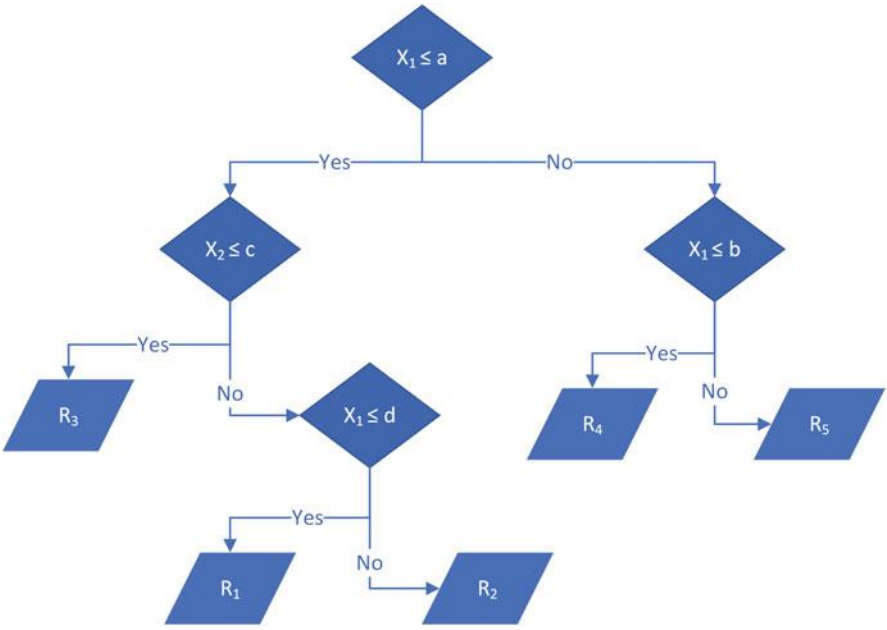


Fig. 6.2 Hierarchical rules defining the decision tree

where $r_k \in \Re$ is a constant value of output in region R_k. If we define the optimization problem as minimizing the mean square error,

$$\sum_{i=1}^{i=p} (y_i - t(x_i)^2) \tag{6.2}$$

then simple calculation would show that the estimate for r_k is given by

$$r_k = \text{ave}(y_i | x_i \in R_k) \tag{6.3}$$

Solving the problem to find the globally optimum regions to minimize the mean square error is an NP-hard problem and cannot be solved in general in finite time. Hence greedy methods resulting in local minimum are employed. Such greedy methods typically result in a large tree that overfits the data. Let us denote such large tree as T_0. Then the algorithm must apply a pruning technique to reduce the tree size to find the optimal tradeoff that captures the most of the structure in the data without overfitting it. This is achieved by using squared error node impurity measure optimization as described in [39].

6.5 Classification Tree

In case of classification, the output is not a continuous numerical value, but a discreet class label. The development of the large tree follows the same steps as described in the regression tree subsection, but the pruning methods need to be updated as the squared error method is not suitable for classification. Three different types of measures are popular in the literature:

- Misclassification error
- Gini index
- Cross-entropy or deviance

Let there be "k" classes and "n" nodes. Let the frequency of class (m) predictions at each node (i) be denoted as f_{mi}. The fraction of the classes predicted as m at node i be denoted as p_{mi}. Let the majority class at node m be c_m. Hence the fraction of classes c_m at node m would be p_{mc_m}.

6.6 Decision Metrics

Let's define the metrics used for making the decision at each node. Differences in the metric definition separate the different decision tree algorithms.

6.6.1 Misclassification Error

Based on the variables defined above the misclassification rate is defined as $1 - p_{mc_m}$. As can be seen from the figure this rate is not a continuous function and hence cannot be differentiated. However, this is one of the most intuitive formulations and hence is fairly popular.

6.6.2 Gini Index

Gini index is the measure of choice in CART. Concept of the Gini index can be summarized as the probability of misclassification of a randomly selected input sample if it was labelled based on the distribution of the classes in the given node. Mathematically it is defined as

$$\mathcal{G} = \sum_{m=1}^{m=k} p_{mi}(1 - p_{mi}) \tag{6.4}$$

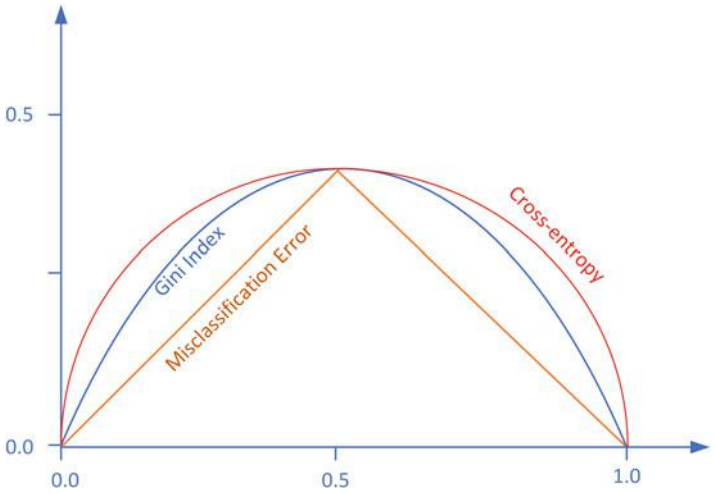


Fig. 6.3 The plot of decision metrics for a case of 2 class problem. X-axis shows the proportion in class 1. Curves are scaled to fit, without loss of generality

As the plot in Fig. 6.3 shows, this is a smooth function of the proportion and is continuously differentiable and can be safely used in optimization.

6.6.3 Cross-Entropy or Deviance

Cross-entropy is an information theoretic metric defined as

$$\mathcal{E} = -\sum_{m=1}^{m=k} p_{mi} \log(p_{mi}). \tag{6.5}$$

This definition resembles classical entropy of a single random variable. However, as the random variable here is already a combination of the class prediction and nodes of the tree, it is called as cross-entropy. ID3 models use cross-entropy as the measure of choice. As the plot in figure shows, this is a smooth function of the proportion and is continuously differentiable and can be safely used in optimization.

6.7 CHAID

Chi-square automatic interaction detector or *CHAID* is a decision tree technique that derives its origin in statistical chi-square test for goodness of fit. It was first published by G. V. Kass in 1980, but some parts of the technique were already in

use in 1950s. This test uses the chi-square distribution to compare a sample with a population and predict at desired statistical significance whether the sample belongs to the population. CHAID technique uses this theory to build a decision tree. Due to the use of chi-square technique in building decision tree, this method is quite different compared to any other types of decision trees discussed so far. Following subsection discusses the details of the algorithm briefly.

6.7.1 CHAID Algorithm

The first task in building the CHAID tree is to find the most dependent variable. This is in a way directly related to what is the final application of the tree. The algorithm works best if a single desired variable can be identified. Once such variable is identified, it is called as root node. Then the algorithm tries to split the node into two or more nodes, called as initial or parent nodes. All the subsequent nodes are called as child nodes, till we reach the final set of nodes that are not split any further. These nodes are called as terminal nodes. Splitting at each node is entirely based on statistical dependence as dictated by chi-square distribution in case of categorical data and by F-test in case of continuous data. As each split is based on dependency of variables, unlike a more complex expression like Gini impurity or cross-entropy in case of CART or ID3-based trees, the tree structure developed using CHAID is more interpretable and human readable in most cases.

6.8 Training Decision Tree

We are not going into full mathematical details of building a decision tree using CART or ID3, but the following steps will explain the methodology in sufficient details and clarity.

6.8.1 Steps

1. Start with the training data.
2. Choose the metric of choice (Gini index or cross-entropy).
3. Choose the root node, such that it splits the data with optimal values of metrics into two branches.
4. Split the data into two parts by applying the decision rule of root node.
5. Repeat the steps 3 and 4 for each branch.
6. Continue the splitting process till leaf nodes are reached in all the branches with predefined stop rule.

6.9 Ensemble Decision Trees

In previous sections we learned ways to develop a single decision tree based on different techniques. In many situations such trees work very well, but there are ways to extract more performance out of the similar architecture if we create multiple such trees and aggregate them. Such techniques are called as ensemble methods and they typically deliver superior performance at the cost of computation and algorithmic complexity. In ensemble methods, a single decision tree is called as a single learner or weak learner and the ensemble methods deal with a group of such learners.

There are various approaches proposed in the literature that can successfully combine multiple weak learners to create a strong overall model.[1] Each weak learner in the ensemble of learners captures certain aspects of the information contained in the data that is used to train it. The job of ensemble tree is to optimally unite the weak learners to have better overall metrics. Primary advantage of ensemble methods is reduction in overfitting.

There are three main types of ensembles:

1. Bagging
2. Random forest
3. Boosting

6.10 Bagging Ensemble Trees

The term bagging finds it origins in *Bootstrap Aggregation*. Coincidentally, literal meaning of bagging, which means putting multiple decision trees in a *bag* is not too far from the way the bagging techniques work. Bagging technique can be described using following steps:

1. Split the total training data into a predetermined number of sets with random sampling with replacement. The term *With replacement* means that same sample can appear in multiple sets. Each sample is called as *Bootstrap* sample.
2. Train decision tree using CART or ID3 method using each of the data sets.
3. Each learned tree is called as a weak learner.
4. Aggregate all the weak learners by averaging the outputs of individual learners for the case of regression and aggregate all the individual weak learners by voting

[1] The words *weak* and *strong* have a different meaning in this context. A weak learner is a decision tree that is trained using only fraction of the total data and is not capable or even expected of giving metrics that are close to the desired ones. Theoretical definition of a weak learner is one whose performance is only slightly better than pure random chance. A strong learner is a single decision tree uses all the data and is capable of producing reasonably good metrics. In ensemble methods individual tree is always a weak learner as it is not exposed to the full data set.

for the case of classification. The aggregation steps involve optimization, such that prediction error is minimized.
5. The output of the aggregate or ensemble of the weak learners is considered as the final output.

The steps described above seem quite straightforward and does not really involve any complex mathematics or calculus. However, the method is quite effective. If the data has some outliers,[2] a single decision tree can get affected by it more than an ensemble can be. This is one of the inherent advantages of bagging methods.

6.11 Random Forest Trees

Bagging process described above improves the resilience of the decision trees with respect to outliers. Random forest methods go one step forward to make the ensemble even more resilient in case of varying feature importances. Even after using a carefully crafted feature space, not all features are equally influential on the outcome. Also certain features can have some interdependencies that can affect their influence on the outcome in counterproductive manner. Random forest tree architecture improves model performance in such situations over previously discussed methods by partitioning feature space as well as data for individual weak learner. Thus each weak learner sees only fraction of samples and fraction of features. The features are also sampled randomly with replacement [48], as data is sampled with replacement in bagging methods. The process is also called as random subspace method, as each weak learner works in a subspace of features. In practice, this sampling improves the diversity among the individual trees and overall makes the model more robust and resilient to noisy data. The original algorithm proposed by *Tin Ho* was then extended by Breiman [49] to merge the multiple existing approaches in the literature to what is now commonly known as random forest method.

6.11.1 Decision Jungles

Recently a modification to the method of random forests was proposed in the form of *Decision Jungles* [62]. One of the drawbacks of random forests is that they can grow

[2]Outliers represent an important concept in the theory of machine learning. Although, its meaning is obvious, its impact on learning is not quite trivial. An outlier is a sample in training data that does not represent the generic trends in the data. Also, from mathematical standpoint, the distance of an outlier from other samples in the data is typically large. Such large distances can throw a machine learning model significantly away from the desired behavior. In other words, a small set of outliers can affect the learning of a machine learning model adversely and can reduce the metrics significantly. It is thus an important property of a machine learning model to be resilient of a reasonable number of outliers.

exponentially with data size and if the compute platform is limited by memory, the depth of the trees needs to be restricted. This can result in suboptimal performance. Decision jungles propose to improve on this by representing each weak learner in random forest method by a directed acyclic graph DAG instead of open-ended tree. The DAG has capability to fuse some of the nodes thereby creating multiple paths to a leaf from root node. As a result decision jungles can represent the same logic as random forest trees, but in a significantly compact manner.

6.12 Boosted Ensemble Trees

Fundamental difference between boosted and bagging (or random forest for that matter) is the sequential training of the trees against the parallel training. In bagging or random forest methods all the individual weak learners are generated using random sampling and random subspaces. As all the individual weak learners are independent of each other, they all can be trained in parallel. Only after they are completely trained their results are aggregated. Boosting technique employs a very different approach, where first tree is trained based on a random sample of data. However, the data used by the second tree depends on the outcome of training of first tree. The second tree is used to focus on the specific samples where first decision tree is not performing well. Thus training of second tree is dependent on the training of first tree and they cannot be trained in parallel. The training continues in this fashion to third tree and fourth and so on. Due to unavailability of parallel computation, the training of boosted trees is significantly slower than training tress using bagging and random forest. Once all the trees are trained then the output of all individual trees is combined with necessary weights to generate final output. In spite of the computational disadvantage exhibited by the boosted trees, they are often preferred over other techniques due to their superior performance in most cases.

6.12.1 AdaBoost

AdaBoost was one of the first boosting algorithms proposed by Freund and Schapire [51]. The algorithm was primarily developed for the case of binary classification and it was quite effective in improving the performance of a decision tree in a systematic iterative manner. The algorithm was then extended to support multi-class classification as well as regression.

6.12.2 Gradient Boosting

Breiman proposed an algorithm called as *ARCing* [50] or *Adaptive Reweighting and Combining*. This algorithm marked the next step in improving the capabilities

of boosting type methods using statistical framework. Gradient boosting is a generalization of AdaBoost algorithm using statistical framework developed by Breiman and Friedman [39]. In gradient boosted trees, the boosting problem is stated as numerical optimization problem with objective to minimize the error by sequentially adding weak learners using gradient descent algorithm. Gradient descent being a greedy method, gradient boosting algorithm is susceptible to overfitting the training data. Hence regularization techniques are always used with gradient boosting to limit the overfitting.

6.13 Conclusion

In this chapter we studied the concept of decision trees. These methods are extremely important and useful in many applications encountered in practice. They are directly motivated by the hierarchical decision making process very similar to the human behavior in tackling real life problems and hence are more intuitive than the other methods. Also, the results obtained using decision trees are easier to interpret and these insights can be used to determine action to be taken after knowing the results. We also looked at ensemble methods that use aggregate of multiple decision trees to optimize the overall performance and make the models more robust and generic.

Chapter 7
Support Vector Machines

7.1 Introduction

Theory of support vector machines or *SVMs* is typically attributed to Vladimir Vapnik. He was working in the field of optimal pattern recognition using statistical methods in the early 1960s. His paper with Lerner [52] on generalized portrait algorithm marks the beginning of the support vector machines. This method was primarily designed to solve the problem of binary classification using construction of optimal hyperplane that separates the two classes with maximum separation.

7.2 Motivation and Scope

Original SVM algorithm is developed for binary classification. Figure shows the concept of SVM in case of linear application. The algorithm of SVM tries to separate the two classes with maximal separation using minimum number of data points, also called as support vectors, as shown in Figs. 7.1 and 7.2. Figure 7.1 shows the case of linearly separable classes and the result is trivial. The solid line represents the hyperplane that optimally separates the two classes. The dotted lines represent the boundaries of the classes as defined by the support vectors. The class separating hyperplane tries to maximize the distance between the class boundaries. However, as can be seen in Fig. 7.2, where the classes are not entirely linearly separable, the algorithm still finds the optimal support vectors. Once the support vectors are identified, the classification does not need the rest of the samples for predicting the class. The beauty of the algorithm lies in the drastic reduction in the number of support vectors compared to number of total training samples.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_7

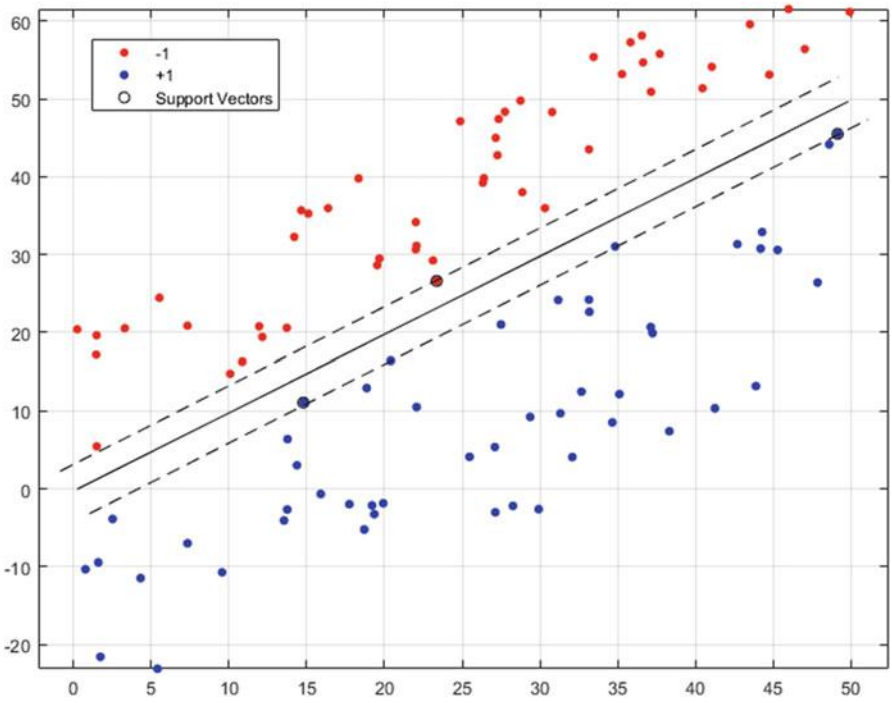


Fig. 7.1 Linear binary SVM applied on separable data

7.2.1 Extension to Multi-Class Classification

As per the conceptual setup of SVM, it is not directly extensible for solving the problem of multi-class classification. However there are few approaches commonly used to extend the framework for such case. One approach is to use SVM as binary classifier to separate each pair of classes and then apply some heuristic to predict the class for each sample. This is extremely time consuming method and not the preferred one. For example, in case of 3-class problem, one has to train the SVM for separating classes 1–2, 1–3, and 2–3, thereby training 3 separate SVMs. The complexity will increase in polynomial rate with more classes. In other approach binary SVM is used to separate each class from the rest of the classes. With this approach, for 3-class problem, one still has to train 3 SVMs, 1-(2,3), 2-(1,3), and 3-(1,2). However, with further increase in the number of classes, the complexity increases linearly.

7.2.2 Extension for Nonlinear Case

The case of nonlinear separation can be solved by using suitable kernels. The original data can be transformed into arbitrarily higher dimensional vectors using

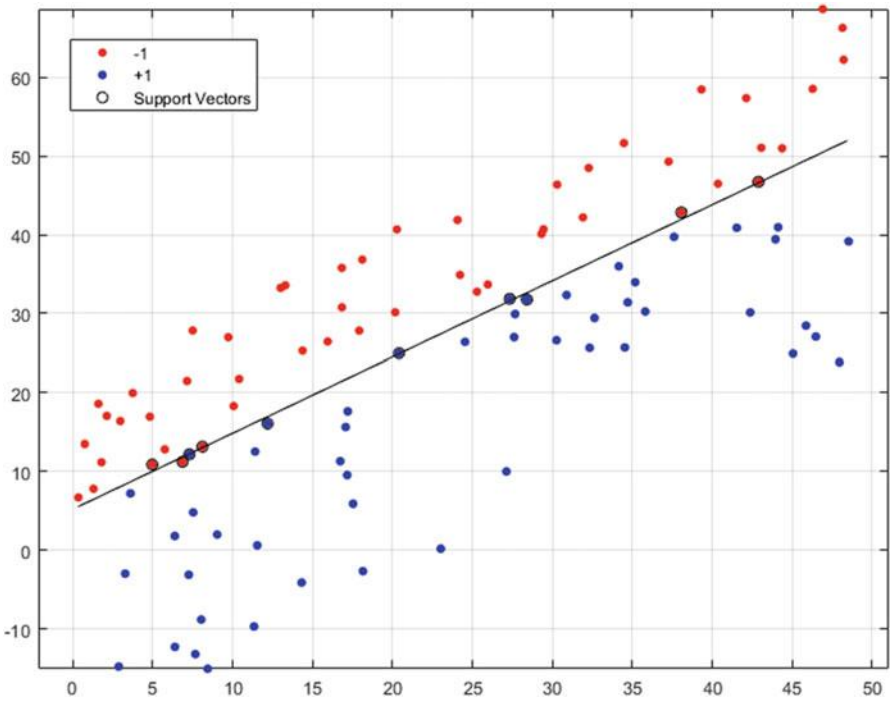


Fig. 7.2 Linear binary SVM applied on non-separable data

suitable kernel functions. After the transformation, the equations of linear SVM are applicable as is leading to optimal classification.

7.3 Theory of SVM

In order to understand how the SVMs are trained it is important to understand the theory behind SVM algorithm. This can get highly mathematical and complex, however, I am going to try to avoid the gory details of derivations. Reader is encouraged to read [37] for detailed theoretical overview. I will state the assumptions on which the derivations are based and then move to final equations that can be used to train the SVM without losing the essence of SVM.

Let us consider a binary classification problem with n-dimensional training data set consisting of p pairs $(\mathbf{x}_i, y_i)$, such that $\mathbf{x}_i \in \Re^n$ and $y_i \in \{-1, +1\}$. Let the equation of the hyperplane that separates the two classes with maximal separation be given as

$$(\mathbf{w} \cdot \mathbf{x}) - w_0 = 0 \tag{7.1}$$

Here $\mathbf{w} \in \Re^n$ same as $\mathbf{x}$. For the samples belonging to each class we can write

$$(\mathbf{w} \cdot \mathbf{x}_i) - w_0 \begin{cases} > 1, & \text{if } y_i = 1 \\ \leq 1, & \text{if } y_i = -1 \end{cases} \tag{7.2}$$

The two equations can be combined into a single equation as

$$y_i[(\mathbf{w} \cdot \mathbf{x}) - w_0] \geq 1, \ i = 1, \ldots, p \tag{7.3}$$

The above equation can be solved to get multiple solutions for weight vector. In order to get the solution that also minimizes the weight vector we impose a constraint to minimize $\Phi(\mathbf{w})$. Where $\Phi(\mathbf{w})$ is given as

$$\Phi(\mathbf{w}) = \|\mathbf{w}'\|^2 \tag{7.4}$$

where $\mathbf{w}'$ is an $(n+1)$ dimensional vector that is combination of $\mathbf{w}$ and w_0.

Thus we have arrived at a point where we can define the optimization problem that needs to be solved. To obtain the precise equation of the hyperplane, we need to minimize the functional $\Phi(\mathbf{w}')$ with constraints defined in Eq. 7.3. Typically such problems are solved using the technique of *Lagrangian*, that combines the functional that is to be minimized and the constraints into a single Lagrange functional that needs to be minimized,

$$\min_{\mathbf{w}} \left\{ \left(\frac{1}{n} \sum_{i=1}^{n} 1 - y_i[(\mathbf{w} \cdot \mathbf{x}) - w_0] \right) + \lambda \Phi(\mathbf{w}) \right\} \tag{7.5}$$

With some manipulations, the Lagrangian in Eq. 7.5 reduces to a subset that contains only a very small number of training samples that are called as support vectors. As can be seen from Fig. 7.1, these support vectors are the vectors that represent the boundary of each class. After some more mathematical trickery performed using well-known Kühn-Tucker conditions, one arrives at convex quadratic optimization problem, which is relatively straightforward to solve. The equation to compute the optimal weight vector $\hat{\mathbf{w}}$ can then be given in terms of Lagrange multipliers $\alpha_i \geq 0$ as

$$\hat{\mathbf{w}} = \sum_{\text{support vectors only}} y_i \alpha_i \mathbf{x}_i \tag{7.6}$$

Once these parameters are computed as part of training process, the classifying function can be given as (for a given sample $\mathbf{x}$),

$$f_c(x) = \text{sign} \left(\sum_{\text{support vectors only}} y_i \alpha_i (\mathbf{x}_i \cdot \mathbf{x}) - \alpha_0 \right) \tag{7.7}$$

7.4 Separability and Margins

The SVM algorithm described above is designed to separate the classes that are in fact completely separable. In other words, when the separating hyperplane is constructed, between the two classes, entirety of the one class lies on one side of the hyperplane and entirety of other class lies on the opposite side of the hyperplane with 100% accuracy in separation. The margins defined in Eq. 7.2 are called as hard margins that impose complete separability between the classes. However, in practice such cases are seldom found. In order to account for the cases that are not entirely separable, *soft margins* were introduced. In order to understand the soft margins, let us write Eq. 7.3 in slightly different manner as

$$0 \geq 1 - y_i[(\mathbf{w}.\mathbf{x}) - w_0], i = 1, \ldots, p \tag{7.8}$$

7.4.1 Regularization and Soft Margin SVM

For all the cases when the samples are not separable, the above inequality will not be satisfied. To accommodate such cases the optimization problem is re-formulated using regularization techniques. New Lagrangian to be minimized is given as

$$\min_{\mathbf{w}} \left\{ \left(\frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i[(\mathbf{w} \cdot \mathbf{x}) - w_0]) \right) + \lambda \Phi(\mathbf{w}) \right\} \tag{7.9}$$

Here, with the max function we are essentially ignoring the cases when there is error in the classification.

7.4.2 Use of Slack Variables

Another way to accommodate for case of non-separable data is use of slack variables denoted as ξ_i. With use of slack variables the cost functional is updated as

$$\Phi(\mathbf{w}) = \|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i \tag{7.10}$$

where $\xi_i \geq 0, i = 1, \ldots, m$. Now the optimization operation also needs to find the values of all the slack variables. This approach is also called as *C-SVM*.

7.5 Nonlinearity and Use of Kernels

Use of kernels is one of the ground breaking discoveries in the field of machine learning. With the help of this method, one can elegantly transform a nonlinear problem into a linear problem. These kernel functions are different from the *link functions* that we discussed in Chap. 4. In order to understand the use of kernels in case of support vector machines, let's look at Eq. 7.7, specifically the term $(\mathbf{x}.\mathbf{x})$. Here we are taking a dot product of input vector with itself and as a result generating a real number. Use of kernel function[1] states that we can replace the dot product operation with a function that accepts two parameters, (in this case both will be input vector) and outputs a real valued number. Mathematically, this kernel function is written as

$$k : (\boldsymbol{\chi} \cdot \boldsymbol{\chi}) \rightarrow \Re \tag{7.11}$$

Although, this representation allows for using any arbitrary kernel function to transform the original data, in order to have deterministic answers in all the situations, the kernel function needs to be *positive definite* function. A positive definite function needs to satisfy a property defined by Mercer's theorem. Mercer's theorem states that for all finite sequence of points $x_1, x_2, \ldots, x_n$ and all real numbers $c_1, c_2, \ldots, c_n$ the kernel function should satisfy,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} k(x_i, x_j) c_i c_j \geq 0 \tag{7.12}$$

With appropriate choice of such positive definite kernel function, one can map the n-dimensional input to a real valued output in a deterministic linear manner. If we know certain nonlinearity trends in the data, we can build a custom kernel function that will transform the problem suitable to be solved by the linear support vector machine. Some of the commonly used kernel functions are:

7.5.1 Radial Basis Function

Radial basis function kernel with variance σ is given as

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{7.13}$$

[1] Sometimes this is also called as kernel trick, although this is far more than a simple trick. A function needs to satisfy certain properties in order to be able called as kernel function. For more details on kernel functions refer to [37].

This representation of SVM resembles closely with radial basis function neural networks that we learnt in Chap. 5. In some cases, use of squared distance between the two inputs can be lead to vanishingly low values. In such cases a variation of above function, called as Laplacian radial basis function is used. It is defined as

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|}{\sigma}\right) \tag{7.14}$$

7.5.2 Polynomial

For polynomial with degree d, the kernel function is given as

$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d \tag{7.15}$$

7.5.3 Sigmoid

Sigmoid kernel can also be used that resembles a traditional neural network. It is defined as

$$k(x_i, x_j) = \tanh(Ax_i' x_j + B) \tag{7.16}$$

7.6 Risk Minimization

Methods based on risk minimization, sometimes called as structural risk minimization [64], essentially aim at learning to optimize the given system with constraints on parameters imposed by regularization as well as problem definition itself. Support vector machines solve this problem of risk minimization in elegant fashion. These methods strike the balance between performance optimization and reduction in overfitting in programmatic manner. Vapnik further extended the theory of structural risk minimization for the cases of generative models using vicinal risk minimization [63]. These methods can be applied to the cases that do not fit in the traditional SVM architecture, such as problems with missing data, or unlabelled data.

7.7 Conclusion

In this chapter we studied an important pillar of machine learning theory, the support vector machines. SVM represents a mathematically elegant architecture to build an optimal classification or regression model. The training process is bit complex and needs tuning of some hyperparameters, but once properly tuned SVM models tend to provide very high accuracy and generalization capabilities.

Chapter 8
Probabilistic Models

8.1 Introduction

Most algorithms studied thus far are based on algebraic, graphical, and/or calculus based methods. In this chapter we are going to focus on the probabilistic methods. Probabilistic methods try to assign some form of uncertainty to the unknown variables and some form of belief probability to known variables and try to find the unknown values using the extensive library of probabilistic models. The probabilistic models are mainly classified into two types:

1. Generative
2. Discriminative

The generative model takes a more holistic approach towards understanding the data compared to discriminative models. Commonly, the difference between the two types is given in terms of the probabilities that they deal with. If we have an observable input X and observable output Y, then the generative models try to model the joint probability $P(X; Y)$, while the discriminative models try to model the conditional probability $P(Y|X)$. Most non-probabilistic approaches discussed so far also belong to the class of discriminative models. Although this definition of separation between the two approaches can be quite vague and confusing at times. Hence, we will try to define the two more intuitively. Before going into the definition we need to add few more concepts. Let there be a hidden entity called state S along with the input and output. The input actually makes some changes into the state of the system, and that change along with the input dictates the output. Now, let's define the discriminative models as the models that try to predict the changes in the output based on only changes in the input. The generative models are the models that try to model the changes in the output based on changes in input as well as the changes in the state. This inclusion and modeling of the state gives a deeper insight into the systemic aspects and the generative models are typically harder to build and need more information and assumptions to start with. However, there are some

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_8

inherent advantages that come with this added complexity as we will see in later sections of this chapter./AQPlease check the sentence "Although this definition..." for clarity.

The probabilistic approaches (discriminative as well as generative) are also sliced based on two universities of thought groups:

1. Maximum likelihood estimation
2. Bayesian approach

8.2 Discriminative Models

We will first discuss the distinction between these two classes from the perspective of discriminative models and then we will turn to generative models.

8.2.1 Maximum Likelihood Estimation

The maximum likelihood estimation or *MLE* approach deals with the problems at the face value and parameterizes the information into variables. The values of the variables that maximize the probability of the observed variables lead to the solution of the problem. Let us define the problem using formal notations. Let there be a function $f(\mathbf{x}; \theta)$ that produces the observed output y. $x \in \Re^n$ represent the input on which we don't have any control over and $\theta \in \Theta$ represent a parameter vector that can be single or multidimensional. The MLE method defines a likelihood function denoted as $L(y|\theta)$. Typically the likelihood function is the joint probability of the parameters and observed variables as $L(y|\theta) = P(y; \theta)$. The objective is to find the optimal values for θ that maximizes the likelihood function as given by

$$\theta^{\text{MLE}} = \arg\max_{\theta \in \Theta} \{L(y|\theta)\} \tag{8.1}$$

or,

$$\theta^{\text{MLE}} = \arg\max_{\theta \in \Theta} \{P(y; \theta)\} \tag{8.2}$$

This is a purely frequentist approach that is only data dependent.

8.2.2 Bayesian Approach

Bayesian approach looks at the problem in a different manner. All the unknowns are modelled as random variables with known prior probability distributions. Let us denote the conditional prior probability of observing the output y for parameter

vector θ as $P(y|\theta)$. The marginal probabilities of these variables are denoted as $P(y)$ and $P(\theta)$. The joint probability of the variables can be written in terms of conditional and marginal probabilities as

$$P(y;\theta) = P(y|\theta) \cdot P(\theta) \tag{8.3}$$

The same joint probability can also be given as

$$P(y;\theta) = P(\theta|y) \cdot P(y) \tag{8.4}$$

Here the probability $P(\theta/y)$ is called as posterior probability. Combining Eqs. 8.3 and 8.4

$$P(\theta|y) \cdot P(y) = P(y|\theta) \cdot P(\theta) \tag{8.5}$$

rearranging the terms we get

$$P(\theta|y) = \frac{P(y|\theta) \cdot P(\theta)}{P(y)} \tag{8.6}$$

Equation 8.6 is called as Bayes' theorem. This theorem gives relationship between the posteriory probability and priori probability in a simple and elegant manner. This equation is the foundation of the entire bayesian framework. Each term in the above equation is given a name, $P(\theta)$ is called as *prior*, $P(y|\theta)$ is called as *likelihood*, $P(y)$ is called as *evidence*, and $P(\theta|y)$ is called as *posterior*. Thus in this worm the Bayes' theorem is stated as/AQ Please check the sentence "Thus in this worm..." for clarity.

$$\text{(posterior)} = \frac{\text{(prior)} \cdot \text{(likelihood)}}{\text{(evidence)}} \tag{8.7}$$

The Bayes' estimate is based on maximizing the posterior. Hence, the optimization problem based on Bayes' theorem can now be stated as

$$\theta^{\text{Bayes}} = \arg\max_{\theta \in \Theta} \{P(\theta|y\} \tag{8.8}$$

expanding the term

$$\theta^{\text{Bayes}} = \arg\max_{\theta \in \Theta} \left\{ \frac{P(y|\theta) \cdot P(\theta)}{P(y)} \right\} \tag{8.9}$$

Comparing this equation with 8.2, we can see that Bayesian approach adds more information in the form of priori probability. Sometimes, this information is available, and then Bayesian approach clearly becomes the preferred, but in cases

when this information is not explicitly available, one can still assume certain default distribution and proceed.

8.2.3 Comparison of MLE and Bayesian Approach

These formulations are relatively abstract and in general can be quite hard to comprehend. In order to understand them to the full extent let us consider a simple numerical example. Let there be an experiment to toss a coin for 5 times. Let's say the two possible outcomes of each toss are H, T, or *Head* or *Tail*. The outcome of our experiment is H, H, T, H, H. The objective is to find the outcome of the 6th toss. Let's work out this problem using MLE and Bayes' approach.

8.2.3.1 Solution Using MLE

The likelihood function is defined as, $L(y|\theta) = P(y;\theta)$, where y denotes the outcome of the trial and θ denotes the property of the coin in the form of probability of getting given outcome. Let probability of getting a *Head* be h and probability of getting a *Tail* will be $1-h$. Now, outcome of each toss is independent of the outcome of the other tosses. Hence the total likelihood of the experiment can be given as

$$P(y;\theta) = P(y = H|\theta = h)^4 \cdot P(y = T|\theta = (1-h)) \tag{8.10}$$

Now, let us solve this equation,

$$\begin{aligned} P(y;\theta) &= h \cdot h \cdot (1-h) \cdot h \cdot h \\ P(y;\theta) &= h^4 - h^5 \end{aligned}$$

In order to maximize the likelihood, we need to use the fundamental principle from differential calculus, that at any maximum or minimum of a continuous function the first order derivative is 0. In order to maximize the likelihood, we will differentiate above equation with respect to h and equate it to 0. Solving the equation (assuming $h \neq 0$) we get,

$$\begin{aligned} \frac{\partial}{\partial h} P(y;\theta) &= 0 \\ \frac{\partial}{\partial h}(h^4 - h^5) &= 0 \\ 4 \cdot h^3 - 5 \cdot h^4 &= 0 \\ 4 \cdot h^3 &= 5 \cdot h^4 \end{aligned}$$

$$4 = 5 \cdot h$$
$$h = \frac{4}{5}$$

This probability of getting *Head* in the next toss would be $\frac{4}{5}$.

8.2.3.2 Solution Using Bayes's Approach

Writing the posterior as per Bayes' theorem,

$$P(\theta|y) = \frac{P(y|\theta) \cdot P(\theta)}{P(y)} \tag{8.11}$$

Comparing this equation with the Eq. 8.10, we can see that the likelihood function is same as the term $P(y|\theta)$ in current equation. However, we need value for another entity $P(\theta)$ and that is the *prior*. This is something that we are going to assume as it is not explicitly given to us. If we assume the prior probability to be uniform, then it is independent of θ and the outcome of Bayes' approach will be same as the outcome of MLE. However, in order to showcase the differences between the two approaches, let us use a different and non-intuitive prior. Let $P(\theta = h) = 2h$. Consequently $P(\theta = T)$. While defining this prior, we need to make sure that it is a valid probability density function. The easiest way to make sure that is to confirm $\int_{h=0}^{h=1} P(\theta = h) = 1$. As can be seen from figure, it is indeed true. There is one more factor in the equation in the form of *evidence*, $P(y)$. However, this value is probability of occurrence of the output without any dependency on the constant bias, and is constant with respect to h. When we differentiate with respect to h, the effect of this parameter is going to vanish. Hence, we can safely ignore this term for the purpose of optimization (Fig. 8.1).

So we can now proceed with the optimization problem as before. In order to maximize the posterior, let's differentiate it with respect to h as before,

$$\frac{\partial}{\partial h} P(\theta|y) = \frac{\partial}{\partial h} \frac{P(y|\theta) \cdot P(\theta)}{P(y)} = 0 \tag{8.12}$$

Substituting the values and solving (assuming $h \neq 0$),

$$\frac{\partial}{\partial h} \frac{P(y|\theta) \cdot P(\theta)}{P(y)} = 0$$
$$\frac{\partial}{\partial h}((2 \cdot h)^5 \cdot P(y = H|\theta = h)^4 \cdot P(y = T|\theta = (1 - h))) = 0$$
$$\frac{\partial}{\partial h}(2^5 \cdot h^5 \cdot (h^4 - h^5)) = 0$$

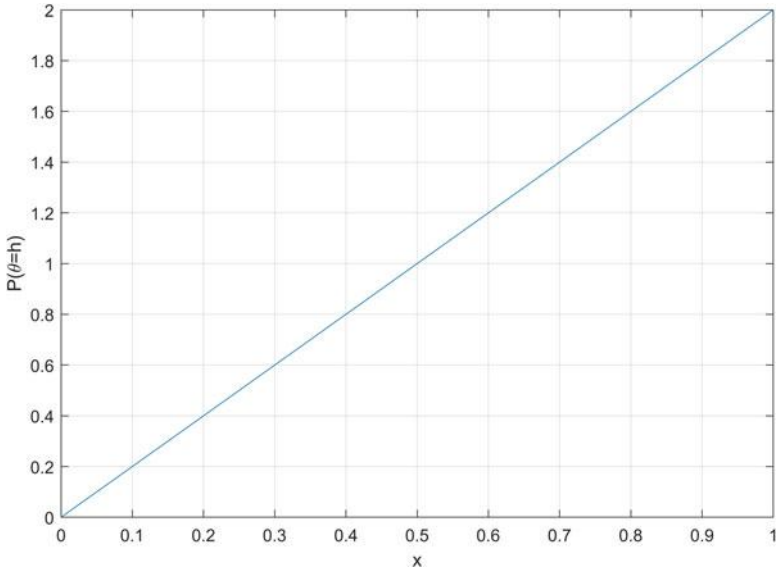


Fig. 8.1 Probability density function (pdf) for the prior

$$\begin{aligned}\frac{\partial}{\partial h}(2^5 \cdot (h^9 - h^{10})) &= 0\\ 9 \cdot h^8 - 10 \cdot h^9 &= 0\\ 9 \cdot h^8 &= 10 \cdot h^9\\ h &= \frac{9}{10}\end{aligned}$$

With Bayes's approach, probability of getting *Head* in the next toss would be $\frac{9}{10}$. Thus assumption of a non-trivial prior with Bayes' approach leads to a different answer compared to MLE.

8.3 Generative Models

As discussed earlier, generative models try to understand how the data that is being analyzed came to be in the first place. They find applications in multiple different fields where we need to synthesize speech, images, or 3D environments that resemble the real life but is not directly copied from any of the real examples. The generative models can be broadly classified into two types: (1) Classical models and (2) Deep learning based models. We will look at few examples of classical generative models briefly.

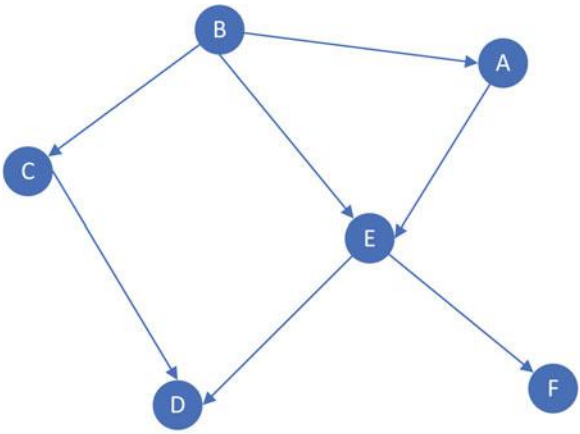


Fig. 8.2 Sample Bayesian network

8.3.1 Mixture Methods

One of the fundamental aspect of generative models is to understand the composition of the input. Understand how the input data came to existence in the first place. Most simplistic case would be to have all the input data as outcome of a single process. If we can identify the parameters describing the process, we can understand the input to its fullest extent. However, typically any input data is far from such ideal case, and it needs to be modelled as an outcome of multiple processes. This gives rise to the concept of mixture models.

8.3.2 Bayesian Networks

Bayesian networks represent directed acyclic graphs as shown in Fig. 8.2. Each node represents an observable variable or a state. The edges represent the conditional dependencies between the nodes. Training of Bayesian network involves identifying the nodes and predicting the conditional probabilities that best represent the given data.

8.4 Some Useful Probability Distributions

We will conclude this chapter with detailing some of the commonly used probability distributions. There are likely hundreds of different distributions studied in the literature of probability and statistics. We don't want to study them all, but in my experience with machine learning projects so far, I have realized that knowledge of few key distributions goes a long way. Hence, I am going to describe these

distributions here without going to into theoretical details of their origins etc. We will look at the probability density functions or *pdf*'s and cumulative density functions or *cdf*'s of these distributions and take a look at the parameters that define these distributions. Here are the definitions of these quantities for reference:

Definition 8.1 pdf A probability density function or pdf is a function $P(X = x)$ that provides probability of occurrence of value x for a given variable X. The plot of $P(X = x)$ is bounded between [0, 1] on y-axis and can spread between $[-\infty, \infty]$ on x-axis and integrates to 1.

Definition 8.2 cdf A cumulative density function or cdf is a function $C(X = x)$ that provides sum of probabilities of occurrences of values of X between $[-\infty, x]$. This plot is also bounded between [0, 1]. Unlike pdf, this plot starts at 0 on left and ends into 1 at the right.

I would strongly advise the reader to go through these distributions and see the trends in the probabilities as the parameters are varied. We come across distributions like these in many situations and if we can match a given distribution to a known distribution, the problem can be solved in far more elegant manner.

8.4.1 Normal or Gaussian Distribution

Normal distribution is one of the most widely used probability distribution. It is also called as bell shaped distribution due to the shape of its pdf. The distribution has vast array of applications including error analysis. It also approximates multitude of other distributions with more complex formulations. Another reason the normal distribution is popular is due to central limit theorem.

Definition 8.3 Central Limit Theorem Central limit theorem states that, if sufficiently large number of samples are taken from a population from any distribution with finite variance, then the mean of the samples asymptotically approaches the mean of the population. In other words, sampling distribution of mean taken from population of any distribution asymptotically approaches normal distribution.

Hence sometimes the normal distribution is also called as distribution of distributions.

Normal distribution is also an example of continuous and unbounded distribution, where the value of x can span $[-\infty, \infty]$. Mathematically, the pdf of normal distribution is given as

$$P_{\text{normal}}(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{8.13}$$

where μ is the mean and σ is the standard deviation of the distribution. Variance is σ^2. cdf of normal distribution is given as

$$C_{\text{normal}}(x|\mu,\sigma) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \tag{8.14}$$

where erf is a standard error function, defined as

$$\text{erf}(x) = \frac{1}{\sqrt{\pi}}\int_{-x}^{x} e^{-t^2} \tag{8.15}$$

functional that is being integrated is symmetric, hence it can also be written as

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}}\int_{0}^{x} e^{-t^2} \tag{8.16}$$

The plots of the pdf and cdf are shown in figure (Figs. 8.3 and 8.4).

8.4.2 Bernoulli Distribution

Bernoulli distribution is an example of discrete distribution and its most common application is probability of coin toss. The distribution owes its name to a great mathematician of the seventeenth century, *Jacob Bernoulli*. The distribution is based on two parameters p and q, which are related as $p = 1 - q$. Typically p is called the probability of success (or in case of coin toss, it can be called as probability of getting a *Head*) and q is called the probability of failure (or in case of coin toss, probability of getting a *Tail*). Based on these parameters, the pdf (sometimes, in case of discrete variables, it is called as *probability mass function* or *pmf*, but for the sake of consistency, we will call it pdf) of Bernoulli distribution is given as

$$P_{\text{bernoulli}}(k|p,q) = \begin{cases} p, & \text{if } k = 1, \\ q = 1 - p, & \text{if } k = 0. \end{cases} \tag{8.17}$$

here, we use the discrete variable k instead of continuous variable x. The cdf is given as

$$C_{\text{bernoulli}}(k|p,q) = \begin{cases} 0, & \text{if } k < 0, \\ q = 1 - p, & \text{if } 0 \leq k < 1, \\ 1, & \text{if } k \geq 1. \end{cases} \tag{8.18}$$

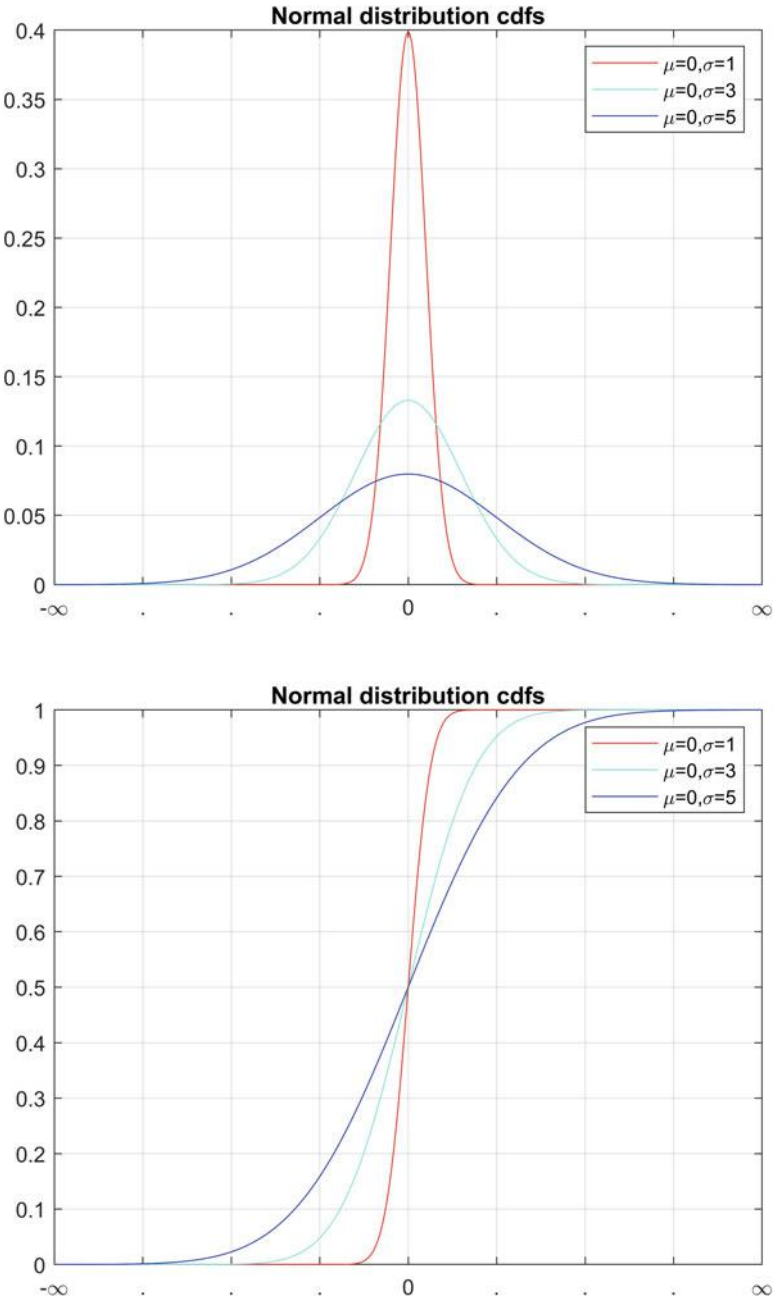


Fig. 8.3 Plot of normal pdfs for 0 mean and different variances

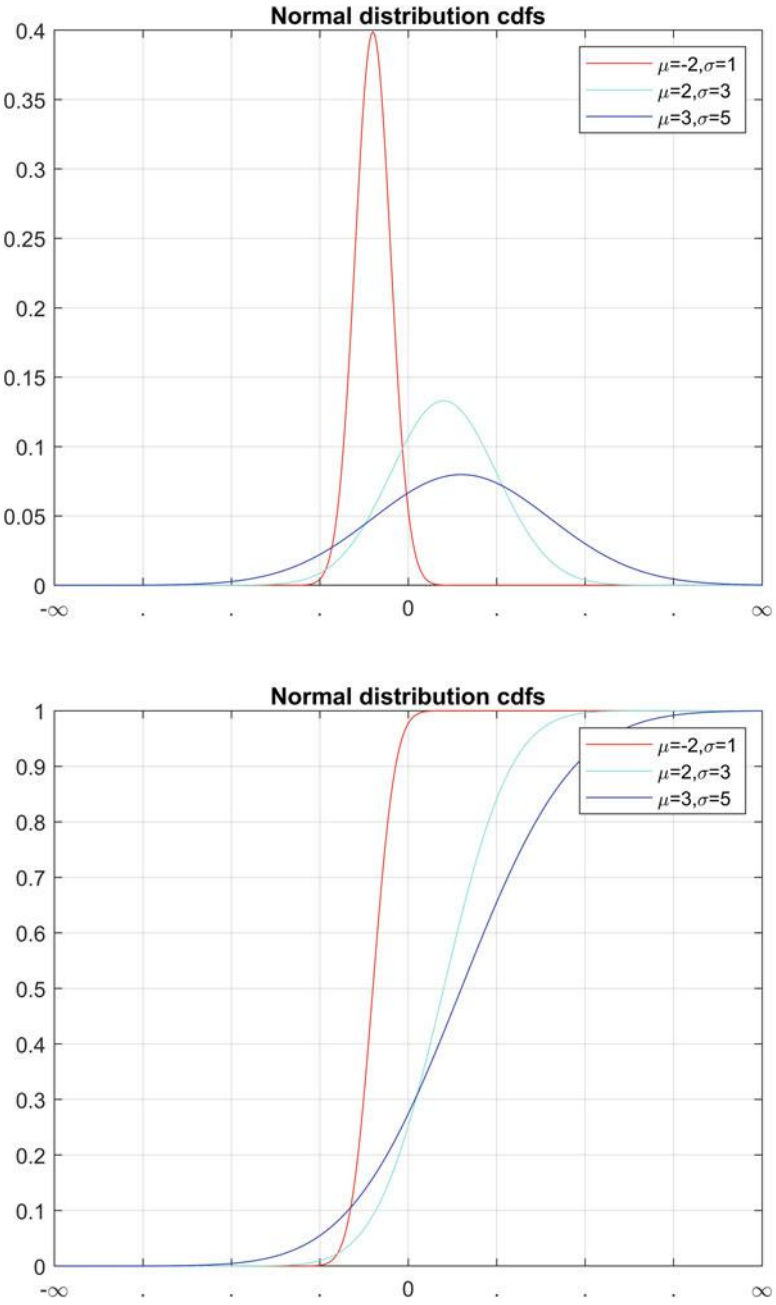


Fig. 8.4 Plot of normal pdfs for different means and different variances

8.4.3 Binomial Distribution

Binomial distribution generalizes Bernoulli distribution for multiple trials. Binomial distribution has two parameters n and p. n is number of trials of the experiment, where probability of success is p. The probability of failure is $q = 1 - p$ just like Bernoulli distribution, but it is not considered as a separate third parameter. The pdf for binomial distribution is given as

$$P_{\text{Binomial}}(k|n, p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{8.19}$$

where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{8.20}$$

is called as binomial coefficient in this context. It also represents the number of combinations of k in n from the permutation-combination theory where it is represented as

$${}^{n}C_k = \binom{n}{k} \tag{8.21}$$

The cdf of binomial distribution is given as

$$C_{\text{Binomial}}(k|n, p) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i} \tag{8.22}$$

8.4.4 Gamma Distribution

Gamma distribution is also one of the very highly studied distribution in theory of statistics. It forms a basic distribution for other commonly used distributions like chi-squared distribution, exponential distribution etc., which are special cases of gamma distribution. It is defined in terms of two parameters: α and β. The pdf of gamma distribution is given as

$$P_{\text{gamma}}(x|\alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)} \tag{8.23}$$

where $x > 0$ and $\alpha, \beta > 0$. The simple definition of $\Gamma(\alpha)$ for integer parameter is given as a factorial function as

$$\Gamma(n) = (n-1)! \tag{8.24}$$

The same definition is generalized for complex numbers with positive real parts as

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx \tag{8.25}$$

This function also follows the recursive property of factorials as

$$\Gamma(\alpha) = (\alpha - 1)\Gamma(\alpha - 1) \tag{8.26}$$

Figure 8.5 shows plots of the pdf for variable α and β values.

The cdf of gamma function cannot be stated easily as a single valued function, but rather is given as sum of an infinite series as

$$C_{\text{gamma}}(x|\alpha, \beta) = e^{-\beta x} \sum_{i=\alpha}^{\infty} \frac{(\beta x)^i}{i!} \tag{8.27}$$

Figure 8.6 shows plots of the cdf for variable α and β values similar to the ones shown for pdf.

8.4.5 Poisson Distribution

Poisson distribution is a discrete distribution loosely similar to Binomial distribution. Poisson distribution is developed to model the number of occurrences of an outcome in fixed interval of time. It is named after a French mathematician *Siméon Poisson*. The pdf of Poisson distribution is given in terms of number of events (k) in the interval as

$$P_{\text{Poisson}}(k) = e^{-\lambda} \frac{\lambda^k}{k!} \tag{8.28}$$

where the single parameter λ is the average number of events in the interval. The cdf of Poisson distribution is given as

$$C_{\text{Poisson}}(k) = e^{-lambda} \sum_{i=0}^{k} \frac{\lambda^i}{i!} \tag{8.29}$$

Figures 8.7 and 8.8 show the pdf and cdf of Poisson distribution for various values of the parameter λ.

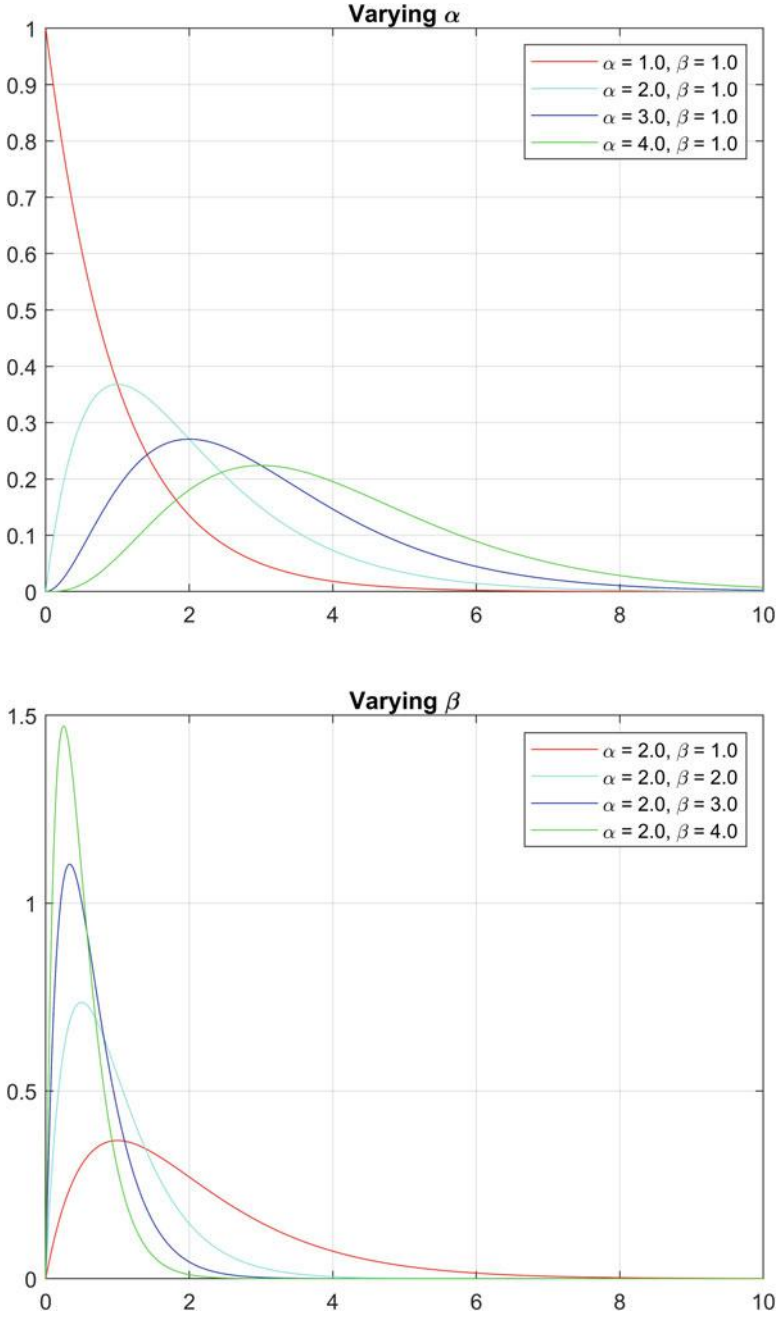


Fig. 8.5 Plot of Gamma pdfs for different values of α and β

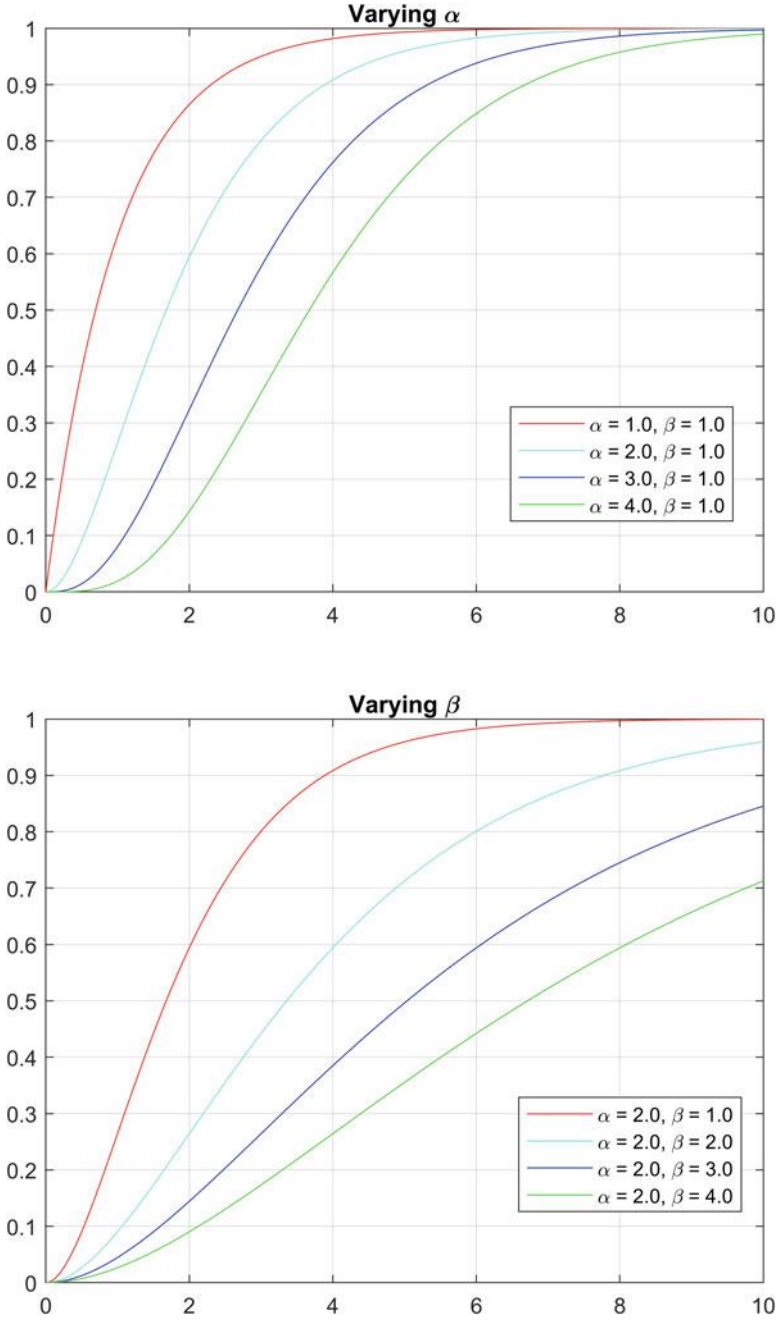


Fig. 8.6 Plot of Gamma cdfs for different values of α and β

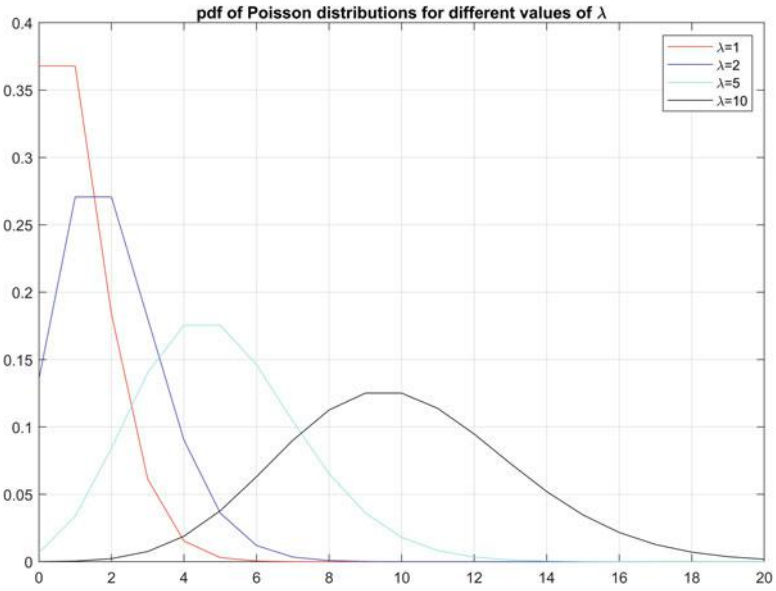


Fig. 8.7 Plot of Poisson pdfs for different values of λ

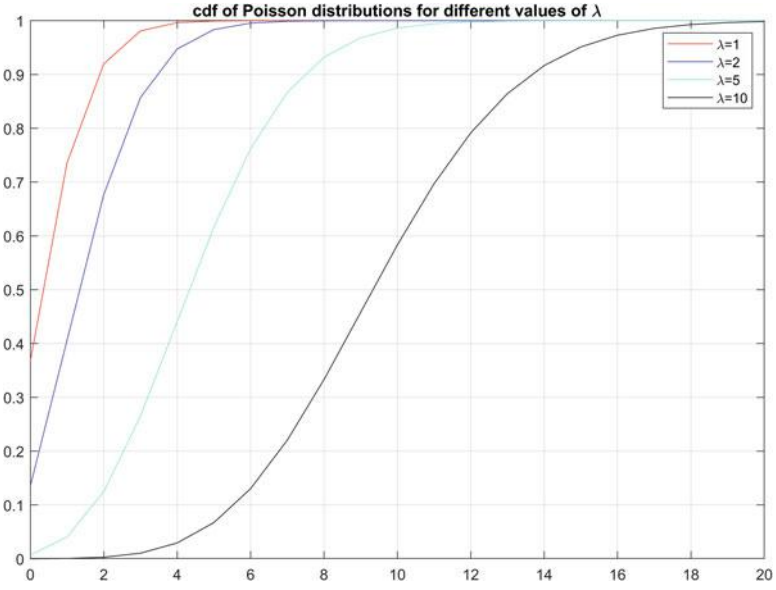


Fig. 8.8 Plot of Poisson cdfs for different values of λ

8.5 Conclusion

In this chapter, we studied various methods based on probabilistic approach. These methods start with some different fundamental assumptions compared to other methods, specifically the ones based on Bayesian theory. The knowledge of priory knowledge separates them from all other methods. If available, this priory knowledge can improve the model performance significantly as we saw in fully worked example. Then we concluded the chapter with learning bunch of different probability distributions with their density and cumulative functions.

Chapter 9
Dynamic Programming and Reinforcement Learning

9.1 Introduction

The theory of dynamic programming was developed by Bellman [38] in 1950s. In the preface of his iconic book, he defines dynamic programming as follows:

> The purpose of this book is to provide introduction to the mathematical theory of multi-stage decision process. Since these constitute a somewhat complex set of terms we have coined the term *dynamic programming* to describe the subject matter.

This is very interesting and apt naming as the set of methods that come under the umbrella of dynamic programming is quite vast. These methods are deeply rooted in pure mathematics, but are more favorably expressed so that they can be directly implemented as computer programs. In general such multi-stage decision problems appear in all sorts of industrial applications, and tackling them is always a daunting task. However, Bellman describes a structured and sometimes iterative manner in which the problem can be broken down and solved sequentially. There exists a notion of state machine in sequential solution of the subproblems and context of subsequent problems changes dynamically based on the solution of previous problems. This non-static behavior of the methods imparts the name *dynamic*. These types of problems also marked the early stages of *Artificial Intelligence* in the form of *expert systems*.

9.2 Fundamental Equation of Dynamic Programming

In general, the problem that dynamic programming tries to solve can be given in the form of a single equation and it is called as *Bellman equation* (Fig. 9.1). Let us consider a process that goes through N steps. At each step there exist a state and possible set of actions. Let the initial state be s_0 and first action taken be a_0. We also

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_9

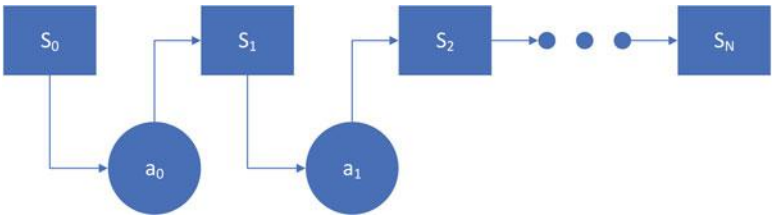


Fig. 9.1 The setup for Bellman equation

constrain the set of possible actions in step t as $a_t \in \Gamma(s_t)$. Depending on the action taken, the next state is reached. Let us call the function that combines the current state and action and produces next state as $T(s, a)$. Hence $s_1 = T(s_0, a_0)$. As the process is going through multiple states, let the problem we are trying to solve is to optimize a value function at step t as $V(s_t)$.

The optimality principle in iterative manner can be stated as: "In order to have the optimum value in the last step, one needs to have optimum value in the previous step that will lead to final optimum value". To translate this into equation, we can write

$$V(s_t) = \max_{a_t \in \Gamma(s_t)} V(T(s_t, a_t)) \tag{9.1}$$

This is a special case of Bellman equation when the payout after each state is not considered. To make the equation more generic, let us add the payout function at step t as $F(s_t, a_t)$. A more general form of Bellman equation can now be given as

$$V(s_t) = \max_{a_t \in \Gamma(s_t)} (F(s_t, a_t) + V(T(s_t, a_t))) \tag{9.2}$$

In some scenarios, the optimality of future value cannot be assumed to be fully attainable, and a discount factor needs to be added as β, where $0 < \beta < 1$. Then the Bellman equation can be written as

$$V(s_t) = \max_{a_t \in \Gamma(s_t)} (F(s_t, a_t) + \beta V(T(s_t, a_t))) \tag{9.3}$$

This generic equation is in fairly abstract form, where we have not defined any specific problem, or specific constraints or even the specific value function that we want to maximize or minimize. However, once we have these defined, we can safely use this equation to solve the problem as long as the functions are continuous and differentiable.

9.3 Classes of Problems Under Dynamic Programming

Dynamic programming defines a generic class of problems that share the same assumptions as theory of machine learning. The exhaustive list of problems that can be solved using theory of dynamic programming is quite large as can be seen here [4]. However, the most notable classes of problems that are studied and applied are:

- Travelling salesman problem
- Recursive Least Squares (RLS) method
- Finding shortest distance between two nodes in graph
- Viterbi algorithm for solving hidden Markov model (HMM)

Other than these specific problems, the area that is most relevant in the context of modern machine learning is *Reinforcement learning* and its derivatives. We will study these concepts in the rest of the chapter.

9.4 Reinforcement Learning

Most of the machine learning techniques we have explored so far and will explore in later chapters primarily focus on two types of learning: (1) Supervised and (2) Unsupervised. Both methods are classified based on the availability of labelled data. However, none of these types really focus on interaction with the environment. Even in supervised learning techniques the labelled data is available beforehand. Reinforcement learning takes a fundamentally different approach towards learning. It follows biological aspects of learning more closely. When a newborn baby starts interacting with the environment, its learning begins. In the initial times, the baby is making mostly some random actions and is being greeted by the environment in some way. This is called as reinforcement learning. It cannot be classified into either of the two types. Let us look at some of the fundamental characteristics of the reinforcement learning to understand precisely how it differs from these methods. The reinforcement learning framework is based on interaction between two primary entities: (1) system and (2) environment.

9.4.1 Characteristics of Reinforcement Learning

1. There is no preset labelled training data available.
2. The action space is predefined that typically can contain very large number of possible actions that the system can take at any given instance.
3. The system chooses to make an action at every instance of time. The meaning of instance is different for each application.

4. At every instance of time a feedback (also called as reward) from the environment is recorded. It can either be positive, negative, or neutral.
5. There can be delay in the feedback.
6. System learns while interacting with the environment.
7. The environment is not static and every action made by the system can potentially change the environment itself.
8. Due to dynamic nature of environment, the total training space is practically infinite.
9. The training phase and application phase are not separate in case of reinforcement learning. The model is continuously learning as it is also predicting.

9.4.2 Framework and Algorithm

It is important to note that reinforcement learning is a framework and not an algorithm like most other methods discussed in this book, hence it can only be compared with other learning frameworks like supervised learning. When an algorithm follows above characteristics, the algorithm is considered as a reinforcement learning algorithm. Figures 9.2 and 9.3 show the architectures of reinforcement learning and supervised learning frameworks. Unsupervised learning is completely different as it does not involve any type of feedback or labels, and is not considered here.

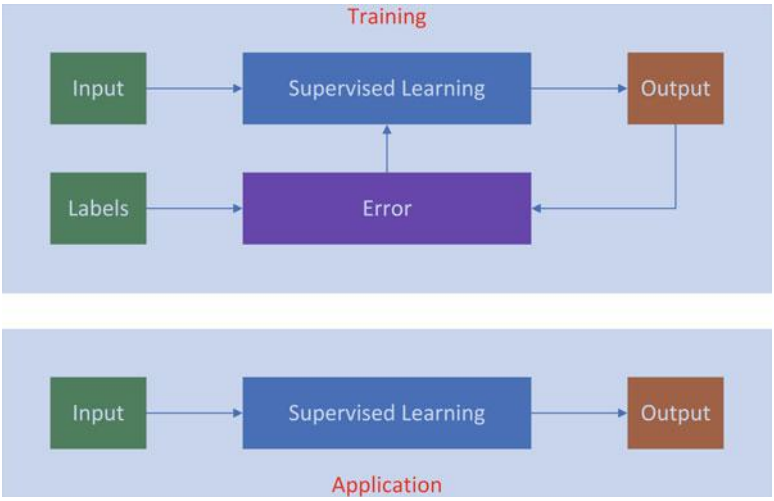


Fig. 9.2 Architecture of supervised learning

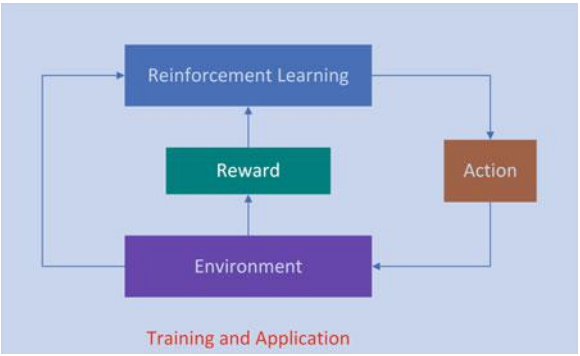


Fig. 9.3 Architecture of reinforcement learning

9.5 Exploration and Exploitation

Reinforcement learning introduces two new concepts in the process of learning called exploration and exploitation. As the system starts its learning process, there is no knowledge learned so far and every action taken by the system is pure random. This is called exploration. During exploration, the system is just trying out different possible actions that it can make and registering the feedback from the system as positive, negative, or neutral reward. After a while in learning phase, when sufficient feedback is gathered, the system can start using the knowledge learned from the previous exploration and start producing actions that are not random but deterministic. This is called exploitation. Reinforcement learning needs to find a good tradeoff between exploration and exploitation. Exploration opens up more possible actions that can lead to better long term rewards in future at the cost of lower possible rewards in short term, while exploitation tends to get better short term rewards at the cost of possibly missing out on greater long term rewards possible due to actions not known at the time.

9.6 Examples of Reinforcement Learning Applications

The theory of reinforcement learning is better understood after looking at some of the real life applications as follows:

1. **Chess programs**: Solving the problem of winning a chess game by computers is one of the classic application of reinforcement learning. Every move that is made by either side opens up a new position on the board. The ultimate objective is it captures the king of the other side, but the short term goals are to capture other pieces of other side or gain control of the center, etc. The action space is practically infinite, as there are 32 pieces in total on 64 squares and each piece has different types of moves allowed. One of the conservative estimates on the number of possible moves in chess is calculated to be around

10^{120}, which is also known as *Shannon number* [10]. Deep Blue system by IBM was a supercomputer specifically designed to play chess was able to defeat reigning world champion Gary Kasparov in 1997 [11]. This was considered as landmark in machine learning. It did use some bits of reinforcement learning, but it was heavily augmented with huge database of past games. Since then the computers have become increasingly better at the game. However, the real victory of reinforcement learning came with Google's AlphaZero system. This system was trained by playing against itself and learning all the concepts of chess. Just after 9 h of training and without any knowledge of previously played games, it was able to defeat the other world champion chess program, called *Stockfish* in 2015 [60].

2. **Robotics**: Training a robot to maneuver in a complex real world is a classical reinforcement learning problem that closely resembles the biological learning. In this case, the action space is defined by the combination of the scope of moving parts of the robot and environment is the area in which the robot needs to maneuver along with all the existing objects in the area. If we want to train the robot to lift one object from one place and drop it at another, then the rewards would be given accordingly.
3. **Video Games**: Solving video games is another interesting application of reinforcement learning problems. A video game creates a simulated environment in which the user needs to navigate and achieve certain goals in the form of say winning a race or killing a monster, etc. Only certain combination of moves allows the user to pass through various challenging levels. The action space is also well defined in the form of up, down, left, right, accelerate brake, or attack with certain weapon, etc. Open AI has created a platform for testing reinforcement learning models to solve video games in the form of *Gym* [13]. Here is one application where the *Gym* is used to solve stages in classical game of *Super Mario* [12].
4. **Personalization**: Various e-commerce websites like Amazon, Netflix have most of their content personalized for each user. This can be achieved with the use of reinforcement learning as well. The action space here is the possible recommendations and reward is user engagement as a result of certain recommendation.

9.7 Theory of Reinforcement Learning

Figure 9.4 shows the signal flow and value updates using reinforcement learning architecture. s_k denotes the state of the system that is combination of environment and the learning system itself at time instance k. a_k is action taken by the system, and r_k is the reward given by the environment at the same time instance. π_k is the policy for determining the action at the same time instance and is function of current state. V^{π} denotes the value function that updates the policy using current state and reward.

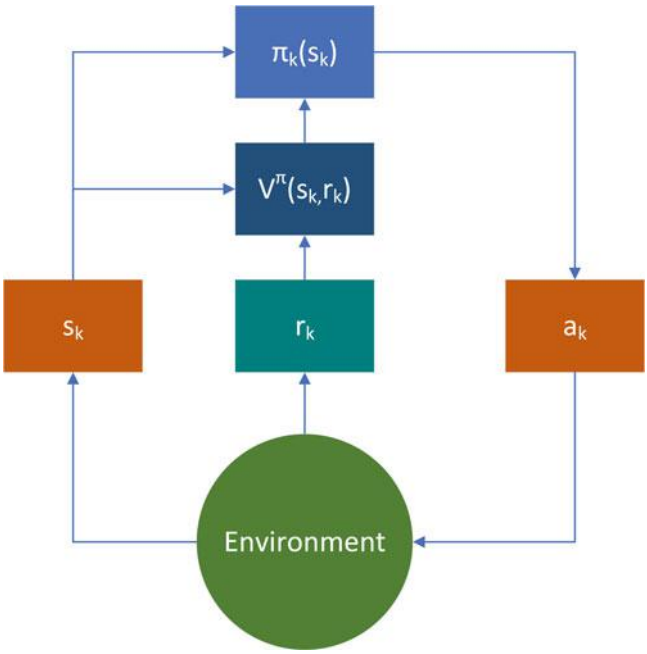


Fig. 9.4 Reinforcement learning model architecture

9.7.1 Variations in Learning

This depiction of reinforcement learning combines various different methods into a single generic representation. Here are the different methods typically used:

1. Q-learning
2. SARSA
3. Monte Carlo

9.7.1.1 Q-Learning

In order to understand Q-learning, let's consider the most generic form of Bellman equation as Eq. 9.3. In Q-learning framework, the function $T(s, a)$ is called as the value function. The technique of Q-learning focusses on learning the values of $T(s, a)$ for all the given states and action combinations. The algorithm of Q-learning can be summarized as,

1. Initialize the Q-table, for all the possible state and action combinations.
2. Initialize the value of β.
3. Choose an action using a tradeoff between exploration and exploitation.

4. Perform the action and measure the reward.
5. Update the corresponding Q value using Eq. 9.3
6. Update the state to next state.
7. Continue the iterations (steps 3–6) till the target is reached.

9.7.1.2 SARSA

SARSA stands for *state-action-reward-state-action* [66]. SARSA algorithm is an incremental update to Q-learning where it adds learning based on policy. Hence it is also sometimes called as *on-policy Q-learning*, which the traditional Q-learning is off-policy. The update equation for SARSA can be given as

$$V'(s_t, a_t) = (1 - \alpha\beta)V(s_t, a_t) + \alpha F(s_t, a_t) + \alpha\beta V(s_{t+1}, a_{t+1}) \tag{9.4}$$

where β is discount factor as before, and α is called as learning rate.

9.8 Conclusion

In this chapter, we studied methods belonging to the class of dynamic programming as defined by Bellman. The specific case of reinforcement learning and its variations mark a topic of their own and we devoted a section for studying these concepts and their applications. Reinforcement learning marks a whole new type of learning that resembles to human learning more so than traditional supervised and unsupervised learning techniques. Reinforcement learning enables fully automated way of learning in a given environment. These techniques are becoming quite popular in the context of deep learning and we will study those aspects in later chapters.

Chapter 10
Evolutionary Algorithms

10.1 Introduction

All the traditional algorithms including the new deep learning framework tackle the problem of optimization using calculus of gradients. The methods have evolved significantly to solve harder problems that were once considered impossible to solve. However, the horizon of the reach of these algorithms is linear and predictable. Evolutionary algorithms try to attack the optimization problems in a fundamentally different manner of massive exploration in a random but supervised manner. This approach opens up whole new types of solutions for the problems at hand. Also, these methods are inherently suitable for embarrassingly parallel computation, which is the *mantra* of modern computation based on GPUs.

10.2 Bottleneck with Traditional Methods

In the applications of machine learning, one comes across many problems for which it is practically impossible to find universally optimal solution. In such cases one has to be satisfied with a solution that is optimal within some reasonable local neighborhood (the neighborhood is from the perspective of the hyperspace spanned by the feature values). Figure 10.1 shows an example of such space.

Most traditional methods employ some form of linear search in a greedy manner.[1] In order to see a greedy method in action, let us zoom into the previous

[1] In general all the algorithms that use gradient based search are called as greedy algorithms. These algorithms use the fact from calculus that at any local optimum (minimum or maximum) the value of gradient is 0. In order to distinguish between whether the optimum is a minimum or a maximum second order gradient is used. When the second order gradient is positive a minimum is reached, otherwise it's a maximum.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_10

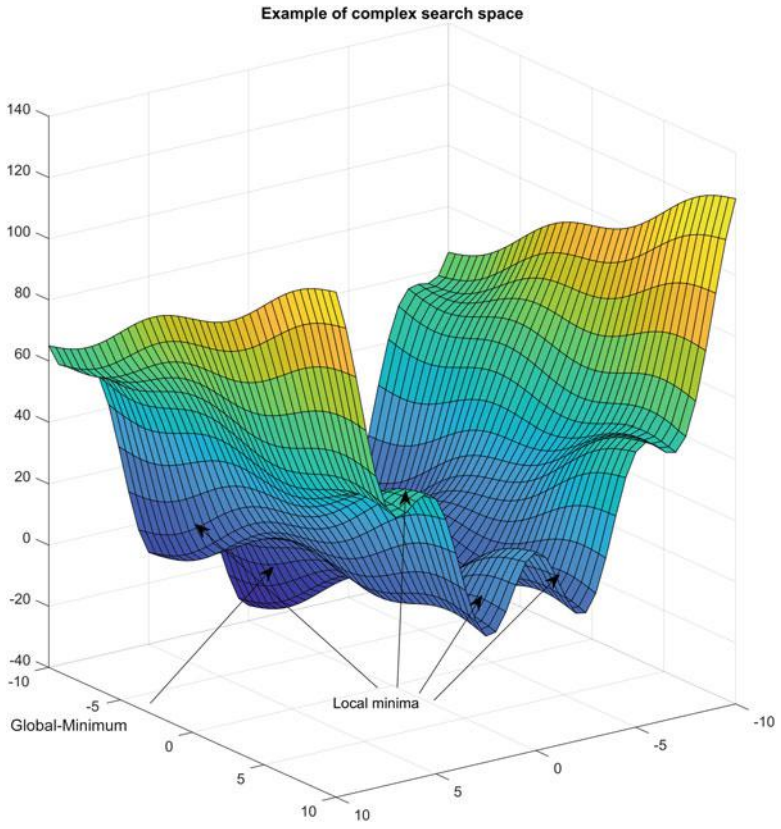


Fig. 10.1 Example of complex search space with multiple local minima and a unique single global minimum

figure, as shown in Fig. 10.2. The red arrows show how the greedy algorithm progresses based on the gradient search and results into a local minimum.

10.3 Darwin's Theory of Evolution

There exists a very close parallel to this problem in the theory of natural evolution. In any given environment, there exist a complex set of constraints in which all the inhabitant animals and plants are fighting for survival and evolving in the process. The setup is quite dynamic and a perfectly ideal species does not exist for any given environment at all times. All the species have some advantages and some limitations at any given time. The evolution of the species is governed by Darwin's theory of evolution by natural selection. The theory can be stated briefly as:

> Over sufficiently long span of time, only those individual organisms survive in a given environment who are better suited for the environment.

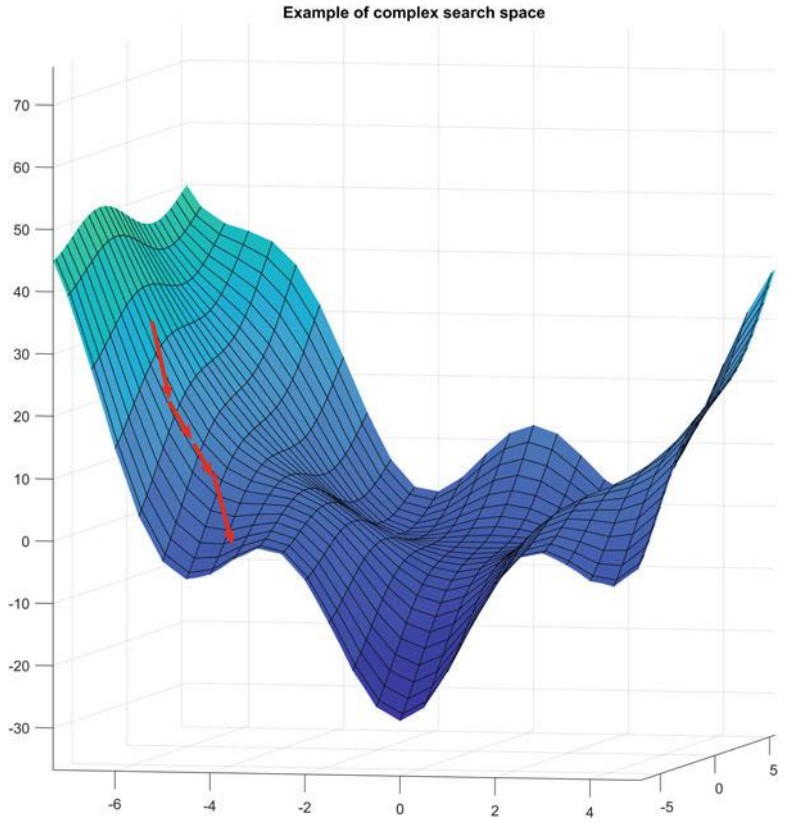


Fig. 10.2 Example of greedy search in action resulting into a local minimum

This time-span can extend over multiple generations and its effects are typically not seen in a matter of few years or even a few hundred years. However, in few thousand years and more the effect of evolution by natural selection is seen beyond doubt. There is one more aspect to the theory of evolution without which it cannot work, and that is random variation in the species that typically happen by the process of mutation. If the process of reproduction kept on producing species that are identical to the parents, there would never be any change in the setup and natural selection cannot successfully happen. However, when we add a random variation in the offsprings during each reproduction, it changes everything. The new features that are created as a result of mutation are put to test in the environment. If the new features make the organism better cope with the environment, the organisms with those features thrive and tend to reproduce more in comparison to the organisms that do not have those features. Thus over time, the offsprings of the weaker organisms are extinct and the species overall have evolved. As a result of continued evolution, over time, the species get better and better with respect to surviving the given environment. In long run, the process of evolution leads the species along the

direction of better adaptation and on aggregate level it never retrogresses. These characteristics of evolution are rather well suited for the problems described at the beginning.

All the algorithms that come under the umbrella of evolutionary algorithms are inspired from this concept. Each such algorithm tries to interpret the concepts of random variation, environmental constraints, and natural selection in its own way to create a resulting evolution. Fortunately, as the processes of random variations and natural selection are implemented using computer running in GHz, they can happen in matter of mere seconds compared to thousands or hundreds of thousands of years in the biological setup.

Thus, evolutionary methods rely on creating an initial population of samples that is chosen randomly, instead of a single starting point used in greedy methods. Then they let processes of mutation based sample variation and natural selection to do their job to find which sample evolves into better estimate. Thus evolutionary algorithms also do not guarantee a global minimum, but they typically have higher chance of finding one.

Following sections describe few most common examples of evolutionary algorithms.

10.4 Genetic Programming

Genetic programming models try to implement Darwin's idea as closely as possible. It maps the concepts of genetic structure to solution spaces and implements the concepts of natural selection and reproduction with possibility of mutation in programmatic manner. Let us look at the steps in the algorithm.

Steps in Genetic Programming

1. Set the process parameters as stopping criteria, mutation fraction, etc.
2. Initialize the population of solution candidates using random selection.
3. Create a fitness index based on the problem at hand.
4. Apply the fitness index to all the population candidates and trip the number of candidates to a predetermined value by eliminating the lowest scoring candidates.
5. Randomly select pairs of candidates from the population as parents and carry out the process of reproduction. The process of reproduction can contain two alternatives:

   (a) Crossover: In crossover, the parent candidates are combined in a predefined structural manner to create the offspring.

(continued)

(b) Mutation: In mutation, the children created by the process of crossover are modified randomly. The mutation is applied only for a fraction of the offsprings as determined as one of the settings of the process.

6. Augment the original population with the newly created offsprings.
7. Repeat steps 4, 5, and 6 till desired stopping criteria are met.

Although the steps listed in the algorithm are fairly straightforward from biological standpoint, they need customization based on the problem at hand for programmatic implementation. In order to illustrate the complexity in customization let us take a real problem. A classic problem that is very hard to solve using traditional methods but is quite suitable for genetic programming is of travelling salesman. The problem is like this:

> There is a salesman that wants to travel to n number of destinations in sequence. The distance between each pair of destinations is given as d_{ij}, where $i, j \in \{1, 2, \ldots, n\}$. The problem is to select the sequence that connects all the destinations only once in shortest overall distance.

Although apparently straightforward looking one, this problem is actually considered as one of the hardest to solve[2] and universally optimal solution to this problem is not possible even when the number of destinations is as small as say 100.

Let's try to solve this problem using above steps.

1. Let us define the stopping criteria as successive improvement in the distance to be less than some value α or maximum number of iterations.
2. We first need to create a random population of solutions containing say k number of distinct solutions. Each solution is random sequence of destination from 1 to n.
3. The fitness test would be given as sum of distances between successive destinations.
4. We will keep k number of top candidates when sorted in decreasing order of total distance.
5. Reproduction step is where things get a little tricky. First we will choose two parents randomly. Now, let's consider the two cases one by one.

   a. For crossover, we will select first $k_1, k_1 < k$ destinations directly from parent-1 and then remaining destinations from parent-2. However, this simple crossover can lead to duplicating some destinations and missing some destinations in the new sequence, called as the offspring sequence. These errors need to fixed by appropriate adjustment.

[2]This problem belongs to a class of problems called as NP-hard. It stands for nondeterministic polynomial time hard problems [27]. The worst case solution time for this problem increases in near exponential time and quickly becomes beyond the scope of current hardware.

b. For mutations, once the crossover based offspring sequence is generated, randomly some destinations are swapped.

6. Once we have reproduced the full population, we will have a population of twice the size. Then we can repeat above steps as described in the algorithm till stopping criteria are reached.

Unfortunately due to the random factor in the design of genetic programs, one cannot have a deterministic bound on how much time it would take to reach the acceptable solution, or how much should be the size of the population or how much should be the percentage of mutations, etc. One has to experiment with multiple values of these parameters to find the optimal solution for each given case. In spite of this uncertainty, genetic programs are known to provide significant improvements in computation times for solutions of certain types of problems and are in general a strong tool to have in the repertoire of machine learning toolkit.

10.5 Swarm Intelligence

Swarm intelligence is a generic term used to denote algorithms that are influenced by the biological aspects of groups primitive organisms. The origin of swarm intelligence techniques can be traced back to 1987, when Craig Reynolds published his work on *boids* [68]. In his work Reynolds designed a system of flock of birds and assigned a set of rules governing the behavior of each of the bird in the flock. When we aggregate the behavior of the group over time, some completely startling and non-trivial trends emerge. This behavior can be attributed to the saying that sometimes, $1 + 1 > 2$. When a single bird is considered as a singular entity and is let loose in the same environment, it has no chance of survival. If all the birds in the flock act as single entities, they all are likely to perish. However, when one aggregates the birds to form a social group that communicates with each other without any specific governing body, the abilities of the group improve significantly. Some of the very long migrations of birds are classic examples of success of swarm intelligence.

In recent years, swarm intelligence is finding applications in computer graphics for simulating groups of animals or even humans in movies and video games. As a matter of fact, the encyclopedia of twenty-first century: *Wikipedia* can also be attributed to swarm intelligence. The techniques of swarm intelligence are also finding applications in controlling a group autonomous flying drones. In general the steps in designing an algorithm that is based on swarm intelligence can be outlined as follows:

1. Initialize the system by introducing a suitable environment by defining constraints.
2. Initialize the individual organism by defining the rules of possible actions and possible ways of communicating with others.

3. Establish the number of organisms and period of evolution.
4. Define the individual goals for each organism and group goals for the whole flock as well as stopping criteria.
5. Define the randomness factor that will affect the decisions made by individual organisms by trading between exploration and exploitation.
6. Repeat the process till the finishing criteria are met.

10.6 Ant Colony Optimization

Although ant colony optimization can be considered as a subset of swarm intelligence, there are some unique aspects to this method and it needs separate consideration. Ant colony optimization algorithms as the name suggests, is based on the behavior of a large groups of ants in a colony. The individual ant possesses a very limited set of skills, e.g., they have a very limited vision, in most cases they can be completely blind, they have a very small brain with very little intellect, and their auditory and olfactory senses are also not quite advanced. In spite of these limitations, the ant colonies as such are known to have some extraordinary capabilities like building complex nest, finding shortest path towards food sources that can be at large distances from the nest. Another important aspect of the ant colonies is that they are a completely decentralized system. There is no central decision maker, a king or queen ant that orders the group to follow certain actions. All the decisions and actions are decided and executed by individual ant based on its own method of functioning. For the ant colony optimization algorithm, we are specifically going to focus on the capabilities of the ants to find the shortest path from the food source to the nest.

At the heart of this technique lies the concept of *pheromones*. A pheromone is a chemical substance that is dropped by each ant as it passes along any route. These dropped pheromones are sensed by the ants that come to follow the same path. When an ant reaches at a junction, it chooses the path with higher level of pheromones with higher probability. This probabilistic behavior combines the random exploration with exploitation of the paths travelled by other ants. The path that connects the food source with the nest in least distance is likely going to be used more often than the other paths. This creates a form of positive feedback and the path with shortest distance keeps getting more and more chosen one over time. In biological terms, the shortest path evolves over time. This is quite a different way of interpreting the process of evolution, but it conforms to the fundamentals nonetheless. All the different paths connecting the nest with the food source mark the initial population. The subsequent choices of different paths, similar to process of reproduction, are governed in probabilistic manner using pheromones. Then the positive feedback created by aggregation of pheromones acts as a fitness test and controls the evolution in general.

These biological concepts related to emission of pheromones and their decay and aggregation can be modelled using mathematical functions to implement this algorithm programmatically. The problem of travelling salesman is also a good candidate to use ant colony optimization algorithm. It is left as an exercise for the reader to experiment with this implementation. It should be noted that: as the ant colony optimization algorithm has graphical nature of the solution at heart it has relatively limited scope compared to genetic programs.

10.7 Simulated Annealing

Simulated annealing [67] is an odd man out in this group of evolutionary algorithms as it finds its origins in metallurgy and not biology. The process of annealing involves heating the metal above a certain temperature called as recrystallization temperature and then slowly cooling it down. When the metal is heated above the recrystallization temperature, the atoms and molecules involved in the crystallization process can move. Typically this movement occurs such that the defects in the crystallization are repaired. After annealing process is complete, the metal typically improves it ductility and machinability as well as electrical conductivity.

Simulated annealing process is applied to solve the problem of finding global minimum (or maximum) in a solution space that contains multiple local minima (or maxima). The idea can be described using Fig. 10.1. Let's say with some initial starting point, the gradient descent algorithm converges to nearest local minimum. Then the simulated annealing program, generates a disturbance into the solution by essentially throwing the algorithm's current state to a random point in a predefined neighborhood. It is expected that the new starting point leads to another local minimum. If the new local minimum is smaller than the previous one, then it is accepted as solution otherwise previous solution is preserved. The algorithm is repeated again till the stopping criteria are reached. By adjusting the neighborhood radius corresponding to the higher temperature in the metallurgical annealing, the algorithm can be fine tuned to get better performance.

10.8 Conclusion

In this chapter, we studied different algorithms inspired by the biological aspects of evolution and adaptation. In general entire machine learning theory is inspired by the human intelligence, but various algorithms used to achieve that goal may not directly be applicable to humans or even other organisms for that matter. However, the evolutionary algorithms are specifically designed to solve some very hard problems using methods that are used by different organisms individually or as a group.

Chapter 11
Time Series Models

11.1 Introduction

All the algorithms discussed so far are based on static analysis of the data. By static it is meant that the data that is used for training purposes is constant and does not change over time. However, there are many situations where the data is not static. For example analysis of stock trends, weather patterns, analysis of audio or video signals, etc. The static models can be used up to certain extent to solve some problems dealing with dynamic data, by taking snapshots of the time series data at a certain time. These snapshots can then be used as static data to train the models. However, this approach is seldom optimal and always results in less than ideal results.

Time series analysis is studied quite extensively for over centuries as part of statistics and signal processing and the theory has quite matured. Typical applications of time series analysis involve trend analysis, forecasting, etc. In signal processing theory the time series analysis also deals with frequency domain which leads to spectral analysis. These techniques are extremely powerful in handling dynamic data. We are going to look at this problem from the perspective of machine learning and we will not delve too much into signal processing aspects of the topic that essentially represent fixed mode analysis. The essence of machine learning is in feedback. When a certain computation is performed on the training data and a result is obtained, the result must somehow be fed back into the computation to improve the result. If this feedback is not present, then it is not a machine learning application. We will use this concept as yardstick to separate the pure signal processing or statistical algorithms from machine learning algorithms and only focus on latter.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_11

11.2 Stationarity

Stationarity is a core concept in the theory of time series and it is important to understand some implications of this before going to modelling the processes. Stationarity or a stationary process is defined as a process for which the unconditional joint probability distribution of its parameters does not change over time. Sometimes this definition is also referred to as *strict stationarity*. A more practical definition based on normality assumption would be the mean and variance of the process remain constant over time. These conditions make the process strictly stationary only when the normality condition is satisfied. If that is not satisfied then the process is called *weak stationary* or *wide sense stationary*. In general, when the process is non-stationary, the joint probability distribution of its parameters changes over time or the mean and variance of its parameters are not constant. It becomes very hard to model such process. Although most processes encountered in real life are non-stationary, we always make the assumptions of stationarity to simplify the modelling process. Then we add concepts of trends and seasonality to address the effects of non-stationarity partially. Seasonality means the mean and variance of the process can change periodically with changing *seasons*. Trends essentially model the slow changes in mean and variance with time. We will see simple models that are built on the assumptions of stationary and then we will look at some of their extensions to take into consideration the seasonality.

To understand the nuances of trends and seasonality let's look at the plots shown in Figs. 11.1 and 11.2. Plot of Microsoft stock price almost seems periodical with a period of roughly 6 months with upward trend, while Amazon stock plot shows irregular changes with overall downward trend. On top of these macro trends, there is additional periodicity on daily basis.

Figure 11.3 shows use of mobile phones per 100 people in Germany from 1960 to 2017. This plot does not show any periodicity, but there is a clear upward trend

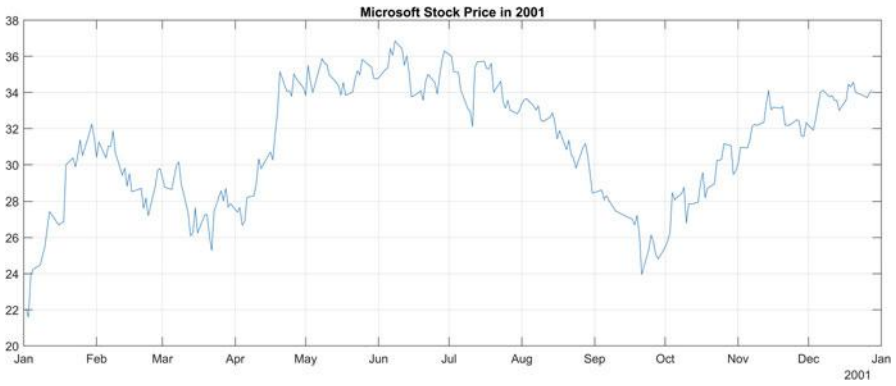


Fig. 11.1 Plot showing Microsoft stock price on daily basis during calendar year 2001

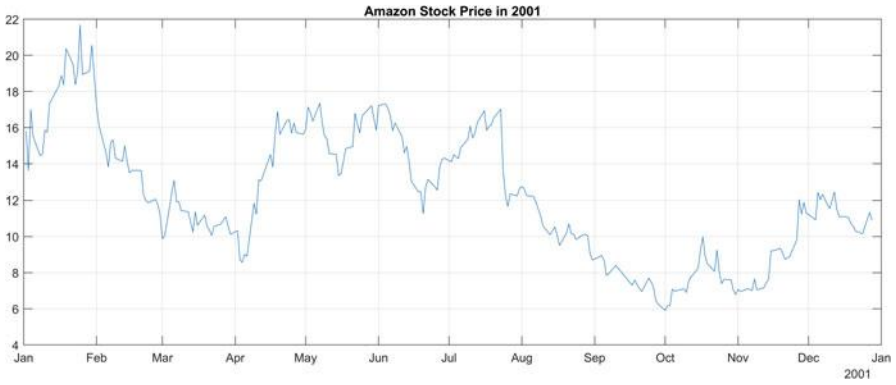


Fig. 11.2 Plot showing Amazon stock price on daily basis during calendar year 2001

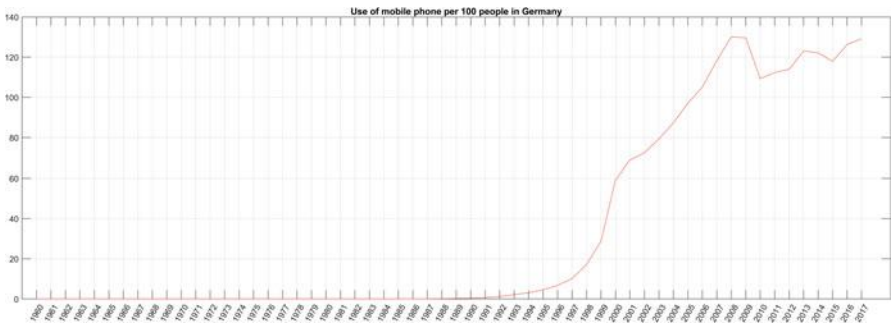


Fig. 11.3 Plot mobile phone use in Germany per 100 people from 1960 to 2017. The data is courtesy of [7]

in the values. The trend is not linear and not uniform. Thus it represents a good example of non-stationary time series.

11.3 Autoregressive and Moving Average Models

Autoregressive moving average or ARMA analysis is one of the simplest of the techniques of univariate time series analysis. As the name suggests this technique is based on two separate concepts: autoregression and moving average. In order to define the two processes mathematically, let's start with defining the system. Let there be a discreet time system that takes white noise inputs denoted as $\epsilon_i, i = 1, \ldots, n$, where i denoted the instance of time. Let the output of the system be denoted as $x_i, i = 1, \ldots, n$. For ease of definition and without loss of generality let's assume all these variables as univariate and numerical.

11.3.1 Autoregressive, or AR Process

An autoregressive or *AR* process is the process in which current output of the system is a function of weighted sum of certain number of previous outputs. We can define an autoregressive process of order p, AR(p) using the established notation as

$$x_i = \sum_{j=i-p}^{i-1} \alpha_j \cdot x_j + \epsilon_i \tag{11.1}$$

α_i are the coefficients or parameters of the AR process along with the error term ϵ_i at instance i. The error term is typically assumed to be a white noise. It is important to note that by design AR process is not necessarily stationary. The AR process can be more efficiently represented with the help of time lag operator L. The operator is defined as

$$Lx_i = x_{i-1} \forall i \tag{11.2}$$

and for kth order lag,

$$L^k x_i = x_{i-k} \forall i \tag{11.3}$$

Using this operator the AR model can now be defined as

$$\left(1 - \sum_{j=1}^{p} \alpha_j L^j\right) x_i = \epsilon_i \tag{11.4}$$

11.3.2 Moving Average, or MA Process

A moving average process is always stationary by design. A moving average or *MA* process is a process in which the current output is a moving average of certain number of past states of the default white noise process. We can define a moving average process of order q, MA(q) as

$$x_i = \epsilon_i + \beta_1 \epsilon_{i-1} + \cdots + \beta_q \epsilon_{i-q} \tag{11.5}$$

which can be written using lag operator as

$$x_i = \left(1 + \sum_{k=1}^{q} \beta_k \cdot L^k\right) \epsilon_i \tag{11.6}$$

β_i are the coefficients or parameters of the MA process.

11.3.3 *Autoregressive Moving Average* ARMA *Process*

Now, we can combine the two processes, into a single ARMA(p, q) process with parameters p and q as

$$x_i = \alpha_1 x_{i-1} + \cdots + \alpha_p x_{i-p} + \epsilon_i + \beta_1 \epsilon_{i-1} + \cdots + \beta_q \epsilon_{i-q} \tag{11.7}$$

or using lag operator as

$$\left(1 - \sum_{j=1}^{p} \alpha_j L^j\right) x_i = \left(1 + \sum_{j=1}^{q} \beta_j L^j\right) \epsilon_i \tag{11.8}$$

11.4 Autoregressive Integrated Moving Average (ARIMA) Models

Although ARMA(p, q) process in general can be non-stationary, it cannot explicitly model a non-stationary process well. This is why the ARIMA process is developed. The added term *integrated* adds differencing terms to the equation. Differencing operation as the name suggested computes the deltas between consecutive values of the outputs as

$$x_d(i)^1 = x(i) - x(i-1) \tag{11.9}$$

Equation 11.9 shows the first order differences. Differencing operation in discrete time is similar to differentiation or derivative operation in continuous time. First order differencing can make polynomial based second order non-stationary processes into stationary ones, just like differentiating a second order polynomial equation leads to a linear equation. Processes with higher polynomial order non-stationarity need higher order differencing to convert into stationary processes. For example second order differencing can be defined as,

$$x_d(i)^2 = x_d(i)^1 - x_d(i-1)^1 \tag{11.10}$$

$$x_d(i)^2 = (x(i) - x(i-1)) - (x(i-1) - x(i-2)) \tag{11.11}$$

$$x_d(i)^2 = x(i) - 2x(i-1) + x(i-2) \tag{11.12}$$

and so on. Using the lag operator the same differencing operations can be written as

$$x_d(i)^1 = (1 - L)x_i \tag{11.13}$$

and

$$x_d(i)^2 = (1 - L)^2 x_i \tag{11.14}$$

This can be quickly generalized into any arbitrary order r as

$$x_d(i)^r = (1 - L)^r x_i \tag{11.15}$$

Now, we are ready to give the equation of the ARIMA process. ARIMA process adds the differencing operation along with AR and MA operations and let the parameter associated with the differencing operations be r. Thus ARIMA(p, q, r) process can be defined as

$$\left(1 - \sum_{j=1}^{p} \alpha_j L^j\right)(1 - L)^r x_i = \left(1 + \sum_{j=1}^{q} \beta_j L^j\right)\epsilon_i \tag{11.16}$$

Thus ARIMA process generalizes the ARMA process for the cases with non-stationarity. When the value of r is 0, the ARIMA process reduces to ARMA process. Similarly, when r and q are 0, the ARIMA process reduces to AR process and when r and p are 0, it reduces to MA process and so on.

11.5 Hidden Markov Models (HMM)

Hidden Markov models or HMMs represent a popular generative modelling tool in time series analysis. The HMMs have evolved from Markov processes in statistical signal processing. Consider a statistical process generating a series of observations represented as $y_1, y_2, \ldots, y_k$. The process is called as a Markov process if the current observation depends only on the previous observation and is independent of all the observations before that. Mathematically it can be stated as

$$y_{k+1} = F(y_k) \tag{11.17}$$

where F is the probabilistic Markov function.

In HMM, there is an additional notion of states. Consider Fig. 11.4. The states are shown as s_{k-1}, s_k, and s_{k+1} and corresponding outcomes or observations are shown as y_{k-1}, y_k, and y_{k+1}. The states follow Markov property such that each state is dependent on the previous state. The outcomes are probabilistic functions of only the corresponding states. The HMMs further assume that the states, although they exist, are invisible to the observer. The observer can only see the series of outcomes, but cannot know or see the actual states that are generating these outcomes. Mathematically it can be stated using two equations as

$$s_{k+1} = F_s(s_k) \tag{11.18}$$

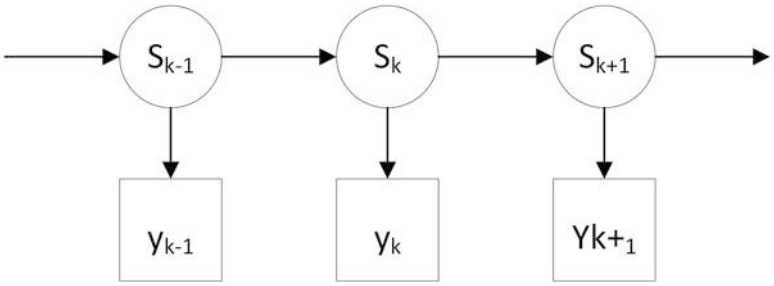


Fig. 11.4 A sequence of states and outcomes that can be modelled using hidden Markov models technique

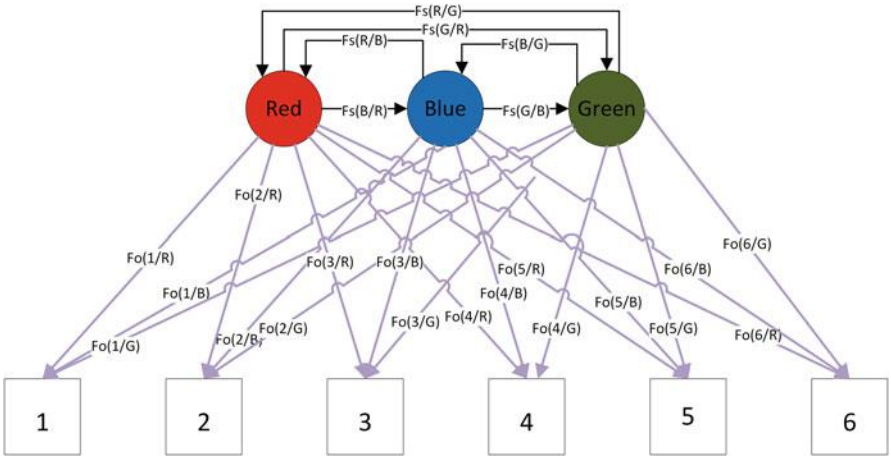


Fig. 11.5 Showing a full HMM with three states as three different die: red, blue, green and six outcomes as 1,2,3,4,5,6

where F_s is the probabilistic function of state transitions.

$$y_{k+1} = F_o(s_{k+1}) \tag{11.19}$$

where F_o is the probabilistic function of observations.

Consider a real life example with three different states represented by three dies: red, blue, and green. Each die is biased differently to produce the outcomes of $(1, 2, 3, 4, 5, 6)$. The state transition probabilities are given by $F_s(s_i/s_j)$ and outcome probabilities are given as $F_o(s_i/o_j)$. Figure 11.5 shows the details of the model.

Once a given problem is modelled using HMM, there exist various techniques to solve the optimization problem using training data and predict all the transition and outcome probabilities [41].

11.5.1 Applications

HMMs have been widely used to solve the problems in natural language processing with notable success. For example part of speech (POS) tagging, speech recognition, machine translation, etc. As generic time series analysis problems, they are also used in financial analysis, genetic sequencing, etc. They are also used with some modifications in image processing applications like handwriting recognition.

11.6 Conditional Random Fields (CRF)

Conditional random fields or CRFs represent a discriminative modelling tool as opposed to HMM which is a generative tool. CRFs were introduced by Lafferty et al. [69] in 2001. In spite of having a fundamental different perspective, CRFs share a significant architecture with HMM. In some ways CRFs can be considered as generalization of HMMs and logistic regression. As generative models typically try to model the structure and distribution of each participating class, discriminative models try to model the discriminative properties between the classes or the boundaries between the classes. As HMMs try to model the state transition probabilities first and then the outcome or observation probabilities based on states, CRFs directly try to model the conditional probabilities of the observations based on the assumptions of similar hidden states. The fundamental function for CRF can be stated as

$$\hat{y} = \arg\max y P(y/\mathbf{X}) \tag{11.20}$$

In order to model the sequential input and states, CRF introduces feature functions. The feature function is defined based on four entities.

Entities in Feature Function

1. Input vectors $\mathbf{X}$.
2. Instance i of the data point being predicted.
3. Label for data point at $(i-1)$th instance, l_{i-1}.
4. Label for data point at (i)th instance, l_i.

The function is then given as

$$f(\mathbf{X}, i, l_{i-1}, l_i) \tag{11.21}$$

Using this feature function the conditional probability is written as

$$P(y/\mathbf{X}, \lambda) = \frac{1}{Z(\mathbf{X})} \exp\left(\sum_{i=1}^{n} \sum_{j} \lambda_j f_i(\mathbf{X}, i, y_{i-1}, y_i)\right) \tag{11.22}$$

where the normalization constant $Z(\mathbf{X})$ is defined as

$$Z(\mathbf{X}) = \sum_{\hat{y} \in y} \sum_{i=1}^{n} \sum_{j} \lambda_j f_i(\mathbf{X}, i, \hat{y_{i-1}}, \hat{y_i}) \tag{11.23}$$

11.7 Conclusion

Time series analysis is an interesting area in the field of machine learning that deals with data that is changing with time. The entire thought process of designing a time series pipeline is fundamentally different than the one used in all the static models that we studied in previous chapters. In this chapter, we studied the concept of stationarity followed by multiple different techniques to analyze and model the time series data to generate insights.

Chapter 12
Deep Learning

12.1 Introduction

In his famous book, *"The nature of Statistical Learning Theory:"* [37], Vladimir Vapnik stated (in 1995) the four historical periods in the area of learning theory as:

1. Constructing first learning machine
2. Constructing fundamentals of the theory
3. Constructing the neural network
4. Constructing alternatives to the neural network

The very existence of the fourth period was a failure of neural networks. Vapnik and others came up with various alternatives to neural network that were far more technologically feasible and efficient in solving the problems at hand at the time and neural networks slowly became history. However, with the modern technological advances, neural networks are back and we are in the fifth historical period: "Re-birth of neural network."

With re-birth, the neural networks are also renamed into *deep neural networks* or simply *deep networks*. The learning process is called as *deep learning*. However, deep learning is probably one of the most widely used and misused term. Since inception, there were no sufficient success stories to back up the over-hyped concept of neural networks. As a result they got pushed back in preference to more targeted ML technologies that showed instant success, e.g., evolutionary algorithms, dynamic programming, support vector machines, etc. The concept of neural networks or artificial neural networks (ANNs) was simple: to replicate the processing methodology of human brain which is composed of connected network of neurons. ANN was proposed as a generic solution to any type of learning problem, just like human brain is capable of learning to solve any problem. The only blocker was that at the time, the processor architecture and manufacturing processes were not sufficiently mature to tackle the gargantuan processing required

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_12

to realistically simulate human brain. Average human brain has roughly 100 billion neurons and over 1000 trillion synaptic connections [1]!! This would compare to a processor with trillion calculations/second processing power backed by over 1000 terabytes of hard drive. Even in 2018 AD, this configuration for a single computer is far-fetched.[1] The generic learning of ANNs can only converge to sufficient accuracy if it is trained with sufficient data followed by corresponding level of computation. It was just beyond the scope of technology in 1990s, and as a result they were not able to realize the full potential of the technology.

With the advent of graphics processor based computation (called as general purpose graphics processing unit or GPGPU) made popular by NVIDIA in the form of CUDA library, the technology started coming close to realizing the potential of ANNs. However, due to the stigma associated with original ANN, the newly introduced neural networks were called as deep neural networks (DNN) and the ML process was called deep learning. In essence there is no fundamental difference between an ANN of the 1990s and deep networks of the twenty-first century. However, there are some more differences that are typically assumed when one talks about deep networks. The original ANNs were primarily used for solving problems with a very small amount of data and very narrow scope of application and consisted of handful of nodes and 1–3 layers of neurons at best. Based on the hardware limitations, this was all that could have been achieved. The deep networks typically consist of hundreds to thousands of nodes per layer and number of layers can easily exceed 10. With this added complexity of order of magnitude, the optimization algorithms have also drastically changed. One of the fundamental changes in the optimization algorithms is heavier use of parallel computation. As the GPGPUs present many hundreds or even thousands of cores that can be used in parallel, the only way to improve the computation performance is to parallelize the training optimization of the deep networks [2]. The deep learning framework is not limited to supervised learning, but some of the unsupervised problems can also be tackled with this technique.

12.2 Origin of Modern Deep Learning

Although the current state of deep learning is made possible by the advances in computation hardware, a lot of efforts have been spent on the optimization algorithms that are needed to successfully use these resources and converge to the optimal solution. It is hard to name one person who invented deep learning, but there are few names that are certainly at the top of list. Seminal paper by

[1]The fastest supercomputer in the world is rated at 200 petaflops, or 200,000 trillion calculations/second. It is backed by storage of 250 petabytes or 250,000 terabytes. So it does significantly better than a single human brain, but at the cost of consuming 13 MW of power and occupying entire floor of a huge office building [18].

Geoffrey Hinton titled *A fast learning algorithm for deep belief networks* [54] truly ushered the era of deep learning. Some of the early applications of deep learning were in the field of acoustic and speech modeling and image recognition. Reference [53] summarizes the steps in the evolution of deep learning as we see today concisely. Typically when one refers to deep learning or deep networks they either belong to convolutional networks or recurrent networks or their variations. Surprisingly enough, both of these concepts were invented before the deep earning. Fukushima introduced convolutional networks back in 1980 [55], while Michael Jordan introduced recurrent neural networks in 1986 [56].

In the following sections we will discuss architectures of convolutional and recurrent deep neural networks.

12.3 Convolutional Neural Networks (CNNs)

Convolutional neural networks or *CNN*s are based on the convolution operation. This operation has its roots in signal processing and before going into details of CNN, let us look at the convolution process itself.

12.3.1 1D Convolution

Mathematically, convolution defines a process by which one real valued function operates on another real valued function to produce new real valued function. Let one real valued continuous function be $f(t)$ and other be $g(t)$, where t denotes continuous time. Let the convolution of the two be denoted as $s(t)$. Convolution operation is typically denoted as $*$.

$$f(t) * g(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{12.1}$$

Also, convolution is a commutative operation meaning, $(f * g)$ is same as $(g * f)$,

$$(f * g)(t) = (g * f)(t) = \int_{-\infty}^{\infty} g(\tau)f(t-\tau)d\tau \tag{12.2}$$

The same equations can also be written for discrete processes, that we typically deal with in machine learning applications. The discrete counterparts of the two equations can be written as $f(k)$ and $g(k)$, where k denoted a discrete instance of time.

$$f(k) * g(k) = (f * g)(k) = \sum_{\delta=-\infty}^{\infty} f(\delta)g(k-\delta) \tag{12.3}$$

and

$$(f * g)(k) = (g * f)(k) = \sum_{\delta=-\infty}^{\infty} g(\delta) f(k - \delta) \tag{12.4}$$

These definitions are very similar to the definition of correlation between two functions. However, the key difference here is the sign of δ in case of discrete convolution and τ in case of continuous convolution is opposite between f and g. If the sign is flipped, the same equations would represent correlation operation. The sign reversal makes one function reversed in time before being multiplied (point-wise) with the other function. This time reversal has far reaching impact and the result of convolution is completely different than the result of correlation. The convolution has also very interesting properties from the perspective of Fourier transform [8] in frequency domain and they are heavily used in signal processing applications.

12.3.2 2D Convolution

Above equations represent 1D convolution. This is typically used in speech applications, where the signal is 1D. Concept of convolution can also be applied in 2-dimensions as well, which makes it suitable for applications on images. In order to define 2D convolution, let us consider two images A and B. The 2D convolution can now be defined as,

$$(A * B)(i, j) = \sum_{m} \sum_{n} A(m, n) B(i - m, j - n) \tag{12.5}$$

Typically the second image B is a small 2D kernel compared to first image A. The convolution operation transforms the original image into another one to enhance the image in some desired way.

12.3.3 Architecture of CNN

Figure 12.1 shows the architecture of CNN. The building block of CNN is composed of three units:

1. Convolution layer
2. Rectified linear unit, also called as ReLU
3. Pooling layer

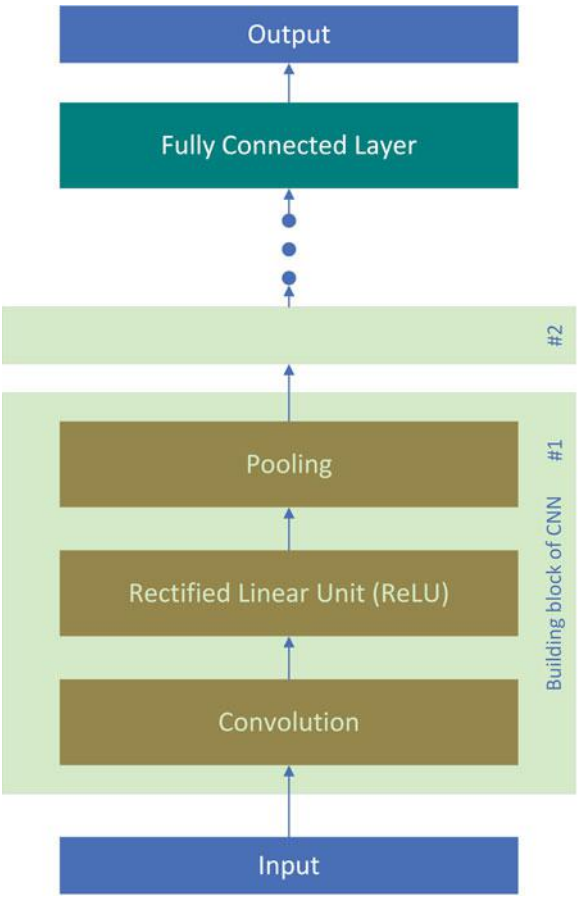


Fig. 12.1 Architecture of convolutional neural network. The building block of the network contains the convolution, ReLU followed by pooling layer. A CNN can have multiple such layers in series. Then there is a fully connected layer that generates the output

12.3.3.1 Convolution Layer

The convolution layer consists of a series of 2D kernels. Each of these kernels is applied to original figure using 2D convolution defined in Eq. 12.5. This generates a 3D output.

12.3.3.2 Rectified Linear Unit (ReLU)

Rectified linear unit of ReLU, as the name suggests, rectifies the output of convolutional layer to convert all the negative values to 0. The function is defined as,

$$f(x) = \max(0, x) \tag{12.6}$$

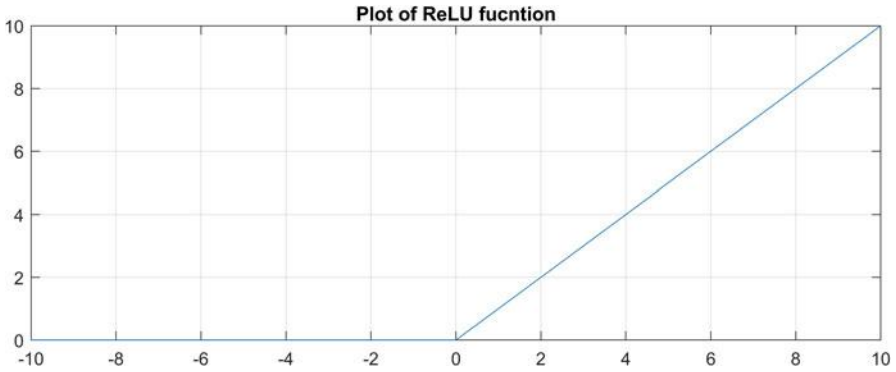


Fig. 12.2 Plot of the rectified linear unit (ReLU)

The ReLU function is shown in Fig. 12.2. This layer also introduces nonlinearity into the network, which is otherwise linear. This layer does not change the dimensionality of the data. *Sigmoid* or *tanh* functions were used earlier to model the nonlinearity in a more continuous way, but it was observed that use of simpler ReLU function is just as good and it also improves the computation speed of the model and also fixes the vanishing gradient problem [57].

12.3.3.3 Pooling

Pooling layer performs down-sampling by replacing larger sized blocks with single value. Most commonly used pooling method is called *Max Pooling*. In this method simply the maximum value of the block is used to replace the entire block. This layer reduces the dimensionality of the data flowing through the network drastically while still maintaining the important information captured as a result of convolution operations. This layer also reduces overfitting.

These three layers together form the basic building block of a CNN. Multiple such blocks can be employed in single CNN.

12.3.3.4 Fully Connected Layer

The previously described layers essentially target certain spatial parts of the input and transform them using the convolutional kernels. The fully connected layers bring this information together in traditional MLP manner to generate the desired output. It typically uses softmax activation function as additional step to normalize the output. Let the input to the softmax be a vector y of dimensions n. Softmax

function is defined as,

$$\sigma(y)_j = \frac{e^{y_j}}{\sum_{i=1}^{n} e^{y_j}} \tag{12.7}$$

Softmax function normalizes the output so that it sums to 1.

12.3.4 Training CNN

CNNs are typically trained using stochastic gradient descent algorithm. Here are the main steps.

1. Gather the sufficient labelled training data.
2. Initialize the convolution filter coefficients weights.
3. Select a single random sample or a mini-batch of samples and pass it through the network and generate the output (class label in case of classification or real value in case of regression).
4. Compare the network output with the expected output and then use the error along with backpropagation to update filter coefficients and weights at each layer.
5. Repeat steps 3 and 4 till algorithm converges to desired error levels.
6. In order to tune hyperparameters (which can be, "size of convolution kernels", "number of convolution blocks", etc.) repeat the entire training process multiple times for different set of hyperparameters and finally choosing the ones that perform best.

12.4 Recurrent Neural Networks (RNN)

All the traditional neural networks as well as CNNs are static models as defined in Chap. 2. They work with data that is already gathered and that does not change with time. Recurrent neural networks or *RNN*s propose a framework for dealing with dynamic or sequential data that changes with time. RNNs exhibit a concept of *state* that is a function of time. Classical RNNs, sometimes called as fully recurrent networks have architecture very similar to MLP, but with addition of a feedback of current state as shown in Fig. 12.3. Thus output of the RNN can be given as,

$$y_t = f_y(V \cdot H_t) \tag{12.8}$$

The state of RNN is updated at every time instance using input and previous state as,

$$H_t = f_H(U \cdot X_t + W \cdot H_{t-1}) \tag{12.9}$$

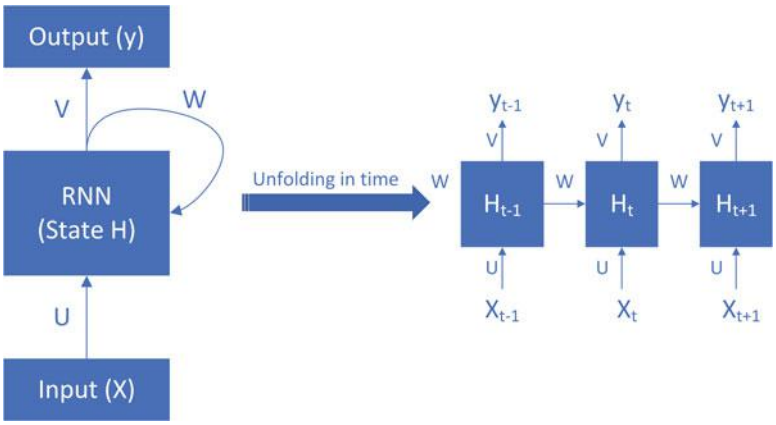


Fig. 12.3 Architecture of classic or fully recurrent RNN. The time unfolded schematic shows how the state of RNN is updated based on sequential inputs

The training of the RNN would be carried out using backpropagation in similar fashion using a labelled input sequence for which the output sequence is known.

12.4.1 Limitation of RNN

RNNs were successful in modeling time series data that was not possible to model using traditional neural networks that dealt with static data. However, when the sequence of data to be analyzed was long with varying trends and seasonalities, the RNNs were not able to adapt to these conditions due to problems of exploding and vanishing gradients and/or oscillating weights, etc. [58]. The learning algorithms appeared to reach a limit after certain number of samples were consumed and any updates after that point were minuscule and not really changing the behavior of the network. In some cases, due to high volatility of the changes in training samples, the weights of the network kept on oscillating and would not converge.

12.4.2 Long Short-Term Memory RNN

Hochreiter and Schmidhuber proposed an improvement to the standard RNN architecture in the form of long short-term memory RNN or *LSTM-RNN*. This architecture improved the RNNs to overcome most of their limitations are discussed in previous section. The fundamental change brought by this architecture was the use of forget gate. The LSTM-RNN had all the advantages of RNN and much reduced limitations. As a result most of modern RNN applications are based

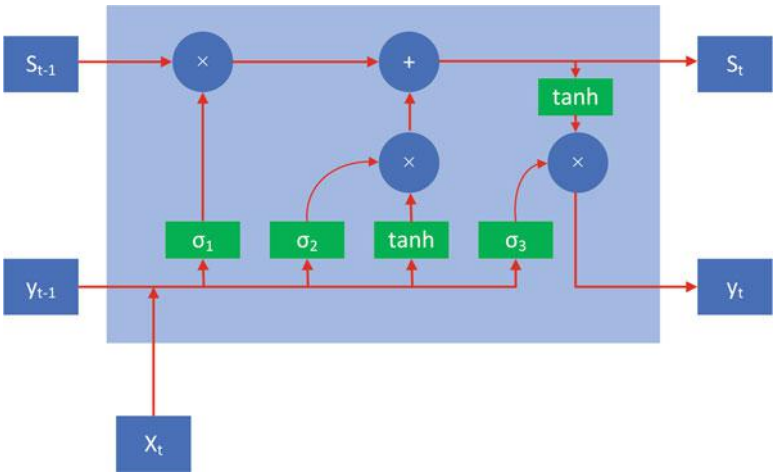


Fig. 12.4 Architecture of LSTM-RNN. σ_1 denotes the forget gate, σ_2 denotes the input gate, and σ_3 denotes the output gate

on LSTM architecture. Figure 12.4 shows the components of LSTM-RNN. The architecture is quite complex with multiple gated operations. In order to understand the full operation, let us dive deeper into the signal flow. S_t denotes the state of the LSTM at time t. y_t denotes the output of the LSTM at time t. X_t denotes the input vector at time t. σ_1, σ_2, and σ_3 denotes the forget gate, input gate and, output gate, respectively. Let us look at the operations at each gate.

12.4.2.1 Forget Gate

At forget gate the input is combined with the previous output to generate a fraction between 0 and 1, that determines how much of the previous state need to be preserved (or in other words, how much of the state should be forgotten). This output is then multiplied with the previous state.

12.4.2.2 Input Gate

Input gate operates on the same signals as the forget gate, but here the objective is to decide which new information is going to enter the state of LSTM. The output of the input gate (again a fraction between 0 and 1) is multiplied with the output of *tanh* block that produces the new values that must be added to previous state. This gated vector is then added to previous state to generate current state.

12.4.2.3 Output Gate

At output gate the input and previous state are gated as before to generate another scaling fraction that is combined with the output of *tanh* block that brings the current state. This output is then given out. The output and state are fed back into the LSTM block as shown.

12.4.3 Advantages of LSTM

The specific gated architecture of LSTM is designed to improve all the following shortcomings of the classical RNN:

1. Avoid the exploding and vanishing gradients, specifically with the use of forget gate at the beginning.
2. Long term memories can be preserved along with learning new trends in the data. This is achieved through combination of gating and maintaining state as separate signal.
3. A priori information on states is not required and the model is capable of learning from default values.
4. Unlike other deep learning architectures there are not many hyperparameters needed to be tuned for model optimization.

12.4.4 Current State of LSTM-RNN

LSTM is one of the important areas in cutting edge of machine learning and many new variants of the model are proposed including bi-directional LSTM, continuous time LSTM, hierarchical LSTM, etc.

12.5 Conclusion

We studied deep learning techniques in this chapter. The theory builds on the basis of classical perceptron or single layer neural network and adds more complexity to solve different types of problems given sufficient training data. We studied two specific applications in the form of convolutional neural networks and recurrent neural networks.

Chapter 13
Emerging Trends in Machine Learning

13.1 Introduction

In the second decade of the twenty-first century, the field of machine learning has taken quantum leaps of progress compared to its much slower evolution over last decades. Deep learning is certainly at the core of this explosive growth; however, there are many novel approaches and techniques that have surfaced and without discussing them, the topic of this book would remain unfinished. Most of these techniques are in their infancy but have shown promising results, while some techniques have already matured. The discussion here is fairly superficial to introduce these techniques. In order to have deeper understanding, references are provided.

13.2 Transfer Learning

The machine learning process can be mathematically represented in the form of equations. However, the intermediate stages in learning of a specific system, especially a deep learning system cannot be expressed or interpreted in words. During the process of training, there are lot of changes happening in values of certain variables or some new variables are being created. This learning of parameters is quite abstract. When something cannot be expressed in terms of words, it is hard to interpret. Without interpretation, it is hard to translate. However, in many situations, the learnings from one problem can be very useful in tackling some other problems that can be superficially quite different. Or in some other cases, we need to use the system that is effectively trained in one domain (e.g., an NLP system trained on English language) needs to be adapted to work in another similar domain, where sufficient training data is not available (e.g., some African language). This is a crucial aspect of human brain, where it is exceedingly smart in adapting

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_13

such domains. We often apply abstract learnings from one experience into another without even realizing the complexity of it. Transfer learning techniques (sometimes also called as domain adaptations) try to mimic this property of human brain into transferring learnings from one system of deep network into another [36, 70], in spite of their abstractness. Specifically with popularity of deep learning, this topic is gaining traction.

13.3 Generative Adversarial Networks (GANs)

We discussed generative models in Chap. 8. Generative models typically take a more holistic approach towards understanding and solving the problem, by trying to model the structure and distribution of the data. *Generative Adversarial Networks* or *GANs* are combination of two neural networks as suggested in the name: generative network and adversarial network. The generative network is designed to imitate the process that generates the sample data, while adversarial network is a type of discriminative network that checks if the data generated by the generative network belongs to the desired class. Thus these two networks working together can generate any type of synthetic data. In order to better understand the architecture of GANs, let's take an example of image synthesis. We would like to generate images of human faces that look like faces of real people, but are completely computer generated. Figure 13.1 shows the architecture of GAN to solve this problem. The generative network is given input as random noise to generate a synthetic face. The generated faces are compared with database of real human face images by the discriminative network to generate the classification. The output of the discriminative network is fed back to train both the networks. With availability of sufficiently large labelled database of human face images, both the networks can be trained. Once the system is trained, it is capable of artificially generating human like faces. GANs represent one of the most novel discoveries in the field of machine learning, specifically in the field of deep learning that was introduced in 2014 by Goodfellow et al. [43].

13.4 Quantum Computation

Quantum computation has been seen as a potential source of ultra-fast computation since early 1080s. Nobel Laureate, Professor Richard Feynman in 1959, hinted about the possibility of quantum computation based on quantum entanglement in his speech given in Annual Physics Society's meeting. However, that thought got unnoticed for couple of decades. In 1980s, the simultaneous papers by Beniof [61] and Manin [14] marked the beginning of quantum computation. Traditional computing chips that are at the core of *PC* or *Mac* or all the mobile devices as well as the *GPU*s are made from silicon with various different manufacturing processes.

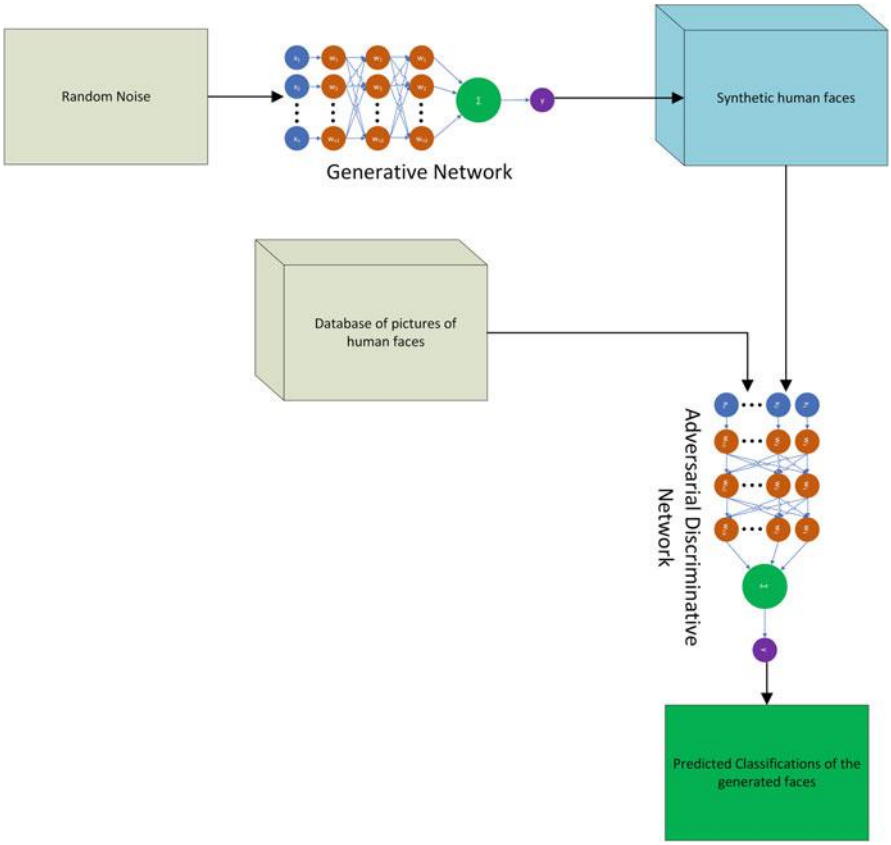


Fig. 13.1 Architecture of generative adversarial network

Thus, they all feature different architectures and power consumption levels, at the core they are all built upon same type of hardware. Quantum computer introduces an entirely new type of computation device that is based on the quantum properties of subatomic particles. Specifically, quantum entanglement and superposition.

13.4.1 Quantum Theory

In general quantum theory in physics introduced some fundamental new ways to think about interactions of subatomic particles in twentieth century. Classical physics always relied to continuity and predictability at all levels of mass and space and time, while quantum physics essentially proved it wrong at subatomic levels. It proposed that when dealing distances and masses that are really small, the continuity

as well as predictability ceases to exist and different rules become applicable. The rules of quantum physics are based on four primary assumptions.

Fundamental Assumptions in Quantum Theory

1. The energy is quantized. In other words, it means that after reducing the energy to certain level, we reach the quantum of energy that cannot be further divided.
2. The mass is quantized in the same way as energy.
3. Particle–wave duality. At subatomic levels, the particles exhibit particle type and wave type behavior at the same time.
4. Uncertainty principle. This was probably one of the most ground breaking discoveries of quantum physics attributed to Heisenberg [15]. The principle states that the product of uncertainty of position and momentum of a particle is greater than or equal to a fixed number defined by *Planck's constant*. Thus, if you try to find the precise position of the particle, the momentum or velocity of the particle becomes quite unpredictable and vice versa. Classical physicists hated this idea, but ultimately had to agree after sufficient evidence.

Quite a few interpretations of quantum theory contradict with classical theory of physics including Einstein's equations, but nonetheless, both theories hold true in their own domains. It is still an active area of research to unify these two theories into a single grand theory of physics.

Although quantum theory is quite fascinating in general, we will focus on the specific areas of quantum physics that are relevant to the concept of quantum computation: quantum entanglement and superposition.

13.4.2 Quantum Entanglement

Quantum entanglement can be described in short as the phenomenon where two particles or a group of particles possibly separated by large distances[1] exhibit correlation in physical properties like momentum, spin, etc. The particles are said to be in state of entanglement. Although this behavior can lead to endless series of paradoxes if applied to large objects, the quantum entanglement of subatomic particles (e.g., photons, electrons, neutrinos) has been proved beyond doubt [16].

[1]Large distances in quantum context mean the distances that are long enough that light takes noticeable time to travel that distance. Noticeable time would imply multiple seconds or a time that can be accurately measured by modern timekeeping devices beyond statistical uncertainty.

13.4.3 Quantum Superposition

Quantum superposition is a direct implication of principle of duality. As two waves can be added or superposed in classical physics, the quantum states of two particles can be superposed in quantum mechanics. Thus any quantum state of a subatomic particle can be represented as a linear combination of two or more quantum states [17].

13.4.4 Computation with Quantum Particles

As the traditional computer works on the smallest chunk of information as bits, (which can have a binary value 0 or 1) the smallest chunk of information in quantum computer is called as quantum bit or *qubit*. Each qubit is represented as a quantum state of a subatomic particle. With quantum superposition, an n qubit quantum computer can represent 2^n n bit numbers at the same time, while an n bit traditional computer can only represent one such number. Quantum entanglement can control different quantum particles driving the quantum computer to change their states in sync as needed. Using these properties, a quantum computer is posed to compute parallel operations at unprecedented levels. However, the problems that need very long sequential computation are not really benefitted with this approach. Also, theoretically adding more qubits can arbitrarily increase the parallel computation power of a quantum computer, there are number of practical limitations that are constraining the scope of quantum computing. However, assuming these limitations are somehow overcome, the quantum computers stand to take the machine learning applications to levels of sophistication, one can only dream as of now.

13.5 AutoML

AutoML is another emerging trend in the area of machine learning and artificial intelligence. As the name suggests, it means automating the machine learning for any problem. We have studied the series of sequential steps that are needed to build a successful machine learning pipeline to solve a given problem at hand. Quite a few steps in this sequence need domain knowledge of the problem as well as theoretical underpinnings of the algorithms being used. It is typically hard to get resources with this unique combination of experience. AutoML [31] concept tries to automate the entire pipeline of machine learning problem by building a machine learning system on top of machine learning system. If implemented successfully as per its grand inspirations, AutoML promises to build an entire machine learning pipeline and optimize it automatically.

The concept of AutoML is relatively in its infancy and is a hot topic of research. The primary areas of automation can be broken down into following categories.

Areas of Automation Targeted Under AutoML

1. Automating the feature engineering. Once all the raw data is gathered and target problem defined in the form of desired metrics, the AutoML system would automatically perform feature selection or transformation as suitable and optimal for achieving the best possible results.
2. Automating the choice of algorithm. For example if we are trying to solve a problem that requires multiclass classification as primary underlying model, the AutoML system would be capable of automatically iterating through certain predefined algorithms like decision trees, or neural networks or support vector machine and come up with the best suited algorithm.
3. Automating the hyperparameter tuning. When the algorithm is auto-selected, the AutoML system can take it further by auto-initializing the hyperparameters followed by auto-defining the optimal parameter grid and finding the optimal set of hyperparameters for the algorithm.
4. Automating the system optimization. Evaluating the results of the tuned model, the AutoML system can iterate on it as desired and then optimize the performance of the system for given hardware resources.

Each of these topics is quite an ambitious goal in itself and currently there are no systems available that can claim to have solved either of them to sufficient level of performance. However, the progress in the area is quite exciting and looks promising [32].

13.6 Conclusion

In this chapter, we looked at some of the emerging trends in the field of machine learning. This list is just a tip of an iceberg where the theory of machine learning is exploding along variety of different dimensions. However, some of the concepts listed here would give the reader a taste of what is coming up.

Chapter 14
Unsupervised Learning

14.1 Introduction

Unsupervised learning methods deal with problems where the labelled data is not available. There is no form of supervised feedback to be gained about the processing that happens. Absence of labelled samples marks a fundamental shift of thought from the other supervised methods discussed so far in the book. It might also appear that such processing might not yield any useful information and one could not be more wrong. There are many situations where unsupervised methods are extremely valuable. The first and foremost is cost of labeling. In many situations, having all the training data fully labelled can be rather expensive and practically impossible and one needs to work with only small set of labelled data. In such cases, one can start with supervised methods using small set of labelled data and then grow the model in unsupervised manner on larger set of data. In other cases, the labels may not be available at all, and one needs to understand the structure and composition of data by doing exploratory unsupervised analysis. Understanding the structure of the data can also help with choosing the right algorithm or better set of initialized parameters for the algorithm in cases where supervised methods will be ultimately used. In general, unsupervised learning marks an important pillar of modern machine learning.

In this chapter, we will learn about the following aspects of unsupervised learning:

1. Clustering
2. Component Analysis
3. Self Organizing Maps (SOM)
4. Autoencoding neural networks

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_14

14.2 Clustering

Clustering is essentially aggregating the samples in the form of groups. The criteria used for deciding the membership to a group is determined by using some form of metric or distance. Clustering being one of the oldest methods in the machine learning repertoire, there are numerous methods described in the literature. We will focus on one of the simple yet quite effective method which with the help of certain modifications can tackle broad range of clustering problems. It is called as *K-means clustering*. The variable K denotes number of clusters. The method expects the user to determine the value of K before starting to apply the algorithm.

14.2.1 k-Means Clustering

Figure 14.1 shows two examples of extreme cases encountered in case of clustering. In top figure the data is naturally distributed into separate non-overlapping clusters, while the bottom figure shows the case where there is no natural separation of the data. Most cases encountered in practice are somewhere in between.

The k-means clustering algorithm can be summarized as follows:

1. Start with a default value of k, which is the number of clusters to find in the given data.
2. Randomly initialize the k cluster centers as k samples in training data, such that there are no duplicates.
3. Assign each of the training samples to one of the k cluster centers based on a chosen distance metric.
4. Once the classes are created, update the centers of each class as mean of all the samples in that class.
5. Repeat steps 2–4 until there is no change in the cluster centers.

The distance metric used in the algorithm is typically the Euclidean distance. However, in some cases different metrics like Mahalanobis distance, Minkowski distance, or Manhattan distance can be used depending on the problem at hand.

Figure 14.2 shows the algorithm in intermediate stages as it converges to desired clusters. This is somewhat an ideal case. In most practical situations, where the clusters are not well separated, or the number of naturally occurring clusters is different than the initialized value of k, the algorithm may not converge. The cluster centers can keep oscillating between two different values in subsequent iterations or they can just keep shifting from one set of clusters to another. In such cases multiple optimizations of the algorithms are proposed in the literature [42]. Some of the optimizations that are commonly used are as follows.

Optimizations for k-Means Clustering

1. Change the stopping criterion from absolute “no change” to cluster centers to allow for a small change in the clusters.
2. Restrict the number of iterations to a maximum number of iterations.

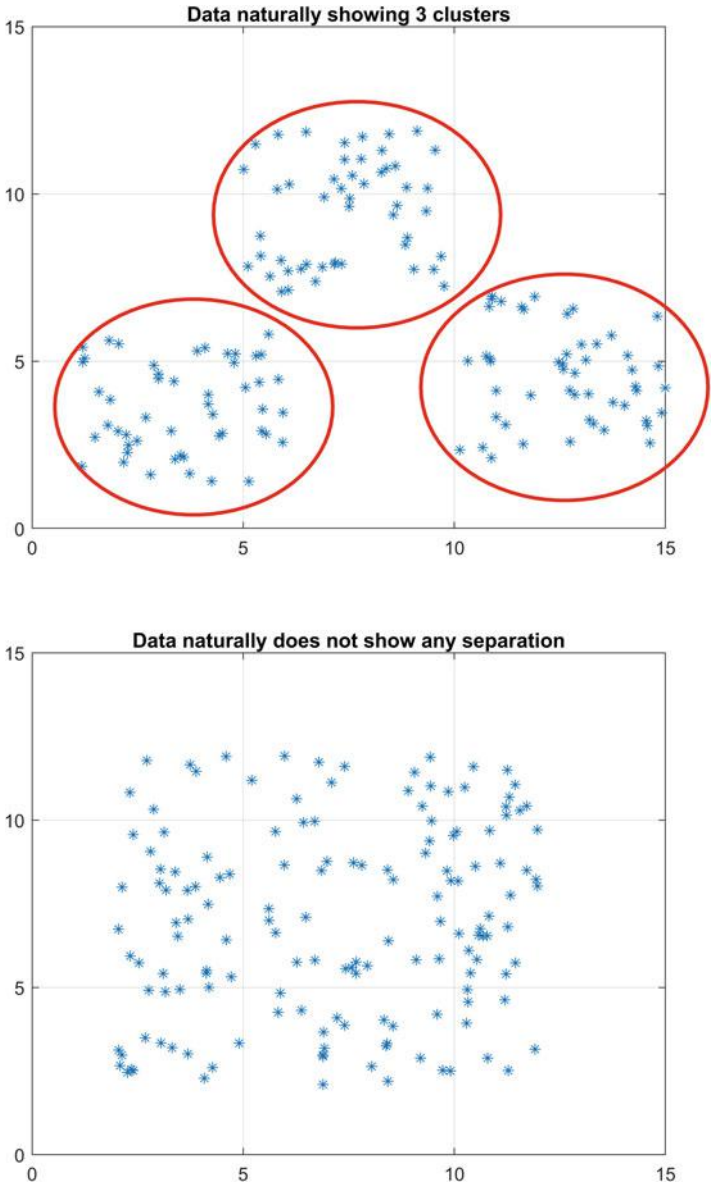


Fig. 14.1 Figures showing the extreme cases of data distribution for clustering

3. Find the number of samples in each cluster and if the number of samples is less than a certain threshold, delete that cluster and repeat the process.
4. Find the intra-cluster distance versus inter-cluster distance, and if two clusters are too close to each other relative to other clusters, merge them and repeat the process.

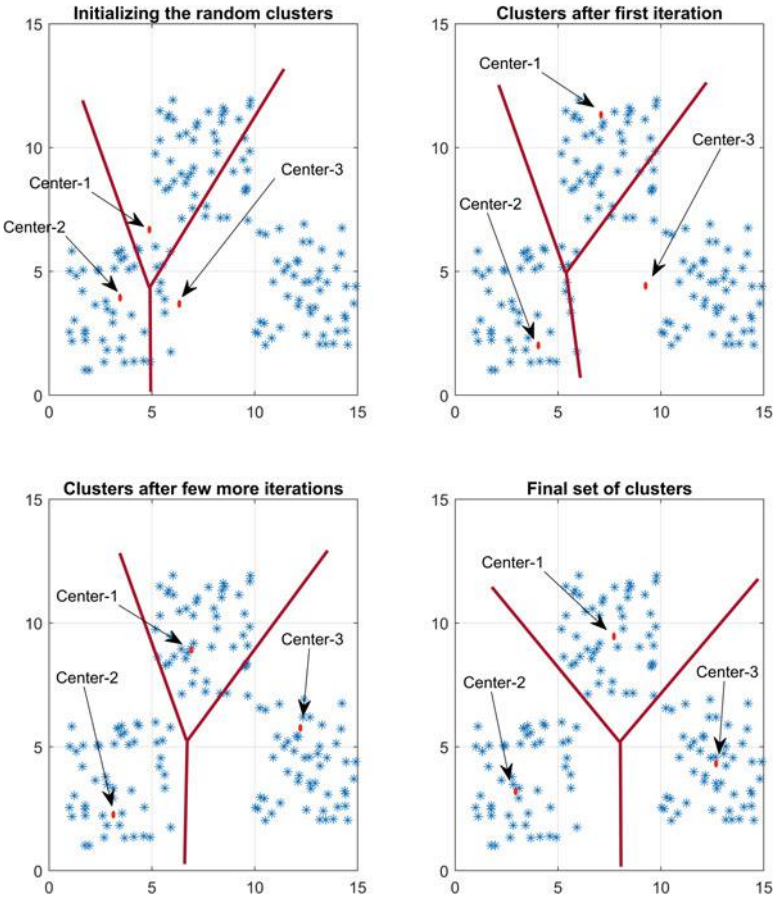


Fig. 14.2 Figures showing the progress of k-means clustering algorithm iterating through steps to converge on the desired clusters

5. If some clusters are getting too big, apply a threshold of maximum number of samples in a cluster and split the cluster into two or more clusters and repeat the process.

14.2.2 Improvements to k-Means Clustering

Even after the multiple optimizations are described in previous subsection, there are cases, when the results are still suboptimal and some further improvements can be applied.

14.2.2.1 Hierarchical *k*-Means Clustering

In some cases, using the same *k*-means clustering algorithm recursively can be helpful. After each successful completion of the clustering, new set of random clusters are initialized inside of each cluster and same algorithm is repeated in the subset of each cluster to find the sub-clusters. This is called as hierarchical k-means clustering method.

14.2.2.2 Fuzzy *k*-Means Clustering

In traditional *k*-means clustering algorithm after the cluster centers are chosen or updated, all the training samples are grouped into nearest cluster. Instead of using such absolute grouping, fuzzy *k*-means algorithm suggests a use of probabilistic grouping. In this case, each sample has non-zero probability of belonging to multiple clusters at the same time. The nearer the cluster, the higher the probability and so on.

14.3 Component Analysis

Another important aspect of unsupervised machine learning is dimensionality reduction. Component analysis methods are quite effective in this regard. Principal component analysis or PCA is one of the most popular techniques in dimensionality reduction in the theory of machine learning as we saw in Chap. 3. In this chapter we will look at another important technique similar to PCA called independent component analysis or ICA.

14.3.1 Independent Component Analysis (ICA)

Although PCA and ICA are both generative methods of extracting the core dimensionality of the data, they both differ in the underlying assumptions. PCA uses variation in the data and tries to model it to find the dimensions in ranked order where the variation is maximized. Matrix algebra and singular value decomposition (SVD) tools are used to find these dimensions. ICA takes a very different and more probabilistic approach towards finding the core dimensions in the data by making the assumption that given data is generated as a result of combination of a finite set of independent components. These independent components are not directly observable and hence sometimes referred to as latent components. Mathematically, the ICA can be defined for given data $(x_i), i = 1, 2, \ldots, n$ as,

$$x_i = \sum_{j=1}^{k} a_j s_j, \forall i \tag{14.1}$$

where a_j represent weights for corresponding k number of s_j independent components. The cost function to find the values of a_j is typically based on mutual information. The fundamental assumption in choosing the components is that they should be statistically independent with non-Gaussian distributions. Conceptually the process of finding ICA is quite similar to the topic of blind source separation in statistics. Typically, in order to make the predictions more robust a noise term is added into the Eq. 14.1.

14.4 Self Organizing Maps (SOM)

Self organizing maps, also called as self organizing feature maps present a neural network based unsupervised learning system, unlike other options discussed before. The neural networks are inherently supervised methods of learning, so their use in unsupervised class of methods is quite novel in that sense. SOMs define a different type of cost function that is based on similarity in the neighborhood. The idea here is to maintain the topological distribution of the data while expressing it in smaller dimensions efficiently. In order to illustrate the functioning of SOM, it will be useful to take an actual example. Top plot in Fig. 14.3 shows data distributed in 3-dimensions. This is a synthetically generated data with ideal distribution to illustrate the concept. The data is essentially a 2-dimensional plane folded into 3-dimensions. SOM unfolds the plane back into 2-dimensions as shown in the bottom figure. With this unfolding, the topological behavior of the original distribution is still preserved. All the samples that are neighbors in original distribution are still neighbors. Also, the relative distances of different points from one another are also preserved in that order.

The mathematical details of the cost function optimization for SOM can be found here [34]. The representation generated by SOM is quite useful and is generally better than first principal component as predicted by PCA analysis. However, PCA also provides multiple subsequent components to fully describe the variation in the data as needed and SOM lacks this capability. However, if one is only interested in efficient representation of the data SOM provides a powerful tool.

14.5 Autoencoding Neural Networks

Autoencoding neural networks or just autoencoders are a type of neural networks that work without any labels and belong to the class of unsupervised learning. Figure 14.4 shows architecture of autoencoding neural network. There is input layer matching the dimensionality of the input and hidden layer with reduced dimensionality followed by an output layer with the same dimensionality as input. The target here is to regenerate the input at the output stage. The network is trained to regenerate the input at the output layer. Thus the labels are essentially same as

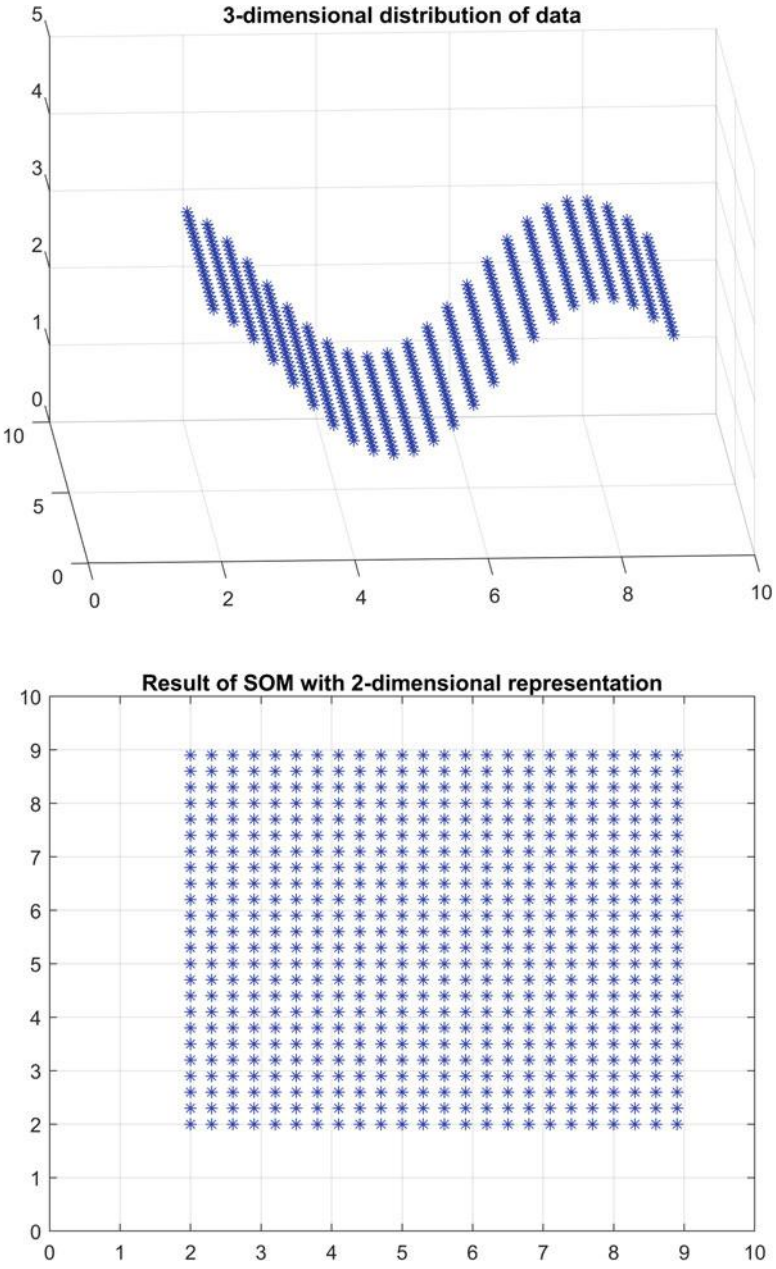


Fig. 14.3 Figures showing original 3-dimensional distribution of the data and its 2-dimensional representation generated by SOM. The SOM essentially unfolds 2-dimensional place folded into 3-dimensional space

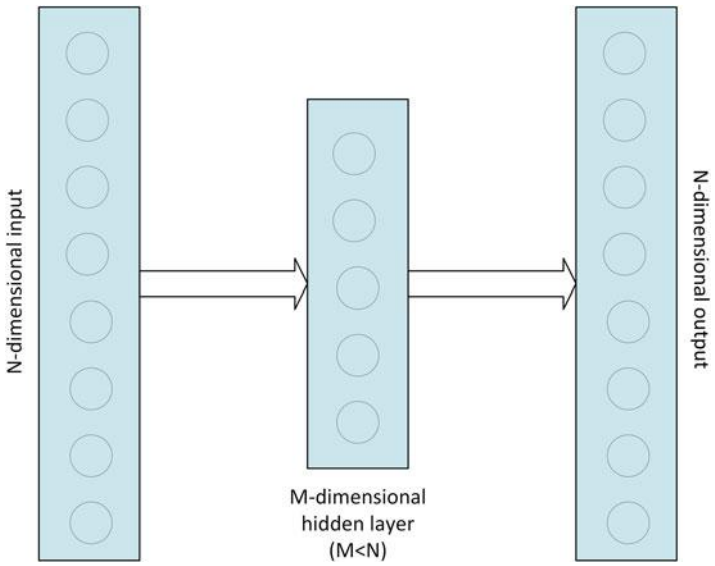


Fig. 14.4 Figure showing architecture of autoencoding neural network

input. The unique aspect of autoencoding networks is reduced dimensionality of the hidden layer. If an autoencoding network is successfully trained within the required error margins, then in essence we are representing the input in lesser dimensional space in the form of coefficients of the nodes at hidden layer. Furthermore, the dimensionality of the hidden layer is programmable. Typically, with the use of linear activation functions, the lower dimensional representation generated by the autoencoding networks resembles the dimensionality reduction obtained from PCA.

14.6 Conclusion

In this chapter, we discussed the new type of algorithms in machine learning literature known as unsupervised learning. These algorithms are characterized by learning without availability of labelled data. These algorithms belong to generative class of models and are used widely in many applications.

Part III
Building End to End Pipelines

David: Is this real or is it a game?
Joshua aka WOPR: What's the difference?

—dialog between David and Joshua aka WOPR, the supercomputer, "WarGames"

Part Synopsis

Building a full solution of a problem using machine learning techniques needs multiple components to be tied together in a sequential chain that operate on the data one after another to finally produce the desired results. In this part we will discuss the details of building an end to end machine learning pipeline using the algorithms described in the previous section.

Chapter 15
Featurization

15.1 Introduction

Building the end to end machine learning system is the topic of this part. Featurization or feature engineering or feature wrangling as it is called is typically the first step in this process. Although human brain typically deals with non-numeric information just as comfortably if not more than the pure numeric information, computers can only deal with numeric information. One of the fundamental aspects of feature engineering is to convert all the different features into some form of numerical features. However, as one can imagine the process is not trivial. In order to illustrate the process let us consider an example problem.

15.2 UCI: Adult Salary Predictor

UCI Machine Learning Repositoty is one of the well-known resources for finding sample problems in machine learning. It hosts multiple data sets targeting variety of different problems. We will use a problem listed there called as *Adult Data Set* [6]. This data set contains a multivariate data with features that present multiple challenges from the perspective of building end to end solutions. The data presented here contains information collected by census about the salaries of people from different work-classes, age groups, locations, etc. The objective is to predict the salary, specifically predict a binary classification between salary greater than $50K/year and less than $50K/year. This is a good representative set for our needs.

In order to understand the details of each step in the featurization pipeline, here are the details of the features given in this data set copied for reference.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_15

Feature Details

1. *age*: continuous.
2. *workclass*: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
3. *fnlwgt*: continuous.
4. *education*: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th–8th, 12th, Masters, 1st–4th, 10th, Doctorate, 5th–6th, Preschool.
5. *education-num*: continuous.
6. *marital-status*: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
7. *occupation*: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
8. *relationship*: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
9. *race*: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. *sex*: Female, Male.
11. *capital-gain*: continuous.
12. *capital-loss*: continuous.
13. *hours-per-week*: continuous.
14. *native-country*: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad and Tobago, Peru, Hong, Holand-Netherlands.

Most of the feature names are descriptive of their meaning, but for the rest, the reader is advised to check [6].

Expected outcome is binary classification as,

1. salary > $50K
2. salary <= $50K

15.3 Identifying the Raw Data, Separating Information from Noise

In present problem, we are given a set of raw data that is already curated and in tabular form. Hence, we already know that we will need to use all the data that is available. This may not be true in some situations. In such cases, the rule of thumb is to use all the data that may even remotely have any influence on the outcome. However, all the data that is not related to the outcome in any way, should be

removed. Adding noise in the data dilutes it and affects the performance of the algorithm. Most classification algorithms can easily handle hundreds of features, but one must be careful in curating these features to have a good balance between what is needed and what is pure noise. This is not a one time decision and one can always experiment with different sets of features and study how they are affecting the outcomes.

15.3.1 Correlation and Causality

There are some techniques that can also be used to understand the relation between individual feature and the outcome. One simple technique is called as *correlation*. Quantitatively it is computed using *correlation coefficient* or *Pearson correlation coefficient* ρ from the mathematician and statistician *Karl Pearson*. Let there be two sets of distributions X and Y. From the current context, let us say X corresponds to one of the features and Y corresponds to the outcome. Probabilistically, the coefficient is defined as,

$$\rho_{X,Y} = \frac{cov(X, Y)}{\sigma_X \sigma_Y} \tag{15.1}$$

where σ_X is the standard deviation of X and σ_Y is the standard deviation of Y and covariance between X and Y, $cov(X, Y)$ is defined as,

$$cov(X, Y) = E[(X - \mu_X)(Y - \mu_Y)] \tag{15.2}$$

$E(z)$ represents the expected value of variable z also known as population average. μ_X and μ_Y represent the sample means of X and Y. Using Eqs. 15.1 and 15.2, we can write,

$$\rho_{X,Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{15.3}$$

In order to numerically compute these values, let us assume there are n samples and denote individual samples from feature and outcome as $x_i, i = 1, \ldots, n$ and $y_i, i = 1, \ldots, n$. We can expand the definitions of these expressions to arrive at,

$$\rho_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\sum_{i=1}^{n}(x_i - \mu_X)^2}\sqrt{\sum_{i=1}^{n}(y_i - \mu_Y)^2}} \tag{15.4}$$

The value of correlation coefficient lies between $[-1, 1]$. Positive values indicate there is direct correlation between the two vectors. In other words, if value of the feature increases, the value of the outcome also increases. Value of 0 means there is no correlation between the feature and outcome and the feature should not be used

in the model. Correlation of 1 shows there is very direct correlation between feature and outcome and one must be suspicious if we are using a feature that is part of the outcome. Typically very high correlation of 1 or a value close to it, is not possible in real applications. Value of -1 means very high but inverse correlation. Typically, negative correlations can be quickly converted to positive by changing the sign of the feature values. But, algorithms are smart enough to do that automatically as well.

This correlation analysis can quickly give us some insight into what features we are using and which features are likely to have more influence on the outcome. There is another caveat that must be recognized as well. This is the confusion between correlation and causality. Sometimes a completely noisy feature can show high correlation with outcome. For example in the current context, say we have gathered the data about postal addresses of the people. If one comes up with a binary feature as,

- feature value = 1, when the street number is even
- feature value = 0, when the street number is odd

Let's say this feature has a relatively high correlation with the outcome of the order of 0.7. Does that mean it is a good feature? From pure numbers, the answer would be yes, but from the *domain knowledge* of the origin of the feature one can easily confirm that this is pure coincidence. Such coincidental features need to be carefully identified and removed from the model, as they may have adverse effect on the model performance.

15.4 Building Feature Set

Once we have identified the set of raw data that contains *good information* as discussed in the previous section, the next step is to convert this information into machine usable features. The process of converting the raw data into features involves some standard options and some domain specific options. Let us first look at the standard options.

15.4.1 Standard Options of Feature Building

The raw information that is curated for model building can be one of the following types from the perspective of computers:

1. Numerical: This can involve positive or negative integers, fractions, real numbers, etc. This does not include dates, or some form of IDs. (sometimes IDs can appear to be numeric, but they should not considered numeric from the model standpoint, as the increment and decrement in the values of IDs has no real meaning from the perspective of the outcome.) If a numeric entity is

known to have only a small set of unique values, it is better to treat them as categorical rather than numeric. Rule of thumb is, if the number of unique values is in few tens, it could be treated as categorical, but if that number is over a hundred, then it's better to treat them as numerical. Ideally a feature should be treated as numerical, if the increase and decrease in the value of the feature has corresponding (direct or inverse) impact on the outcome.
2. Categorical: These can be string features or numeric features. String features (pure string or alphanumeric) should be treated as categorical using similar rule as stated above when the unique values it can take is in few tens. If there are more than hundred unique values, they should be treated as string features.
3. String: These are pure string features, where we expect to have some unique value in each sample. Typically these features would contain multiple words or even sentences in some cases.
4. Datetime: These would come as alphanumeric features and seem like string features, but they should be treated differently. They can have only date, only time, or datetime.

Now, let us dive deeper and see how each standard of feature should be dealt with.

15.4.1.1 Numerical Features

Numerical features are the simplest to treat, as they are already in machine understandable format. In most cases, we keep these features as is. In some cases following operations can be performed:

1. Rounding: the fractional or real valued features can be rounded to nearest or lower or upper integer if the added granularity of the fractional value is of no use.
2. Quantization: Rather than just rounding, the values can be quantized into a set of predefined buckets. This process can take the feature closer towards a categorical feature, but in many cases, this still provides a different information and can be desirable.

15.4.1.2 Categorical Features

Treatment of categorical features is significantly different compared to the numerical features. As stated earlier, the feature is considered as categorical when the number of unique classes is in few tens. However, in some cases with large data sets, it may be appropriate to use categorical features even with hundreds of unique categories. At the heart of this treatment is a procedure called as *One Hot Encoding*. One hot encoding stands for conversion to a vector of 0's and 1's such that there is only single 1 in the entire vector. The length of the vector equals to the number of categories. Let's take the example of [*workclass*] from our example. There are eight different categories. Hence each category would be encoded into a vector

of length 8. The one hot encoding of "Private" would be [1, 0, 0, 0, 0, 0, 0, 0], of "Self-emp-not-inc" would be [0, 1, 0, 0, 0, 0, 0, 0], and so on. Each value in this vector is a different feature and the names of these features would be "workclass-Private", "workclass-Self-emp-not-inc," and so on. So now, we have expanded a single categorical feature into eight binary features. One can ask a question as why should we complicate this process into building so many more features, when we can just convert them into a single integer feature and assign the values from say 1–8, or 0–7 or even 00000001–10000000. The reason for not doing this way lies in the relation between two numbers. For example number 1 is closer to number 2 than number 3 and so on. Does this relation holds for the values we are encoding? Does the value "Private" is closer to value "Self-emp-not-inc" than say "Without-pay"? The answer to these questions is *NO*. All these values have a meaning of their own and cannot be compared to each other like we compare two numbers. This can only be expressed numerically by putting them in different dimensions of their own. One hot encoding achieves precisely that.

15.4.1.3 String Features

Generic string features, also called as text features can be processed in multiple ways. But in general before converting them into features, a set of preprocessing steps are recommended. These include:

1. Removing punctuation characters: In most cases, the punctuation characters do not carry any useful information and can be safely removed.
2. Conversion to lower case: In most cases, converting all the text into lower case is highly recommended. If the same word or set of words appears in multiple places with even a slight difference in casing, machine treats them as two completely separate entities. It not a desired in most cases, and converting everything to lower case makes the next processing more streamlined.
3. Removal of common words: Words like articles (a, an, the), conjunctions (for, but, yet, etc.), prepositions (in, under, etc.) are used quite commonly and are typically not quite informative from the machine learning model perspective. These are also termed as *stop words* and can be safely removed. Due to high frequency of these words, removal of these words reduces the complexity significantly. A pre-populated list of stop words is typically available in machine learning libraries and it can be used as off-the-shelf component.
4. Spelling fixes: It is quite common to have spelling errors in the data and misspelled words can give rise to lot of noise. Standard spell check can be safely applied to improve the data quality.
5. Grammar: This is a complex topic in itself. In many cases, applying various grammar checks can improve the quality of data, but these modifications are not necessarily generic and should be applied on a case by case basis. The common techniques include: stemming and lemmatization (converting words like "walking" or "walked" to "walk," etc.), tagging parts of speech (POS), etc.

The featurization of the text can be broadly classified into simple and advanced. Simple featurization can include counts and frequencies of different entities. Some examples are:

1. Number of words
2. Frequency of repeated words
3. Number of stop words
4. Number of special characters
5. Number of sentences

Advanced featurization includes:

1. N-grams analysis: N-gram analysis segments the given text into buckets of consecutive n words. Each unique sequence of n words is collected as a unique n-gram and they are encoded based on the frequency of occurrence of each n-gram. For example consider a sentence "Michael ran as fast as he can to catch the bus." Assuming we applied the lower case conversion as preprocessing step, unigram or 1-gram analysis would give: "michael," "ran," "as," "fast," "he," "can," "to," "catch," "the," "train" unique 1-grams. The encoded vector will be [1, 1, 2, 1, 1, 1, 1, 1, 1, 1]. Bigram or 2-gram analysis of the same sentence will generate following unique 2-grams "michael ran," "ran as," "as fast," "fast as," "as he," "he can," "can to," "to catch," "catch the," "the bus". The encoded vector will be [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]. As you can see the difference between 1-grams and 2-grams, word "as" appears only once in the feature space of 1-grams, but the bigrams "as fast" and "fast as" appear separately in 2-grams feature space. In cases like this elimination of stop words (such as "as") can reduce the feature space drastically.

   One more thing to notice here is that n-gram analysis with $n > 1$, captures the sequential information of the words and not just their frequency. Especially when n gets larger, the uniqueness of features increase exponentially. Also, if n is too large the space gets too sparse, so one needs to carefully do the tradeoff between higher value of n vs lower value.
2. Bag of Words: Bag of words is typically described as most popular method of featurizing string entities. It essentially represents the n-gram analysis with n=1.
3. TF-IDF analysis: It stands for term frequency–inverse document frequency. This analysis is typically useful in case of long texts. For shorter texts, n-grams or bag of words is typically sufficient. Conceptually it quantifies the importance of a word in a given text compared to the collection of texts. Let us look at the mathematical definition of this.

   Let the term frequency (TF) of a word w_j in a text t_j be given as $f_w(w_i, t_j)$. This is calculated as the number of times word w_i appears in text t_j. Let there be total of n words over the entire corpus of texts. Let there be m number of texts.

The inverse document frequency (IDF) $f_{id}(w_i)$ of word w_i is defined as,

$$f_{id}(w_i) = \log \frac{n}{\sum_{j=1}^{m} \left(\begin{cases} 1, & \text{if } w_i \in t_j \\ 0, & \text{otherwise.} \end{cases} \right)} \tag{15.5}$$

the denominator essentially counts the number of texts that contain the word w_i. The joint TF-IDF expression is then the combination of TF and IDF as $f_w(w_i, t_j) f_{id}(w_i)$.

To illustrate the concept, the stop words will have a high TF in all the documents, but low IDF overall, and as a result their TF score will be discounted accordingly. However, if a certain word appears only in one document, its TF in that document will be significantly boosted by IDF.

15.4.1.4 Datetime Features

Datetime features expose a very different type of information and it needs a little bit of domain knowledge to make sense. That way, they belong more in the next section of custom features. However, they also allow for some standard processing and hence are discussed here. In string format the datetime feature is quite meaningless, at the same time converting datetime directly to a numeric value also brings little to the table. Commonly extracted datetime features include:

1. Day of week
2. Day of month
3. Day of year
4. Week of month
5. Week of year
6. Month of year
7. Quarter of year
8. Days from today
9. Hour of day
10. Minute of hour

15.4.2 Custom Options of Feature Building

Most of the featurization options discusses in the previous section are standard in the sense that they are valid irrespective of the context or in absence of any domain knowledge. When we are building a custom machine model to solve a specific problem in artificial intelligence, we already have a domain knowledge that can be applied to extract features that may not make sense in generic settings. For example, let us consider the example of salary prediction. One of the numeric feature is "age."

It is numeric and continuous feature, and as per standard featurization guideline we can use it as is or we can round it integer, or even bin it to multiples of 5, etc. However, for current problem of salary, this age is not just a number between say 0–100, but it has more to it. We know that typical starting age for employment is say around 18 and typical retirement age is say around 65. Then we can add custom bins within that range to create a more meaningful feature in current context.

Another important aspect of custom feature building is in the form of interactions between two features. This is completely non-trivial from generic standpoint and can gain a significant information that is otherwise not possible. For example from the adult salary example, there are two features "Capital-gain" and "Capital-loss." We can join these two features in the form of difference as "(Capital-gain) − (Capital-loss)" to see the big picture in single feature. Or we have some survey information that gives a specific relationship between the "age" and "education-num," which we can use to join these two features in some proportion to create a more informative new feature. These features are non-trivial and would certainly add to the information that the model can use.

Datetime features are also a string candidate for creating interaction features. Difference of days between two dates is always a useful feature. Another way to join features is applying mean, min, max, etc. operators on collection of numeric features. The options are quite unlimited. However, it must be noted that creating large amounts of custom features is not necessarily going to improve the performance arbitrarily and there would soon be a point of saturation. Nevertheless, this is always a good option to experiment.

15.5 Handling Missing Values

Missing values is a very common occurrence in real life situations and they must be substituted with some values or the models will fail to execute. If the missing values are present in the training data, trivial solution is to just ignore these samples. However, in some cases there is just too many samples with some missing information and we cannot just ignore them. Also, if the values are missing in test data, where we need to make a prediction, we have no choice but to substitute the missing value with some reasonable estimate. Most common ways to handle the missing values are:

1. Use the mean or mode or median value in case of numeric values.
2. Use the mode in case of categorical values.
3. If we have some prior knowledge, we can always replace the missing value with a predetermined value.
4. In case of categorical features, we can create a new category as "unknown" and use that instead. However, there is one caveat here and that is if the "unknown" category pops only in the test data and is not available in the training data some models can fail. However, if there are samples in training data with missing

values that are also substituted with "unknown," then it is typically the most elegant solution.
5. Predicting the missing value. This can be achieved by variety of ways, ranging from simple regression based methods to more complicated *Expectation Maximization* or *EM* algorithms. If the feature that is being predicted has a very high variance, then this method can result in bad estimates, but in general this is better suited than simple "mean/mode/median" substitution.
6. Use the algorithms that support missing values. There are certain algorithms that inherently use the features independently, e.g., Naive Bayes, or k-nearest neighbor clustering. Such algorithms can work with missing values without needing to substitute it with some estimate.

15.6 Visualizing the Features

The last step in featurization is visualizing the features and their relationship with the labels. Looking at the visualizations, one can go back to some feature modifications as well and iterate over the process. To illustrate this, let us use the adult salary data.

15.6.1 Numeric Features

First let us look at the numeric features. The numeric features are:

1. *age*
2. *fnlwgt*
3. *education-num*
4. *capital-gain*
5. *capital-loss*
6. *hours-per-week*

Now, let's look at the plots of individual feature values compared to the label as shown in Figs. 15.1, 15.2, and 15.3.

Table 15.1 shows the correlation coefficients of each of the features with the label.

As we can see from the figures and the correlation coefficients, feature *fnlwgt* contains almost no information and most other features a relatively weak features on their own. As discussed before the negative sign only shows the relationship between their values and label is inverse. Looking at this data, we can safely ignore the feature *fnlwgt* for model training purpose. However, considering that there are only 14 features to start with, we can keep it.

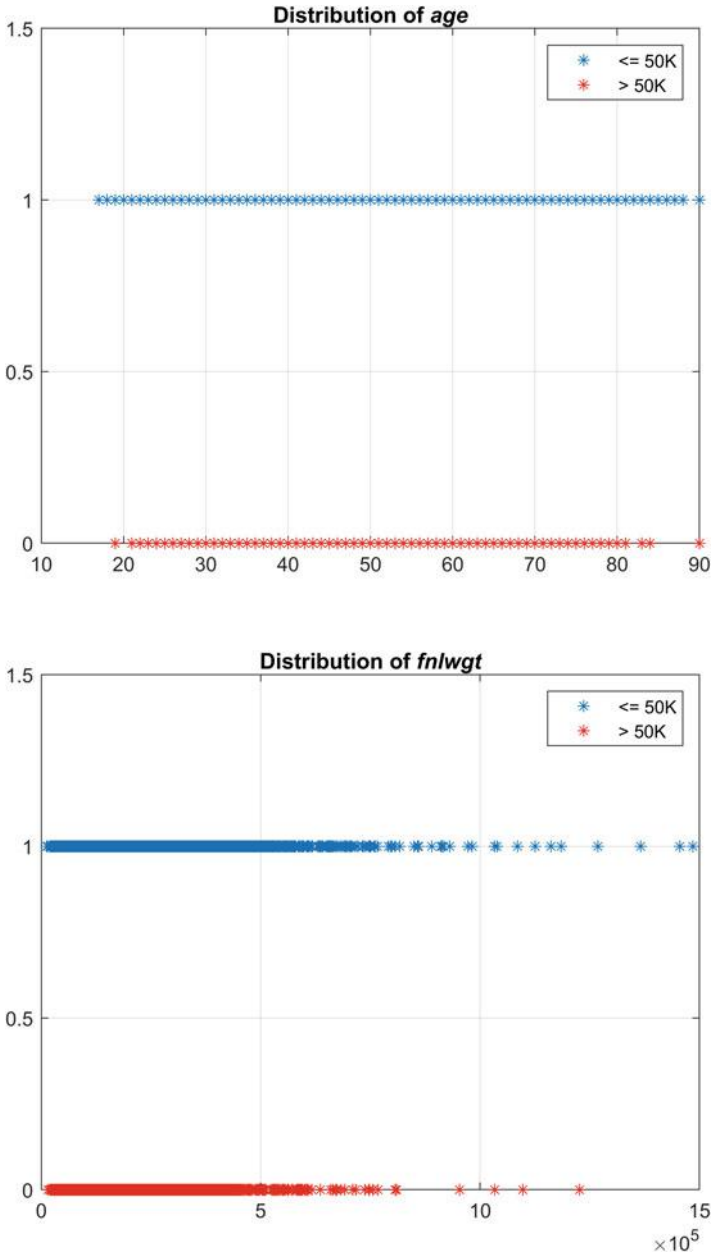


Fig. 15.1 Showing class separation using individual features *age* and *fnlwgt*

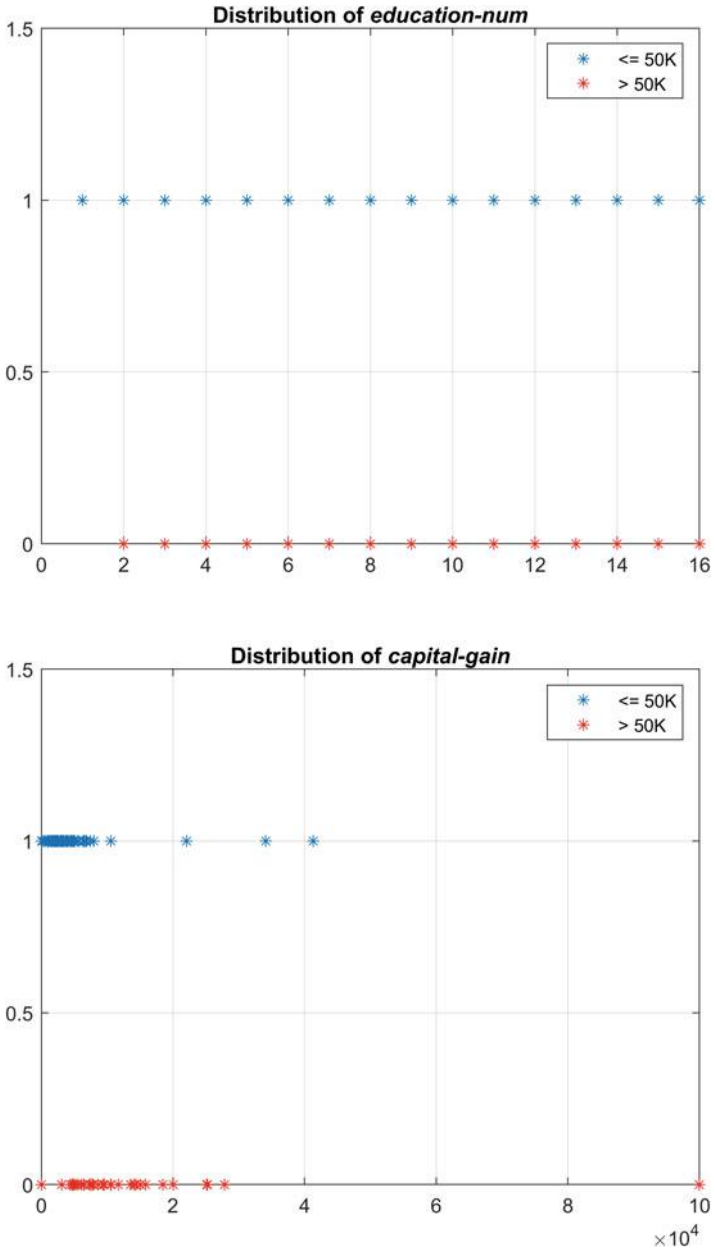


Fig. 15.2 Showing class separation using individual features *education-num* and *capital-gain*

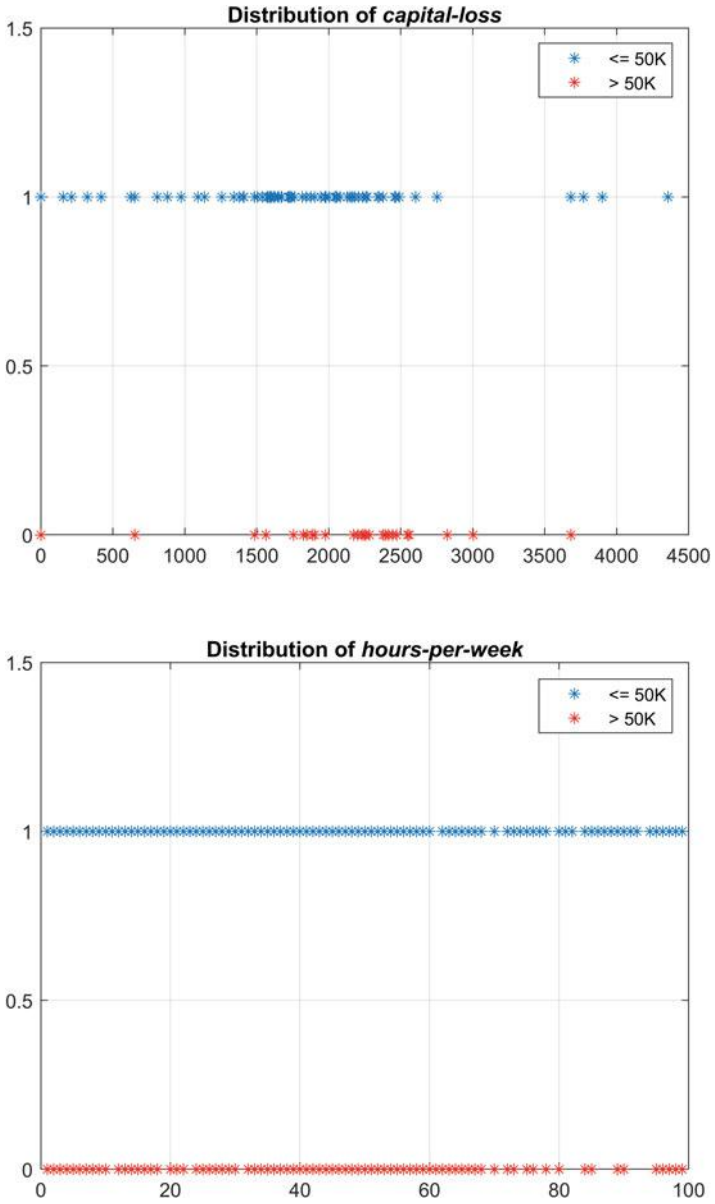


Fig. 15.3 Showing class separation using individual features *capital-loss* and *hours-per-week*

15.6.2 Categorical Features

Now, we will look at the categorical features one by one. As described before, each value of a categorical feature is treated as separate feature by using one hot encoding technique.

Table 15.1 Correlation coefficients between numerical features and label

Feature name	Correlation coefficient
age	−0.23
fnlwgt	+0.01
education-num	−0.34
capital-gain	−0.22
capital-loss	−0.15
hours-per-week	−0.23

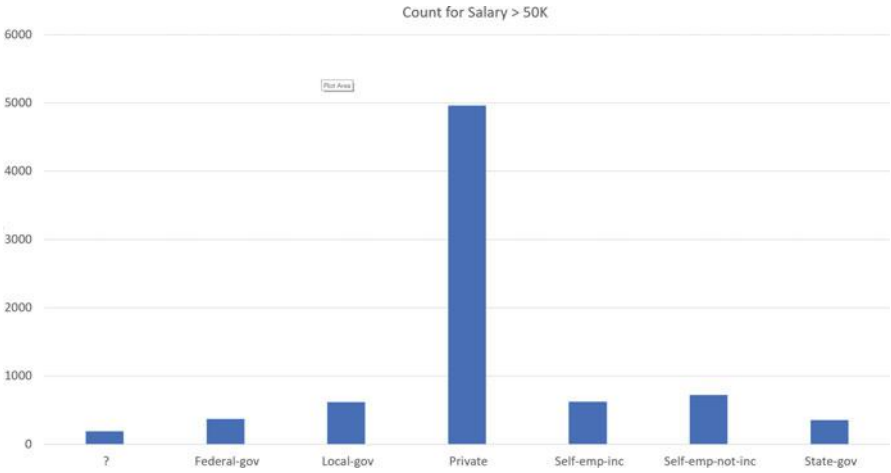


Fig. 15.4 Pivot table showing distribution of the label (>50K) for each value of the feature. *?* means missing value

15.6.2.1 Feature: *Workclass*

Feature *workclass* is a categorical feature with possible values in {Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked}. It would be much harder to visualize the effect of all the different values of the features in a single plot the way we did in case of numerical features. Also, we don't want to create separate plot for each value, as that would lead to just too many plots. Instead we will use a technique called pivot table or pivot chart. In this technique we put all the values of one parameter on one axis and count for any other aggregate function (sum, max, min, median, mean, etc.) of the values of the other parameters on the other axis. Figures 15.4 and 15.5 show the pivot charts for all the values of this feature in case of salary >50K and salary ≤50K. As can be seen from the charts, the value of *Private* has similar distribution in both cases, but the values *Federal-gov* and *self-emp-inc* show significantly different distribution in both cases. Thus we expect that the separate features created using these values should be more influential in the model.

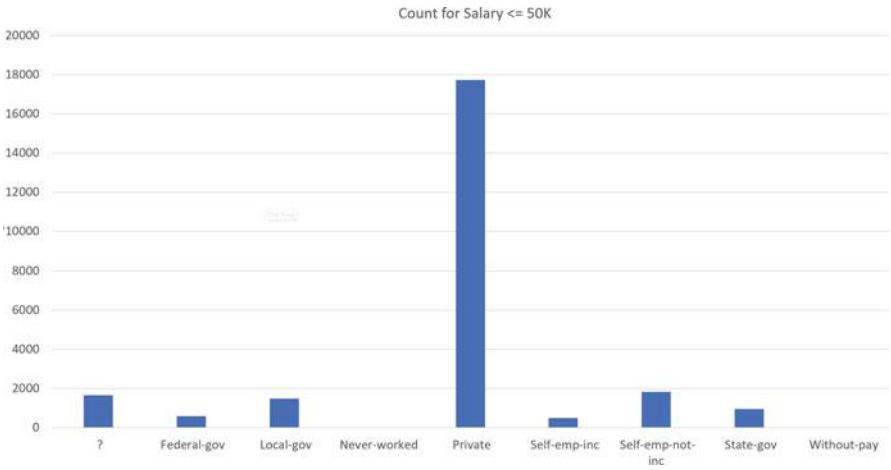


Fig. 15.5 Pivot table showing distribution of the label (≤50K) for each value of the feature. *?* means missing value

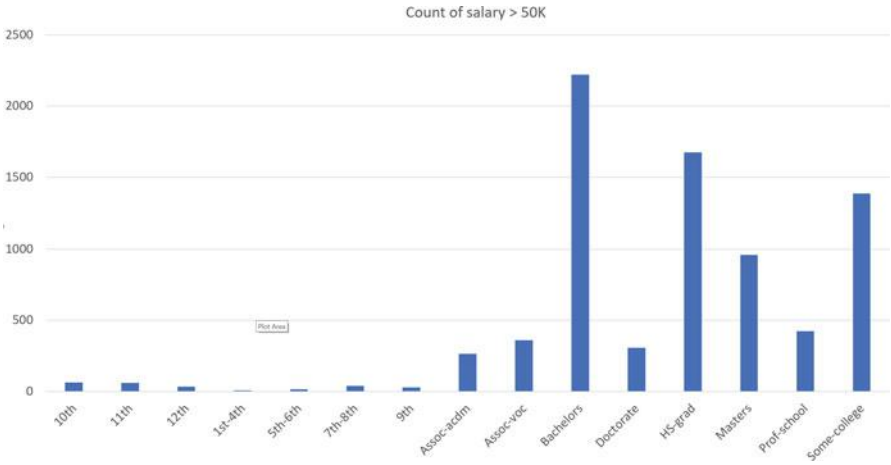


Fig. 15.6 Pivot table showing distribution of the label (>50K) for each value of the feature. *?* means missing value

15.6.2.2 Feature: *Education*

Feature *education* is a categorical feature with values in {Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th–8th, 12th, Masters, 1st–4th, 10th, Doctorate, 5th–6th, Preschool}. We will use the pivot chart as before to see the trends between value of this feature and label. Figures 15.6 and 15.7 show the pivot charts. Some features are altogether missing between the plots as a result of having zero count. These values would be expected to have a very high influence

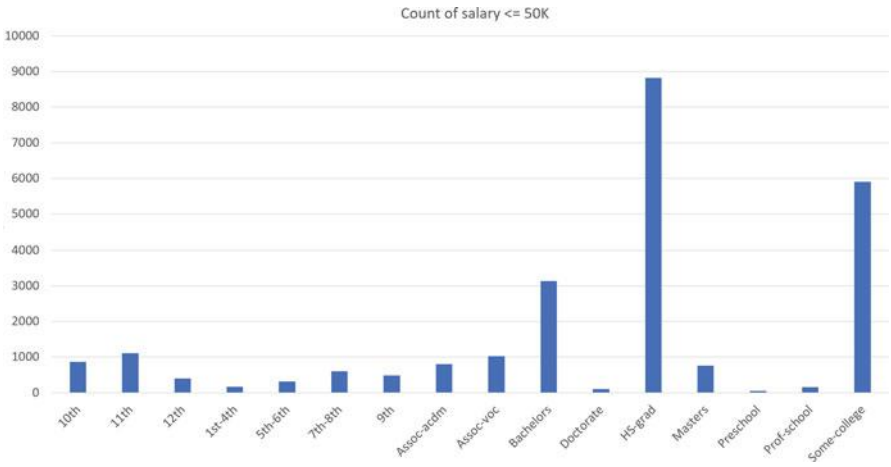


Fig. 15.7 Pivot table showing distribution of the label ($\leq$50K) for each value of the feature. *?* means missing value

on the outcome. Also the values corresponding to higher education show strong separation between the two classes.

15.6.2.3 Other Features

Remaining categorical features *marital-status*, *occupation*, *relationship*, *race*, *sex*, *native-country* can be analyzed in similar fashion. The insights gained by looking at the trends visually can be extremely effective in choosing the right mode, tuning it, selecting the constraints for regularization, etc.

15.7 Conclusion

This chapter described the concept of featurization in detailed manner. We started with generic description of the techniques and to make them real, we took an example of the adult salary data and performed the featurization of it by taking each individual feature one at a time. We also used the help of visualization to draw some inferences on the influencing capabilities of each feature on the outcome.

Chapter 16
Designing and Tuning Model Pipelines

16.1 Introduction

Once the features are ready as discussed in the previous chapter, the next step is to choose the technique or algorithm to use for the desired application. Then the available data is split into two or three sets depending on how the tuning is implemented. The three sets are called as training, validation, and testing. The training set is used for training the model, optional validation set is used to tune the parameters, and test set is used to predict the performance metrics of the algorithm that we would expect when it would be applied to the real data. We will go through these steps in detail in the following sections.

16.2 Choosing the Technique or Algorithm

Typically the application dictates the broad category of method we should use. For example if we are dealing with a regression problem then we are restricted to all the regression type techniques, which would eliminate the clustering or recommendation algorithms[1], etc. Alternatively, if we are dealing with unsupervised clustering application, then all the supervised algorithms like decision trees, support vector machines, or neural networks cannot be used.

[1]The terms *model* and *algorithm* are used interchangeably in most machine learning literature, and it can be confusing. The precise definition of these two terms is: Algorithm is the underlying theoretical infrastructure and model is the abstract binary structure that is obtained when we go through the training process. In other words, model is a trained algorithm. With different training data, we get a different model, but underlying algorithm remains unchanged. The trained model then can be used to predict outcomes on the unlabelled data.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_16

In order to illustrate the thought process in choosing the model, let's consider the problem of binary classification of the adult salary data as described in the previous chapter.

16.2.1 Choosing Technique for Adult Salary Classification

As the application dictates we have to narrow our scope to classification algorithms. Regression algorithms can always be used in classification applications with addition of threshold, but are less preferred. We discussed about the types of field given to us. Here is a snapshot of actual data for first few samples (Table 16.1).

As can be seen here, the data types we are dealing with here are either continuous valued integers or string categorical. There are also some missing features represented with "?".

Decision tree based methods are typically better suited for problems that have categorical features as these algorithms inherently use the categorical information. Most other algorithms like logistic regression, support vector machines, neural networks, or probabilistic approaches are better suited for numerical features. This is not to say that you cannot use them at all in case of categorical features, but this suggestion should be taken as rule of thumb. Also, it is better to start with a simpler algorithm to begin with to establish a baseline performance. Then we can try with mode complex algorithms and compare the performance with baseline. Complex algorithms are typically the ones that need elaborate setup and initialization of large number of parameters, also called as hyperparameters. We will deal with this topic later in this chapter.

Single decision tree can be used as one of the simplest possible starting algorithm; however, random forest decision tree is also a good choice. As described in Chap. 6, random forest algorithm provides significant improvements over single decision tree without adding much complexity from training perspective.

16.3 Splitting the Data

Once an algorithm is selected for use, the next step is to separate the whole labelled training data into 2 or 3 sets, called as training, (optional) validation, and test. In this case, as we plan to do hyperparameter tuning, we will divide the data into three sets. Typically the division of the three sets in percentage is done as: 60–20–20, or 70–15–15. More the samples available for training the better, but we need to also keep sufficient samples in validation and test sets to have statistically significant metrics. This aspect is discussed in greater detail in Chap. 17. Typically the validation and test sets are of same size. But all these decisions are empirical and one can choose to customize them as needed.

Table 16.1 Sample rows from UCI adult salary data

Age	Workclass	fnlwgt	Education	Education-num	Marital-status	Occupation	Relationship	Race	Sex	Capital-gain	Capital-loss	Hours-per-week	Native-country	Label
38	Private	215,646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
53	Private	234,721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
28	Private	338,409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
37	Private	284,582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	Unites-States	<=50K
49	Private	160,187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica	<=50K
19	Private	168,294	HS-grad	9	Never-married	Craft-repair	Own-child	White	Male	0	0	40	United-States	<=50K
54	?	180,211	Some-college	10	Married-civ-spouse	?	Husband	Asian-Pac-Islander	Male	0	0	60	South	>50K
39	Private	367,260	HS-grad	9	Divorced	Exec-managerial	Not-in-family	White	Male	0	0	80	United-States	<=50K
49	Private	193,366	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K

16.3.1 Stratified Sampling

The splitting is always done in random fashion and not sequential manner to have statistically even spread of data between all the three sets. There is one more aspect that needs careful attention, specifically in case of classification applications. This is called as *stratified sampling*. Stratified sampling ensures a certain known distribution of classes in split parts. When we are dealing with say n number of classes, the number of samples per class is not always same. Sometimes there can be large skew in this distribution, meaning number of samples of one class can be significantly larger than others. In such cases the uniform random split does not yield optimal separation. In order to train the model without bias towards any specific class, we need to have roughly same number of samples from each class in the training set. The distribution in validation and test set is less important, but it is always a good practice to have roughly balanced distribution across the classes in all the three sets. When the original distribution is unbalanced, there are two choices:

1. Ignore a number of samples from the classes that have more samples to match the classes that have less number of samples.
2. Use the samples from the classes that have less number of samples repeatedly to match with number of samples from the classes that have larger number of samples.

Which option to choose now depends on the original number of samples. If there are sufficient number of samples even after dropping samples from larger sets, then that is always a better option rather than repeated sampling.

Also, it is not mandatory to use stratified sampling, if the bias between classes is acceptable. In the current problem of adult salary classification, they have given the training and test data in separate sets already. There are 24,720 samples from the class $\leq$50K and only 7841 samples from the class $>$50K in training data that contains total of 32,561 samples. The test set contains 12,435 samples from the class $\leq$50K and only 3846 samples from the class $>$50K with total of 16,281 samples. Thus there is significant skew in the sample distribution. However, as per the guidance given by the creators of the data set, the bias between these two classes of salary ranges is known and needs to be preserved. Uniform random sampling typically gives similar distribution between the classes in different parts when data is large enough, but if there is significant difference, one can repeat the sampling to ensure that, or one can use stratified sampling to enforce certain distribution of classes in each part. In current problem we just need to split the training set into two parts training and validation. We will use 70–30 split for this purpose.

16.4 Training

Once the data is split into training and test sets, the training set is used to train the algorithm and build the model. Each algorithm has its own specific method associated with it for training. The underlying concept common in all the different technique is to find the right set of parameters of the model that can map the input to output as far as training data is concerned. In other words, to find the set of parameters that minimize the error (one can choose different types of errors as defined in Chap. 17) between the predictions and the expected values of the output in training set. Some methods are iterative, and they go through multiple loops of finding and improving the parameters. Some methods are one shot, where in single loop the optimal set of parameters are computed.

The parameters of the model are also classified into two main types:

1. The parameters that can be computed using the training process to minimize the error in the prediction.
2. The parameter that cannot be directly computed using the training process. These parameters are also called as hyperparameters. Each unique set of hyperparameters can be thought of as a different model in itself.

16.4.1 Tuning the Hyperparameters

The hyperparameters are typically a set of parameters that can be unbounded, e.g., number of nodes in a hidden layer of a neural network. One can theoretically choose any number between 1 and ∞. One cannot simply use the training data to find the right number of nodes. Hence the bounds on hyperparameters are created by the scientist, who is in charge of building the model. These bounds are created based on multiple constraints like computation requirements, dimensionality, and size of data, etc. Typically one needs to create such bounds for more than one hyperparameter for each model. Thus we end up with an n-dimensional grid of hyperparameters to choose.

As stated in previous section a single set of training set can be used to get results with one set of hyperparameters. Hence in order to have multiple such sets to tune hyperparameters, one needs to split the training data further into two parts called as training and validation. There are multiple techniques described in the literature for generating these sets. However, the test set must be kept separate and never be used in the tuning or training processes. If one uses the test set in this process, then all the labelled data is used for the training-tuning process and there would no data left to predict the behavior of the trained-tuned model on unseen samples. This is a fundamental rule in machine learning theory that must never be broken.

For a given set of hyperparameters, the training set is the only data available for training and the trained model is applied on the validation set to compute the accuracy metrics. Once all the different set of hyperparameters are used, the set that

provides best performance on the validation set is used as best model. Then the test data is used only once to predict the accuracy of the tuned and trained machine learning model.

16.5 Accuracy Measurement

Measuring the accuracy of a model is the last step in the design of machine learning system. However, considering the scope of this topic entire next chapter is devoted on this step.

16.6 Explainability of Features

Typically once the model is trained with sufficient accuracy, the job is over. However, in recent times, an additional step is getting more and more importance. This step does not find roots in theory of machine learning but has emerged more due to the collision of the traditional heuristic methods with machine learnt methods. When a domain expert builds a simper heuristic model to predict certain outcomes, all the rules that are part of the heuristic model are human readable and have obvious interpretability. When the outcome is high or low, the reason for that specific outcome can quickly be seen from the rules and interpreted. These interpretations also lead to concrete actions that can be taken based on them.

Most of the machine learnt models, specifically the neural net type models, lack this reasoning and interpretability completely. One must accept the outcome as it stands based on the accuracy metrics that are provided based on the test data. However, this lack of reasoning or explainability and interpretability came into strong criticism when the machine learnt models started to replace the older heuristic models. There are variety of different techniques proposed that can be applied after the model is trained to add explanability of the results, or importances of the features used. The core idea here is to vary the value of individual features and see their effect on the outcome. The features that have stronger impact on the outcome are more important and vice versa. However, this technique does not take into account interdependency of the features. This article [30] discusses some of the current advances in this field. Also, the explainability of the features needs to be built for aggregate level as well as case by case level.

16.7 Practical Considerations

All the machine learning algorithms discussed so far in the book are based on certain assumptions about the nature of the data. Either the data is static, or it is in the form of time series. Either data is strictly linear or it can be converted into

linear with suitable link function or it is purely nonlinear. The number of classes defined in the training data cannot be different from the ones in test data. And so on. However, when we deal with data in practice none of these assumptions are precisely applicable. There is always some gray area and it is very crucial to identify such areas and address them explicitly. We will discuss some of the more commonly observed situations in this section, but this list should not be treated as comprehensive.

16.7.1 Data Leakage

In selecting the underlying algorithm for solving a given machine learning problem one must understand the trends in the data. If the data is static then there is a set of static algorithms that one can choose from. Examples of static problem could be classification of images into ones that contain cars and ones that don't. In this case the images that are already gathered for the classification are not going to change over time and is purely a case of static analysis. However, if the data is changing over time, it creates additional layer of complexity. If one is interested in modeling the trends of change in data over time, it becomes a time series type of problem and one must choose appropriate model e.g. ARIMA as described in Chap. 11. For example stock price prediction. However, in some cases, we are dealing with data that is changing, but we are only interested in the behavior of the data for a given snapshot in time. This situation does not warrant a time series model and once a snapshot of the data is taken, it becomes a strictly static problem. However, when such snapshot of the data is taken for building a machine learning model, one must be careful about the time-line of changes of all the individual features till the time of snapshot.

An example will help understand this phenomenon better. Consider a business of auto mechanic. We are trying to predict the customers who are likely to visit more than once in a period of say 6 months. We gathered the sales data for past 6 months. Here are some of the columns in the data:

1. Number of visits
2. Year of manufacture of the car
3. Make of the car
4. Model of the car
5. Miles on the odometer
6. Amount of sale
7. Method of payment
8. Category of customer

The first column is actually the label of the data, where if the number of visits is greater than 2, we classify it as "True" and classify as "False" otherwise as per the definition of the problem. This column is then removed from the feature space and all the remaining columns are used as features. However, the last column in the list is

a categorical feature where we classify each customer, based on the historical data. Only the customers that are repeat customers, we put them into "Known" category, and rest of them are put into "Unknown" category. Now, this column is not exactly same as the label column; however, the "Known" customers are more likely to be classified as "True" as per our definition compared to the "Unknown" customers. Also, it must be noted that the value in this column is going to be updated after each visit. Thus if we use this column as one of the features, it is going to leak partial information from the label back into the feature space. This will cause the model to perform much better than expected based on this single feature alone. Also, it must be noted that this feature will not be available for the customers when they are going to visit for the first time. Hence it would be best to not use this feature in training the model. This leakage of information is also called as *Target Leaking*.

16.7.2 Coincidence and Causality

In practice when dealing with data with large number of features for predicting relatively small number of classes or predicting a relatively simple regression function, the noise levels in the training data can significantly affect the model quality. When there are too many features present in the data that have no causal effect on the outcome, there is a chance that some of these noisy features may show *lucky* correlation with the outcome. Identifying such noisy features is typically done with the domain knowledge of the problem space, but sometimes the scientist developing the model may lack this knowledge. In such cases, it becomes extremely difficult to identify the features that are coincidentally correlated with the outcome versus the features that actually have causal effect on the outcome. The coincidental features might help in getting high accuracy on training data, but will make the model quite weak in predicting the outcome on the unseen test data. In other words the generalization performance of the model will be quite poor.

Unfortunately, there exists no theoretical method to identify and separate the coincidental features from the causal features purely from the data analysis. There are some probabilistic methods proposed based on conditional dependence between features, but all such methods make some assumptions on the dependency and causality and these assumptions may not hold true on the real data. This is a relatively novel aspect of machine learning and is under study. Here is one article that discusses some aspects of this phenomenon [29]. Although this is a problem with no concrete theoretical solution, one can circumvent this situation by taking following measures:

1. Using cross validation even if hyperparameter tuning is not used. This splits the data in multiple different combinations thereby reducing the chance of coincidental features getting more importance.
2. Using ensemble methods as opposed to single model methods to improve the robustness and reduce coincidence.

3. Applying feature interpretation as an additional step after the model training to make sure all the important features have domain specific explainability. Having such explainability makes the models more resilient of coincidences.

16.7.3 Unknown Categories

Existence of missing data is a common occurrence in case of training classification problems. However, the case of unknown categories is little more deeply rooted and can be confusing. When a feature type is treated as categorical, it is then processed accordingly as described in Chap. 15. Each new category encountered is added to the list of categories and accordingly the categories are encoded into numerical features. However, sometimes, it may happen that a new category is encountered only in the test set, which was completely missing in the training or validation set. Most models can fail in such situations. However, if the model is built to take into consideration of this possibility this failure can be avoided. Here are few examples of handling such situation:

1. The new category is treated as unknown category and such unknown category is preprogrammed into the featurization of the model. There may be an unknown category in the training data where the data is actually missing and then the newly discovered category can be considered as missing data. This can be acceptable behavior if planned for.
2. If a new category is discovered in the test data, it is explicitly treated as missing data and the model is trained to ignore this feature and still be able to produce the result.
3. The newly discovered category is treated as one of the known category and model is applied as trained.

All these cases are way of doing some form of approximation to make the model robust in case of unseen data, and which solution to use in any given case must be determined at the time of model building.

16.8 Conclusion

In this chapter we integrated the concepts learnt so far with respect to various algorithms and data preprocessing and discussed the elements of designing an end to end machine learning pipeline. We also discussed about the lesser known aspects in this design like data leakage and coincidence against causality and how to address those to successfully build the system.

Chapter 17
Performance Measurement

17.1 Introduction

Any discussion about machine learning techniques cannot be complete without the understanding of performance measurement. Performance can be measured qualitatively by looking subjectively at a set of results or objectively at the value of an expression. Whenever size of data is large, subjective and qualitative tests can no longer give any reliable information about the general performance of the system and objective methods are the only way to go. There are various such mathematical expressions, called as metrics defined in the field for assessing performance of different types of machine learning systems. In this chapter we are going to focus on the theory of performance measurement and metrics.

Whenever a data set is selected for training a machine learning model, one must keep a fraction of the set aside for testing the performance. This *test* data should not be used in any way during the training process. This is extremely important aspect and anyone who wants to venture into this field should treat this as sanctimonious and uber principle. Typically the *training* and *test* split is done as 70–30% or 75–25%. The rule of thumb here is two fold:

1. Model should get as much as training data as possible.
2. The test set should contain sufficient samples to have statistical confidence in the metrics produced using it.

There is a whole topic dedicated on statistical significance in statistics, but the gist of it is: one needs at least 30 samples for testing the performance of single one-dimensional variable. When there are more than one variable and more than one dimension, the rule gets much more complex depending on the dependency between the dimensions and variables, etc. In such cases the sufficient number of training samples need to be decided on case by case basis.

Following sections define most commonly used performance metrics. Some of them are quite trivial, but they are still provided for completeness. All the metrics

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_17

are based on the assumption of discrete data. In case of continuous functions, the summation is replaced with integral, but the concepts remain the same.

17.2 Metrics Based on Numerical Error

These are the simplest form of metrics. When we have a list of expected values, say $(y_i, i = 1, \ldots, n)$ and we have a list of predicted values, say $(\hat{y}_i, i = 1, \ldots, n)$. The error in prediction can be given using following metrics:

17.2.1 Mean Absolute Error

Mean absolute error is defined as:

$$e_{\text{mae}} = \frac{\sum_{i=1}^{i=n} |\hat{y}_i - y_i|}{n} \tag{17.1}$$

Use of only *Mean Error* is typically avoided, as it can lead to unusually low values due to cancellation of the negative and positive errors.

17.2.2 Mean Squared Error

Mean squared error is defined as:

$$e_{\text{mse}} = \frac{\sum_{i=1}^{i=n} (\hat{y}_i - y_i)^2}{n} \tag{17.2}$$

MSE typically penalizes less number larger errors (outliers) more heavily compared to more number of smaller errors. One can choose to use either of them based on specific problem or use both.

17.2.3 Root Mean Squared Error

Root mean squared error is defined as:

$$e_{\text{rmse}} = \sqrt{\frac{\sum_{i=1}^{i=n} |\hat{y}_i - y_i|}{n}} \tag{17.3}$$

RMSE reduces the sensitivity of the error to few outliers, but still is more sensitive compared to the MAE.

17.2.4 Normalized Error

In many cases, all the above error metrics can produce some arbitrary number between $-\infty$ and ∞. These numbers only make sense in a relative manner. For example we can compare the performance of the two systems operating on the same data by comparing the errors produced by them. However, if we are looking at a single instance of the single system, then the single value of error can be quite arbitrary. This situation can be improved by using some form of normalization of the error, so that the error is bounded by lower and upper bounds. Typically the bounds used are (-1 to $+1$), (0–1), or (0–100). This way, even a single instance of normalized error can make sense on its own. All the above error definitions can have their own normalized counterpart.

17.3 Metrics Based on Categorical Error

Performance metrics based on categorical data are quite a bit different. In order to quantitatively define the metrics, we need to introduce certain terminology. Consider a problem of binary classification. Let there be a total of n_1 samples of class 1 and n_2 samples of class 2. Total number of samples is $n = n_1 + n_2$. The classifier predicts $\hat{n}_1$ number of samples for class 1 and $\hat{n}_2$ number of samples for class 2, such that $\hat{n}_1 + \hat{n}_2 = n$. In such situations the metrics can be calculated either from the perspective of only class 1, or only class 2, or with a joint perspective.

17.3.1 Accuracy

To quantitatively define the metrics, let's define certain parameters. Let n_{ij} be the number of samples originally of class i that are classified as class j. With joint perspective, the metrics are given in terms of accuracy A as,

$$A = \frac{n_{11} + n_{22}}{n_{11} + n_{12} + n_{21} + n_{22}} \tag{17.4}$$

17.3.2 Precision and Recall

We need to define few more terms for defining the metrics from perspective of any one class. Let TP be the number of true positives from the perspective of class 1. Hence TP $= n_{11}$. Let FP be the false positives from the perspective of class 1. False positive means the sample is actually from class 2, but is classified as class 1. Hence FP $= n_21$. They are also called as *false calls*. Let TN be the number of false negatives from the perspective of class 1. True negative means the sample is actually from class 2 and is classified as class 2. Hence TN $= n_{22}$. Let FN be the number of false negatives from the perspective of class 1. False negative means the sample is actually from class 1, but is classified as class 2. They are also alternatively called as *misses*. Hence FN $= n_{12}$. Now we can define the two metrics from perspective of class 1 as *precision*, P and *recall*, R.

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{17.5}$$

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{17.6}$$

As can be seen the numerator is same in both equations, but denominator is different. In order to subjectively understand these entities, one can follow these rules of thumbs.

Rules of Thumb for Understanding Precision and Recall

1. Precision can be interpreted as how many samples actually belong to class 1 from the pool of samples that are classified as class 1.
2. Recall can be interpreted as how many samples actually belong to class 1 from the pool of samples that actually belong to class 1.

17.3.2.1 F-Score

In order to combine these two metrics into a single metric, sometimes another metric is defined as *F-score*, which is essentially harmonic mean of precision and recall. It is sometimes called as *F-measure* or *F1 score*, but they all mean the same quantity.

$$F = 2 \cdot \frac{P \cdot R}{P + R} \tag{17.7}$$

17.3.2.2 Confusion Matrix

All the equations that are described above can be reformulated from the perspective of class 2. Also, the same analysis can be further generalized for case of n-classes.

Along with these metrics often times the full matrix of misclassifications $n_{ij}, i = 1, \ldots n$ and $j = 1, \ldots, n$ is also useful and is called as confusion matrix.

17.3.3 Receiver Operating Characteristics (ROC) Curve Analysis

The precision and recall are in a way competing metrics. When a classifier is designed, there is some form of threshold or a condition that is ultimately involved that separate the two classes. If we move the threshold in any direction, it affects precision and recall in opposite manner. In other words, if we try to improve the precision, the recall typically gets worse and vice versa. Hence in order to understand the core performance of the classifier in separating the two classes, a plot called receiver operating characteristics (ROC) is generated. The name ROC can be quite confusing, as there is no receiver in any of the considerations here. This whole theory has evolved in the subject of communications. In an electronic communication system there is a transmitter and receiver. In ideal situation, the receiver needs to decode the signals with no error as sent by the transmitter. However, due to noise on the transmission medium, there are always some errors. The ROC curve based analysis was developed to provide quantitative measure on the performance of the receiver. ROC curve is typically plotted in terms of *True Positive Rate, TPR* and *False Positive Rate, FPR*. These terms are defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{17.8}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{17.9}$$

As can be seen from the equations, *TPR* has same definition as *recall* and with some calculation we can see that *FPR* is related to *precision*. However, in current context, they are called as rate and we are going to vary these parameters as a function of threshold.

Figures 17.1 and 17.2 show two examples of ROC curve. The x-axis marks the FPR and the y-axis marks TPR. At the origin, both the rates are 0, as threshold is such that we are classifying all samples into non-desirable class. Then as we move the threshold, ideally the TPR should increase much faster than FPR. At the opposite end, we are classifying all samples into desired class and both rates are 1. In any given application we need to choose a threshold that provides the optimal performance under given constraints. However, one aspect of the ROC curve provides a metric that is independent of the application and is considered as the most fundamental property of the classifier and that is the area under the ROC curve or just *AUC*. The greater the area, the better the classifier.

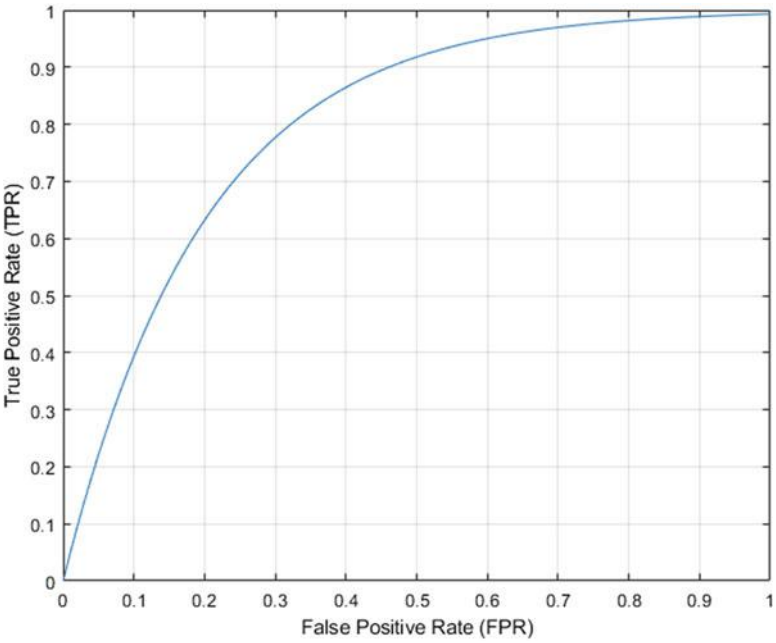


Fig. 17.1 Example of a relatively bad ROC curve

17.4 Hypothesis Testing

Hypothesis testing is essentially a probabilistic process of determining whether a given hypothesis is true. Typical explanation of this process involves quite a few concepts that are deeply rooted in theory of statistics and for a person not conversant with these concepts can quickly get lost. However, the concept of hypothesis testing is quite generic and we will study this concept with sufficient clarity here without going into too much detail.

17.4.1 Background

Before we dive deep into the process of hypothesis testing, let us first understand the background where we will apply this technique. We will take up a simple practical example to illustrate this. Consider daily weather predictions from the local news. Quite often we see that they are wrong, so we want to find out whether they actually have some scientific truth in them or they are just random predictions. Hypothesis testing framework can help in this analysis. Let us assume that there are 4 different types of weather conditions that are possible in a given region: sunny, cloudy, rainy, or snowy. Thus if one starts to make simple random predictions, those predictions

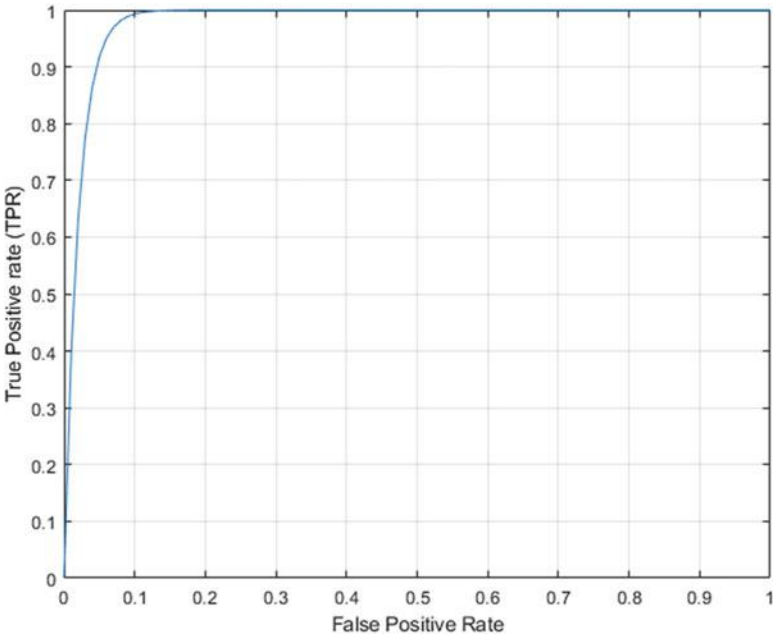


Fig. 17.2 Example of a relatively good ROC curve

would have only 25% chance of becoming true each day. Then we note the weather predictions for each day for over a period of say 6 months or 180 days. Then we find out what was the accuracy of the weather predictions of the local news channel and compare them with the default random value of 25% and determine whether the weather predictions from the local news channel are any better than pure chance using the theory of probability.

17.4.2 Steps in Hypothesis Testing

Entire process of hypothesis testing is split into four broad parts.

Steps in Hypothesis Testing

1. First step in the process involves defining what is called as null hypothesis H_0. Null hypothesis is essentially the default random behavior. In the example that we just described, null hypothesis would be: "weather predictions by the local news channel are random and purely based on chance."
2. Second step is to gather experimental or observational data. In the case of current example, that would mean the difference between the weather predictions recorded every day from the local news channel and actual observed weather

conditions. These observations constitute a random variable, called as *test statistic*.
3. Third step involves computation of the probability (also called as *P-value*) that the test statistic actually proves the null hypothesis.
4. Fourth step involves comparing the *P-value* with the predetermined level of significance, denoted as α. (Typically value of α lies between 5 and 1%) to either accept or reject the null hypothesis. The level of significance can be thought of as probability of test statistic lying outside of the expected region if null hypothesis is true.

17.4.3 A/B Testing

In many machine learning scenarios, we typically encounter comparison of more than one model for the solution of a problem. In such cases, we can extend the concept of hypothesis testing to what is called as *A/B testing*. In A/B testing, rather than using a default null hypothesis, we use hypothesis A, which is the outcome from the model-A. Then compare this with the outcome of model-B as second hypothesis. Following similar steps we can find out which model produces better results based on predetermined significance level.

17.5 Conclusion

Performance measurement is the most critical aspect of building a machine learning system. Without having a proper definition of quantitative metrics, one cannot effectively compare and contract different approaches or models. In this chapter, we studied the different metrics used for measuring the performance of machine learning systems. Different metrics are needed based on the application as well as the type and size of data, and depending on the needs, one can choose which metrics to use.

Part IV
Artificial Intelligence

> I know I've made some very poor decisions recently, but I can give you my complete assurance that my work will be back to normal. I've still got the greatest enthusiasm and confidence in the mission. And I want to help you.
>
> —HAL 9000, "2001: A Space Odyssey"

Part Synopsis

This part focusses on the implementation of the ML models to develop Artificially Intelligent applications.

Chapter 18
Classification

18.1 Introduction

We have discussed various algorithms that are designed for solving the classification problems in previous part. In this chapter, we are going to look at the topic in slightly different way. We are going to look at some real world problems that need underlying classification algorithms to solve. We will list a few problems that are well known and have made significant impact in the consumer space. We will try to build a machine learning pipeline to solve those problems in a rudimentary manner. Although we will not explicitly solve the problems, the process of tackling these classes of problems will give the reader a practical insight into the applications of machine learning in real world.

18.2 Examples of Real World Problems in Classification

Although classification is one of the simple applications of machine learning theory, the real world situations are not typically as simple as text book problems like classification of flowers illustrated by IRIS data set [3]. Here are few examples that have classification as underlying machine learning model:

1. Spam email detection: Classifying emails as genuine or spam. This is one of the crucial components of most email application and services these days. If an application can successfully separate the spams emails from genuine emails, it can quickly become application of choice by millions of people. The spam can further be divided into categories like deal alerts, newsletters, fishing emails, etc. making it a multiclass problem.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_18

2. Image classification: There are numerous applications of image classification. The images can be of faces and objective is to classify the faces into males and females. Or the images can be of animals and the problem can be to classify them into different species. Or the images can be of different vehicles and objective is to classify them into various types of cars.
3. Medical diagnosis: There are different types of sensing technologies available in medical diagnosis, like ultrasound, X-ray, magnetic resonance imaging, etc. This diagnostic sensory data can then be given to a machine learning system which can then analyze the signals and classify the medical conditions into different predetermined types.
4. Musical genre identification: When a piece of music is played, the machine learning system can automatically detect the genre of the music.

In order to understand the nuances of real life machine learning we will look at the first problem in greater detail in the rest of the chapter.

18.3 Spam Email Detection

Detecting spam emails against genuine emails is a very important and real world problem that most email services e.g. *Gmail*, *Hotmail*, *Yahoo Mail* etc. as well as email clients e.g. various email apps on mobile platforms, Microsoft Office 365 Outlook app, etc. face. However, the scope of the solution for each is very different. For example, email services can have access to all the emails from all the accounts in their platform, while the email clients can only access the accounts that are accessed through them. Again, as per the ever increasingly strict privacy rules, the extent to which these emails can be accessed is limited. In order to illustrate the problem solving methodology we will try to solve the problem of spam detection from the perspective of email service. The problem at hand is a very good example of binary classification. Let's define the precise scope of the problem.

18.3.1 Defining Scope

Let us consider that we have our own email service called as *email.com*. Let us assume we have access to all the emails for all the accounts that are opened on our email service in *aggregate* and *anonymous* manner. This means, only the automated algorithms can access the emails, no human can look at any specific message. Also, when we are dealing with errors and metrics, only aggregate level metrics can be accessed. There is no way to check why any specific message is being misclassified.

18.3.2 Assumptions

In real life the problem definition is never crisp like the one that appears in a textbook. The problem is typically identified in a broad sense and micro details need to be figured out. Case at hand is no different. Hence, we need to start making some assumptions in order to narrow down the scope of the problem further. These assumptions are essentially an extension of the problem definition itself.

18.3.2.1 Assumptions About the Spam Emails

1. The spam emails can be originated from *email.com* as well as from other domains.
2. The sender of the spam emails must be sending large amount of emails from his/her account.
3. Most emails sent by spammer account would be almost character-wise identical to each other with slight differences possible if the emails have been customized for each user. In such cases, each email will have a different greeting name, but the other contents would still be same.
4. Spam emails can fall into two categories: (1) Marketing and advertising emails; (2) Emails that are used to steal identity of a person or to make him/her commit some fraudulent financial transaction. In either of the case, the spam email would contain some URLs or fake email address(es) that will take the user to insecure destinations.
5. Spam emails are also created to look appealing and attractive and hence they are likely to be formatted as HTML rather than plain text.

18.3.2.2 Assumptions About the Genuine Emails

1. The genuine emails are the ones that are sent to the recipients with conveying useful information.
2. The recipient whether expects those emails or reads those emails to get the new information.

18.3.2.3 Assumptions About Precision and Recall Tradeoff

The assumptions listed above are likely to be true only for a certain percentage of times and we are certainly going to encounter exceptional cases where these assumptions would completely fail. This is the reason we can almost never achieve an accuracy of 100%. The objective is to find the optimal tradeoff between the two classes to maximize the overall accuracy. The key concept here is *optimal*. The concept of optimal can have multiple meanings based on the context. It need not

mean we should be equally accurate in detecting spam versus genuine, in other words it can be biased towards one. It may be desired to have the model so tuned that we will have much higher accuracy in detecting genuine emails (in other words high *recall* of detecting genuine emails) at the cost of detecting some spam emails as genuine (in other words low *precision* in detecting genuine emails), but when the model classifies an email as spam it is almost certainly a spam (in other words high *precision* for spam detection). Or we can have a completely opposite strategy or somewhere in between. In current context, let us keep the problem symmetric and unbiased and try to design a model that has roughly equal precision in detecting both classes.

18.3.3 Skew in the Data

The optimization of tradeoff between precision and recall is also affected by skew in the data. This is another dimension to the design of the classification system. Most of the statements in the previous section indirectly assume that there are equal number of samples from spam and genuine emails, or more generally speaking, the distribution of spam and genuine emails in the real world is roughly equal. This may not be the case. Let's assume for argument that the number of spam emails are only 40% and remaining 60% are genuine emails. In this situation, if we design the model to be equally accurate (say 80%) for precision of both classes, then resulting overall accuracy is also going to be 80%. However, consider a case when we design the model to be 90% accurate for genuine emails and as a result it is only 70% accurate for spam. Now let us see what the total accuracy is. The overall accuracy is going to be weights by the distribution of each class. Hence overall accuracy will be given as,

$$A_{\text{overall}} = (0.6 \times 0.9) + (0.4 \times 0.7) = 0.82 \tag{18.1}$$

Thus overall accuracy is slightly higher than the accuracy achieved with previous approach of treating both accuracies equally important. Small amount of skew is typically not as impacting on the overall accuracy, but when there is large skew e.g. (10–90%, etc.) then the design of the model needs to be carefully evaluated. If we want to remove the effect of the skew in the data in training model, we can use the technique of stratified sampling as discussed in Chap. 16. It is useful to note that as stratified sampling can be used to remove the skew, it can also be used to add a predetermined skew in the training data. For now, we will make the assumption that there is 60–40 skew in the data favoring genuine emails to spam emails.

18.3.4 Supervised Learning

It is safe to assume that this problem is strictly of supervised learning type and we have access to a labelled training set of spam and genuine emails. A set of about

100,000 labelled emails where say 60,000 emails are genuine and remaining 40,000 emails are spam should be sufficient to train the model. Unsupervised methods like clustering are capable of classifying the data into different clusters; however, as the labels are not used in separating the clusters, the resulting classes predicted by unsupervised clustering would be difficult to quantify.

18.3.5 Feature Engineering

A lot of important information is contained in the header of an email and its format is quite complex and contains multiple fields other than the simple *From*, *To*, *date* and *Subject*, like *User-Agent*, *MIME-Version*, and so on. All these fields can be part of the feature space used to train the classifier model.

The emails can be written as simple text or then can be formatted as HTML. The formatting type itself can be one feature. Then quite a bit of information can be extracted from the actual content. For example, number of words, number of lines, number of paragraphs, most repeated keywords, number of misspelled words, etc.

18.3.6 Model Training

A suitable classification algorithm or a set of algorithms need to be selected to train. With categorical features, decision tree type methods work very well. In this case, an optimized decision tree method of random forest is likely to work quite well. For simple training process, the 70–30 split would be sufficient. If one wants to further improve the model with hyperparameter tuning one can split the data 70–15–15 or 60–20–20 for train-validation-test. The validation step is used to train the hyperparameters.

If all the features are numerical, then neural networks can work well too. With use of neural networks validation step becomes mandatory. Also, to reduce the overfitting, regularization techniques must be used in the form of L1 or L2 regularization.

18.3.7 Iterating the Process for Optimization

Once the first version of the model is completed, it defines the baseline performance level. We can then iterate over each step of the process to improve the performance. One can play with addition and removal of more features, creation of compound features that are combination of existing features based on the domain knowledge of the field. Examples of such features can be: ratio of number of words to number of paragraphs, or difference between dates of original email and its reply, etc. Then one

can try and expand the hyperparameter search space, or play with stratified sampling and see the effect of on the performance. Apart from improving the performance in a static case, more important is to continuously update the model on a periodic level like every week or month, etc. The data is continuously changing and with time, the most optimized model of today can become outdated and inefficient. Also, in cases like spam, there always exists a continuous adaptation from the spammers to change the format of the spam emails to circumvent the spam detectors. Hence such systems need to be designed to be continuously evolving.

18.4 Conclusion

In this chapter, we looked at various popular real life problems that need underlying classification models to solve. We then looked at the specific example of email spam detection in detail by going from the assumptions need to be made to constrain the problem so that it can be solved using a suitable training data and machine learning algorithm. In the end we looked at how we can improve the overall process by iterating through all the steps.

Chapter 19
Regression

19.1 Introduction

In this chapter we will look at a real life application of regression technique. Regression problems are characterized by prediction of real valued output. The input features need not be numerical only though. They can range from numerical, categorial to pure string valued. We will take up the problem in its raw form, and then construct a problem suitable for a machine learning system to solve by making suitable assumptions, and also build a careful framework to analyze the results of the system in the form suitable for quantitative metrics.

19.2 Predicting Real Estate Prices

Predicting values of real estate is always one of the hot topics in our lives at various stages. The solution of the problem is useful to consumers as well as banks as well as real estate agents and so on. The prediction is going to be a real valued number of the dollar amount, and hence fit perfectly as a regression problem. It is important to identify the nuances of the problem further to narrow down the scope of it. When one talks about the real estate values there are many aspects to it and we must narrow down the definition to the specific problem we want to solve.

19.2.1 Defining Regression Specific Problem

To define the specific problem that we want to solve, let us first list down all the different aspects when one talks about real estate pricing.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_19

Aspects in Real Estate Value Prediction

1. Trends in price of a certain specific property over time, specifically in different seasons.
2. Price of certain type of property, e.g., a 2 bed condo, in certain area.
3. Average price of house in certain area.
4. Range of prices of a type of property in various different areas.
5. Price of a certain specific property at a given time.

Although the above list is far from comprehensive, it gives an idea about the possible different directions in which one can think when dealing with real estate prices. Each aspect defines a different problem from the machine learning perspective and will need different set of features and possibly even different type of model and different type of training data. For current context we will choose the last of the problem from the list to predict price of a certain specific property at a given time. Even this problem can be interpreted in significantly different directions when the type of property is taken into account. The property can be an empty land or a commercial property or a residential property with single family. In each case the factors that affect the price are drastically different and ultimately would lead to a completely different machine learning problem. For now, let us consider the problem of residential property with single family, specifically a single family house.

19.2.2 Gather Labelled Data

The next step is to identify the labelled set of data. As regression is a supervised learning problem without availability of appropriate labelled set of data, we cannot proceed.

From the perspective of the narrowed down problem we need a data that should include prices of a set of houses in a specific time frame, say from January 1 2019 to January 31 2019. We need to include the houses that are sold in that time frame in local neighborhood where the target property is located as well as some extended region around the neighborhood. The more the houses available the better. The rule of thumb here is to have at least 30 houses in the test set so that the predictions would have sufficient statistical significance. Typically we split the training and test set as 70–30%, so we need at least 100 houses in total. From model training perspective almost universally more is better within reason. It is also important to have the houses selected in training data to be similar to the house we want to predict on, but should have distribution on either side of it. For example if we want to predict the price of a house with 3 bedrooms and 2 bathrooms with 2500 square foot of area, we should have houses with less than and greater than 3 bedrooms, less than and greater than 2 bathrooms, and so on. This distribution makes the prediction problem as *interpolation* problem against an *extrapolation* problem. Most machine learning models perform significantly better in interpolation region compared to

extrapolation region. Figure 19.1 shows the difference between interpolation and extrapolation. Consider that x and y axes are plotting values of some features from the feature set, e.g., area of the house and area of the lot, etc. Interpolation is an easier and more deterministic problem where the behavior of the function that we are trying to map using suitable machine learning model is known. In the case of extrapolation, the behavior of the function in the specific neighborhood is not known. As long as the behavior is not too different in the region of extrapolation, the model can still predict reasonably accurate results, but the confidence in the prediction is lower.

19.2.2.1 Splitting the Data

Once the labelled data is gathered, we need to split it into 2 or 3 sets, as

1. Training Set
2. (Optional) validation set
3. Test set

The training and optional validation set is typically selected as 70 or 75% and test set is selected accordingly as 30 or 25%. The validation set is required to optimize the hyperparameters using methods like cross validation. If the model does not need hyperparameter tuning, validation set is not required.

Sampling
Ideally the training set should have identical distribution of the data compared to the test set, but in practice it is difficult to achieve. When the data sets are large (in thousands) simple random sampling can be sufficient. However, when the data set is small a technique called as *stratified sampling*[1] can be used to achieve this.

19.2.3 Feature Engineering

If you talk to a real estate agent, he or she may make the problem very simple by identifying three key features as *location*, *location*, and *location* and make the machine learning almost unnecessary. However, for now, let us use the single feature suggested by the agent and append it with our own feature set. Here are the factors that we can list down that would affect the price.

[1] When a population contains multiple sub-populations, especially with uneven distributions, the uniform random sampling from the joint population cannot guarantee a sample with sub-population distributions resembling to the original population. In such cases a technique called stratified sampling is used. In stratified sampling, each subpopulation is sampled separately and then these samples are mixed to create the overall sample.

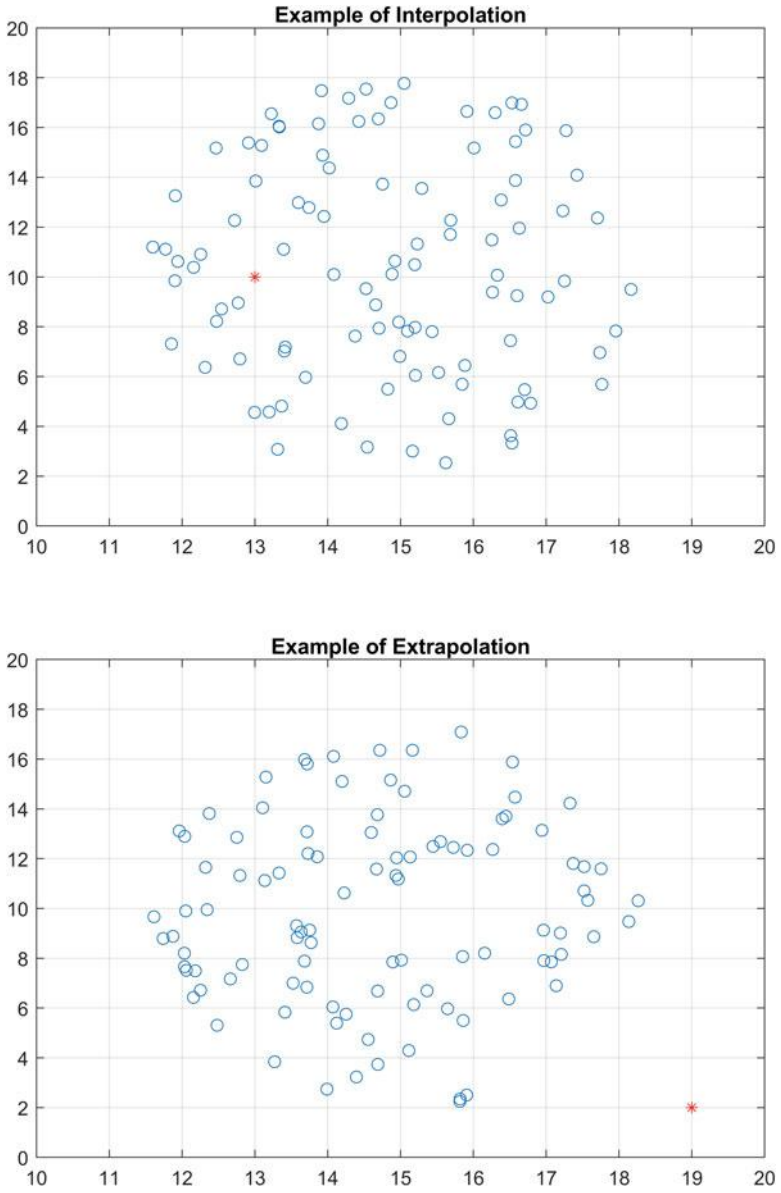


Fig. 19.1 Point where a value of the function needs to be predicted is shown as *asterisk*. Top figure shows an interpolation case where the point lies inside the region covered by the other points in the training set. Bottom figure shows an extrapolation case where the point lies outside the region covered by the other points in the training set

Factors Affecting the Price in Raw Form

1. Location
2. Size of the house
3. Type of the house
4. Builder Name
5. Year built
6. Information about car parking
7. Details of the interior

We can extend this list further but for the conceptual understanding this is sufficient. After identifying the factors in raw form as listed above, we need to convert them into more specific parameters that we can identify into the standard data types as numerical, strings, etc.

Factors in Standard Data Format

Most of the features listed in Table 19.1 can be directly used in machine learning model, except for the datetime features. The datetime features need to be handled based on the context. The direct use of the datetime string is not quite useful from the model perspective. From the model perspective the information that the field is giving is the age of the house. Thus we can convert the built date into age in the form of either number of years or number of months or even number of days based on whatever is appropriate. Some of the numerical features marked with superscript "a" can also be used as categorical as those fields are only going to have a small set

Table 19.1 Features for predicting price of single family house

Feature name	Data type
Location-zipcode	Numerical-categorical
Location-cityname	String-categorical
Location-county	String-categorical
Location-state	String-categorical
Location-streetname	String-categorical
Area of the house	Numerical
Area of the lot	Numerical
Number of bedrooms	Numerical[a]
Number of bathrooms	Numerical[a]
Number of floors	Numerical[a]
Building material	String-categorical
Builder name	String-categorical
Date built	String-datetime[b]
Car garage present	Binary
Size of car garage	Numerical[a]
Flooring type	String-categorical

[a]These numerical features can also be considered as categorical features

[b]Datetime features need to handled separately before using into machine learning model as discussed later in the chapter

of deterministic values. However, it is completely safe to use them as is in numerical form.

Thus we have identified a proper set of features that we can use to build the model. It is a good practice to plot the features along with labels to see the distribution of feature space and its correlation with the outcome. Techniques like *Principal Component Analysis* can be useful in this situation. Such plots give additional insights into the data that is going to be passed into the model and these insights help when we are iterating over the training process to optimize the performance metrics.

19.2.4 Model Selection

Using the repertoire of the algorithms learned in the previous part of the book we can select any of the suitable algorithms to solve the current problem. Here are few options:

1. Decision tree based regression
2. Logistic regression
3. Neural network
4. Support Vector Regression

Once the model is chosen, we can go through the corresponding training-validation process to optimize the model performance on the training data.

19.2.5 Model Performance

It is quite important to decide the correct set of metrics to compute the model performance. With incorrect choice of metrics, the entire process can lead to a very under-performing model providing suboptimal results. Given current case of regression, the suitable metrics could be:

1. RMS error in predicted value
2. MA error in predicted value
3. Mean and standard deviation or variance of the absolute error
4. Max absolute error

It is important to decide the bounds on each of the metric that make the resulting model acceptable. If the metrics do not lie within the bounds one need to iterate the whole or partial process. Here are few possible ways to iterate:

1. Use different model
2. Change the cross validation policy
3. Change, add or remove the features
4. Re-split the data

19.3 Other Applications of Regression

Above example illustrated the details in building a regression based application. There are many other areas where regression techniques are required. One such application of great importance marks an entire field of research, called as *Non-destructive Testing and Evaluation* or *NDT/E*. NDT/E encapsulates all inspection type applications of a system where the inspection needs to be performed without affecting or stopping the regular operation of the system. For example inspection of aircrafts, inspection of various components in a nuclear power plants or inspection of gas or oil carrying pipelines [65]. These inspections need to be performed without breaking open the aircraft, stopping the nuclear reactor, or stopping the pipeline from carrying any oil or gas. Various transducers are used to inspect the systems and the inspection data is collected in digital form. Then the data is analyzed to predict any defects or flaws in the system. Most such problems pose good examples of application regression techniques.

19.4 Conclusion

In this chapter, we studied the real applications based on the theory of regression. This analysis gives a glimpse towards the various practical considerations one needs to go through in the form of quantitative problem definition, constrained optimization followed by certain assumptions.

Chapter 20
Ranking

20.1 Introduction

Ranking at heart is essentially sorting of information. Unconsciously we are always ranking things around us based on some metrics. We rank the products based on their reviews. We rank players in various sports based on their performance in a season. We rank movies and music albums based on their earnings. We rank stocks based on their predicted growth, etc. In this regard ranking should be a simple rule in basic mathematics rather than being an aspect of machine learning. However, based on their applications, ranking has become quite popular and hot topic in machine learning and artificially intelligent systems. One of the glaring examples here would be exponential growth of Google in the twenty first century, which was truly based on their unique way to rank web pages.

Common applications of ranking are seen in search and information retrieval problems, recommendation systems, machine translation, data mining, etc. When one queries for a string in a search engine or queries for a movie name in online streaming websites, typically there are tens or hundreds of results that match the query with certain margin. No user is actually going to go through all these results, so some other quantitative measures need to be created that will help sort these hundreds and thousands of results in a manner that we can present only top 5 or top 10 of them to the user and present more results if requested by the user in continuing order. This would make the user experience vastly superior than random presentation of the results. Ranking defines the underlying model that is capable of ordering the results in optimum manner before presenting to the user. As the ranking problem fundamentally differs from the other machine learning problems, it also needs a separate set of metrics to analyze. In this chapter we will look at different practical examples of ranking systems and also look at various techniques and metrics used in the process.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_20

20.2 Measuring Ranking Performance

Subjectively, ideal ranking algorithm would be defined as:

Definition 20.1 The algorithm that can rank the items in strictly non-increasing order of relevance.

The relevance factor here is a non-trivial measure that needs to be defined for each ranking problem. Now, let's try to convert this subjective definition to mathematical expression. Let there be n number of items that need to be ranked and each has a relevance score of $r_i, i = 1, 2, \ldots, n$. A simple measure, called as cumulative gain (CG) is defined on these relevances as,

$$\text{CG} = \sum_{i=1}^{n} r_i \tag{20.1}$$

CG essentially represents the overall quality of aggregate search results. As there is no positional factor in the expression, CG does not provide any information about the ranking. Hence a modified measure is defined as discounted cumulative gain or DCG. DCG is defined as,

$$\text{DCG} = \sum_{i=1}^{n} \frac{r_i}{\log_b i + 1} \tag{20.2}$$

The base b of the logarithm is typically used as 2. The reason for using $i + 1$ instead of i is to account for the case when $i = 1$. In this case, the denominator would be 0 and the metric would be meaningless. This expression penalizes the ranking if highly relevant item is ranked lower. In order to see the effect of this expression, let's take a real example. Table 20.1 shows a set of items with corresponding relevance scores ranging from 0.0 to 1.0. Score of 0.0 means no relevance at all and 1.0 is the highest possible relevance; however, the maximum relevance in the given set is 0.6. The third column in the table gives the corresponding discounted relevance score as defined in Eq. 20.2 (without summing the values).

If we sum the discounted scores, then we get DCG as 1.20. As we can see this ranking is not ideal, as the highest ranked item is at rank 7 and all the items are fairly arbitrarily ranked. Now, let's fix the rankings and recalculated the discounted relevance scores as shown in Table 20.2.

Now, if we sum the discounted relevance scores, we get 1.53, which is significantly higher than the previous score, that was produced with random ordering of the items. Thus we have a good metric in the form of DCG that can compare two different sets of rankings if we have corresponding relevance scores. Although there is one slight drawback in this system and that is presence and position of the items with 0.0 score. If we remove the items with score 0.0, we will still have the same score. However, in reality, if we present the items to the user, the set without items

Table 20.1 Sample set of items to be ranked with relevance score

Item no.	Relevance score	Discounted relevance score
1	0.4	0.4
2	0.25	0.16
3	0.1	0.05
4	0.0	0.0
5	0.3	0.12
6	0.13	0.05
7	0.6	0.2
8	0.0	0.0
9	0.56	0.17
10	0.22	0.06

In this case they are ranked arbitrarily

Table 20.2 Sample set of items ideally ranked with non-increasing relevance score

Item no.	Relevance score	Discounted relevance score
1	0.6	0.6
2	0.56	0.35
3	0.4	0.2
4	0.3	0.13
5	0.25	0.1
6	0.22	0.08
7	0.13	0.04
8	0.1	0.032
9	0.0	0.0
10	0.0	0.0

with 0.0 score would be better than the one with them. However, penalizing the items with score of 0.0 without affecting the regular operation of DCG is quite a bit more complicated and typically used as is.

There is one more problem with DCG and that is the actual value of a score is quite arbitrary based on the range of values of relevance scores. For example, if recalibrate our relevance scores to range from 0 to 5 instead of 0–1 as before, our ideal DCG would jump from 1.53 to 7.66 and our random DCG would jump from 1.20 to 6.01. All these values are fairly arbitrary and the value of 6.01 has not much meaning without knowing the ideal value of DCG, which is 7.66. Hence another metric is introduced as normalized DCG or nDCG. nDCG is defined as,

$$\text{nDCG} = \frac{\text{DCG}}{\text{iDCG}} \tag{20.3}$$

where iDCG is the value of ideal DCG.

20.3 Ranking Search Results and Google's PageRank

All the metrics discussed above hold true and provide a good measure of ranking quality, only if we have a single numeric measure of the relevance score for each item. However, coming up with such score is not always a trivial task. Rather the goodness of such measure is the true deciding factor in building a good ranking system.

Having better search results was the key aspect of exponential growth of Google in first decade of this millennium. There were already many search engines available at that time, and they were not really bad. However, Google just had better results. All the search engines were crawling the same set of websites and there was no secret database that Google was privy of. The single aspect where it just did order of magnitude better than the competition was the ranking of the search results. The concept of *PageRank* [33] was at the heart of Google's rankings. At the time before Google's PageRank came to existence, the commonly used technique for ranking the pages was number of times a particular query appeared on the page. However, PageRank proposed an entirely new way of ranking the pages based on their importance. This importance was calculated based on how many other pages are referencing given page and importance of those pages. This system created a new definition of relevance metric and ranking based on this metric proved to be extremely effective in getting better search results and rest is history!

20.4 Techniques Used in Ranking Systems

Information retrieval and text mining are the core concepts that are important in the ranking system that relate to searching websites or movies, etc. We will look specifically at a technique called as keyword identification/extraction and word cloud generation. This method underpins most of the text mining systems and knowledge of these techniques is invaluable.

20.4.1 Keyword Identification/Extraction

Figure 20.1 shows a word cloud from the famous Gettysburg address by Abraham Lincoln in 1863. A word cloud is a graphical representation of the keywords identified from a document or set of documents. The size of each keyword represents the relative importance of it. Let's look at the steps required to find these importances.

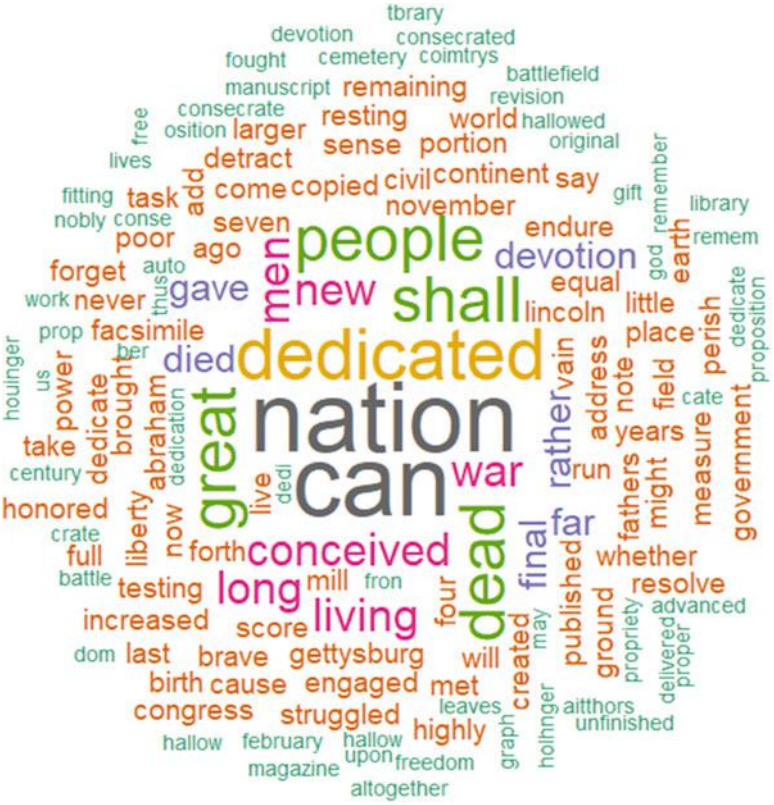


Fig. 20.1 Word cloud from famous Gettysburg address by Abraham Lincoln

Steps in Building a Word Cloud

1. Cleaning up the data. This step involves removing all the formatting characters as well as punctuation marks.
2. Normalization of the data. This step involves making all the characters lower case, unless it is required to distinguish the upper case letters from lower case. Another aspect of normalization is lemmatization or stemming. This step involves applying grammar to find the root word for every word form. For example root word for *running* is *run* and so on.
3. Removal of stop words. This step involves removing all the common words that are typically used in any text and have no meaning of their own. For example a, them for, to, and etc.
4. Compute the frequency of occurrence of each of the remaining words.
5. Sort the words in descending order of frequency.
6. Plot the top-n keywords graphically to generate the word cloud.

Above analysis works well for a single document case. When there are multiple documents, then there is one more technique that is useful. It is called as *TF-IDF*, or *term frequency - inverse document frequency*. Term frequency is calculated in similar way as described above. The inverse document frequency is an interesting concept and it is based on the notion of how much important the given keyword is in the current document as opposed to other documents. Thus a word that appears frequently across all the documents will have low value of IDF in any of these documents. However, when a keyword appears frequently in only one document, but is almost not present in the rest of the documents, then its importance in the given document is even higher. Mathematically, both the terms are defined as,

$$\text{tf} = \frac{\text{frequency of the word in given document}}{\text{Maximum frequency of any word in the given document}} \tag{20.4}$$

$$\text{idf} = \frac{\text{frequency of the word in given document}}{\text{Total number of documents in which the word appears}} \tag{20.5}$$

Thus combining tf and idf, we can generate even better measure for keyword importance when dealing with multiple documents.

20.5 Conclusion

In this chapter we looked at the problem of ranking and its evolution in the context of machine learning and artificial intelligence from simple ordering of items to information retrieval and resulting user experience. We studied the different measures for computing relevance and metrics for comparing different sets of ranking along with different techniques used in ranking documents.

Chapter 21
Recommendations Systems

21.1 Introduction

Recommendation system is a relatively new concept in the field of machine learning. In practical terms a recommendation is a simple suggestion given by a friend or colleague to another friend or colleague to watch a movie or to eat in a certain restaurant. Most important thing about the recommendations is that they are very personal. What we recommend to one friend as top choice for restaurant, we may not recommend to another one at all depending on his/her likings. Typically, the person recommending does not have any ulterior motive behind the recommendations, but in some cases that may not be true as well. For example if one of my good friend has started a new restaurant, then I might recommend it to my other friends irrespective of their likings too.

When building a recommendation system on large scale, one must express these relationships quantitatively and formulate a theory to optimize a cost function to obtain a trained recommendation system. The core mathematical theory underlying modern recommendation systems is commonly called as collaborative filtering. Companies like Amazon and Netflix have influenced this genre of machine learning significantly for personalizing shopping and movie watching experiences respectively. Both the systems started off with techniques almost as simple as table joins, but soon evolved into a sophisticated recommendation algorithms. Netflix had actually announced an open competition for developing the best performing collaborative filtering algorithm to predict user ratings for films [35]. A team called *BellKor* won the grand prize in 2009 by achieving more than 10% better accuracy on predicted ratings than Netflix's existing algorithm.

We will first study the concepts in collaborative filtering and then we will look at both the cases of Amazon and Netflix in the context of recommendation system and understand the nuances of each in this chapter along with learning various concepts and algorithms that are involved in building such systems.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_21

21.2 Collaborative Filtering

Being quite new and cutting edge technology, there are multiple different interpretations of the collaborative filtering based on the context and application. But in broad sense the collaborative filtering can be defined as a mathematical process of predicting the interests or preferences of a given user based on a database of interests or preferences of other users. Very loosely speaking, these predictions are based on the nearest neighbor principle. Users with similar interests in one aspect tend to share similar interest in other similar aspects. To consider an example and oversimplifying the concept, if persons A, B and C like movie *Die Hard - 1* and persons A and B also like movie *Die Hard - 2*, then person C is also probably going to like the movie *Die Hard - 2*.

Collaborative filters always deal with 2-dimensional data in the form of a matrix. One dimension being the list of users and the other dimension being the entity that is being liked or watched or purchased, etc. Table 21.1 shows a representative table. The table is always partially filled, representing the training data. The goal is to predict all the missing values. Table 21.1 shows values that are binary, but they can as well be numeric or categorical with more than two categories. Also, you can see that the top 10 users have mostly known ratings, while bottom 10 users have mostly

Table 21.1 Sample training data for building a recommendation system

Persons	Electronics	Books	Travel	Household	Cars
Person1	Like	?	Dislike	Like	?
Person2	Dislike	Like	Dislike	Like	Like
Person3	Dislike	Like	Like	?	Dislike
Person4	Like	Like	Like	Like	Dislike
Person5	Like	Dislike	Dislike	Dislike	Like
Person6	?	Like	?	?	Dislike
Person7	Like	?	Dislike	Like	?
Person8	Like	Like	?	?	Like
Person9	Dislike	?	Dislike	Like	Like
Person10	Dislike	?	Like	Like	?
Person11	?	?	Dislike	?	?
Person12	?	?	?	?	?
Person13	?	?	Like	?	?
Person14	?	Like	?	?	?
Person15	?	?	?	?	Like
Person16	?	?	?	?	?
Person17	Like	?	?	?	?
Person18	?	Like	?	?	?
Person19	?	?	Dislike	?	?
Person20	Dislike	?	?	?	?

unknown ratings. This is a typical situation in practice, where substantial amount of data can be quite sparse.

21.2.1 Solution Approaches

The solution of the problem defined in the previous subsection can be tackled in few different options. However, the core information that can be possibly available is of three types.

Information Types

1. Information about the users in the form of their profiles. The profiles can have key aspects like age, gender, location, employment type, number of kids, etc.
2. Information about the interests. For movies, it can be in the form of languages, genres, lead actors and actresses, release dates, etc.
3. Joint information of users' liking or rating their interests as shown in the Table 21.1.

In some cases, one or more sets of information may not be available. So the algorithm needs to be robust enough to handle such situations. Based on the type of information used, the algorithms can be classified into three different types.

Algorithm Types

1. Algorithms exploiting the topological or neighborhood information. These algorithms are primarily based on the joint historical information about the users' ratings. They use nearest neighbor type methods to predict the unknown ratings. These algorithms cannot work if the historical data is missing.
2. Algorithms that exploit the structure of the relationships. These methods assume a structural relationship between users and their ratings on the interests. These relationships are modelled using probabilistic networks or latent variable methods like component analysis (principal or independent) or singular value decomposition. These methods use the joint information as well as the separate information about the user profiles and interest details. These methods are better suited to tackle sparse data and ill conditioned problems, but they miss out neighborhood information.
3. Hybrid approach. Most recommendation systems go through different phases of operation, where in the beginning the ratings data is not available and the only data that is available is user profiles and interest details. This is typically called as cold start. The neighborhood based algorithms simply cannot operate in such situations. Hence these hybrid systems start with algorithms that are more influenced by the structural models in the early stages and later on combine the advantages of neighborhood based models.

21.3 Amazon's Personal Shopping Experience

In order to understand the problem from Amazon's perspective, we need to explain the problem in more detail. First and foremost Amazon is a shopping platform where Amazon itself sells products and services as well as it lets third party sellers sell their products. Each shopper that comes to Amazon comes with some idea about the product that he/she wants to purchase along with some budget for the cost of the product that he/she is ready to spend on the product as well as some expectation of date by which the product must be delivered. The first interaction in the experience would typically begin with searching the name or the category of the product, e.g., "digital camera." This would trigger a search experience, which would be similar to the search discussed in the previous chapter on ranking. This search would come up with a list of products that are ranked by relevance and importance of the different items that are present in Amazon's catalog. Amazon then lets user alter the sorting by price, relevance, featured, or average customer review. These altered results are still just extension of the ranking algorithm and not really the recommendations. The "featured" criteria typically is influenced by the advertising of the products by sellers (including Amazon). Once user clicks on the one item in the list that he/she finds interesting based on price, customer reviews, or brand, etc., the user is shown the details of the product along with additional set of products gathered as *Frequently bought together* or *Continue your search* or *Sponsored products related to this item*. These sets represent the direct outcome of recommendation algorithm at work.

21.3.1 Context Based Recommendation

Amazon probably has many millions of products in their catalog, but how many products should be recommended needs have some limitation. If Amazon starts putting a list of hundreds of products are recommended ones, user is going to get lost in the list and may not even look at them or even worse, move to another website for getting confused. Also, if too few or no recommendations are shown then Amazon is losing potential increase in sales of related items. Thus an optimal balance between the two must be observed in order to maximize the sale and minimizing the user distraction. The product recommendations shown here are context based and are related to the search query only. Once the number of recommendations needs to be shown is finalized, then comes the question of which products to recommend. There can be multiple aspects guiding this decision.

Aspects Guiding the Context Based Recommendations

1. Suggesting other similar products that are cheaper than the product selected. So, if the cost is the only aspect stopping the user from buying the selected item, the recommended item can solve the problem.

2. Suggesting a similar product from a more popular brand. This might attract user to buy a potentially more expensive product that is coming from more popular brand, so more reliable or better quality.
3. Suggesting a product that has better customer reviews.
4. Suggesting a set of products that are typically bundled with the selected product. These suggestions would right away increase the potential sale. For example suggesting carry bag or battery charger or a memory card when selected item is a digital camera.

21.3.2 Personalization Based Recommendation

The recommendations based on search results are not quite personal and would be similar for most users. However, the real personalization is seen when one opens the www.amazon.com the first time before querying for any specific product. The first screen that is shown to the user is heavily customized for the user based on his/her previous searches and purchases. If the user is not logged in, the historical purchases may not be available. In such cases the browsing history or cookies can be used to suggest the recommended products. If nothing is available, then the products are shown that are in general popular among the whole set of Amazon buyers. These different situations decide which algorithm of the collaborative learning needs to be applied. For example the last case represents the cold start situation and needs a model based approach to be used, while in other cases, one can use a hybrid approach.

21.4 Netflix's Streaming Video Recommendations

Netflix was one of the pioneers of the modern day recommendation systems, specifically through the open contest as well as being one of the first players in the online streaming genre. In early days the ratings provided on Netflix were similar to the ones on www.imdb.com or other movie rating portals. These portals have different ways to generate these ratings. Some ratings are generated by the movie editors that specifically curate these ratings for a given site, some ratings are created as aggregates from different newspapers and other websites, while some ratings are aggregated by user ratings. One of the most important differences here is that the ratings given on websites are same for all the people visiting the sites. These movie sites are not personalized for any user and as a result the ratings that are available on these websites are average ratings that might not mean much for any individual user other than finding some of the top rated movies of all time or from some specific genre.

Netflix's rating system differs here in the fundamental way. Each user sees a very different screen when he/she logs in to their Netflix account, in spite of Netflix having same set of movies available for all the users to watch. Hence the ratings

Table 21.2 Sample training data for Netflix to build video recommendation system

	Movie-1	Movie-2	Movie-3	Series-1	Series-2
Persons	Rating, views	Rating, views	Rating, views	Rating, views	Rating, views
Person1	8[a], 2	2, 1	?, 1	7, 2	?, ?
Person2	1, 1	9, 2	4, 1	7, 1	8, 1
Person3	3, 1	6, 2	7, 1	?, 1	3, 1
Person4	9, 3	8, 2	5, 1	7, 2	1, 1
Person5	6, 1	2, 1	2, 1	4, 1	9, 3
Person6	?, 0	6, 0	?, 0	5, 1	4, 2
Person7	7, 1	?, 2	2, 3	9, 4	?, 1
Person8	10, 2	8, 2	7, 1	?, 0	8, 0
Person9	3, 1	?, 0	?, 0	2, 2	10, 0
Person10	1, 1	?, 0	8, 1	10, 5	4, 1
Person11	?, 0	?, 0	2, 1	?, 0	?, 0
Person12	?, 0	?, 0	?, 0	?, 0	?, 0
Person13	?, 0	?	8, 2	?, 0	?, 0
Person14	2, 1	5, 1	?, 0	?, 0	?, 0
Person15	?, 0	?, 0	?, 0	?, 0	7, 1
Person16	?, 0	?, 0	?, 0	?, 0	?, 0
Person17	6, 2	?, 0	?, 0	?, 0	?, 0
Person18	?, 0	9, 2	?, 0	?, 0	?, 0
Person19	?, 0	?, 0	1	1, 0	?, 0
Person20	1, 1	?, 0	?, 0	?, 0	?, 0

[a]Ratings are between 1 to 10. 10 being most liked and 1 being least liked

or recommendations that Netflix shows to each user are personalized for that user based on their past viewing and/or search history. Netflix is not interested in having a generic set of ratings that produce top rated movies of all time or specific time in specific genre, but is more interested in catering to the specific user that is currently logged in and is interested in watching something. The recommendations have one single motive and that is to maximize the watching time on Netflix.

A sample data set that Netflix would have to use to build their recommendation system would be similar to the one shown in Table 21.2. For new users representing the bottom half of the table, there would be little to no ratings/viewings available, while for older users there will be richer data.

21.5 Conclusion

In this chapter, we studied the collaborative filtering technique. We then looked at the specific cases of recommendation systems deployed by Amazon and Netflix. Recommendation systems represent a relatively new entry into the field of machine learning and it is a cutting edge of the area and is under strong development.

Part V
Implementations

Do or do not, there is no try!!

—Yoda, “Star Wars: The Empire Strikes Back”

Part Synopsis
In this part we will go over couple of options to implement simple ML pipelines. This will make all the concepts discussed so far concrete and will give a hand-on experience to the reader.

Chapter 22
Azure Machine Learning

22.1 Introduction

Having learned the different machine learning techniques and how they can be used to build artificially intelligent applications in the previous chapters, we have now reached a point where we are ready to implement such systems and use them. There are multiple options to choose when one comes to implementation. Most of them involve open source libraries in *Python* or *R*[1] or even more mainstream programming languages like *Java*, *C#*, etc. However, using these libraries is a little more involved task and we will look at that option in the next chapter.

22.2 Azure Machine Learning Studio

In this chapter, we will focus on Microsoft *Azure Machine Learning studio*, also called as *AML(V1)*. We will call it just AML henceforth. AML provides a very simple and intuitive flowchart type platform for machine learning model implementations. Most of the functionality that we discussed in the previous chapters is available in the form of pre-built blocks. The blocks can be connected to form the data processing pipeline. Each block allows for some customization. Although AML studio is not exactly open source option where you can look through the code behind all the models, and update as needed, it is nonetheless a free to use option that is perfect to start. One can start building sophisticated machine learning models using AML studio in matter of hours. For the person who is completely new to the field, this is a much desired option that will boost the confidence.

[1] R is a statistical analysis language that is open source. It has a huge community of followers and developers who have added lot of packages in machine learning as well.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_22

22.2.1 How to Start?

The first place one needs to visit is: https://studio.azureml.net/. Here user is greeted with the screen shown in Fig. 22.1. Here one can sign in with a Microsoft account, e.g., hotmail.com or outlook.com, etc. No purchase is necessary to start building machine learning models. Once you sign in, you are presented with a screen as shown in Fig. 22.2 There are multiple options available as you can see the menu on the left side, but we are only going to use the feature called *Experiments*. Currently it shows that there are no experiments created. So let's start with the first one. We will use the classic *Iris Data* based multiclass classification to illustrate the main components in the machine learning pipeline using AML studio.

Let's click on the + *New* option on the bottom left of the screen. This presents with screen shown in Fig. 22.3. It gives options to load multitude of pre-configured experiments. One can go through them to understand different features of the studio, but for now, we will start with *Blank Experiment*. This way we can see how one can start from scratch and build the end to end machine learning pipeline using AML studio.

When the option of *Blank Experiment* is selected, user is presented with a blank experiment canvas in the center with all the pre-built machine learning tools available on the left panel. The right side panel shows properties of a specific tool selected in the canvas as shown in Fig. 22.4. Now, we are all set to get the Iris data in the experiment and build the machine learning pipeline.

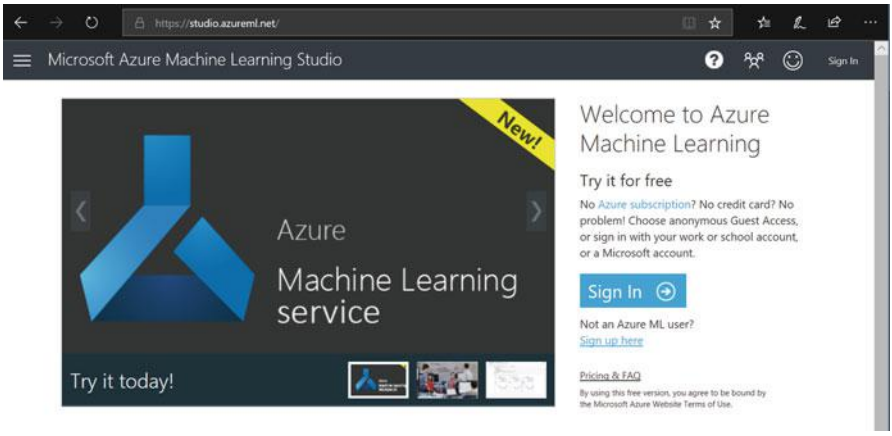


Fig. 22.1 Sign In screen for Azure machine learning studio

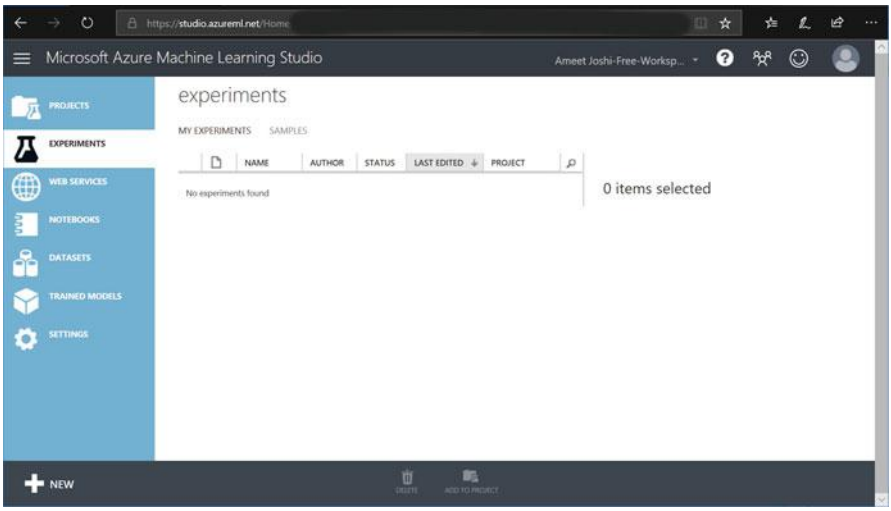


Fig. 22.2 Default screen after you login to Azure machine learning studio

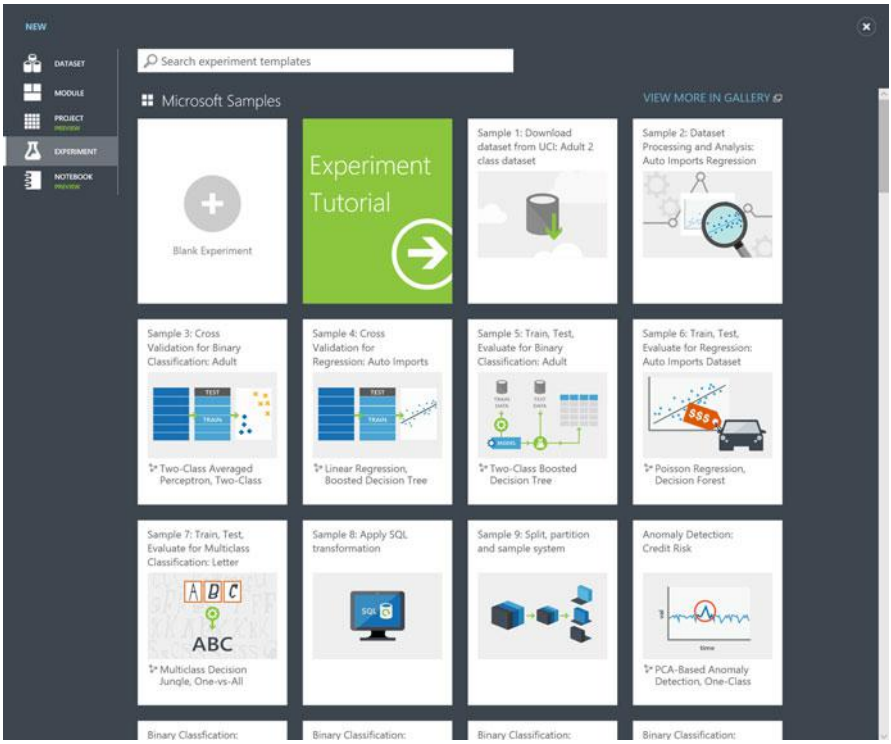


Fig. 22.3 New experiment screen

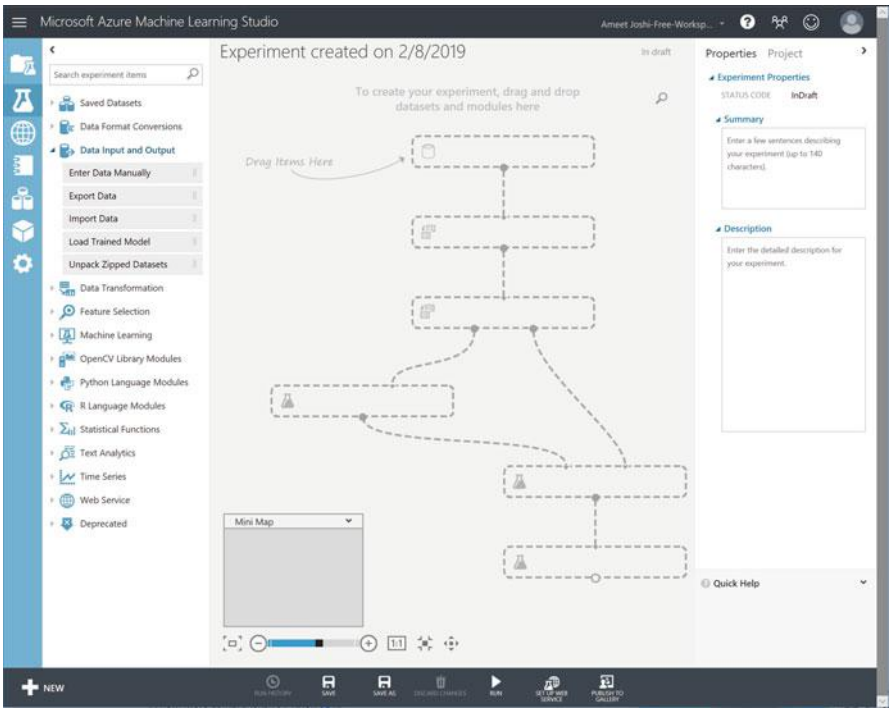


Fig. 22.4 Blank canvas for a new experiment

22.3 Building ML Pipeline Using AML Studio

In this section we will go over all the steps to build the multiclass classification pipeline using AML studio.

22.3.1 Get the Data

The first step in the process is to import the data. AML provides various different options to import data using *Import Data* block as shown in Fig. 22.4. Figure 22.5 shows all the options. However, as Iris data is quite small, we will just use the block named *Enter Data Manually*. This block can be brought to the experiment canvas by drag and drop. Once you click on the block on the right panel it shows options on how you want to enter the data. Let's choose the default option of *CSV* and paste the CSV data in *Data* block as shown in Fig. 22.6. You can copy the data directly from here [3].

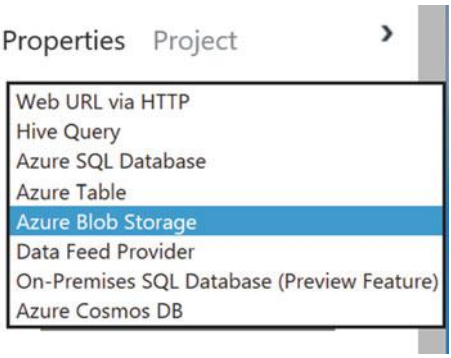


Fig. 22.5 Various data import options

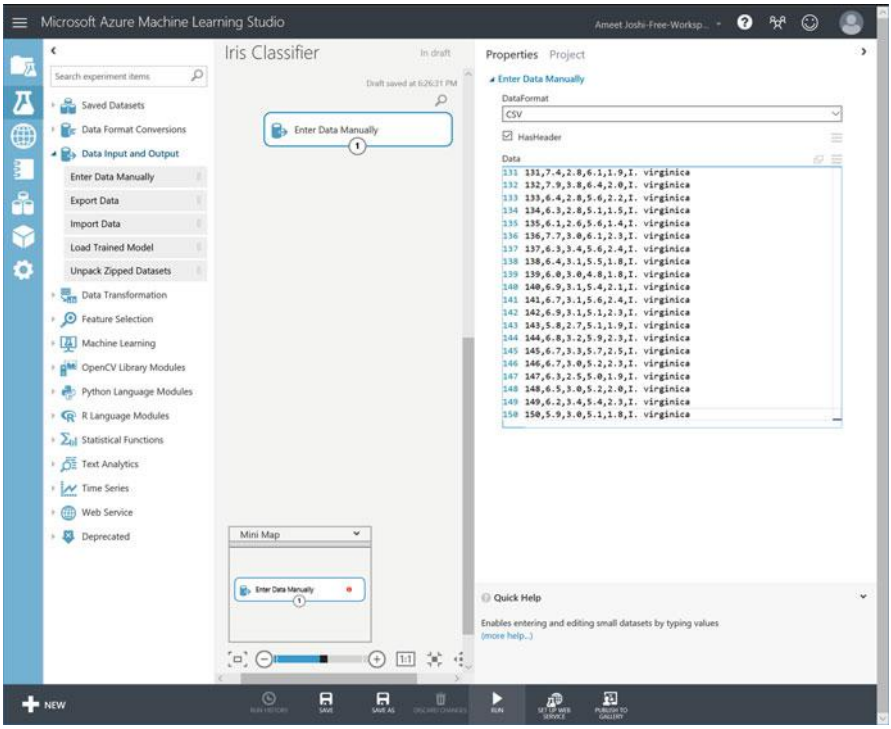


Fig. 22.6 Entering the data manually

22.3.2 Data Preprocessing

The manually entered data contain features and labels all in single set. We need to separate them and label them accordingly so that AML studio can process the features and labels accordingly. This can be done in two steps:

1. Use *Select Columns in Dataset* block in *Data Transformation->Manipulation* in the left panel.
2. Use the block two times to select features and label separately.

To select the columns, just click on *Launch column selector* in right panel. That brings the UI shown in Fig. 22.7. Use the UI twice to select features in set and labels

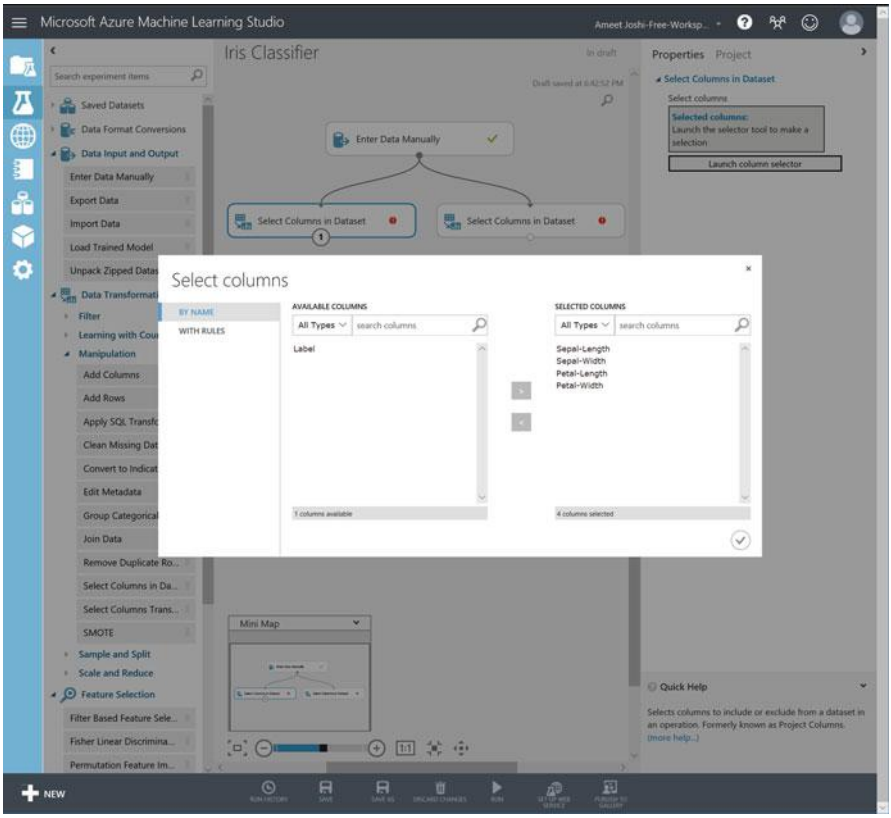


Fig. 22.7 UI for selecting columns

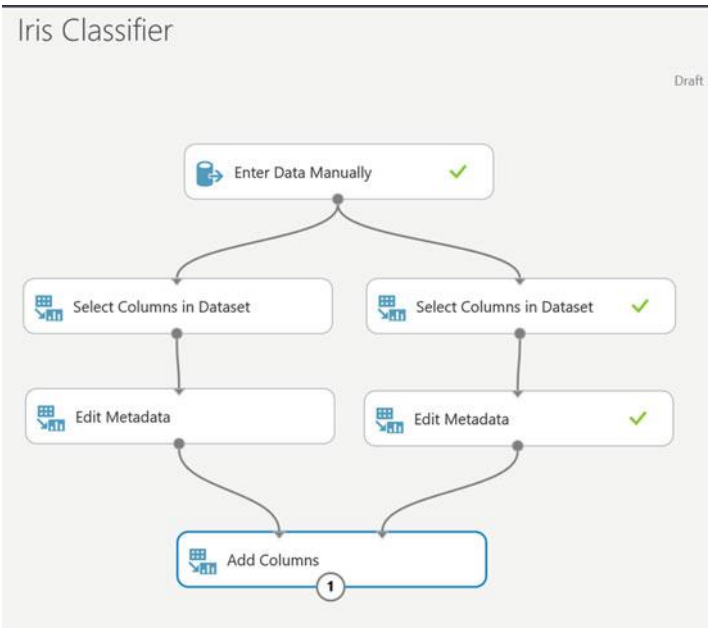


Fig. 22.8 ML pipeline at the end of data preprocessing

in another. The subsequent block in the pipeline can be connected by dragging the mouse from the connector bubbles at the top and bottom of the blocks as shown in Fig. 22.7. The next step is to edit the metadata to let AML studio know the features and labels. This can be done using the *Edit Metadata* block from the left panel. At any point we can run the pipeline that is built so far by clicking on the *Run* button at the bottom panel. Also at this point, you can give a suitable name to the experiment in the top left corner of the canvas and save the experiment using the save button in the bottom panel.

Once features and labels metadata is updated, we can add the columns back into a single data set using *Add Columns* block. The pipeline should be like the one shown in Fig. 22.8 so far. This data is clean and has no missing values. Also, as all the features are already in numeric form, our data preprocessing step is now concluded.

Once any block of the pipeline is executed successfully, it is marked with a green colored tick mark. The output of such blocks can be visualized for getting insights into the data processing. Figures 22.9 and 22.10 show couple of examples of the visualization, where you can also see the histogram of the data.

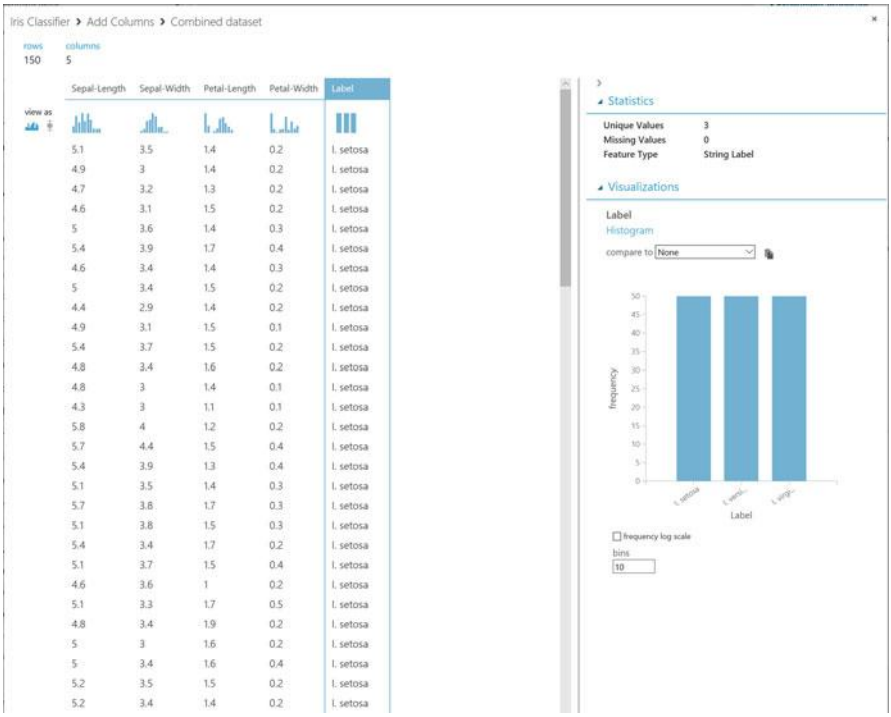


Fig. 22.9 Visualization of the labels

22.3.3 Training the Classifier Model

Before we start training process, we need to choose appropriate algorithm to use. In case of multiclass classifier, we can use decision trees or neural network or simple logistic regression based classification. As first attempt let us choose decision jungle, which is an example of ensemble decision tree. We can bring the block *multiclass decision jungle* from *Machine Learning -> Initialize Model -> Classification* on the left panel.

We also need to split the data into training and test sets. The block named as *split data* can be brought to experiment canvas from *Data Transformation-> Sample and Split* on left panel. As we are dealing with a very small sized data, uniform random sampling might distort the distributions between the two sets, so we should use stratified sampling based on the label values. When you click on the *split data* block it shows the option to enable the stratified sampling on the right panel, and then lets

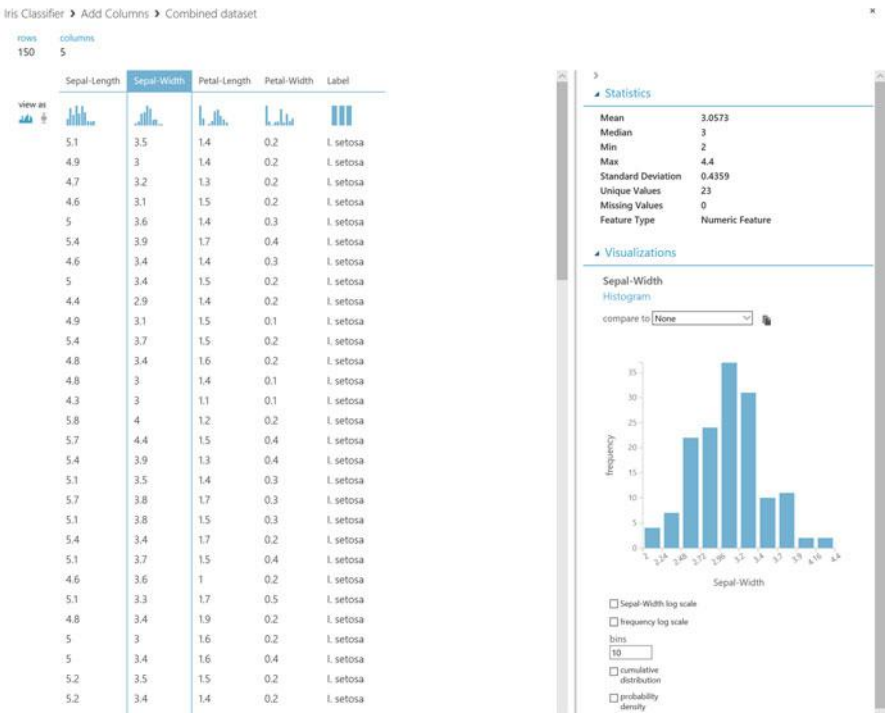


Fig. 22.10 Visualization of one of the features

you choose the column on which the sampling should be based. Let's choose 70% data for training and 30% for test.

Then, bring in the block *Machine Learning -> Train -> Train Model* to the experiment canvas. It takes two inputs: (1) The output of the *multiclass decision jungle* and (2) The training set from *split data*. The experiment pipeline should look as shown in Fig. 22.11.

22.4 Scoring and Performance Metrics

The next step in the processing is to score the test set with the trained model. We can use the block *Score Model* from *Machine Learning -> Score* on left panel. It takes the two inputs as trained model and test data. After we score the test set we

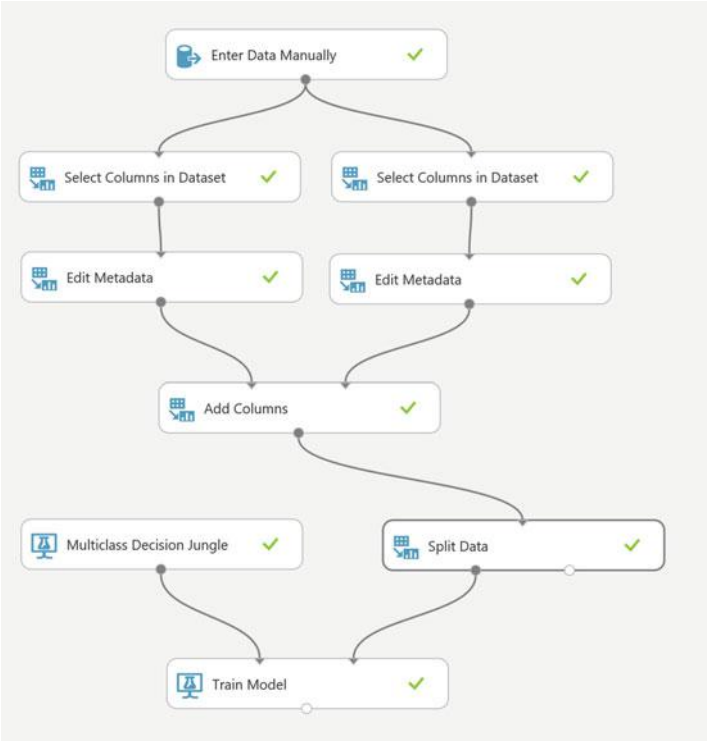


Fig. 22.11 ML pipeline till training the model

can compare the test results with the expected classes from the known labels.[2] This can be done using the block *Evaluate Model* from *Machine Learning -> Evaluate* on the left panel. Once all the blocks are connected, the whole pipeline should be like the one shown in Fig. 22.12. The evaluate block not only computes the accuracy, it also shows the full *confusion matrix* that shows how the model classifies each sample as shown in Fig. 22.13. As we can see from confusion matrix, the classifier is classifying the class *Versicolor* with 100% accuracy but is not able to separate the other two classes as well. As the data is split randomly, one can repeat the runs to see how much of difference it makes to the model performance. If there is significant difference between two such runs, it means that there is a strong skew in the data. Ideally multiple runs should produce very similar results.

[2]This set of known labels from the test data is commonly called as ground truth in machine learning literature.

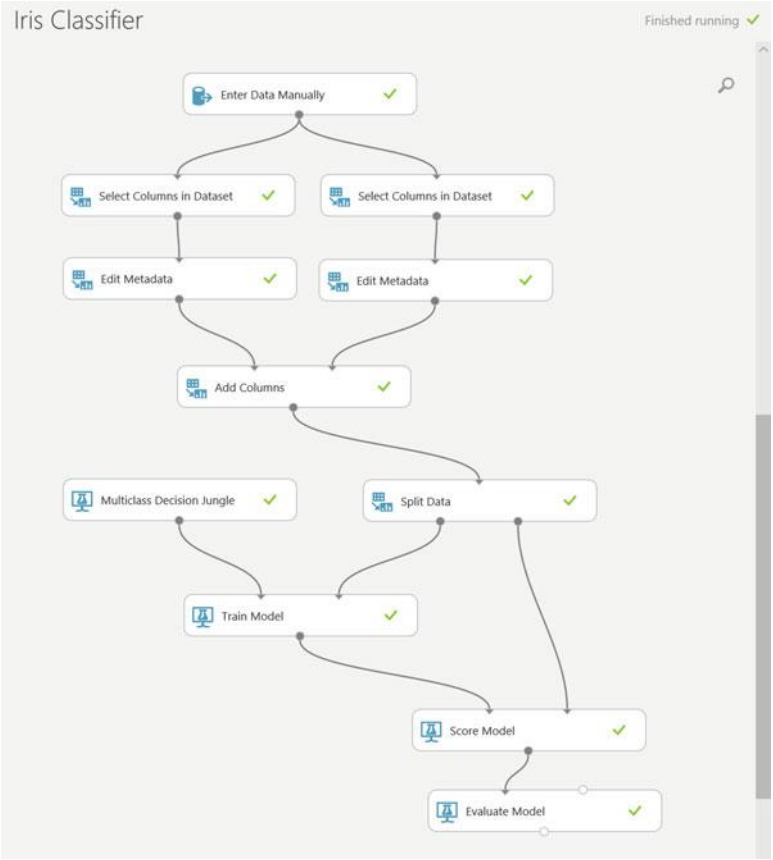


Fig. 22.12 Full ML pipeline from entering the data to evaluating the model using decision jungle multiclass classifier

22.4.1 Comparing Two Models

Now, we have a full machine learning pipeline working with decision jungle as the underlying algorithm. Next, we can train the model using another algorithm and compare the two results of the two. As you can see in Fig. 22.12, the evaluate block can take two inputs, but we have only provided one so far. The second input is from another model trained using same data. We will repeat the steps described

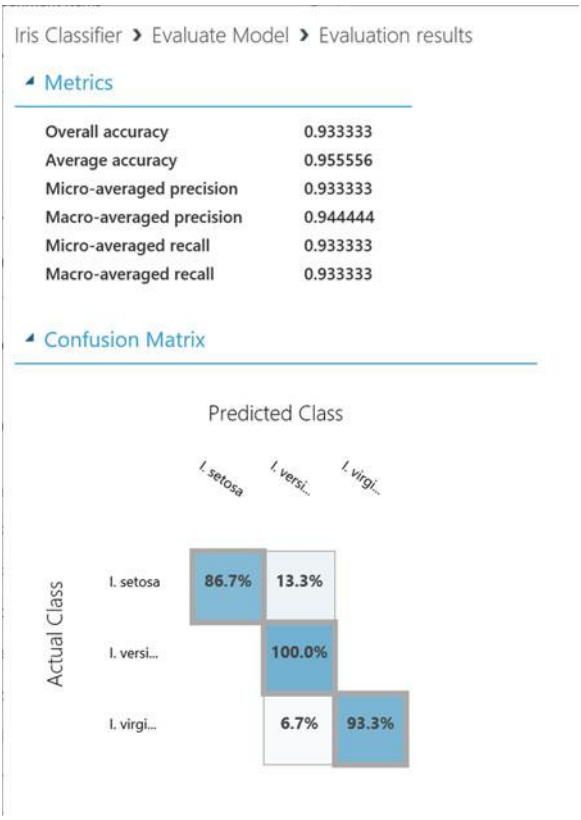


Fig. 22.13 Visualization of the evaluate block

before to add multiclass neural network as second model and compare it with decision jungle using evaluate block. The machine learning pipeline should now look like the one shown in Fig. 22.14. Once the new pipeline is executed, we can see the comparative evaluation results as shown in Fig. 22.15. As we can see from the comparison results, multiclass neural network seems to perform better than the decision jungle. As the data is strictly numerical, decision tree type model does not have advantage (decision tree models have algorithmic advantage when dealing with categorical data.) and properly tuned[3] neural networks generally perform better with pure numeric data.

[3] When we click on the *Initialize Model* block for multiclass neural network or multiclass decision jungle we can see a whole list of parameters on the right panel. Currently we have chosen the default values for all of them, but one can use the block *Tune Model Hyperparameters* from the left panel to further optimize the neural network model. This block replaced the simple *Train Model* block and performs hyper parameter tuning using grid search. However we will defer this discussion to next chapter.

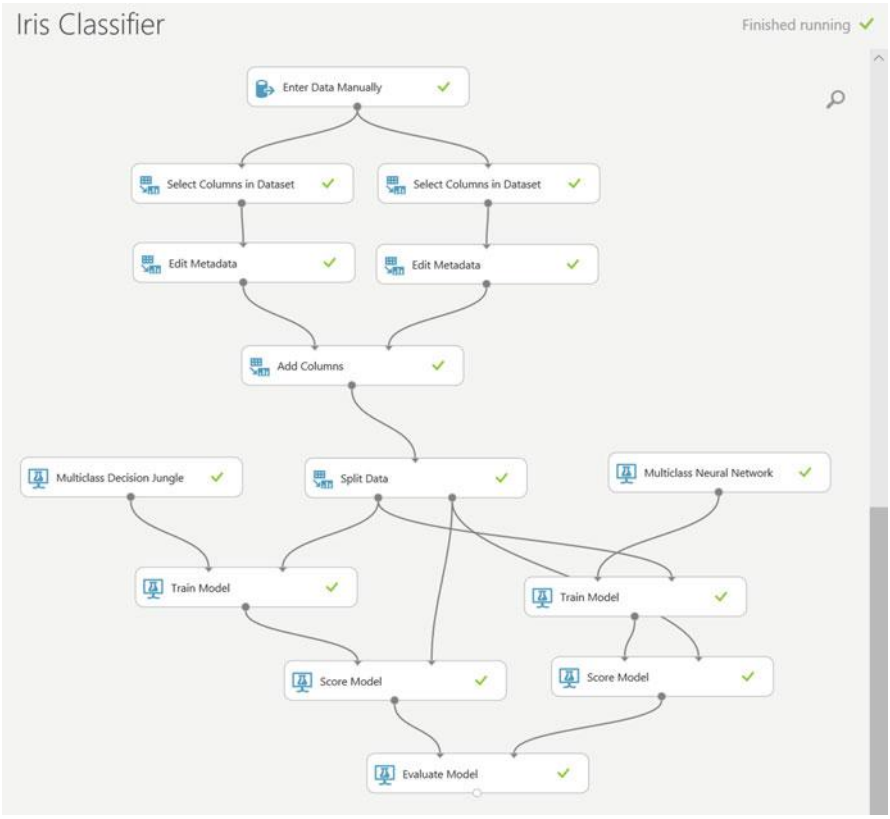


Fig. 22.14 Full ML pipeline with two models as decision jungle and neural network used as multiclass classifiers

22.5 Conclusion

In this chapter, we used Azure machine learning (AML) studio as first implementation tool. We used the Iris data that we had discussed before in the book, to illustrate the development of end to end pipeline of machine learning model. We started with manually entering the data in the AML studio experiment canvas and then adding all the necessary blocks for preprocessing, model training followed by scoring and evaluation. We also saw how we can iterate on the model for improving the performance and compare two different algorithms on same data to see which one gives better performance.

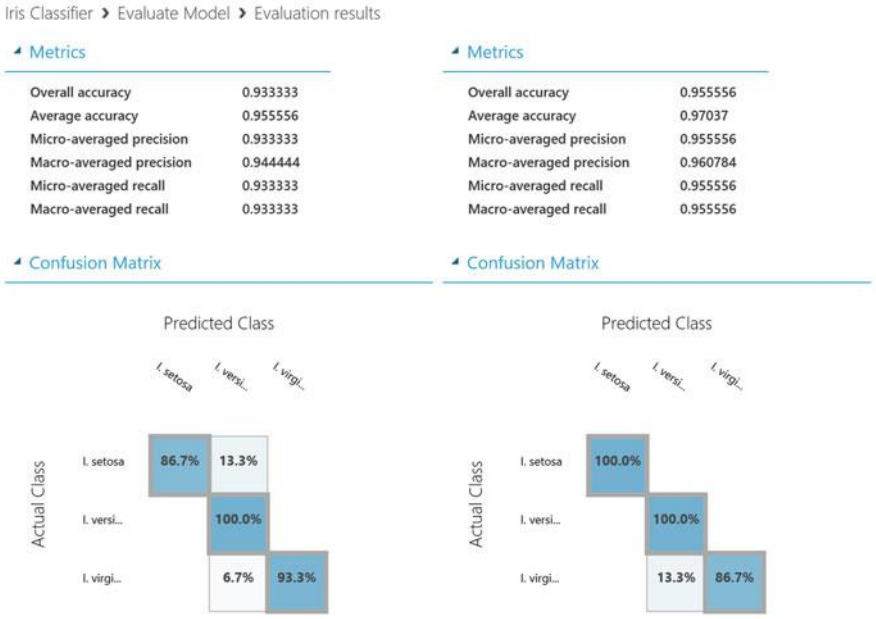


Fig. 22.15 Visualization of the evaluate block with comparative results from both the models

Chapter 23
Open Source Machine Learning Libraries

23.1 Introduction

Now that we have built the first machine learning pipeline from end to end, using AML studio, we cannot delve into more complex problems and their solutions using open source libraries. There have been nothing short of explosion of different machine learning libraries in last few years. Almost every university and big technology company has published their own version of libraries. As the libraries are not programming platform themselves they always need to tag along with an existing platform. The platform that is getting most attention is *Python*. Python is an interesting platform. It is foremost an interpreted language of programming.[1] Being interpreted makes it easy to start, as you can just write your first line of code and just run it. In spite of being interpreted language it has some really advanced features that have attracted lot of companies and universities to make this de-facto standard for machine learning library development.

[1]The programming languages are primarily divided into two types: (1) Compiled and (2) Interpreted. In interpreted languages, when a code is executed, each line is executed one after another. Hence if there are any bugs in the code, they do not surface until that line gets executed. Also, the code written in interpreted language is less optimal as it is not really optimized holistically. Compiled languages are different. Before even first line of code is executed, a part of the language, called compiler, reads all the lines and finds if they have any bugs. If there are, then compilation process itself fails and user is notified of the bug. Once the code is bug-free, the compiler translates the English language code written as per syntax of the language into binary form that is then executed. As the compiler looks at the entire code before running any individual like it can optimize the entire chunk of code for faster execution or smaller memory footprint. However, the added complexity of compilation can scare away beginners.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_23

23.2 Options of Machine Learning Libraries

Here is a list of some of the more popular machine learning libraries available in open source:

1. **Scikit-learn** Scikit-learn, also called as *sklearn* is one of the most popular machine learning library and it also has support for most types of models. This is built on Python ecosystem.
2. **TensorFlow** TensorFlow is a library published by Google as open source. It supports multiple languages including Python, C#, Java, etc. TensorFlow is more targeted towards deep learning applications and does not support quite a few of classical machine learning[2] techniques.
3. **Theano** Theano is another machine learning library that is targeted for deep learning applications. It is primarily based on Python ecosystem.
4. **CNTK** Microsoft Cognitive Toolkit, also called *CNTK*, is another deep learning targeted machine learning library published as open source from Microsoft. CNTK supports Python and C++.
5. **Keras** Keras is an interesting option. It provides an even higher level interface on top of existing libraries like Theano, TensorFlow, and CNTK. It provides unified commands to build deep learning pipeline using either of three platforms using Python.
6. **Caffe** Convolutional Architecture for Fast Feature Embedding or Caffe is another machine learning library targeted towards neural networks and deep learning written for Python.
7. **Torch** Yet another deep learning library that is primarily written for another language called *Lua*. Lua has deeper ties with C, and hence Torch is also compatible with C or C++.

Most of the deep learning libraries also support GPU[3] based computation. This does not mean that they don't work with systems without GPUs, but if you have a powerful GPU (specifically Nvidia GPU) then computation can be accelerated using CUDA support.

[2]The term *classical machine learning* is used to refer to the machine learning techniques that do not use deep learning. They would include SVM, decision trees, probabilistic methods, etc.

[3]Graphics Processing Unit or GPU is also sometimes called as graphics card in the computer. Traditionally the graphics cards are either embedded on the motherboard of the computer or can be added as discrete cards. The primary function of the GPU is to manage the displaying of the information on monitor. They are heavily used in video games. GPU has fundamentally different architecture compared to CPU, and are not suitable for typical computation performed by CPU. However, GPUs typically provide massive parallel processing support. As in typical graphical computations, similar computations need to be performed for each of the pixel that is rendered on the screen. Nvidia identified this advantage and created a library called CUDA that exposes some CPU like commands that are specifically tuned for deep learning type applications. Most deep learning libraries mentioned above support CUDA to accelerate the training of deep networks. Although in order to really see the performance gains one needs a really high end GPU.

Apart from these open source libraries there are some premium platforms available for developing machine learning models such as *Matlab*. They offer better integration of the libraries with the existing development environments, but they are not quite necessary to start off. There exists another interesting machine learning library called *MLLIB*. However, it is supported as the machine learning platform on distributed computer network called *Spark*. Quite often the size of data used to develop machine learning model goes well beyond the scope of single PC, (in other words when dealing with *Big Data*) then one has to resort to distributed computation. Spark is one of the popular open source distributed computation platforms, and MLLIB is exclusively supported on Spark.

As the scope of this book goes well beyond only deep learning, but we need to delve into big data for now, we will use sklearn as our library of choice. Once having familiarity with sklearn, jumping to other libraries for deep learning applications is also relatively easy.

23.3 Scikit-Learn Library

Scikit-learn or sklearn library is built on core Python and few additional libraries like *NumPy*, *SciPy*, and *Pandas*. The plotting library *Matplotlib* is also needed. I will not go into the details of installation of Python and these other libraries here. There are multiple options available to do that based on the operating system used. Here are few links that have sufficient information [19–25].

23.3.1 Development Environment

Python does not need any specific development environment, and one can easily use any text editor (preferably with some syntax highlighting) and write the Python code. Then the Python file can be executed using the command line interface. As Python is an interpreted language, one can also directly enter line by line code on the Python command line and see the results. However, for this chapter we will use an online interface called as *Jupyter Notebook* [26]. Once installed on the PC, you can run it either from command line or from the desktop icon and it opens up a browser page with interface as shown in Fig. 23.1 The interface shows a file browser from your local computer. You can create a suitable folder to keep all the Python files and create a new file to create an end to end machine learning pipeline using sklearn. The files created in jupyter notebook are named as **.ipynb*, that stands for interactive Python notebook. When you click on the notebook file, it will open a notebook for editing in a new browser tab. The interface is in the form of sequence of cells. You can execute each cell at a time and see the output of that in the same window, just below the cell.

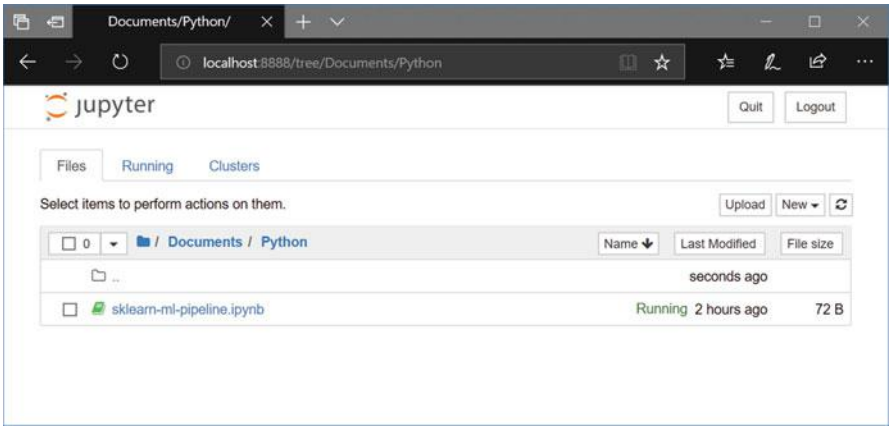


Fig. 23.1 Starting interface of Jupyter online interactive editor for Python

```
In [4]: iris_data = pd.read_csv('Iris.csv')
        iris_data
```

Out[4]:

	Sepal-Length	Sepal-Width	Petal-Length	Petal-Width	Label
0	5.1	3.5	1.4	0.2	I. setosa
1	4.9	3.0	1.4	0.2	I. setosa
2	4.7	3.2	1.3	0.2	I. setosa
3	4.6	3.1	1.5	0.2	I. setosa
4	5.0	3.6	1.4	0.3	I. setosa
5	5.4	3.9	1.7	0.4	I. setosa
6	4.6	3.4	1.4	0.3	I. setosa
7	5.0	3.4	1.5	0.2	I. setosa
8	4.4	2.9	1.4	0.2	I. setosa
9	4.9	3.1	1.5	0.1	I. setosa
10	5.4	3.7	1.5	0.2	I. setosa
11	4.8	3.4	1.6	0.2	I. setosa
12	4.8	3.0	1.4	0.1	I. setosa
13	4.3	3.0	1.1	0.1	I. setosa
14	5.8	4.0	1.2	0.2	I. setosa
15	5.7	4.4	1.5	0.4	I. setosa
16	5.4	3.9	1.3	0.4	I. setosa
17	5.1	3.5	1.4	0.3	I. setosa

Fig. 23.2 Python code showing how to read CSV data followed by displaying the data that is read

23.3.2 Importing Data

In order to compare and contrast with the example studied in the previous chapter using Azure machine learning studio, we will use the same Iris data to solve the same problem of multiclass classification. To read the data into Python, we will use a *CSV* (comma separated variable) file that contains all the Iris data along with the headings. We will use a CSV reader from *Pandas* library and display the table to make sure we have read all the data correctly as shown in Fig. 23.2. This

import converts the CSV file into an internal tabular form called *Pandas DataFrame*. This dataframe structure is quite important from the machine learning development perspective and is extensively used. The dataframe structure provides multiple table operations like joining two tables, doing column based operations, etc. that simple 2D array does not.

23.3.3 Data Preprocessing

As described in the previous chapter, this data does not contain any missing values and all the data is numeric. Hence there is no need to perform any data clean up and transformation operations. However, we still need to identify the features and labels separately from the data similar to what we did in AML studio. Figure 23.3 shows[4] these steps with scikit-learn.

```
In [18]: features = iris[['Sepal-Length','Sepal-Width','Petal-Length','Petal-Width']]
         labels = iris[['Label']]

In [59]: features.head()

Out[59]:
```

	Sepal-Length	Sepal-Width	Petal-Length	Petal-Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.3

```
In [58]: labels.head()

Out[58]:
```

	Label
0	I. setosa
1	I. setosa
2	I. setosa
3	I. setosa
4	I. setosa

Fig. 23.3 Python code showing data preprocessing

[4]The function *head()* shows top 5 rows of the given dataframe.

23.3.4 Splitting the Data Using Stratified Sampling

Before invoking the training algorithm we need to split the data into training and test. As discussed in previous chapter, we need to use stratified sampling to make sure we have similar class distribution in training and test set. Figure 23.4 shows this operation.

The inputs to the data split function are self-explanatory and it generates four different variables as output as listed below.

1. **xtrain** denotes the features of the samples to be used for training.
2. **ytrain** denotes the labels for the samples to be used for training.
3. **xtest** denotes the features of the samples to be used for testing.
4. **ytest** denotes the labels of the samples to be used for testing.

23.3.5 Training a Multiclass Classification Model

To illustrate a parallel with previous chapter, we will start with a multiclass classifier based on ensemble decision tree. The specific decision jungle classifier that we used in the previous chapter is not available in *Scikit-learn*, hence we will use random forest as a similar alternative. We will use the same values for hyperparameters as we used for decision jungle to have apples to apples comparison. The code for these steps is shown in Fig. 23.4.

```
In [78]: from sklearn.model_selection import train_test_split
         xtrain, xtest, ytrain, ytest = train_test_split(features, labels, test_size=0.3, stratify=labels)

In [79]: from sklearn.ensemble import RandomForestClassifier
         randforest = RandomForestClassifier(n_estimators=8, max_depth=32, max_leaf_nodes=128)

In [80]: randforest

Out[80]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=32, max_features='auto', max_leaf_nodes=128,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=8, n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

In [81]: randforest.fit(xtrain, ytrain['Label'])
```

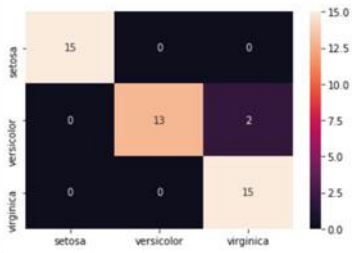
Fig. 23.4 Python code showing the stratified sampling based train-test split followed by use of random forest classifier training

```
In [82]: from sklearn.metrics import accuracy_score
         ypred = randforest.predict(xtest)
         accuracy = accuracy_score(ytest, ypred)
         accuracy

Out[82]: 0.9333333333333333

In [67]: import seaborn as sb
         types = ['setosa', 'versicolor', 'virginica']
         confmat = pd.DataFrame(confusion_matrix(ytest, ypred), columns=types, index=types)
         sb.heatmap(confmat, annot=True)

Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x17235c64b38>
```




```
In [83]: confmat
```

Fig. 23.5 Performance of the classification using metrics of accuracy and confusion matrix

23.3.6 Computing Metrics

Once the models are successfully trained, we can apply it on the test data and generate predictions. The predictions can then be compared with actual labels, or ground truth to generate the metrics in the form of accuracy and confusion matrix. Figure 23.5 shows the code for these operations. As can be seen, the accuracy matches exactly, but confusion matrices does not match exactly compared to what we got using AML studio. There are multiple reasons for this. We used a similar albeit a different algorithm for classification and the random stratified split was also different in both cases. In spite of these variations the differences are quite minimal. The confusion matrix was shown as percent values in case of AML studio, but the default way using scikit-learn shows actual number of misclassifications rather than percentages, but those two numbers can be easily converted.

23.3.7 Using Alternate Model

Now, we will try to use neural network based multiclass classifier and compare its output with the output of random forest, precisely how we did before. Figure 23.6 shows the full code and results of this. The accuracy score with neural network is identical compared to random forest. However, the confusion matrix shows slightly different errors in misclassification. This score is less than what was obtained with neural network based classifier in AML studio. However, the differences are within

```
In [102]: from sklearn.neural_network import MLPClassifier
          neuralnet = MLPClassifier(hidden_layer_sizes=(4,100), tol=0.005)

In [103]: neuralnet.fit(xtrain, ytrain['Label'])

Out[103]: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                        beta_2=0.999, early_stopping=False, epsilon=1e-08,
                        hidden_layer_sizes=(4, 100), learning_rate='constant',
                        learning_rate_init=0.001, max_iter=200, momentum=0.9,
                        n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                        random_state=None, shuffle=True, solver='adam', tol=0.005,
                        validation_fraction=0.1, verbose=False, warm_start=False)

In [104]: ypred = randforest.predict(xtest)
          accuracy = accuracy_score(ytest, ypred)
          accuracy

Out[104]: 0.9333333333333333

In [105]: confmat = pd.DataFrame(confusion_matrix(ytest, ypred), columns=types, index=types)
          sb.heatmap(confmat, annot=True)

Out[105]: <matplotlib.axes._subplots.AxesSubplot at 0x172360d5a58>
```

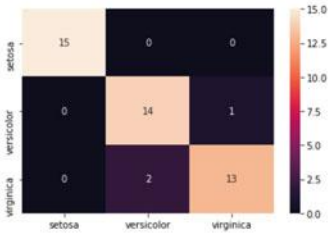


Fig. 23.6 Use of neural network based multiclass classifier as alternate model and comparison of metrics

2% and are quite reasonable considering the small size of data and variation in sampling. Also, there are multitude of hyperparameters that we need to initialize for neural network, and AML studio exposes slightly different set of parameters compared to scikit-learn, and that can also account for variation in the performance. It is important to note that there are multiple variations in which the same algorithm is coded in AML studio and scikit-learn and these variations lead to variation in hyperparameters. Hence, it is important to understand the details of parameters in framework that is chosen for implementing the machine learning pipeline.

23.4 Model Tuning and Optimization

We purposely skipped the aspect of model tuning and optimization in the previous chapter for keeping the pipeline simple. When the first pass of the machine learning pipeline is complete it establishes a baseline for performance metrics. The numbers we cannot regress from. However, there is significant scope to improve the metrics further. This improvement process is called as model tuning and optimization. This can be achieved using a multiple options such as:

1. **Grid search** In grid search all the possible combination of each hyperparameter are listed. Then the model training is performed for each unique combination

of all the hyperparameters. Let there be 5 hyperparameters and possible values for each of the parameter are (2,3,2,5,4). Then we will need to run the training for $2 \times 3 \times 2 \times 5 \times 4$ or 240 times. Thus the computation complexity increases exponentially with added hyperparameter. Grid search is an exhaustive search option and provides best possible combination, but if the number of hyperparameters is large and each has large number of possible values, then it is not possible to use. However, as each combination has no influence on another one, all the training operations can be carried out in perfectly parallel manner utilizing GPU hardware if available.

2. **Gradient Search** This is an improvement over the grid search. Applying gradient search for hyperparameter tuning is using an algorithm on top of another algorithm that is used for training the underlying model. However, when the grid of hyperparameters is large, gradient search converges much quicker than fully exhaustive grid search. Hence even being conceptually more complex, it reduces the computational complexity.
3. **Evolutionary Methods** The evolutionary methods like genetic programming or simulated annealing can also be effectively used to find the optimal set of hyperparameters.
4. **Probabilistic Methods** Bayesian methods can also be used if we can impose a known probability distribution on the effect of hyperparameters on the training accuracy.

23.4.1 Generalization

When a hyperparameter optimization search is performed on the given set of training data, it is quite likely to overuse and effectively overfit the model for that set of data. In order to improve generalization and reduce overfitting a technique of cross-validation is used. In cross-validation, the whole of labelled data is split into 3 parts, called *train*, *validate*, and *test*. Also, the data between train set and validation set is resampled multiple times to create multiple sets of train and validation sets. The above mentioned training iterations are performed on each of the train and validation sets. The test set is not used during this, and is kept as blind set for the trained model only for the purpose of metrics calculation. This process in spite of adding more computational complexity, improves the generalization performance of the model.

23.5 Comparison Between AML Studio and Scikit-Learn

Figures 23.7 and 23.8 show the lists of algorithms supported by each. The list of AML blocks is comprehensive, while list in scikit-learn is higher level list such that each item has multiple variations that are supported.

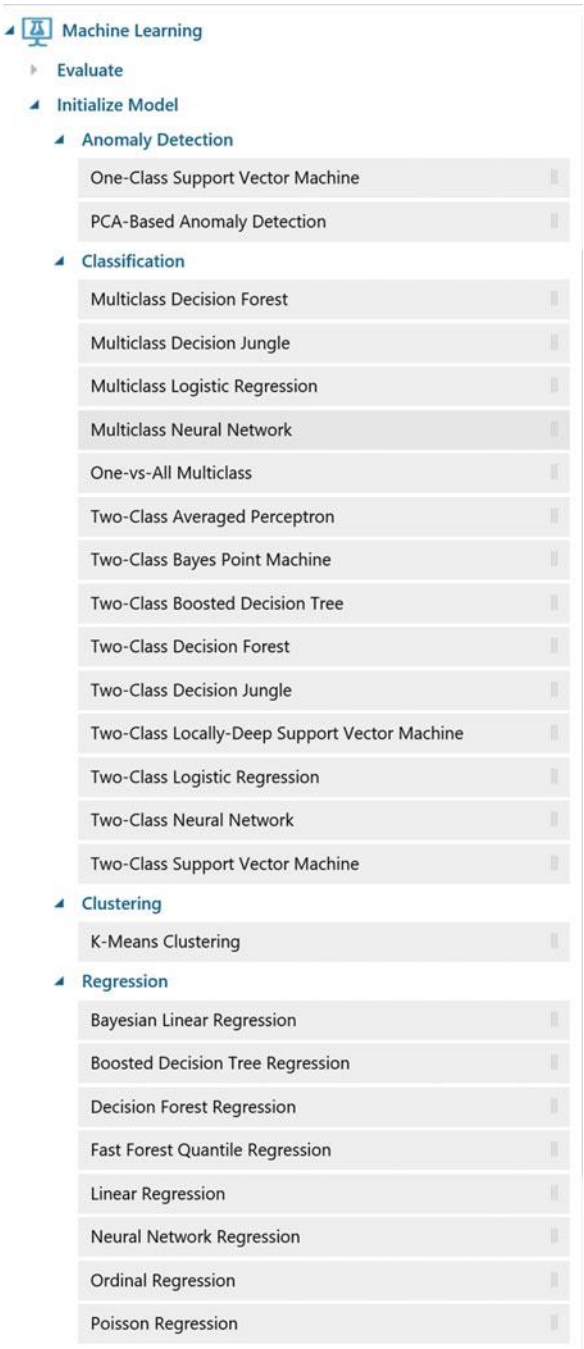


Fig. 23.7 List of all the supported algorithm blocks in Azure machine learning studio

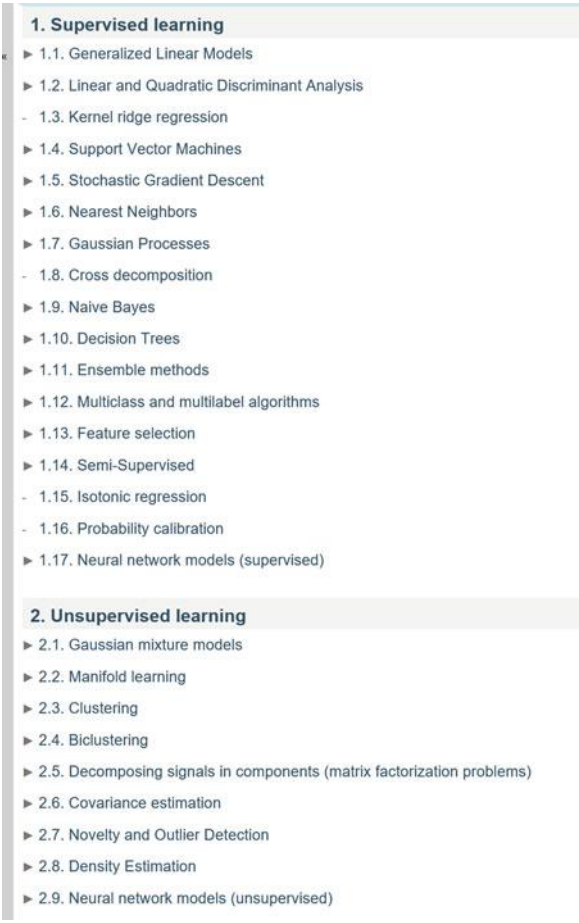


Fig. 23.8 List of all the algorithms in scikit-learn

As mentioned earlier, AML studio is easier and more visual option to get started with the topic and once having sufficient proficiency with all the techniques moving on the scikit-learn is recommended. With scikit-learn one has more flexibility with importing data, preprocessing and also with model selection and tuning. With additional help from other Python libraries in plotting and general numeric and statistical analyses one can easily extend the scope of the pipeline for any arbitrary type of processing. Going even further, using core Python syntax, one can come up with their own implementations of the algorithms or their variations and augment the existing libraries with them.

23.6 Conclusion

In this chapter we implemented an end to end machine learning pipeline using scikit-learn library. We used the same set of data and similar machine learning algorithms in both cases to draw parallel between these two options. It was seen that the performance metrics obtained between the systems and two algorithms were quite close. We also learned that due to high complexity of coding these algorithms, different libraries expose slightly different sets of hyperparameters. It is therefore important to understand the details of all the input parameters for framework chosen for implementation of the machine learning pipeline to be able to optimize it. We then learned the use of hyperparameter tuning and generalization of the algorithm with cross-validation. We also compared the AML studio and scikit-learn libraries based on their scope and functionality.

Chapter 24
Amazon's Machine Learning Toolkit: Sagemaker

24.1 Introduction

Sagemaker is Amazon's version of free to use framework for implementing and productizing machine learning models. In contrast to AzureML studio as we studied in Chap. 22, Sagemaker provides a Jupyter notebook interface for implementing the models as we saw in Chap. 23. It offers the Amazon's own implementation of a library of models along with open source models, e.g., scikit-learn as we used in Chap. 23. In this chapter we will look at how to setup Sagemaker for first use and how to implement the model using Sagemaker.

24.2 Setting Up Sagemaker

Figure 24.1 shows the screen where you can sign up to start using Amazon Sagemaker. Once you create your free account, you can start with building your first machine learning pipeline. Sagemaker interface is a little bit more advanced with stronger focus on productizing the machine learning pipeline. Once you have created the account, you are taken to Sagemaker home as shown in Fig. 24.2. From there, you can navigate to Sagemaker dashboard, see Fig. 24.3. The dashboard shows the various options you can do inside the Sagemaker. It also shows the previous activity. Currently it is empty, as we are just getting started. However, in the future, it will show the status of the model execution here. We would like to build the machine learning model using the notebook, so we will select the option of *Notebook*.

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_24

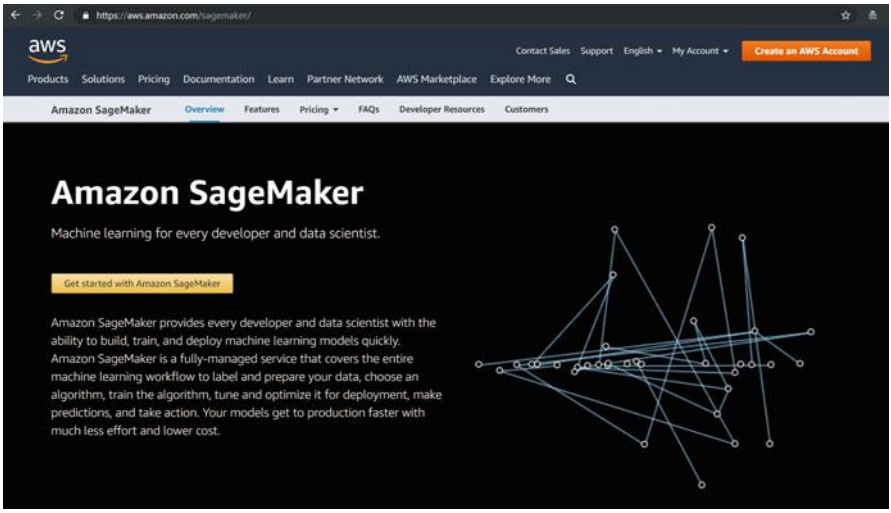


Fig. 24.1 Screenshot showing the getting started screen for Amazon Sagemaker

Then a customization screen is shown as can be seen in Fig. 24.4 to create the notebook instance. We will name the notebook as *ClassifyIris*. It should be noted that *underscore character* is not allowed in this name. Most of the other options can be kept at default. It should be noted that 5 GB is minimum required volume, even if we are going to need only a small fraction of it. The IAM role needs to be created. This role enforces access restrictions. These would be useful in productizing the model later on. For now, we will just create a simple role as shown in Fig. 24.5. We will select the role that can access any bucket in the linked S3 storage. Then we can go ahead with notebook creation. Once clicked on *Create Notebook Instance*, you are shown with the confirmation as can be seen in Fig. 24.6. The status is pending, but within few minutes, it should be complete.

24.3 Uploading Data to S3 Storage

The next step is to onboard the data. For that we need to step out of Sagemaker interface and go to list of services offered by AWS, and select S3 storage as shown in Fig. 24.7. You first need to add a bucket as the primary container of the data. This step is shown in Fig. 24.8. We will name the bucket as *mycustomdata*. We will accept the default options in the subsequent screens to complete the creation of the bucket. Then we will create a folder inside the bucket with name *iris*. We will upload the same *Iris.csv* file that we used in previous chapters in the folder (Fig. 24.9).

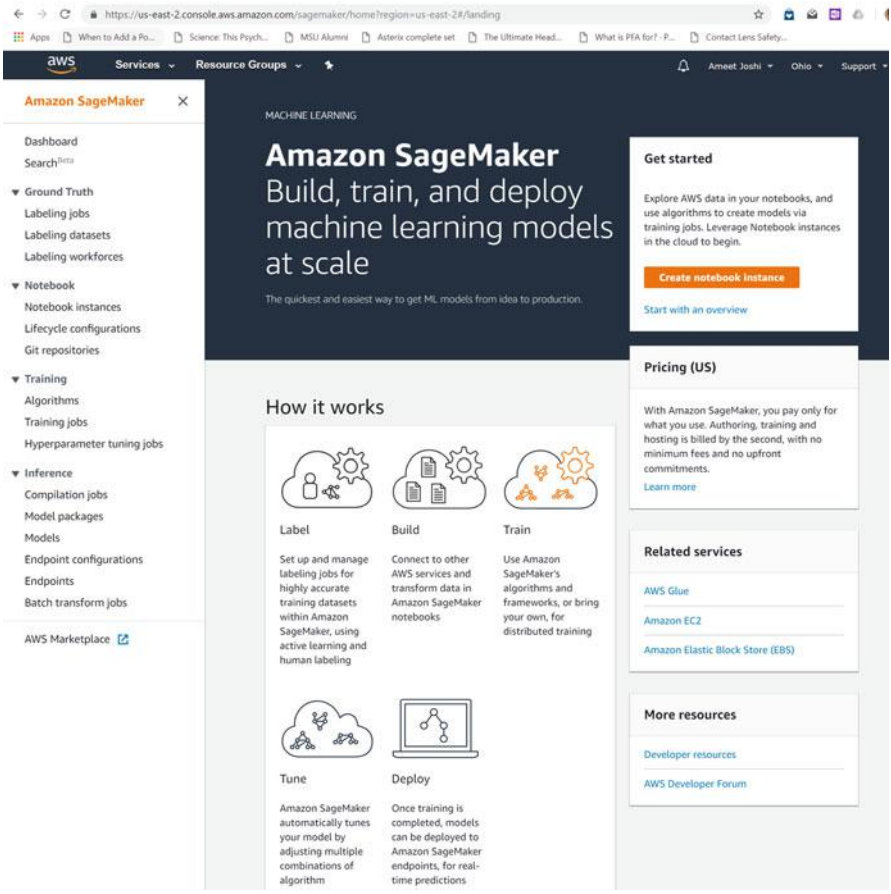


Fig. 24.2 Screenshot showing the home screen for Amazon Sagemaker

24.4 Writing the Machine Learning Pipeline Using Python

Now, we can go back to the notebook instance that we have created in Sagemaker. This instance acts like a project environment where we can add multiple scripts. For our purpose, we will need a single script or notebook. If you click on the *new* option on the top right of the notebook instance interface, you are presented with multiple options as shown in Fig. 24.10. Some of the options are for building the models in *Spark*, which is a big data environment. Some are specific to deep learning libraries like Tensorflow and PyTorch. Each one is available with different versions of python. For current experiment, we will choose *conda_poython3*. This option will enable standard Python (version 3) script. The first step would be to import the data that we uploaded in S3. Figure 24.11 shows the code to import the data.

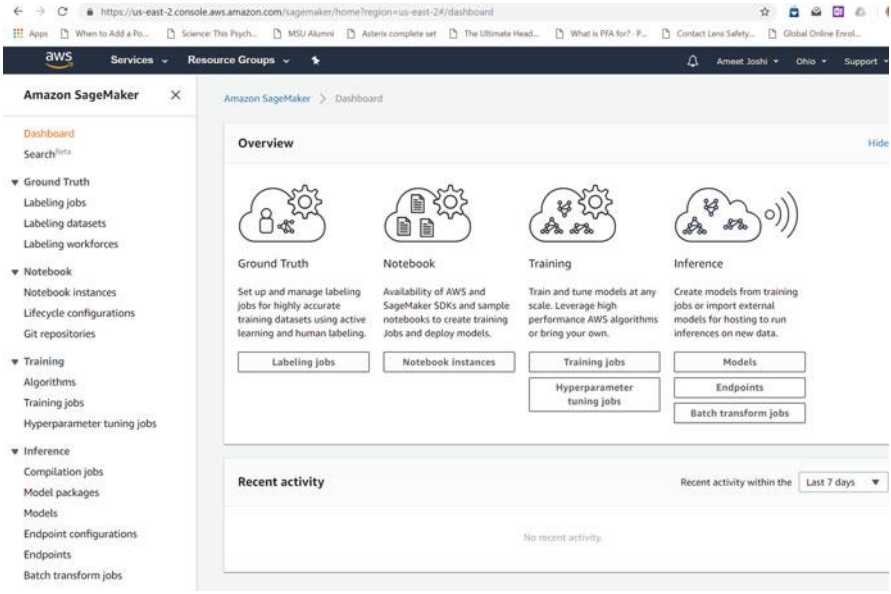


Fig. 24.3 Screenshot showing the dashboard for Amazon Sagemaker

From here onwards, we essentially have the same environment as we discussed in the previous chapter. We can use the scikit-learn libraries to carry out the multiclass classification, or we can use Amazon's proprietary library function *XGBoost*. Figure 24.12 shows the full list of custom models supported by Amazon Sagemaker. These models use the same underlying theory described in previous chapters, but they are optimized for performance and execution for AWS and Sagemaker system. It is left as exercise to the reader to build the remaining pipeline using different models and see the difference in the performance.

24.5 Conclusion

In this chapter we studied how to build a machine learning pipeline using Amazon's interface called as Sagemaker. The initial setup initializes the productization aspects of the machine learning pipeline. Then the user is presented with the familiar Jupyter notebook interface to build the pipeline using either open source options of Amazon's proprietary options. Once the model is built the Sagemaker system provides various options to productize it, monitor the state and performance of the pipeline at various stages and debug it.

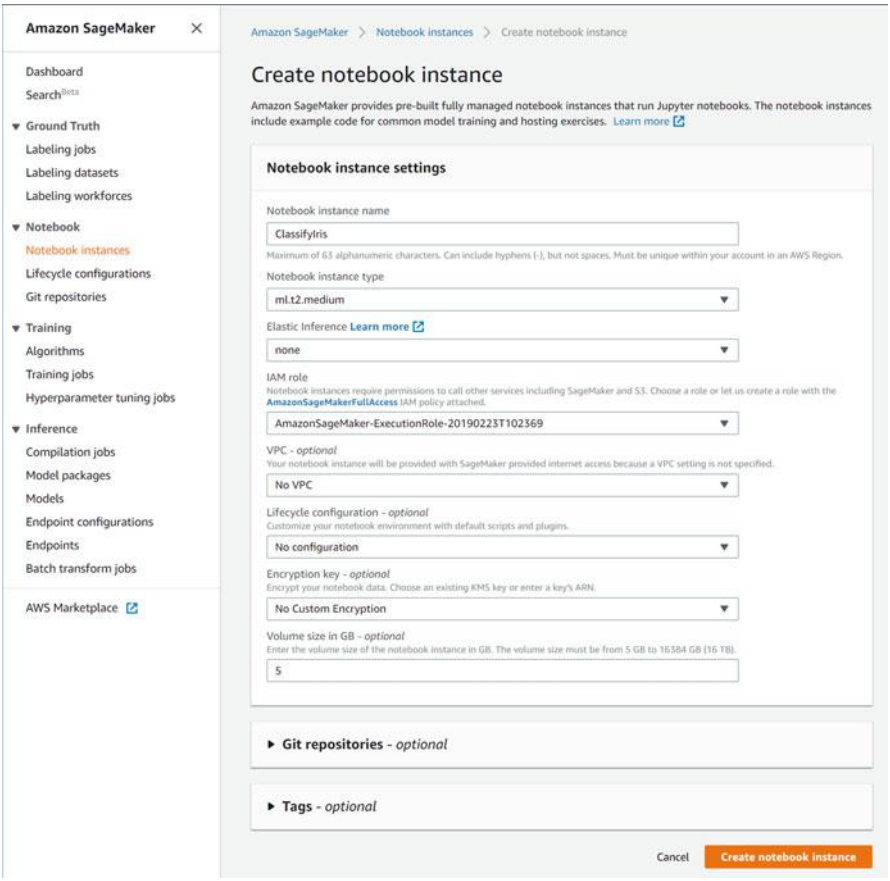


Fig. 24.4 Screenshot showing create notebook interface for Amazon Sagemaker

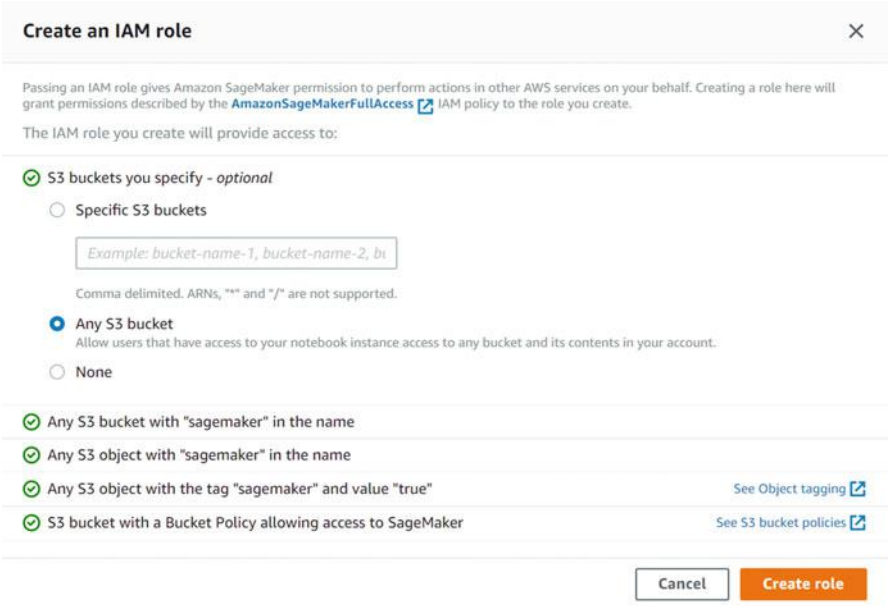


Fig. 24.5 Screenshot showing create role setup for Amazon Sagemaker

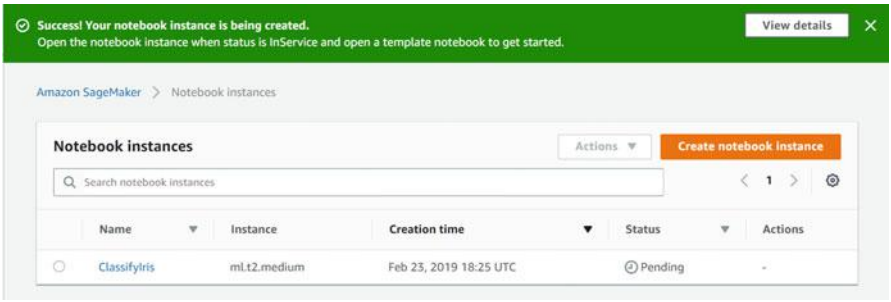


Fig. 24.6 Screenshot showing that the notebook is successfully created in Amazon Sagemaker

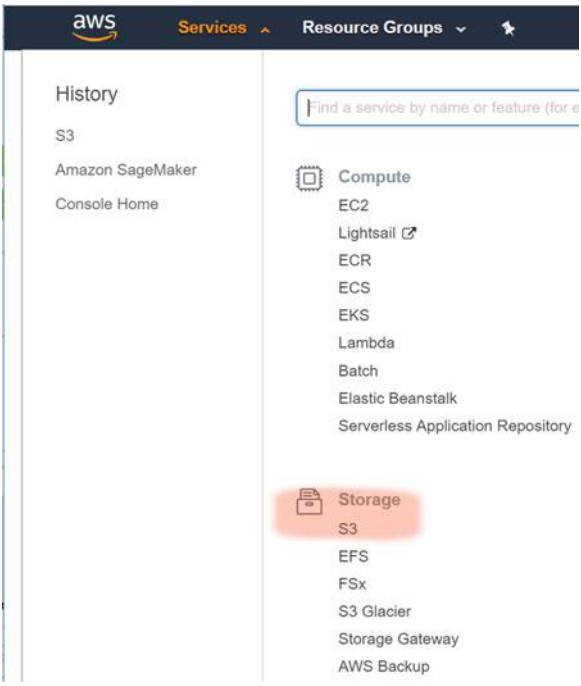


Fig. 24.7 Screenshot showing the option for selecting the S3 storage in AWS

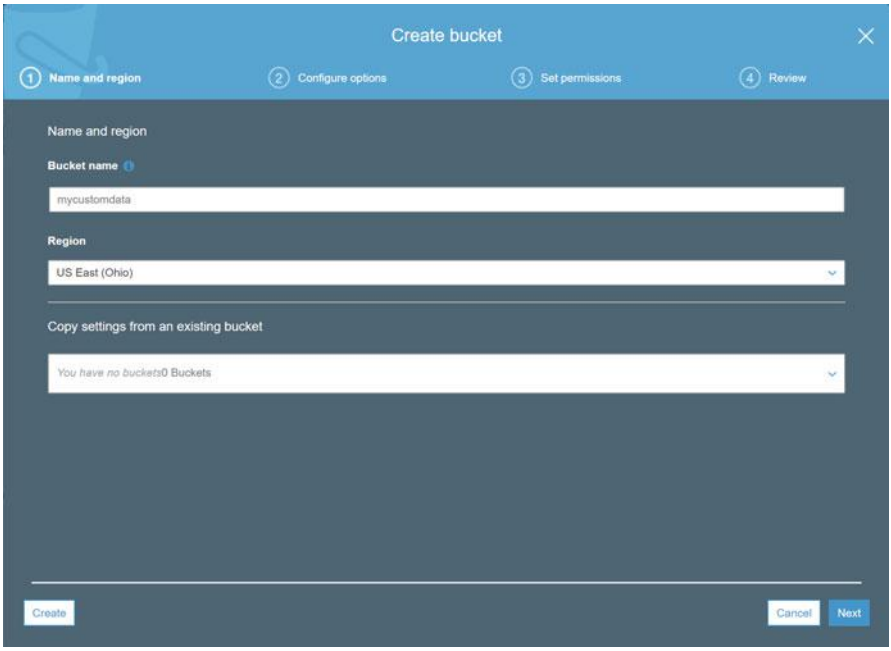


Fig. 24.8 Screenshot showing the option for creating custom dataset in S3 storage in AWS

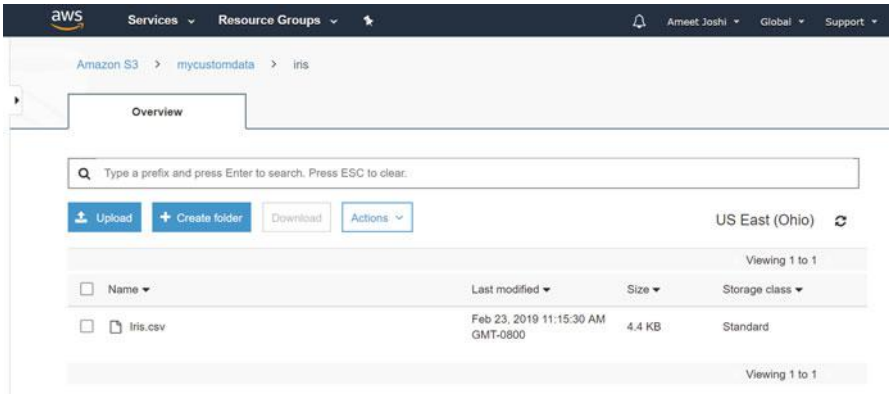


Fig. 24.9 Screenshot showing that Iris data is successfully uploaded in the S3 storage in AWS

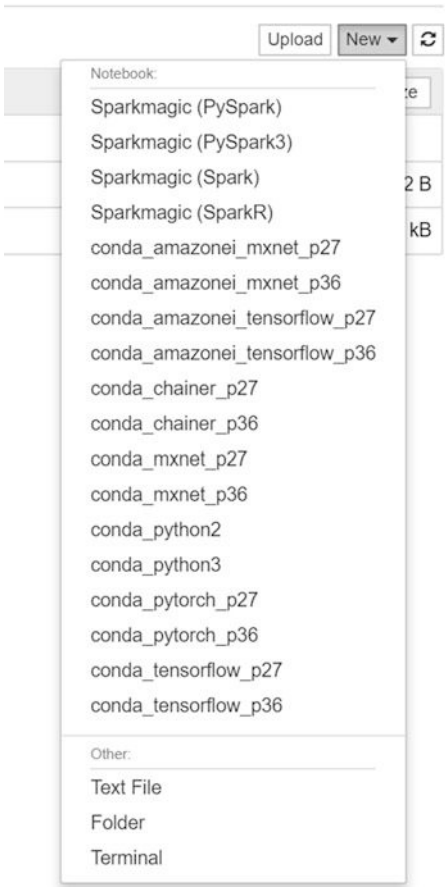
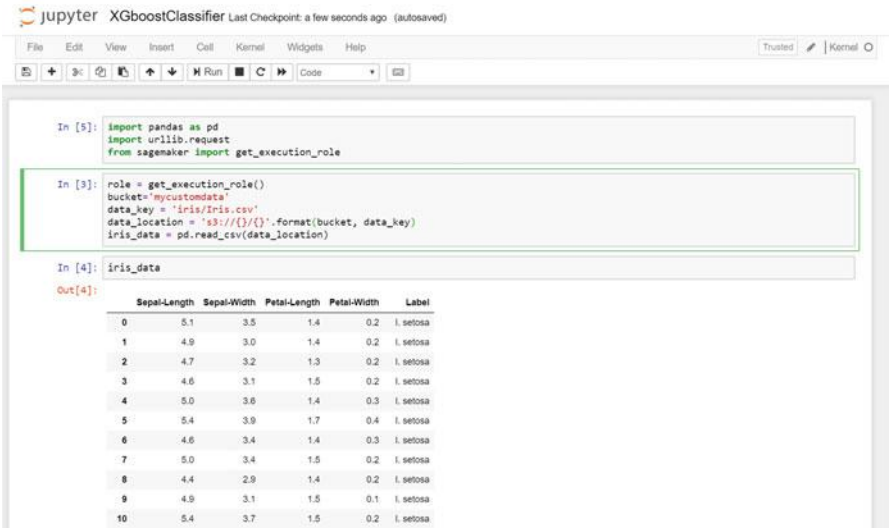


Fig. 24.10 Screenshot showing all the possible options for creating notebook in Amazon Sagemaker



	Sepal-Length	Sepal-Width	Petal-Length	Petal-Width	Label
0	5.1	3.5	1.4	0.2	I. setosa
1	4.9	3.0	1.4	0.2	I. setosa
2	4.7	3.2	1.3	0.2	I. setosa
3	4.6	3.1	1.5	0.2	I. setosa
4	5.0	3.6	1.4	0.3	I. setosa
5	5.4	3.9	1.7	0.4	I. setosa
6	4.6	3.4	1.4	0.3	I. setosa
7	5.0	3.4	1.5	0.2	I. setosa
8	4.4	2.9	1.4	0.2	I. setosa
9	4.9	3.1	1.5	0.1	I. setosa
10	5.4	3.7	1.5	0.2	I. setosa

Fig. 24.11 Screenshot showing the python code for importing the data set from S3 storage and using in the machine learning pipeline in Amazon Sagemaker notebook

Fig. 24.12 Screenshot showing Amazon's proprietary machine learning models available in Amazon Sagemaker

Part VI
Conclusion

> Do or do not, there is no try!!
>
> —Yoda, “Star Wars: The Empire Strikes Back”

Part Synopsis
This part concludes the book with key takeaways from the book and road ahead.

Chapter 25
Conclusion and Next Steps

25.1 Overview

The concepts of machine learning and artificial intelligence have become quite mainstream and focus of lot of cutting edge research in science and technology. Although most of the concepts in machine learning are derived from the mathematical and statistical foundations, which are quite hard to understand without having background in graduate level courses in those topics, the artificially intelligent applications based on machine learning are becoming quite commonplace and accessible to anyone. This simplicity of applications has opened the doors to these topics for a huge number of engineers and analysts, which otherwise were only accessible to people doing their Ph.Ds. This is good and bad in a way. *Good* because these topics do have applications in technology that is used in every little aspect of our modern lives. The people who are building all these technology are not necessarily the people who are pursuing their Ph.Ds in mathematics and statistics. Hence, when more people are armed with these powerful techniques, the faster is the pace of evolution of technology and revolution of lives of all the people. *Bad* because, oversimplification of the concepts can lead to misunderstanding and that can have catastrophically worse if not dangerous effects.

Many books, Wikipedia, and many other websites are dedicated on this topic, but in my investigation, the material presented in most sources is focused on a small subset of topics and typically either goes too deep in theory or stays too superficial that the applications may seem like black magic. I had come across many colleagues and friends who had asked questions about different problems or applications in day-to-day life that were all ultimately connected with machine learning or artificial intelligence and created profound confusion. Most of these sources are useful when you know exactly what you are looking for, and in most cases, that is not true. In such case it is easy to get lost in plethora of this information.

I had tried to scope the book to include most of the concepts that one will come across in the fields of machine learning or artificial intelligence, specifically in day-

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6_25

to-day life, but still present the material in more intuitive and conceptual focused manner rather than going deep into theory. However, to make the concepts concrete and avoid oversimplification, theory is presented whenever it is necessary and easy to understand with undergraduate level mathematics and statistics background. Anyone interested in this field, working on developing applications, or even developing machine learning systems will get something new from this book. It will connect the dots floating around from different contexts in a meaningful way and connect them with their roots in theory. In no way this book claims to give comprehensive context of the field, but it will certainly equip the reader to tackle any new concept with confidence.

25.2 What's Next

After completing this book, I would say, should mark the beginning for the reader to embark on whatever is next in the broader field of data science in general. There are many ways in which one can proceed depending on the interest or need. Here is a couple of mainstream areas:

1. Target towards big data applications. One needs to learn more about the hardware architecture for managing the big data, and the technologies that manage it. For example Hadoop, Spark, etc.
2. Target towards any specific application, e.g., speech or image processing or computer vision or robotics, etc. In such cases, one would have to learn more about the domain knowledge about the area. Understand the nuances of the applications, understand what user of the application is going to look for. Then connect the machine learning context with it to build the artificial intelligence.

Each individual aspect of this technology when looked in micro detail, starts to seem obvious, trivial even. However, when all these individual components come together, the problem that is solved, and experience that is delivered is just magical. And that is what keeps me excited about this topic every single day. Hope to have passed this excitement through this book to every reader!

References

1. The Human Memory http://www.human-memory.net/brain_neurons.html
2. Wikipedia - Deep Learning https://en.wikipedia.org/wiki/Deep_learning
3. Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/iris. Irvine, CA: University of California, School of Information and Computer Science.
4. Wikipedia - Dynamic Programming Applications https://en.wikipedia.org/wiki/Dynamic_programming#Algorithms_that_use_dynamic_programming
5. Wikipedia - Linear discriminant analysis https://en.wikipedia.org/wiki/Linear_discriminant_analysis
6. UCI - Machine Learning Repository (Center for Machine Learning and Intelligent Systems) - Adult Data Set https://archive.ics.uci.edu/ml/datasets/Adult
7. Mobile cellular subscriptions (per 100 people) https://data.worldbank.org/indicator/IT.CEL.SETS.P2?end=2017&start=1991
8. Convolution Theorem https://en.wikipedia.org/wiki/Convolution_theorem
9. Diminishing Returns https://en.wikipedia.org/wiki/Diminishing_returns
10. Shannon number https://en.wikipedia.org/wiki/Shannon_number
11. Deep Blue (chess computer) https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)
12. Setting up Mario Bros. in OpenAI's gym https://becominghuman.ai/getting-mario-back-into-the-gym-setting-up-super-mario-bros-in-openais-gym-8e39a96c1e41
13. Open AI Gym http://gym.openai.com/
14. Quantum Computing https://en.wikipedia.org/wiki/Quantum_computing
15. Uncertainty Principle https://en.wikipedia.org/wiki/Uncertainty_principle
16. Quantum Entanglement https://en.wikipedia.org/wiki/Quantum_entanglement
17. Quantum Superposition https://en.wikipedia.org/wiki/Quantum_superposition
18. Summit Supercomputer https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
19. Python https://www.python.org/
20. Anaconda https://www.anaconda.com/distribution/
21. Pip https://pypi.org/project/pip/
22. Numpy http://www.numpy.org/
23. Scipy https://scipy.org/install.html
24. Matplotlib https://matplotlib.org/users/installing.html
25. Scikit Learn https://scikit-learn.org/stable/index.html
26. Jupyter Notebook https://jupyter.org/
27. Travelling Salesman Problem https://en.wikipedia.org/wiki/Travelling_salesman_problem
28. Mahalanobis distance https://en.wikipedia.org/wiki/Mahalanobis_distance

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6

29. Causality in machine learning http://www.unofficialgoogledatascience.com/2017/01/causality-in-machine-learning.html
30. A Brief History of Machine Learning Models Explainability https://medium.com/@Zelros/a-brief-history-of-machine-learning-models-explainability-f1c3301be9dc
31. Automated Machine Learning https://en.wikipedia.org/wiki/Automated_machine_learning
32. AutomML.org https://www.ml4aad.org/automl/
33. PageRank https://en.wikipedia.org/wiki/PageRank
34. Self Organizing Maps https://en.wikipedia.org/wiki/Self-organizing_map
35. Netflix Prize https://www.netflixprize.com/rules.html
36. A Gentle Introduction to Transfer Learning for Deep Learning https://machinelearningmastery.com/transfer-learning-for-deep-learning/
37. Vladimir N. Vapnik, *The Nature of Statistical Learning Theory*, 2nd edn. (Springer, New York, 1995).
38. Richard Bellman, *Dynamic Programming*, (Dover Publications, Inc., New York, 2003).
39. Trevor Hastie, Robert Tibshirani, Jerome Friedman *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd edn. (Springer, New York, 2016).
40. Joseph Giarratano, Gary Riley, *Expert Systems: Principles and Programming*, PWS Publishing Company, 1994.
41. Olivier Cappé, Eric Moulines, Tobias Rydén, *Inference in Hidden Markov Models* (Springer, New York, 2005).
42. Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern Classification* John Wiley and Sons, 2006.
43. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio *Generative Adversarial Nets*, NIPS, 2014.
44. Frank Rosenblatt, *The Perceptron - a perceiving and recognizing automation*, Report 85–460, Cornell Aeronautical Laboratory, 1957.
45. Zafer CÖMERT and Adnan Fatih KOCAMAZ, *A study of artificial neural network training algorithms forclassification of cardiotocography signals*, Journal of Science and Technology, Y 7(2)(2017) 93–103.
46. Babak Hassibi, David Stork, Gregory Wolff, Takahiro Watanabe *Optimal brain surgeon: extensions and performance comparisons*, NIPS, 1993.
47. Yann LeCun, John Denker, Sara Solla, *Optimal Brain Damage*, NIPS 1989.
48. Tin Kam Ho, *Random Decision Forests*, Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.
49. Leo Breiman, *Random Forests*, Machine learning 45.1 (2001): 5–32.
50. Leo Breimian, *Prediction Games and ARCing Algorithms*, Technical Report 504, Statistics Department, University of California, Berkerley, CA, 1998.
51. Yoav Freund, Robert Schapire *A Short Introduction to Boosting*, Journal of Japanese Society for Artificial Intelligence, 14(5):771–780, September, 1999.
52. V. N. Vapnik and A. Y. Lerner *Pattern Recognition using Generazlied Portraits* Automation and Remote Control, 24, 1963.
53. Haohan Wang, Bhiksha Raj *On the Origin of Deep Learning*, ArXiv e-prints, 2017.
54. Geoffrey Hinton, Simon Osidero, Yee-Whye Teh A fast learning algorithm for deep belief networks, Neural Computation, 2006.
55. Kunihiko Fukushima, *Neocognition: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position* Biological cybernetics, 36(4), 193–202, 1980.
56. Michael I Jordan, *Serial order: A parallel distributed processing approach.* Advances in psychology, 121:471–495, 1986.
57. Vinod Nair, Geoffrey Hinton *Rectified Linear Units Improve Restricted Boltzmann Machines*, 27th International Conference on Machine Learning, Haifa, Isreal, 2010.
58. Sepp Hochreiter, JÃijrgen Schmidhuber *Long Short-Term Memory* Neural Computation, vol-9, Issue 8, 1997.

59. David Wolpert, William Macready, *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, April, 1997.
60. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, AxXiv e-prints, Dec 2017.
61. Paul Beniof, *The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines*, Journal of Statistical Physics, Vol. 22, No. 5, 1980.
62. Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, Antonio Criminisi, *Decision Jungles: COmpact and Rich Models for Classification*, NIPS 2013.
63. Olivier Chapelle, Jason Weston, Leon Bottou and Vladimir Vapnik, *Vicinal Risk Minization*, NIPS, 2000.
64. Vladimir Vapnik, *Principles of Risk Minimization for Learning Theory*, NIPS 1991.
65. Ameet Joshi, Lalita Udpa, Satish Udpa, Antonello Tamburrino, *Adaptive Wavelets for Characterizing Magnetic Flux Leakage Signals from Pipeline Inspection*, IEEE Transactions on Magentics, Vol. 42, No. 10, October 2006.
66. G. A. Rummery, Mahesh Niranjan *On-Line Q-Learning using Connectionist Systems*, volume 37. University of Cambridge, Department of Engineering.
67. S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, *Optimzation by Simulated Annealing*, Science, New Series, Vol. 220, No. 4598, 1983.
68. Craig W. Reynolds *Flocks, Herd and SChools: A Distributed Behavioral Model*, Computer Graphics, 21(4), July 1987, pp 25–34.
69. Lafferty, J., McCallum, A., Pereira, F. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*. Proc. 18th International Conf. on Machine Learning. Morgan Kaufmann. pp. 282–289, 2001.
70. Sinno Jialin Pan, Qiang Yang, *A Survey on Transfer Learning* IEEE Transactions on Knowledge and Data and Engineering, Vol. 22, No. 10, October 2010.

Index

A. V. Joshi, *Machine Learning and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-26622-6