MACHINE LEARNING

Practical Applications Using Amazon Web Services (AWS)

A Practical Introduction to Modern Machine Learning Techniques

Chapter 1

Introduction to Machine Learning

WHAT'S IN THIS CHAPTER

- Introduction to the basics of machine learning
- Tools commonly used by data scientists
- Applications of machine learning
- Types of machine learning systems
- Comparison between a traditional and a machine learning system

Hello and welcome to the exciting world of machine learning with Amazon Web Services (AWS). If you have never heard of machine learning until now, you may be tempted to think that it is a recent innovation in computer science that will result in sentient computer programs, significantly more intelligent than humans, that will one day make humans obsolete. There is very little truth in that idea of machine learning. For starters, it is not a recent development. For decades computer scientists have been researching ways to make computers more intelligent, attempting to find ways to teach computers to reason and to make decisions, generalizations, and predictions much like humans do.

Machine learning specifically deals with the problem of creating computer programs that can generalize and predict information reliably, quickly, and with accuracy resembling what a human would do with similar information. Building machine learning models can require a lot of processing power and storage space, and until recently was only possible to implement in very large companies or in academic institutions. Recent advances in storage, processor speeds, GPU technology, and the ability to rapidly create new virtual computing resources in the cloud have finally provided the processing power require to build and deploy machine learning systems at scale, and get results in real time.

Another factor that has contributed to the recent increase in machine learning applications is the availability of excellent tools such as Pandas, Matplotlib, TensorFlow, Scikit-learn, PyTorch, and Jupyter Notebook, which have made it possible for newcomers to start building real-world machine learning applications without having to delve into the complex underlying mathematical concepts.

Cloud computing as we know it today was born in 2006 when Amazon launched its Elastic Compute Cloud (EC2) service. Soon after in 2008, Microsoft launched its Azure service. This was followed by competing offers from other players, including Rackspace, Google, Oracle, and Apple. Building and deploying machine learning applications in the cloud is extremely popular. Most major cloud providers offer services to build and deploy some kind of machine learning applications.

You can find more information on the basics of cloud computing and Amazon Web Services in Chapters 6 and 7. In this chapter you will learn about what machine learning is, how machine learning systems are classified, and examples of real-world applications of machine learning.

What Is Machine Learning?

Machine learning is a discipline within Artificial Intelligence (AI) that deals with creating algorithms that learn from data. Machine learning traces its roots to a computer program created in 1959 by the computer scientist Arthur Samuel while he was working for IBM. Samuel's program could play a game of checkers and was based on assigning each position on the board a score that indicated the likelihood of leading toward winning the game. The positional scores were refined by having the program play against itself, and with each iteration the performance of the program improved. The program was, in effect, learning from experience, and the field of machine learning was born.

A machine learning system can be described as a set of algorithms based on mathematical principles that can mine data to find patterns in the data and then make predictions on new data as it is encountered. Rule-based systems can also make predictions on new data; however, rule-based systems and machine learning systems are not the same. A rule-based system requires a human to find patterns in the data and define a set of rules that can be applied by the algorithm. The rules are typically a series of `if-then-else` statements that are executed in a specific sequence. A machine learning system, on the other hand, discovers its own patterns and can continue to learn with each new prediction on unseen data.

Tools Commonly Used by Data Scientists

In this section you will learn about some of the tools commonly used by data scientists to build machine learning solutions. Most machine learning scientists use one of two programming languages: Python or R. R is a language commonly used by statisticians. While R has historically been the more popular choice, availability of machine learning–specific libraries in Python have made Python the more popular choice today. This book uses Python 3.6.5 and the rest of this section will focus on the most popular machine learning tools for Python:

- *Jupyter Notebook:* This is a very popular web-based interactive development environment for data science projects. A notebook combines code, execution results, and visualization results all in a single document. Jupyter Notebook is a successor to an older project called IPython Notebook. You can find out more about Jupyter Notebook at `http://jupyter.org`. Appendix A contains instructions on installing Anaconda Navigator and setting up Jupyter Notebook on your own computer.
- *Anaconda Navigator:* This is a commonly used package manager for data scientists. It allows users to conveniently install and manage Python and R libraries and quickly switch between different sets of libraries and language combinations. It also includes popular tools such as Jupyter Notebook, Spyder Python IDE, and R Studio. You can find more information on Anaconda at `https://www.anaconda.com`.
- *Scikit-learn:* This is a Python library that provides implementations of several machine learning algorithms for classification, regression, and clustering applications. It also provides powerful data preprocessing and dimensionality reduction capabilities. You can find more information on Scikit-learn at `https://www.scikit-learn.org`.

- *NumPy:* This is a Python library that is commonly used for scientific computing applications. It contains several useful operations such as random number generation and Fourier transforms. The most popular NumPy features for data scientists are N-dimensional data arrays (known as ndarrays) and functions that manipulate these arrays. NumPy ndarrays allow you to perform vector and matrix operations on arrays, which is significantly faster than using loops to perform element-wise mathematical operations. You can find more information on NumPy at `https://www.numpy.org`.
- *Pandas:* This is a Python library that provides a number of tools for data analysis. Pandas builds upon the NumPy ndarray and provides two objects that are frequently used by data scientists: the Series and the DataFrame. You can find more information on Pandas at `https://pandas.pydata.org`.
- *Matplotlib:* This is a popular 2D plotting library. It is used by data scientists for data visualization tasks. You can find more information on Matplotlib at `https://matplotlib.org`.
- *Pillow:* This is a library that provides a variety of functions to load, save, and manipulate digital images. It is used when the machine learning system needs to work with images. You can find more information on Pillow at `https://python-pillow.org`.
- *TensorFlow:* This is Python library for numerical computation. It was developed by Google and eventually released as an open source project in 2015. TensorFlow is commonly used to build deep-learning systems. It uses a unique computation-graph–based approach and requires users to build a computation graph where each node in the graph represents a mathematical operation and the connections between the nodes represent data (tensors). You can find more information on TensorFlow at `https://www.tensorflow.org`.
- *PyTorch:* This is another popular Python library for training and using deep-learning networks. It was built by Facebook, and many newcomers find it easier to work with than TensorFlow. You can find more information on PyTorch at `https://pytorch.org`.

Common Terminology

In this section we will examine some of the common machine learning–specific terminology that you are likely to encounter. While this list is not exhaustive, it should be useful to someone looking to get started:

- *Machine learning model:* This is the algorithm that is used to make predictions on data. It can also be thought of as a function that can be applied to the input data to arrive at the output predictions. The machine learning algorithm often has a set of parameters associated with it that influence its behavior, and these parameters are determined by a process known as *training*.
- *Data acquisition:* The process of gathering the data needed to train a machine learning model. This could include activities ranging from downloading ready-to-use CSV files to scraping the web for data.
- *Input variables:* These are the inputs that your machine learning model uses to generate its prediction. A collection of N input variables are generally denoted by lowercase x_i with i = 1, 2, 3, ...N. Input variables are also known as *features*.

- *Feature engineering:* This is the process of selecting the best set of input variables and often involves modifying the original input variables in creative ways to come up with new variables that are more meaningful in the context of the problem domain. Feature engineering is predominantly a manual task.
- *Target variable:* This is the value you are trying to predict and is generally denoted by a lowercase *y*. When you are training your model, you have a number of training samples for which you know the expected value of the target variable. The individual values of the target variable for N samples are often referred to as y_i with i = 1, 2, ...N.
- *Training data:* A set of data that contains all the input features as well as any engineered features and is used to train the model. For each item in the set, the value of the target variable is known.
- *Test data:* A set of data that contains all the input features (including engineered features), as well as the values of the target variable. This set is not used while training the model, but instead is used to measure the accuracy of the model's predictions.
- *Regression:* A statistical technique that attempts to find a mathematical relationship between a dependent variable and a set of independent variables. The dependent variable is usually called the target, and the independent variables are called the features.
- *Classification:* The task of using an algorithm to assign observations a label from a fixed set of predefined labels.
- *Linear regression:* A statistical technique that attempts to fit a straight line, plane, or hyperplane to a set of data points. Linear regression is commonly used to create machine learning models that can be used to predict continuous numeric values (such as height, width, age, etc.)
- *Logistic regression:* A statistical technique that uses the output of linear regression and converts it to a probability between 0 and 1 using a sigmoid function. Logistic regression is commonly used to create machine learning models that can predict class-wise probabilities. For example, the probability that a person will develop an illness later in life, or the probability that an applicant will default on a loan payment.
- *Decision tree:* A tree-like data structure that can be used for classification and prediction problems. Each node in the tree represents a condition, and each leaf represents a decision. Building a decision tree model involves examining the training data and determining the node structure that achieves the most accurate results.
- *Error function:* A mathematical function that takes as input the predicted and actual values and returns a numerical measure that captures the prediction error. The goal of the training function is to minimize the error function.
- *Neural networks:* A machine learning model that mimics the structure of the human brain. A neural network consists of multiple interconnected nodes, organized into distinct layers—the input layer, the in-between layers (also known as the hidden layers), and the output layer. Nodes are commonly known as neurons. The number of neurons in the

input layer correspond to the number of input features, and the number of neurons in the output layer correspond to the number of classes that are being predicted/classified.

- *Deep learning:* A branch of machine learning that utilizes multi-layer neural networks with a large number of nodes in each layer. It is also quite common for deep-learning models to use multiple deep neural networks in parallel.

Real-World Applications of Machine Learning

Machine learning is transforming business across several industries at an unprecedented rate. In this section you will learn about some of the applications of machine learning–based solutions:

- *Fraud detection:* Machine learning is commonly used in banks and financial institutions to make a decision on the overall risk associated with a payment instruction. Payments in this context include money transfers and purchases (payments to providers) using cards. The risk decision is based on several factors, including the transactional history. If the risk is low, the transaction is allowed to proceed. If the risk is too high, the transaction is declined. If the risk is deemed to lie in an acceptable threshold, the customer may be asked to perform some form of step-up authentication to allow the transaction to proceed.
- *Credit scoring:* Whenever a customer applies for a credit product such as a loan or credit card, a machine learning system computes a score to indicate the overall risk of the customer not being able to repay the loan.
- *Insurance premium calculation:* Machine learning systems are commonly used to compute the insurance premium that is quoted to customers when they apply to purchase an insurance product.
- *Behavioral biometrics:* Machine learning systems can be trained to build a profile of users based on the manner in which they use a website or a mobile application. Specifically, such systems create a profile of the user based on analyzing information on the location of the user at the time when the system was accessed, the time of the day, the precise click locations on a page, the length of time spent on a page, the speed at which the mouse is moved across the page, etc. Once the system has been trained, it can be used to provide real-time information on the likelihood that someone is impersonating a customer.
- *Product recommendations:* Machine learning systems are commonly used by online retailers (such as Amazon) to provide a list of recommendations to customers based on their purchase history. These systems can even predict when a customer is likely to run out of groceries or consumables and send reminders to order the item.
- *Churn prediction:* Machine learning systems are commonly used to predict which customers are likely to cancel their subscription to a product or service in the next few days. This information gives businesses an opportunity to try to retain the customer by offering a promotion.
- *Music and video recommendations:* Online content providers such as Netflix and Spotify use machine learning systems to build complex recommendation engines that analyze the movies and songs you listen to and provide recommendations on other content that you may like.

Types of Machine Learning Systems

There are several different types of machine learning systems today. The classification of a machine learning system is usually based on the manner in which the system is trained and the manner in which the system can make predictions. Machine learning systems are classified as follows:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning
- Batch Learning
- Incremental Learning
- Instance-based Learning
- Model-based Learning

These labels are not mutually exclusive; it is quite common to come across a machine learning system that falls into multiple categories. For example, a system that uses behavioral usage data to detect potential fraudsters on a banking website could be classified as a supervised model–based machine learning system. Let's examine these classification labels in more detail.

Supervised Learning

Supervised learning refers to the training phase of the machine learning system. During the supervised training phase, the machine learning algorithm is presented with sets of training data. Each set consists of inputs that the algorithm should use to make predictions as well as the desired (correct) result (Figure 1.1).

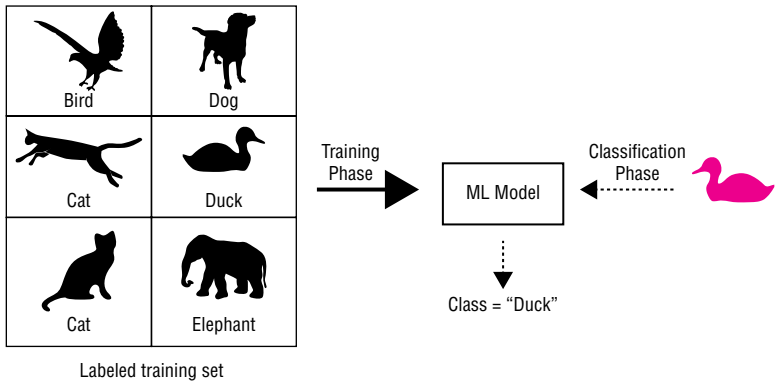


FIGURE 1.1
Supervised learning

Training consists of iterating over each set, presenting the inputs to the algorithm, and comparing the output of the algorithm with the desired result. The difference between the actual output and the desired output is used to adjust parameters of the algorithm so as make the output of the algorithm closer to (or equal to) the desired output.

Human supervision is typically needed to define the desired output for each input in the training set. Once the algorithm has learned to make predictions on the training set, it can be used to make predictions on data it has not previously encountered.

Most real-world machine learning applications are trained using supervised learning techniques. Some applications of supervised learning are:

- Finding objects in digital images
- Spam filtering
- Predicting the possibility of developing a medical condition based on lifestyle factors
- Predicting the likelihood of a financial transaction being fraudulent
- Predicting the price of property
- Recommending a product to a customer based on historical purchasing data
- A music streaming servicing suggesting a song to a customer based on what the customer has been listening to

Unsupervised Learning

Unsupervised learning also refers to the training phase of a machine learning system. However, unlike supervised learning, the algorithm is not given any information on the class/category associated with each item in the training set. Unsupervised learning algorithms are used when you want to discover new patterns in existing data. Unsupervised learning algorithms fall into two main categories:

Clustering These algorithms group the input data into a number of clusters based on patterns in the data. Visualizing these clusters can give you helpful insight into your data. Figure 1.2 shows the results of a clustering algorithm applied to the data on the heights and ages of children under 6 years of age. Some of the most popular clustering algorithms are k-means clustering, and Hierarchical Cluster Analysis (HCA).

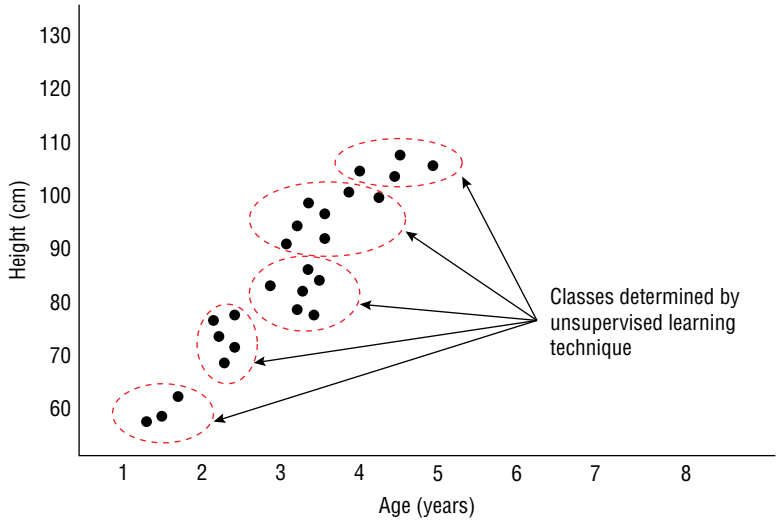


Figure 1.2 Clustering technique used to find patterns in the data

Dimensionality reduction These algorithms are used to combine a large number of input features into a smaller number of features without losing too much information. Typically, the features that are combined have a high degree of correlation with each other. Dimensionality reduction reduces the number of input features and therefore the risk of overfitting; it also reduces the computational complexity of a machine learning model. Some examples of algorithms in this category are Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Autoencoding. Dimensionality reduction is often used when the number of features in a dataset is too large for a data scientist to meaningfully analyze.

Semi-Supervised Learning

Semi-supervised learning is a mix of both supervised and unsupervised learning. In many situations it is practically impossible for a data scientist to label millions of samples for a supervised learning approach; however, the data scientist is already aware that there are known classifications in the data. In such a case, the data scientist labels a small portion of the data to indicate what the known classifications are, and the algorithm then processes the unlabeled data to better define the boundaries between these classes as well as potentially discover new classes altogether. Figure 1.3 depicts the results of applying semi-supervised learning to the same data on the heights and ages of children under 6 years of age, with some of the samples labeled to indicate the level of education.

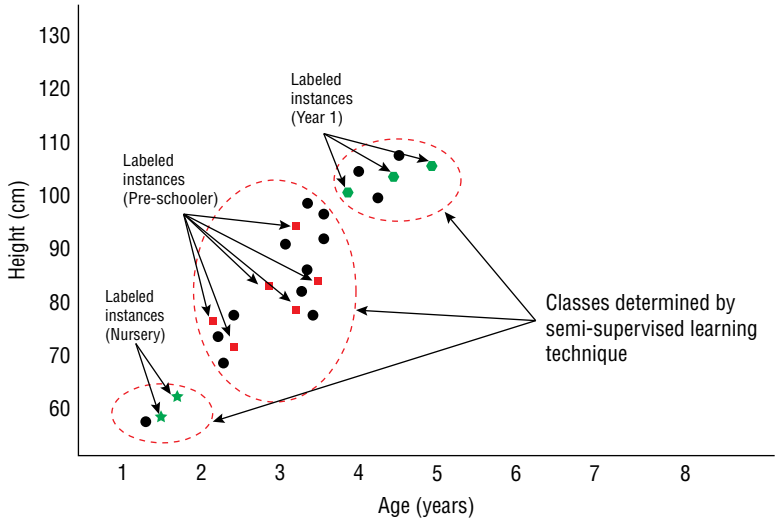


Figure 1.3
Semi-supervised learning

Real-world problems that involve extremely large datasets use this approach. Applications include speech recognition, natural language parsing, and gene sequencing.

A distinct advantage of semi-supervised learning is that it is less prone to labeling bias than supervised learning. This is because a data scientist is only labeling a small portion of the data, and the effects of any personal labeling bias introduced by the scientist can be corrected by the unsupervised learning part of the algorithm based on the sheer number of unlabeled samples it will process.

Semi-supervised learning algorithms make an assumption that the decision boundaries are geometrically simple; in other words, points that are close to each other are actually related in some way.

Reinforcement Learning

Reinforcement learning is a computational approach that attempts to learn using techniques similar to how humans learn: by interacting with their environment and associating positive and negative rewards with different actions. For instance, human beings have learned over time that touching a fire is a bad thing; however, using the same fire for cooking or providing warmth is a good thing. Therefore when we come across a fire we use it in positive ways.

A reinforcement learning–based system is typically called an *agent* and has a number of fixed actions that it can take at any given point in time, and with each action is an associated reward or penalty. The goal of the system is to maximize the cumulative reward over a series of actions. The knowledge gained by the agent is represented as a set of policies that dictate the actions that must be taken when the agent encounters a given situation. Reinforcement learning–based systems are used with two types of tasks:

- **Episodic tasks:** This is when the problem has a finite end point, such as winning a game.
- **Continuous tasks:** This is when the problem has no end point, such as maximizing the value of a stock portfolio.

Reinforcement learning algorithms often work in an environment where the results of choices are often delayed. Consider, for instance, an automated stock trading bot. It has several real-time inputs, manages several stocks, and can take one of three given actions at any point in time: buy a stock, sell a stock, or hold on to the stock. The result of several buy and sell decisions could be an eventual increase in the value of the portfolio (reward), or decrease in the value of the portfolio (penalty).

The bot does not know beforehand if taking a given action will result in reward or penalty. The bot must also incorporate a delay mechanism and not attempt to gauge reward/penalty immediately after making a transaction. The goal of the bot is to maximize the number of rewards.

In time the bot would have learned of trading strategies (policies) that it can apply in different situations. Reinforcement learning coupled with deep learning neural networks is a hotly researched topic among machine learning scientists.

Batch Learning

Batch learning refers to the practice of training a machine learning model on the entire dataset before using the model to make predictions. A batch learning system is unable to learn incrementally from new data it encounters in the future. If you have some new additional data that you want to use to train a batch learning system, you need to create a new model on all the data you have used previously plus the new additional data, and replace the older version of the model with this newly trained one.

Batch learning systems work in two distinct modes: learning and prediction. In learning mode the system is in training and cannot be used to make predictions. In prediction mode the system does not learn from observations.

The training process can be quite lengthy, and require several weeks and several computing resources. You also need to retain all training data in the event that you need to train a new version of the model. This can be problematic if the training data is large and requires several gigabytes of storage.

Incremental Learning

Incremental learning, also known as online learning, refers to the practice of training a machine learning model continuously using small batches of data and incrementally improving the performance of the model. The size of a mini batch can range from a single item to several hundred items. Incremental learning is useful in scenarios where the training data is available in small chunks over a period of time, or the training data is too large to fit in memory at once. The aim of incremental learning is to not have to retrain the model with all the previous training data; instead, the model's parameters (knowledge) are updated by a small increment with each new mini batch that it encounters. Incremental learning is often applied to real-time streaming data or very large datasets.

Instance-based Learning

Instance-based learning systems make a prediction on new unseen data by picking the closest instance from instances in the training dataset. In effect the machine learning system memorizes the training dataset and prediction is simply a matter of finding the closest matching item. The training phase of instance-based learning systems involves organizing all the training data in an appropriate data structure so that finding the closest item during the prediction phase will be quicker. There is little (if any) model tuning involved, although some level of preprocessing may have been performed on the training data before presenting it to the machine learning system.

The advantage of an instance-based learning system is that both training and prediction are relatively quick, and adding more items to the training set will generally improve the accuracy of the system. It is important to note that the prediction from an instance-based system will be an instance that exists in the training set. For example, consider an instance-based machine learning system that predicts the shoe size of an individual given a height and weight value as input. Let's assume this system is trained using a training set of 100 items, where each item consists of a height, weight, and shoe size value.

If this instance-based machine learning system were to be used to predict the shoe size for a new individual given a height and weight of the new individual as input values, the predicted shoe size would be the value of the closest matching item in the training set. To put it another way, the machine learning system would never output a shoe size that was not in the training set to start with.

Model-based Learning

Model-based learning systems attempt to build a mathematical, hierarchical, or graph-based model that models the relationships between the inputs and the output. Prediction involves solving the model to arrive at the result. Most machine learning algorithms fall in this category.

While model-based systems require more time to build, the size of the model itself is a fraction of the size of the training data, and the training data need not be retained (or presented to the model) in order to get a prediction. For example, a model built from a training dataset of over a million items could be stored in a small five-element vector.

The Traditional Versus the Machine Learning Approach

As humans, all of us are familiar with the concept of learning. Learning takes two major forms: memorization and understanding. There is a clear difference between memorizing your password and learning to drive a car. The latter involves understanding how the vehicle works, and how to react in different situations on the road. You do not memorize the exact sequence of activities you need to perform to drive between your home and your place of work; instead, you apply the understanding you have gained while assessing the situation on the road.

The capabilities of machine learning algorithms are not as sophisticated as human learning. Machines do not have the capability to understand and reason; however, they can predict and generalize, and they are capable of processing data much faster than humans. In this section you will examine a hypothetical situation and understand how a machine learning system can be applied to solve the problem at hand.

Imagine you have been hired by the credit cards team at a bank and tasked with creating a solution to offer a new credit card product to customers. Eligible customers can, under this new product, get cards with credit limits up to $25,000 at low interest rates. The bank would like to keep its losses to a minimum and requires that the credit card only be offered to customers who are likely to pay the money back to the bank.

To start with, you decide to create a new application form that customers will need to fill out to apply for this credit card. On the application form, you ask for the following information:

- Name
- Age
- Sex
- Number of years lived at current address
- Number of addresses in the last 5 years
- Number of credit cards held with other banks
- Total amount of loan (excluding mortgage)
- Total income after tax
- Estimated regular monthly outgoings
- Total monthly repayments
- Homeowner status
- Marital status
- Number of dependents in the household

In a real-world scenario you would ask for a lot more information and would also use a credit scoring company to provide the applicant's credit rating, but for the purposes of this example this list will do. Let's also assume that the mechanism to allow customers to apply for the credit card is available on the bank's website, and data from all application forms is available in a table in a SQL database within the bank.

When the product is launched, your bank expects it to be popular with customers and you need to have a plan in place to process a large number of applications each week. It is also crucial for your bank to remain competitive in the credit card space, and therefore you cannot keep applicants waiting for weeks to find out the outcome of a credit card application. You need to ensure that the bank's standard terms of service apply to your new product and therefore at least 51% of all credit applications need to be decided within a few minutes of the application being received by the bank.

Clearly, it is not cost effective to a hire few hundred analysts to scrutinize each loan application and make the necessary checks to arrive at a decision. You have two choices before you:

- You can build a rule-based decisioning system
- You can build a machine learning system

Let's first examine how a rule-based system could be used to help with this problem.

A Rule-based Decision System

Faced with the prospect of choosing between an easy-to-understand rule-based approach, and a somewhat unknown, black-box machine learning system, you decide to go with a rule-based decision system.

The business rules themselves are to be stored in a database, and the business logic to apply the rules to a loan application will be encapsulated into a server-side application. You plan to have a cluster of these servers that can be scaled up as necessary to deal with increased volumes. Figure 1.4 depicts the architecture of the proposed system.

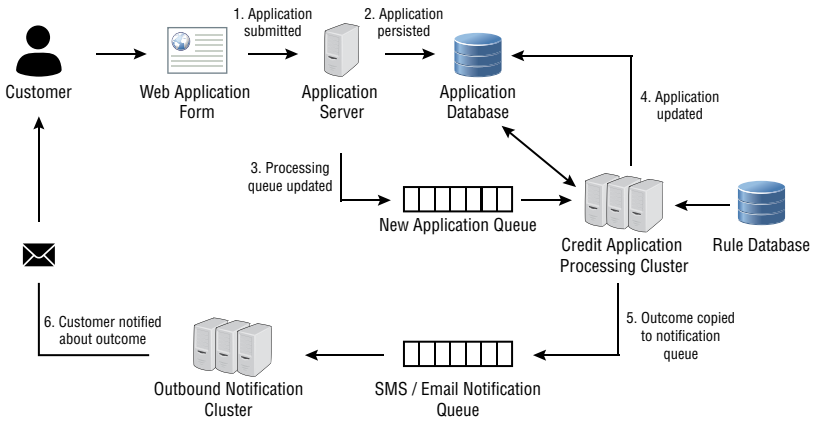


FIGURE 1.4
Architecture of a rule-based decision system

When an applicant submits a credit card application on the bank's website, a unique card application identifier is generated, and a row is added to a database by the web application. The web application also publishes a message onto a message queue; the content of the message contains the unique application identifier. A cluster of loan processing servers subscribe to the message queue and the first available servers will pick up the message and begin processing the credit card application. When the server reaches a decision, it will update the loan application in

the database and publish a message on another message queue. The contents of this new message will contain all the information needed to send an SMS or email notification to the customer, informing the customer of the outcome.

How do you define the rules that will be used to make the decision? To start with you will need to create the rules based on your experience of the problem domain and qualities of the loan application form that you feel are favorable. If your business has another rule-based system that your solution is trying to replace, you may also be able to port some of the rules from the existing system.

Let's assume you do not have access to an existing system and need to define the business rules yourself. Looking at the fields in your loan application form, you decide that the following fields can be discarded from the decision-making process as you feel they are not likely to have any impact on the ability of individuals to keep up their monthly repayments:

- Name
- Sex
- Marital status
- Number of dependents in the household
- Homeowner status

You are now left with the following information that you can use to build your rules:

- Number of years lived at current address
- Number of addresses in the last 5 years
- Number of credit cards held with other banks
- Total amount of loan (excluding mortgage)
- Total income after tax
- Estimated regular monthly outgoings
- Total monthly repayments

You decide to create rules that will reject applications that meet any of the following criteria:

1. Applicants that have lived less than 1 year at their current address.
2. Applicants that have had more than three addresses in the last 5 years.
3. Applicants that have two or more credit cards with other banks.
4. Applicants whose disposable monthly income is less than 15% of their total income after tax.

These rules can be coded in any programming language using `if-then-else` statements. Figure 1.5 contains a flowchart that depicts the decision-making process using the rules you have defined.

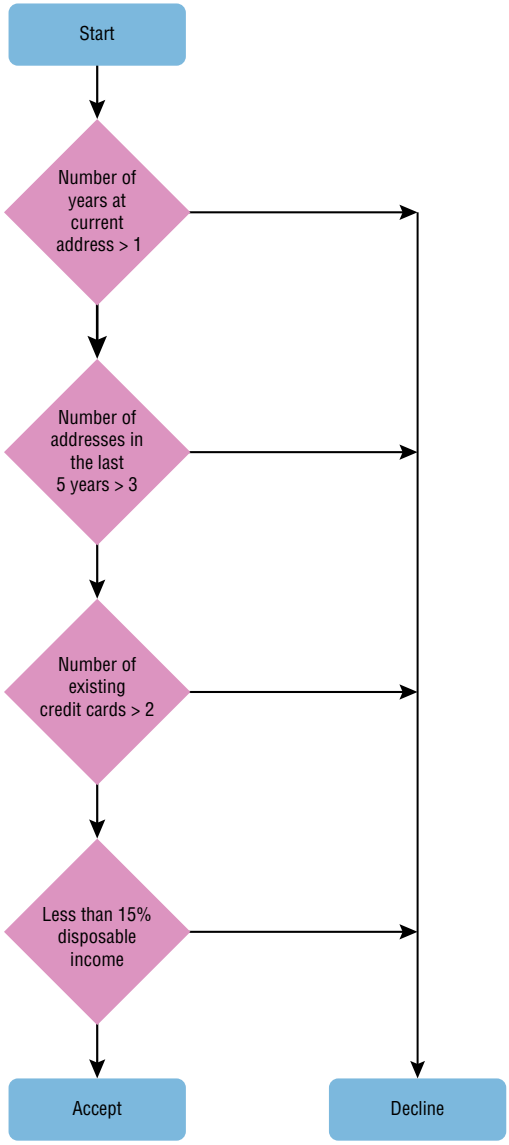


FIGURE 1.5
A flowchart depicting the decision-making process for a rule-based system

There is a possibility that the rules you have created and the initial assumptions you have made were not optimal, but you can always tailor the rules if you observe the losses due to missed payments exceed the bank's risk threshold. In time, as the volume of credit applications increases you will need to tailor your business rules to deal with the increased volume, and you are likely to encounter a few problems:

- Your rules become very complicated and interdependent; it becomes increasingly complex to replace old rules with new ones.

- Your rules have not been based on any numerical analysis; they have been created ad hoc to address increasing demand.
- Your rules do not account for changing patterns in the economy; for example, in the midst of a recession a larger number of people may end up missing their payments even though your rules predict that they should not. You will need to constantly keep updating your rules and that can be a costly and time-consuming process.

Let's now see how a machine learning–based system could be used to address these problems.

A Machine Learning–based System

Unlike a rule-based solution, you cannot create a machine learning solution without historical data from previous applicants. It is important to have data that is accurate and relevant to the problem you are trying to solve. You may be tempted to build your machine learning model using data from a different loan product, such as a personal loan. This is not recommended, as there could be trends in that data that are applicable only for personal loan applications.

You may be wondering why there was no need for historical data while building the rule-based system, and why defining the rules based on an intuitive understanding of the questions alone was sufficient. The reason is that in the rule-based approach, you were defining the rules based on your personal knowledge and experience of the problem domain. A machine-learning system, on the other hand, does not start with any prior knowledge of the problem domain. It does not have the intuitive capability of a human being. Instead, a machine learning system attempts to make its own inferences purely from the data it encounters. Not only do you need data, but also relevant data. If the data that is used to train the machine learning system is significantly different from what the system will encounter when asked to make predictions, the predictions are likely to be incorrect.

Just how much data will you need? There is no simple answer to that question. In general, machine learning algorithms require a lot of data to be trained effectively, but it is not just the quantity that matters; it is also the quality of the data. If your data has too many samples that follow a particular trend, then your machine learning system will be biased toward that trend.

For the purpose of this example, let's assume you have decided to pick the data for 5000 applicants randomly from your database, and exported these rows to a CSV file. These are all applicants that have applied for your credit card product and have either kept up with their regular payments or missed at most one payment. The input variables in this case would be answers to the questions asked on the form; the target variable will be a Boolean that indicates whether the applicant missed at most one payment.

Such type of data, where you already know the value of the target variable, is known as *labeled data* and is commonly used for training machine learning models. Not all data is labeled; in fact, the vast majority of data in the world is unlabeled. Using unlabeled (or partially labeled) data to train machine learning algorithms is an active area of research.

What if you want to build a machine learning solution, but have absolutely no data of your own, and can't wait months (or years) to collect high-quality labeled data? Your best option in this case would be to try to find existing datasets from free or paid sources on the Internet that are as close as possible to the problem domain you are working in. Chapter 2 provides links to several public machine learning datasets.

Picking Input Features

Once you have collected the labeled input data, you will need to perform basic statistical analysis on the data to pick input variables that are most likely to be relevant and prepare the input data for the machine learning model. Having the data in a CSV file makes it easy to load into data analysis tools. Most cloud-based data analysis tools allow you to upload CSV files, and some also support importing data from cloud-based relational databases.

Data visualization techniques are commonly used to get an indication of the spread and correlation of individual features. You will most likely build and train a model using an initial set of features and calculate the value of a performance metric while making predictions with the model. You will then retrain the model with additional input features and repeat as necessary as long as you observe a significant improvement in the value of the performance metric.

The number of features you choose will also impact the size of the training dataset you are likely to need. A model with 3 features could be built using a dataset of 500 items, whereas a model with 10 features will likely need a training dataset that consists of several thousands of items.

It may surprise you to learn that the bulk of a data scientist's job is looking at data and working out ways to use it. This work is often referred to as *data munging*, and involves several steps, including but not limited to:

- Finding out if there are any missing values in the data
- Working out the best way to handle missing values
- Examining the statistical distribution of the data
- Examining the correlation between input variables
- Creating new input variables by splitting or combining existing input variables

NOTE *Munge* is a technical term coined by MIT students and means to transform data using a series of reversible steps to arrive at a different representation. Data munging is also known as data wrangling and feature engineering.

R, Python, Jupyter Notebook, NumPy, Pandas, and Matplotlib are commonly used for statistical analysis and feature engineering. Chapters 2 and 3 cover a number of techniques that can be used for feature engineering and data visualization to arrive at the final list of input features.

Let's now examine the questions on the application form and perform some simple feature engineering, based on your experience with the rule-based approach, to arrive at the inputs to our first machine learning model. Table 1.1 lists all the questions on the loan application form with their data types, range of expected values, and range of actual values as observed in 5000 samples.

It is quite clear that some of the answers are categorical while others are numeric. The numeric features themselves are either discrete or continuous and have different ranges of values.

Table 1.1: Type and Range of Data across 100 Sample Applications

Question	Type of Data	Actual Range	Maximum Range
Name	Free Text	Characters A–Z, some special characters	Characters A–Z, some special characters
Age	Continuous Numeric	22–45	18–150
Sex	Categorical	Male, Female, Undisclosed	Male, Female, Undisclosed
Number of years lived at current address	Continuous Numeric	0–7	0–150
Number of addresses in the last 5 years	Continuous Numeric	1–6	1–10
Number of credit cards held with other banks	Discrete Numeric	1, 2, 3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Total amount of loan (excluding mortgage)	Continuous Numeric	0–17500	0–10 million
Total income after tax	Continuous Numeric	50000–200000	0–10 million
Estimated regular monthly outgoings	Continuous Numeric	3000–15000	0–10 million
Total monthly repayments	Continuous Numeric	0–1500	0–10 million
Homeowner status	Categorical	Yes or No	Yes or No
Marital status	Categorical	Single, Married, Undisclosed	Single, Married, Undisclosed
Number of dependents in the household	Discrete Numeric	1, 2, 3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Most machine learning models are designed to perform better on one type of feature over the other. Some statistical-based learning models are also sensitive to large numeric values. After examining the type of questions on the application form, the range of input values, and sample data for 5000 applicants, you decide that you want to use all of the answers on the application form as input features, except for the following:

- Name of the applicant
- Number of dependents
- Number of credit cards held with other banks

You also decide to engineer a new feature called Total Disposable Income, and split three categorical features into discrete numeric features as listed in Table 1.2.

TABLE 1.2: Transforming Categorical Features into Numeric Features

ORIGINAL CATEGORICAL FEATURE	NEW NUMERIC FEATURE	ALLOWED VALUES/RANGE
Sex	SexIsMale	0 or 1
	SexIsFemale	0 or 1
	SexIsUndisclosed	0 or 1
Homeowner status	IsHomeOwner	0 or 1
	IsNotHomeOwner	0 or 1
Marital status	MaritalStatusIsSingle	0 or 1
	MaritalStatusIsMarried	0 or 1
	MaritalStatusIsUndisclosed	0 or 1

The new engineered feature Total Disposable Income is defined as:

Total Disposable Income = Total Income After Tax – Regular Monthly outgoings – Monthly Loan Repayments

Feature engineering techniques are covered in Chapter 2, but it is worth noting that some of the questions that have been answered as *undisclosed* are being treated as first-class values, as opposed to treating the question as missing an answer.

Table 1.3 lists the new set of input and engineered features, the expected range of values for each feature, and the actual range of values as observed in 5000 sample application forms.

TABLE 1.3: Modified Input Features

FEATURE NAME	FEATURE DESCRIPTION	TYPE OF DATA	OBSERVED RANGE	MAXIMUM RANGE
F1	Age	Continuous Numeric	22–45	18–150
F2	SexIsMale	Discrete Numeric	0, 1	0 or 1
F3	SexIsFemale	Discrete Numeric	0, 1	0 or 1
F4	SexIsUndisclosed	Discrete Numeric	0, 1	0 or 1
F5	Number of years lived at current address	Continuous Numeric	0–7	0–150

Table 1.3: Modified Input Features *(continued)*

Feature Name	Feature Description	Type of Data	Observed Range	Maximum Range
F6	Number of addresses in the last 5 years	Continuous Numeric	1–6	0–150
F7	Total Disposable Income	Continuous Numeric	0–5000	0–10 million
F8	Total amount of loan (excluding mortgage)	Continuous Numeric	0–17500	0–10 million
F9	Total income after tax	Continuous Numeric	50000–200000	0–10 million
F10	Estimated regular monthly outgoings	Continuous Numeric	3000–15000	0–10 million
F11	Total monthly repayments	Continuous Numeric	0–1500	0–10 million
F12	IsHomeOwner	Discrete Numeric	0, 1	0 or 1
F13	IsNotHomeOwner	Discrete Numeric	0, 1	0 or 1
F14	MaritalStatusIsSingle	Discrete Numeric	0, 1	0 or 1
F15	MaritalStatusIsMarried	Discrete Numeric	0, 1	0 or 1
F16	MaritalStatusIsUndisclosed	Discrete Numeric	0, 1	0 or 1

It is quite clear that the range of allowed values for the features are not all the same. For example, F1 (Age) values lie in the range [18,100], whereas other feature values lie in the ranges [0,1], [0, 150], [1,10] and [0, 10000000]. Before you can use this data you will need to normalize the values of all the features to lie in the same range [0, 1]. Normalization is covered in Chapter 2, and involves transforming the value of each feature so that it lies in the interval [0, 1]. You will need to normalize both the training data as well as data on which your solution will make predictions.

You could also have chosen to transform the numeric features into categorical features by defining ranges of allowed values. For example, a given response to the *total income after tax* question, which is a continuous value between 0 and 10 million, could be converted into the following categorical value:

- Between0and100000
- Between100001and500000
- Above500001

There is no right or wrong way to engineer features. Sometimes the choice of machine learning algorithm dictates the type of features (numerical or categorical). Some algorithms, such as linear regression, work with numeric features; others that are based on decision trees work

better with categorical features. Sometimes you start with a set of features only to realize that the predictive accuracy of the model trained on those features is not good enough and you need to start from scratch with new features, or perhaps a different algorithm or training technique.

Preparing the Training and Test Set

Before you can build a machine learning model, you need to create separate training and testing datasets. The training set is used to build the model, whereas the test set is used to evaluate the performance of the model. The reason to have a separate training and test set is that a well-trained model will always perform well on the training set, as it has learned all the items in the training set. To truly measure the performance of the model, you need to present it data that it has not encountered during training.

Typically you will use 70% of the data for training and hold out 30% of the data for testing; however, other splits, such as 60:40, are also commonly used. When preparing the training and test set it is important to ensure that the members of the test set are well distributed, and do not exhibit bias toward any particular trend. For instance, if the training set consisted of a large proportion of applicants that were female and had annual incomes exceeding $100,000, the model is likely to incorporate this trend as part of its learning and perform poorly on other types of applicants. When a model performs well on the training set and poorly on the test set, the model is said to overfit the training data.

Cross-validation can help minimize the possibility of the model picking up on unexpected bias in the training set. The idea behind cross-validation is to shuffle the entire dataset randomly and divide it into a number of smaller sets (known as folds). If, for instance, the data were divided into 10 equal folds, the model would be trained and evaluated 10 times. During each training and evaluation cycle, one of the folds will be held out as the test set and the remaining nine will make the training set. This is illustrated in Figure 1.6.

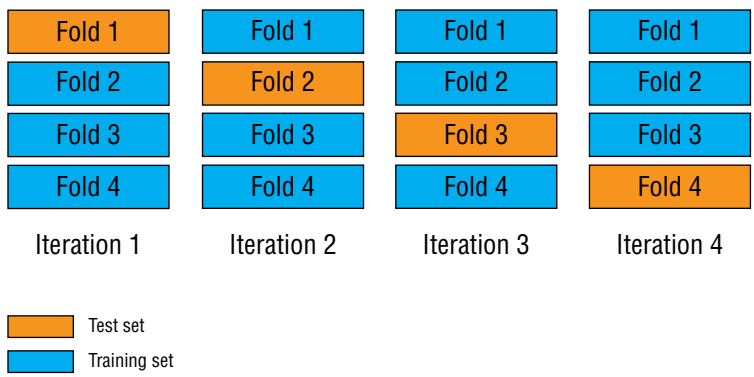


Figure 1.6
Cross-validation using multiple folds

Sometimes the bias is introduced during the data collection phase (also known as sampling bias), and no amount of cross-validation can help remove it. For instance, you may be building a model based on a dataset that contains information on consumer shopping trends collected during the Christmas holiday season. This data will be biased toward trends that are specific to holiday shopping, and a model built using this data will perform poorly if used to predict what consumers may buy over the course of the entire year. In such cases the best option is to collect more data to remove the sampling bias.

Picking a Machine Learning Algorithm

There are a number of machine learning algorithms that can be used for classification tasks. In the current example, the objective is to decide, at the point of application, whether an applicant should be issued a credit card. The machine learning system that makes this decision is to be trained on data that is labeled to indicate if the customer has historically missed any payments (negative outcome), or if the customer did not miss a single payment (positive outcome).

The decision to issue the credit card is directly related to whether the machine learning system indicates the customer will default or not. This is a typical classification problem and a number of algorithms could be used:

- Logistic regression
- Decision trees
- Random forests
- XGBoost
- Neural networks
- Clustering-based techniques

The choice of algorithm is often influenced by factors such as desired accuracy, number of classes desired (binary vs. multi-class classification), availability of sufficient training data, time taken to train the model, memory footprint of the trained model, and resources required to deploy the model into production.

In this example we will make use of an algorithm called *logistic regression*, which is an algorithm that performs well for binary classification problems—problems that involve classifying something into one of two classes. Logistic regression builds upon the output of an algorithm called *linear regression*. Linear regression is a simple and effective algorithm for predicting continuous numeric values and assumes that the output variable can be expressed as a linear combination of the input features.

In effect, linear regression attempts to find the best line (or hyperplane in higher dimensions) that fits all the data points. The output of linear regression is a continuous, unbounded value. It can be a positive number or a negative number. It can have any value, depending on the inputs with which the model was trained.

In order to use a continuous value for binary classification, logistic regression converts it into a probability value between 0.0 and 1.0 by feeding the output of linear regression into a sigmoid function. The graph of the sigmoid function is presented in Figure 1.7. The output of the sigmoid function will never go below 0.0 or above 1.0, regardless of the value of the input.

The output of the sigmoid function can be used for binary classification by setting a threshold value and treating all values below that as class A and everything above the threshold as class B (Figure 1.8).

While logistic regression is simple to understand, it may fail to provide good results if the relationship between the target variable and the input features is complex. In such a case you can consider using a tree-based model such as decision trees or an instance learning–based clustering model. You will learn more about model building in Chapter 4.

FIGURE 1.7
The sigmoid function

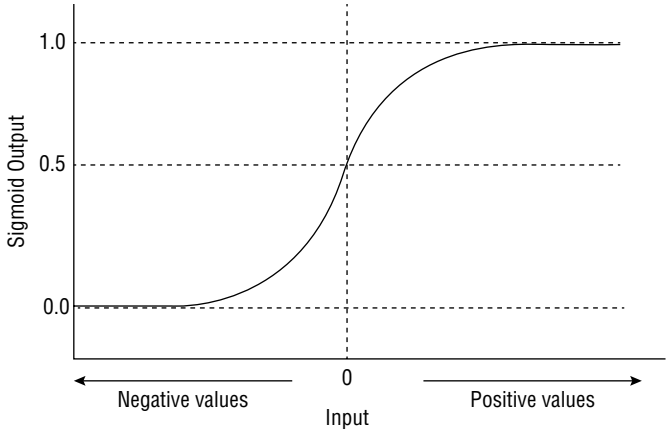
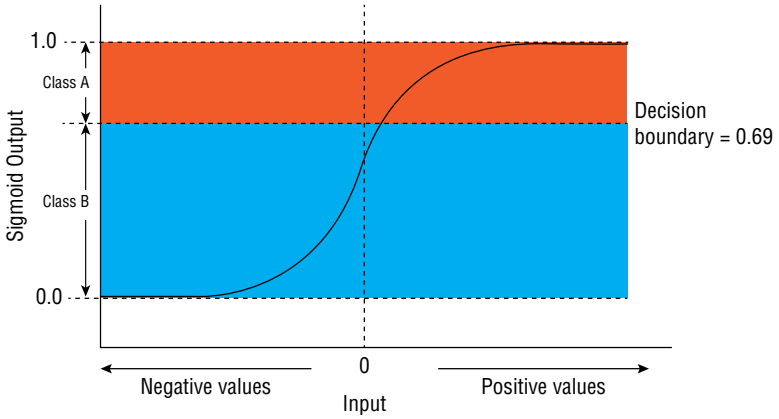


FIGURE 1.8
Using the sigmoid function for binary classification



EVALUATING MODEL PERFORMANCE

Once you have built a model that can be used to determine if an applicant should be issued a credit card, you need to be able to quantitatively measure its performance. The purpose of evaluating a model is to determine how well it works on new data—models are evaluated using the labeled test set that was set aside during the training phase.

Since the model in this example attempts to solve a classification problem, you could simply count the number of times the model predicted the correct outcome and use this as an evaluation metric.

If issuing a credit card to an applicant is considered as the positive outcome, you can create a few additional metrics that go beyond simple counting:

- **True positive count:** The number of times the model predicted that a credit card should be issued to an applicant, and this decision matched the outcome in labeled test dataset.

- **False positive count:** The number of times the model predicted that a credit card should be issued to an applicant where the labeled test data indicated that applicant should not have been successful.
- **True negative count:** The number of times the model predicted that a credit card should not be issued to an applicant, and this decision matched the outcome in test dataset.
- **False negative count:** The number of times the model predicted that a credit card should not be issued to an applicant, and this decision was not correct according to the labeled test data.

You will learn to use these metrics and other performance evaluation and model tuning techniques in Chapter 5.

Summary

- Machine learning is a discipline within artificial intelligence that deals with creating algorithms that learn from data.
- Machine learning deals with the problem of creating computer programs that can generalize and predict information reliably and quickly.
- Machine learning is commonly used to implement fraud-detection systems, credit-scoring systems, authentication-decision engines, behavioral biometric systems, churn prediction, and product-recommendation engines.
- The type of training data that is required to train a machine learning system can be used to classify the machine learning system into supervised, unsupervised, or semi-supervised learning.
- Batch learning refers to the practice of training a machine learning model on the entire dataset before using the model to make predictions.
- Incremental learning, also known as online learning, refers to the practice of training a machine learning model continuously using small batches of data and incrementally improving the performance of the model.
- Instance-based learning systems make a prediction on new unseen data by picking the closest instance from instances in the training dataset.
- Model-based learning systems attempt to build a mathematical, hierarchical, or graph-based model that models the relationships between the inputs and the output.

Chapter 2

Data Collection and Preprocessing

WHAT'S IN THIS CHAPTER

- Sources to obtain training data
- Techniques to explore data
- Techniques to impute missing values
- Feature engineering techniques

In the previous chapter, you were given a general overview of machine learning, and learned about the different types of machine learning systems. In this chapter you will learn to use NumPy, Pandas, and Scikit-learn to perform common feature engineering tasks.

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from `www.wiley.com/go/machinelearningawscloud` or from GitHub using the following URL:

`https://github.com/asmtechnology/awsmlbook-chapter2.git`

Machine Learning Datasets

Training a machine learning model requires high-quality data. In fact, lack of quality training data can result in the poor performance of models built using the best-known machine learning algorithms. Quality in this case refers to the ability of the training data to accurately capture the nuances of the underlying problem domain, and to be reasonably free of errors and omissions.

Some of the common sources for publicly available machine learning data are explored next.

Scikit-learn Datasets

The datasets package within Scikit-learn includes down-sampled versions of popular machine learning datasets such as the Iris, Boston, and Digits datasets. These datasets are often referred to as *toy datasets*. Scikit-learn provides functions to load the toy dataset into a dictionary-like object with the following attributes:

- *DESCR*: Returns a human-readable description of the dataset.
- *data*: Returns a NumPy array that contains the data for all the features.

- *feature_names*: Returns a NumPy array that contains the names of the features. Not all toy datasets support this attribute.
- *target*: Returns a NumPy array that contains the data for the target variable.
- *target_names*: Returns a NumPy array that contains the values of categorical target variables. The digits, Boston house prices, and diabetes datasets do not support this attribute.

The following snippet loads the toy version of the popular Iris dataset and explores the attributes of the dataset:

```
#load Scikit-learn's downsampled iris dataset
from sklearn import datasets
iris_dataset = datasets.load_iris()

# explore the dataset
print (iris_dataset.DESCR)
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
```

```
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

.....

print (iris_dataset.data.shape)
(150, 4)

print (iris_dataset.feature_names)
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']

print (iris_dataset.target.shape)
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']

print (iris_dataset.target_names)
['setosa' 'versicolor' 'virginica']
```

The list of toy datasets included with Scikit-learn are:

- *Boston house prices dataset*: This is a popular dataset used for building regression models. The toy version of this dataset can be loaded using the `load_boston()` function. You can find the full version of this dataset at `https://archive.ics.uci.edu/ml/machine-learning-databases/housing/`.
- *Iris plants dataset*: This is a popular dataset used for building classification models. The toy version of this dataset can be loaded using the `load_iris()` function. You can find the full version of this dataset at `https://archive.ics.uci.edu/ml/datasets/iris`.
- *Onset of diabetes dataset*: This is a popular dataset used for building regression models. The toy version of this dataset can be loaded using the `load_diabetes()` function. You can find the full version of this dataset at `http://www4.stat.ncsu.edu/~boos/var.select/diabetes.html`.
- *Handwritten digits dataset*: This is a dataset of images of handwritten digits 0 to 9 and is used in classification tasks. The toy version of this dataset can be loaded using the `load_digits()` function. You can find the full version of this dataset at `http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`.
- *Linnerud dataset*: This is a dataset of exercise variables measured in middle-aged men and is used for multivariate regression. The toy version of this dataset can be loaded using the `load_linnerud()` function. You can find the full version of this dataset at `https://rdrr.io/cran/mixOmics/man/linnerud.html`.

- *Wine recognition dataset*: This dataset is a result of chemical analysis performed on wines grown in Italy. It is used for classification tasks. The toy version of this dataset can be loaded using the `load_wine()` function. You can find the full version of this dataset at `https://archive.ics.uci.edu/ml/machine-learning-databases/wine/`.
- *Breast cancer dataset*: This dataset describes the characteristics of cell nuclei of breast cancer tumors. It is used for classification tasks. The toy version of this dataset can be loaded using the `load_breast_cancer()` function. You can find the full version of this dataset at `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`.

AWS Public Datasets

Amazon hosts a repository of public machine learning datasets that can be easily integrated into applications that are deployed onto AWS. The datasets are available as S3 buckets or EBS volumes. Datasets that are available in S3 buckets can be accessed using the AWS CLI, AWS SDKs, or the S3 HTTP query API. Datasets that are available in EBS volumes will need to be attached to an EC2 instance. Public datasets are available in the following categories:

- *Biology*: Includes popular datasets such as the Human Genome Project.
- *Chemistry*: Includes multiple versions of PubChem and other content. PubChem is a database of chemical molecules that can be accessed at `https://pubchem.ncbi.nlm.nih.gov`.
- *Economics*: Includes census data and other content.
- *Encyclopedic*: Includes Wikipedia content and other content.

You can browse the list of AWS public datasets at `https://registry.opendata.aws`.

`Kaggle.com` Datasets

`Kaggle.com` is a popular website that hosts machine learning competitions. `Kaggle.com` also contains a large number of datasets for general use that can be accessed at `https://www.kaggle.com/datasets`. In addition to the general-use datasets listed on the page, competitions on `Kaggle.com` also have their own datasets that can be accessed by taking part in the competition. The dataset files can be downloaded onto your local computer and can then be loaded into Pandas dataframes. You can get a list of current and past competitions at `https://www.kaggle.com/competitions`.

UCI Machine Learning Repository

The UCI machine learning repository is a public collection of over 450 datasets that is maintained by the Center for Machine Learning and Intelligent Systems at UC Irvine. It is one of the oldest sources of machine learning datasets and is often the go-to destination for beginners and experienced professionals alike. The datasets are contributed by the general public and vary in the level of preprocessing you will need to perform in order to use them for model building. The datasets can be downloaded onto your local computer and then processed using tools like Pandas and Scikit-learn. You can browse the complete list of datasets at `https://archive.ics.uci.edu/ml/datasets.php`.

A small selection of the most popular UCI machine learning repository datasets is also hosted at `Kaggle.com` and can be accessed at `https://www.kaggle.com/uciml`.

Data Preprocessing Techniques

In Chapter 1, you learned about the different types of machine learning systems and the general process in building a machine learning–based solution. It should come as no surprise that the performance of a machine learning system is heavily dependent on the quality of training data. In this section, you will learn some of the common ways in which data is prepared for machine learning models. The examples in this section will use datasets commonly found on the Internet and included with the downloads that accompany this lesson.

Obtaining an Overview of the Data

When building a machine learning model, one of the first things you will want to do is explore the data to get an overview of the variables and the target. This section uses the Titanic dataset in a Jupyter notebook with NumPy and Pandas. The dataset and the notebook files are included with the lesson's code resources.

The Titanic dataset is a very popular dataset that contains information on the demographic and ticket information of 1309 passengers on board the Titanic, with the goal being to predict which of the passengers were more likely to survive. The full dataset is available from the Department of BioStatistics at Vanderbilt University (`https://biostat.mc.vanderbilt.edu/wiki/Main/DataSets`). Versions of the titanic3 dataset are also available from several other sources, including a popular `Kaggle.com` competition titled Titanic: Machine Learning From Disaster (`https://www.kaggle.com/c/titanic`). The Kaggle version is included with the resources that accompany this chapter and has the benefit of being shuffled and pre-split into a training and validation set. The training set is contained in a file named `train.csv` and the validation set is `test.csv`.

The description of the attributes of the Kaggle version of the Titanic dataset are as follows:

- *PassengerId*: A text variable that acts as a row identifier.
- *Survived*: A Boolean variable that indicates if the person survived the disaster. 0 = No, 1 = Yes.
- *Pclass*: A categorical variable that indicates the ticket class. 1 = 1st class, 2 = 2nd class, 3 = 3rd class.
- *Name*: The name of the passenger.
- *Sex*: A categorical variable that indicates the sex of the passenger.
- *Age*: A numeric variable that indicates the age of the passenger.
- *SibSp*: A numeric variable that indicates the number of siblings/spouses traveling together.
- *Parch*: A numeric variable that indicates the number of parents and children traveling together.

- *Ticket*: A text variable containing the ticket number.
- *Fare*: A numeric variable that indicates the fare paid in Pre-1970 British pounds.
- *Cabin*: A textual variable that indicates the cabin number.
- *Embarked*: A categorical variable that indicates the port of embarkation. C = Cherbourg, Q = Queenstown, S = Southampton.

To load the Titanic training set from a CSV file located on your computer into a Pandas dataframe, use the following snippet:

```
import numpy as np
import pandas as pd

# load the contents of a file into a pandas Dataframe
input_file = './datasets/titanic_dataset/original/train.csv'
df_titanic = pd.read_csv(input_file)
```

The first thing to do is to get information on the number of rows and columns of the dataset. The `shape` attribute of the dataframe can be used to provide this information:

```
df_titanic.shape

(891, 12)
```

You can see that the dataframe has 891 rows and 12 columns (or attributes). The following snippet can be used to get the names of the columns:

```
# titles of the 12 columns
print (df_titanic.columns.values)

['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

One of the most common problems that data scientists have to deal with is that of missing values. Raw datasets often have missing values in one or more columns. There can be a number of reasons why the values are missing, ranging from human error to data simply being unavailable for that observation. When you load a CSV file into a Pandas dataframe, Pandas uses `NaN` as a marker to signify missing values. There are various ways to find out if a column in a dataframe contains missing values. One way is to use the `info()` function as illustrated in the following snippet:

```
# how many missing values?
df_titanic.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
Survived    891 non-null int64
Pclass      891 non-null int64
```

```
Name        891 non-null object
Sex         891 non-null object
Age         714 non-null float64
SibSp       891 non-null int64
Parch       891 non-null int64
Ticket      891 non-null object
Fare        891 non-null float64
Cabin       204 non-null object
Embarked    889 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 83.5+ KB
```

Looking at the results of the `info()` function, it is clear that most columns have 891 values, whereas three columns—`Age`, `Cabin`, and `Embarked`—have less than 891 values. The use of the `info()` function to detect missing values only works if the value is truly missing in the CSV file, which means Pandas has been able to detect the missing value and substitute it with a `NaN` marker in the dataframe. If, however, the process by which the data was generated used a blank space, or a special character such as `!` to represent a missing value, then Pandas will not automatically interpret these characters to represent missing data.

Another way to get information on missing values is to chain the output of the Pandas `isnull()` function with the `sum()` function. The `isnull()` function, when applied on a dataframe, returns a dataframe of boolean values that has the same dimensions as the original dataframe. Each position in the new dataframe has a value of `True` if the corresponding position in the original dataframe has a value of `None` or `NaN`. The `sum()` function, when applied to the new dataframe of boolean values, will returns a list with the number of values in each column that are `True`. The following snippet shows the result of chaining the `isnull()` and `sum()` functions to obtain the number of missing values in each column of the dataframe:

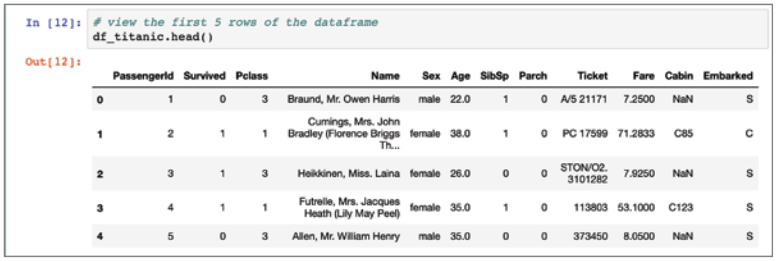
```
# another way to determine the number of missing
# values in a dataframe.
df_titanic.isnull().sum()

PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

It is quite clear from the results that a significant number of `Age` and `Cabin` values are missing. We will look at ways to deal with missing values later in this chapter.

Sometimes the best way to get a feel for the data is to visually inspect the contents of the dataframe. You can use the `head()` function of the dataframe object to view the contents of the first few rows of the dataframe (Figure 2.1).

FIGURE 2.1
The `head()` function displays rows from the beginning of a Pandas dataframe.



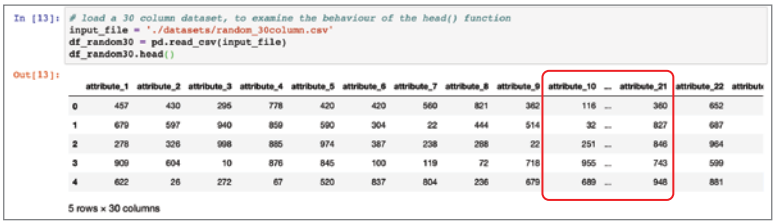
```
In [12]: # view the first 5 rows of the dataframe
         df_titanic.head()
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

If the number of columns is too many for Pandas to fit horizontally into a single line, then Pandas, by default, displays a subset of the columns of the dataframe. The subset consists of a few columns from the left of the dataframe, and a few from the right. Figure 2.2 illustrates the effect of using the `head()` function on a dataframe with 30 columns.

FIGURE 2.2
The `head()` function displays truncated data for large dataframes.



```
In [13]: # load a 30 column dataset, to examine the behaviour of the head() function
         input_file = './datasets/random_30column.csv'
         df_random30 = pd.read_csv(input_file)
         df_random30.head()
```

Out[13]:

	attribute_1	attribute_2	attribute_3	attribute_4	attribute_5	attribute_6	attribute_7	attribute_8	attribute_9	attribute_10	...	attribute_21	attribute_22	attribute
0	457	430	295	778	420	420	560	821	362	116	...	360	652	
1	679	597	940	859	590	304	22	444	514	32	...	827	687	
2	278	326	998	885	974	387	238	268	22	251	...	846	964	
3	909	604	10	876	845	100	119	72	718	955	...	743	599	
4	622	26	272	67	520	837	804	236	679	689	...	948	881	

5 rows × 30 columns

You can change the maximum number of columns that will be displayed by Pandas by setting the `display.max_columns` Pandas property. For example, the following snippet will ensure Pandas displays no more than four columns:

```
pd.set_option('display.max_columns', 4)
df_random30.head()

     attribute_1.   attribute_2    ...    attribute_29.   target
0.   457.           430.           ...    8                 1
1.   679.           597.           ...    253.              1
2.   278.           326.           ...    706.              0
3.   909.           604.           ...    263.              1
4    622.           26.            ...    675.              1

5 rows × 30 columns
```

If you would like all columns to be displayed, set the value of `display.max_columns` to `None`:

```
pd.set_option('display.max_columns', None)
```

Astute readers may have noticed that the PasengerId attribute of the Titanic dataset is a numeric row identifier and does not provide any useful input as far as model building is concerned. Every Pandas dataframe can have an index that contains a unique value for each row of the dataframe. By default, Pandas does not create an index for a dataframe; you can find out if a dataframe has an index by using the following snippet:

```
# Does this dataframe have a named index? If so, what is it?
print (df_titanic.index.name)

None
```

To make the PassengerId attribute the index of the df_titanic dataframe, use the following snippet:

```
df_titanic.set_index("PassengerId", inplace=True)
```

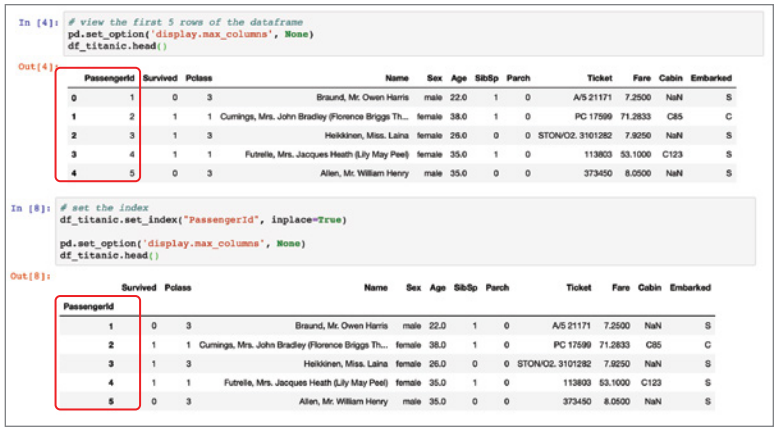
If you were to examine the index of the dataframe after executing the set_index() function, you would see that the PassengerId attribute is now the index:

```
# Does this dataframe have a named index? If so, what is it?
print (df_titanic.index.name)

PasengerId
```

Figure 2.3 shows the results of applying the head() function to the df_titanic dataframe before and after the index has been set up.

FIGURE 2.3
Impact of the set_index function on a dataframe



If you now use the shape attribute of the dataframe to get the number of rows and columns, you will notice that the number of columns is now reported as 11 instead of 12. This is illustrated in the following snippet:

```
# how many rows and columns in the dataframe
# after the index has been set?
df_titanic.shape

(891, 11)
```

You may have noticed that the Survived attribute is one of the 11 remaining attributes in the df_titanic dataframe after PassengerId has been used as the index. During the training process, you need to ensure you do not include the target attribute as one of the input features. There are various means by which you could ensure this, but perhaps the simplest option is to separate the feature variables and the target variables into separate dataframes. The following snippet will extract the Survived attribute from the df_titanic dataframe into a separate dataframe called df_titanic_target, and the 10 feature variables from the df_titanic dataframe into a separate dataframe called df_titanic_features:

```
# extract the target attribute into its own dataframe
df_titanic_target = df_titanic.loc[:,['Survived']]

# create a dataframe that contains the 10 feature variables
df_titanic_features = df_titanic.drop(['Survived'], axis=1)
```

The Survived attribute is a binary attribute in which a value of 1 implies that the individual survived. When the target that your machine learning model is trying to predict is categorical (binary or multi-class), it is useful to know the distribution of values in the training dataset per category. You can get the distribution of target values in this example by using the following snippet:

```
# what is the split between the two classes of the target variable?
df_titanic_target['Survived'].value_counts()

0    549
1    342
Name: Survived, dtype: int64
```

The value_counts() function can be used on numeric columns in addition to categorical columns. However, by default, the value_counts() function will not pick out NaN values. To have the value_counts() function include counts of NaN markers, include the dropna=false parameter as demonstrated in the following snippet:

```
# unique values and counts of categorical attribute 'Embarked'
# includes NaN markers
df_titanic_features['Embarked'].value_counts(dropna=False)

S      644
C      168
Q       77
NaN      2
Name: Embarked, dtype: int64
```

If you prefer a visual representation of the distribution of target values, you can use the following snippet to create a histogram. The histogram is depicted in Figure 2.4.

```
# histogram of target variable
%matplotlib inline
import matplotlib.pyplot as plt
df_titanic_target.hist(figsize=(5,5))
```

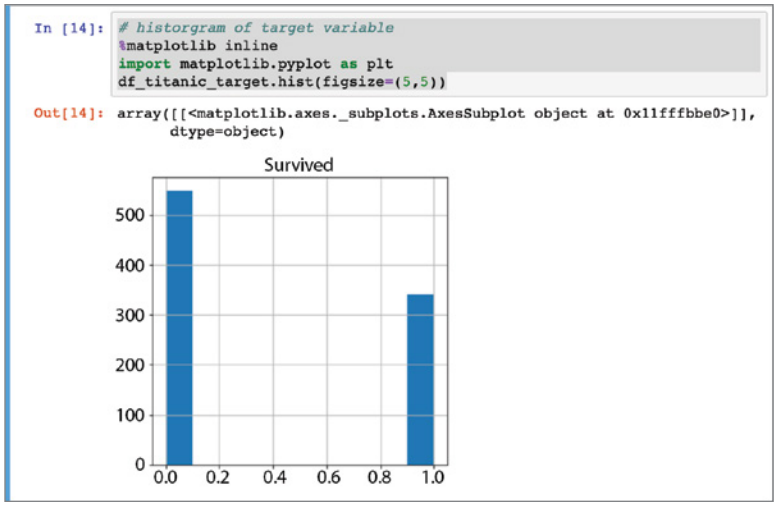


Figure 2.4
Distribution of values for the Survived attribute

In addition to a histogram of the target variables, it is also useful to use histograms to get an overview of the distribution of feature values. The `hist()` function provided by the Pandas dataframe object will only generate histograms for numeric values. The only numerical features in the Titanic dataset are `Age`, `Fare`, `Pclass`, `Parch`, and `SibSp`. The following snippet can be used to generate histograms of numeric features. The resulting feature histograms are depicted in Figure 2.5.

```
#histogram of features
df_titanic_features.hist(figsize=(10,10))
```

If you have a background in statistics, then you will be aware that the appearance of a histogram is influenced by the width of the bins. Data scientists often generate multiple histograms of the same variable with different bin widths to get a better understanding of the distribution of the data. The following snippet can be used to create a histogram of a single numeric attribute and specify the number of equal-width bins along the x-axis. Figure 2.6 depicts the histograms obtained by choosing a number of different bin widths for the same numerical feature.

```
#histogram of single feature - Age
# it is a good idea to try different bin widths to get a better idea of
# the distribution of values.
df_titanic_features.hist(column='Age', figsize=(5,5), bins=2)
```

Since the `value_counts()` function works on both numeric and categorical features, you could generate a histogram of a categorical feature by using the output of the `value_counts()` function. The following snippet demonstrates this approach on the Embarked categorical feature. The resulting histogram is depicted in Figure 2.7.

```
# histogram of categorical attribute 'Embarked'
# computed from the output of the value_counts() function
vc = df_titanic_features['Embarked'].value_counts(dropna=False)
vc.plot(kind='bar')
```

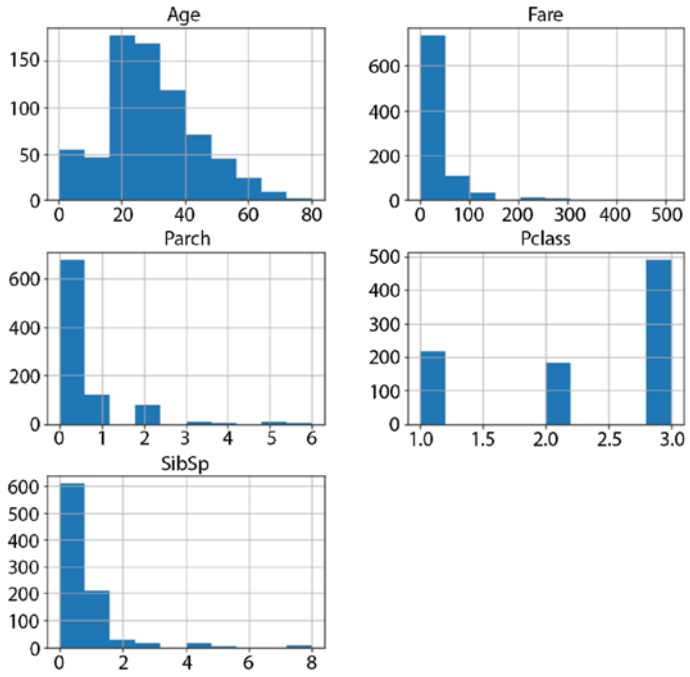


FIGURE 2.5
Histogram of numeric features

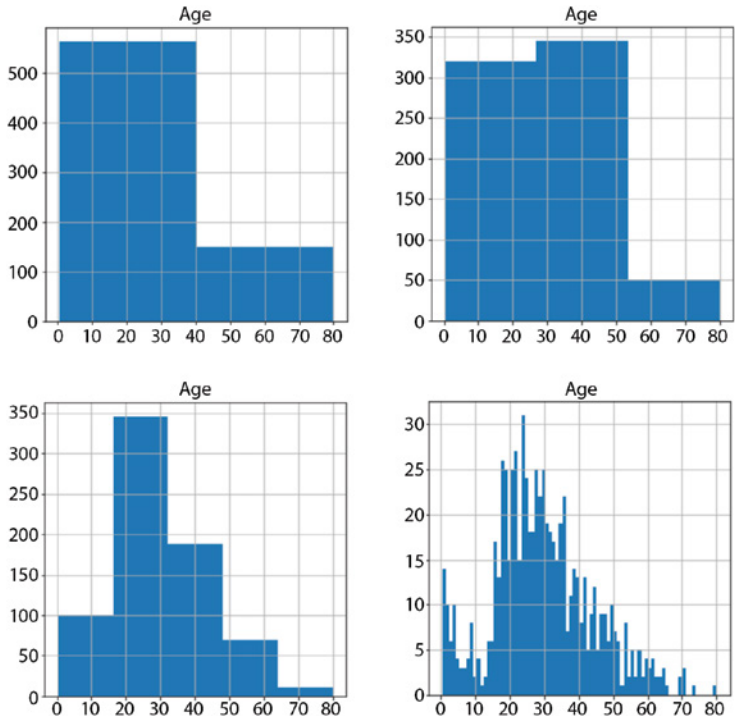
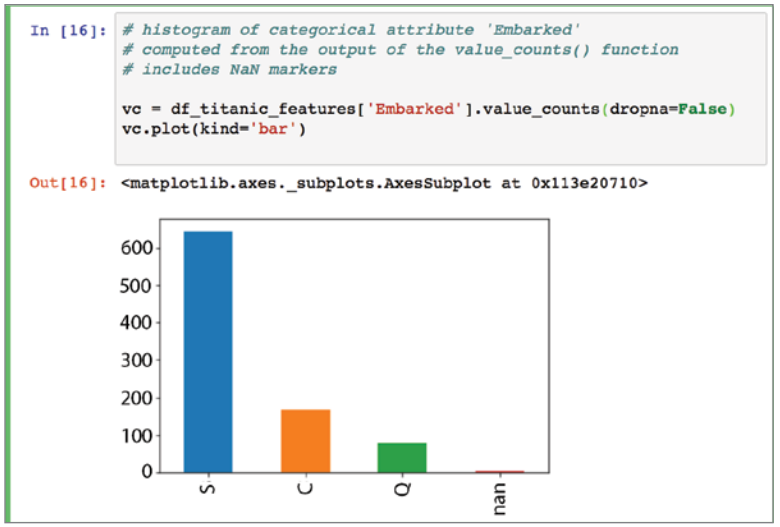


FIGURE 2.6
Histogram of numeric feature "Age" using different bin widths (2, 3, 5, 80)

FIGURE 2.7
Histogram of categorical feature "Embarked"



In addition to information on the distribution of features and target variables, the statistical characteristics of these variables and the correlation between them can provide useful insights into the training data. Pandas provides a `describe()` function that can be used on dataframes to obtain statistical information on the numerical attributes within the dataframe. The following snippet shows the results of the `describe()` function on the `df_titanic_features` dataset:

```
# get statistical characteristics of the data
df_titanic_features.describe()
```

	Pclass	Age.	SibSp.	Parch	Fare
Count.	891.000000	714.000000.	891.000000.	891.000000.	891.000000
Mean.	2.308642.	29.699118.	0.523008.	0.381594.	32.204208
Std.	0.836071.	14.526497.	1.102743.	0.806057.	49.693429
Min.	1.000000.	0.420000.	0.000000.	0.000000.	0.000000
25%.	2.000000	20.125000.	0.000000.	0.000000.	7.910400
50%.	3.000000.	28.000000.	0.000000.	0.000000.	14.454200
75%.	3.000000.	38.000000	1.000000.	0.000000.	31.000000
Max.	3.000000.	80.000000.	8.000000.	6.000000.	512.329200

Information provided by the `describe()` function includes the minimum value, maximum value, mean value, standard deviation, and the quartiles of each numerical feature. A quartile is a value below which a certain percent of observations can be found. For example, the first quartile is the value below which 25% of the observations can be found. The first quartile of the `Age` feature is 20.12, which means that 25% of the people captured by the dataset have ages less than 20 years. Information on quartiles and statistical characteristics of a feature is often represented using a box plot. Box plots are covered in Chapter 4; however, you can use the `boxplot()` function of the dataframe to create a box plot of all numeric features. The following snippet demonstrates the use of the `boxplot()` function. The resulting box plot is depicted in Figure 2.8.

```
# create a box plot of numeric features.
df_titanic_features.boxplot(figsize=(10,6))
```

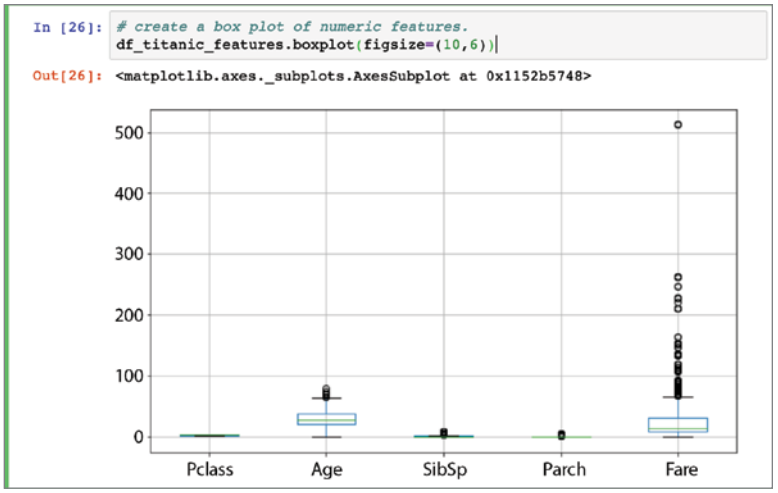
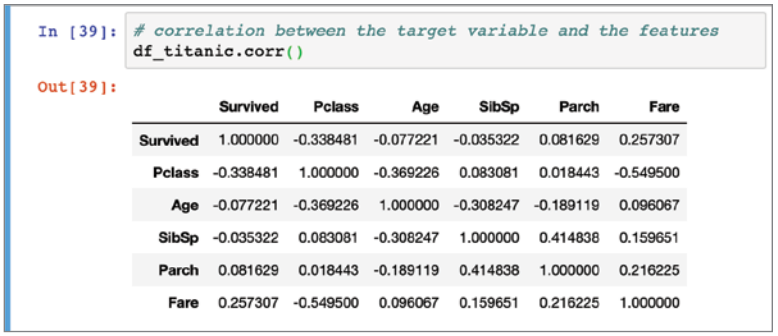


FIGURE 2.8
Box plot of numeric features

Information on the correlation between input features and the target can be helpful in picking out the best features from the data to use for model building and predictions. Information on the correlation between the features themselves can be helpful in reducing the number of features and the general risk of overfitting. Pandas provides a `corr()` function that can be used to compute Pearson's correlation coefficient between the columns of a dataframe. The results of applying the `corr()` function on the `df_titanic` dataframe is depicted in Figure 2.9.



	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

FIGURE 2.9
Linear correlation between numeric columns

It is important to note that Pearson's correlation coefficient will only detect linear correlation between variables. The `corr()` function allows you to choose from standard correlation coefficients such as Pearson, Kendall, and Spearman. You can find more information at `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html`.

The following snippet lists the correlation between the numeric features and the target variable, sorted by descending value:

```
# what features show the strongest correlation with the target variable?
corr_matrix = df_titanic.corr()
corr_matrix['Survived'].sort_values(ascending=False)

Survived    1.000000
Fare        0.257307
Parch       0.081629
SibSp      -0.035322
Age        -0.077221
Pclass     -0.338481
Name: Survived, dtype: float64
```

Computing correlation coefficients between pairs of attributes is not the only way to get information on the correlation between features. You can also create scatter plots between pairs of features to visualize their relationship. The following snippet uses the Pandas `scatter_matrix()` function to create scatter plots of all numeric features with each other. The resulting scatter plot is depicted in Figure 2.10.

```
# visualize relationship between features using a
# matrix of scatter plots.
from pandas.plotting import scatter_matrix
scatter_matrix(df_titanic, figsize=(12,12))
```

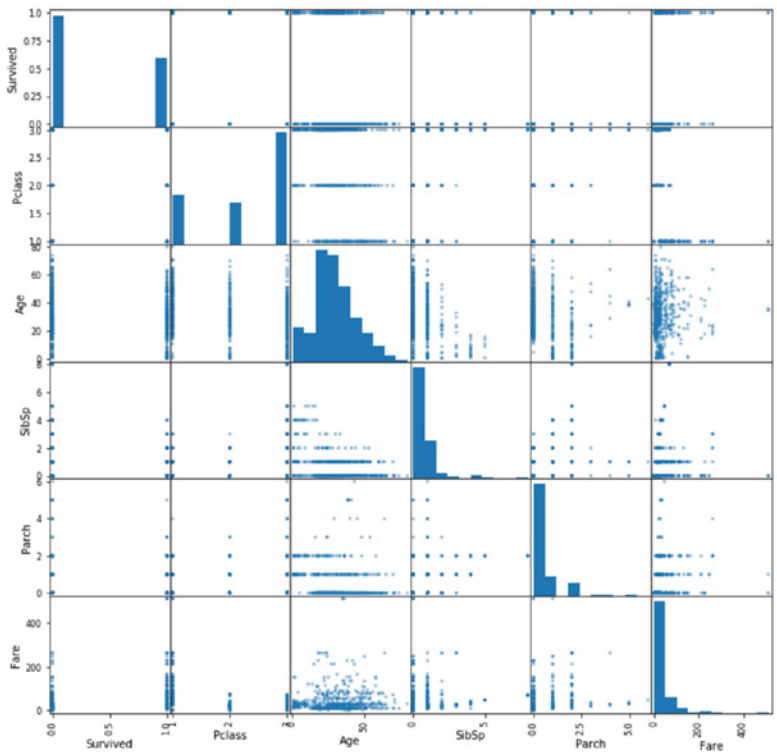


FIGURE 2.10
Matrix of scatter plots between pairs of numeric attributes

Handling Missing Values

In the previous section, you learned of techniques that can be used to explore the data. While exploring the Titanic dataset, you learned that the `Age`, `Cabin`, and `Embarked` classes have missing values. `Age` is a numeric feature, and we can use a box plot to get a quick overview of the statistical characteristics of the values that make up this feature using the following snippet. Figure 2.11 depicts a box plot of the `Age` feature variable.

```
# boxplot of 'Age'
%matplotlib inline
import matplotlib.pyplot as plt
df_titanic_features.boxplot(column='Age', figsize=(7,7))
```

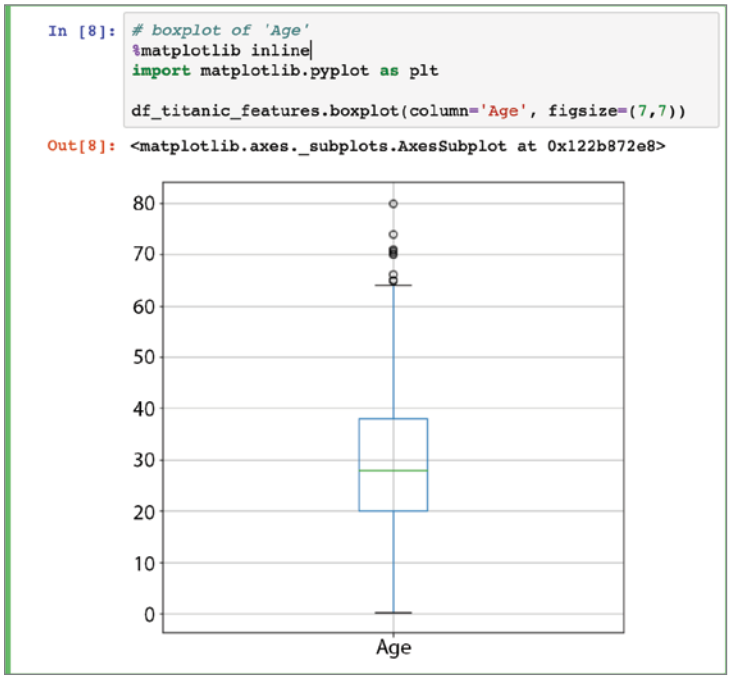


FIGURE 2.11
Box plot of the Age feature variable

The median value of the age, according to the box plot, is just under 30. Pandas provides the `fillna()` function that can be used to replace missing values with a new value. The following snippet uses the `fillna()` function to replace the missing values of the `Age` attribute with the median value:

```
# fill missing values with the median
median_age = df_titanic_features['Age'].median()
print (median_age)
28.0

df_titanic_features["Age"].fillna(median_age, inplace=True)
```

NOTE Although it is not demonstrated in this section, you must ensure that any feature engineering or imputation that is carried out on the training data is also carried out on the test data.

The `Embarked` attribute is categorical, and since the number of missing values is small (just 2), a reasonable approach is to substitute the missing values with the most frequently occurring value in the `Embarked` column. The following snippet uses the `fillna()` function to achieve this:

```
# fill missing values of the Embarked attribute
# with the most common value in the column
embarked_value_counts = df_titanic_features['Embarked'].value_counts(dropna=True)
most_common_value = embarked_value_counts.index[0]

print (most_common_value)
S

df_titanic_features["Embarked"].fillna(most_common_value, inplace=True)
```

The `Cabin` attribute is also categorical but has a very large number of missing values (687). Using the same strategy that was used to impute missing values for the `Embarked` attribute will not work in this case as you will create a significant bias in the data. The best approach in this situation is to create a new boolean feature `CabinIsKnown`, which will have a value of `True` if the `Cabin` attribute is known, and `False` otherwise. You may be tempted to use the integer 1 to signify known cabin values and 0 to signify missing cabin values, but if you were to do this you would create an unintentional order in the data (1 being greater than 0), and this could influence the output of some models. The following snippet creates a new column called `CabinIsKnown` and drops the original `Cabin` column from the dataframe:

```
# create a boolean feature 'CabinIsKnown'
# which will have True if the Cabin column
# does not have missing data
df_titanic_features['CabinIsKnown'] = ~df_titanic_features.Cabin.isnull()

# drop the Cabin column from the dataframe
df_titanic_features.drop(['Cabin'], axis=1, inplace=True)
```

With the changes described in this section, you have imputed missing values where possible and created a new column in the dataframe. There should be no missing values in the dataframe, a new column called `CabinIsKnown` should be visible in the dataframe, and the `Cabin` column should have been deleted from the dataframe. All of this can be validated by executing the following snippet:

```
# display the columns of the dataframe.
print (df_titanic_features.columns.values)

# display number of missing values in the columns
df_titanic_features.isnull().sum()

['Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch' 'Ticket' 'Fare' 'Embarked'
 'CabinIsKnown']
Out[10]:
```

```
Pclass          0
Name            0
Sex             0
Age             0
SibSp           0
Parch           0
Ticket          0
Fare            0
Embarked        0
CabinIsKnown    0
dtype: int64
```

Creating New Features

If you observe the descriptions of the columns of the Titanic dataset, you will come across the `SibSp` and `Parch` columns. From the description of the dataset:

- *SibSp*: A numeric variable that indicates the number of siblings/spouses traveling together.
- *Parch*: A numeric variable that indicates the number of parents and children traveling together.

It may make sense to combine these values into a single numeric value that represents the size of the family traveling together. It is not possible to tell at this stage if the model will perform better with this additional synthesized feature, but having this new feature in the data will give you more options when it comes to building and evaluating models. The following snippet creates a new attribute in the dataframe called `FamilySize`, which is computed as the arithmetic sum of the `SibSp` and `Parch` attributes:

```
# create a numeric feature called FamilySize that is
# the sum of the SibSp and Parch features.
df_titanic_features['FamilySize'] = df_titanic_features.SibSp + df_titanic_
features.Parch
```

NOTE Although it is not demonstrated in this section, you must ensure that any feature engineering that is carried out on the training data is also carried out on the test data.

The `Age` and `Fare` features are numeric and take on a range of values; it may be useful to bin the value of these features and create categorical features. During model building you may discover that the categorical (binned) values of `Age` and `Fare` provide better results. To create a new categorical feature called `AgeCategory`, you can use the Pandas `cut()` function as demonstrated in the following snippet:

```
# generate new categorical feature AgeCategory
bins_age = [0,20,30,40,50,150]
labels_age = ['<20','20-30','30-40','40-50','>50']
```

```
df_titanic_features['AgeCategory'] = pd.cut(df_titanic_features.Age,
                                            bins=bins_age,
                                            labels=labels_age,
                                            include_lowest=True)
```

Figure 2.12 depicts the output of the `head()` function on the `df_titanic_features` dataframe after the `AgeCategory` feature has been created.

FIGURE 2.12
Dataframe with engineered feature AgeCategory

```
In [15]: # examine the first 10 rows of the dataset
         df_titanic_features.head(10)
```

Out[15]:

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	CabinIsKnown	FamilySize	AgeCategory
1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	False	1	20-30
2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	True	1	30-40
3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	False	0	20-30
4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	True	1	30-40
5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	False	0	30-40
6	3	Moran, Mr. James	male	28.0	0	0	330877	8.4583	Q	False	0	20-30
7	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	S	True	0	>50
8	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	S	False	4	<20
9	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	S	False	2	20-30
10	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	C	False	1	<20

Downloads

The `cut()` function has several parameters. In this example, the `bins` parameter contains a sequence of numbers that define the edges of the bins; the lowest and highest values are deliberately chosen to be outside the range of values observed in the `Age` feature. The `labels` parameter contains a list of strings that serve as the labels of the bins (and the values of the categorical feature that will be generated as a result of executing the `cut()` function). The `include_lowest` parameter is set to `True` to indicate that the first interval is left-inclusive. You can find information on the full list of parameters for the `cut()` function at `https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.cut.html`.

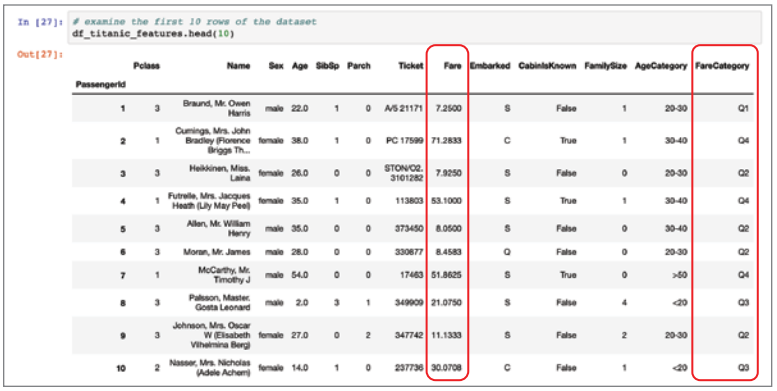
There is no set formula to determine the correct number of bins and the widths of the bins. During the model-building process, you may find yourself experimenting with different binning strategies and pick the strategy that results in the best-performing model. If you want to split a continuous numeric variable into a categorical variable by using the quantiles as bin boundaries, you can use the Pandas `qcut()` function. The following snippet uses the `qcut()` function to create a new categorical feature called `FareCategory` using the quartiles as bin boundaries:

```
# generate new categorical feature FareCategory
df_titanic_features['FareCategory'] = pd.qcut(df_titanic_features.Fare,
                                              q=4,
                                              labels=['Q1', 'Q2', 'Q3', 'Q4'])
```

The second parameter, `q=4`, indicates that you want to use the quartiles as bin boundaries. Information on the `qcut()` function is available at `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.qcut.html`.

Figure 2.13 depicts the output of the `head()` function on the `df_titanic_features` dataframe after the `FareCategory` feature has been created.

FIGURE 2.13
Dataframe with engineered feature FareCategory



```
In [27]: # examine the first 10 rows of the dataset
         df_titanic_features.head(10)
```

Out[27]:

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	CabinIsKnown	FamilySize	AgeCategory	FareCategory
1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	False	1	20-30	Q1
2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	True	1	30-40	Q4
3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	False	0	20-30	Q2
4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	True	1	30-40	Q4
5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	False	0	30-40	Q2
6	3	Moran, Mr. James	male	28.0	0	0	330877	8.4583	Q	False	0	20-30	Q2
7	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	S	True	0	>50	Q4
8	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	S	False	4	<20	Q3
9	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	S	False	2	20-30	Q2
10	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	C	False	1	<20	Q3

Transforming Numeric Features

After having created the categorical features `AgeCategory` and `FareCategory` in the previous section, you may wish to drop the original `Age` and `Fare` attributes from the dataset. The decision to drop the original numerical values will largely depend on the type of model you are going to build.

When building a model with continuous numeric variables, you may need to transform numeric attributes. Several machine learning models converge faster and work better when the values of numeric attributes are small and have a distribution that is close to a standard normal distribution with mean 0 and variance 1.

Normalization and standardization are the two most common types of transformations performed on numerical attributes. The result of normalizing a feature is that the values of the feature will be scaled to fall within 0 and 1. The result of standardizing a feature is that the distribution of the new values will have a mean of 0 and a standard deviation of 1, but the range of the standardized values is not guaranteed to be between 0 and 1. Standardization is used when the model you want to build assumes the feature variables have a Gaussian distribution. Normalization is often used with neural network models, which require inputs to lie within the range [0, 1].

Scikit-learn provides a number of classes to assist in scaling numeric attributes. The `MinMaxScaler` class is commonly used for normalizing features, and the `StandardScaler` class is commonly used for standardization. The following snippet creates two new columns, `NormalizedAge` and `StandardizedAge`, in the `df_titanic_features` dataframe. Figure 2.14 compares the histogram of the `Age`, `NormalizedAge`, and `StandardizedAge` features:

```
# generate new feature NormalizedAge using MinMaxScaler
from sklearn import preprocessing

minmax_scaler = preprocessing.MinMaxScaler()
ndNormalizedAge = minmax_scaler.fit_transform(df_titanic_
features[['Age']].values)
df_titanic_features['NormalizedAge']  = pd.DataFrame(ndNormalizedAge)

# generate new feature StandardizedAge using StandardScaler
standard_scaler = preprocessing.StandardScaler()
```

```
ndStandardizedAge = standard_scaler.fit_transform(df_titanic_
features[['Age']].values)
df_titanic_features['StandardizedAge']  = pd.DataFrame(ndStandardizedAge)

# histogram of Age, NormalizedAge, StandardizedAge
df_titanic_features[['Age', 'NormalizedAge', 'StandardizedAge']].
hist(figsize=(10,10), bins=5)
```

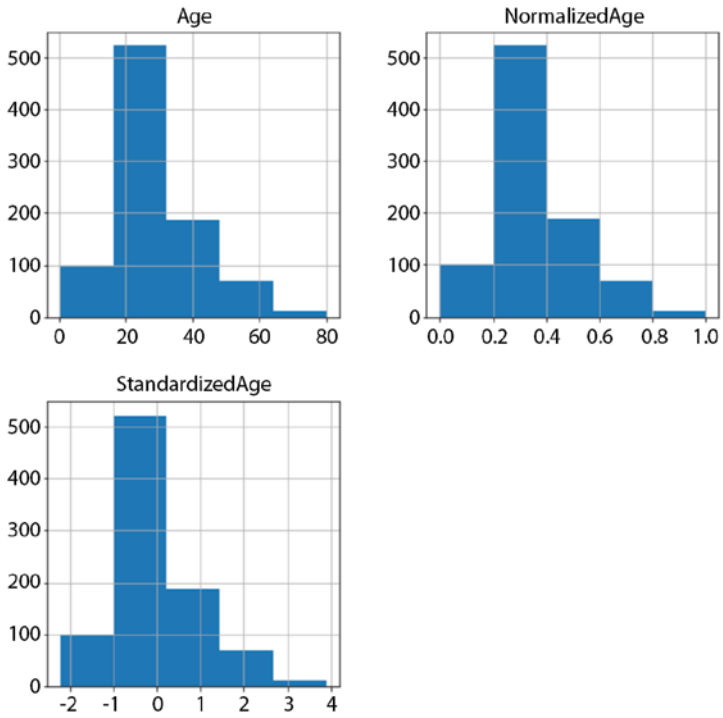


Figure 2.14
Histogram of Age, NormalizedAge, and StandardizedAge

One-Hot Encoding Categorical Features

The final topic that we will look at in this chapter will be converting categorical features into numeric features using *one-hot encoding*. You may be wondering why you would want to convert categorical features to numeric, especially since a previous section in this chapter has discussed techniques to do the opposite—converting numeric features into categorical.

Not all machine learning algorithms can deal with categorical data. One-hot encoding is a technique that converts a categorical feature into a number of binary numeric features, one per category. For example, linear regression and logistic regression are only capable of using numeric features. Algorithms like XGBoost and random forests are capable of using categorical features without any problems.

Pandas provides the `get_dummies()` function to help with one-hot encoding. The following snippet will convert the categorical features `Sex`, `Embarked`, `CabinIsKnown`, `AgeCategory`, and `FareCategory` into binary numeric features and list the columns of the dataframe:

```
# use one-hot encoding to convert categorical attributes
# into binary numeric attributes
df_titanic_features = pd.get_dummies(df_titanic_features,
columns=['Sex','Embarked','CabinIsKnown','AgeCategory','FareCategory'])

# display the columns of the dataframe.
print (df_titanic_features.columns.values)

['Pclass' 'Name' 'Age' 'SibSp' 'Parch' 'Ticket' 'Fare' 'FamilySize'
 'NormalizedAge' 'StandardizedAge' 'Sex_female' 'Sex_male' 'Embarked_C'
 'Embarked_Q' 'Embarked_S' 'CabinIsKnown_False' 'CabinIsKnown_True'
 'AgeCategory_<20' 'AgeCategory_20-30' 'AgeCategory_30-40'
 'AgeCategory_40-50' 'AgeCategory_>50' 'FareCategory_Q1' 'FareCategory_Q2'
 'FareCategory_Q3' 'FareCategory_Q4']
```

As can be seen from the preceding snippet, the original categorical attributes are no longer present in the `df_titanic_features` dataframe; however, a number of new columns have been added. To understand how Pandas has created the additional columns, consider the `Sex` categorical attribute. This attribute has two values: `male`, and `female`. To convert this categorical attribute into a binary numeric attribute, Pandas has created two new columns in the dataframe called `Sex_male` and `Sex_female`. Other categorical attributes such as `Embarked`, `CabinIsKnown`, etc. have been processed using a similar approach. The following snippet lists the values of the `Sex_male` and `Sex_female` columns for the first five rows of the dataframe:

```
df_titanic_features[['Sex_male', 'Sex_female']].head()

                Sex_male        Sex_female
PassengerId
1                 1                 0
2                 0                 1
3                 0                 1
4                 0                 1
5                 1                 0
```

Astute readers may notice that since the values taken by the `Sex` attribute in the original `df_titanic_features` dataset is either `male` or `female`, you don't need both `Sex_male` and `Sex_female` attributes because you can infer one from the other. These two attributes have a very strong negative correlation of –1.0 and you should not use both of them as inputs to a machine learning model. The following snippet demonstrates the strong negative correlation between the two attributes:

```
# strong negative correlation between Sex_male and Sex_female.
# one of these can be dropped.
corr_matrix = df_titanic_features[['Sex_male', 'Sex_female']].corr()
print(corr_matrix)
```

```
             Sex_male  Sex_female
Sex_male         1.0        -1.0
Sex_female      -1.0         1.0
```

The situation is similar with the `CabinIsKnown_False` and `CabinIsKnown_True` features. The following snippet drops the `Sex_female` and `CabinIsKnown_False` attributes along with non-numeric attributes `Name` and `Ticket` to arrive at a dataframe that contains only numeric attributes:

```
# drop the Name, Ticket, Sex_female, CabinIsKnown_False features
# to get a dataframe that can be used for linear or logistic regression
df_titanic_features_numeric = df_titanic_features.drop(['Name', 'Ticket',
'Sex_female', 'CabinIsKnown_False'], axis=1)
```

If you compute the correlation between the target attribute `Survived` and these numeric features, you will see a significantly better correlation than achieved earlier in this chapter prior to feature engineering. The following snippet demonstrates this:

```
# what features show the strongest correlation with the target variable?
corr_matrix = df_temporary.corr()
corr_matrix['Survived'].sort_values(ascending=False)

Survived             1.000000
CabinIsKnown_True    0.316912
Fare                 0.257307
FareCategory_Q4      0.233638
Embarked_C           0.168240
FareCategory_Q3      0.084239
Parch                0.081629
AgeCategory_<20      0.076565
AgeCategory_30-40    0.057867
FamilySize           0.016639
Embarked_Q           0.003650
AgeCategory_40-50   -0.000079
NormalizedAge       -0.001654
StandardizedAge     -0.001654
AgeCategory_>50     -0.022932
SibSp               -0.035322
Age                 -0.064910
AgeCategory_20-30   -0.093689
FareCategory_Q2     -0.095648
Embarked_S          -0.149683
FareCategory_Q1     -0.221610
Pclass              -0.338481
Sex_male            -0.543351
Name: Survived, dtype: float64
```

Note the particularly strong negative correlation between the chances of survival and `Sex_male`, and the strong positive and negative correlation with `FareCategory_Q4` and `FareCategory_Q1`, respectively.

Feature engineering is the most laborious and time-consuming aspect of data science, and there is no set formula that can be applied to a given situation. In this chapter you have learned some of the techniques that can be used to explore data, impute missing values, and engineer features.

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from `www.wiley.com/go/machinelearningawscloud` or from GitHub using the following URL:

```
https://github.com/asmtechnology/awsmlbook-chapter2.git
```

Summary

- A number of sources of datasets can be used to create machine learning models. Some of the popular sources are the UCI machine learning repository, `Kaggle.com`, and AWS public datasets.
- Scikit-learn includes subsampled versions of popular datasets that are often used by beginners.
- Pandas provides the `read_csv()` function that allows you to load a CSV file into a dataframe.
- You can use the Pandas `isnull()` and `sum()` functions together to obtain the number of missing values in each column of a dataframe.
- The `hist()` function exposed by the Pandas dataframe object will only generate histograms for numeric values.
- Pandas provides a `describe()` function that can be used on dataframes to obtain statistical information on the numerical attributes within the dataframe.
- Pandas provides a `corr()` function that can be used to compute Pearson's correlation coefficient between the columns of a dataframe.
- You can also create scatter plots between pairs of features to visualize their relationship.
- Pandas provides the `fillna()` function that can be used to replace missing values with a new value.

Chapter 3

Data Visualization with Python

WHAT'S IN THIS CHAPTER

- Introduction to Matplotlib
- Learn to create histograms, bar charts, and pie charts
- Learn to create box plots and scatter plots
- Learn to use Pandas plotting functions

In the previous chapter, you learned about techniques to explore data and perform feature engineering with NumPy, Pandas, and Scikit-learn. In this chapter you will learn to use Matplotlib to visualize data. Data visualization helps to understand the characteristics and relationships between the features during the data exploration phase but becomes particularly important when you are dealing with very large datasets that have several hundreds of features.

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from `www.wiley.com/go/machinelearningawscloud` or from GitHub using the following URL:

`https://github.com/asmtechnology/awsmlbook-chapter3.git`

Introducing Matplotlib

Matplotlib is a plotting library for Python that offers functionality to generate numerous types of plots and the ability to customize these plots. It was created by John Hunter to provide Python users with a plotting library with capabilities similar to MATLAB. MATLAB is the standard for data visualization in the scientific community. The Matplotlib package has an extensive codebase and was designed to address the needs of a variety of users and provide capabilities at different levels of abstraction. Some users may want to simply present Matplotlib with some data in an array and ask for a specific type of plot (such as a scatter plot) to be created using as few commands as possible; these users may not want to control detailed attributes of the plot (such as positioning, scaling, line style, color). On the other hand, some users may want the ability to control every single attribute of a plot, down to the level of individual pixels.

For most common plotting tasks, you will use the `pyplot` module within Matplotlib, which provides the highest level of abstraction. The `pyplot` module implements a functional interface, powered by a state-machine design—you use functions to set up attributes of the plotting engine

such as colors and fonts, and these apply to all subsequent plots until you issue commands to change them. Beneath the `pyplot` level of abstraction is the object-oriented interface, which offers more flexibility.

A Note about Seaborn

Seaborn is another Python plotting package that builds on top of Matplotlib. Seaborn provides additional plot types and some individuals may find the figures it generates to be more visually appealing. Matplotlib and Seaborn are complementary packages; the one you end up using in a given situation can come down to various factors including aesthetics, personal preferences, or the ease of making a specific type of plot with one package over the other. Seaborn is not covered in this chapter. If you are interested in learning about Seaborn, visit `https://seaborn.pydata.org`.

The conventional alias for the `pyplot` module is `plt`, the alias for the Matplotlib package is `mpl`, and the alias for the Seaborn package is `sns`. The following statements demonstrate how to import Matplotlib and Seaborn in a Python project:

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
```

The reason to import both the `pyplot` submodule and the `matplotlib` package is to allow you to use functions from both the higher level of abstraction provided by `pyplot` and the lower-level object-oriented API exposed by Matplotlib. If you are using Matplotlib in a Jupyter Notebook, you must also add the `%matplotlib inline` statement before drawing any figures to ensure that plots render within the cells of the notebook.

Before looking at the components of a Matplotlib figure, let's examine the code to plot a simple curve using the `pyplot` module. The following snippet plots the function $y_1=4x^2 + 2x + 2$ and the function $y_2 = 3x + 4$ for values of x between 1 and 7. Figure 3.1 depicts the figure generated by these statements.

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl

import numpy as np
import pandas as pd

# prepare x values
x = np.linspace(1, 7, 10)

# prepare y1 and y2
y1 = 4*x*x + 2*x + 2
y2 = 3*x + 4

# create a new figure
plt.figure(figsize=(7,7))
```

```
# plot y1 = 4*x*x + 2*x + 2
plt.plot(x, y1)

#y2 = 3*x + 4
plt.plot(x, y2)

# set up axis labels
plt.xlabel('X values')
plt.ylabel('Y values')

# show the plot.
plt.show()
```

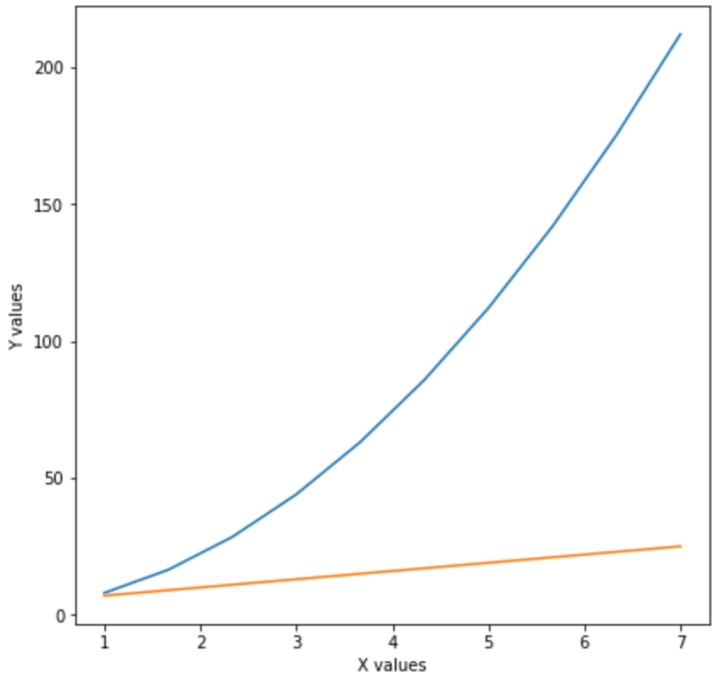


FIGURE 3.1
Plotting two curves using Matplotlib

The code snippet uses NumPy functions to generate ndarrays `x`, `y1`, and `y2`. A Matplotlib figure object with dimensions 7 × 7 inches is then created using the statement `plt.figure(figsize=(7,7))`. The first curve $y_1=4x^2 + 2x + 2$ is plotted using the `plt.plot(x, y1)` statement; the inputs to the plot function are 2D coordinates of the points that need to be plotted. The `plt.xlabel('X values')` and `plt.ylabel('Y values')` statements are for aesthetic purposes and add labels along the x- and y-axes of the plot, respectively. Finally, the `plt.show()` statement is used to render the figure.

Components of a Plot

Regardless of the type of plot you make with Matplotlib or whether you use the high-level `pyplot` interface or the lower-level object-oriented interface, there are common components and terminology associated with plots. This section presents an overview of these concepts. Figure 3.2 depicts the parts of a plot.

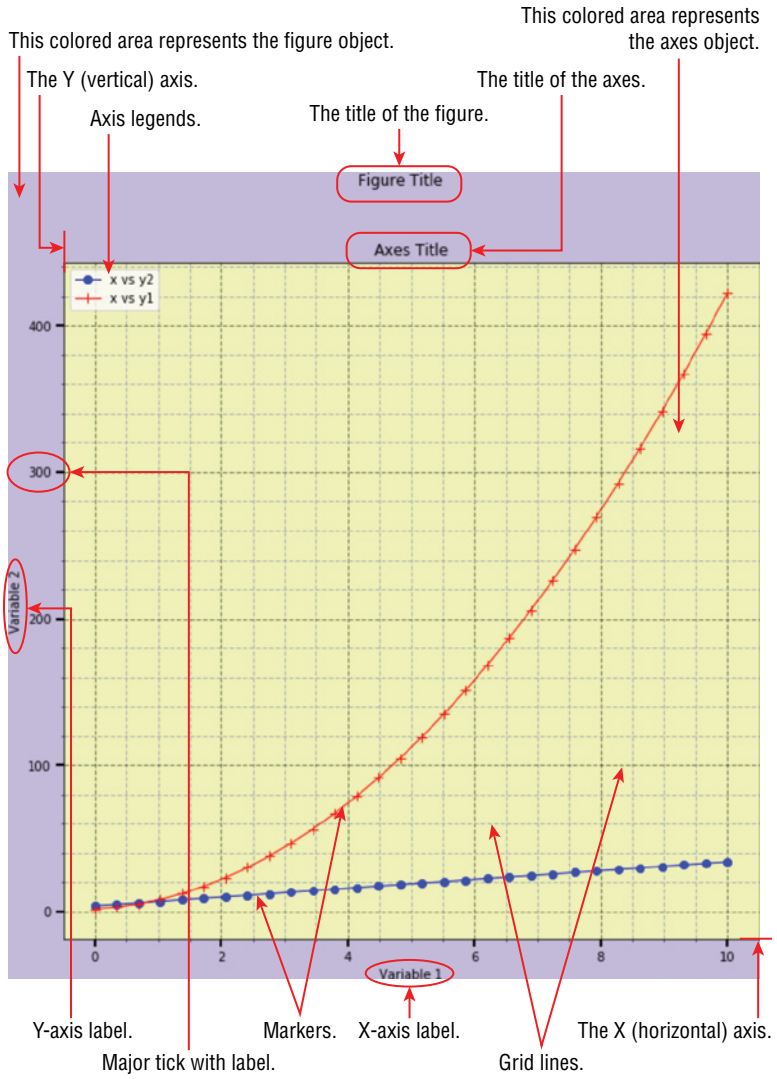


FIGURE 3.2
Components of a Matplotlib plot

Figure

A figure object can be thought of as the entire diagram, with all the lines and text. You can think of it as the container; everything you plot using Matplotlib must belong to a figure. Figures are generally created using a `pyplot` function, even if you then want to manipulate the figure with the lower-level object-oriented API. Use the following `pyplot` command to create an empty figure:

```
plt.figure()
```

If you want the figure to have specific dimensions, you can provide a tuple with the x-axis and y-axis dimensions in inches:

```
plt.figure(figsize=(10,5))
```

Besides the `figsize` attribute, the `pyplot` figure function can accept various other attributes that you can use to customize aspects of the figure at the point of creation. You can find out more about these attributes at `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html`.

The `pyplot` figure function creates a figure and makes it the active figure on which subsequent drawing operations will have effect. The figure is not displayed until there is something drawn to it. If you want to use the object-oriented API to control aspects of the figure after it has been created, you need to store a reference to the figure object in your code and then use the object-oriented API with this reference. The following code snippet creates a figure using the `pyplot figure()` function and then uses the object-oriented API to change the background color of the figure:

```
figure1 = plt.figure(figsize=(5,3))
figure1.suptitle('Figure Title')
figure1.set_facecolor('#c3badc')
```

Axes

An axes object is what you would normally consider as a plot. It is the actual graph with various characteristics commonly associated with plots, such as data points, markers, colors, scales, etc. A figure object can contain multiple axes, which in effect are multiple subplots within a larger diagram. The following snippet uses `pyplot` to create a figure with a 2 × 2 grid of axes (subplots), and stores references to both the figure and the axes objects. Each axes object has its own title, which is different from the figure title. The object-oriented API is used to set the title for the figure and the four subplots within the figure. Figure 3.3 depicts the figure generated by executing the code snippet.

```
# create a figure with a 2 x 2 grid of axes objects
figure, axes_list  = plt.subplots(2,2, figsize=(9,9))

# title for the figure object
figure.suptitle('This is the title of the figure')

# title for each axis
axes_list[0,0].set_title('Subplot 0')
axes_list[0,1].set_title('Subplot 1')
axes_list[1,0].set_title('Subplot 2')
axes_list[1,1].set_title('Subplot 3')
```

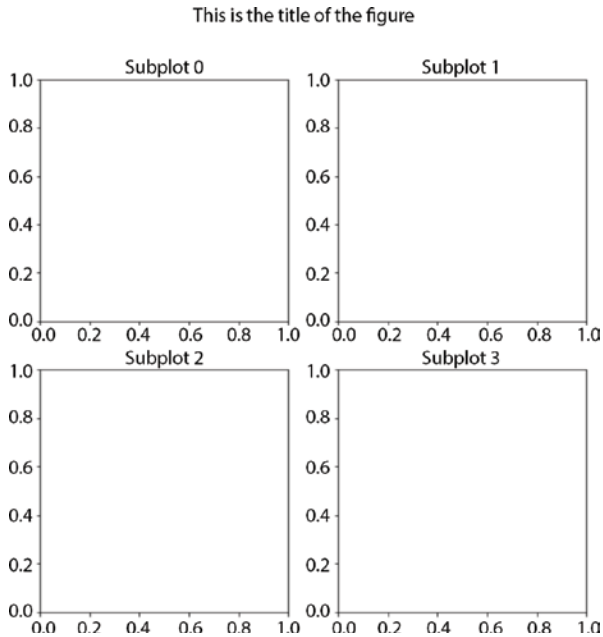


FIGURE 3.3
A figure object with four axes objects

The Matplotlib `figure` and `axes` classes are the primary entry points for working with the lower-level object-oriented API.

Axis

The axis object represents a dimension within a subplot. Two-dimensional plots have two axis objects: one for the horizontal direction and the other for the vertical direction. The `axes` class, which is part of the object-oriented interface, provides several methods to modify attributes of the underlying x- and y-axis.

Axis Labels

The axis label is displayed beneath (or beside) each axis of the plot. Axis labels can be configured by calling the `set_xlabel()` and `set_ylabel()` methods on an axes object. The following snippet demonstrates the use of these methods:

```
figure, axes  = plt.subplots(figsize=(10,10))
axes.set_xlabel('Variable 1')
axes.set_ylabel('Variable 2')
```

If you do not want to use the object-oriented API exposed by the axes object, you can use the `xlabel()` and `ylabel()` functions from the `pyplot` high-level interface that set the axis labels for the active plot. The use of these functions is demonstrated in the following snippet:

```
plt.figure(figsize=(7,7))
plt.xlabel('X values')
plt.ylabel('Y values')
```

Grids

A grid is a set of horizontal and vertical lines inside the plot area that helps in reading values. The axes object provides a method called `grid()` that can be used to customize the appearance of the grid. You can find out more about the `grid()` method of the `axes` class at `https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.grid.html`. The `pyplot` module also provides a `grid()` function that is identical to the similarly named method of the `axes` class.

The following snippet demonstrates the use of the `grid()` method of the `axes` class:

```
figure, axes  = plt.subplots(figsize=(10,10))
axes.grid(b=True, which='both', linestyle='--', linewidth=1)
```

Figure 3.4 depicts two plots side by side, one with a grid and one without.

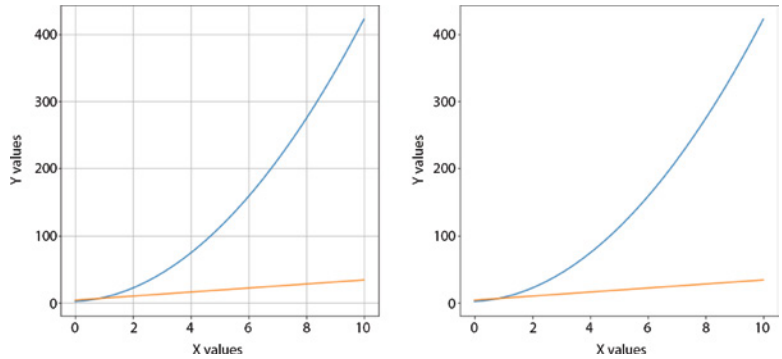


FIGURE 3.4
Comparison of plots with and without grids

Title

The title is a string that is displayed on top of a plot. Both the figure object and the axes objects within the figure can have titles. Titles are displayed, by default, at the top of the figure or axes object, center aligned. The figure title can be changed use the `suptitle()` method of the `figure` class, or the `suptitle()` function of the `pyplot` module. These functions are identical to each other in syntax and function. You can find more information on the `pyplot suptitle()` function at `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.suptitle.html#matplotlib.pyplot.suptitle`.

The following snippet demonstrates the use of the `suptitle()` method of the `figure` class, and the `suptitle()` function of the `pyplot` module:

```
# create a figure with one axes object.
# Call the suptitle() method on the figure instance.
figure, axes  = plt.subplots(figsize=(10,10))
figure.suptitle('Figure Title')

# create a new figure with one axes object
# use the pyplot suptitle() function
plt.figure(figsize=(7,7))
plt.suptitle('Figure Title')
```

The title of the axes object can be changed by calling the `set_title()` method on the axes object. You can find more information on using the `set_title()` method at `https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.set_title.html#matplotlib.axes.Axes.set_title`.

The `pyplot` module provides a convenience function called `title()`, which operates on the active axes object and has the same signature as the `set_title()` method of the `axes` class. The following snippet demonstrates the use of the `set_title()` method of the `axes` class and the `title()` function of the `pyplot` module:

```
# create a figure with one axes object.
# Call the set_title() method on the axes instance.
figure, axes  = plt.subplots(figsize=(10,10))
axes.suptitle(Axes Title')

# create a new figure with one axes object
# use the pyplot suptitle() function
plt.figure(figsize=(7,7))
plt.title(Axes Title')
```

Common Plots

In the previous section you learned the aspects of a typical Matplotlib plot, and that there are often different ways to configure a plot. The high-level `pyplot` API provides a functional interface and the lower-level Matplotlib API operates using an object-oriented interface. In this section, you will learn to create common types of plots using Matplotlib.

Histograms

A histogram is commonly used to visualize the distribution of a numeric variable. Histograms are not applicable when dealing with a categorical variable. The following snippet uses functions from the `pyplot` module to generate a histogram of the `Age` attribute of the popular Titanic dataset. The resulting figure is depicted in Figure 3.5.

```
import numpy as np
import pandas as pd

# load the contents of a file into a pandas Dataframe
input_file = './datasets/titanic_dataset/original/train.csv'
df_titanic = pd.read_csv(input_file)

# set the index
df_titanic.set_index("PassengerId", inplace=True)

# use pyplot functions to plot a histogram of the 'Age' attribute
fig = plt.figure(figsize=(7,7))
plt.xlabel('Passenger Age')
plt.ylabel('Count')
plt.grid()
```

```
n, bins, patches = plt.hist(df_titanic['Age'], histtype='bar',
                            color='#0dc28d', align='mid',
                            rwidth=0.90, bins=7)
```

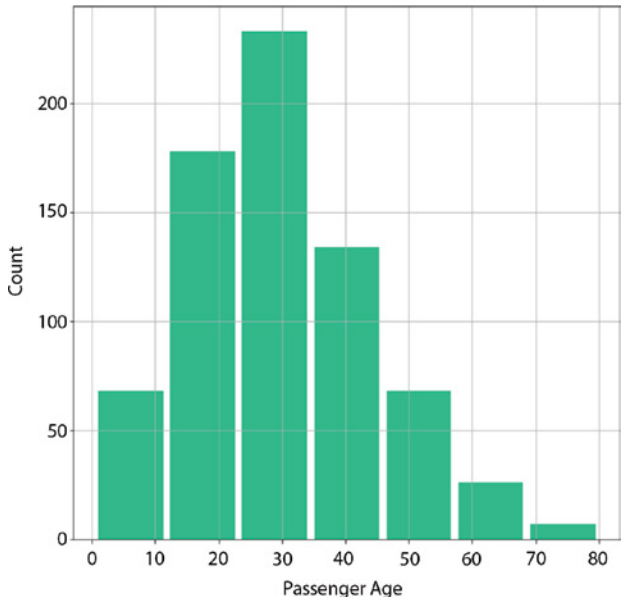


FIGURE 3.5
Histogram of Passenger Age values

The `pyplot hist()` function is used to create a histogram. The function takes several arguments. Some of the arguments that have been used in the preceding snippet are `histtype='bar'`, which signifies that you want a standard histogram; `rwidth=0.9`, which leaves space between the bars; and `bins=7`, which is the number of bars. You can find a full list of parameters supported by the `hist()` function at `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html#matplotlib.pyplot.hist`.

The appearance of the histogram is influenced by the binning strategy, which in turn is controlled by the value you pass into the `bins` parameter. If you pass an integer (as in the preceding snippet), Matplotlib will create the specified number of equal-width bins and use the boundaries (edges) of the bins to determine the number of values in each bin. You can optionally specify the bin edges instead of the number of bins. In most cases you simply specify the number of bins.

The `pyplot hist()` function returns a tuple, `(n, bins, patches)`. The first element of the tuple, `n`, is an array with the counts for each bin. The second element, `bins`, is an array of floating-point values that contains the bin edges, and the third element, `patches`, is an array of rectangle objects that represent the low-level drawing primitives used by Matplotlib to make the bars. You can use the `print()` statement to inspect the contents of the tuple:

```
print (n)
[ 68. 178. 233. 134.  68.  26.   7.]

print (bins)
[0.42
```

```
11.78857143
23.15714286
34.52571429
45.89428571
57.26285714
68.63142857
80.0]

print (patches[0])
Rectangle(xy=(0.988429, 0), width=10.2317, height=68, angle=0)
```

The following snippet uses `pyplot` functions to create a figure with four subplots (axes objects) and uses the object-oriented API to create histograms of the same data in each of the subplots, but with different numbers of bins. The result of this snippet is depicted in Figure 3.6.

```
# use pyplot functions and matplotlib object-oriented API
# to plot multiple histograms of the same data, with different
# binning strategies.

fig, axes_list = plt.subplots(2,2, figsize=(11,11))

# plot a histogram with 3 bins
axes_list[0,0].set_xlabel('Passenger Age')
axes_list[0,0].set_ylabel('Count')
axes_list[0,0].grid()

n1, bins1, patches1 = axes_list[0,0].hist(df_titanic['Age'], histtype='bar',
                            color='#0dc28d', align='mid',
                            rwidth=0.90, bins=3)


# plot a histogram with 10 bins
axes_list[0,1].set_xlabel('Passenger Age')
axes_list[0,1].set_ylabel('Count')
axes_list[0,1].grid()
n2, bins2, patches2 = axes_list[0,1].hist(df_titanic['Age'], histtype='bar',
                            color='#0dc28d', align='mid',
                            rwidth=0.90, bins=10)

# plot a histogram with 30 bins
axes_list[1,0].set_xlabel('Passenger Age')
axes_list[1,0].set_ylabel('Count')
axes_list[1,0].grid()
n3, bins3, patches3 = axes_list[1,0].hist(df_titanic['Age'], histtype='bar',
                            color='#0dc28d', align='mid',
                            rwidth=0.90, bins=30)
```

```
# plot a histogram with 100 bins
axes_list[1,1].set_xlabel('Passenger Age')
axes_list[1,1].set_ylabel('Count')
axes_list[1,1].grid()
n4, bins4, patches4 = axes_list[1,1].hist(df_titanic['Age'], histtype='bar',
                              color='#0dc28d', align='mid',
                              rwidth=0.90, bins=100)
```

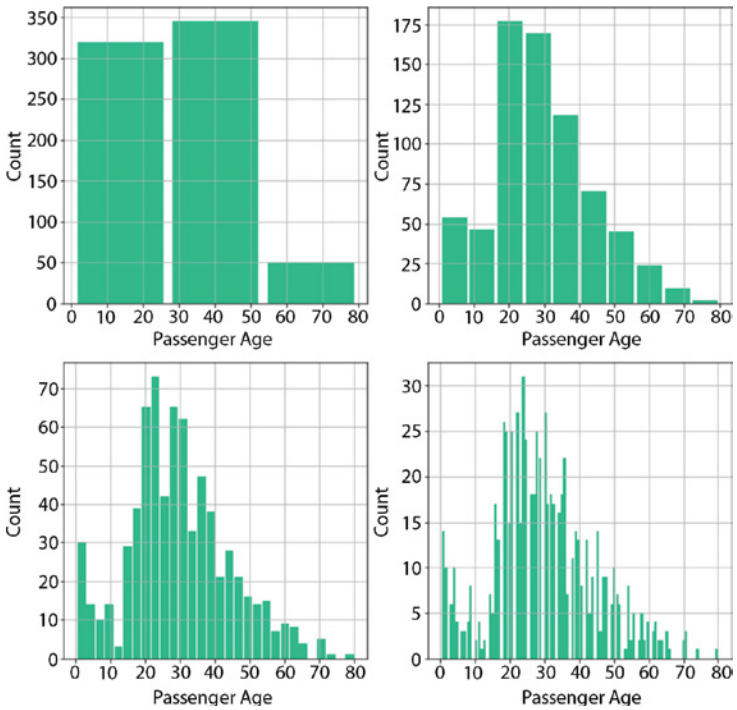


FIGURE 3.6
Histograms of Passenger Age values created using different binning strategies

As you can infer, the binning strategy significantly affects the appearance of the histogram and the inferences that you can make from the histogram. There is no set rule to the number of bins that must be used; often data scientists use a number of different binning strategies to reveal characteristics of the data that were not previously visible. A common rule of thumb is to set the number of bins to be the square root of the number of values as a starting point, and then update as necessary. A commonly used approach in statistics to select the bin width for histograms was proposed in 1981 by David Freedman and Persi Diaconis and is known as the Freedman-Diaconis rule. The general idea is to set the bin width to be 2 × IQR / (number of observations) $^{1/3}$. Using this equation to compute the bin width, you can divide the range of values by the bin width to work out the number of bins. You can get more information on this rule from the original paper published in 1981 titled "On the histogram as a density estimator." You can access a copy of the paper at `https://statistics.stanford.edu/sites/g/files/sbiybj6031/f/EFS%20NSF%20159.pdf`.

The Pandas dataframe object also provides limited plotting capabilities. These capabilities are built on top on Matplotlib, but in some situations, you may find the Pandas plotting functions simpler to use. The following snippet shows how you could create a simple histogram using the Pandas dataframe plot function:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import pandas as pd

# load the contents of a file into a pandas Dataframe
input_file = './datasets/titanic_dataset/original/train.csv'
df_titanic = pd.read_csv(input_file)

# set the index
df_titanic.set_index("PassengerId", inplace=True)

fig = plt.figure(figsize=(7,7))
plt.xlabel('Passenger Age')
plt.ylabel('Count')

df_titanic['Age'].plot.hist(color='#0dc28d', align='mid',
                            rwidth=0.90, bins=7, grid=True)
```

Bar Chart

Bar charts are commonly used when you are dealing with a categorical variable. Each bar in a bar chart represents some information about a categorical attribute, such as a count, mean, or other measure. Bar charts can be used with both nominal and ordinal categorical data. When plotting a bar chart for nominal categorical data it is common practice to order the bars so that the height (or length) of the bars increases in an orderly fashion.

This is possible because a category that contains nominal data does not have any inherent order between the values, and hence you can freely order the placement of the bars to create a visually pleasing figure. Continuing with the use of the Titanic dataset from the previous section, the following snippet creates a bar plot of the `Embarked` attribute. The resulting bar chart is depicted in Figure 3.7.

```
# use pyplot functions to plot a bar chart of the 'Embarked' attribute
fig = plt.figure(figsize=(9,9))
plt.xlabel('Embarkation Point')
plt.ylabel('Count')
plt.grid()

values = df_titanic['Embarked'].unique()
counts = df_titanic['Embarked'].value_counts(dropna=False)
x_positions = np.arange(len(values))

plt.bar(x_positions, counts, align='center')
plt.xticks(x_positions, values)
```

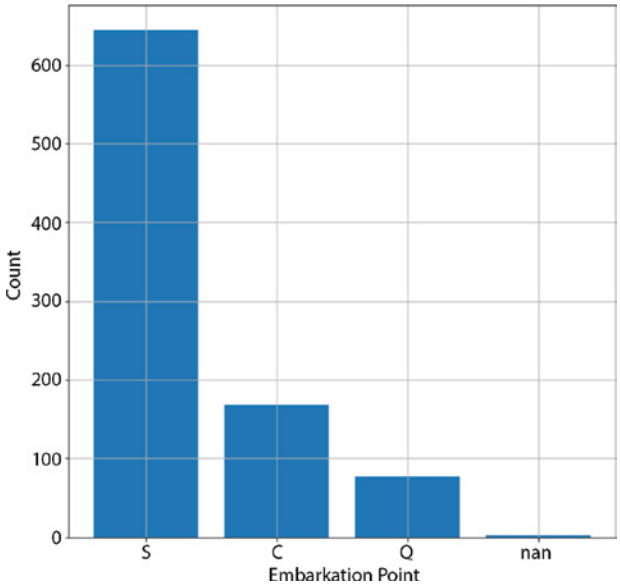


FIGURE 3.7
Bar chart of the Embarked attribute

Plotting bar charts is significantly simpler with the Pandas dataframe plot function. The following snippet demonstrates how you can create the same bar chart using Pandas functions:

```
# use Pandas dataframe functions to plot a bar chart of the 'Embarked' attribute
fig = plt.figure(figsize=(7,7))
plt.xlabel('Embarkation Point')
plt.ylabel('Count')
plt.grid()

df_titanic['Embarked'].value_counts(dropna=False).plot.bar(grid=True)
```

Grouped Bar Chart

If you want to show information about different subgroups within each category, a grouped bar chart can be used. A grouped bar chart, as its name suggests, is a chart with groups of bars clustered together. Each bar group provides information on one category, and the length of bars within the group provides information on the individual subgroups within the category.

For example, a grouped bar chart could be used to visualize the distribution of the number of individuals who survived and the number who did not, for each embarkation point. The following snippet creates a grouped bar chart for the `Embarked` attribute with two bars in each group. The resulting bar chart is depicted in Figure 3.8.

```
# a grouped bar chart for the Embarked attribute with
# two bars per group.
survived_df = df_titanic[df_titanic['Survived']==1]
not_survived_df = df_titanic[df_titanic['Survived']==0]
```

```
values = df_titanic['Embarked'].dropna().unique()
embarked_counts_survived = survived_df['Embarked'].value_counts(dropna=True)
embarked_counts_not_survived =
not_survived_df['Embarked'].value_counts(dropna=True)

x_positions = np.arange(len(values))

fig = plt.figure(figsize=(9,9))
plt.xlabel('Embarkation Point')
plt.ylabel('Count')
plt.grid()

bar_width = 0.35

plt.bar(x_positions, embarked_counts_survived, bar_width, color='#009bdb',
label='Survived')
plt.bar(x_positions + bar_width, embarked_counts_not_survived, bar_width,
color='#00974d', label='Not Survived')

plt.xticks(x_positions + bar_width, values)
plt.legend()

plt.show()
```

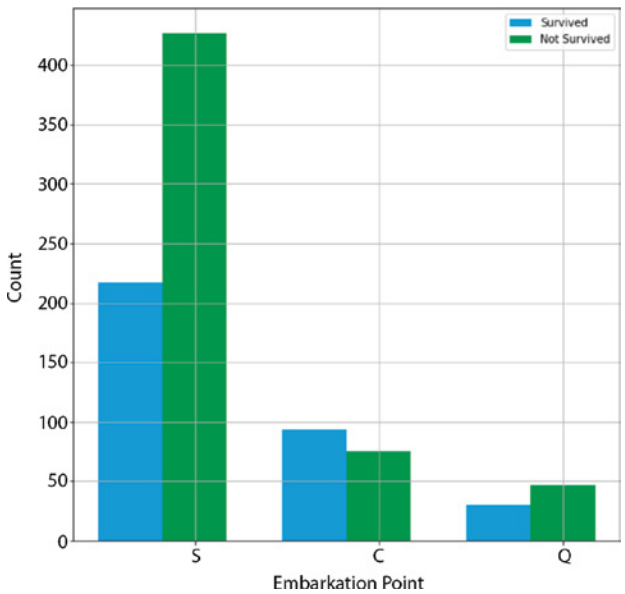


FIGURE 3.8
Grouped bar chart of the Embarked attribute

The following snippet shows how to use the Pandas plotting functions to draw the same grouped bar chart:

```
# a grouped bar chart for the Embarked attribute with
# two bars per group.
survived_df = df_titanic[df_titanic['Survived']==1]
not_survived_df = df_titanic[df_titanic['Survived']==0]

values = df_titanic['Embarked'].dropna().unique()
embarked_counts_survived = survived_df['Embarked'].value_counts(dropna=True)
embarked_counts_not_survived =
not_survived_df['Embarked'].value_counts(dropna=True)

embarked_counts_survived.name = 'Survived'
embarked_counts_not_survived.name = 'Not Survived'
df = pd.concat([embarked_counts_survived, embarked_counts_not_survived],
axis=1)

fig, axes = plt.subplots(figsize=(9,9))
plt.xlabel('Embarkation Point')
plt.ylabel('Count')

df.plot.bar(grid=True, ax=axes, color=['#009bdb', '#00974d'])
```

Stacked Bar Chart

A stacked bar chart provides another way to visualize the same information. Instead of having multiple bars in clustered groups, a stacked bar chart uses one bar per categorical value and splits the bar to depict the distribution of subgroups within the category.

The following snippet creates a stacked bar chart for the `Embarked` attribute showing the distribution of survivors from each embarkation point. The resulting bar chart is depicted in Figure 3.9.

```
# a stacked bar chart for the Embarked attribute
# showing the number of survivors in each category
survived_df = df_titanic[df_titanic['Survived']==1]
not_survived_df = df_titanic[df_titanic['Survived']==0]

values = df_titanic['Embarked'].dropna().unique()
embarked_counts_survived = survived_df['Embarked'].value_counts(dropna=True)
embarked_counts_not_survived =
not_survived_df['Embarked'].value_counts(dropna=True)

x_positions = np.arange(len(values))

fig = plt.figure(figsize=(9,9))
plt.xlabel('Embarkation Point')
plt.ylabel('Count')
plt.grid()

plt.bar(x_positions, embarked_counts_survived, color='#009bdb',
label='Survived')
plt.bar(x_positions, embarked_counts_not_survived, color='#00974d', label='Not
Survived', bottom=embarked_counts_survived)
```

```
plt.xticks(x_positions, values)
plt.legend()

plt.show()
```

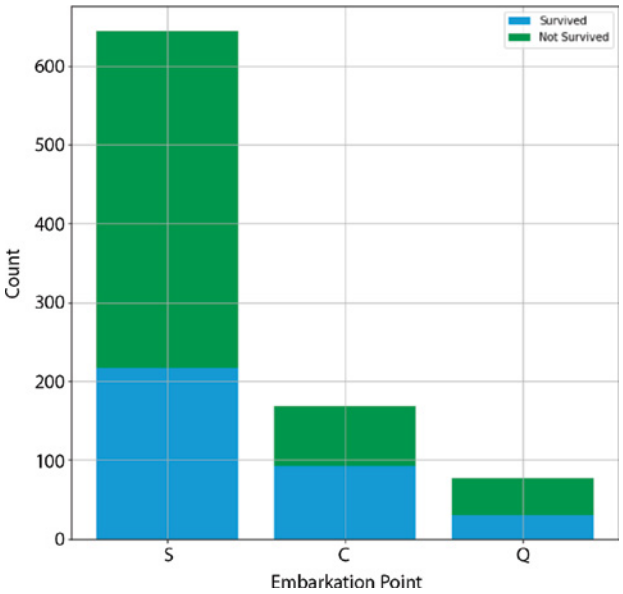


FIGURE 3.9
Stacked bar chart of the Embarked attribute

Creating a stacked bar chart with Pandas plotting functions is simply a matter of adding the `stacked=True` attribute to the argument list of the `dataframe.plot.bar()` function. The following snippet using Pandas plotting functions to draw the same stacked bar chart:

```
# a stacked percentage bar chart for the Embarked attribute
# showing the number of survivors in each category
survived_df = df_titanic[df_titanic['Survived']==1]
not_survived_df = df_titanic[df_titanic['Survived']==0]

values = df_titanic['Embarked'].dropna().unique()
embarked_counts_survived = survived_df['Embarked'].value_counts(dropna=True)
embarked_counts_not_survived =
not_survived_df['Embarked'].value_counts(dropna=True)

embarked_counts_survived.name = 'Survived'
embarked_counts_not_survived.name = 'Not Survived'
df = pd.concat([embarked_counts_survived, embarked_counts_not_survived],
axis=1)

fig, axes = plt.subplots(figsize=(9,9))
plt.xlabel('Embarkation Point')
plt.ylabel('Count')

df.plot.bar(stacked=True, grid=True, ax=axes, color=['#009bdb', '#00974d'])
```

The power of Pandas plotting functions over `pyplot` and Matplotlib is evident with stacked bar charts, especially if you have more than two groups per bar. With Matplotlib and `pyplot` functions, you will have to plot each group on top of the other using multiple `plt.bar()` statements. With Pandas, all you need to do is get your data in a dataframe and make a single call to `dataframe.plot.bar()`.

Stacked Percentage Bar Chart

If you want to show the percentage contribution of each subgroup within the categories, you can use a stacked percentage bar chart. The bars in a stacked percentage bar chart are all the same height. The following snippet creates a stacked percentage bar chart for the `Embarked` attribute showing the percentage of survivors from each embarkation point. The resulting bar chart is depicted in Figure 3.10.

```
# a stacked percentage bar chart for the Embarked attribute
# showing the number of survivors in each category
survived_df = df_titanic[df_titanic['Survived']==1]
not_survived_df = df_titanic[df_titanic['Survived']==0]

values = df_titanic['Embarked'].dropna().unique()
counts = df_titanic['Embarked'].value_counts(dropna=True)

embarked_counts_survived = survived_df['Embarked'].value_counts(dropna=True)
embarked_counts_not_survived =
not_survived_df['Embarked'].value_counts(dropna=True)

embarked_counts_survived_percent = embarked_counts_survived / counts * 100
embarked_counts_not_survived_percent = embarked_counts_not_survived / counts *
100

x_positions = np.arange(len(values))

fig = plt.figure(figsize=(9,9))
plt.xlabel('Embarkation Point')
plt.ylabel('Percentage')
plt.grid()

plt.bar(x_positions, embarked_counts_survived_percent, color='#009bdb',
label='Survived')
plt.bar(x_positions, embarked_counts_not_survived_percent, color='#00974d',
label='Not Survived', bottom=embarked_counts_survived_percent)

plt.xticks(x_positions, values)
plt.legend()

plt.show()
```

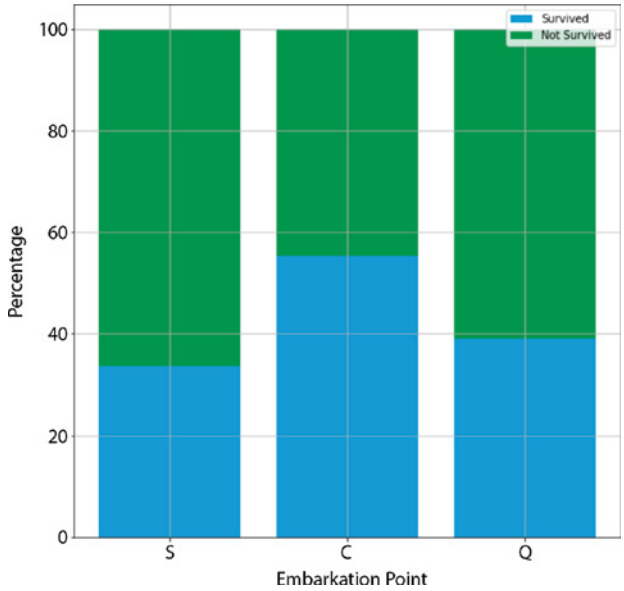


FIGURE 3.10
Stacked percentage bar chart of the Embarked attribute

The following snippet creates an equivalent chart using Pandas plotting functions:

```
# a stacked percentage bar chart for the Embarked attribute
# showing the number of survivors in each category
survived_df = df_titanic[df_titanic['Survived']==1]
not_survived_df = df_titanic[df_titanic['Survived']==0]

values = df_titanic['Embarked'].dropna().unique()
counts = df_titanic['Embarked'].value_counts(dropna=True)

embarked_counts_survived = survived_df['Embarked'].value_counts(dropna=True)
embarked_counts_not_survived =
not_survived_df['Embarked'].value_counts(dropna=True)

embarked_counts_survived_percent = embarked_counts_survived / counts * 100
embarked_counts_not_survived_percent = embarked_counts_not_survived / counts *
100

embarked_counts_survived_percent.name = 'Survived'
embarked_counts_not_survived_percent.name = 'Not Survived'
df = pd.concat([embarked_counts_survived_percent,
embarked_counts_not_survived_percent], axis=1)

fig, axes = plt.subplots(figsize=(9,9))
plt.xlabel('Embarkation Point')
plt.ylabel('% Survived ')

df.plot.bar(stacked=True, grid=True, ax=axes, color=['#009bdb', '#00974d'])
```

Pie Charts

Pie charts are an alternative to bar charts and can be used to plot the proportion of unique values in a categorical attribute. The `pyplot` module provides the `pie()` function that can be used to create a pie chart. The pie function of the `pyplot` module uses the underlying `pie()` method of the `axes` class. You can access the documentation for the `pyplot pie()` function at `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.pie.html`.

The following snippet uses the `pyplot pie()` function on the contents of the `Embarked` column of the Titanic dataset to create a pie chart that depicts the percentage of passengers boarding from each embarkation point. Figure 3.11 depicts the resulting pie chart.

```
# use pyplot functions to plot a pie chart of the 'Embarked' attribute
fig = plt.figure(figsize=(9,9))

embarkation_ports = df_titanic['Embarked'].dropna().unique()
counts = df_titanic['Embarked'].value_counts(dropna=True)

total_embarked = counts.values.sum()
counts_percentage = counts / total_embarked * 100

counts_percentage.values
plt.pie(counts_percentage.values,
        labels=embarkation_ports,
        autopct='%1.1f%%', shadow=True, startangle=90)
```

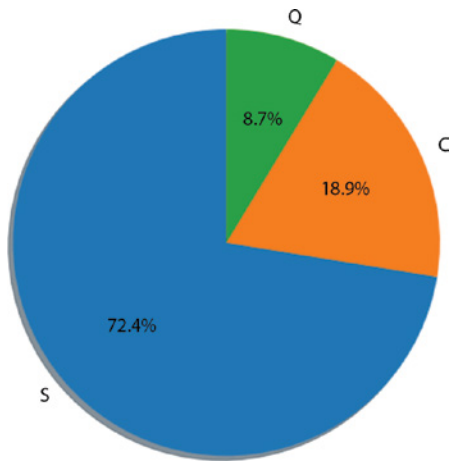


FIGURE 3.11
Pie chart of proportion of passengers embarking from different ports

You can also use the Pandas dataframe `plot.pie()` function to create pie charts. If your data is already in a Pandas dataframe, this approach will require significantly less code. The downside to using the Pandas plotting function is the reduced level of options to customize the pie chart. The following snippet uses Pandas plotting functions to make the same pie chart:

```
# use Pandas functions to plot a pie chart of the 'Embarked' attribute
fig = plt.figure(figsize=(7,7))
df_titanic['Embarked'].value_counts(dropna=True).plot.pie()
```

When the target attribute of a dataset is binary, a pie chart can help convey the distribution of values in each categorical attribute, grouped by the target binary attribute. The following snippet uses Matplotlib's `Axes.pie()` method to create three pie charts showing the percentage of survivors from the three embarkation ports in the Titanic dataset. Figure 3.12 depicts the resulting pie charts.

```
# three pie charts, showing the proportion
# of survivors for each embarkation point
# S = Southampton
# C = Cherbourg
# Q = Queenstown
S_df = df_titanic[df_titanic['Embarked'] == 'S']
C_df = df_titanic[df_titanic['Embarked'] == 'C']
Q_df = df_titanic[df_titanic['Embarked'] == 'Q']

S_Total_Embarked = S_df['Embarked'].count()
S_Survived_Count = S_df[S_df['Survived']==1].Embarked.count()
S_Survived_Percentage = S_Survived_Count / S_Total_Embarked * 100
S_Not_Survived_Percentage = 100.0 - S_Survived_Percentage

C_Total_Embarked = C_df['Embarked'].count()
C_Survived_Count = C_df[C_df['Survived']==1].Embarked.count()
C_Survived_Percentage = C_Survived_Count / C_Total_Embarked * 100
C_Not_Survived_Percentage = 100.0 - C_Survived_Percentage

Q_Total_Embarked = Q_df['Embarked'].count()
Q_Survived_Count = Q_df[Q_df['Survived']==1].Embarked.count()
Q_Survived_Percentage = Q_Survived_Count / Q_Total_Embarked * 100
Q_Not_Survived_Percentage = 100.0 - Q_Survived_Percentage


fig, axes = plt.subplots(1, 3, figsize=(16,4))

Wedge_Labels = ['Survived', 'Not Survived']
S_Wedge_Sizes = [S_Survived_Percentage, S_Not_Survived_Percentage]
C_Wedge_Sizes = [C_Survived_Percentage, C_Not_Survived_Percentage]
Q_Wedge_Sizes = [Q_Survived_Percentage, Q_Not_Survived_Percentage]

axes[0].pie(S_Wedge_Sizes, labels=Wedge_Labels, autopct='%1.1f%%', shadow=True,
startangle=90)
axes[0].set_title('Southampton')

axes[1].pie(C_Wedge_Sizes, labels=Wedge_Labels, autopct='%1.1f%%', shadow=True,
startangle=90)
axes[1].set_title('Cherbourg')

axes[2].pie(Q_Wedge_Sizes, labels=Wedge_Labels, autopct='%1.1f%%', shadow=True,
startangle=90)
axes[2].set_title('Queenstown')
```

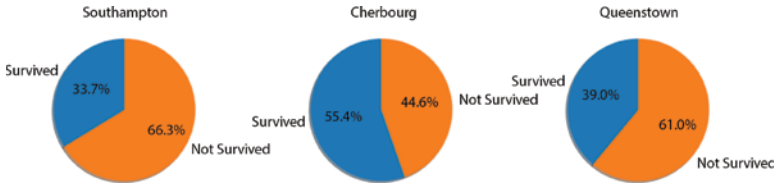


FIGURE 3.12
Pie charts showing the proportion of survivors from each embarkation point

The same pie charts can be created using far less code if you were to use the Pandas plotting functions, at the expense of loss in customizability. The following snippet will generate the same chart using Pandas plotting functions:

```
# three pie charts, showing the proportion
# of survivors for each embarkation point
# S = Southampton
# C = Cherbourg
# Q = Queenstown

survived_df = df_titanic[df_titanic['Survived']==1]
not_survived_df = df_titanic[df_titanic['Survived']==0]

values = df_titanic['Embarked'].dropna().unique()
counts = df_titanic['Embarked'].value_counts(dropna=True)

embarked_counts_survived = survived_df['Embarked'].value_counts(dropna=True)
embarked_counts_not_survived =
not_survived_df['Embarked'].value_counts(dropna=True)

embarked_counts_survived_percent = embarked_counts_survived / counts * 100
embarked_counts_not_survived_percent = embarked_counts_not_survived / counts *
100

embarked_counts_survived_percent.name = 'Survived'
embarked_counts_not_survived_percent.name = 'Not Survived'
df = pd.concat([embarked_counts_survived_percent,
embarked_counts_not_survived_percent], axis=1)

df.T.plot.pie(sharex=True, subplots=True, figsize=(16, 4))
```

Box Plot

A box plot provides a way to view the distribution of a numerical attribute. It was created in 1969 by John Tukey. A box plot is used to find out if the values of the attribute are symmetrically distributed, the overall spread of the values, and information on outliers. A box plot provides information on five statistical qualities of the attribute values:

- *First quartile:* 25% of the values of the attribute are less than this number. This is also known as the 25th percentile.
- *Second quartile:* 50% of the values of the attribute are less than this number. This is also known as the 50th percentile, or the median value.

- *Third quartile:* 75% of the values of the attribute are less than this number. This is also known as the 75th percentile.
- *Minimum:* This value is computed using the formula Min = Q1 – 1.5 * IQR, where IQR is the inter-quartile range, defined as Q3 – Q2. Any attribute values that are lower than this minimum will be treated as outliers in the plot.
- *Maximum:* This value is computed using the formula Min = Q3 + 1.5 * IQR, where IQR is the inter-quartile range, defined as Q3 – Q2. Any attribute values that are greater than this maximum will be treated as outliers in the plot.

The `pyplot` module provides the `boxplot()` function that can be used to create a box plot. The `boxplot()` function of the `pyplot` module uses the underlying `boxplot()` method of the `axes` class. You can access the documentation for the `pyplot boxplot()` function at `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html`.

The following snippet uses the `pyplot boxplot()` function on the contents of the `Age` column of the Titanic dataset to create a box plot that depicts the spreads of the values of the attribute. Figure 3.13 depicts the resulting box plot.

```
# use pyplot functions to create a box plot of the 'Embarked' attribute
fig , axes = plt.subplots(figsize=(9,9))
box_plot = plt.boxplot(df_titanic['Age'].dropna())
axes.set_xticklabels(['Age'])
```

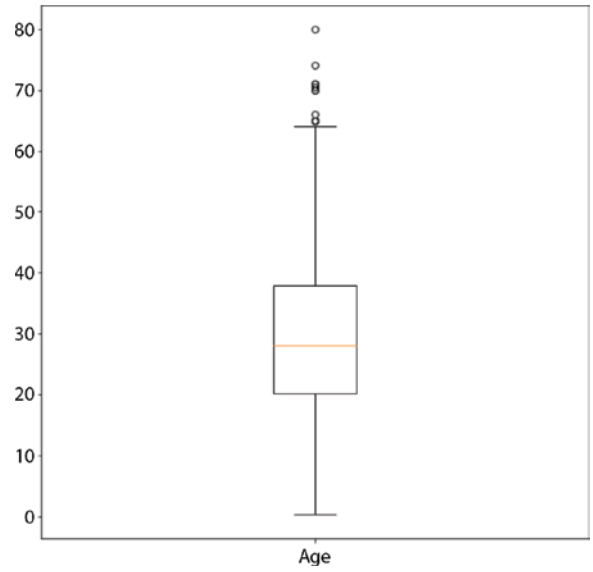


FIGURE 3.13
Box plot showing the distribution of the Age attribute

The following snippet will generate the same box plot using Pandas plotting functions:

```
df_titanic.boxplot(column = 'Age', figsize=(9,9), grid=False);
```

The compact nature of box plots makes them extremely useful to compare the distributions of different numerical attributes, or the distributions of subgroups within a numerical attribute. The

following snippet uses `pyplot` functions to create two box plots of the `Age` attribute, one for those that survived the Titanic disaster, and the other for those that did not. Figure 3.14 depicts the resulting box plots.

```
# compare box plots of the Age attribute for those who survived
# against those that did not.
survived_df = df_titanic[df_titanic['Survived']==1].dropna()
not_survived_df = df_titanic[df_titanic['Survived']==0].dropna()

fig , axes = plt.subplots(figsize=(9,9))
box_plot = plt.boxplot([survived_df['Age'], not_survived_df['Age']],
                       labels=['Survived', 'Not Survived'])
```

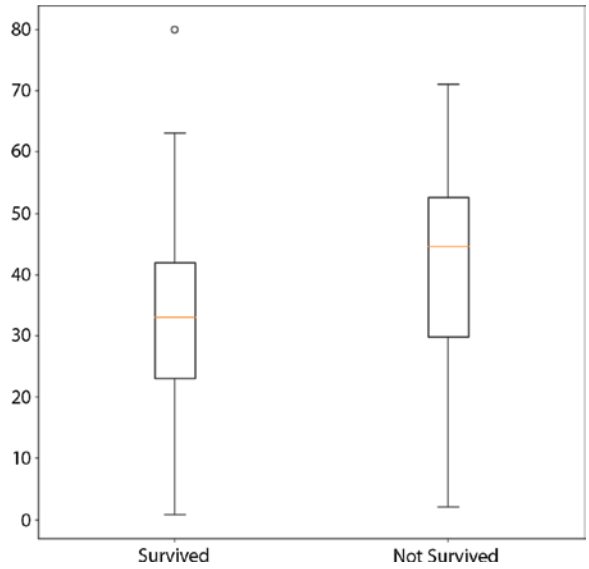


Figure 3.14
Box plots of the Age attribute comparing the distribution of survivors with those who did not survive the Titanic disaster

The following snippet will generate the same box plot using Pandas plotting functions:

```
df_titanic.boxplot(column = 'Age', by = 'Survived', figsize=(9,9), grid=False);
```

Scatter Plots

A scatter plot is a 2D plot that plots two continuous numeric attributes against each other. One attribute is plotted along the x-axis and the other is plotted along the y-axis. Scatter plots are a collection of points and are typically used to plot the correlation between variables, with each point of the scatter plot representing the value of two variables. Scatter plots can also be used to visualize the grouping of data. The following snippet creates a scatter plot of the `Age` and `Fare` attributes from the Titanic dataset after normalizing the values and imputing missing values with the mean of the attribute. Figure 3.15 depicts the resulting scatter plot.

```
# impute missing values:
# Age with the median age
# Fare with the mean fare
median_age = df_titanic['Age'].median()
df_titanic["Age"].fillna(median_age, inplace=True)

mean_fare = df_titanic['Fare'].mean()
df_titanic["Fare"].fillna(mean_fare, inplace=True)

# use pyplot functions to create a scatter plot of the 'Age' and 'Fare' attribute
fig , axes = plt.subplots(figsize=(9,9))
plt.xlabel('Age')
plt.ylabel('Fare')
plt.grid()

plt.scatter(df_titanic['Age'], df_titanic['Fare'])
```

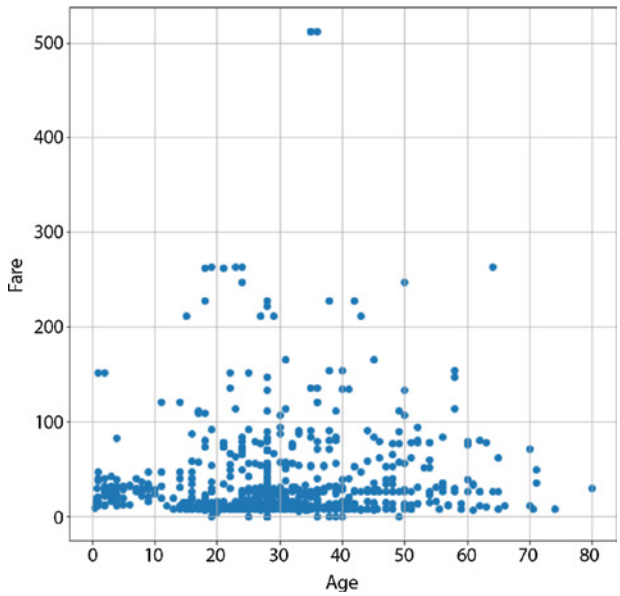


Figure 3.15
Scatter plot of the Age attribute against the Fare attribute

The Pandas dataframe contains the `plot.scatter()` function that can be used to create a scatter plot. The following snippet demonstrates the use of this function to create an equivalent scatter plot:

```
df_titanic.plot.scatter(x='Age', y='Fare', figsize=(9,9))
```

You can use a scatter plot to get a visual indicator of the degree of correlation between two attributes. It is common to create a matrix of scatter plots of each attribute in the dataset against every other attribute; this, however, is only practical for a small number of attributes. Figure 3.16 shows the scatter plot of attributes that have the ideal strong positive and strong negative correlation. The ideal strong positive correlation would occur when most of the points lie along a straight line from the bottom-left corner to the top-right corner of the plot. The ideal strong

negative correlation would occur when most of the points lie along a straight line from the top-left corner to the bottom-right corner of the plot.

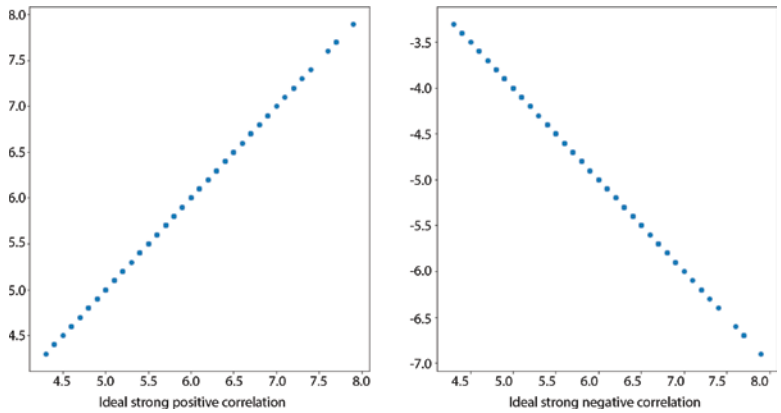


FIGURE 3.16
Scatter plots depicting the ideal strong positive and strong negative correlation

The following snippet presents a function that can be used to generate a scatter plot matrix out of the contents of a Pandas dataframe. The function takes three arguments. The first is the dataframe object, the second is the height of the figure (in inches), and the third is the width of the figure (in inches):

```
# Generates a M X M scatterplot matrix of subplots.
def generate_scatterplot_matrix(df_input, size_h, size_w):
    num_points, num_attributes = df_input.shape
    fig, axes = plt.subplots(num_attributes, num_attributes, figsize=(size_h,
size_w))

    column_names = df_input.columns.values

    for x in range(0, num_attributes):
        for y in range(0, num_attributes):
            axes[x , y].scatter(df_input.iloc[:,x], df_input.iloc[:,y])

            # configure the ticks
            axes[x , y].xaxis.set_visible(False)
            axes[x , y].yaxis.set_visible(False)

            # Set up ticks only on one side for the "edge" subplots...
            if axes[x , y].is_first_col():
                axes[x , y].yaxis.set_ticks_position('left')
                axes[x , y].yaxis.set_visible(True)
                axes[x , y].set_ylabel(column_names[x])

            if axes[x , y].is_last_col():
                axes[x , y].yaxis.set_ticks_position('right')
                axes[x , y].yaxis.set_visible(True)
```

```
            if axes[x , y].is_first_row():
                axes[x , y].xaxis.set_ticks_position('top')
                axes[x , y].xaxis.set_visible(True)

            if axes[x , y].is_last_row():
                axes[x , y].xaxis.set_ticks_position('bottom')
                axes[x , y].xaxis.set_visible(True)
                axes[x , y].set_xlabel(column_names[y])


    return fig, axes
```

To see a scatter plot matrix, let's use the `generate_scatter_plot` function on the popular Iris dataset. Recall from Chapter 2 that Scikit-learn contains a toy version of the Iris dataset. The dataset contains the heights and widths of the sepals and petals of iris flowers. The following snippet loads the Iris dataset into a dataframe and uses the `generate_scatter_plot` function to create a scatter plot matrix. The resulting figure is depicted in Figure 3.17.

```
import sklearn
iris = sklearn.datasets.load_iris()
df_iris = pd.DataFrame(iris.data, columns = iris.feature_names)
generate_scatterplot_matrix (df_iris, 20, 20)
```

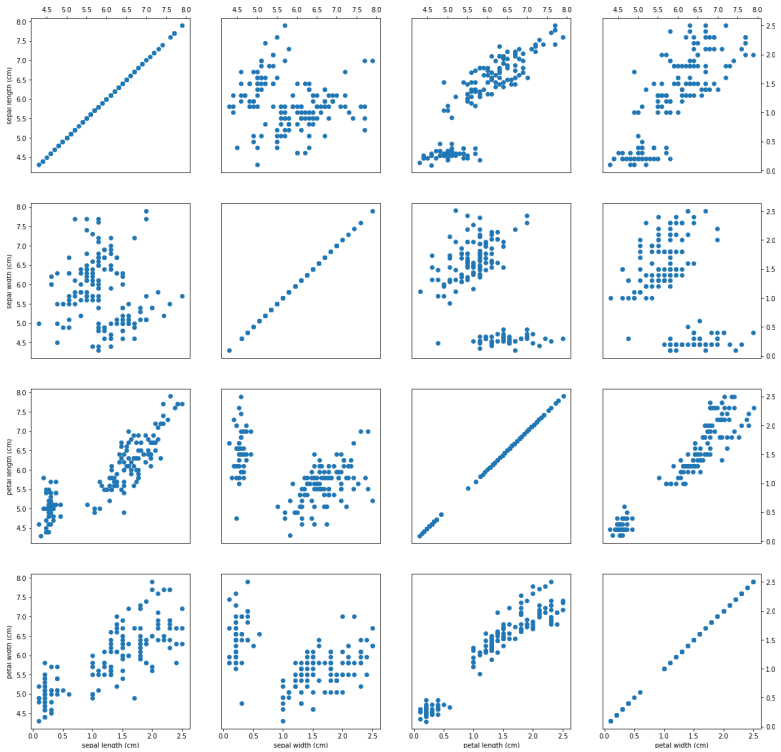


FIGURE 3.17
Scatter plot matrix of the features of the Iris dataset

As you can see, Matplotlib does not have a built-in function to create a scatter plot matrix. The Pandas plotting module has a function called `scatter_matrix()` that can be used to generate a scatter plot matrix from a dataframe. The following snippet demonstrates the use of the `scatter_matrix()` function:

```
import sklearn.datasets
import pandas.plotting

iris = sklearn.datasets.load_iris()
df_iris = pd.DataFrame(iris.data, columns = iris.feature_names)

pandas.plotting.scatter_matrix(df_iris, figsize=(12, 12))
```

Scatter plots can also be used to visualize clusters within data. The following snippet creates a synthetic dataset of x, y values in four clusters and plots all the values in a scatter plot. The synthetic data is created using Scikit-learn's `make_blobs()` function. You can learn more about this function at `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`. Figure 3.18 depicts the resulting scatter plot.

```
# scatter plots can also be used to visualize
# groups within data. This is illustrated below
# using a synthetic dataset

from sklearn.datasets import make_blobs
coordinates, clusters = make_blobs(n_samples = 500, n_features = 2, centers=4,
random_state=12)

coordinates_cluster1 = coordinates[clusters==0]
coordinates_cluster2 = coordinates[clusters==1]
coordinates_cluster3 = coordinates[clusters==2]
coordinates_cluster4 = coordinates[clusters==3]

fig , axes = plt.subplots(figsize=(9,9))
plt.xlabel('X Values')
plt.ylabel('Y Values')
plt.grid()

plt.scatter(coordinates_cluster1[:,0], coordinates_cluster1[:,1])
plt.scatter(coordinates_cluster2[:,0], coordinates_cluster2[:,1])
plt.scatter(coordinates_cluster3[:,0], coordinates_cluster3[:,1])
plt.scatter(coordinates_cluster4[:,0], coordinates_cluster4[:,1])
```

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from `www.wiley.com/go/machinelearningawscloud` or from GitHub using the following URL:

`https://github.com/asmtechnology/awsmlbook-chapter3.git`

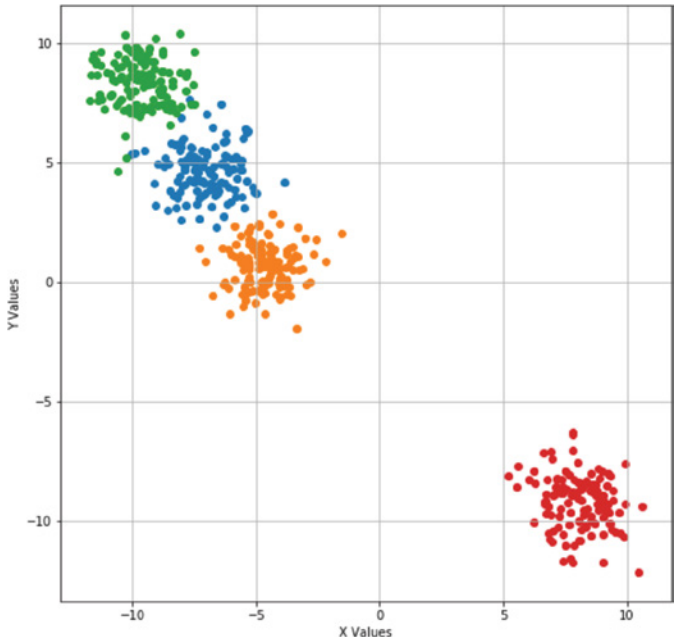


FIGURE 3.18
Scatter plot of four clusters of data

Summary

- Matplotlib is a plotting library for Python that offers functionality to generate numerous types of plots and the ability to customize these plots.
- The `pyplot` module within Matplotlib provides a high-level functional plotting interface.
- Matplotlib also provides a lower-level object-oriented API that can be used on its own, or in conjunction with the `pyplot` module.
- Seaborn is another Python plotting package that builds on top of Matplotlib.
- A histogram is commonly used to visualize the distribution of a numeric variable. Histograms are not applicable when dealing with categorical variables.
- The binning strategy significantly affects the appearance of a histogram.
- Bar charts are commonly used when you are dealing with categorical variables. Bar charts can be used with both nominal and ordinal categorical data.
- A stacked bar chart uses one bar per categorical value and splits the bar to depict the distribution of subgroups within the category.
- A box plot provides a way to view the distribution of a numerical attribute.

Chapter 4

Creating Machine Learning Models with Scikit-learn

WHAT'S IN THIS CHAPTER

- Introduction to Scikit-learn
- Learn to split your training data into training and testing sets
- Learn to use k-fold cross validation
- Learn to create different types of machine learning models

In Chapter 2, you learned about techniques to explore data and perform feature engineering. In this chapter you will learn to use Scikit-learn to split your training data into training and test sets, and to create different types of machine learning models. This chapter will use the Titanic and Iris datasets to illustrate different types of model-building techniques. A copy of these datasets is included with the files that accompany this chapter.

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from www.wiley.com/go/machinelearningawscloud or from GitHub using the following URL:

```
https://github.com/asmtechnology/awsmlbook-chapter4.git
```

Introducing Scikit-learn

Scikit-learn is a Python library that provides a number of features that are suitable for machine learning engineers and data scientists. It was developed by David Cournapeau in 2007 and today provides ready-to-use implementations of several popular machine learning algorithms such as linear regression, logistic regression, support vector machines, clustering, and random forests. In addition to providing ready-to-use implementations of popular machine learning algorithms, Scikit-learn also provides tools to split datasets into test-train subsets, implement k-fold cross validation, evaluate model performance using popular metrics, and detect outliers, and includes algorithms to aid with feature selection. Scikit-learn builds upon several other libraries such as NumPy, Pandas, and Matplotlib, and at its core is focused on model-building and -evaluation tasks, and not tasks such as data loading and visualization.

Scikit-learn is one of the reasons for the rise in number of real-world applications of machine learning. Scikit-learn's vast collection of algorithms allow you to get started with machine learning without a significant background in mathematics and statistics.

In order to use your data with Scikit-learn, it must be loaded into NumPy arrays or Pandas dataframes. You can get more information on the capabilities of Scikit-learn at `https://scikit-learn.org/stable/`.

Creating a Training and Test Dataset

At its heart, building a machine learning model involves creating a computer program that can draw inferences from the features during the training phase and then testing the quality of the model by making predictions. A common practice involves setting aside some of the labeled training data before the model-building phase and testing the model using this data that the model has not previously encountered. The performance of the model on this unseen data is used to determine if the model is good enough, or if improvements are needed. The benefit of having a separate training and testing set is that it ensures the model has not memorized the training examples (a phenomenon known as overfitting). It is important to note that in the case of supervised learning, both the training and test sets are labeled and the original dataset must be evenly shuffled before the subsets are created.

Scikit-learn provides a function called `train_test_split()` in the `model_selection` submodule that can be used to split a Pandas dataframe into two dataframes, one for model building and the other for model evaluation. The `test_train_split()` function has several parameters, most of which have default values. The most commonly used parameters are:

- `test_size`: This value can be an integer or floating-point number. When the value is an integer, it specifies the number of elements that should be retained for the test set. When the value is a floating-point number, it specifies the percentage of the original dataset to include in the test set.
- `random_state`: This is an integer value that is used to seed the random-number generator used to shuffle the samples.

The output of the `train_test_split()` function is a list of four arrays in the following order:

- The first item of the list is an array that contains the training set features.
- The second item of the list is an array that contains the test set features.
- The third item of the list is an array that contains the training set labels (target variable).
- The fourth item of the list is an array that contains the test set labels.

You can find detailed information on the parameters of the `train_test_split()` function at `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`.

The following snippet demonstrates the use of this function to split the Iris flowers dataset into a training and test set, with 25% of the data reserved for the test set:

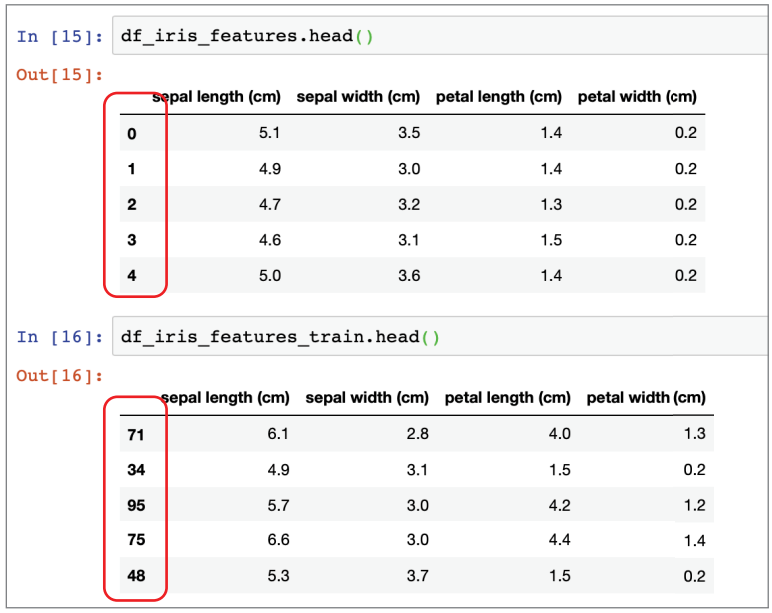
```
import numpy as np
import pandas as pd
# load iris data set
```

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
df_iris_features = pd.DataFrame(data = iris_dataset.data,
columns=iris_dataset.feature_names)
df_iris_target = pd.DataFrame(data = iris_dataset.target, columns=['class'])
# split iris dataset
iris_split = train_test_split(df_iris_features, df_iris_target,
                test_size=0.25, random_state=17)
df_iris_features_train = iris_split[0]
df_iris_features_test = iris_split[1]
df_iris_target_train = iris_split[2]
df_iris_target_test = iris_split[3]
```

You can use the dataframe's `shape` property to inspect the size of the training and test datasets created by the `train_test_split()` function:

```
df_iris_features.shape, df_iris_target.shape
((150, 4), (150, 1))
df_iris_features_train.shape, df_iris_target_train.shape
((112, 4), (112, 1))
df_iris_features_test.shape, df_iris_target_test.shape
((38, 4), (38, 1))
```

If you use the dataframe's `head()` method to inspect the first five rows of the original `df_iris_features` dataset and compare it with the first five rows of the `df_iris_features_train` dataset, you will notice that the `train_test_split()` function has automatically shuffled the data before splitting. This is illustrated in Figure 4.1.



```
In [15]: df_iris_features.head()
```

Out[15]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
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	0.2

```
In [16]: df_iris_features_train.head()
```

Out[16]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
71	6.1	2.8	4.0	1.3
34	4.9	3.1	1.5	0.2
95	5.7	3.0	4.2	1.2
75	6.6	3.0	4.4	1.4
48	5.3	3.7	1.5	0.2

FIGURE 4.1 Scikit-learn's `train_test_split()` method automatically shuffles the data prior to splitting.

The default behavior of the `train_test_split()` function is to shuffle the data, then determine the boundary observation where the training set should end and prepare two datasets by splitting at this boundary position. If the problem you are trying to solve is one of multi-class classification and your original data has a disproportionate number of samples from one category over the other, then it is important to ensure that the split datasets also have similar proportions. The `train_test_split()` function has a parameter called `stratify` that can be used to achieve a stratified split, maintaining the proportions of categorical observations before and after the split. The following snippet demonstrates the use of the `stratify` parameter:

```
# iris dataset, with stratified sampling
iris_split_strat = train_test_split(df_iris_features, df_iris_target,
                                    test_size=0.25, random_state=17,
stratify=df_iris_target)
df_iris_features_train2 = iris_split_strat[0]
df_iris_features_test2 = iris_split_strat[1]
df_iris_target_train2 = iris_split_strat[2]
df_iris_target_test2 = iris_split_strat[3]
```

The following snippet uses Pandas' plotting functions to create a bar chart of the distribution of categories in the original dataset, the unstratified training set, and the stratified training set. Note that the distribution in the stratified set is closer to the original, though not identical. The resulting plots are depicted in Figure 4.2.

```
# visualize the distribution of target values in the
# original dataset and the training sets created by the train_test_split
# function, with and without stratification
# use Pandas dataframe functions to plot a bar chart of the 'Embarked' attribute
fig, axes = plt.subplots(1, 3, figsize=(15,5))
axes[0].set_title('df_iris_target')
df_iris_target['class'].value_counts(dropna=False).plot.bar(grid=True,
ax=axes[0])
axes[1].set_title('df_iris_target_train')
df_iris_target_train['class'].value_counts(dropna=False).plot.bar(grid=True,
ax=axes[1])
axes[2].set_title('df_iris_target_train2')
df_iris_target_train2['class'].value_counts(dropna=False).plot.bar(grid=True,
ax=axes[2])
```

The distribution of target values between the three categories of flowers is identical in the Iris flowers dataset. This can be seen in the first histogram in Figure 4.2, with each category having 50 values. To better illustrate the use of stratified sampling, the following snippet loads the toy version of the UCI ML wines dataset and plots the difference between stratified and unstratified splits. The UI ML wines dataset is another popular dataset used by beginners for multi-class classification problems. It contains a number of numeric features that contain the results of chemical analysis on wines grown in four different regions of Italy, and a categorical target that indicates the overall quality of the wine. You can find information on the attributes of this dataset at `https://scikit-learn.org/stable/datasets/index.html`. The resulting plots are depicted in Figure 4.3.

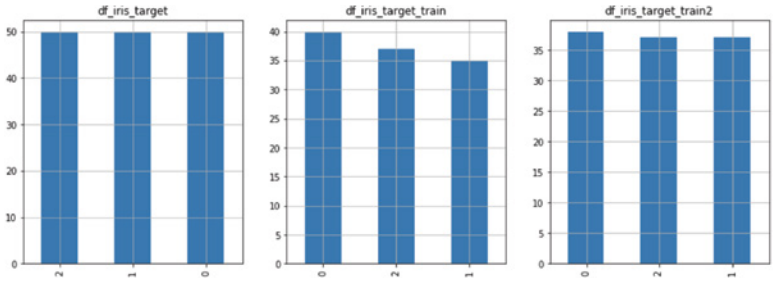


FIGURE 4.2
Comparison of the distribution of target variables in the original and split datasets, with and without stratified sampling

```
# Load the UCI ML Wines dataset
from sklearn.datasets import load_wine
wine_dataset = load_wine()
df_wine_features = pd.DataFrame(data = wine_dataset.data,
columns=wine_dataset.feature_names)
df_wine_target = pd.DataFrame(data = wine_dataset.target, columns=['class'])
#wines dataset
wines_split = train_test_split(df_wine_features, df_wine_target,
                               test_size=0.25, random_state=17)
df_wine_features_train = wines_split[0]
df_wine_features_test = wines_split[1]
df_wine_target_train = wines_split[2]
df_wine_target_test = wines_split[3]
# wines dataset, with stratified sampling
wines_split_strat = train_test_split(df_wine_features, df_wine_target,
                                     test_size=0.25, random_state=17,
stratify=df_wine_target)
df_wine_features_train2 = wines_split_strat[0]
df_wine_features_test2 = wines_split_strat[1]
df_wine_target_train2 = wines_split_strat[2]
df_wine_target_test2 = wines_split_strat[3]

# visualize the distribution of target values in the
# original wines dataset and the training sets created by the train_test_split
# function, with and without stratification
# use Pandas dataframe functions to plot a bar chart of the 'Embarked' attribute
fig, axes = plt.subplots(1, 3, figsize=(15,5))
axes[0].set_title('df_wine_target')
df_wine_target['class'].value_counts(dropna=False).plot.bar(grid=True,
ax=axes[0])
axes[1].set_title('df_wine_target_train')
df_wine_target_train['class'].value_counts(dropna=False).plot.bar(grid=True,
ax=axes[1])
axes[2].set_title('df_wine_target_train2')
df_wine_target_train2['class'].value_counts(dropna=False).plot.bar(grid=True,
ax=axes[2])
```

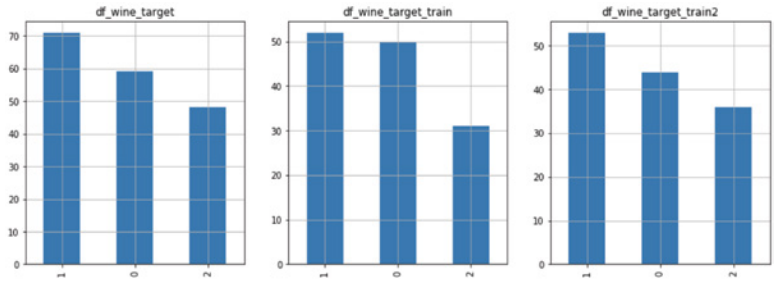


FIGURE 4.3
Comparison of the distribution of target variables in the original and split versions of the UCI ML wines dataset, with and without stratified sampling

K-Fold Cross Validation

The main drawback with the idea of splitting the training dataset into a training and validation set is that it is possible for the samples in the training set to exhibit characteristics that may not be found in any of the samples in the test set. While shuffling the data can help mitigate this, the extent of the mitigation depends on various factors such as the size of the original dataset, and the proportion of samples that exhibit a particular characteristic. The solution to avoid creating a model that is susceptible to characteristics only found in the training set is embodied in a technique called k-fold cross validation. At a very high level, the k-fold cross validation technique works as follows:

1. Choose a value of k.
2. Shuffle the data.
3. Split the data into k equal subsets.
4. For each value of k:
    a. Train a model that uses the kth subset as the test set and the samples of the k-1 subsets as the training set.
    b. Record the performance of the model when making predictions on the kth subset.
5. Compute the mean performance of the individual models to work out the overall performance.

K-fold cross validation can help minimize the possibility of the model picking up on unexpected bias in the training set. The idea behind k-fold cross validation is to shuffle the entire dataset randomly and divide it into a number of smaller sets (known as folds) and train multiple models (or the same model multiple times). During each training and evaluation cycle, one of the folds will be held out as the test set and the remaining will make the training set. This is illustrated in Figure 4.4.

If k=1, then the k-fold cross validation approach becomes similar to the train/test split method discussed earlier in this chapter. If k=n, the number of samples in the training set, then in effect the test set contains only one sample, and each sample will get to be part of the test set during one of the iterations. This technique is also known as leave-one-out cross validation. Many academic research papers use k=5 or k=10; however, there is no hard-and-fast rule governing the value of k.

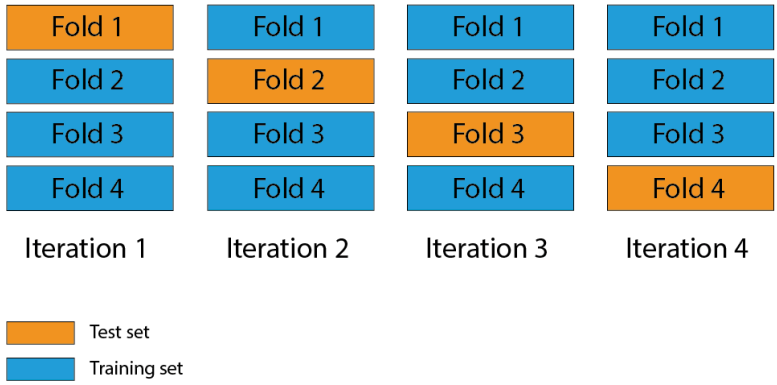


FIGURE 4.4
Cross-validation using k-folds

Scikit-learn provides a class called `KFold` as part of the `model-building` module that can be used to create the folds and enumerate through the folds. The constructor for the `KFold` class takes three parameters:

- `n_splits`: An integer that represents the number of folds required.
- `shuffle`: An optional Boolean value that indicates whether the data should be shuffled before the folds are created.
- `random_state`: An optional integer that is used to seed the random-number generator used to shuffle the data.

The `KFold` class provides two methods:

- `get_n_splits()`: Returns the number of folds
- `split()`: Gets the indices of the training and test set members for each fold.

You can find more information on the `KFold` class at `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html`. The following snippet demonstrates the use of the `KFold` class to split the contents of the Iris dataset into 10 folds and generate the test and training sets:

```
# perform 10-fold split on the Iris dataset
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle=True)
fold_number = 1
for train_indices, test_indices in kf.split(df_iris_features):

    print("Fold number:", fold_number)
    print("Training indices:", train_indices)
    print("Testing indices::", test_indices)

    fold_number = fold_number + 1

    df_iris_features_train = df_iris_features.iloc[train_index]
    df_iris_target_train = df_iris_target.iloc[train_index]
```

```
    df_iris_features_test = df_iris_features.iloc[test_index]
    df_iris_target_test = df_iris_target.iloc[test_index]
```

You can inspect the index positions that constitute the training and test sets for each iteration. The indices for the first two iterations are presented here:

```
Fold number: 1
Training indices: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
15  16  18
  19  20  21  22  23  26  27  28  29  31  32  33  34  35  36  38  39  41
  42  43  44  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  68  69  70  71  72  73  74  75  76  77  78  79
  80  82  83  84  85  86  87  88  89  90  91  94  95  96  97  98  99 100
 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
 120 121 122 123 124 125 127 128 129 130 131 132 133 134 135 136 137 138
 139 140 141 142 143 145 146 147 148]
Testing indices:: [ 17  24  25  30  37  40  45  67  81  92  93 119 126 144 149]

Fold number: 2
Training indices: [  1   2   3   4   5   6   7   8   9  10  12  13  14  16  17
18  19  20
  21  22  23  24  25  26  27  28  29  30  31  32  34  35  36  37  39  40
  41  42  43  44  45  46  47  48  50  51  52  53  55  56  57  58  59  60
  61  62  63  64  65  67  68  69  70  71  73  74  75  76  77  78  80  81
  83  84  85  86  87  88  89  91  92  93  94  95  96  97  98  99 100 101
 102 103 104 105 106 107 109 110 111 112 113 114 115 116 117 118 119 120
 121 122 123 124 125 126 128 130 131 132 133 134 135 136 137 138 139 140
 141 142 143 144 145 146 147 148 149]
Testing indices:: [  0  11  15  33  38  49  54  66  72  79  82  90 108 127 129]
```

Creating Machine Learning Models

In this section you will learn to create models that can be used to both predict the value of the target variable from the feature variables and classify data. A selection of models will be looked at, some of which assume a linear relationship between the target and the features, as well as models that can be used when the relationship is nonlinear.

Linear Regression

Linear regression is a statistical technique that aims to find the equation of a line (or hyperplane) that is closest to all the points in the dataset. To understand how linear regression works, let's assume you have a training dataset of 100 rows, and each row consists of three features, Y1, Y2, and Y3, and a known target value X. Linear regression will assume that the relationship between the target variable X and input features Y1, Y2, and Y3 is linear, and can be expressed by this equation:

$$X_i = \alpha Y1_i + \beta Y2_i + \gamma Y3_i + \varepsilon.$$

where:

- X_i is the predicted value of the i^{th} target variable.
- $Y1_i$ is the i^{th} value of feature Y1.

- $Y2_i$ is the i^{th} value of feature Y2.
- $Y3_i$ is the i^{th} value of feature Y3.
- α, β, γ are the coefficients of the features Y1, Y2, Y3.
- ε is a constant term, also known as the bias term or intercept.

The training process will iterate over the entire training set multiple times and calculate the best values of α, β, γ, and ε. A set of values is considered better if they minimize an error function. An error function is a mathematical function that captures the difference between the predicted and actual values of X_i. Root mean square error (RMSE) is a commonly used error function and is expressed mathematically as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\left(X_i' - X_i\right)^2}{N}}$$

In effect, linear regression attempts to find the best line (or hyperplane in higher dimensions) that fits all the data points. The output of linear regression is a continuous, unbounded value. It can be a positive number or a negative number, and it can have any value, depending on the inputs with which the model was trained. Therefore, linear regression models are commonly used to predict a continuous numeric value.

Scikit-learn implements linear regression in a class called `LinearRegression`, which is part of the `linear_model` module. We will now use this class to implement a linear regression model on the popular Boston housing dataset. The dataset consists of 506 rows, and each row consists of 13 continuous numeric features that contain information such as the per-capita crime rate, average number of rooms per house, rate of property tax, and pupil-teacher ratio. The target value contains the median house price of owner-occupied homes in various parts of Boston.

The dataset does not contain any missing values, and Scikit-learn includes the entire dataset as part of its `datasets` module. You can find more information on the attributes of this dataset at `https://scikit-learn.org/stable/datasets/index.html`. Recall from Chapter 2 that Scikit-learn provides a function called `DESCR` that can be used to print the description of a toy dataset. The following snippet loads the Boston housing dataset and uses the `DESCR` function to print the description of the dataset:

```
# load boston house dataset
from sklearn.datasets import load_boston
boston_dataset = load_boston()
df_boston_features = pd.DataFrame(data = boston_dataset.data,
columns=boston_dataset.feature_names)
df_boston_target = pd.DataFrame(data = boston_dataset.target, columns=['price'])
# print a description of the dataset.
print(boston_dataset.DESCR)
Boston house prices dataset
---------------------------
**Data Set Characteristics:**
    :Number of Instances: 506
    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.
```

```
    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
    :Missing Attribute Values: None
    :Creator: Harrison, D. and Rubinfeld, D.L.
This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.
The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.
The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References
   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

Creating the linear model involves splitting the 506 rows of the Boston housing dataset into a training set and a validation set, and using the training set to train the model. The following code snippet creates a 75/25 split of the 506 rows and uses 75% of the original data to train a linear regression model:

```
# create a training dataset and a test dataset using a 75/25 split.
from sklearn.model_selection import train_test_split
boston_split = train_test_split(df_boston_features, df_boston_target,
```

```
                                test_size=0.25, random_state=17)
df_boston_features_train = boston_split[0]
df_boston_features_test = boston_split[1]
df_boston_target_train = boston_split[2]
df_boston_target_test = boston_split[3]
# train a linear model
from sklearn.linear_model import LinearRegression
linear_regression_model = LinearRegression(fit_intercept=True)
linear_regression_model.fit(df_boston_features_train, df_boston_target_train)
```

You can instantiate a linear regression model by using the class constructor. The constructor has four parameters, all of which are optional. In most cases, you will instantiate a `LinearRegression` instance using the default zero-parameter constructor:

```
linear_regression_model = LinearRegression()
```

In the preceding snippet, the `fit_intercept` constructor parameter is set to `True`. The `fit_intercept` parameter is used to indicate that the samples are not zero-centered and that the model should calculate the intercept term. If `fit_intercept` is `False`, then the model will assume the y-axis intercept is 0 and will only attempt to fit lines (or hyperplanes) that satisfy this constraint. You can find more information on the constructor parameters at `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`.

Once you have created a `LinearRegression` instance, training the model is a simple matter of calling the `fit()` method, and passing in a dataframe that contains the features and a dataframe that contains the known target values. Once training is complete, you can access the coefficients and intercept terms of the linear model using the `coef_` and `intercept_` attributes of the `LinearRegression` instance. The following snippet lists the coefficients and intercept terms after training a linear regression model using the Boston house prices dataset. Note there are 13 coefficients, corresponding to the 13 feature variables:

```
print (linear_regression_model.coef_)
[[-1.12960344e-01  5.48578928e-02  6.71605489e-02  3.26195457e+00
  -1.70702665e+01  3.49123817e+00  7.03121906e-05 -1.37355630e+00
   3.12880217e-01 -1.32867294e-02 -9.57749225e-01  7.70369247e-03
  -5.59461017e-01]]
print (linear_regression_model.intercept_)
[38.51522467]
```

Once you have the trained model, you can use the model to make predictions. The following snippet uses the `linear_regression_model` object to make predictions on the test set (25% of the original 506 samples):

```
# use the linear model to create predictions on the test set.
predicted_median_house_prices =
linear_regression_model.predict(df_boston_features_test)
```

You will learn about ways to evaluate machine learning models in Chapter 5, but in this case you could get an idea of the quality of predictions created by this model by creating a scatter plot of the predictions made by the model on the test set against the actual house prices in the test set. The following snippet uses functions from Matplotlib's `pyplot` module to create a scatter plot. The resulting plot is depicted in Figure 4.5.

```
%matplotlib inline
import matplotlib.pyplot as plt
# use pyplot module to create a scatter plot of predicted vs expected values
fig, axes = plt.subplots(1, 1, figsize=(9,9))
plt.scatter(df_boston_target_test, predicted_median_house_prices)
plt.xlabel("Expected Prices")
plt.ylabel("Predicted prices")
plt.title("Expected vs Predicted prices")
```

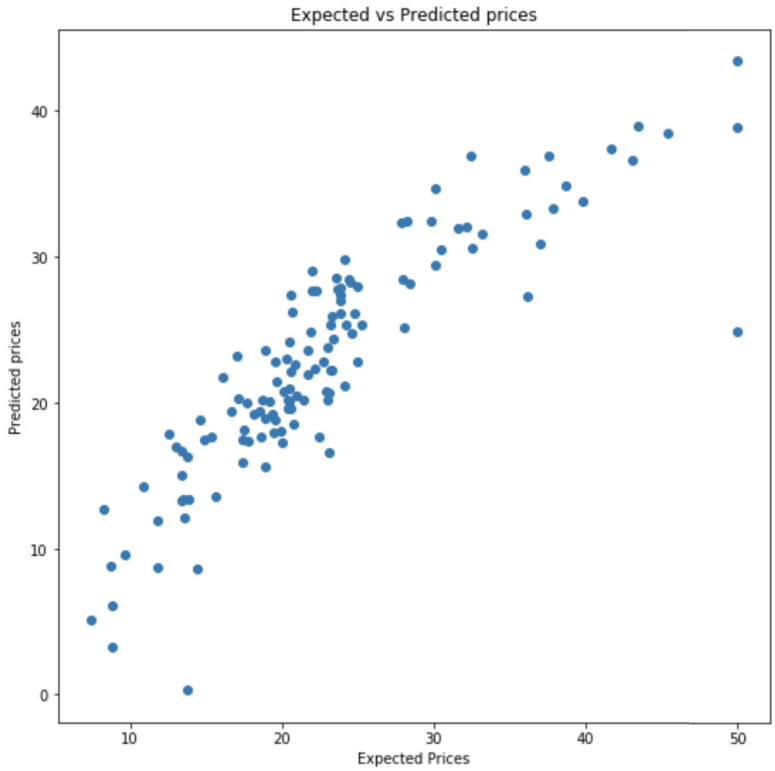


FIGURE 4.5
Scatter plot of expected vs. predicted house prices

In the ideal case, the scatter plot of the expected house prices against the values predicted by your model should be close to a straight line.

To better understand the result of changing the `fit_intercept` parameter while creating the `LinearRegression` instance, the following snippet creates a synthetic dataset of 50 random two-dimensional points and attempts to create two linear regression models on the data. The first model is created with `fit_intercept = False`, and the second model is created with `fit_intercept = True`. Both models are presented the X and Y coordinates of the 50 points as

training data, with the X coordinate values representing the feature variable and the Y coordinate values representing the target.

In effect, after training, the models will be able to predict the Y coordinate from the X coordinate. Since the X coordinate is the only input feature of this model, the model will contain only one coefficient term. The snippet then creates a scatter plot of the 50 points and overlays the regression line generated by each model. The resulting plot is depicted in Figure 4.6.

```
# create a synthetic regression dataset of X, Y values.
from sklearn.datasets import make_regression
from sklearn.preprocessing import MinMaxScaler
SyntheticX, SyntheticY = make_regression(n_samples=50, n_features=1, noise=35.0,
random_state=17)
x_scaler = MinMaxScaler()
x_scaler.fit(SyntheticX.reshape(-1,1))
SyntheticX = x_scaler.transform(SyntheticX.reshape(-1,1))
y_scaler = MinMaxScaler()
y_scaler.fit(SyntheticY.reshape(-1,1))
SyntheticY = y_scaler.transform(SyntheticY.reshape(-1,1))

# demonstrate effect of fit_intercept parameter on a simple synthetic dataset.
linear_regression_model_synthetic1 = LinearRegression(fit_intercept=True)
linear_regression_model_synthetic1.fit(SyntheticX, SyntheticY)
linear_regression_model_synthetic2 = LinearRegression(fit_intercept=False)
linear_regression_model_synthetic2.fit(SyntheticX, SyntheticY)
c1 = linear_regression_model_synthetic1.coef_
i1 = linear_regression_model_synthetic1.intercept_
YPredicted1 = np.dot(SyntheticX, c1) + i1
c2 = linear_regression_model_synthetic2.coef_
i2 = linear_regression_model_synthetic2.intercept_
YPredicted2 = np.dot(SyntheticX, c2) + i2
# use pyplot module to create a scatter plot of synthetic dataset
# and overlay the regression line from the two models.
fig, axes = plt.subplots(1, 1, figsize=(9,9))
axes.axhline(y=0, color='k')
axes.axvline(x=0, color='k')
plt.scatter(SyntheticX, SyntheticY)
plt.plot(SyntheticX, YPredicted1, color='#042fed', label='fit_intercept=True')
plt.plot(SyntheticX, YPredicted2, color='#d02fed', label='fit_intercept=False')
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
```

In Figure 4.6, you can see that the regression line generated by the model with `fit_intercept = False` is anchored at Y = 0. The model is therefore constrained in terms of the lines it can generate. On the other hand, the line generated by the model with `fit_intercept = True` is not anchored at Y = 0, and therefore the model is able to determine the best value for the Y intercept as a result of the training process.

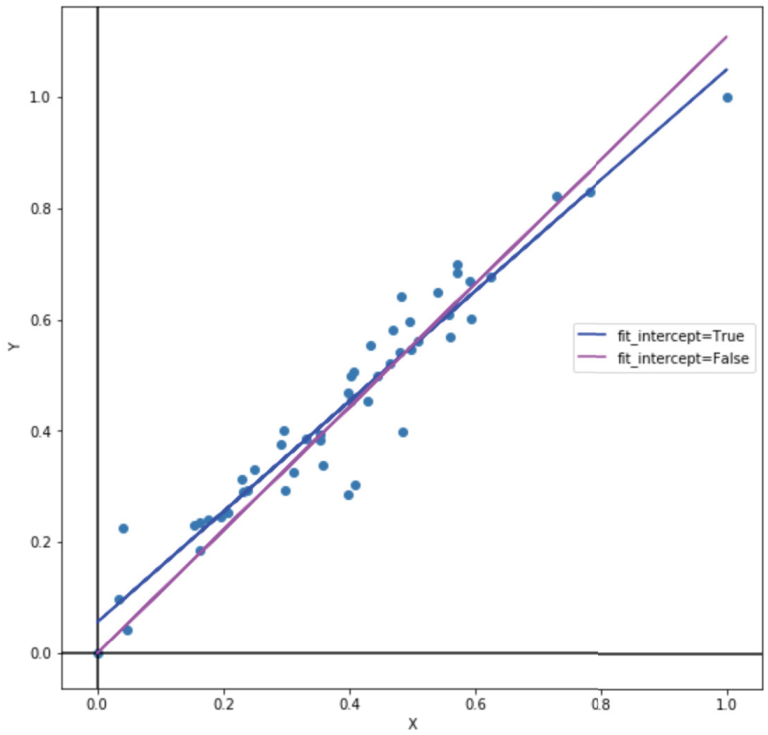


FIGURE 4.6
Scatter plot of synthetic dataset along with regression lines

Support Vector Machines

A support vector machine (SVM) is a versatile model that can be used for a variety of tasks, including classification, regression, and outlier detection. The original algorithm was invented in 1963 by Vladimir Vapnik and Alexey Chervonenkis as a binary classification algorithm. During the training process, support vector machine models aim to create a decision boundary that can partition the data points into classes. If the dataset has just two features, then this decision boundary is two-dimensional and can be conveniently represented in a scatter plot. If the decision boundary is linear, it will take the form of a straight line in two dimensions, a plane in three dimensions, and a hyperplane in n-dimensions. As humans, we cannot visualize more than three dimensions, which is why in order to understand how SVMs work, we'll consider a two-dimensional example with a fictional dataset with two features, and each point belonging to one of two classes. Let's also assume that the data is linearly separable—that is, you can draw a line that can separate them. Figure 4.7 depicts a scatter plot of feature values of points from this fictional dataset, with one class of observations represented as circles and the other as stars. The figure also presents three possible linear decision boundaries, each capable of separating the observations into two different sets.

The decision boundary to the left is too close to the first set of observations and there is a risk that a model with that decision boundary could misclassify real-world observations were they only slightly different from the training set. The decision boundary to the right has a similar problem in that it is too close to the second set of observations. The decision boundary in the middle is optimal because it is as far away as possible from both classes

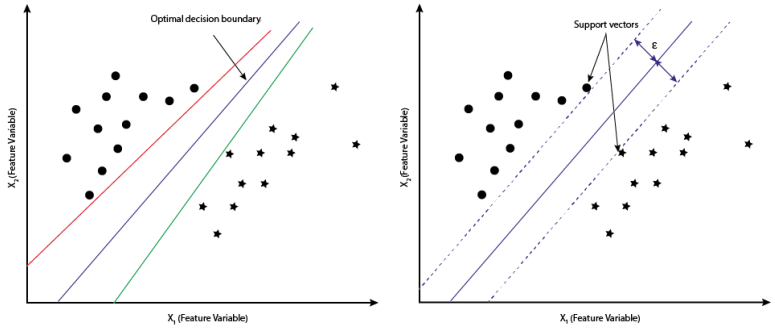


FIGURE 4.7
Three potential decision boundaries

and at the same time clearly separates both classes. SVM models aim to find this middle (optimal), or to put it in another way, aim to find the decision boundary that maximizes the distance between the two classes of observations on either side. The half-width of the margin is denoted by the Greek letter ε (epsilon). The points on the edges of the margin are called support vectors (the vector is assumed to originate at the origin and terminate at these points). In effect one could say these vectors are supporting the margins—hence the name support vectors.

Most real-world data is not linearly separable as the dataset depicted in Figure 4.7, and therefore linear decision boundaries are unable to clearly separate the classes. Furthermore, even when the data is linearly separable, there is a possibility that the margin of separation is not as wide as the fictional example in Figure 4.7. Data points are often too close together to allow for wide, clear margins between the decision boundary and the support vectors on either side, and to handle this, SVM implementations include the concept of a tolerance parameter that controls the number of support vectors that can be inside the margins, which in turn has an impact on the width of the margin. Setting a large tolerance value results in a wider margin with more samples in the margin, whereas setting a small tolerance value will result in a narrow margin. Having a wide margin is not necessarily a bad thing, as long as most of the points in the margin are on the correct side of the decision boundary.

Scikit-learn provides an implementation of support vector machine–based classifiers in the `SVC` class, which is part of the `sklearn.svm` module. We will now use this class to implement an SVM-based classification model on the popular Pima Indians diabetes dataset. The database consists of eight feature variables that represent various medical measurements such as blood pressure, plasma glucose concentration, BMI, and insulin levels, and contains a binary target variable called Outcome, which indicates whether the individual in question has diabetes. The dataset was originally created by the National Institute of Diabetes and Digestive and Kidney Diseases, and you can find the Kaggle version of the dataset at `https://www.kaggle.com/uciml/pima-indians-diabetes-database`. A copy of the dataset has been included with the resources that accompany this chapter.

The following snippet can be used to load the dataset from a CSV file, create Pandas data-frames with the feature and target data, normalize the feature data, and create a 75/25 test-train split:

```
# load Pima Indians Diabetes dataset
diabetes_dataset_file = './datasets/diabetes_dataset/diabetes.csv'
df_diabetes = pd.read_csv(diabetes_dataset_file)
df_diabetes_target = df_diabetes.loc[:,['Outcome']]
```

```
df_diabetes_features = df_diabetes.drop(['Outcome'], axis=1)
# normalize attribute values
from sklearn.preprocessing import MinMaxScaler
diabetes_scaler = MinMaxScaler()
diabetes_scaler.fit(df_diabetes_features)
nd_diabetes_features = diabetes_scaler.transform(df_diabetes_features)
df_diabetes_features_normalized = pd.DataFrame(data=nd_diabetes_features,
columns=df_diabetes_features.columns)
# create a training dataset and a test dataset using a 75/25 split.
diabetes_split = train_test_split(df_diabetes_features_normalized,
df_diabetes_target,
                                  test_size=0.25, random_state=17)
df_diabetes_features_train = diabetes_split[0]
df_diabetes_features_test = diabetes_split[1]
df_diabetes_target_train = diabetes_split[2]
df_diabetes_target_test = diabetes_split[3]
```

The following snippet creates an instance of the `SVC` class that attempts to find a linear decision boundary, trains the `SVC` instance using the training set (75% of the samples), and uses the `predict()` method to make predictions on the test set. You can learn more about instantiating an `SVC` instance at `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`.

```
# create an SVM classifier for the features of the diabetes dataset using a
linear kernel
from sklearn.svm import SVC
svc_model = SVC(kernel='linear', C=1)
svc_model.fit(df_diabetes_features_train, df_diabetes_target_train)
# use the SVC model to create predictions on the test set.
predicted_diabetes = svc_model.predict(df_diabetes_features_test)
```

Chapter 5 covers techniques to evaluate the performance of classification models, but for now you can examine the predictions themselves using the Python `print()` function:

```
print (predicted_diabetes)
[0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0
 0 0 0 0 1 0 0]
```

The real power of SVM-based classifiers is their ability to create nonlinear decision boundaries. Support vector models use a mathematical function called a kernel that is used to transform each input point into a higher-dimensional space where a linear decision boundary can be found. This will be easier to understand with an example. Figure 4.8 shows a scatter plot of another fictional dataset with two features per data point, and each data point belonging to one of two classes. In this case, it is quite clear that there is no linear decision boundary (straight line) that can classify all data points correctly—no matter which way you draw a straight line you will always end up with some samples on the wrong side of the line.

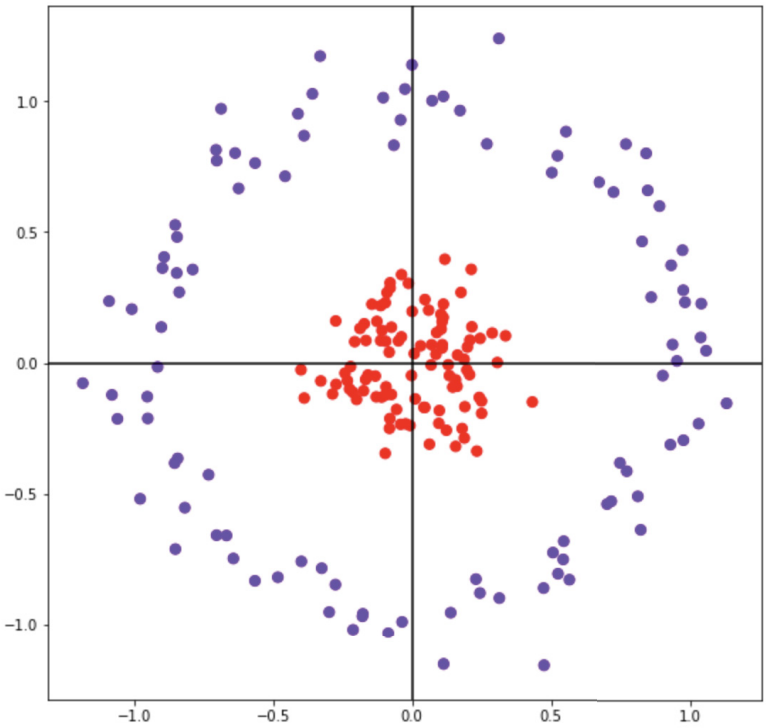


FIGURE 4.8
Data that cannot be classified using a linear decision boundary in two-dimensional space

If, however, you add an extra dimension (z-axis) to the data, and compute z values for each point using the equation `z = x² + y²`, then the data becomes linearly separable along the z-axis using a plane at `z = 0.3`. This is depicted in Figure 4.9.

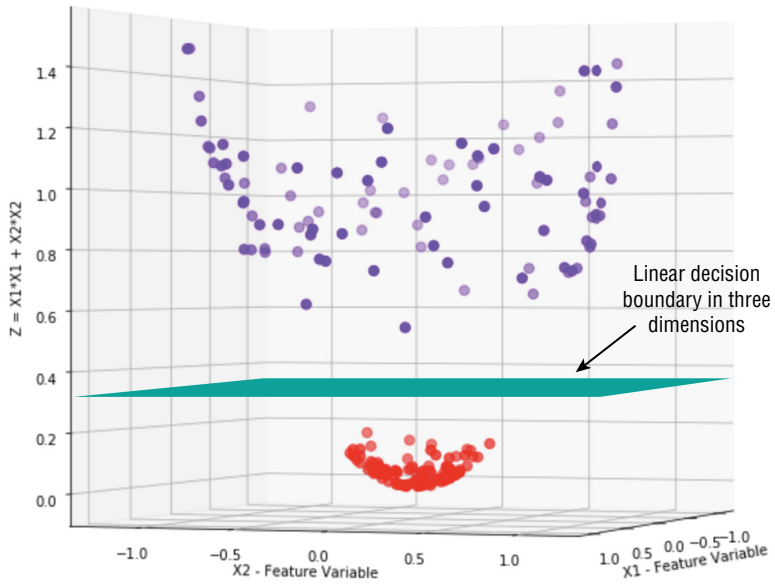


FIGURE 4.9
Data that cannot be classified using a linear decision boundary in two-dimensional space can be classified in three-dimensional space.

Since z was computed as $x^2 + y^2$, the decision plane at `z = 0.3` implies $x^2 + y^2 = 0.3$, which is nothing but the equation of a circle in two dimensions. Therefore, the linear decision boundary in three-dimensional space has become a nonlinear decision boundary in two-dimensional space. This is illustrated in Figure 4.10.

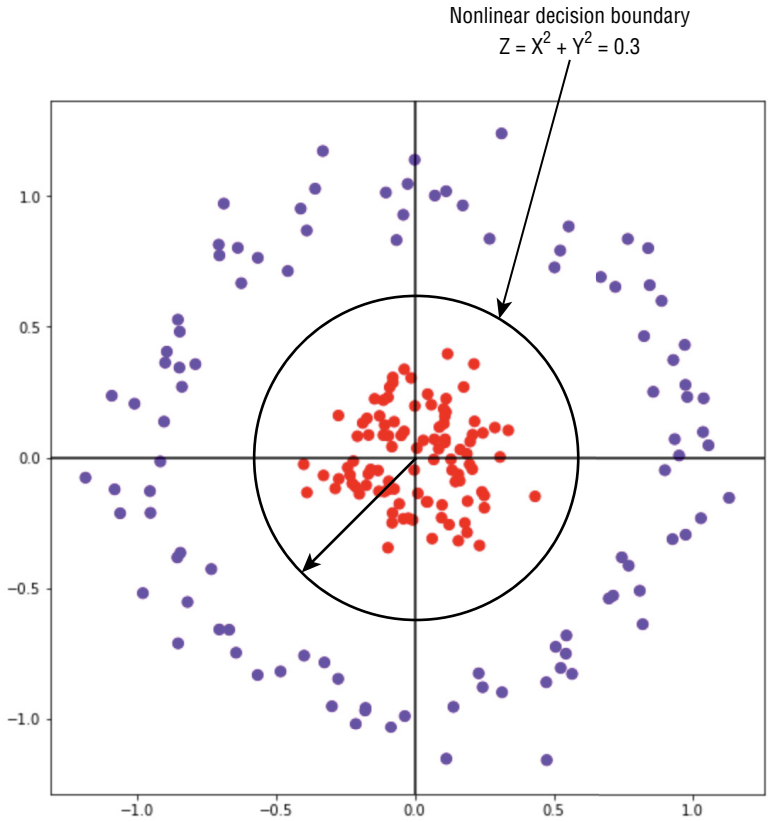


FIGURE 4.10 Nonlinear decision boundary in two-dimensional space

This is an oversimplification of how kernels work, and if you are interested in learning more about SVM kernels you should read *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* by Nello Cristianini and John Shawe-Taylor (`https://www.cambridge.org/core/books/an-introduction-to-support-vector-machines-and-other-kernelbased-learning-methods/A6A6F4084056A4B23F88648DDBFDD6FC`).

Scikit-learn allows you to choose from a number of common kernels when creating the SVC instance, including linear, polynomial, radial basis functions, and custom kernels. You can find out more about the different types of kernel functions at `https://scikit-learn.org/stable/modules/svm.html#svm-kernels`. In order to visualize the effect of different kernels, let's train multiple SVM classifiers with different kernel functions on a dataset. The following snippet trains multiple SVM-based classifiers on a synthetic binary classification dataset with two features. The first classifier uses a linear kernel, the second classifier uses a 2nd-degree polynomial kernel, the third classifier uses a 15th-degree polynomial kernel, and the fourth kernel uses a radial basis function (RBF) kernel:

```
# create a synthetic binary classification dataset with 2 features.
from sklearn.datasets import make_classification
Synthetic_BinaryClassX, Synthetic_BinaryClassY =
make_classification(n_samples=50, n_features=2, n_redundant=0, n_classes=2)

# scale synthetic dataset between -3, 3
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-3,3))
scaler.fit(Synthetic_BinaryClassX.reshape(-1,1))
Synthetic_BinaryClassX = scaler.transform(Synthetic_BinaryClassX)
# create multiple SVM classifiers
from sklearn.svm import SVC
svc_model_linear = SVC(kernel='linear', C=1, gamma='auto')
svc_model_polynomial2 = SVC(kernel='poly', degree=2, C=1, gamma='auto')
svc_model_polynomial15 = SVC(kernel='poly', degree=15, C=1, gamma='auto')
svc_model_rbf = SVC(kernel='rbf', C=1, gamma='auto')
svc_model_linear.fit(Synthetic_BinaryClassX, Synthetic_BinaryClassY)
svc_model_polynomial2.fit(Synthetic_BinaryClassX, Synthetic_BinaryClassY)
svc_model_polynomial15.fit(Synthetic_BinaryClassX, Synthetic_BinaryClassY)
svc_model_rbf.fit(Synthetic_BinaryClassX, Synthetic_BinaryClassY)
```

With the models created, we can use Matplotlib functions to plot the data points as well as decision boundaries of each classifier. The following snippet shows how to visualize the decision boundary of an SVM classifier (portions of the code are taken from `https://scikit-learn.org/stable/auto_examples/exercises/plot_iris_exercise.html`). The resulting plots are depicted in Figure 4.11.

```
#
# portions of this code are from
# source: https://scikit-learn.org/stable/auto_examples/svm/plot_iris.html
#
%matplotlib inline
import matplotlib.pyplot as plt
def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.
    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
def make_meshgrid(x, y, h=.02):
```

```
    """Create a mesh of points to plot in
    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional
    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy
# pick out 2 features X0 and X1
X0 = Synthetic_BinaryClassX[:,0]
X1 = Synthetic_BinaryClassX[:,1]
xx, yy = make_meshgrid(X0, X1, 0.02)#np.meshgrid(np.arange(-3, 3, 0.002),
np.arange(-3, 3, 0.002))

fig, axes = plt.subplots(2, 2, figsize=(16,16))
# plot linear kernel
plot_contours(axes[0,0], svc_model_linear,
              xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
axes[0,0].scatter(X0, X1, s=30,  c=Synthetic_BinaryClassY)
axes[0,0].set_xlim(xx.min(), xx.max())
axes[0,0].set_ylim(yy.min(), yy.max())
axes[0,0].set_title('Linear Kernel')

# plot 2nd degree polynomial kernel
plot_contours(axes[0,1], svc_model_polynomial2, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.8)
axes[0,1].scatter(X0, X1, s=30, c=Synthetic_BinaryClassY)
axes[0,1].set_xlim(xx.min(), xx.max())
axes[0,1].set_ylim(yy.min(), yy.max())
axes[0,1].set_title('2nd Degree Polynomial Kernel')
# plot 15 degree polynomial kernel
plot_contours(axes[1,0], svc_model_polynomial15, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.8)
axes[1,0].scatter(X0, X1, s=30, c=Synthetic_BinaryClassY)
axes[1,0].set_xlim(xx.min(), xx.max())
axes[1,0].set_ylim(yy.min(), yy.max())
axes[1,0].set_title('5th Degree Polynomial Kernel')
# plot RBF kernel
plot_contours(axes[1,1], svc_model_rbf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.8)
```

```
axes[1,1].scatter(X0, X1, s=30, c=Synthetic_BinaryClassY)
axes[1,1].set_xlim(xx.min(), xx.max())
axes[1,1].set_ylim(yy.min(), yy.max())
axes[1,1].set_title('RBF Kernel')
plt.show()
```

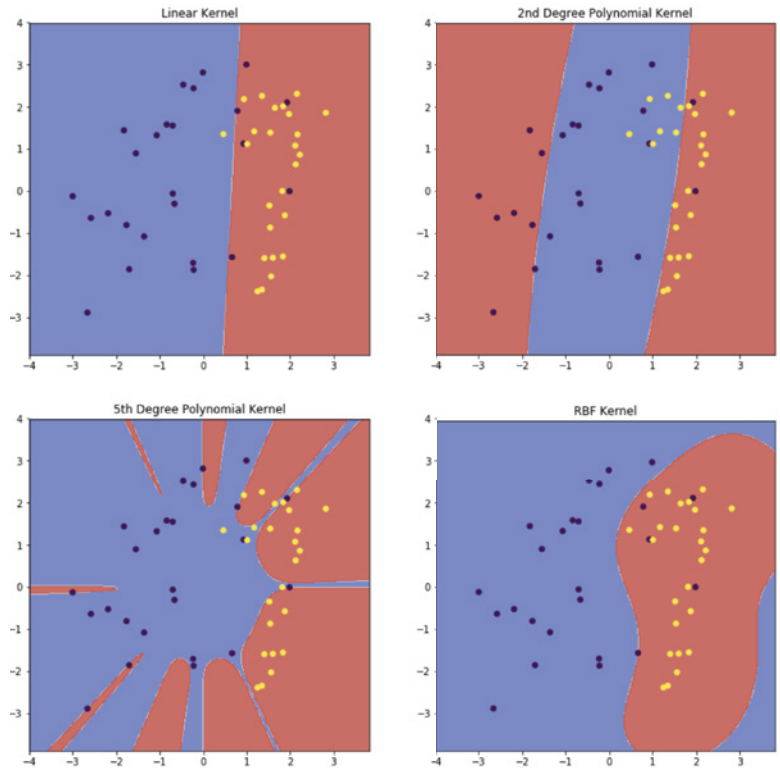


FIGURE 4.11
Effect of kernel choice on decision boundaries

Support vector machine–based models can also be used for linear regression tasks. When used for linear regression tasks, you are no longer looking for a decision boundary that can spilt the points between classes, but instead the line (or hyperplane) that best fits the samples. Support vector regression (SVR) is a technique that attempts to find the best line, or hyperplane, that fits the training variables. The difference between linear regression and SVR is the manner in which this hyperplane is determined. Linear regression in two dimensions fundamentally attempts to find the line that minimizes the sum of the distances of the data points from the line. SVR, on the other hand, attempts to find the line that contains the largest number of data points within a fixed distance from the hyperplane. Figure 4.12 illustrates the difference between linear regression and SVR in a two-dimensional scenario.

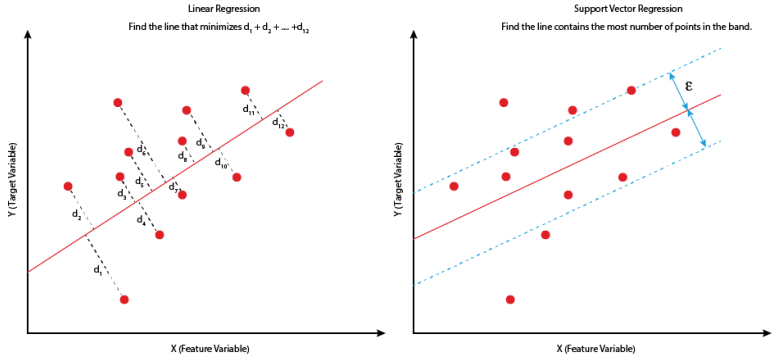


FIGURE 4.12
Linear regression vs. support vector regression

Scikit-learn provides an implementation of support vector machine–based regressors in the `SVR` class, which is part of the `sklearn.svm` module. The following snippet will use the `SVR` class to implement an SVM-based regression model on the Boston housing prices dataset to predict the median house price. A scatter plot of the predicted vs. actual house prices is presented in Figure 4.13.

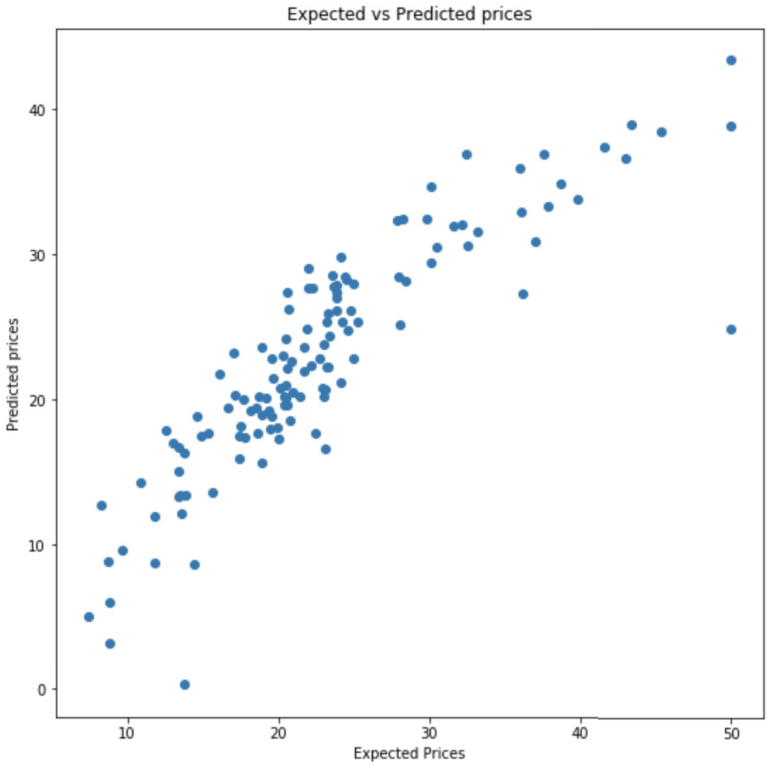


FIGURE 4.13
SVR predictions on Boston housing dataset

```
# train a linear model on the Boston house prices dataset.
from sklearn.svm import SVR
svr_model = SVR(kernel='linear', C=1.5, gamma='auto', epsilon=1.5)
svr_model.fit(df_boston_features_train, df_boston_target_train.values.ravel())
# use the SVR model to create predictions on the test set.
svr_predicted_prices = svr_model.predict(df_boston_features_test)
%matplotlib inline
import matplotlib.pyplot as plt
# use pyplot module to create a scatter plot of predicted vs expected values
fig, axes = plt.subplots(1, 1, figsize=(9,9))
plt.scatter(df_boston_target_test, predicted_median_house_prices)
plt.xlabel("Expected Prices")
plt.ylabel("Predicted prices")
plt.title("Expected vs Predicted prices")
```

Logistic Regression

Logistic regression, despite having the word *regression* in its name, is a technique that can be used to build binary and multi-class classifiers. Logistic regression (also known as logit regression) builds upon the output of linear regression and returns a probability that the data point is of one class or another. Recall that the output of linear regression is a continuous unbounded value, whereas probabilities are continuous bounded values—bounded between 0.0 and 1.0.

In order to use a continuous value for binary classification, logistic regression converts it into a probability value between 0.0 and 1.0 by feeding the output of linear regression into a logistic function. In statistics a logistic function is a type of function that converts values from [–infinity, + infinity] to [0, 1]. In the case of logistic regression, the logistic function is the sigmoid function, which is defined as:

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$$

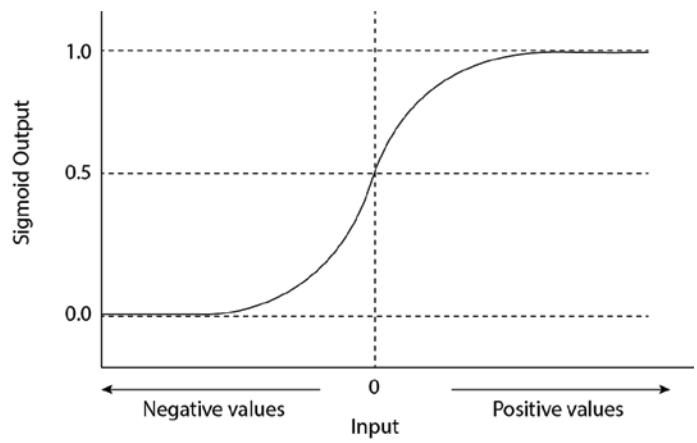


FIGURE 4.14
The sigmoid function

The graph of the sigmoid function is presented in Figure 4.14. The output of the sigmoid function will never go below 0.0 or above 1.0, regardless of the value of the input.

The output of the sigmoid function can be used for binary classification by setting a threshold value and treating all values below that as class A and everything above the threshold as class B (Figure 4.15).

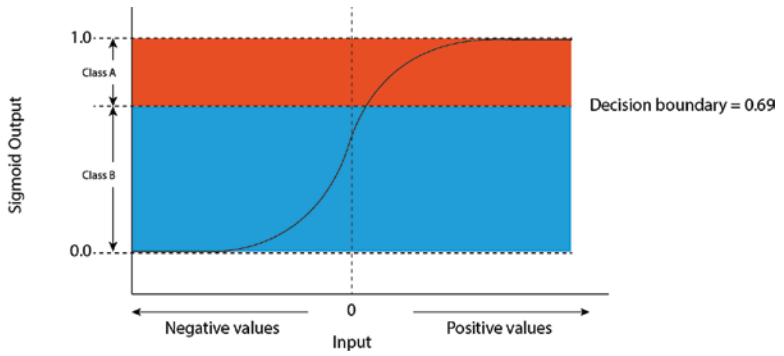


FIGURE 4.15
Using the sigmoid function for binary classification

Scikit-learn provides the `LogisticRegression` class as part of the `linear_model` module. We will now use this class to implement a logistic regression–based binary classification model on the popular Pima Indians diabetes dataset:

```
# load Pima Indians Diabetes dataset
diabetes_dataset_file = './datasets/diabetes_dataset/diabetes.csv'
df_diabetes = pd.read_csv(diabetes_dataset_file)
df_diabetes_target = df_diabetes.loc[:,['Outcome']]
df_diabetes_features = df_diabetes.drop(['Outcome'], axis=1)
# normalize attribute values
from sklearn.preprocessing import MinMaxScaler
diabetes_scaler = MinMaxScaler()
diabetes_scaler.fit(df_diabetes_features)
nd_diabetes_features = diabetes_scaler.transform(df_diabetes_features)
df_diabetes_features_normalized = pd.DataFrame(data=nd_diabetes_features,
columns=df_diabetes_features.columns)
# create a training dataset and a test dataset using a 75/25 split.
diabetes_split = train_test_split(df_diabetes_features_normalized,
df_diabetes_target,
                                  test_size=0.25, random_state=17)
df_diabetes_features_train = diabetes_split[0]
df_diabetes_features_test = diabetes_split[1]
df_diabetes_target_train = diabetes_split[2]
df_diabetes_target_test = diabetes_split[3]
# train a logistic regression model on the diabetes dataset.
from sklearn.linear_model import LogisticRegression
logistic_regression_model = LogisticRegression(penalty='l2', fit_intercept=True,
solver='liblinear')
logistic_regression_model.fit(df_diabetes_features_train, df_diabetes_target_
train.values.ravel())
# use the  model to create predictions on the test set, with a threshold of 0.5
logistic_regression_predictions =
logistic_regression_model.predict(df_diabetes_features_test)
```

The constructor of the `LogisticRegression` class takes several parameters, some of which are common with the `LinearRegression` class. You can find out more about these parameters at `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`. The predictions made by the model can be examined using the Python `print()` function:

```
print (logistic_regression_predictions)
[0 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1
 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0
 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1
 0 0 0 0 1 0 0]
```

The binary predictions are made using a probability cut-off of 0.5. If the probability value estimated by the underlying linear regression model is > 0.5, the output class will be 0. Scikit-learn does not allow you to change this probability cut-off; however, you can use the `predict_proba()` method of the `LogisticRegression` instance to access the prediction probabilities before the thresholding operation was applied:

```
# access class-wise probabilities
logistic_regression_probabilities =
logistic_regression_model.predict_proba(df_diabetes_features_test)
```

Since there are two output classes, the `predict_proba()` method will give you two probabilities per data point. The first column contains the probability that the point will be labeled 0, and the second column contains the probability that the point will be labeled 1:

```
print (logistic_regression_probabilities)
[[0.85694005 0.14305995]
 [0.37165061 0.62834939]
 [0.73695232 0.26304768]
 [0.880803   0.119197  ]
...
...
[0.41292724 0.58707276]
 [0.63547121 0.36452879]
 [0.52728275 0.47271725]]
```

Because these numbers represent probabilities, the sum of the prediction probabilities for any data point will be 1.0. Furthermore, since there are only two classes, you can use the information in any one column to work out the value of the other column by subtracting from 1.0. The following snippet uses the information in the first column (probability that the output class is 0) and implements custom thresholding logic at 0.8. Any probabilities greater than 0.8 will be labeled 0:

```
# implement custom thresholding logic
dfProbabilities = pd.DataFrame(logistic_regression_probabilities[:,0])
predictions = dfProbabilities.applymap(lambda x: 0 if x > 0.8 else 1)
```

You can examine the predictions with this new threshold of 0.8 by printing the contents of predictions. Compare these predictions with the predictions made by the model with Scikit-learn's default cut-off threshold of 0.5:

```
print (predictions.values.ravel())
[0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 1 0 1
 1 0 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0
 0 1 0 1 1 1 1 1 1 0 1 1 1 1 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0
 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1
 1 1 1 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1
 1 1 1 1 1 1 1]
```

As mentioned earlier, logistic regression builds upon the output of linear regression. You can inspect the coefficients and intercept of the underlying linear model through the `coef_` and `intercept_` attributes of the model:

```
print (logistic_regression_model.coef_)
[[ 1.48972976  3.4891602  -0.7344297  -0.07461329  0.16776565  1.81409369
   1.39383873  1.03554067]]
print (logistic_regression_model.intercept_)
[-4.06714158]
```

Logistic regression is inherently a binary classifier, but it can be used as a multi-class classifier for datasets where the target variable can belong to more than two classes. There are two fundamental approaches that can be used to use a binary classifier for multi-class problems:

- *One-versus-rest approach:* This is also known as the OVR approach, and it involves creating a number of binary classification models, with each model predicting the probability that the output is one of the subclasses. This approach will create N models for N classes, and the final class output by the multi-class classifier corresponds to the model that predicted the highest probability. Consider the popular Iris flowers dataset, where the target variable can have one of three values [0, 1, 2], corresponding to the type of Iris flower. In this case, the OVR approach would involve training three logistic regression models. The first model would predict the probability that the output class is 0 or not 0. Likewise, the second model would only predict the probability that the output class is 1 or not 1, and so on. The one-versus-rest approach is sometimes also referred to as the one-versus-all (OVA) approach
- *One-versus-one approach:* This is known as the OVO approach, and it also involves creating a number of binary classification models and picking the class that corresponds to the model that outputs the largest probability value. The difference between the OVO approach and the OVR approach is in the number of models created. The OVO approach creates one model for each pairwise combination of output classes. In the case of the Iris flowers dataset, the OVO approach would also create three models:
  - Logistic regression model that predicts output class as 0 or 1
  - Logistic regression model that predicts output class as 0 or 2
  - Logistic regression model that predicts output class as 1 or 2

As you can see, the number of models generated increases with the number of features.

Scikit-learn provides the `OneVsOneClassifier` and the `OneVsRestClassifier` classes in the `multiclass` module that encapsulate the complexity of creating multiple binary classification models and training them. The constructor for these classes takes a binary classification model as input. Some model classes within Scikit-learn are inherently capable of multi-class classification, and you may be surprised to learn that `LogisticRegression` is one of them. However, before we discuss the implementation of inherent multi-class classification in the `LogisticRegression` class, let's examine how we can use the `OneVsRestClassifier` to create an ensemble of binary `LogisticRegression` models and use the ensemble as a multi-class classifier. You can learn more about the classes in the `multiclass` package at `https://scikit-learn.org/stable/modules/multiclass.html`.

The following snippet demonstrates using the `OneVsRestClassifier` class to create a multi-class classifier from an ensemble of binary `LogisticRegression` models on the Iris flowers dataset:

```
# load Iris flowers dataset
from sklearn.datasets import load_iris
iris_dataset = load_iris()
df_iris_features = pd.DataFrame(data = iris_dataset.data,
columns=iris_dataset.feature_names)
df_iris_target = pd.DataFrame(data = iris_dataset.target, columns=['class'])
# normalize attribute values
from sklearn.preprocessing import MinMaxScaler
iris_scaler = MinMaxScaler()
iris_scaler.fit(df_iris_features)
nd_iris_features = iris_scaler.transform(df_iris_features)
df_iris_features_normalized = pd.DataFrame(data=nd_iris_features, columns=df_
iris_features.columns)
# create a training dataset and a test dataset using a 75/25 split.
from sklearn.model_selection import train_test_split
iris_split = train_test_split(df_iris_features_normalized, df_iris_target,
                              test_size=0.25, random_state=17)
df_iris_features_train = iris_split[0]
df_iris_features_test = iris_split[1]
df_iris_target_train = iris_split[2]
df_iris_target_test = iris_split[3]
# implement multi-class classification using
# OVA (a.ka. OVR) approach and LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
logit_model = LogisticRegression(penalty='l2', fit_intercept=True,
solver='liblinear')
ovr_logit_model = OneVsRestClassifier(logit_model)
ovr_logit_model.fit(df_iris_features_train, df_iris_target_train.values.ravel())
# use the  model to create predictions on the test set, with a threshold of 0.5
ovr_logit_predictions = ovr_logit_model.predict(df_iris_features_test)
```

You can inspect the classes predicted by the OVR logistic regression model by using the Python `print()` function. As you can see, the output classes predicted for the members of the test set are one of three values—0, 1, or 2:

```
print (ovr_logit_predictions)
[0 2 2 1 2 2 2 2 1 2 2 0 1 0 2 0 0 2 2 2 2 0 2 1 2 2 1 1 0 1 0 1 0 0 1 2 1
 2]
```

You can inspect the class-wise probabilities from the `OneVsRestClassifier` instance by using the `predict_proba()` method, just as you did earlier with the `LogisticRegression` instance. This time, though, the result will have three values for each member of the test set. The first number is the probability that the output class is 0, the second number is the probability that the output class is 1, and so on:

```
# access class-wise probabilities
ovr_logit_probs = ovr_logit_model.predict_proba(df_iris_features_test)
print(ovr_logit_probs)
[[0.82272514 0.12785864 0.04941622]
 [0.12044579 0.40056122 0.47899299]
 [0.02542865 0.32329645 0.6512749 ]
 [0.18305903 0.42111625 0.39582472]
 [0.05944138 0.38763397 0.55292465]
 [0.07236737 0.36312485 0.56450777]
 [0.16344427 0.37963956 0.45691617]
 [0.01998424 0.24601841 0.73399734]
 [0.18950936 0.48395363 0.32653701]
 [0.03663432 0.40209894 0.56126674]
 [0.02062532 0.27783051 0.70154417]
 [0.73577162 0.22066942 0.04355896]
 [0.15270279 0.42746281 0.41983439]
 [0.77216659 0.18251154 0.04532187]
 [0.05309898 0.32231709 0.62458393]
 [0.815817   0.13825926 0.04592374]
 [0.73489217 0.22191513 0.0431927 ]
 [0.04491288 0.36458749 0.59049964]
 [0.02065056 0.27871118 0.70063826]
 [0.02127991 0.35388486 0.62483523]
 [0.07152985 0.41695375 0.5115164 ]
 [0.7706894  0.18349734 0.04581325]
 [0.07040028 0.36307885 0.56652087]
 [0.19267192 0.4727485  0.33457958]
 [0.15280003 0.38212573 0.46507424]
 [0.17395557 0.31901921 0.50702523]
 [0.12736739 0.48820204 0.38443056]
 [0.13568065 0.44198711 0.42233224]
 [0.7867313  0.16963785 0.04363084]
 [0.17115366 0.45770086 0.37114548]
 [0.74540203 0.20735953 0.04723843]
 [0.31041971 0.43132172 0.25825857]
```

```
[0.80839308 0.15516489 0.03644203]
[0.80848648 0.13549109 0.05602242]
[0.21762134 0.48521286 0.29716579]
[0.15584948 0.41625218 0.42789834]
[0.19201639 0.40706352 0.4009201 ]
[0.03199536 0.34175085 0.62625378]]
```

While training an ensemble of binary models is one way to build models capable of multi-class classification, some algorithms like logistic regression can be modified to inherently support multi-class classification. In the case of logistic regression, the modification involves training multiple linear regression models internally, and replacing the sigmoid function with another function—the softmax function. The softmax function is capable of receiving inputs from multiple linear regression models and outputting class-wise probabilities. The softmax function is also known as the normalized exponential function, and its equation is illustrated in Figure 4.16.

To understand how this function works, let's consider the Iris flowers dataset. Each row of the dataset contains four continuous numeric attributes and a multi-class target with three possible output classes: 0, 1, 2. When a softmax logistic regression model is trained on this dataset, it will contain three linear regression models within it, one for each target class. When the model is used for making predictions, each linear regression model will output a continuous numeric value that will be fed into the softmax function, which will in turn output three class-wise probabilities.

Scikit-learn's implementation of the `LogisticRegression` class is inherently capable of multinomial classification; all you need to do is include the `multi_class = 'multinomial'` and `solver = 'lbfgs'` constructor arguments while instantiating the class. The following snippet uses Scikit-learn's `LogisticRegression` class on a multi-class classification problem with softmax regression:

```
# implement multi-class classification using
# softmax (a.k.a multinomial regression) classifier
from sklearn.linear_model import LogisticRegression
softmax_logit_model = LogisticRegression(penalty='l2', fit_intercept=True,
solver='lbfgs', multi_class='multinomial')
softmax_logit_model.fit(df_iris_features_train,
df_iris_target_train.values.ravel())
# use the  model to create predictions on the test set
softmax_logit_predictions = softmax_logit_model.predict(df_iris_features_test)
```

You can inspect the classes predicted by the softmax logistic regression model by using the Python `print()` function. As you can see, the output classes predicted for the members of the test set are one of three values—0, 1, or 2:

```
print (softmax_logit_predictions)
[0 1 2 1 2 2 1 2 1 2 2 0 1 0 2 0 0 2 2 2 1 0 2 1 1 2 1 1 0 1 0 1 0 0 1 1 1
 2]
```

You can inspect the class-wise probabilities by calling the `predict_proba()` method on the `LogisticRegression` instance. As you would expect, the result has three values for each member of the test set. The first number is the probability that the output class is 0, the second number is the probability that the output class is 1, and so on (see the code example on page 109).

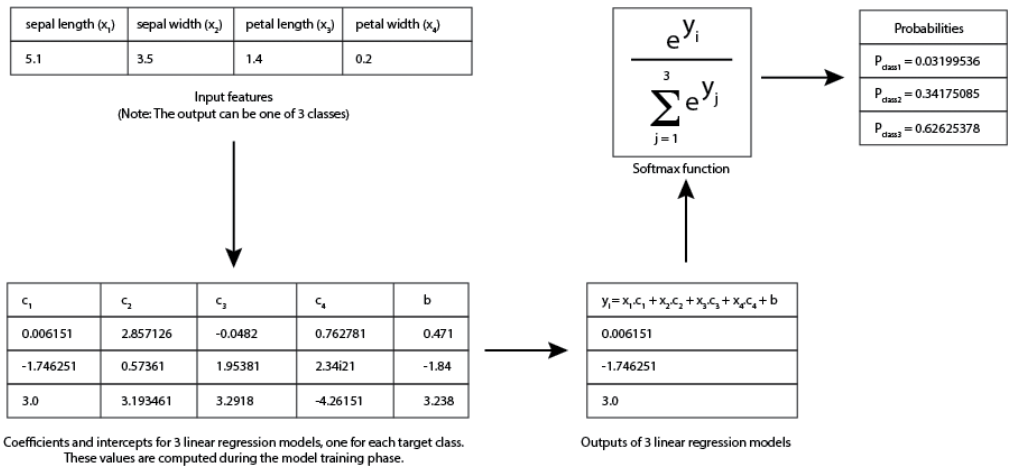


FIGURE 4.16 Softmax logistic regression

```
# access class-wise probabilities
softmax_logit_probs = softmax_logit_model.predict_proba(df_iris_features_test)
print(softmax_logit_probs)
[[0.89582633 0.09444564 0.00972803]
 [0.09889138 0.50828121 0.39282741]
 [0.01311439 0.23998685 0.74689876]
 [0.1645445  0.56290434 0.27255115]
 [0.04525701 0.42873174 0.52601125]
 [0.05219811 0.40090084 0.54690105]
 [0.1412285  0.49318467 0.36558684]
 [0.00512255 0.1226479  0.87222954]
 [0.18517246 0.6381944  0.17663314]
 [0.0301524  0.40920296 0.56064464]
 [0.00767857 0.15588852 0.83643291]
 [0.85228205 0.14095824 0.00675971]
 [0.1344201  0.56001268 0.30556722]
 [0.87716333 0.11550045 0.00733622]
 [0.03095013 0.30494959 0.66410028]
 [0.89925442 0.09260716 0.00813842]
 [0.86906457 0.12521241 0.00572302]
 [0.0286017  0.35588746 0.61551083]
 [0.00777372 0.15176958 0.84045669]
 [0.01219112 0.26741715 0.72039173]
 [0.05557848 0.49114599 0.45327553]
 [0.86733298 0.12475638 0.00791064]
 [0.05107527 0.39427714 0.55464759]
 [0.18367433 0.62661521 0.18971046]
 [0.1288656  0.49409071 0.37704369]
 [0.12588161 0.41550633 0.45861205]
 [0.12692498 0.64014023 0.23293479]
 [0.11348037 0.57595155 0.31056808]
 [0.85890116 0.13346798 0.00763086]
 [0.32189697 0.55802372 0.12007931]
 [0.91075124 0.08432501 0.00492375]
 [0.87466417 0.11230055 0.01303528]
 [0.21216432 0.63742445 0.15041124]
 [0.12872766 0.55202598 0.31924635]
 [0.16895463 0.54601673 0.28502864]
 [0.01604996 0.28654283 0.69740721]]
```

Decision Trees

Decision trees are, as their name suggests, tree-like structures where each parent node represents a decision boundary and child nodes represent outcomes of the decision. The topmost node of the tree is known as the root node. Building a decision tree model involves picking a suitable attribute for the decision at the root node, and then recursively partitioning the tree into nodes until some optimal criteria are met.

Decision trees are very versatile and can be used for both classification and regression tasks. When used for classification tasks, they are inherently capable of handling multi-class problems and are not affected by the scale of individual features. Predictions made by decision trees also have the advantage of being easy to explain—all you need to do is traverse the nodes of the decision tree and you will be able to explain the prediction. This is not the case for models such as neural networks, where it is not possible to explain why the model predicts something. Models such as decision trees that allow you to easily understand the reasoning behind a prediction are called white-box models, whereas models such as neural networks that do not provide the ability to explain a prediction are called black-box models.

Scikit-learn provides the `DecisionTreeClassifier` class as part of the `tree` package. We will now use this class to implement a decision tree–based multi-class classification model on the popular Iris flowers dataset:

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
df_iris_features = pd.DataFrame(data = iris_dataset.data,
columns=iris_dataset.feature_names)
df_iris_target = pd.DataFrame(data = iris_dataset.target, columns=['class'])
# normalize attribute values
from sklearn.preprocessing import MinMaxScaler
iris_scaler = MinMaxScaler()
iris_scaler.fit(df_iris_features)
nd_iris_features = iris_scaler.transform(df_iris_features)
df_iris_features_normalized = pd.DataFrame(data=nd_iris_features,
columns=df_iris_features.columns)
# create a training dataset and a test dataset using a 75/25 split.
from sklearn.model_selection import train_test_split
iris_split = train_test_split(df_iris_features_normalized, df_iris_target,
                              test_size=0.25, random_state=17)
df_iris_features_train = iris_split[0]
df_iris_features_test = iris_split[1]
df_iris_target_train = iris_split[2]
df_iris_target_test = iris_split[3]
# create a decision tree based multi-class classifier.
from sklearn.tree import DecisionTreeClassifier
dtree_model = DecisionTreeClassifier(max_depth=4)
dtree_model.fit(df_iris_features_train, df_iris_target_train.values.ravel())
# use the  model to create predictions on the test set
dtree_predictions = dtree_model.predict(df_iris_features_test)
```

The constructor of the `DecisionTreeClassifier` class takes several parameters, most of which begin with `max_` or `min_` and are used to enforce constraints on the decision tree. Unlike other model types, decision trees do not have any inherent form of regularization and will aim to fit the training data near-perfectly. The problem with this is that decision tree models are likely to overfit the training data, and the way to prevent overfitting is to enforce constraints on the tree-building process such as the maximum depth of the tree, minimum number of samples in a leaf node, etc. You can find out more about the constructor parameters at `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`. The predictions made by the model can be examined using the Python `print()` function:

```
print (dtree_predictions)
[0 1 2 1 2 2 1 2 1 2 2 0 1 0 2 0 0 2 2 2 1 0 2 1 1 1 1 1 0 1 0 1 0 0 1 1 1
 2]
```

To visualize the decision tree you will first need to use the `sklearn.tree.export_graphviz()` function to export the nodes of the tree into the Graphviz DOT file format, and then use Graphviz functions to convert the DOT file into an image. You can learn more about Graphviz at `https://pydotplus.readthedocs.io/reference.html`. The following snippet can be used to create a graph from a decision tree classifier. The resulting graph is depicted in Figure 4.17.

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(dtree_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

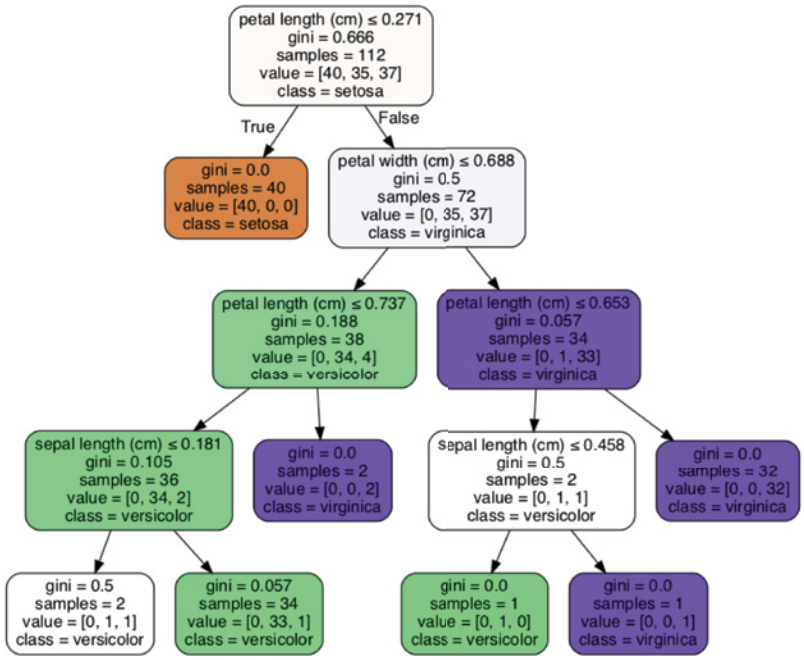


FIGURE 4.17
Decision tree visualization

To make predictions with this decision tree, you start with the condition on the root node: `petal length <= 0.271`. There are two branches from this node—the branch on the left should be traversed if the condition is met, and the branch on the right is traversed if the condition is not met. You then repeat this process until you reach a leaf node, and the class associated with the leaf node will be the prediction.

The root node also contains some additional information: `samples=112` implies that there are 112 samples in the dataset that this condition applies to. The value array implies that out of 112 samples, 40 belong to the first class, 35 to the second class, and 37 to the third class. The `gini=0.666` value represents the Gini score associated with the node. A Gini score is a measure of impurity and is one of two impurity measures that Scikit-learn's implementation of decision trees provides, the other one being Entropy. A pure node is one which has elements that all belong to the same class, and a Gini score of 0.0. During the model-building process, the decisions that form the nodes (such as `petal_length <= 0.271`) are chosen so as to create the purest subsets on both child nodes. Tree building is a recursive process and stops when either the Gini score associated with a node is 0.0 or a criterion such as maximum permissible depth of the tree has been reached. You can learn more about Gini scores in *The Gini Methodology: A Primer on Statistical Methodology* by Shlomo Yitzhaki and Edna Schechtman (`https://www.springer.com/gb/book/9781461447191`).

A decision tree can also be used for regression problems, and Scikit-learn provides the `DecisionTreeRegressor` class to create decision trees for regression. A decision tree for regression is very similar to a tree used for classification, with the key difference being that each node predicts a numeric value instead of a class. The following snippet uses the `DecisionTreeRegressor` class to create a decision tree on the Boston housing dataset and uses the model to predict median house prices for the members of the test set. Figure 4.18 contains the decision tree generated by the model.

```
# create a decision tree based regressor on the Boston housing dataset.
from sklearn.tree import DecisionTreeRegressor
dtree_reg_model = DecisionTreeRegressor(max_depth=4)
dtree_reg_model.fit(df_boston_features_train,
df_boston_target_train.values.ravel())
# use the  model to create predictions on the test set
dtree_reg_predictions = dtree_reg_model.predict(df_boston_features_test)
```

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from `www.wiley.com/go/machinelearningawscloud` or from GitHub using the following URL:

`https://github.com/asmtechnology/awsmlbook-chapter4.git`

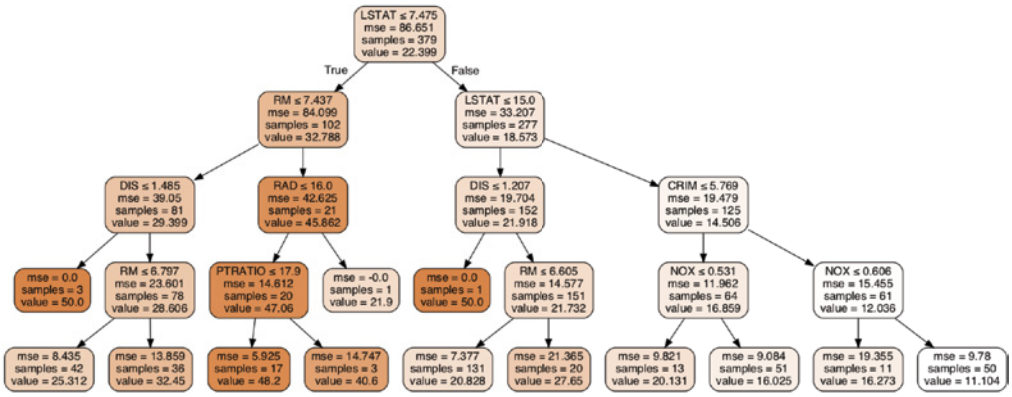


Figure 4.18
Decision tree for regression

Summary

- Scikit-learn is a Python library that provides a number of features that are suitable for machine learning engineers and data scientists.
- Scikit-learn provides a function called `train_test_split()` in the `model_selection` module that can be used to split a Pandas dataframe into two dataframes, one for model building and the other for model evaluation.
- K-fold cross validation can help minimize the possibility of the model picking up on unexpected bias in the training set.
- Linear regression is a statistical technique that aims to find the equation of a line (or hyperplane) that is closest to all the points in the dataset.
- Scikit-learn implements linear regression in a class called `LinearRegression`, which is part of the `linear_model` module.
- A support vector machine (SVM) is a versatile model that can be used for a variety of tasks, including classification, regression, and outlier detection.
- Scikit-learn provides an implementation of support vector machine–based classifiers in the, `SVC` class, which is part of the `sklearn.svm` module.
- Support vector models use a mathematical function called a kernel that is used to transform each input point into a higher-dimensional space where a linear decision boundary can be found.
- Logistic regression, despite having the word *regression* in its name, is a technique that can be used to build binary and multi-class classifiers.
- Scikit-learn provides the `LogisticRegression` class as part of the `linear_model` module.
- Logistic regression is inherently a binary classifier, but it can be used as a multi-class classifier for datasets where the target variable can belong to more than two classes.
- Decision trees are very versatile and can be used for both classification and regression tasks. When used for classification tasks they are inherently capable of handling multi-class problems and are not affected by the scale of individual features.

Chapter 5

Evaluating Machine Learning Models

WHAT'S IN THIS CHAPTER

- Learn how to evaluate the performance of regression models
- Learn how to evaluate the performance of classification models
- Learn to use the grid-search technique to choose the optimal set of hyperparameters for your model

In the previous chapter, you learned how to use Scikit-learn to create different types of machine learning models. In this chapter you will learn to use Scikit-learn to evaluate the performance of the models you have trained and techniques to select the values of hyperparameters that will result in an optimal model.

Since the purpose of a machine learning model is to predict something correctly, you will want to ensure that the predictive accuracy of your model is good enough for you to deploy it into production. It is therefore important to evaluate the performance of the model on data that the model has not seen previously, so as to get an accurate picture of how the model is likely to perform on real-world data (which it will also not have seen previously). Techniques such as creating test-train splits and k-fold cross validation, both of which have been discussed in Chapter 4, allow you to keep aside some of the training data for evaluation.

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from `www.wiley.com/go/machinelearning-awscloud` or from GitHub using the following URL:

```
https://github.com/asmtechnology/awsmlbook-chapter5.git
```

Evaluating Regression Models

The purpose of a linear regression model is to predict a continuous numeric value, such as a house price. There are two types of techniques that can be used to evaluate the predictive accuracy of a regression model: creating a scatter plot of the true and predicted values and computing a statistical metric that captures the total prediction error across members of the test set. The visual results obtained from a 2D scatter plot are simple to understand: the x-axis

contains the actual values, and the y-axis contains the predicted values. The closer the points are to a 45-degree line anchored at the origin, the better the prediction will be.

The following snippet uses Scikit-learn to load the Boston housing prices dataset and train a linear regression model and a decision tree–based model on the data. A scatter plot of the actual vs. predicted value for both models is presented side-by-side in Figure 5.1.

```
import numpy as np
import pandas as pd

# load boston house prices dataset
from sklearn.datasets import load_boston
boston_dataset = load_boston()
df_boston_features = pd.DataFrame(data = boston_dataset.data, columns=boston_
dataset.feature_names)
df_boston_target = pd.DataFrame(data = boston_dataset.target, columns=['price'])

# create a training dataset and a test dataset using a 75/25 split.
from sklearn.model_selection import train_test_split

boston_split = train_test_split(df_boston_features, df_boston_target,
                                test_size=0.25, random_state=17)
df_boston_features_train = boston_split[0]
df_boston_features_test = boston_split[1]
df_boston_target_train = boston_split[2]
df_boston_target_test = boston_split[3]

# train a linear model on the Boston house prices dataset.
from sklearn.linear_model import LinearRegression
linear_reg_model = LinearRegression(fit_intercept=True)
linear_reg_model.fit(df_boston_features_train, df_boston_target_train)

# create a decision tree based regressor on the Boston house prices dataset.
from sklearn.tree import DecisionTreeRegressor
dtree_reg_model = DecisionTreeRegressor(max_depth=10)
dtree_reg_model.fit(df_boston_features_train, df_boston_target_train.
values.ravel())

# use the  models to create predictions on the test set
linear_reg_predictions = linear_reg_model.predict(df_boston_features_test)
dtree_reg_predictions = dtree_reg_model.predict(df_boston_features_test)

# create a scatter plot of predicted vs actual values

%matplotlib inline
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(18,9))
axes[0].scatter(df_boston_target_test, linear_reg_predictions)
```

```
axes[0].set_xlabel("Expected Prices")
axes[0].set_ylabel("Predicted prices")
axes[0].set_title("Linear Regression Model")

axes[1].scatter(df_boston_target_test, dtree_reg_predictions)
axes[1].set_xlabel("Expected Prices")
axes[1].set_ylabel("Predicted prices")
axes[1].set_title("Decision Tree Model")

# plot the ideal prediction line
IdealPrices = np.linspace(0.0, df_boston_target_test.values.max(), 50)
IdealPredictions = IdealPrices
axes[0].plot(IdealPrices, IdealPredictions, color='#ff0000', label='ideal
prediction line')
axes[1].plot(IdealPrices, IdealPredictions, color='#ff0000', label='ideal
prediction line')

axes[0].legend()
axes[1].legend()
```

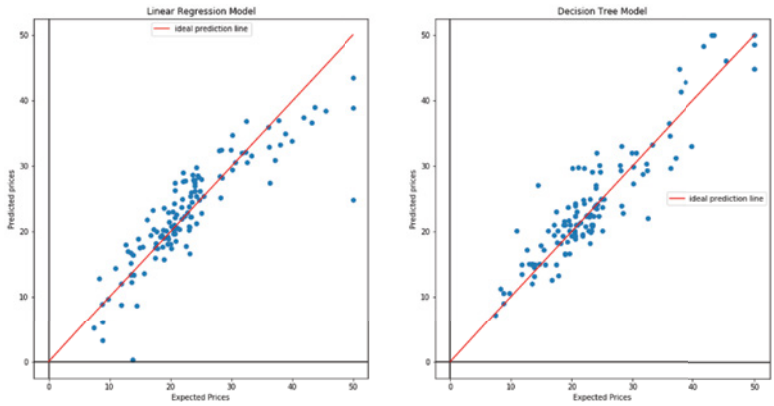


FIGURE 5.1
Comparison of predictive accuracies of a linear regression model and decision tree model on the Boston housing data

In addition to obtaining visual measures of accuracy, you can also use statistical techniques to evaluate the quality of a regression model. We will look at some of the commonly used metrics next.

RMSE Metric

The root mean squared error (RMSE) metric is popular when it comes to evaluating the performance of a regression model. As the name suggests, root mean square is the square root of the mean squared error (MSE). For a given item in the test set, the error in prediction is the difference between the predicted value and the actual value; this error can be either positive or negative, and squaring it ensures that the direction of the error does not matter (as squared numbers are always positive). The mean squared error is the mean of the squared prediction errors of all the items in the training set. This is illustrated in Figure 5.2.

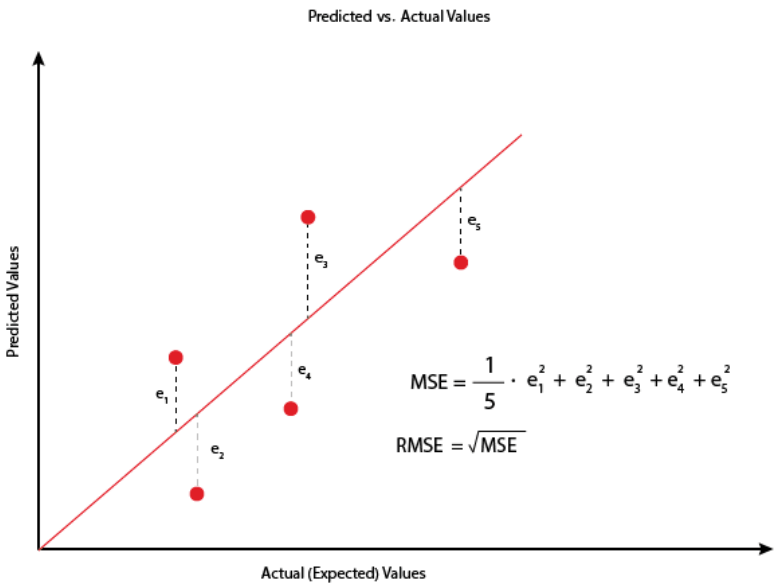


FIGURE 5.2
Mean squared error and root mean squared error

Scikit-learn encapsulates the computation of the mean squared error in the `sklearn.metrics.mean_squared_error()` function. The following snippet uses this function to compute the MSE and RMSE of the predictions made by the linear regression model and the decision tree regression model on the Boston housing prices dataset:

```
# compute MSE , RMSE, using Scikit-learn

from sklearn.metrics import mean_squared_error
mse_linear_reg_model = mean_squared_error(df_boston_target_test,
linear_reg_predictions)
mse_dtree_reg_model = mean_squared_error(df_boston_target_test, dtree_reg_
predictions)

from math import sqrt
rmse_linear_reg_model = sqrt(mse_linear_reg_model)
rmse_dtree_reg_model = sqrt(mse_dtree_reg_model)
```

You can examine the RMSE values using the Python `print()` function. As you can see, the RMSE for the decision tree–based model is lower than the RMSE for the linear regression model, which implies the decision tree–based model is the better of the two:

```
print (rmse_linear_reg_model, rmse_dtree_reg_model)
4.256139223131222 3.502392685222358
```

The benefit of root mean squared error is that its value is in the same units as the variable you are trying to predict. For example, the house prices in the Boston housing prices dataset are expressed in units of $1,000.00. An RMSE of 3.502 implies that on an average the house prices predicted by this model will be off from the true house price by $3502.00. Since the value of the RMSE is in the units of the target variable, it is easy to understand.

Unfortunately, the drawback of RMSE is that its value is sensitive to the magnitude of the target variable. If the target variables are larger numbers, the value of the RMSE will be a larger number, and therefore the RMSE cannot be used to compare models trained on different datasets.

R^2 Metric

The R^2 metric is another statistical metric that can be used to get an idea of the quality of the model. However, unlike RMSE, the value of the R^2 metric always lies between 0.0 and 1.0. The R^2 metric is also known as the coefficient of determination, and is a measure of the distance of the predicted values from the regression line.

Scikit-learn encapsulates the computation of the coefficient of determination in the `sklearn.metrics.mean_r2_score()` function. The following snippet uses this function to compute the R^2 score of the predictions made by the linear regression model and the decision tree regression model on the Boston housing prices dataset:

```
# compute coefficient of determination (r2 score)

from sklearn.metrics import r2_score
r2_linear_reg_model = r2_score(df_boston_target_test, linear_reg_predictions)
r2_dtree_reg_model = r2_score(df_boston_target_test, dtree_reg_predictions)
```

You can examine the R2 values using the Python `print()` function. As you can see, the R2 for the decision tree–based model is higher than the R2 score for the linear regression model, which implies the decision tree–based model is, once again, the better of the two:

```
print (r2_linear_reg_model, r2_dtree_reg_model)

0.7663974484228384 0.8418112461085272
```

Evaluating Classification Models

There are two types of classification models: binary and multi-class. Binary classification models are used when the target attribute can have only two discrete values (or classes). Multi-class classification models are used when the target attribute can have more than one discrete value. Let's consider binary classification models first.

Binary Classification Models

One of the simplest metrics that can be used to gauge the quality of the model is to simply count the number of times the model predicts the correct class. Whether or not this value has any meaning would depend on the proportion of samples that belong to each class, and the significance of the classes themselves. For instance, if a test set contains 100 samples, 50 of which are from class A and the other 50 from class B, with neither class being more significant to the problem domain than the other, then a model that predicts the correct class 80% of the time is straightforward to understand. If, however, 95 items in the test set were from class A and only 5 were from class B, the model that predicts the correct class 80% of the time is not so good after all. The problem can be significantly worse if the model was meant to predict whether an individual had a deadly illness, and the five samples from class B were the only ones that indicated presence of the illness. This is not an impossible situation; there is a very high possibility that a random sample of the general population will have very few individuals with a specific illness.

Before we look at better measures of a binary classification model's performance, let's first see how we can compute this simple metric. The following snippet trains a logistic regression, support vector machine (SVM), and decision tree classifier on the Pima Indians diabetes dataset and computes the percentage of correct predictions:

```
import numpy as np
import pandas as pd
import os

# load Pima Indians Diabetes dataset
diabetes_dataset_file = './datasets/diabetes_dataset/diabetes.csv'
df_diabetes = pd.read_csv(diabetes_dataset_file)
df_diabetes_target = df_diabetes.loc[:,['Outcome']]
df_diabetes_features = df_diabetes.drop(['Outcome'], axis=1)

# normalize attribute values
from sklearn.preprocessing import MinMaxScaler

diabetes_scaler = MinMaxScaler()
diabetes_scaler.fit(df_diabetes_features)
nd_diabetes_features = diabetes_scaler.transform(df_diabetes_features)
df_diabetes_features_normalized = pd.DataFrame(data=nd_diabetes_features,
columns=df_diabetes_features.columns)

# create a training dataset and a test dataset using a 75/25 split.
from sklearn.model_selection import train_test_split

diabetes_split = train_test_split(df_diabetes_features_normalized,
df_diabetes_target,
                                  test_size=0.25, random_state=17)
df_diabetes_features_train = diabetes_split[0]
df_diabetes_features_test = diabetes_split[1]
df_diabetes_target_train = diabetes_split[2]
df_diabetes_target_test = diabetes_split[3]

# train an SVM classifier for the features of the diabetes dataset using an RBF
kernel from sklearn.svm import SVC
svc_model = SVC(kernel='rbf', C=1, gamma='auto')
svc_model.fit(df_diabetes_features_train,
df_diabetes_target_train.values.ravel())

# train a logistic regression model on the diabetes dataset
from sklearn.linear_model import LogisticRegression
logit_model = LogisticRegression(penalty='l2', fit_intercept=True,
solver='liblinear')
logit_model.fit(df_diabetes_features_train,
df_diabetes_target_train.values.ravel())

# train a decision tree based binary classifier.
from sklearn.tree import DecisionTreeClassifier
```

```
dtree_model = DecisionTreeClassifier(max_depth=4)
dtree_model.fit(df_diabetes_features_train,
df_diabetes_target_train.values.ravel())

# use the models to create predictions on the diabetes test set
svc_predictions = svc_model.predict(df_diabetes_features_test)
logit_predictions = logit_model.predict(df_diabetes_features_test)
dtree_predictions = dtree_model.predict(df_diabetes_features_test)

# simplistic metric - the percentage of correct predictions
svc_correct = svc_predictions == df_diabetes_target_test.values.ravel()
svc_correct_percent = np.count_nonzero(svc_correct) / svc_predictions.size * 100

logit_correct = logit_predictions == df_diabetes_target_test.values.ravel()
logit_correct_percent = np.count_nonzero(logit_correct) /
logit_predictions.size * 100

dtree_correct = dtree_predictions == df_diabetes_target_test.values.ravel()
dtree_correct_percent = np.count_nonzero(dtree_correct) / dtree_predictions.
size * 100
```

Using the Python `print()` function, the results indicate that the logistic regression model and the decision tree model perform about the same, but the logistic regression model has the edge:

```
print (svc_correct_percent, logit_correct_percent, dtree_correct_percent)

73.95833333333334 76.5625 75.52083333333334
```

Now let's look at other performance metrics that could be used in binary classification problems. Some of the problems with simply counting the number of times a model predicts the correct answer were introduced earlier in this chapter. To get a better idea of the model's performance, what you need is a set of metrics that capture the class-wise performance of the model. The most commonly used primary metrics for binary classification are listed next. These metrics assume one of the two target classes represents a positive outcome, whereas the other represents a negative outcome:

- *True positive (TP) count:* The number of times the model predicted a positive outcome, and the prediction was correct.
- *False positive (FP) count:* The number of times the model predicted a positive outcome, and the prediction was incorrect.
- *True negative (TN) count:* The number of times the model predicted a negative outcome, and the prediction was correct.
- *False negative (FN) count:* The number of times the model predicted a negative outcome, and the prediction was incorrect.

This class-wise prediction accuracy is also sometimes referred to as the class-wise confusion, and when these four primary metrics are placed in a 2 × 2 matrix, the resulting matrix is called the confusion matrix. Figure 5.3 depicts a confusion matrix and explains what the individual numbers mean.

FIGURE 5.3
A class-wise confusion matrix

Features			Actual target	Predicted target
X_1	X_2	X_3	Y	Y
...	...	...	Y	Y
...	...	...	N	Y
...	...	...	Y	Y
...	...	...	Y	Y
...	...	...	Y	Y
...	...	...	N	N
...	...	...	N	N
...	...	...	N	N
...	...	...	N	Y
...	...	...	Y	N
...	...	...	Y	N
...	...	...	Y	N

	Predicted class: Positive	Predicted class: Negative	
Actual class: Positive	4/12 True Positives	3/12 False Negatives	→ Total number of actual positives = 4 + 3 = 7
Actual class: Negative	2/12 False Positives	3/12 True Negatives	→ Total number of actual negatives = 3 + 2 = 5
	↓ Total number of positive predictions = 4 + 2 = 6	↓ Total number of negative predictions = 3 + 3 = 6	

Scikit-learn provides a function called `confusion_matrix()` in the `metrics` submodule that can be used to compute the confusion matrix for a classification model. You can find detailed information on the parameters of the `confusion_matrix()` function at `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`.

The following snippet demonstrates the use of this function to compute the confusion matrix for the three binary classification models created earlier in this section:

```
# compute confusion matrix
from sklearn.metrics import confusion_matrix

cm_svc = confusion_matrix(df_diabetes_target_test.values.ravel(), svc_predictions)
cm_logit = confusion_matrix(df_diabetes_target_test.values.ravel(),
logit_predictions)
cm_dtree = confusion_matrix(df_diabetes_target_test.values.ravel(),
dtree_predictions)

# extract true negative , false positive, false negative, true positive.
#
# the sklearn confusion_matrix() function returns in the following matrix
#
#       TN     FP
#       FN     TP

tn_svc, fp_svc, fn_svc, tp_svc = cm_svc.ravel()
tn_logit, fp_logit, fn_logit, tp_logit = cm_logit.ravel()
tn_dtree, fp_dtree, fn_dtree, tp_dtree = cm_dtree.ravel()
```

You can now examine the values of the primary metrics for the three models using the Python `print()` function:

```
print (tn_svc, fp_svc, fn_svc, tp_svc)
>> 113 8 42 29

print (tn_logit, fp_logit, fn_logit, tp_logit)
>> 113 8 37 34

print (tn_dtree, fp_dtree, fn_dtree, tp_dtree)
>> 103 18 29 42
```

In addition to these primary statistical metrics, data scientists use the following secondary metrics. The values of these metrics are computed from the primary metrics:

- *Accuracy:* This is defined as (TP + TN) / (Total number of predictions). This is basically the same as counting the number of times the model predicted the correct value.
- *Precision:* This is defined as TP / (TP + FP). If you look at Figure 5.3, you will notice that the denominator is the total number of positive predictions made by the model. Therefore, precision can also be written as TP / (Total number of positive prediction), and it measures how precise your model is. Precision is a good measure to use when there is a high cost associated with a false positive. The closer the value is to 1.0, the more precise are the positive predictions.

- *Recall:* This is defined as TP / (TP + FN). Looking at Figure 5.3, you will notice that the denominator of this expression is the actual number of positive samples in the dataset. Therefore, the expression to compute recall can be rewritten as TP / (Total number of positive samples in the evaluation set). Recall is a good measure to use when there is a high cost associated with false negatives.

The following snippet computes the accuracy, precision, and recall values for the three models, using the information obtained from the confusion matrix:

```
# compute accuracy, precision, and recall

accuracy_svc = (tp_svc + tn_svc) / (tn_svc + fp_svc + fn_svc + tp_svc)
accuracy_logit = (tp_logit + tn_logit) / (tn_logit + fp_logit + fn_logit + tp_logit)
accuracy_dtree = (tp_dtree + tn_dtree) / (tn_dtree + fp_dtree + fn_dtree + tp_dtree)

precision_svc = tp_svc / (tp_svc + fp_svc)
precision_logit = tp_logit / (tp_logit + fp_logit)
precision_dtree = tp_dtree / (tp_dtree + fp_dtree)

recall_svc = tp_svc / (tp_svc + fn_svc)
recall_logit = tp_logit / (tp_svc + fn_logit)
recall_dtree = tp_dtree / (tp_dtree + fn_dtree)
```

Using the Python `print()` function, you can examine the accuracy, precision, and recall:

```
print (accuracy_svc, accuracy_logit, accuracy_dtree)
>> 0.7395833333333334   0.765625   0.7552083333333334

print (precision_svc, precision_logit, precision_dtree)
>> 0.7837837837837838   0.8095238095238095    0.7

print (recall_svc, recall_logit, recall_dtree)
0.4084507042253521   0.5151515151515151   0.5915492957746479
```

The logistic regression model has the highest precision of 80.95%, whereas the decision tree has the highest recall of 59.14%. Your choice of model will be influenced by the problem domain. What matters more: precision or recall?

It is important to note that all three classification models trained in this section compute class-wise prediction probabilities, and then threshold the probabilities to arrive at the binary output class. Scikit-learn uses a threshold of 0.5, and while it does not allow you to change this threshold, it does provide access to the underlying probabilities so that you can implement your own threshold if you want to. This means that the performance metrics discussed in this section will change for different values of the threshold. One possibility is to compute the confusion matrix for 100 thresholds between 0.0 and 1.0 and pick the threshold value that provides the best balance between accuracy, precision, and recall.

Data scientists often use a visualization tool called the ROC (receiver operating characteristics) curve, and a metric called AUC (area under ROC curve) to evaluate the quality of a classification model. The ROC curve is computed by plotting the rate of true positives on the y-axis, against the rate of false positives on the x-axis for a number of different confusion matrices, computed for threshold values between 0.0 and 0.1.

Scikit-learn provides a function called `roc_curve()` in the `metrics` submodule that can be used to compute the true and false positive rates (TPR and FPR) for a number of different

threshold values. You can find detailed information on the parameters of the `roc_curve()` function at `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html`.

The following snippet demonstrates the use of the `roc_curve()` function to compute the true and false positive rates for the three binary classification models created earlier in this section, and plots the ROC curves using Matplotlib's `pyplot` module. The resulting ROC curves are depicted in Figure 5.4.

```
# plot ROC curves for the three classifiers.

# compute prediction probabilities
svc_probabilities = svc_model.predict_proba(df_diabetes_features_test)
logit_probabilities = logit_model.predict_proba(df_diabetes_features_test)
dtree_probabilities = dtree_model.predict_proba(df_diabetes_features_test)

# calculate the FPR and TPR for all thresholds of the SVC model
import sklearn.metrics as metrics
svc_fpr, svc_tpr, svc_thresholds =
metrics.roc_curve(df_diabetes_target_test.values.ravel(),
                                                    svc_probabilities[:,1],
                                                    pos_label=1,
                                                    drop_intermediate=False)

# calculate the FPR and TPR for all thresholds of the logistic regression model
logit_fpr, logit_tpr, logit_thresholds =
metrics.roc_curve(df_diabetes_target_test.values.ravel(),

logit_probabilities[:,1],
                                                    pos_label=1,
drop_intermediate=False)

# calculate the FPR and TPR for all thresholds of the decision tree model
dtree_fpr, dtree_tpr, dtree_thresholds = metrics.roc_curve(df_diabetes_target_
test.values.ravel(),

dtree_probabilities[:,1],
                                                    pos_label=1,
drop_intermediate=False)



# plot ROC curves
%matplotlib inline
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 3, figsize=(18,6))

axes[0].set_title('ROC curve: SVC model')
axes[0].set_xlabel("True Positive Rate")
axes[0].set_ylabel("False Positive Rate")
axes[0].plot(svc_fpr, svc_tpr)
axes[0].axhline(y=0, color='k')
axes[0].axvline(x=0, color='k')
```

```
axes[1].set_title('ROC curve: Logit model')
axes[1].set_xlabel("True Positive Rate")
axes[1].set_ylabel("False Positive Rate")
axes[1].plot(logit_fpr, logit_tpr)
axes[1].axhline(y=0, color='k')
axes[1].axvline(x=0, color='k')

axes[2].set_title('ROC curve: Tree model')
axes[2].set_xlabel("True Positive Rate")
axes[2].set_ylabel("False Positive Rate")
axes[2].plot(dtree_fpr, dtree_tpr)
axes[2].axhline(y=0, color='k')
axes[2].axvline(x=0, color='k')
```

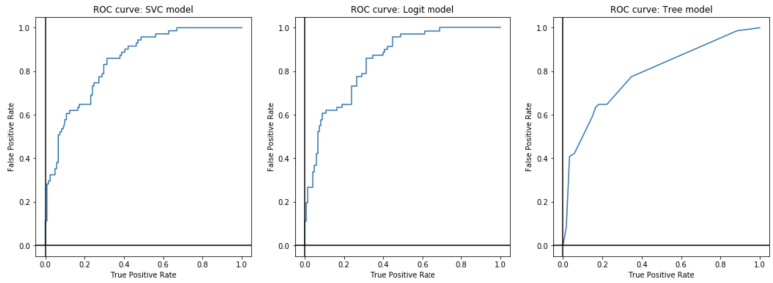


FIGURE 5.4
ROC curves for three binary classification models

Scikit-learn also provides the `auc()` function in the `metrics` module that can compute the area under the ROC curve from the true and false positive rates returned by the `roc_curve()` function. The following snippet computes the AUC metric for the three classification models:

```
# compute AUC metrics for the three models
svc_auc = metrics.auc(svc_fpr, svc_tpr)
logit_auc = metrics.auc(logit_fpr, logit_tpr)
dtree_auc = metrics.auc(dtree_fpr, dtree_tpr)
```

Using the Python `print()` function to examine the values of the AUC metrics for the three models, the logistic regression model provides the highest AUC value:

```
print (svc_auc, logit_auc, dtree_auc)
0.845186823419858     0.8467000349202654     0.7846001629612384
```

Multi-Class Classification Models

Multi-class classification models are used when the target variable can belong to more than two classes. Fortunately, most of the techniques used to evaluate the performance of binary classification models can be applied to their multi-class counterparts. The concept of the confusion matrix can easily be extended to multi-class classification. The confusion matrix for an n-class classification would be represented as an n × n matrix where the rows represent the actual classes, and the columns represent the predicted classes. Figure 5.5 depicts a multi-class confusion matrix for a five-class problem.

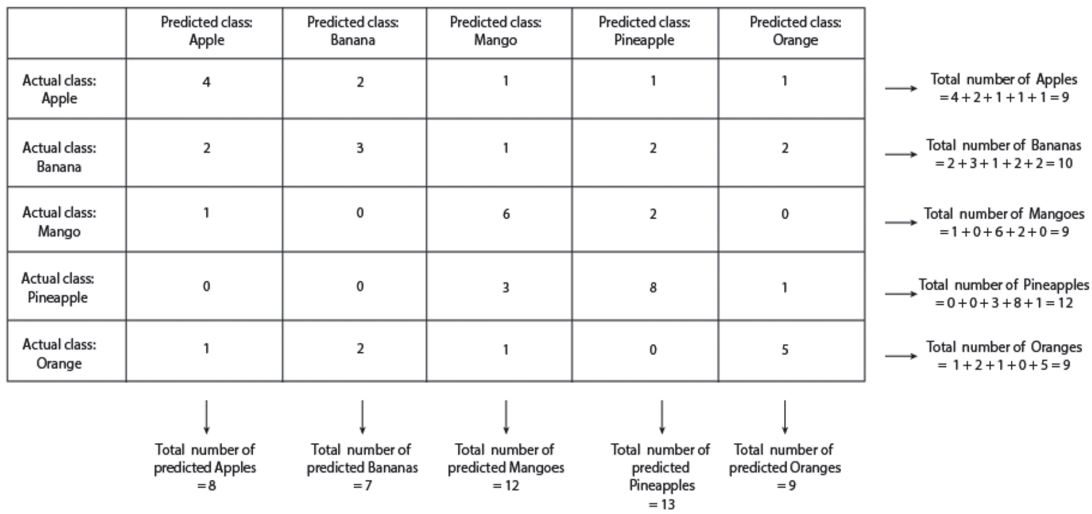


FIGURE 5.5 Multi-class confusion matrix for a five-class dataset

Scikit-learn's `confusion_matrix()` function can also be used to compute the confusion matrix for multi-class classification models. The following code snippet trains a softmax logistic and decision tree model on the Iris flowers training dataset, uses the models to make predictions, and computes the confusion matrix:

```
# load Iris flowers dataset
from sklearn.datasets import load_iris
iris_dataset = load_iris()
df_iris_features = pd.DataFrame(data = iris_dataset.data, columns=iris_dataset.
feature_names)
df_iris_target = pd.DataFrame(data = iris_dataset.target, columns=['class'])

# normalize attribute values
from sklearn.preprocessing import MinMaxScaler
iris_scaler = MinMaxScaler()
iris_scaler.fit(df_iris_features)
nd_iris_features = iris_scaler.transform(df_iris_features)
df_iris_features_normalized = pd.DataFrame(data=nd_iris_features,
columns=df_iris_features.columns)

# create a training dataset and a test dataset using a 75/25 split.
from sklearn.model_selection import train_test_split
iris_split = train_test_split(df_iris_features_normalized, df_iris_target,
                              test_size=0.25, random_state=17)
df_iris_features_train = iris_split[0]
df_iris_features_test = iris_split[1]
df_iris_target_train = iris_split[2]
df_iris_target_test = iris_split[3]

# softmax (a.k.a multinomial regression) classifier
from sklearn.linear_model import LogisticRegression
softmax_logit_model = LogisticRegression(penalty='l2', fit_intercept=True,
solver='lbfgs', multi_class='multinomial')
softmax_logit_model.fit(df_iris_features_train, df_iris_target_train.
values.ravel())

# create a decision tree based multi-class classifier.
from sklearn.tree import DecisionTreeClassifier
mc_dtree_model = DecisionTreeClassifier(max_depth=4)
mc_dtree_model.fit(df_iris_features_train, df_iris_target_train.values.ravel())

# use the  model to create predictions on the test set
softmax_logit_predictions = softmax_logit_model.predict(df_iris_features_test)
mc_predictions = mc_dtree_model.predict(df_iris_features_test)

# compute confusion matrix
from sklearn.metrics import confusion_matrix

cm_softmax = confusion_matrix(df_iris_target_test.values.ravel(), softmax_logit_
predictions)
```

```
cm_mc_dtree = confusion_matrix(df_iris_target_test.values.ravel(), mc_
predictions)
```

You can inspect the confusion matrices generated by the two classifiers by using the Python `print()` function:

```
print(cm_softmax)

[[10  0  0]
 [ 0 14  1]
 [ 0  1 12]]

print(cm_mc_dtree)

[[10  0  0]
 [ 0 15  0]
 [ 0  1 12]]
```

Multi-class confusion matrices are often visualized as heat maps. The following snippet visualizes the two confusion matrices as heat maps side by side. Portions of the snippet have been adapted from `https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html`. The resulting heatmaps are depicted in Figure 5.6.

```
# plot confusion matrices as heatmaps
%matplotlib inline
import matplotlib.pyplot as plt

# plot the confusion matrix as a heat map using Matplotlib functions.
#
# portions of this code have been adapted from
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_
matrix.html
def plot_confusion_matrix(cmatrix, class_labels, axes, title, cmap):

    heatmap_image = axes.imshow(cmatrix, interpolation='nearest', cmap=cmap)
    axes.figure.colorbar(heatmap_image, ax=axes)

    num_rows = cmatrix.shape[0]
    num_cols = cmatrix.shape[1]

    axes.set_title(title)
    axes.set_xlabel('Predicted')
    axes.set_ylabel('True')

    axes.set_xticks(np.arange(num_cols))
    axes.set_yticks(np.arange(num_rows))
    axes.set_xticklabels(class_labels)
    axes.set_yticklabels(class_labels)

    # Loop over data dimensions and create text annotations.
    #fmt = '.2f' if normalize else 'd'
```

```
    thresh = cmatrix.max() / 2.
    for y in range(num_rows):
        for x in range(num_cols):
            axes.text(x, y, format(cmatrix[y, x], '.0f'),
                    ha="center", va="center",
                    color="white" if cmatrix[y, x] > thresh else "black")

fig, axes = plt.subplots(1, 2, figsize=(18,6))

plot_confusion_matrix(cm_softmax,
                      iris_dataset.target_names, axes[0],
                      "Softmax Regression",
                      plt.cm.Greens)

plot_confusion_matrix(cm_mc_dtree,
                      iris_dataset.target_names, axes[1],
                      "Decision Tree Regression",
                      plt.cm.Greens)
```

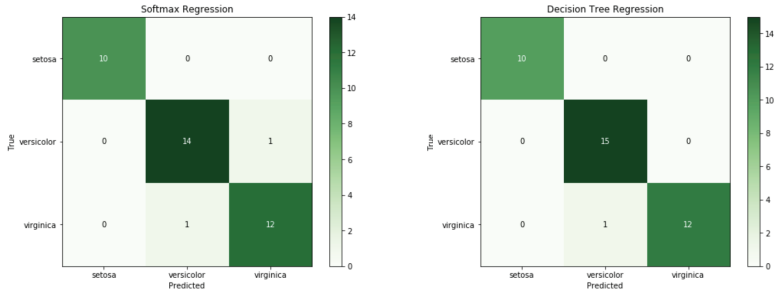


FIGURE 5.6
Multi-class confusion matrix for two models trained on the Iris flowers dataset

For a multi-class classification model, you can compute the overall accuracy metric, and the class-wise precision, and the recall. These metrics would be defined as follows:

- *Accuracy:* In a multi-class classification model, the overall accuracy of the model is the number of times the model predicted the correct answer. Using a confusion matrix, accuracy can be represented as the Sum of the diagonal elements / Total number of predictions. In the case of the confusion matrix for the decision tree model depicted in Figure 5.6, the overall accuracy is 37/38.
- *Precision:* In a multi-class classification problem, the class-wise precision for a given class would be defined as the Number of times the model predicted the class correctly / Total number of predictions made for that class. Using a confusion matrix, precision can be represented graphically as the ratio of the number in the diagonal element to the column total. In the case of the confusion matrix for the decision tree model depicted in Figure 5.6, the class-wise precision for the Setosa class is 10/10, Versicolor is 15/16, and Virginica is 12/12.
- *Recall:* In a multi-class classification problem, the class-wise recall for a given class would be defined as the Number of times the model predicted the class correctly / Total number of elements of that class. Using a confusion matrix, recall can be represented graphically as

the ratio of the number in the diagonal element to the row total. In the case of the confusion matrix for the decision tree model depicted in Figure 5.6, the class-wise recall for the Setosa class is 10/10, the Versicolor class is 15/15, and Virginica is 12/13.

Choosing Hyperparameter Values

During a machine learning project, creating a single model and computing metrics is not enough. As you have learned in this chapter and the previous ones, there are a number of factors that could influence the performance of the model, from feature engineering to the choice of model and hyperparameters used during the model building. In order to automate the process of finding the optimal model, data scientists often use the grid search technique to try various combinations of hyperparameters and pick the model that appears to perform best. Scikit-learn provides the `GridSearchCV` class in the `model_selection` module that can be used to perform an exhaustive search over a set of parameter values. You can learn more about the `GridSearchCV` class at `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`.

The following code snippet uses the `GridSearchCV` class to try different hyperparameter combinations for a multi-class decision tree classifier on the Iris flowers dataset and returns the hyperparameters that result in the best precision score:

```
# use grid search to find the hyperparameters that result
# in the best accuracy score for a decision tree
# based classifier on the Iris Flowers dataset
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

grid_params = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'max_features': ['auto', 'sqrt', 'log2'],
    'presort': [True, False]
}

grid_search = GridSearchCV(estimator=DecisionTreeClassifier(),
                           param_grid=grid_params, scoring='accuracy',
                           cv=10, n_jobs=-1)

grid_search.fit(df_iris_features.values, df_iris_target)
```

In this snippet, `grid_params` is a dictionary of hyperparameters and the values for each parameter that you want to try. The elements of the dictionary will depend on the model that you want to train. This example uses a decision tree classification model and the dictionary has six entries that correspond to some of the hyperparameters of the `DecisionTreeClassifier` class. The grid search process will build a model with each combination of hyperparameters, which in this case will be $2 \times 2 \times 11 \times 11 \times 3 \times 2 = 15{,}488$ models.

The `cv=10` parameter in the constructor of the `GridSearchCV` class is used to control the number of folds, which in this case is set to 10 and therefore each of the 15,488 models will be trained using 10-fold cross validation. Depending on the speed of your computer, this code can

take a significant amount of time to execute. Once the grid search is complete, you can inspect the value of the hyperparameters that resulted in the best model, and the accuracy of that model, using the following statements:

```
best_parameters = grid_search.best_params_
print(best_parameters)

{'criterion': 'entropy', 'max_depth': 7, 'max_features': 'sqrt', 'min_samples_
split': 12, 'presort': False, 'splitter': 'random'}

best_accuracy = grid_search.best_score_
print(best_accuracy)

0.98
```

NOTE To follow along with this chapter ensure you have installed Anaconda Navigator and Jupyter Notebook as described in Appendix A.

You can download the code files for this chapter from `www.wiley.com/go/machinelearningawscloud` or from GitHub using the following URL:

`https://github.com/asmtechnology/awsmlbook-chapter5.git`

Summary

- There are two types of techniques that can be used to evaluate the predictive accuracy of a regression model: creating a scatter plot of the true and predicted values, and computing a statistical metric that captures the total prediction error across members of the test set.
- The root mean squared error (RMSE) metric is popular when it comes to evaluating the performance of a regression model. As the name suggests, root mean square is the square root of the mean squared error (MSE).
- The R^2 metric is another statistical metric that can be used to get an idea of the quality of the model.
- The R^2 metric is also known as the coefficient of determination. It is a measure of the distance of the predicted values from the regression line.
- The true positive count for a binary classification model is defined as the number of times the model predicted a positive outcome, and the prediction was correct.
- The false positive count for a binary classification model is defined as the number of times the model predicted a positive outcome, and the prediction was incorrect.
- The true negative count for a binary classification model is defined as the number of times the model predicted a negative outcome, and the prediction was correct.
- The false negative count for a binary classification model is defined as the number of times the model predicted a negative outcome, and the prediction was incorrect.
- Accuracy, precision, and recall are additional metrics that can be used to evaluate a binary classification model.

Part 2

Machine Learning with Amazon Web Services

Chapter 6

Introduction to Amazon Web Services

WHAT'S IN THIS CHAPTER

- Introduction to the basics of cloud computing
- Introduction to the AWS ecosystem and key services used to build machine learning solutions
- Signing up for an account under the AWS free tier

In this chapter, you learn about what cloud computing is, read about common models of abstraction used when discussing cloud-based services, and discover a high-level overview of Amazon's offerings in the cloud-computing space, with emphasis on services that help build machine learning solutions. The chapter wraps up by walking you through signing up for an AWS account.

What Is Cloud Computing?

Cloud computing is defined by the U.S. National Institute of Standards and Technology (NIST) as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (such as networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[1]

NIST defines five essential characteristics in this model. Each of these is briefly examined next:

- *Broad network access:* A consumer should be able to access services from anywhere.
- *Resource pooling:* A provider's computing resources are pooled to support multiple customers.
- *On-demand self-service:* A consumer should be able to provision computing resources (such as virtual servers) as needed, with minimal human interaction.
- *Measured service:* A consumer should be able to use computing resources on a pay-as-you-use basis.

[1] Peter Mell and Timothy Grance, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145. September 2011. (http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf)

- *Elasticity:* A consumer should be able to provision additional resources automatically and on demand. To ensure this, the provider pools computing resources to provide horizontal scalability to the consumer.

Cloud-computing solutions provide two major advantages to businesses:

- *Costs:* The cloud-computing paradigm is based on sharing and optimal utilization of hardware resources. A business need only pay for the time during which it utilizes a resource. When a resource is not needed, the business can relinquish it and make it available for someone else to use. This reduces both the upfront hardware investment cost for a business as well as ongoing maintenance costs. The cloud service provider, not the consumer, handles the maintenance of the underlying hardware.
- *Availability:* The time to provision a ready-to-use resource in the cloud is significantly lower than having to set up a similar resource in-house. For instance, a business could provision a virtual server with a cloud provider within seconds, whereas the actual process of procuring new server hardware and software usually takes a few months in most medium to large organizations.

Cloud Service Models

Cloud computing is built on virtualization technology. Fundamentally, there are two types of virtualization:

- *Application virtualization:* A single machine hosts one or more applications that are delivered to one or more users over the Internet.
- *Hardware virtualization:* Also known as server virtualization, in this model a single physical machine hosts multiple virtual machines. Each virtual machine can have its own operating system (different from the operating system of the underlying physical machine) and its own unique set of applications.

As an end user, you expect to consume one or more services from your cloud-computing provider over the Internet. These services can range from bare-bones virtual machines with a basic operating system to entire suites of applications. Five common models for cloud services are shown in Figure 6.1. These can be conceptualized using a layered model, with higher layers building upon the services offered by lower layers.

- *Infrastructure as a service (IaaS):* You specify the low-level details of the virtual server you require, including the number of CPUs, RAM, hard disk space, networking capabilities, and operating system. The cloud provider offers a virtual machine to match these requirements. In addition to virtual servers, the definition of IaaS includes networking peripherals such as firewalls, load balancers, and storage. Therefore provisioning multiple load-balanced Java application servers on the cloud as well as storing your files on a cloud-based disk would both come under the IaaS model.

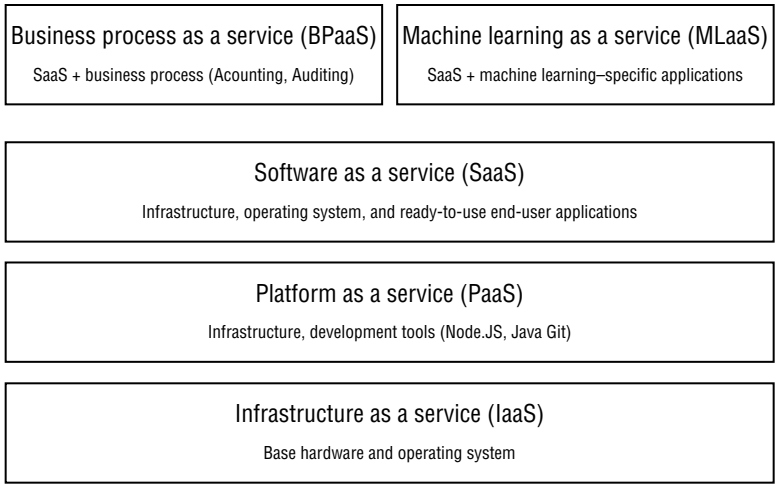


FIGURE 6.1
Common cloud service models

- *Platform as a service (PaaS):* You choose between combinations of infrastructure and preconfigured software that best suits your needs. The cloud provider offers a virtual machine with preconfigured internal applications to match your requirements. The PaaS model is generally easier to use as you do not have to set up the underlying hardware and software; however, this can also be restrictive as your level of control on the underlying systems is significantly reduced compared to the IaaS model. The choice of infrastructure and pre-installed software differs between cloud providers, and if a cloud vendor does not provide something off the shelf that meets your needs, you are out of luck.
- *Software as a service (SaaS):* You specify the kind of software application you want to use, such as a word processor. The cloud provider provisions the required infrastructure, operating system, and applications to match your requirement. Most SaaS cloud providers include a limited choice of the hardware characteristics that run the application and as a user you usually have no direct access to the underlying hardware that runs your application. You are also tied to the limitations of the hardware and software chosen by your cloud provider. If, for instance, a cinema chain is using a SaaS cloud-based booking system to manage ticket sales and the system is unavailable due to an internal error with the hardware used by the cloud provider, there is little the cinema chain can do but wait until the cloud provider has fixed the issue.
- *Business process as a service (BPaaS):* You specify a business process that you want to outsource to a cloud provider. The cloud service provider provisions the hardware, operating system, support software, and web applications to provide the required service. A good example of BPaaS would be a cloud-based service to compute quarterly value-added tax (VAT) returns and submit these returns to the relevant tax authority on your behalf. Such a service could present you with a browser-based front end in which you

upload your invoices and business bank statements. The service could then extract relevant information from the uploaded documents (using OCR, perhaps), fill out the relevant tax authority's forms, ask you to review the results, and submit the return on your behalf once you are happy with the numbers.

- *Machine learning as a service (MLaaS):* The cloud provider provides a number of services to assist with data modeling, machine learning model building, data transformation, data visualization, natural language processing, facial recognition, prediction, and deep learning. The services themselves may be classed as either software/application-level or platform-level services. The classification depends on how much control and flexibility is provided to the end user. The cloud provider automatically provisions the underlying infrastructure with sufficient capacity to scale up with demand.

Cloud Deployment Models

A deployment model answers the following questions:

- Who can access a computing resource?
- How can a user access a computing resource?
- Where is the physical hardware?

Cloud-computing solutions have four distinct deployment models:

- *Public cloud:* A public cloud provides services over the Internet to a consumer located anywhere in the world. The physical resources utilized by the provider to supply these services can also be anywhere in the world. This type of service could represent potential challenges to organizations such as banks that are prevented by regulatory requirements from storing confidential data on external systems. The cloud provider is generally responsible for procurement, setup, physical security, and maintenance of the physical infrastructure. To an extent the cloud provider is also responsible for the security of the application and data; this would depend on the service model (IaaS/PaaS/SaaS).
- *Private cloud:* A private cloud offers services to a single organization. Services are provided over a secure internal network and are not accessible to the general public over the Internet. The organization owns the physical hardware that supplies underlying services and is responsible for setup, maintenance, and security of both the infrastructure and all the software that runs on the infrastructure. Because of the large infrastructure costs associated with this model, only very large corporations can afford to have their own private clouds. A private cloud is commonly referred to as an on-premises cloud, and can offer any combination of IaaS/SaaS/PaaS to the organization.
- *Community cloud:* A community cloud provides services to a small group of entities (individuals, universities, or corporations) over a secure network. The underlying resources used to supply the services are owned by the entities that the community cloud serves. In essence, this type of cloud service can be thought of as something between a public cloud and a private cloud. The service is not accessible to members of the general public and does not put a significant drain on any one entity's finances. The entities involved usually share a common goal or provide services in a common industry sector.

- *Hybrid cloud:* A hybrid cloud is essentially a cloud service that is composed of other types of cloud services. For example, a hybrid cloud could consist of both public and private clouds. The public subcloud could provide services that are intended for consumption by any user over the Internet. A private cloud could offer services that are sensitive to the business. Most large organizations use a hybrid cloud model. It lets them provision public cloud resources when needed for a specific business case, while continuing to enjoy the security and control that a private cloud allows for all their existing processes.

The AWS Ecosystem

Amazon Web Services is the most rapidly evolving cloud-computing service in the market. The current AWS offering consists of over a hundred services offered in locations around the world, with new services being added every year.

New additions to AWS are announced at AWS re:Invent, which is the official annual AWS conference. You can find more information on AWS re:Invent at `https://reinvent.awsevents.com`.

Figure 6.2 describes a brief timeline of AWS. AWS was born in a paper presented by Chris Pinkham and Benjamin Black to Jeff Bezos in 2003.

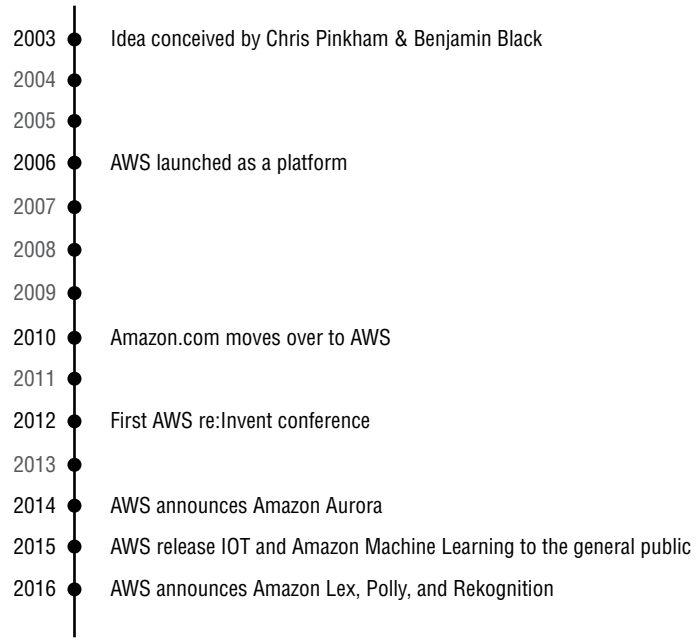


Figure 6.2
Brief timeline of Amazon Web Services

This paper suggested selling Amazon's internal infrastructure as a service to the world. In 2006 AWS was officially launched as a platform with a few key services, including EC2 and S3.

In November 2010, Amazon announced that all of Amazon.com had migrated to AWS. The first AWS re:Invent conference took place from November 27 to November 29, 2012, at the Venetian Hotel in Las Vegas. In 2015, Amazon launched its own machine learning platform called

Amazon Machine Learning, and in 2016 announced three new API-based services specifically geared toward machine learning applications: Amazon Rekognition, Amazon Polly, and Amazon Lex.

At the time this book was written, AWS offered 130 services, each of which resided in one of 20 different service categories. The service categories are as follows:

- Compute
- Storage
- Database
- Migration
- Networking & Content Delivery
- Developer Tools
- Management Tools
- Media Services
- Machine Learning
- Analytics
- Security, Identity & Compliance
- Mobile Services
- AR & VR
- Application Integration
- AWS Cost Management
- Customer Engagement
- Business Productivity
- Desktop & App Streaming
- Internet of Things
- Game Development

You can find details on all AWS services within these categories at `https://aws.amazon.com/products/`.

Amazon's offerings in the machine learning space are grouped into two categories: application services and platform services.

Machine Learning Application Services

These services are designed to solve specific machine learning problems out of the box and can be integrated into your own applications via APIs. Some of the services in this category are listed here:

- *Amazon Comprehend:* This is a service that allows you to build applications that need to understand the structure and content of text. Amazon Comprehend uses Natural

Language Processing (NLP) to extract insights into the content of documents. The insights can be entities (people, places), key phrases, sentiment (positive, neutral, mixed, or negative), and syntax. Amazon Comprehend is applicable across a variety of use cases; for example, an application that can examine the contents of forums to understand topics that your customers are interested in. Amazon Comprehend is covered in Chapter 13.

- *Amazon Lex:* This is a service that allows you to build conversational interfaces (chatbots) that support both text and voice. Amazon Lex uses deep learning to implement Natural Language Understanding (NLU) and Automatic Speech Recognition (ASR) and is the same engine that is used in Amazon Alexa. Amazon Lex is covered in Chapter 14.
- *Amazon Polly:* This is a text-to-speech service that you can use in your application. It supports multiple languages and a selection of voices. Amazon Polly can be used in several real-world applications, including newsreaders, games, and eLearning platforms. Amazon Polly is not covered in this book. You can learn more about Amazon Polly at `https://aws.amazon.com/polly`.
- *Amazon Rekognition:* This is a service that provides APIs for deep learning–based object detection and recognition in images and videos. Amazon Rekognition can be used in a variety of real-world applications, including content-based search in images and videos, facial biometric verification, and inappropriate-content detection. Amazon Rekognition is covered in Chapter 18.
- *Amazon Translate:* This is a document-translation service that can be used to translate text between a variety of languages. You can use it to translate unstructured text or build applications that support multiple languages. Amazon Translate is not covered in this book. You can find more information on Amazon Translate at `https://aws.amazon.com/translate/`.
- *Amazon Transcribe:* This is a speech-to-text service that can transcribe speech in audio files into text. It can be used for a variety of applications, including generating closed-caption text for a video. Amazon Transcribe is not covered in this book. You can find more information on Amazon Transcribe at `https://aws.amazon.com/transcribe/`.

Machine Learning Platform Services

These services provide you with tools to build, train, and evaluate machine learning models. Platform services do not address any specific machine learning problem out of the box; instead, you need to use them to build a machine learning solution from scratch to address the problem you are working on. Some of the services in this category are listed here:

- *Amazon Machine Learning*: This is a cloud-based service that lets you quickly build machine learning models using a wizard-based interface. Amazon Machine Learning is intended for simpler applications that can be solved using linear and logarithmic regression models and does not require the user to write any code. You can set up APIs to expose your Amazon Machine Learning models with minimal effort. Amazon Machine Learning is covered in Chapter 15.
- *Amazon SageMaker*: This is a fully managed service that lets you build, train, and deploy your own machine learning models using a variety of algorithms and frameworks on dedicated machine learning–optimized compute infrastructure. Amazon SageMaker also allows you to create notebook instances that can be used for data visualization,

exploration, and analysis. These notebook instances come pre-installed with a Jupyter Notebook server, a number of popular Python machine learning libraries, and a number of conda kernels. Amazon SageMaker is covered in Chapters 16 and 17.

- *AWS DeepLens:* This is a wireless video camera and an integrated cloud-based development platform. You can train Convolutional Neural Networks (CNNs) on the development platform and deploy the models to the wireless video cameras. AWS DeepLens is not covered in this book. You can get more information on AWS DeepLens at `https://aws.amazon.com/deeplens/`.

Support Services

In the previous sections you learned about Amazon's machine learning application and platform services. These services do not operate in isolation—often you will find yourself using a number of other AWS services during the build and deployment phases of your machine learning application. In this section you will look at a few AWS services that you are likely to encounter while building and deploying machine learning solutions:

- *AWS IAM:* Amazon Identity and Access Management (IAM) lets you securely control who can access your AWS resources, what resources they can access, and what they can do with these resources. IAM is covered in Chapter 8.
- *AWS Lambda:* AWS Lambda lets users run snippets of code without provisioning an infrastructure. This service is billed on a pay-as-you-go model, with users only paying for the execution time of their lambda code. There is no charge when code is not running. Lambda code can be set up to automatically trigger from other AWS services or called directly from any web or mobile app. AWS Lambda is covered in Chapter 12.
- *Amazon S3:* Amazon Simple Storage Service (S3) is a secure, durable, and scalable cloud-based object store. Using this service, you can store your files in the cloud. Amazon S3 is covered in Chapter 9.
- *Amazon DynamoDB:* Amazon DynamoDB is a high-performance, scalable cloud-based NoSQL database service. Amazon DynamoDB is covered in Chapter 11.
- *Amazon Cognito:* Amazon Cognito allows you to create identity profiles for your app's users and allow them to sign in to the app with their Amazon, Facebook, Twitter, or Google accounts. Once users have authenticated from the app, the app is given a token that can be used to access AWS cloud resources securely. Amazon Cognito also offers a service that allows authenticated users to sync their app data on different devices. Amazon Cognito is covered in Chapter 10.

Sign Up for an AWS Free-Tier Account

To use AWS, you need to sign up for an AWS account. If you do not already have one, you can sign up for an account under the AWS free tier. An AWS account under the free tier is designed to enable you to try some of the AWS offerings free for 12 months, subject to certain usage limits. Go to `https://aws.amazon.com/free/` to obtain information on what is included in an AWS free-tier account. Amazon Machine Learning is not available under an AWS free-tier account.

To create an AWS account under the free tier, you need to go through a five-step process. Some of the steps have multiple substeps:

1. Contact Information
2. Payment Information
3. Identity Verification
4. Support Plan Selection
5. Confirmation

Step 1: Contact Information

To start the sign-up process for an AWS account under the free tier, visit `aws.amazon.com` and click the Create an AWS Account link on the top-right corner of the page (see Figure 6.3).

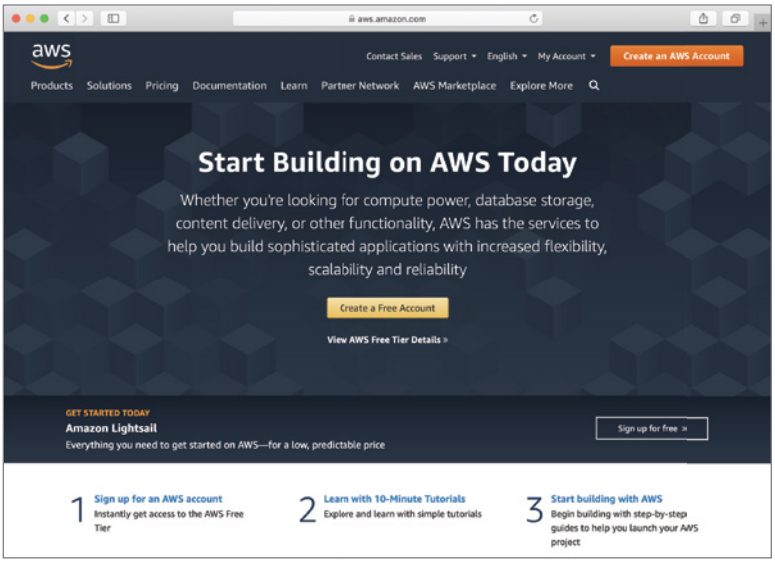


FIGURE 6.3
Amazon Web Services home page

Amazon frequently tries out new user experiences with its customers, so this page may look different from the screenshot. However, you should still be able to find the relevant option to create an AWS account on the page.

Type in a valid email address, password, and account name on the Create an AWS Account screen (see Figure 6.4) and click the Continue button. The account name is a personal identifier you can use for this account.

You will be asked to indicate if the account is for an individual or a company (Figure 6.5). Select the option appropriate to your situation. This chapter assumes you have opted to create a Personal (individual) account.

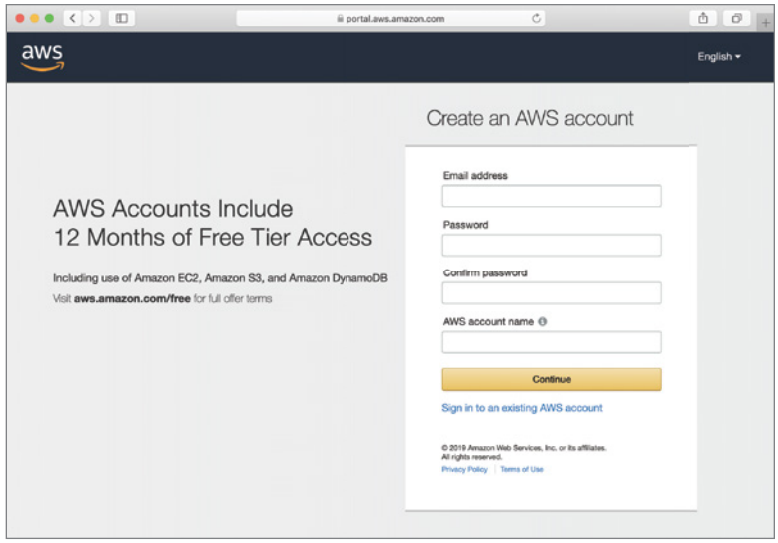


FIGURE 6.4
AWS sign-in screen

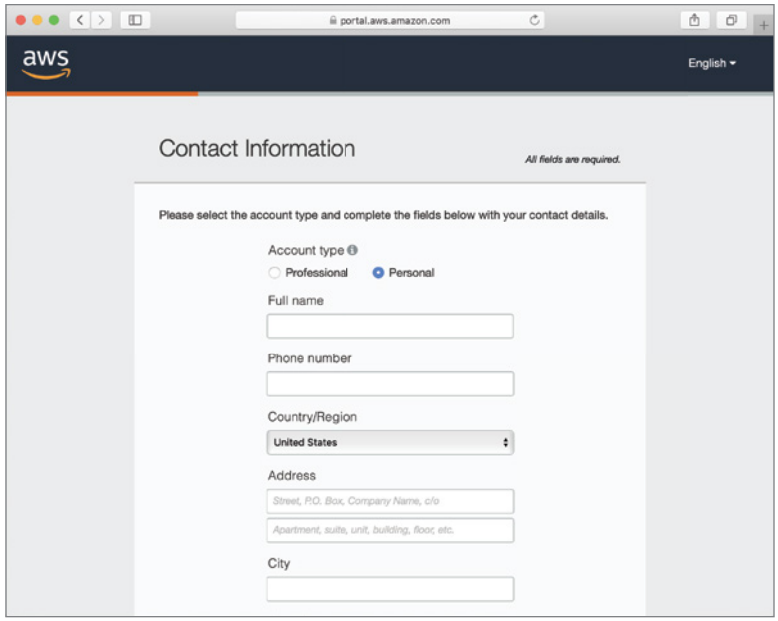


FIGURE 6.5
Contact
Information screen

You will be asked to provide contact information (including a phone number) on the Contact Information screen. You must provide a phone number that you have immediate access to and can receive a call on. Scroll down to the bottom of the page if necessary, read and accept the terms and conditions of the AWS customer agreement, and click Create Account and Continue to move to the next step.

Step 2: Payment Information

You need to provide credit/debit card details (see Figure 6.6). Although an account under the free tier provides access to some AWS services for free, not all services are included in the free tier. If you use services not included under the free-tier account or exceed the usage limits of services under the free tier, the card you provide is charged.

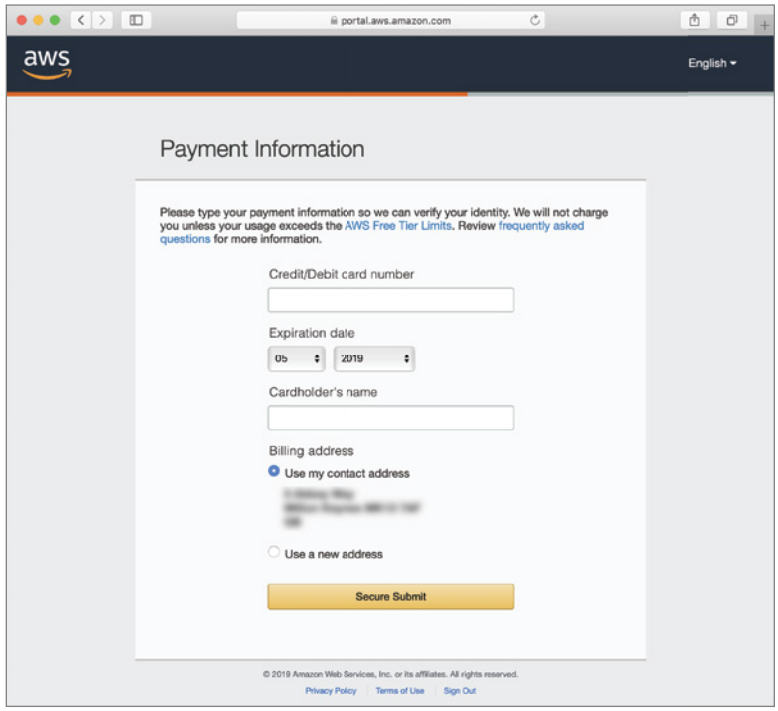


FIGURE 6.6
Payment Information screen

The precise services and options that are available under the free tier can change from time to time. Every effort will be made in this book to inform you whether an example utilizes AWS features outside those available in the free tier. To get up-to-date information on what is included in the free tier, visit `https://aws.amazon.com/free/`.

Type your credit/debit card details and click the Secure Submit button to move on to the identity verification step.

Step 3: Identity Verification

The identity verification process involves receiving a call from an automated system on a number you provide and entering a four-digit PIN into your phone when prompted. Type a telephone number and click Call Me Now (see Figure 6.7).

A four-digit PIN then appears on the web page (see Figure 6.8). You receive a call on the telephone number you have provided and are asked to enter the four-digit PIN you see on the web page.

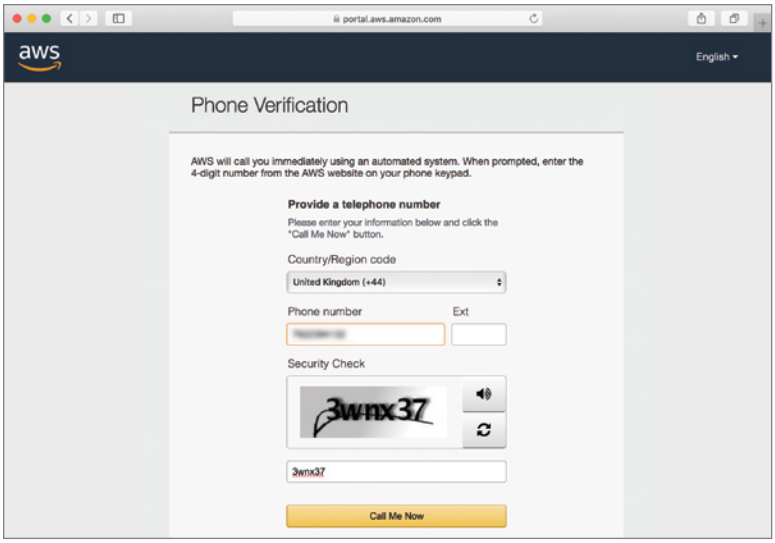


FIGURE 6.7
Phone
Verification screen

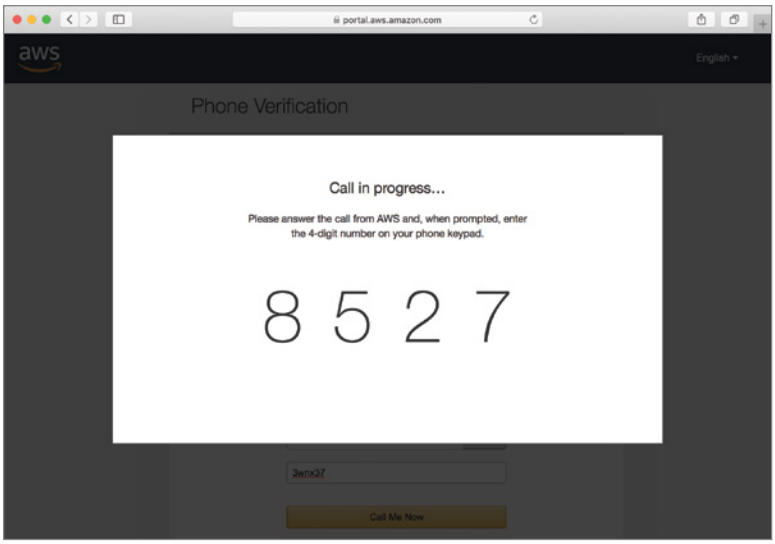


FIGURE 6.8
Phone verification PIN

The identity verification process completes once you key in the four-digit PIN over the phone. The web page refreshes to reflect this (see Figure 6.9).

Click Continue to move on to the next step of the account creation process.

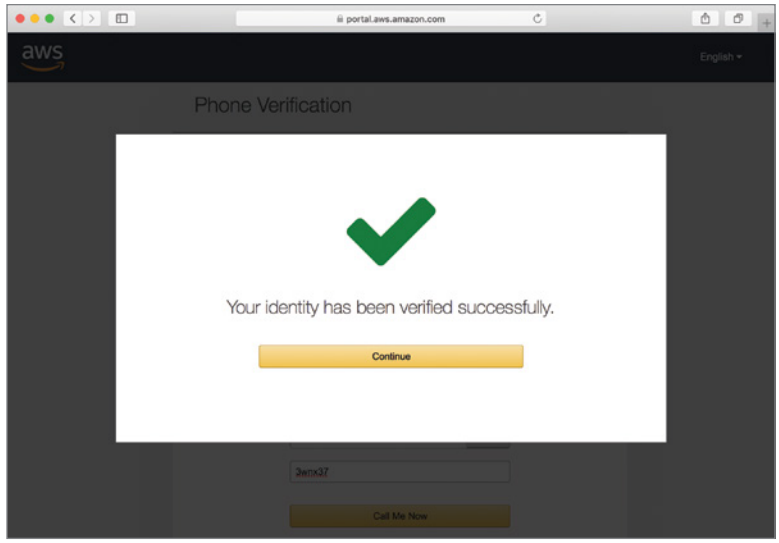


FIGURE 6.9
Completing the identity verification process

Step 4: Support Plan Selection

Select a support plan from the list of options available (see Figure 6.10). The options are:

- Basic
- Developer
- Business
- Enterprise

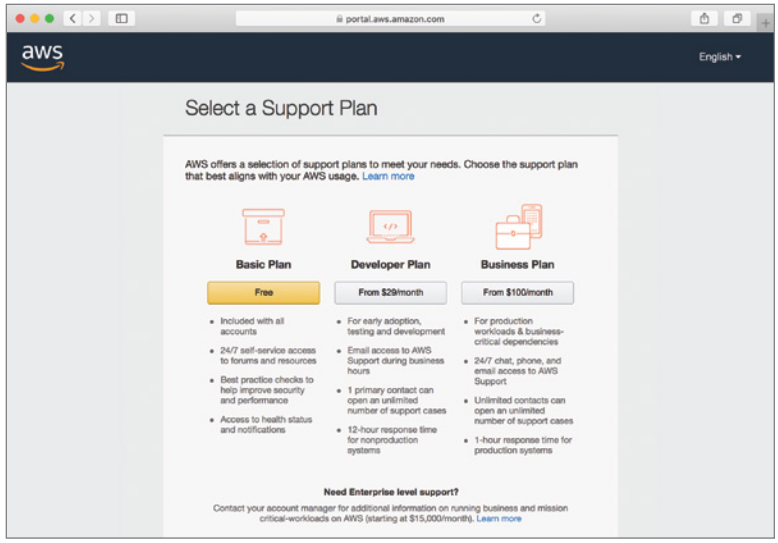


FIGURE 6.10
Support plan selection

The support plans are cumulative and differ in the level of support that you receive, with the more expensive options giving you access to an Amazon employee to answer your questions. The default, selected option is Basic, and it is free. For the purposes of this book, the Basic support plan suffices. Select the default Basic support plan and click Continue.

Step 5: Confirmation

You receive confirmation that your AWS free-tier account is now set up (see Figure 6.11).

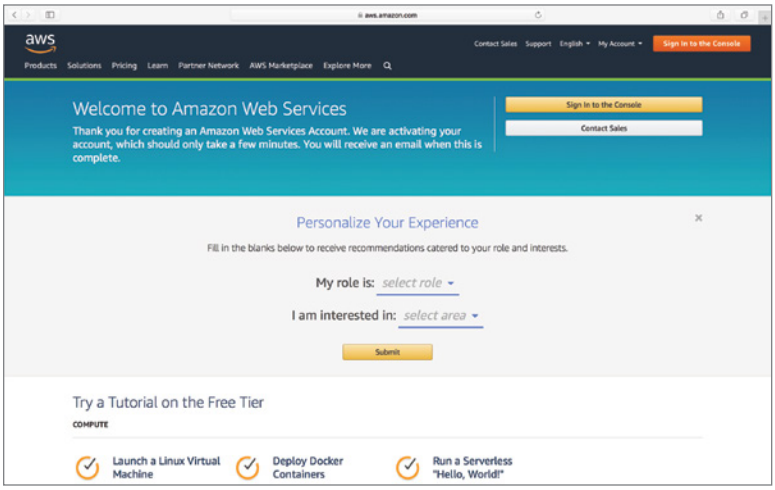


FIGURE 6.11 Completing the sign-up process

You also receive a confirmation message by email to the address you used during the sign-up process. In the next chapter you learn different ways you can access AWS and how to secure the account you have just created.

Summary

- Cloud computing is defined by the U.S. National Institute of Standards and Technology (NIST) as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."
- Cloud computing provides both cost and availability benefits to businesses.
- Common cloud-computing service models include infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS), business process as a service (BPaaS), and machine learning as a service (MLaaS).
- Cloud solutions are deployed using standard deployment models. A deployment model defines how a computing resource can be accessed, who can access the resource, and where the physical hardware is located.

- There are four distinct deployment models that are commonly used for cloud solutions: private cloud, public cloud, community cloud, and hybrid cloud.
- AWS offers over 100 different cloud services, grouped into 20 different service categories. Amazon is continuously adding to the services that are available.
- An AWS account under the free tier is designed to enable you to try some of the AWS offerings free for 12 months, subject to certain usage limits.
- Amazon's cloud-based machine learning services can be classified into two categories: application services and platform services.

Chapter 7

AWS Global Infrastructure

WHAT'S IN THIS CHAPTER

- Introduction to the AWS global infrastructure
- A tour of the AWS management console

In the previous chapter, you got an overview of the cloud services offered by Amazon, with emphasis on services for machine learning applications. In this chapter you will learn about the AWS global infrastructure. AWS physical infrastructure consists of a system of geographical regions, Availability Zones, and content distribution edge locations. Not all AWS services are available in every region.

Regions and Availability Zones

An AWS region is a physical location in the world from which cloud-based services are offered. Amazon ensures that you get to choose the region in which your data is physically located, making it easy for you to meet regional regulatory requirements.

An AWS region is divided into multiple Availability Zones (AZs) (see Figure 7.1).

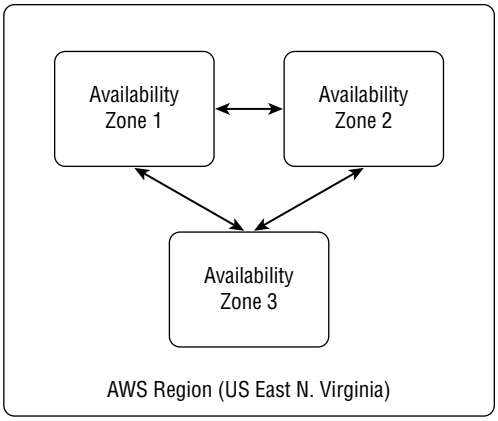


FIGURE 7.1
Multiple Availability Zones in a single region

An Availability Zone consists of one or more data centers, housed in separate facilities, each with redundant power, networking, and connectivity. These data centers are connected to each other with private fiber-optic networking and enable you to build and operate scalable and fault-tolerant applications that are not possible from a single data center.

Availability Zones let you architect applications that automatically fail over between the AZs in a region without interruption. As of this writing, nineteen regions are spread throughout the world. Additionally, Amazon has announced that it is planning new regions in Bahrain, SAR China, and Sweden. You can find a complete list of regions and Availability Zones at `https://aws.amazon.com/about-aws/global-infrastructure/`.

Table 7.1 lists the current AWS regions and AZs within each region.

TABLE 7.1: AWS Regions and Availability Zones

REGION	AVAILABILITY ZONES	COMMENTS
U.S. West (Oregon)	3	Launched in 2011.
U.S. West (Northern California)	3	Launched in 2009.
U.S. East (Northern Virginia)	6	Launched in 2006.
U.S. East (Ohio)	3	Launched in 2016.
AWS GovCloud	3	Launched in 2011. Only accessible to U.S. government employees.
Canada (Central)	2	Launched in 2016.
EU (Ireland)	3	Launched in 2007.
EU (Frankfurt)	3	Launched in 2014.
EU (London)	3	Launched in 2016.
EU (Paris)	3	Launched in 2017.
Asia Pacific (Singapore)	2	Launched in 2010.
Asia Pacific (Tokyo)	4	Launched in 2011.
Asia Pacific (Osaka)	1	Launched in 2018.
Asia Pacific (Sydney)	3	Launched in 2012.
Asia Pacific (Seoul)	2	Launched in 2016.
Asia Pacific (Mumbai)	2	Launched in 2016.

TABLE 7.1: AWS Regions and Availability Zones *(CONTINUED)*

REGION	AVAILABILITY ZONES	COMMENTS
China (Beijing)	2	Launched in 2014 in partnership with Beijing Sinnet Technology Co., Ltd. ("Sinnet"), the service operator and provider for AWS China (Beijing) Region.
China (Ningxia)	3	Launched in 2014 in partnership with Ningxia Western Cloud Data Technology Co., Ltd. ("NWCD"), the service operator and provider for AWS China (Ningxia) Region.
South America (São Paulo)	3	Launched in 2011.
Canada	2	Launched in 2016

NOTE Not all cloud-based services are available in every region. To get a comprehensive list of services available in each region, visit `https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/`.

When you start out using AWS, you will most likely base all your cloud-based applications in a single region. The default region applied to new AWS account sign-ups from the UK and the United States is U.S. East (Northern Virginia).

At some point in the future, you may want to base some of your cloud services in different regions to serve customers there more quickly. Cross-region replication is not automatically applied and usually involves additional effort and costs.

Edge Locations

An edge location is a content-distribution end point for CloudFront. Amazon CloudFront is a secure content delivery service that integrates with Amazon's S3 and allows caching of frequently used media files closer to the point of consumption. More than 50 edge locations are found around the world. You can get a complete list of edge locations at `https://aws.amazon.com/about-aws/global-infrastructure/`.

To understand how edge locations work, let's assume you have a video file in an S3 bucket in the Asia Pacific (Tokyo) region that your users want to access. This video file has a URL that your users can employ to download the video.

Every time a user decides to download your video, no matter where he is, he needs to connect across the Internet to a data center in Tokyo. This can involve significant delays depending on how far your users are from the data center (see Figure 7.2).

You can place copies of this video file in S3 buckets in additional AWS regions like Beijing and Singapore to mitigate the problem to an extent.

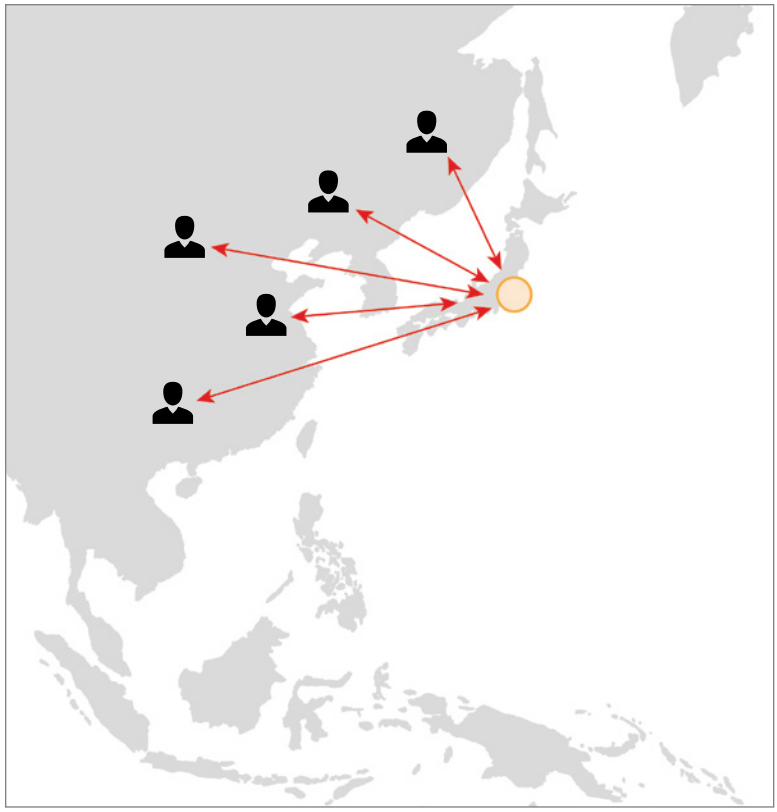


FIGURE 7.2
Geographically distant users accessing a video file from Tokyo

If you decided to use CloudFront to distribute this video file, give your users a new CloudFront URL for the video, not the original S3 URL. The first user who accesses your video still ends up connecting to a data center in Tokyo. When CloudFront receives the first request, it automatically caches this video at an edge location, closer to the user, for subsequent access. If another user in the same geographical area as the first user were to request the same file, CloudFront would use the cached copy from the edge location, resulting in significantly lower latency for the second user (see Figure 7.3).

Using CloudFront with S3 involves additional costs and setup, but if your application requires your user to download large files frequently, CloudFront can result in a significantly improved experience for your users.

Accessing AWS

In this section you learn about the different means by which you (or your application) can connect to AWS. An individual or application can connect to AWS in four ways:

- Using the AWS management console
- Using the command-line interface

- Using platform-specific developer SDKs
- Using RESTful web services

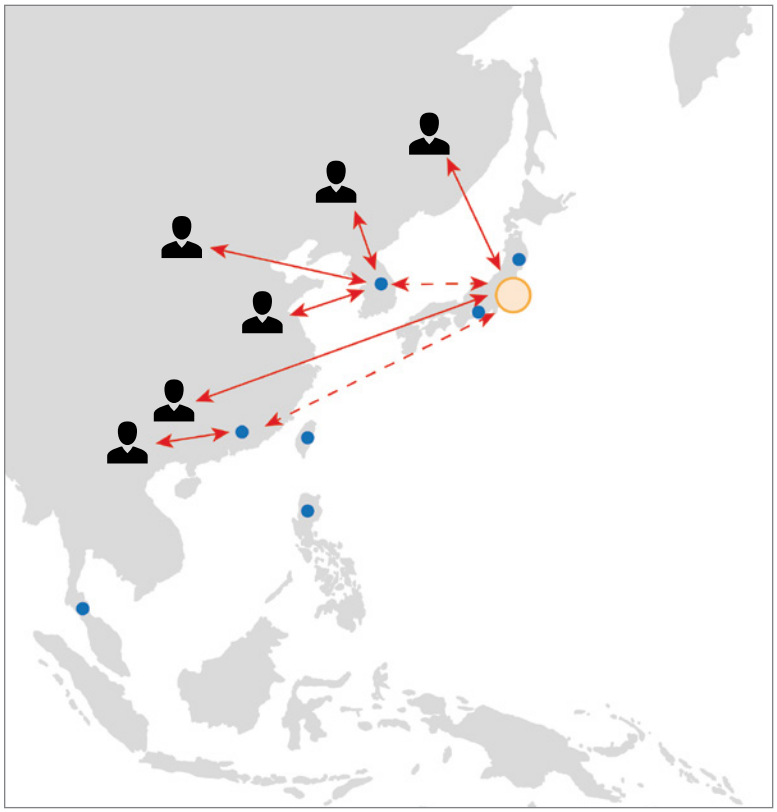


FIGURE 7.3
Edge locations can be used to cache frequently used content

Beneath its surface and true to its name, AWS is a collection of RESTful web services. You can access every service AWS offers via a RESTful web service. The management console, command-line interface, and platform-specific SDKs build upon the underlying RESTful web service API. The manner in which you choose to access AWS depends on your job function.

If you are managing or administering services, you are likely to prefer the management console's web-based, user-friendly interface. If you are a DevOps person who frequently executes scripts, you are likely to prefer the command-line interface. And if you are an app developer, you are likely to use an SDK specific to your platform if one is available.

As of this writing, developer SDKs are available for the following platforms:

- Python
- Ruby
- C++
- iOS

- Android
- Java
- .NET
- Node.js
- PHP
- Go

You can get an up-to-date list of platform-specific developer SDKs along with installation instructions and SDK documentation at `https://aws.amazon.com/tools/`. The lessons in this book primarily utilize the AWS management console and occasionally the AWS SDK for Python.

The AWS Management Console

The AWS management console is a web-based application that permits you to manage your AWS account and configure cloud-based services. Log in to the AWS management console at `https://aws.amazon.com`.

Click the Sign in to The Console link located at the upper-right corner of the website (see Figure 7.4). You are asked to provide your AWS account username and password.

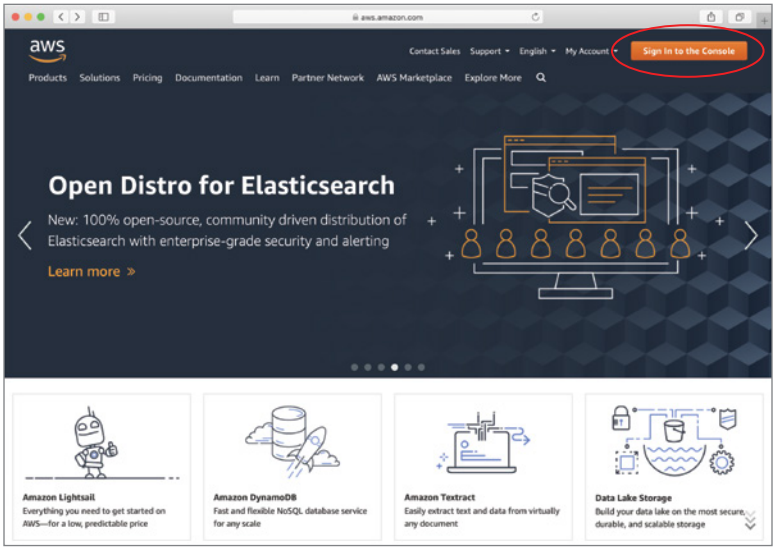


FIGURE 7.4
AWS home page

The landing page of the management console provides quick links to configuration pages for various AWS services, as well as links to training videos (see Figure 7.5). The look and feel of the landing page is constantly updated; therefore, the appearance of the landing page may differ from the screenshots.

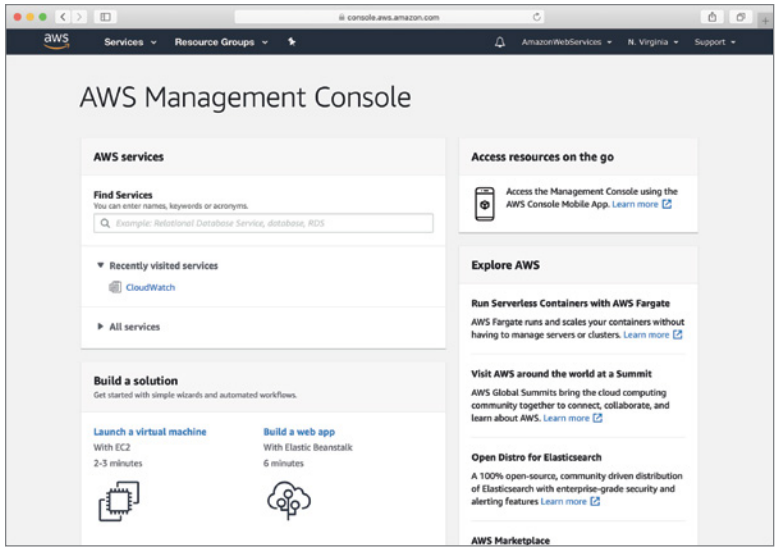


Figure 7.5
AWS management console home page

The menu bar at the top of the management console offers several useful options (see Figure 7.6). This navigation menu does not change when you move to different pages within the management console.

Figure 7.6
AWS management console menu bar

Home Menu

The leftmost icon in the menu bar is the home icon. This menu item can be used to access the home screen of the management console from any page.

Services Menu

The Services menu (see Figure 7.7) contains links to all AWS services and can be used to quickly jump to the relevant subsection within the management console for any service.

Resource Groups Menu

The Resource Groups menu allows you to access a subset of your own AWS resources (such as EC2 instances, load balancers, and databases) that have been tagged. A new AWS account has no resource groups configured; in such a case, the Resource Groups menu resembles Figure 7.8.

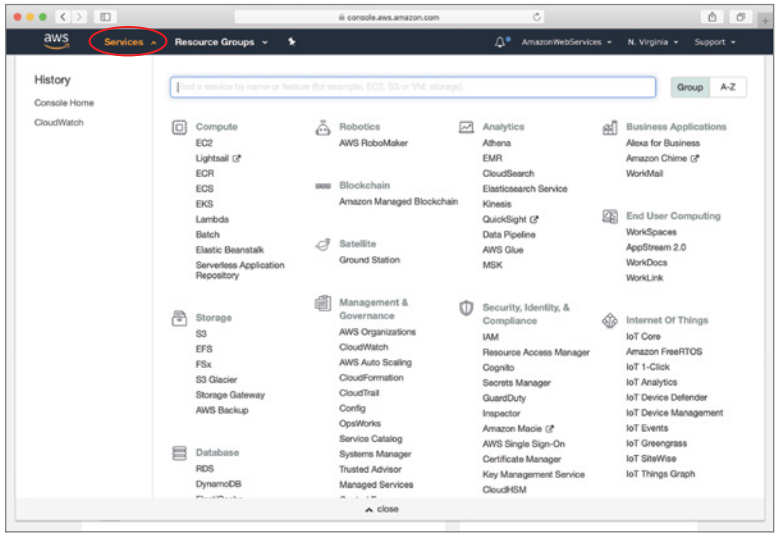


Figure 7.7
Accessing the Services menu in the AWS management console

Figure 7.8
Resource Groups menu

Let's assume that you have created a Java-based application called *CustomerAPI* and are using a number of EC2 instances to support the application in the AWS EU (London) region. If all the EC2 instances that support the application have been assigned a tag called Name with value CustomerAPI, you could then create a resource group to logically group all the EC2 instances and view all these grouped resources on one screen (see Figure 7.9).

Existing resource groups can be accessed by clicking the Saved Groups menu item under the Resource Groups menu (see Figure 7.10).

Clicking the CustomerAPI-Infrastructure resource group takes you a screen where you can see all AWS resources that are included in the group (see Figure 7.11).

Resource groups provide a convenient means to access resources quickly. Membership of a resource in a resource group is based on the value assigned to a few tags, the type of the resource, and the region in which it resides. Membership of a resource in a resource group does not mean individual resources automatically belong to a virtual network, have restricted IP addresses, or are assigned security permissions.

Account Menu

You can use the Account menu to configure account settings, access contact information and billing reports, and update security credentials. Unlike other menus discussed so far, the Account menu appears with the name used when creating the AWS account and not the name Account (see Figure 7.12).

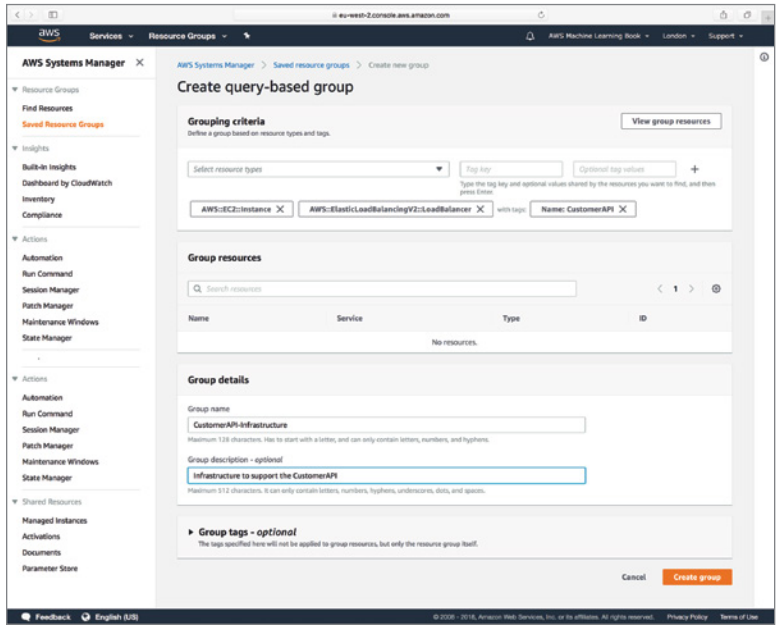


FIGURE 7.9
Creating a resource group

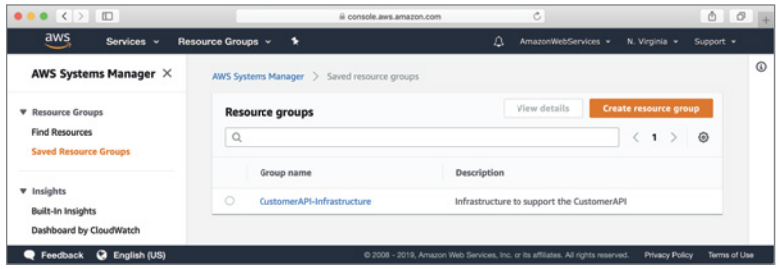


FIGURE 7.10
Tagged resources are visible in the Resource Groups menu.

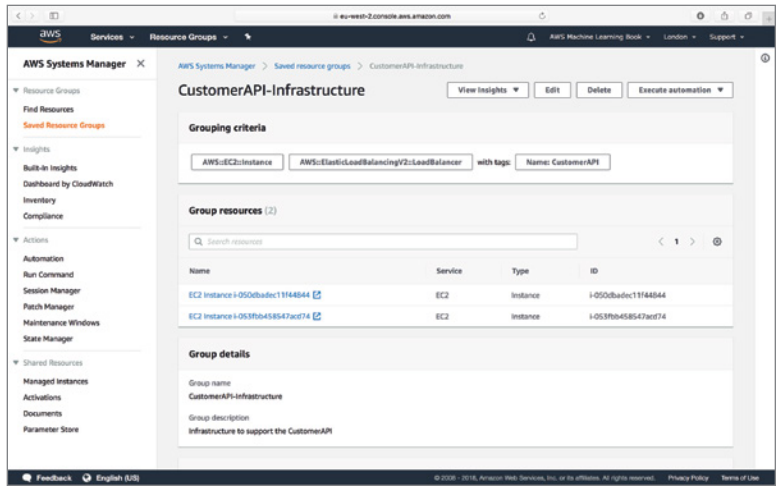
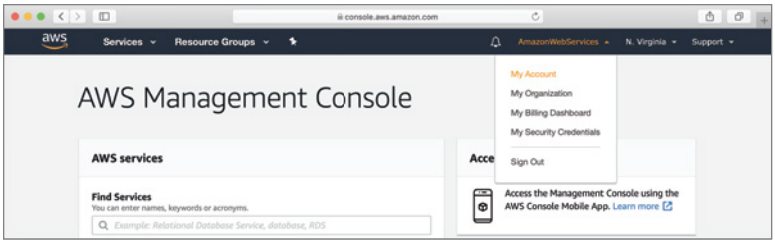


FIGURE 7.11
Resources in the CustomerAPI-Infrastructure resource group

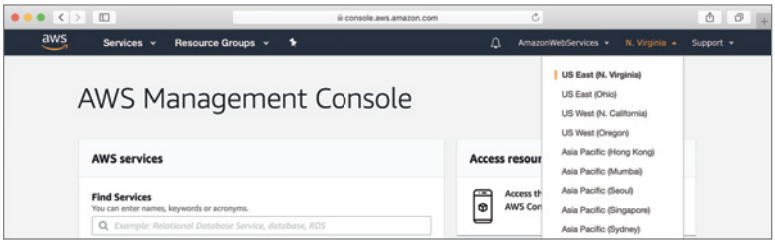
FIGURE 7.12
Account menu



REGIONS MENU

With the Regions menu, you can select the AWS region to which the management console is attached. By default, the management console is set to use the U.S. East (Northern Virginia) region, and any resources you allocate will be built there. To change regions, simply click the menu and select a different region (see Figure 7.13).

FIGURE 7.13
Regions menu



SUPPORT MENU

The Support menu is the rightmost option in the menu bar. You can use the options under this menu to contact AWS customer support and access documentation.

Summary

- Amazon uses a system of geographical regions to provide cloud-based services to end users. Not all AWS services are available in every region.
- An AWS region is a physical location in the world from which cloud-based services are offered.
- A region is divided into multiple Availability Zones. An Availability Zone consists of one or more data centers.
- An edge location is a content-distribution end point for CloudFront. Amazon CloudFront is a secure content delivery service that integrates with Amazon's S3 and allows caching of frequently used media files closer to the point of consumption.
- You can access AWS services using the AWS management console, the command-line interface, platform-specific developer SDKs, and a set of RESTful web services.

Chapter 8

Identity and Access Management

WHAT'S IN THIS CHAPTER

- Introduction to the basic concepts of Identity and Access Management (IAM)
- Creating users, groups, and roles
- Securing the root account with multifactor authentication
- Setting up a password rotation policy

Identity and Access Management (IAM) is a web service that allows you to securely manage users, configure security credentials, set up password rotation policies, configure multifactor authentication, and control which AWS resources users can access. Using IAM, you can control who can access your AWS resources, what resources they can access, and what they can do with those resources.

IAM is commonly accessed using the AWS management console, or the AWS command-line tools. In this chapter, you learn how to use the management console to set up users, groups, roles, and policies, and secure your root account.

Key Concepts

In this section, you learn some of the key concepts you will encounter when working with IAM.

Root Account

When you sign up to use AWS, you are asked to provide an e-mail address and a password as part of the sign-up process. The end of a successful sign-up journey creates a root identity using the e-mail address and password you provided. This root account has unrestricted access to all resources in your AWS account, including billing and the ability to change your password.

When you log in to the AWS management console using the e-mail address and password that were used during the sign-up process, you are, in effect, logging in as the root user.

Amazon recommends that you do not use your root account for everyday access and that you never share your root account credentials with anyone. For an additional layer of security, it is highly recommended that you enable multifactor authentication (MFA) on the root account.

For everyday use, the recommended practice is to use IAM to create separate users and employ groups and policies to set up the appropriate levels of access for these users. You learn about configuring MFA in the "Securing the Root Account with MFA" section and creating users in the "Creating a User" section later in this chapter.

User

An IAM user corresponds to a user or application in your organization. Each IAM user has a dedicated sign-in link, password, and access keys. However, IAM users are not separate AWS accounts; they live within the root AWS account (see Figure 8.1).

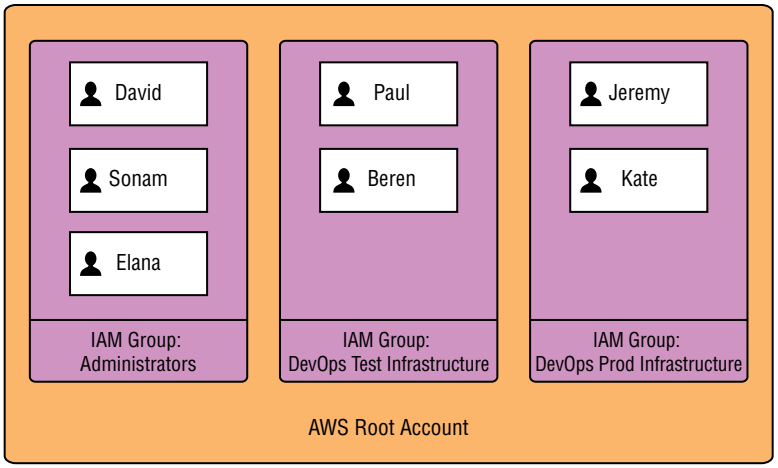


FIGURE 8.1
IAM users exist under the root AWS account.

IAM users are provided a dedicated sign-in link and password to log in to the AWS management console and a set of access keys to programmatically access AWS services. The permissions associated with an IAM user dictate which sections of the AWS management console can be accessed by the IAM user.

Because IAM users can have their own access keys, you can create IAM users for applications that need to access AWS programmatically. Therefore, an IAM user does not necessarily represent an actual individual.

Amazon recommends that you create an IAM user account with administrative privileges and utilize that user to create other IAM users for individuals/applications in your company.

Identity Federation

Identity federation allows individuals/applications who are authenticated through other means (such as Active Directory, SAML (Security Assertion Markup Language) token, Facebook) to receive a temporary IAM user account that can programmatically access AWS services. Identity federation requires the user to first sign in to the external identity provider, retrieve a token from the external identity provider, and finally present the token to IAM to receive a temporary set of credentials for AWS resources (see Figure 8.2).

Identity federation can be helpful when your users have existing identities in a corporate directory or Internet identities. IAM supports two types of identity federation. The distinction is made on the basis of where the external identity is stored:

- Enterprise identity federation
- Web identity federation

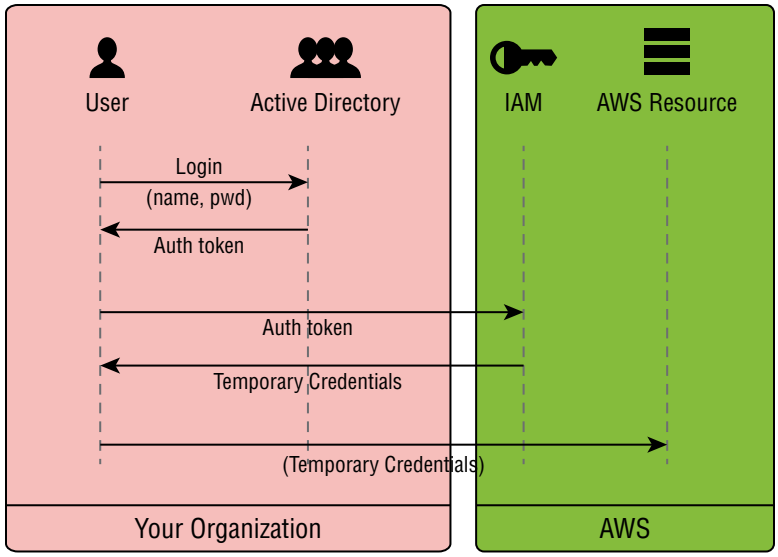


FIGURE 8.2
Obtaining temporary credentials

ENTERPRISE IDENTITY FEDERATION

In enterprise identity federation, the identity is stored outside AWS, in your own enterprise directory. Users who are already authenticated in your organization's network do not need to sign in with a new set of credentials to access AWS.

Users with identities in Microsoft Active Directory (AD) can employ AWS Directory Service to establish trust between their AD accounts and your AWS account.

Users with identities in a SAML 2.0–compatible corporate directory can configure the directory to provide single sign-on (SSO) access to the AWS management console.

WEB IDENTITY FEDERATION

In web identity federation, the identity is stored with a well-known third-party provider such as Facebook, Google, or Amazon. This is ideal if you are developing a web/mobile app that allows your users to log in using their existing Facebook, Amazon, or Google accounts. Using web identity federation in a web/mobile app does not require you to distribute long-term security credentials, or create a custom sign-in and identity management code within the app. Amazon recommends that you use Amazon Cognito Identity to manage identity federation.

NOTE For more information on identity federation, visit `http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_identity-management.html`.

Group

A group is a logical entity that can organize IAM users. Groups can have attached policies that apply to all members of the group. Groups simplify permissions management instead of managing permissions for every user of your organization; you can attach sets of users to groups and administer permissions at the group level (see Figure 8.3).

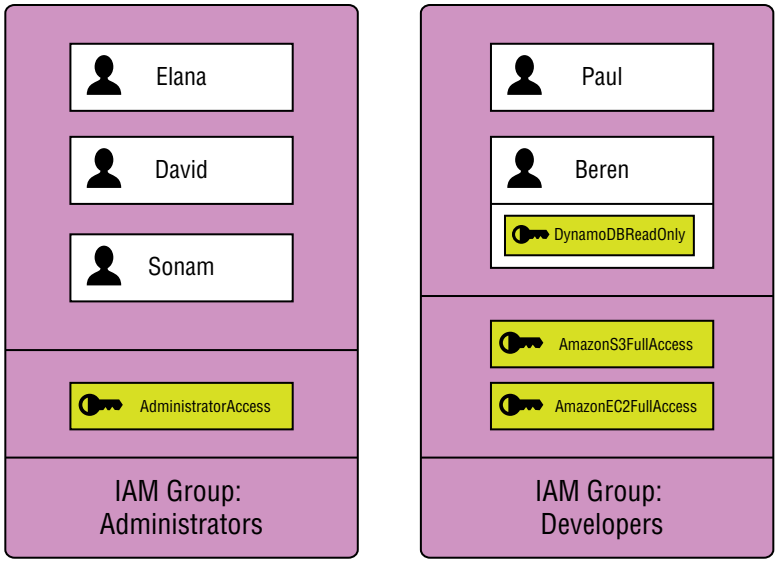


FIGURE 8.3
IAM groups contain users and permissions.

In larger organizations, it is quite common to have groups aligned with the departments and job roles associated within the company.

Policy

A policy is a JSON document that grants permissions and can be associated with a user, group, or role. A policy document lists the actions that can be performed and the resources that can be affected. When utilizing groups for user management, both users and groups can have policies attached to them. In such a case, the net set of actions a user can perform is the result of the combination of the policies applied to the user and those inherited via group membership. Policies can be one of two types:

- *User-based policy:* A user-based policy is attached to a user. It describes the actions the user can perform and the resources the user can access.
- *Resource-based policy:* A resource-based policy is attached to a resource. It describes the users who can access the resource and the actions they can perform on the resource. Not all AWS services support resource-based policies. The main use of a resource-based policy is to allow cross-account access to your AWS resources. Cross-account access occurs when IAM users from another AWS account access resources from your AWS account.

Role

A role is an identity object similar to a user that can have policies attached to it. However, unlike a user, a role does not have credentials associated with it and is not uniquely associated with a single person. A role can be assumed by a person or a service.

Roles are primarily for providing an AWS service access to another AWS service in your account, such as to allow an EC2 instance access to an S3 bucket. Roles are also involved when IAM users created in another organization's AWS account, or users authenticated with other identity providers, want to access your AWS services.

When a user assumes a role, the permissions associated with the role temporarily supersede his existing permissions. When the user stops using the role, the user's original permissions are restored.

When a role is assumed by a service (such as EC2), the service is given a temporary set of access credentials for the role. Thus, if you wanted your EC2 instance to access objects from a DynamoDB database, you would not create a username/password for the EC2 instance. Instead, you would create a role with the correct permissions and assign the EC2 instance to the role.

A role has two separate policies associated with it:

- *Trust policy:* This policy specifies who is allowed to assume the role.
- *Permissions policy:* This policy specifies what actions and resources the person who assumes the role is allowed to use.

Common Tasks

In this section, you learn to manage users, groups, roles, and permissions using IAM and the AWS management console. The topics in this section require access to the AWS management console. If you have not created users under your root account, you need to log in to the AWS management console with your root account credentials (see Figure 8.4).

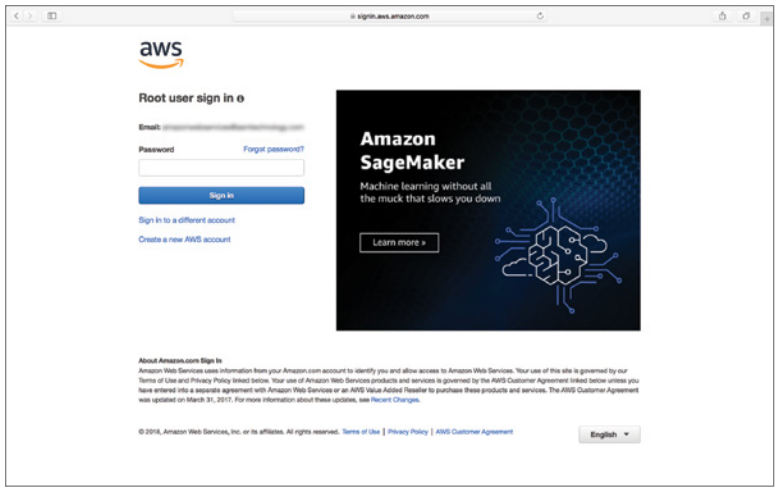


FIGURE 8.4
Root account login screen

If you have already created an IAM user account for day-to-day administrative tasks, you should use the dedicated sign-in link for the user account to access the AWS management console (see Figure 8.5).

You cannot use your root account credentials to sign in through a dedicated sign-in link. IAM accounts are not restricted to the region that is currently set up in the management console (see Figure 8.6); they apply across all AWS regions. While employing the management console to configure IAM, it is good practice to select a region that is physically closest to you. Doing so ensures you experience the least latency while using the management console.

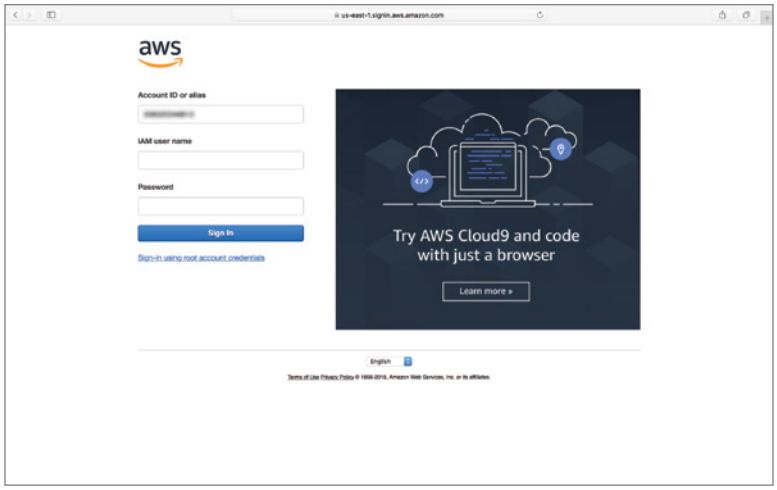


FIGURE 8.5
IAM user-specific login screen

FIGURE 8.6
AWS management console region selector

Once you have logged in to the AWS management console, select the IAM link from the Services drop-down menu (see Figure 8.7).

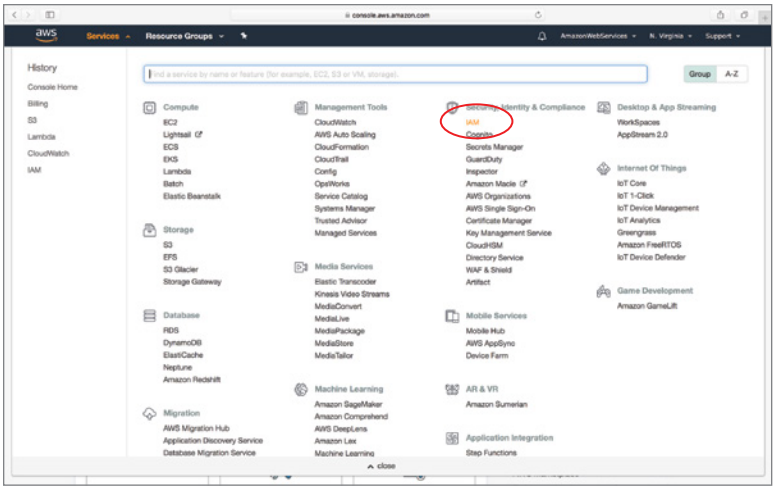
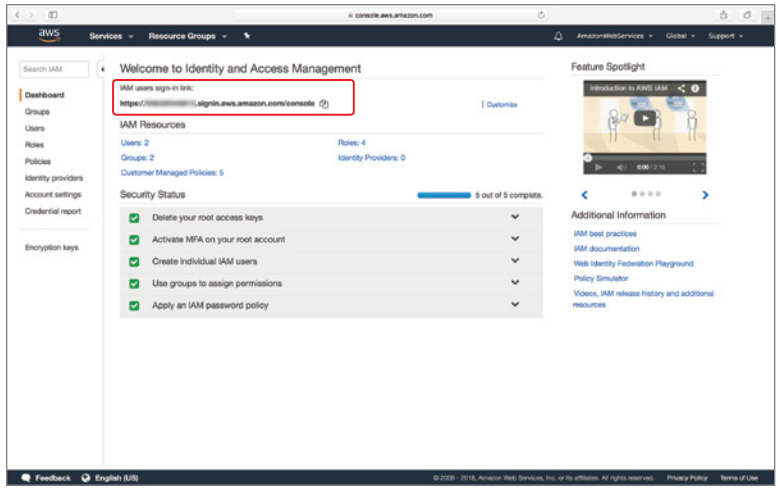


FIGURE 8.7
Accessing the IAM management console

The IAM dashboard appears. At the top of the dashboard, you see the IAM users sign-in link. This is the dedicated sign-in link that you should give your users (see Figure 8.8).

FIGURE 8.8
User-specific IAM sign-in link



Below the IAM users sign-in link, you are presented with a summary of the number of IAM resources that have been created, as well as a checklist of common tasks you need to perform to secure your IAM account (see Figure 8.9).

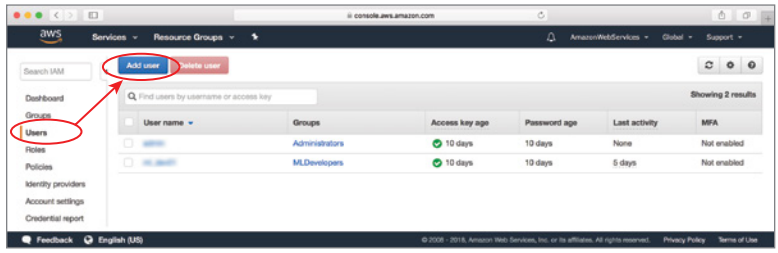
FIGURE 8.9
IAM resource dashboard

Creating a User

Follow these steps to create a user:

1. Click the Users link in the IAM dashboard to load the user management page. Click the Add User button to start the process of creating a user under your root account (see Figure 8.10).

FIGURE 8.10
Creating an IAM user



2. Specify a username as well as the access type for the new user (see Figure 8.11).

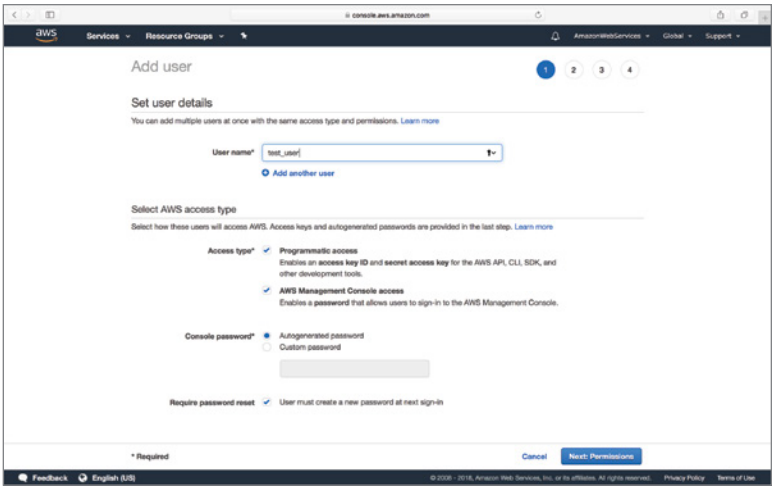


FIGURE 8.11
User details screen

The access type has two options. A user can have either one or both access types enabled:

- *Programmatic access:* Checking this option generates a set of credentials (access key ID, secret access key) that can connect to AWS services from the command-line tools, device-specific SDKs, or RESTful API.
- *AWS Management Console access:* Checking this option generates a password that enables you to log in to the AWS management console using the dedicated users sign-in link.

If you allow access to the AWS management console, you have the option to provide a custom password as well as require the user to change the password on first login. Click Next to proceed to the next step.

3. Once you have specified a username and access type, you are asked to configure permissions for the user (see Figure 8.12).

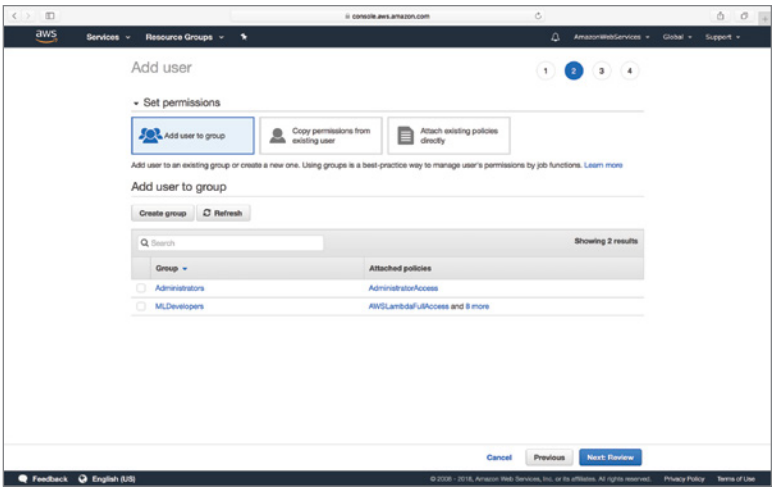


FIGURE 8.12
Configuring user permissions

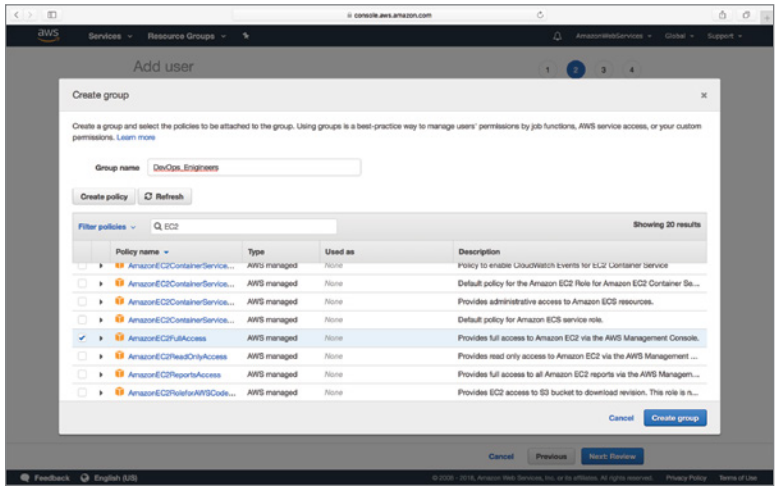
Permissions are represented by JSON documents called policies. You can assign permissions to a user in three ways:

- *Adding the user to a group:* You can add the user to a group and assign permissions to the group. Permissions set on a group apply to the users within the group.
- *Directly attaching policies:* You can attach one or more policies directly to a user.
- *Copy permissions from an existing user:* You can copy the policies associated with an existing user.

Using groups to manage permissions is the recommended approach. Ensure the option labeled Add User To Group is selected, and click the Create Group button.

4. Type a name for the new group and select one or more policies to associate with the group. Click the Create Group button to finish creating the group. In this example the name of the group is DevOps_Engineers with a single policy called EC2FullAccess (Figure 8.13).

FIGURE 8.13
Creating a new group



5. On clicking the Create Group button, you will be taken back to the previous screen and will see your new group listed alongside existing groups (Figure 8.14).

You can select multiple groups from the list, in which case the combined effect of the policies across the selected groups will apply to the user.

The list of groups also contains information on the policies associated with the group. Clicking a policy name will load the policy editor in a new window (Figure 8.15).

A policy is expressed as a JSON document, formatted according to the rules of the IAM Policy Language. You can learn more about the IAM Policy Language at `http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies.html`.

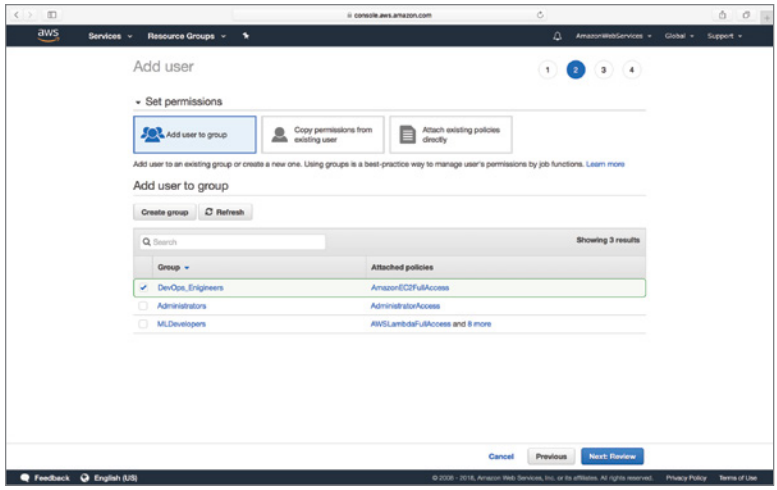


FIGURE 8.14
The new group appears alongside existing groups.

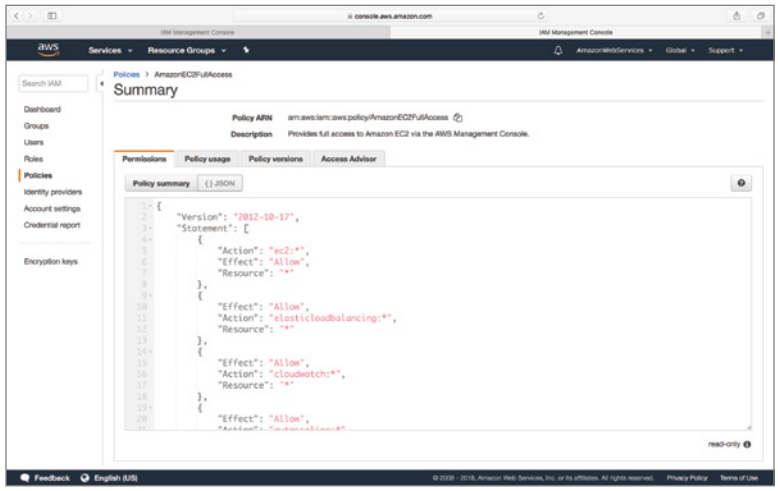


FIGURE 8.15
The EC2FullAccess policy loaded in the policy editor

The JSON document for the EC2FullAccess policy contains the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "ec2:*",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "elasticloadbalancing:*",
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": "cloudwatch:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "autoscaling.amazonaws.com",
                        "ec2scheduled.amazonaws.com",
                        "elasticloadbalancing.amazonaws.com",
                        "spot.amazonaws.com",
                        "spotfleet.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Each policy document begins with a string key called `"Version"`, which in this case is the date this AWS-provided policy was created.

A policy also always has an array of statements. Each element of this array describes the ability (or inability) to perform an action on a resource. In case of the EC2FullAccess policy, there are several statements. One such statement allows access to all Elastic Load Balancer (ELB) resources:

6. After you have selected the groups to which you want the user to belong, click the Next button to display the screen shown in Figure 8.16, where you review the settings for the new user you are about create.

7. Click the Create User button to finish creating the user.

You are presented with a confirmation screen like the one in Figure 8.17 that contains the name of the user just created as well as access credentials.

This is the only opportunity to record these access credentials somewhere safely and share them with the user for whom you have created the user account.

8. Use the Download .csv button to download the full set of credentials for the user, and click the Close button to go back to the IAM home screen.

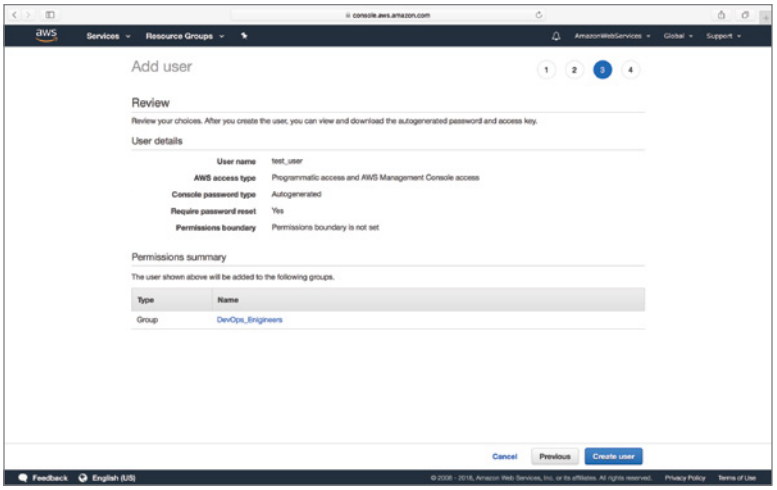


FIGURE 8.16
Review user settings screen

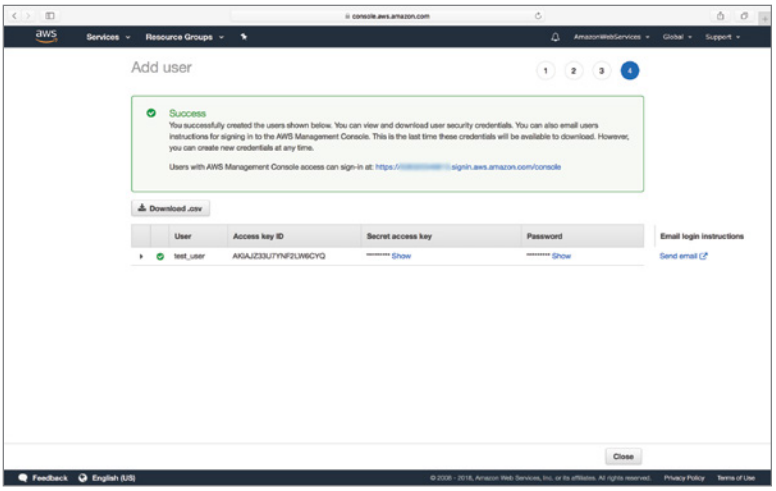


FIGURE 8.17
User confirmation screen

Modifying Permissions Associated with an Existing Group

To modify the permissions that apply to an existing group, click the Groups link on the IAM home screen and select the name of the group from the list of existing groups (Figure 8.18).

You are presented with a summary page that provides options to modify the permissions that apply to the group (see Figure 8.19).

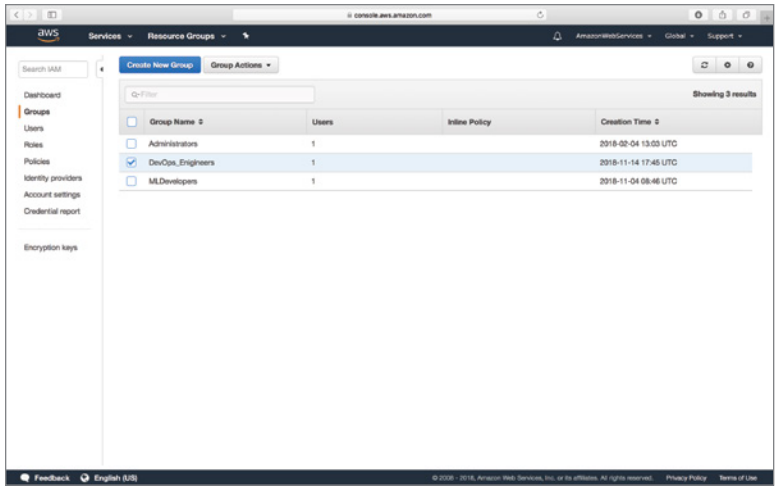


FIGURE 8.18
List of groups

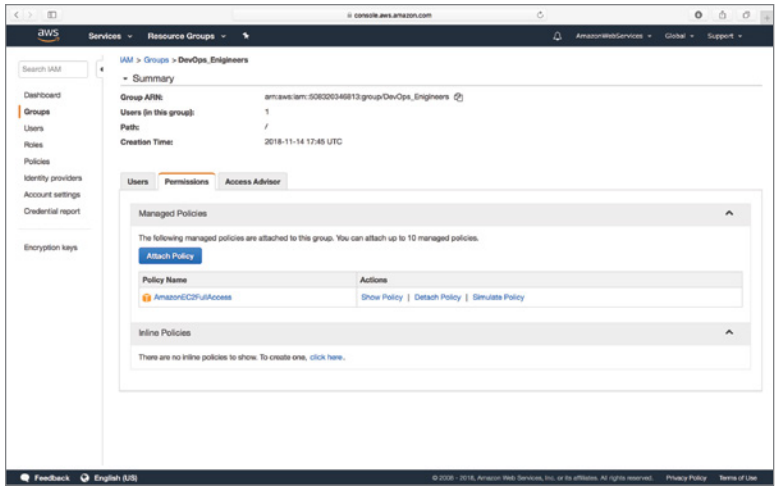


FIGURE 8.19
Group permissions summary

Creating a Role

To create a new role, complete the steps in this section:

1. Click the Roles link on the IAM home screen and click the Create Role button (see Figure 8.20).
2. Select the type of entity that will assume this role. Recall that a role has two policies attached to it: a trust policy and a permissions policy. A trust policy defines who can assume the role. IAM allows you to select from one of four types of entities:
   - *AWS Service:* Select this option if the role is to be used by an AWS service, such as Amazon EC2. Roles that are used by AWS services are also known as service roles.

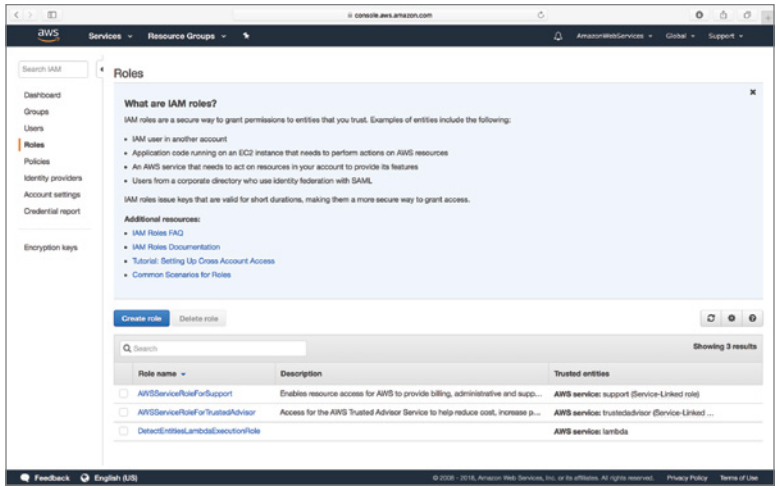


FIGURE 8.20
Creating a new role using the IAM console

- *Another AWS Account:* Select this option if the role is to be used for IAM users who belong to a different AWS account.
- *Web Identity:* Select this option if the role is to be assumed by users who have been authenticated using an OIDC provider.
- *SAML 2.0 Federation:* Select this option if the role is to be assumed by users who have been authenticated using SAML 2.0.

The example in this section creates an AWS service role that will be assumed by EC2 instances, and will allow access to Amazon DynamoDB resources in the same account. Select AWS Service as the entity type and click EC2 from the list of available AWS service names (Figure 8.21). Click the Next button to proceed.

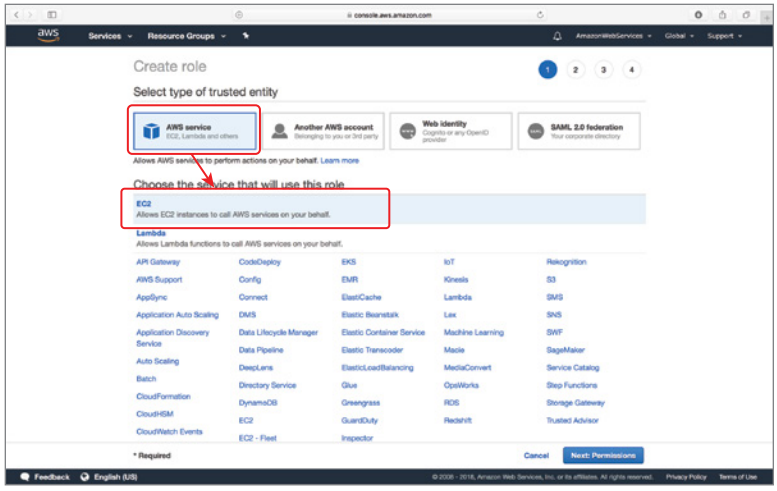


FIGURE 8.21
Creating a service role for EC2 instances

3. Configure the permissions policy of the role. The permissions policy defines what actions a user or service can perform once it has assumed this role. You can attach up to 10 permissions to a single role.

   Type the word **Dynamo** in the search field, select the policy called AmazonDynamoDBFullAccess, and click the Next button (Figure 8.22).

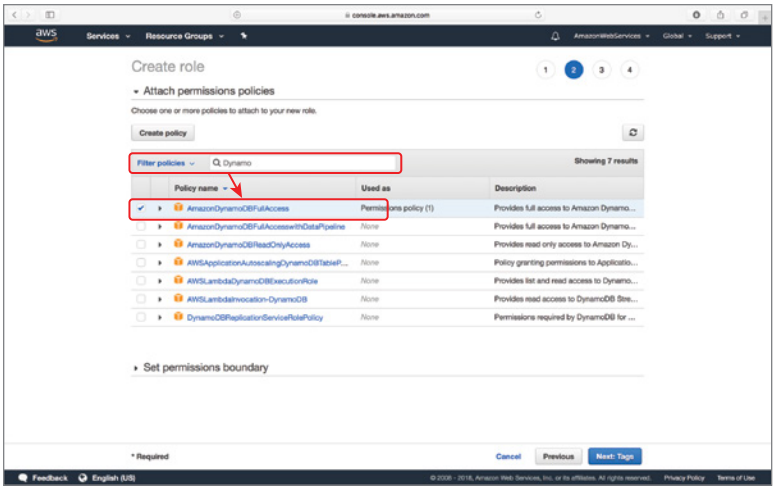


FIGURE 8.22
Attaching a policy to a role

4. The next screen allows you to create optional tags and associate them with the role (Figure 8.23). Tags are optional key-value pairs that can be associated with a role; a role can have up to 50 tags. The role that is being created in this section does not have any tags associated with it; however, you are free to add tags if you wish. Click the Next button to move to the next step.

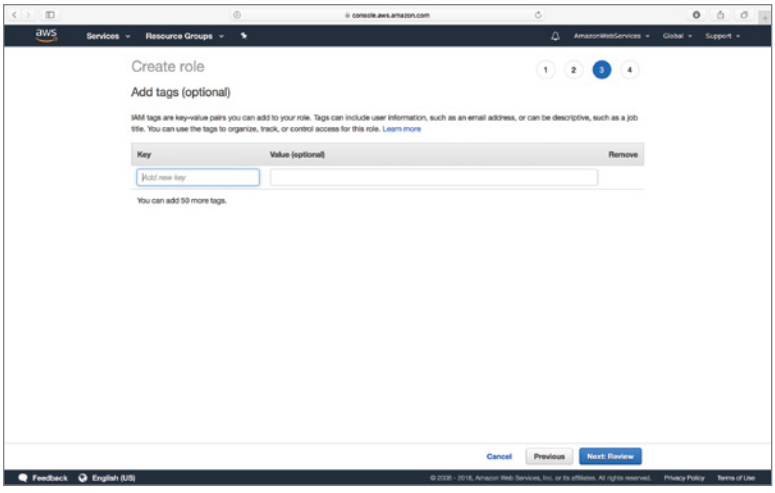
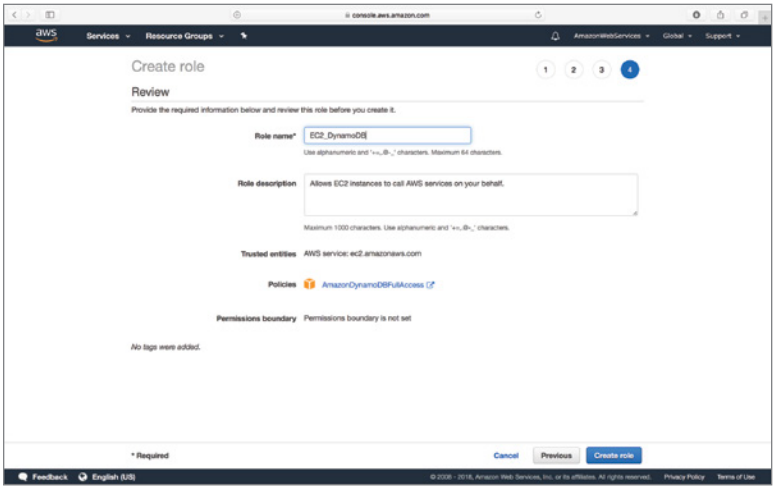


FIGURE 8.23
You can associate up to 50 optional tags with a role.

5. The next screen provides a summary of the new role and allows you to provide a name for the role. Role names can be up to 64 characters in length and cannot be edited once the role is created. When you have finished typing the name of the role, click the Create Role button to finish creating the role (see Figure 8.24).

FIGURE 8.24
Review new role screen



Securing the Root Account with MFA

In this section, you learn to secure your root AWS account using multifactor authentication (MFA). The concept of MFA involves using at least two factors to authenticate a user. The factors are:

- Something you know (such as a password)
- Something you have (such as a hardware token, or a smartphone app)
- Something you are (such as a biometric fingerprint, or voice)

MFA is commonly used to secure sensitive resources and is increasingly being adopted by systems that you use as part of your day-to-day life, including Internet banking. It is highly recommended that you enable MFA where possible. When enabled, MFA adds an additional security step when someone attempts to use root account credentials to log in to the AWS management console. In this additional step, the individual is asked to provide a temporary and unique six-digit numeric code that an authentication device generates.

1. To configure MFA on your root account, log in to the AWS management console using your root account credentials and navigate to the IAM dashboard. Click the chevron beside the Activate MFA row under the Security Status dashboard (Figure 8.25).

   The Activate MFA row will expand to reveal the Manage MFA button. When you click this button, a dialog box may appear reminding you that it is best practice to use a dedicated IAM user with administrative privileges as opposed to the root account (Figure 8.26). Click the Continue To Security Credentials button to proceed.

FIGURE 8.25
Accessing MFA settings

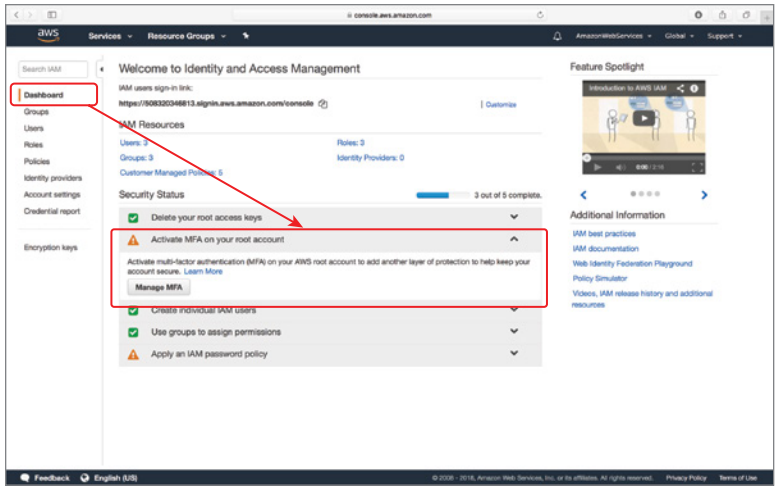
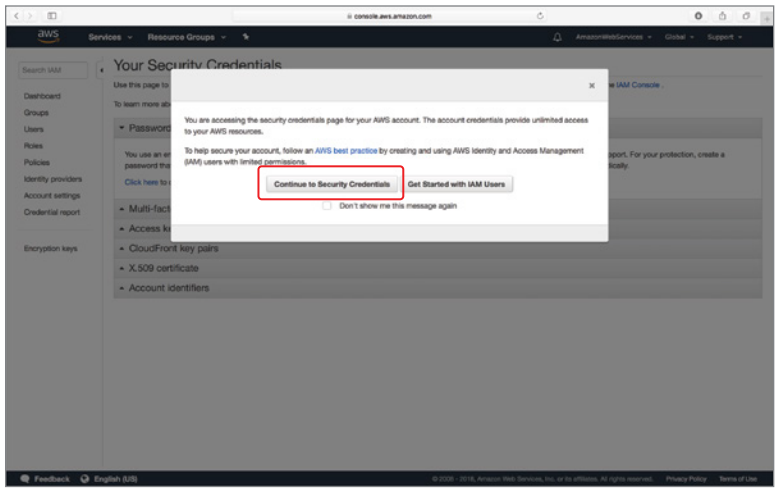


FIGURE 8.26
Configure security credentials warning



2. You will be taken to the security credentials page. Expand the multifactor authentication chevron to reveal the Activate MFA button (which will be in blue now) and click the Activate MFA button (Figure 8.27).

3. When you click this button, a dialog box will appear asking you to choose the type of MFA device you want to activate. You can use one of two types of MFA devices:

   - *Virtual MFA Device:* A virtual MFA device is a software application like Google Authenticator that runs on smartphones. Google Authenticator is available for both iOS and Android devices.
   - *U2F Key:* U2F is an open authentication standard created by the FIDO alliance. A U2F key is a USB key that you attach to your computer and physically interact with (instead of entering a code).

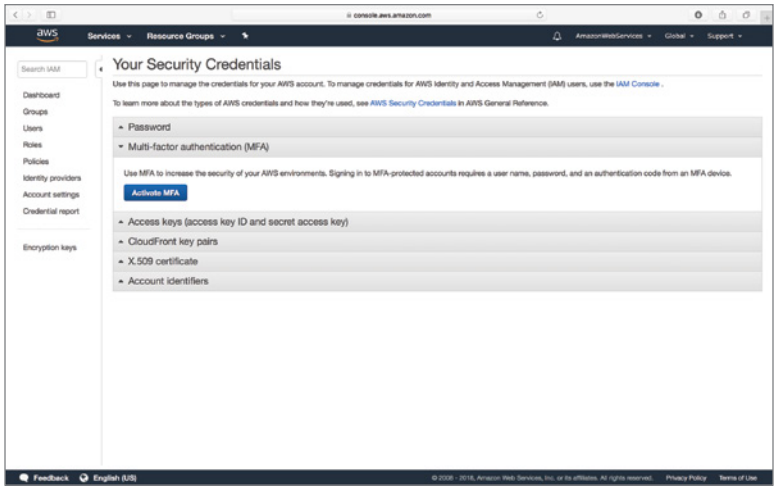


FIGURE 8.27
The Activate MFA button is enabled.

- *Hardware MFA Device:* This is a physical authentication device you need to have purchased before activation. As of this writing, the device must be provided by Gemalto—a third-party MFA device manufacturer whose devices are compatible with AWS MFA.

NOTE Because Google Authenticator runs on a smartphone that you will probably carry around with you every day, there is always the possibility of the phone being stolen and the MFA authenticator being compromised.

A hardware MFA device (or U2F key) that is locked away is always more secure than a virtual MFA device on a smartphone that you carry with you.

To find out more about purchasing a hardware MFA device and other compatible virtual MFA apps, visit `https://aws.amazon.com/iam/details/mfa/`.

In this example, we are going to use the Google Authenticator app. Select the Virtual MFA Device option and click the Continue button (see Figure 8.28).

4. The next screen contains a QR code and two text fields. Launch the Google Authenticator app on your smartphone and scan the QR code. If the QR code is not visible on the web page, click the Show QR Code link to reveal it (Figure 8.29).
5. A six-digit code appears on the app. Type this code into the Authentication Code 1 field below the QR code. Wait for a minute, and a new code appears in the app. Type the new code into the Authentication Code 2 field and click the Assign MFA button to finish enabling MFA on the root account.

Multifactor authentication is not a feature that is exclusive to the root account. Any IAM user can choose to enable MFA on his own user accounts, and enabling MFA on the root account does not automatically enable MFA for IAM user accounts under the root account.

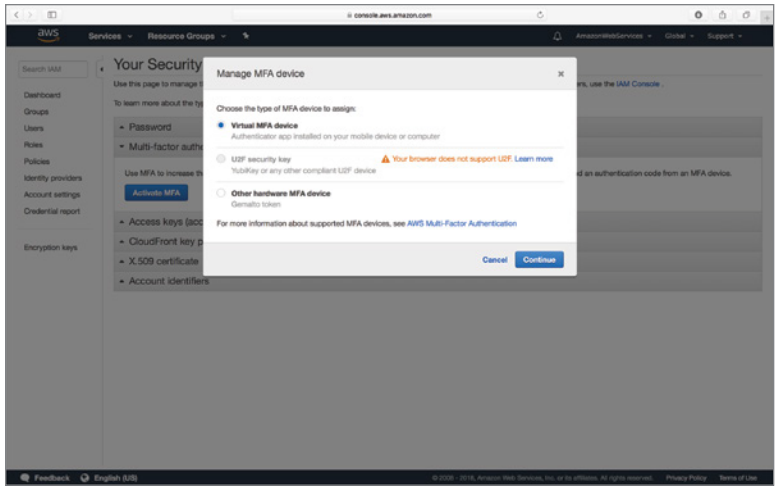


FIGURE 8.28
Choosing the MFA device type

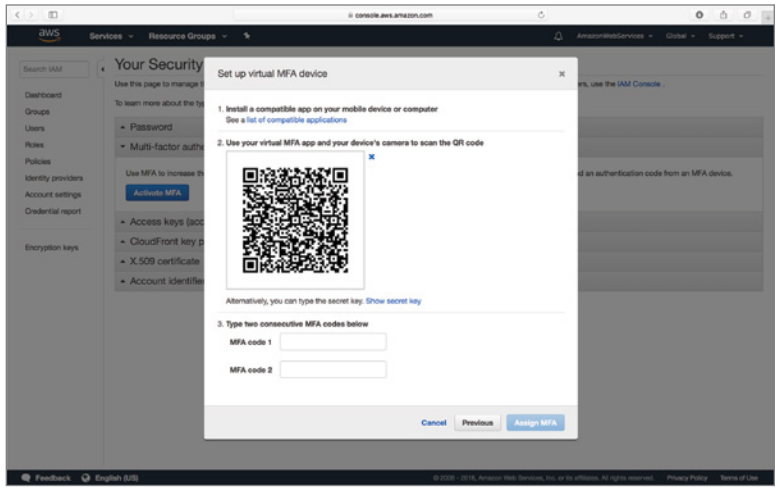


FIGURE 8.29
Configuring a step-up authenticator

Now that you have enabled MFA on the root account, if you log out of the AWS management console and try to sign in again, you will be asked to provide an authentication code after you have entered your root account credentials. Launch Google Authenticator on your smartphone and type the six-digit authentication code from Google Authenticator into the text field on the web page.

Setting Up an IAM Password Rotation Policy

In the following steps, you configure a password rotation policy for your IAM users:

1. Log in to the AWS management console using your root account credentials and navigate to the IAM dashboard. Expand the Apply An IAM Password Policy menu item and click the Manage Password Policy button.

2. Select from the available options to create a suitable IAM password rotation policy and click the Apply Password Policy button (Figure 8.30).

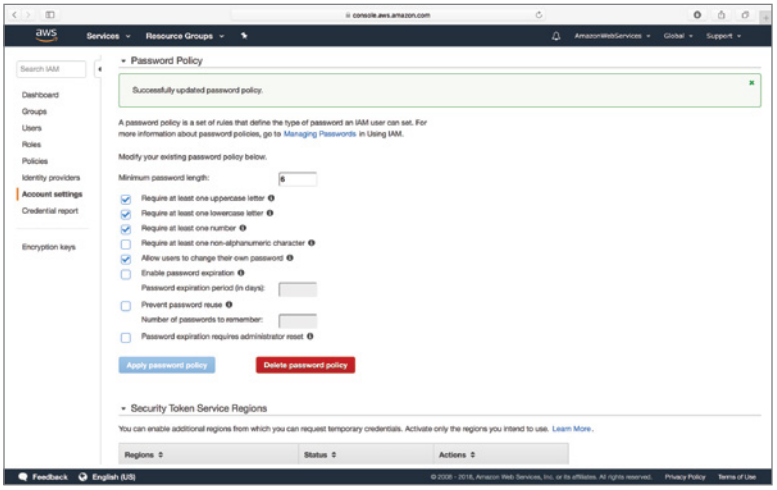


FIGURE 8.30 IAM password policy settings

Summary

- Identity and Access Management (IAM) is a service that allows you to manage users and what resources they can access.
- The IAM service is typically accessed using the AWS management console.
- When you sign up to AWS you receive a root identity using the e-mail address and password you provided.
- Your root AWS account has unrestricted access to all resources in your account and must not be used for everyday use.
- For everyday use, you should set up IAM users, groups, and policies to set up appropriate access levels for these users.
- An IAM user can represent an individual or application.
- Identity federation allows individuals/applications who are authenticated through other means (such as Active Directory, Facebook) to receive a temporary IAM user account that can be used to programmatically access AWS services.
- A group is a logical entity that can organize IAM users. Groups can have attached policies that apply to all members of the group.
- A policy is a JSON document that grants permissions to a user, group, or role.
- A role is an identity object similar to a user but does not have credentials associated with it. A role has a set of permissions associated with it.
- Roles are primarily for providing an AWS service access to another AWS service in your account.

Chapter 9

Amazon S3

WHAT'S IN THIS CHAPTER

- Introduction to the basic concepts of Amazon S3
- Learn to create Amazon S3 buckets
- Learn to upload objects into Amazon S3 buckets
- Learn to download objects from Amazon S3 buckets
- Learn to interact with Amazon S3 using the AWS CLI tools

NOTE Amazon occasionally updates the user interface of the Amazon S3 management console. As a result, some of the screenshots in this chapter may not match what you see in your web browser. However, the general concepts that you will learn in this chapter will still be applicable.

Amazon Simple Storage Service (S3) is a highly reliable web service that allows you to securely store and retrieve object data in the AWS cloud. After Amazon EC2, Amazon S3 is one of the most commonly used services. Data on Amazon S3 is spread across multiple devices and availability zones within a region automatically.

Amazon S3 is an object-based storage service (not block-based). It is ideal for storing files but cannot be used to install an operating system; thus, it cannot provide the storage for an EC2 instance.

Data within Amazon S3 is stored using a key-value system, with keys being globally unique. There is no limit to how much data can be stored on Amazon S3; however, the maximum size of a single file cannot exceed 5 TB.

Key Concepts

In this section, you learn some of the key concepts you will encounter when working with Amazon S3.

Bucket

A bucket is a folder on Amazon S3 where you can store your files. Bucket names are globally unique; therefore, no two users can own a bucket with the same name. Amazon S3 does not internally implement a hierarchical file system similar to what you encounter on your computer's operating system. All files across all Amazon S3 buckets are stored within a global flat file system. However, your bucket names can contain the forward path delimiter character (/). Therefore, you can name your buckets in such a way so as to create the appearance of a nested folder structure.

For each bucket you create, you can set up permissions that control who can access the bucket and what they can do with the bucket. Each object you store in an Amazon S3 bucket has an object key and metadata associated with it.

Object Key

An object key is a sequence of UTF-8 characters that identifies an object in an Amazon S3 bucket. The key is assigned to the object when it is first uploaded into an Amazon S3 bucket and can be up to 1024 bytes long.

The key name is basically the name of the file you have uploaded to the bucket. Amazon S3 internally stores data alphabetically, which means files with similar names are stored next to each other on the same physical disks. This can be an important factor to consider if the files you are planning on storing in Amazon S3 are going to have some kind of sequential naming scheme, or share a common prefix with each other. If this is the case, you could encounter performance bottlenecks when reading the data out of Amazon S3; you may want to consider naming the files differently or adding a short random string, or a timestamp to the start of the filename.

Object Value

The object value is the data that you are storing. It is a sequence of bytes and can be up to 5 TB in length.

Version ID

The version ID is a string value that identifies the version of the object. Amazon S3 assigns a version ID when you upload an object to a bucket. If object versioning is subsequently enabled, every update to the object creates a new version ID. Together, the object key and the version ID uniquely identify an object.

Storage Class

Each object in Amazon S3 has a storage class associated with it. The storage class determines how Amazon S3 stores the data for the object and if you will be charged additional costs to read the data. Amazon S3 offers the following storage classes:

- *Standard:* This is the default storage class for objects when they are uploaded to Amazon S3. It is ideal if your use case requires high reliability, durability, and quick access times. This storage class has been designed for 99.99% availability, 99.999999999% durability. Data is stored redundantly across devices and facilities and can withstand the loss of two facilities simultaneously.
- *Standard - IA:* IA is an acronym for *infrequently accessed.* This storage class is designed for long-lived objects that are accessed less frequently, costs less to use than the Standard storage class, and is designed to provide the same availability and durability as the Standard storage class. You can access your objects in real time, but each retrieval has an additional charge associated with it.
- *Reduced Redundancy Storage (RRS):* This storage class is designed for noncritical objects that can easily be reproduced. The objects cost less to store than the Standard storage class but are stored at lower levels of redundancy. This storage class is designed for 99.99% availability and 99.99% durability.

- *Glacier:* Amazon Glacier is an independent, low-cost cloud-based archival solution. This storage class uses Amazon Glacier to store your objects and is suitable for data-archiving tasks. Storage costs are extremely low, but it can take up to 5 hours to read the data.

Costs

Amazon charges you for the following aspects when you use Amazon S3. The specific costs differ between regions.

- *Storage:* You are charged for the objects you store in your Amazon S3 buckets.
- *Requests:* You are charged for the number of requests being made for objects in your Amazon S3 buckets.
- *Storage Management Pricing:* In November 2016, Amazon announced a new feature called Amazon S3 Object Tagging. Amazon S3 allows you to create object-based tags, and these tags can be created, updated, and deleted at any time during the life of the object. These tags can be used to get information on which objects are being accessed more than others. Amazon charges you a small fee per tag. For more information on Amazon S3 Object Tagging, visit `https://aws.amazon.com/about-aws/whats-new/2016/11/revolutionizing-s3-storage-management-with-4-new-features/`.
- *Data Transfer Pricing:* Additional costs are involved if you want to replicate your Amazon S3 buckets across different regions.
- *Transfer Acceleration:* Amazon S3 Transfer Acceleration is a feature that allows you to leverage Amazon CloudFront's CDN endpoints to offer your users access to the content of your Amazon S3 buckets. For instance, if your bucket were located in Tokyo, without Transfer Acceleration your users from around the world would have to make requests to Tokyo. With Transfer Acceleration enabled, they would only have to make requests to the nearest CloudFront CDN endpoint, which in many cases would be located much closer to them than the Amazon S3 bucket.

You can visit the following site to get an idea of the difference in access times with and without Amazon S3 Transfer Acceleration:

`http://s3-accelerate-speedtest.s3-accelerate.amazonaws.com/en/accelerate-speed-comparsion.html`

To get an updated list of charges, visit `https://aws.amazon.com/s3/pricing/`.

Subresources

Every bucket and object in Amazon S3 has a set of subordinate objects associated with it. These subordinate objects are called subresources of the object. Subresources cannot exist on their own; they are always associated with a bucket or an object. When this chapter was written, two subresources were associated with Amazon S3 objects:

- *acl:* This is an access control list that defines the list of people who have access to the resource as well as what they can do with the resource.
- *torrent:* You can use this to retrieve a `.torrent` file associated with the specific resource.

Object Metadata

Two kinds of metadata are associated with each object in Amazon S3: system-defined and user-defined.

System-Defined Metadata

As the name suggests, system-defined metadata is automatically maintained by Amazon S3 and includes information such as object creation date, object size, and more. Users cannot edit all system-defined metadata fields. Table 9.1 lists the system-defined metadata fields associated with an object.

Table 9.1: Amazon S3 System-Defined Metadata

Name	Description	User Editable
Date	Date when the object was created.	No
Content-Length	Size of the object in bytes.	No
Last-Modified	Date when the object was last modified (or created if the object has never been modified).	No
Content-MD5	MD5 hash of the object.	No
x-amz-server-side-encryption	Indicates whether server-side encryption is enabled for the object and which service is providing the encryption.	No
x-amz-version-id	The version number of the object, only applicable to objects that have versioning enabled.	No
x-amz-delete-marker	Only applicable to objects that have versioning enabled; for such objects this field indicates whether the object is a delete marker.	No
x-amz-storage-class	Storage class used for storing the object.	Yes
x-amz-website-redirect-location	If configured, allows you to redirect requests for the object to another object or external URL.	Yes
x-amz-server-side-encryption-aws-kms-key-id	Applicable only if server-side encryption is enabled on the object. Contains the ID of the encryption key that encrypted the object.	Yes
x-amz-server-side-encryption-customer-algorithm	Indicates if server-side encryption is enabled on the object using customer-provided keys.	Yes

User-Defined Metadata

User-defined metadata is any additional key-value metadata provided by the user when the object was created.

Common Tasks

In this section, you learn to use the AWS management console to create Amazon S3 buckets and manage the content in these buckets. Log in to the IAM console using your dedicated IAM user-specific sign-in link and navigate to the Amazon S3 service home page (Figure 9.1).

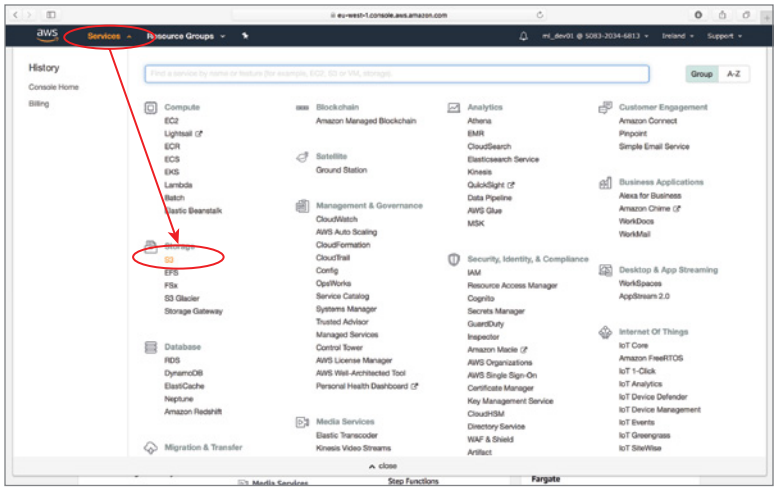


Figure 9.1
Accessing the Amazon S3 management console

Creating a Bucket

To create a new Amazon S3 bucket, follow these steps.

1. Click the Create Bucket button. The Amazon S3 service is available in all regions, so you do not need to select a region in the management console. If you have never created an Amazon S3 bucket you will be presented with the S3 management console welcome page (see Figure 9.2).

   If you have existing buckets in your Amazon S3 account, you will be presented with a page that lists them (Figure 9.3).

2. A bucket, on the other hand, is region-specific, and you will need to provide a unique name for your bucket as well as select the region in which you want to create it (Figure 9.4). Click Next to proceed to the next screen once you have specified the bucket name and region.

   In this section, the name of the bucket being created is `com.asmtechnology.samplebucket` and is located in the EU (Ireland) region. The name you choose for your bucket must be globally unique, and prefixing a reverse domain name is a common practice to ensure unique naming.

FIGURE 9.2
Amazon S3 management console welcome page

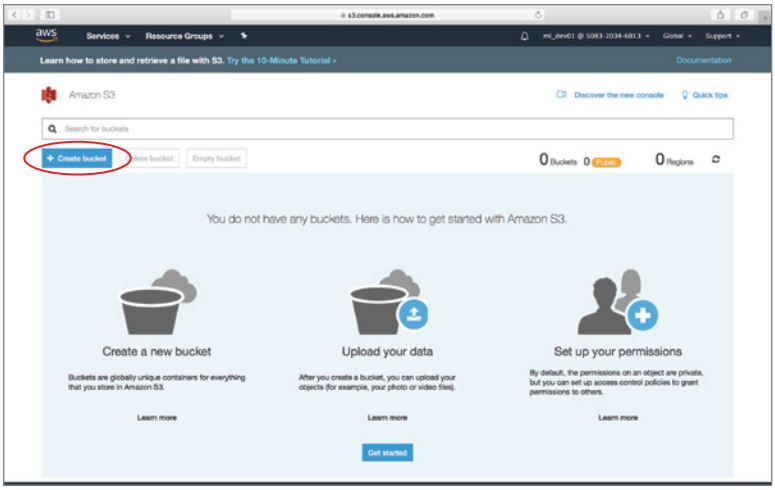
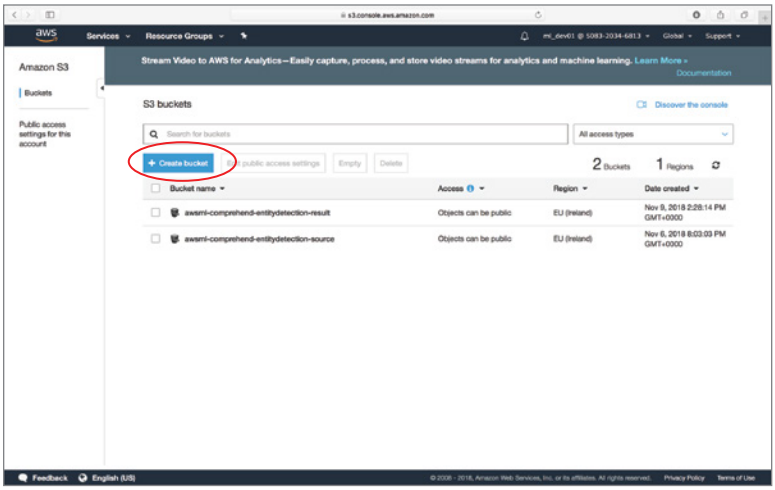


FIGURE 9.3
List of Amazon S3 buckets



3. You are presented with a screen that will let you configure bucket versioning, logging, and cost allocation tags (Figure 9.5). You do not need to set up these options at this stage; click Next.

4. You are presented with a screen that will let you configure permissions for the new bucket (Figure 9.6). By default, a new bucket is not accessible publicly, and can only be accessed by the user who created it via the AWS CLI or the AWS management console.

   Access to Amazon S3 resources are controlled using resource-based IAM policies. A resource-based IAM policy is a JSON document that describes which IAM users have access to a resource, and what they can do with the resource. Amazon S3 buckets and objects within the buckets have independent resource-based policies, and objects do not inherit permissions from a bucket.

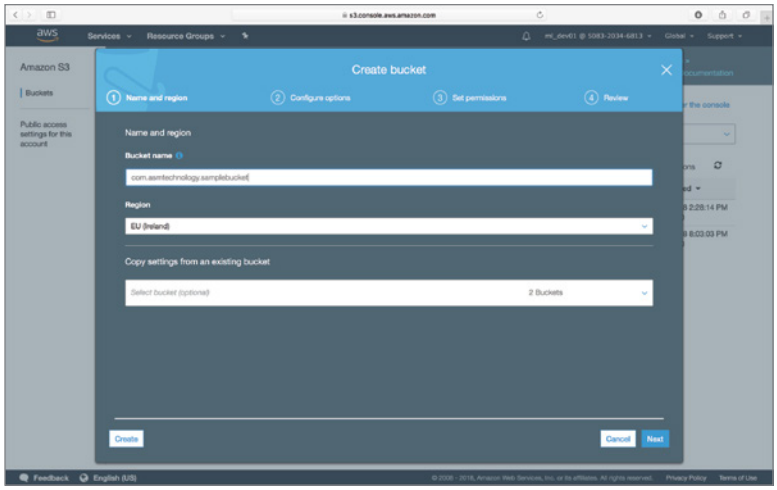


FIGURE 9.4
Specifying the bucket name and region

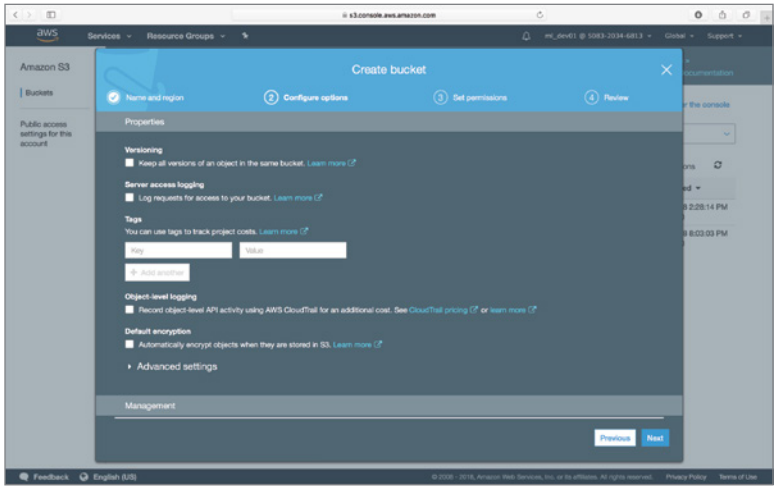


FIGURE 9.5
Configuring versioning, logging, and cost allocation tags

Each bucket also has an XML document associated with it, called an access control list (ACL). The ACL is used to control access to the bucket from other AWS accounts, and the general public.

It is highly recommended to leave the default options unchanged on this screen, and change them (if needed) at a later point in time. Click Next to proceed.

5. You are presented with a screen that summarizes the options and settings for the bucket that will be created (Figure 9.7). Click the Create Bucket button to create the bucket.

FIGURE 9.6
Configuring bucket permissions

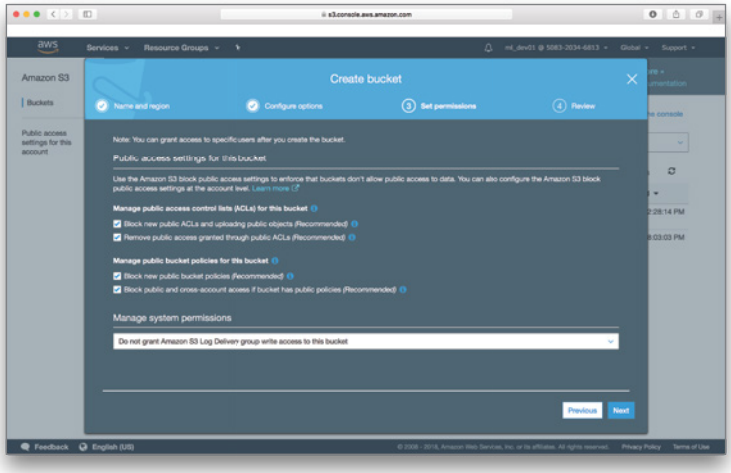
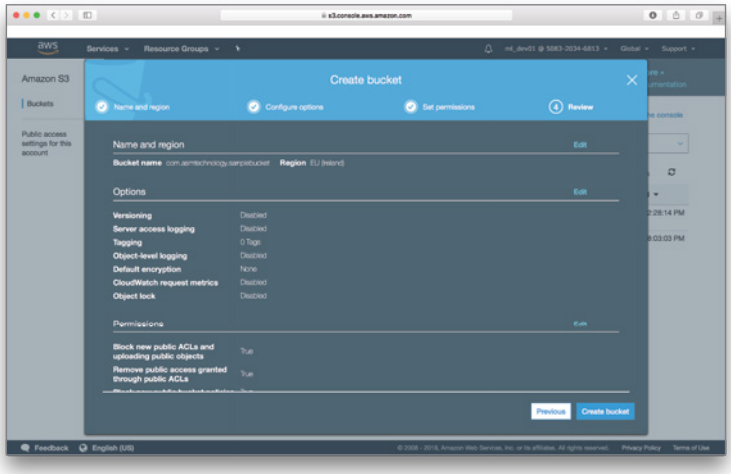


FIGURE 9.7
Bucket summary page



6. The bucket will be created, and you will be taken to a page that lists all your Amazon S3 buckets. When you click the icon beside the name of a bucket from the list, a pop-up window will appear with shortcuts to options that allow you to configure bucket-specific settings (Figure 9.8).

If you have one or more buckets, this screen becomes the home screen presented to you whenever you visit the Amazon S3 console.

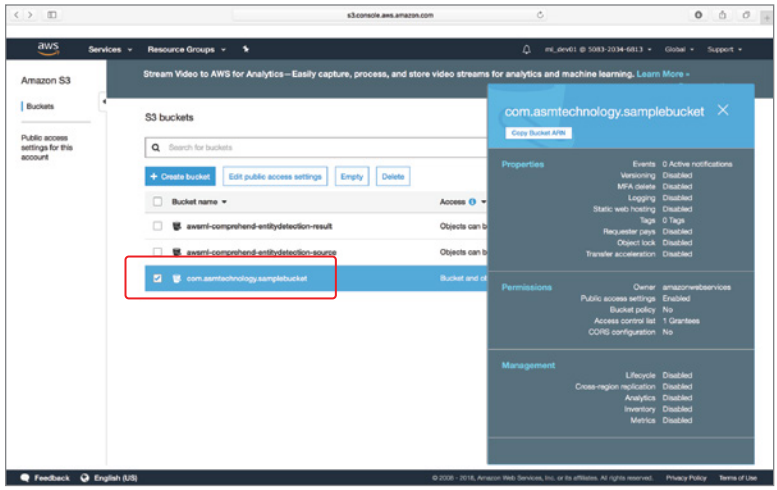


FIGURE 9.8
List of Amazon S3 buckets in your account

Uploading an Object

Complete these steps to upload an object to an existing bucket.

1. Click the name of the bucket in the list of buckets to access its contents (Figure 9.9).

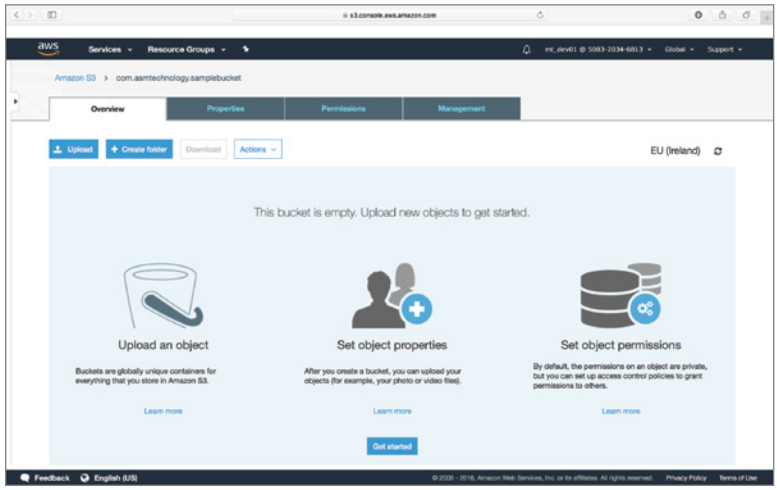


FIGURE 9.9
Contents of an Amazon S3 bucket

2. Click the Upload button to bring up the File Upload dialog box (Figure 9.10). Use the options in the File Upload dialog box to select one or more files from your computer; then click the Next button.

3. You are presented with a screen that will let you configure permissions for the new object (Figure 9.11). By default, a new object can only be accessed by the user who created it via the AWS CLI or the AWS management console. If you want the object to be accessible to

other AWS accounts, you can add the accounts to the ACL for the bucket. If you would like the object to be accessible to users on the Internet via a URL, ensure you change the value in the Manage Public Permissions drop-down menu from Do Not Grant Public Read Access To This Object(s) to Grant Public Read Access To This Object in the Manage Permissions drop-down menu.

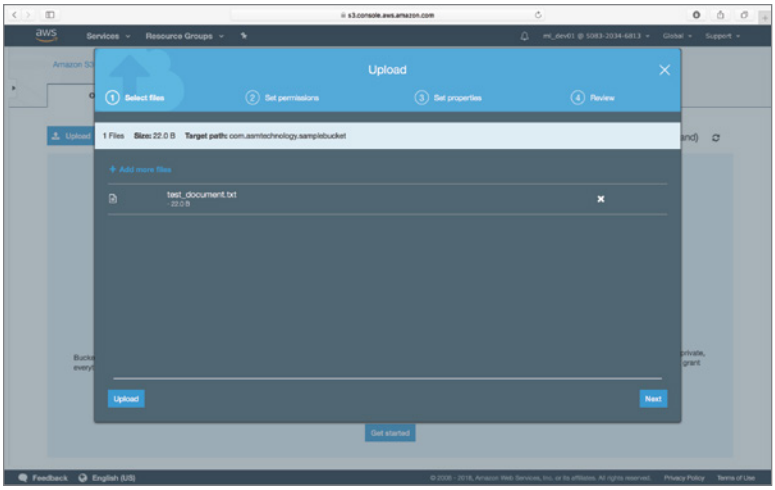


FIGURE 9.10
Selecting files in the File Upload dialog box

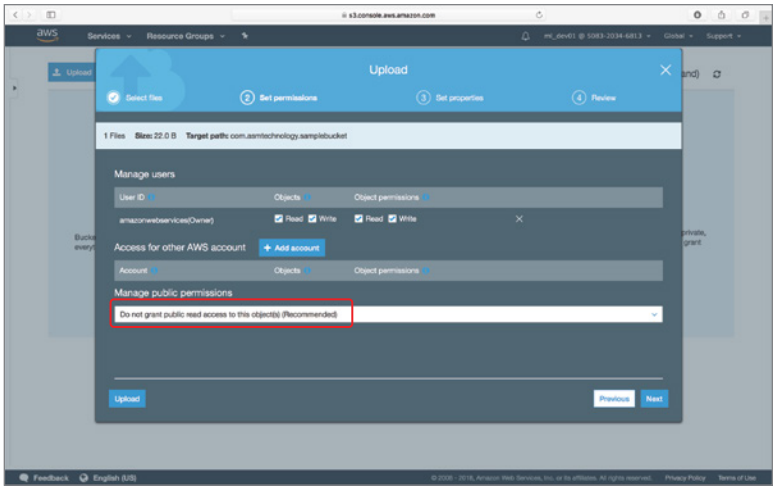


FIGURE 9.11
Configuring object permissions

4. You are presented with a screen that will let you select a storage class and an encryption setting, and specify any user-defined metadata for the new file (Figure 9.12). By default, a new file uses the Standard storage class and no encryption. User-defined metadata is a set of key-value pairs that can only be specified at the point when an object is created. Accept the default options and click Next.

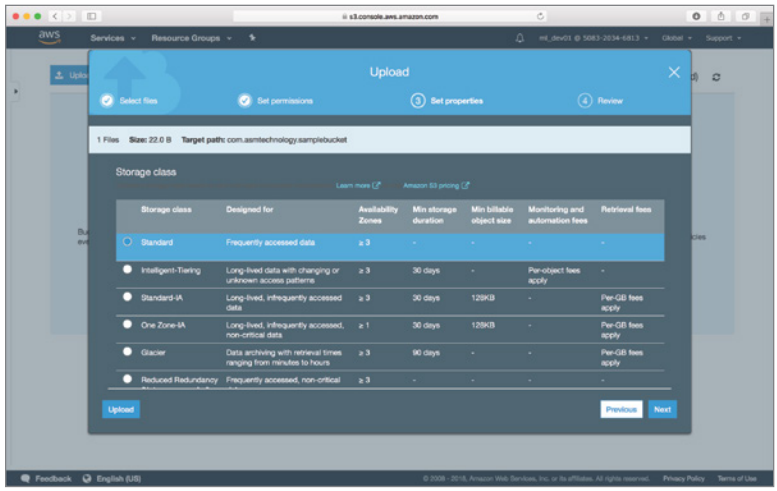


FIGURE 9.12
Configuring file storage class and encryption

5. You are presented with a screen that summarizes the options and settings for the file that will be uploaded (Figure 9.13). Click the Upload button to upload the file to the bucket.

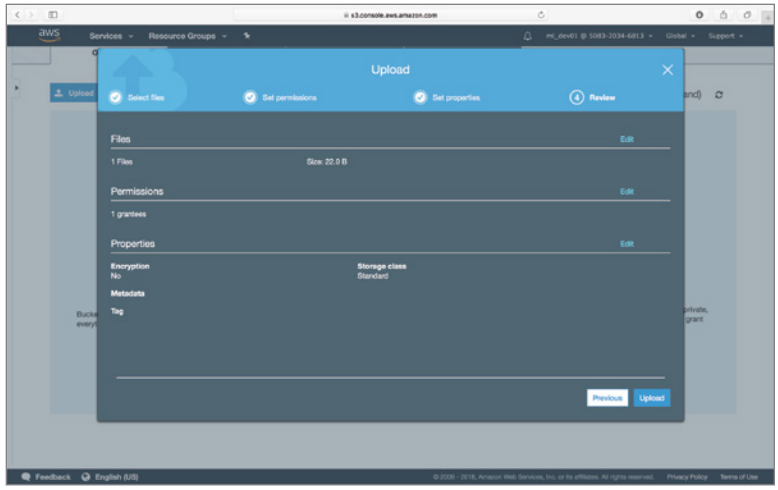


FIGURE 9.13
File summary page

Once the file has finished uploading, it appears in your bucket (Figure 9.14).

Accessing an Object

To download an object from your Amazon S3 bucket onto your computer, follow these steps:

1. Navigate to the bucket using the Amazon S3 management console, select the icon beside the name of the bucket, and click the Download button in the pop-up dialog that appears on the screen (Figure 9.15).

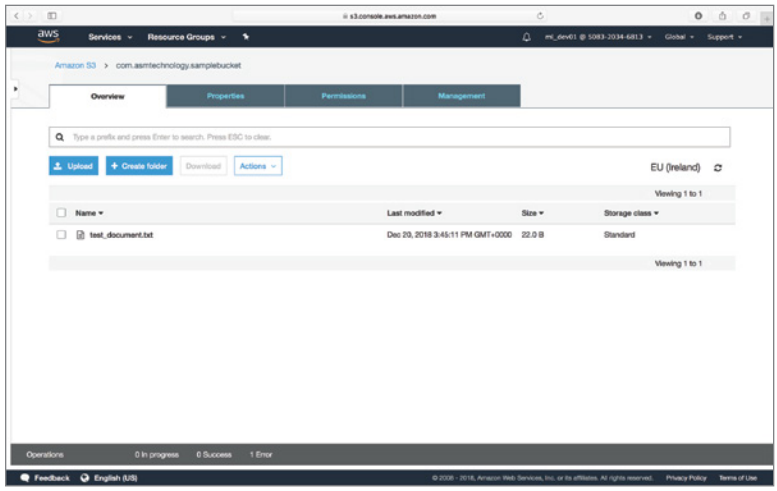


FIGURE 9.14
Amazon S3 bucket showing a file

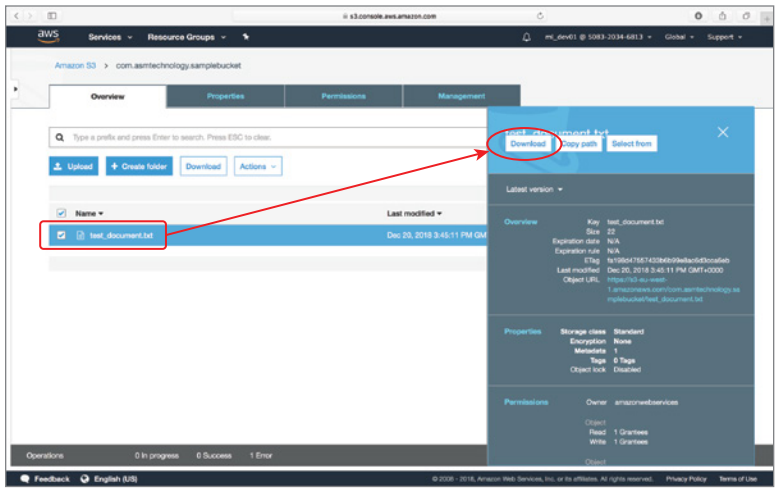


FIGURE 9.15
Downloading a file from a bucket

If you do not want to use the management console, you can also access any object in Amazon S3 using a URL.

2. To find the URL for an object in an Amazon S3 bucket, navigate to the bucket in the management console and select the object within the bucket. Copy the value of the Object URL field in the pop-up dialog (Figure 9.16).

   The value within the Object URL field is a URL that follows this naming convention:

   ```
   https://s3.<region name>.amazonaws.com/<bucket name> /<file name>
   ```

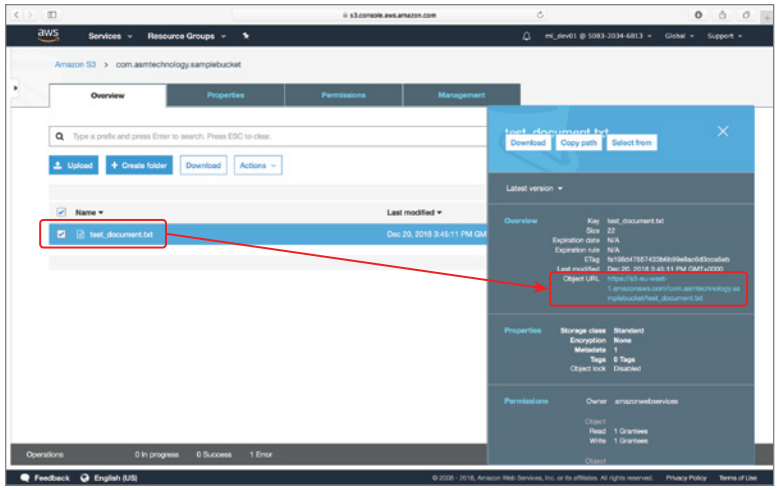


FIGURE 9.16
Locating the Amazon S3 Object URL

For example, a file called `sunset.jpg`, in a bucket called `com.asmtechnology.awsbook.testbucket1`, in the eu-west-2 region can be accessed using the following URL:

```
https://s3.eu-west-2.amazonaws.com/com.asmtechnology.awsbook.testbucket1/sunset.jpg
```

If both the bucket and the file you are accessing are not publicly accessible, you will receive an access denied error when you try the URL in a web browser (Figure 9.17).

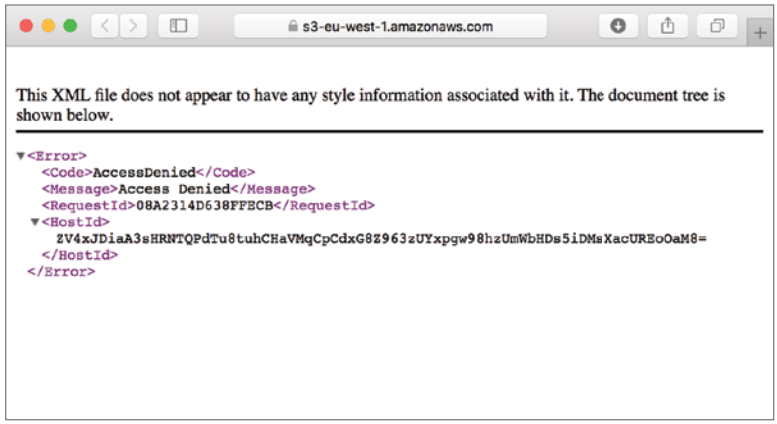


FIGURE 9.17
Non-public buckets and files are not accessible using a URL.

3. Before you can make the object publicly accessible, you will need to change the public access settings for the bucket. Select the bucket and switch to the Permissions tab (Figure 9.18).

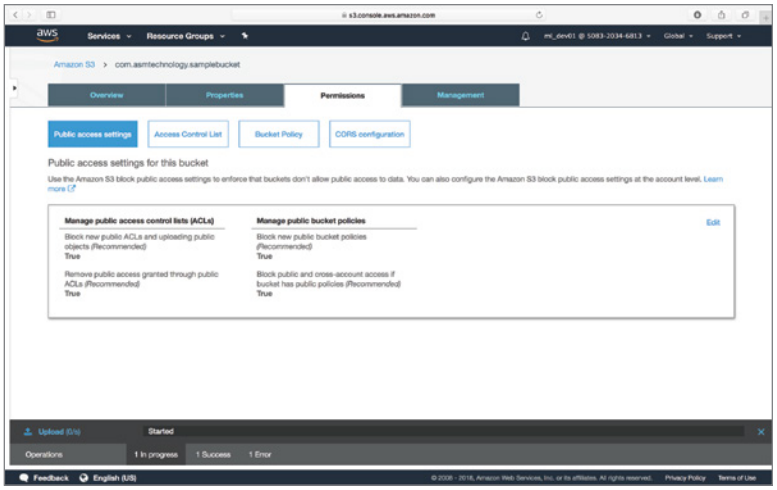


FIGURE 9.18
Accessing Amazon S3 bucket permissions

4. Click the Edit button and uncheck all four options (Figure 9.19).

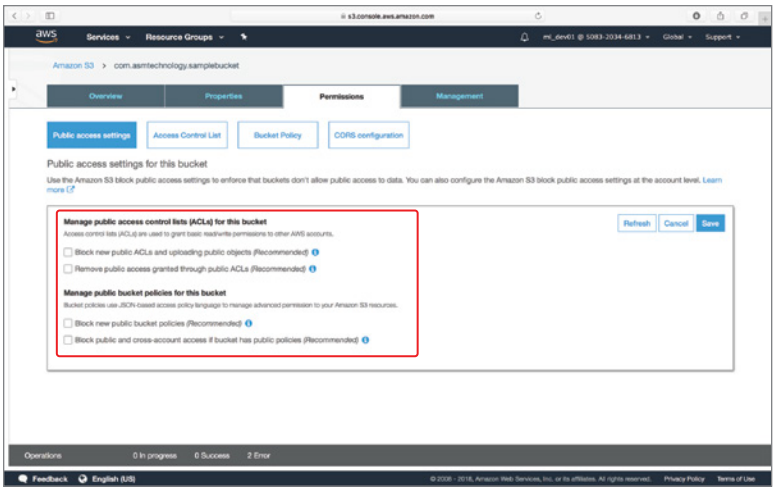


FIGURE 9.19
Configuring Amazon S3 bucket permissions

5. Click the Save button.
6. Switch to the Overview tab and select the object in the Amazon S3 bucket. Click the Make Public menu item under the Actions drop-down menu (Figure 9.20).
7. Click the Make Public button in the pop-up dialog that appears on the screen (Figure 9.21).

If you retry the URL in a web browser, you will be able to access the file. You can either set permissions at an individual object level, or you can set up permissions for the entire bucket.

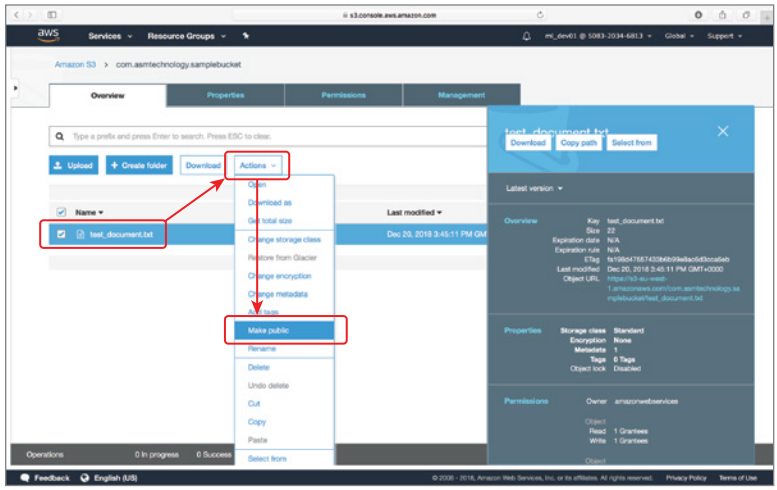


FIGURE 9.20
Accessing the Make Public option

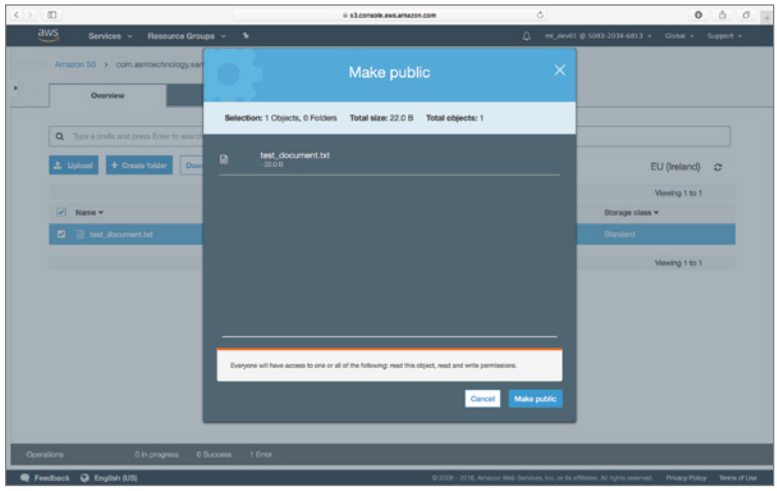


FIGURE 9.21
Making a file publicly accessible

Changing the Storage Class of an Object

The default storage class of objects on Amazon S3 is Standard. To change the storage class of an object:

1. Navigate to the bucket using the Amazon S3 management console and select the object from the contents of the bucket.
2. Select the Change Storage Class menu item under the Actions drop-down menu (Figure 9.22).
3. Select an option for the storage class and click the Save button.

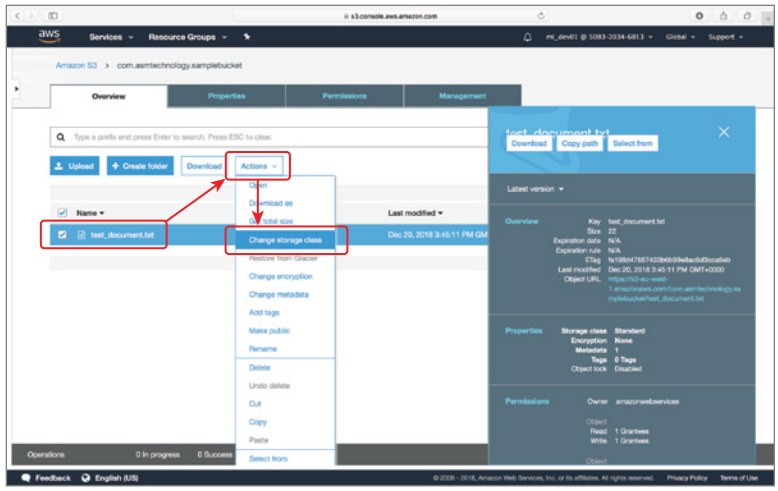


FIGURE 9.22
Changing the storage class of an object

Deleting an Object

To delete an object from an Amazon S3 bucket:

1. Navigate to the bucket using the Amazon S3 management console and select the object from the contents of the bucket.
2. Select the Delete menu item under the Actions drop-down menu (Figure 9.23).

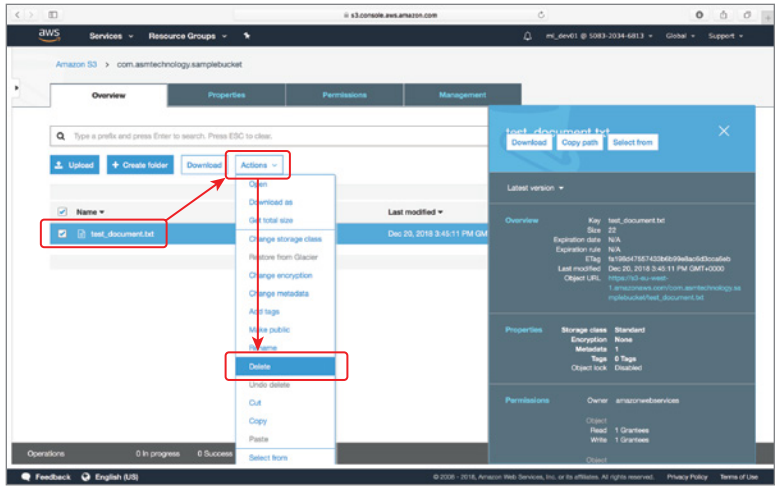


FIGURE 9.23
Deleting an object from an Amazon S3 bucket

Once you delete an object, it is permanently removed from Amazon S3. The only exception to this rule occurs when versioning has been enabled on a bucket, in which case an object that has been deleted from a bucket can be restored.

Amazon S3 Bucket Versioning

Versioning is a bucket-level concept that, when enabled, stores all versions of an object. You can download an older version of an object, and you can even recover an object after it has been deleted. Once versioning is enabled on a bucket, you cannot remove it. You can, however, temporarily suspend versioning.

To enable versioning on a bucket:

1. Navigate to the bucket in the management console and click the Properties tab.
2. Expand the Versioning section, select the Enable Versioning option, and click the Save button (Figure 9.24).

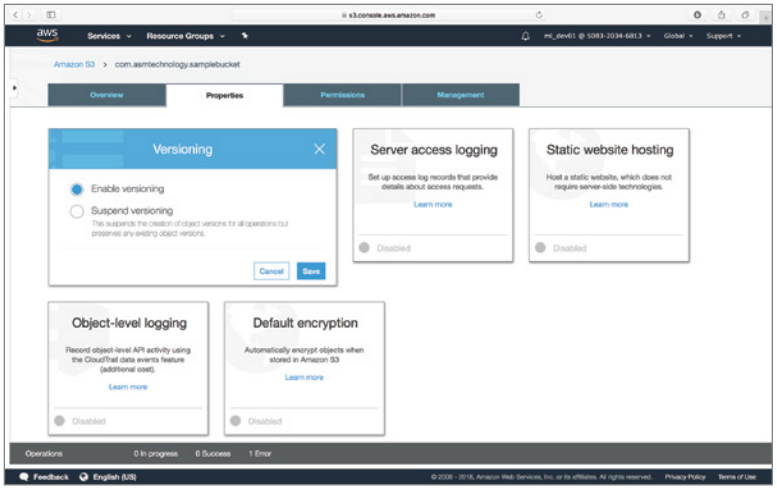


FIGURE 9.24
Enabling bucket versioning

To understand how versioning works:

1. Create a new text document on your computer called `welcome_letter.txt` and in that document type the following line:

   ```
   Welcome to the world of Amazon Web Services.
   ```

2. Save the document on your computer and upload it to a bucket that has versioning enabled. To make the document accessible to the public you can select Grant Public Read Access To This Object(s) in the Manage Public Permissions drop-down (Figure 9.25). This will ensure you can access the document from a web browser.
3. Obtain the URL for the document and open the document in a web browser. Your web browser should render the contents of the text document.
4. Open the `welcome_letter.txt` file that you had previously saved on your computer, and edit its contents to resemble the following:

   ```
   Welcome to the world of Amazon Web Services.
   Amazon S3 versioning allows you to access older versions of documents.
   ```

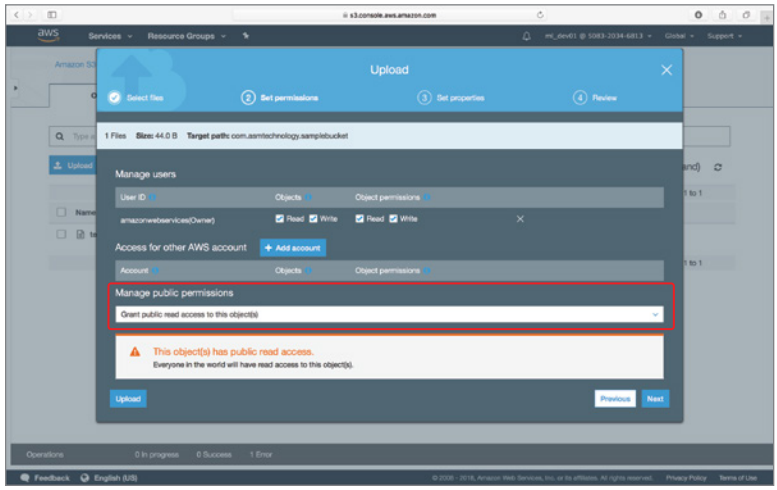


FIGURE 9.25
Making an object publicly accessible while uploading it

5. Save the file and upload it again to the same bucket.
6. Once the document has finished uploading, click the row titled `welcome_letter.txt` to reveal a pop-up dialog with options. Expand the versions drop-down menu in the pop-up dialog to reveal links to the different versions of the document (Figure 9.26).

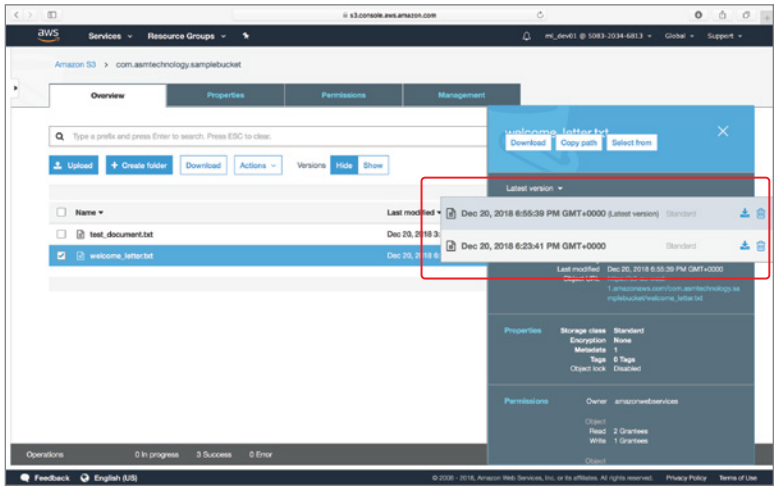


FIGURE 9.26
Accessing document versions

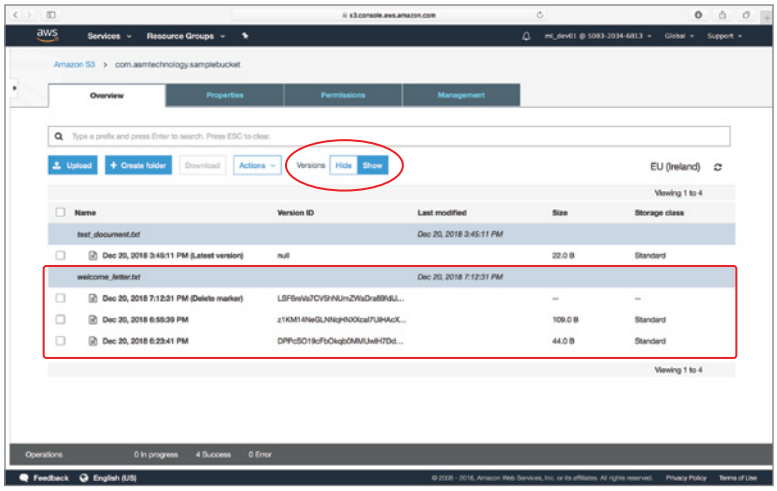
The newest version of the document is always listed at the top. It is important to remember that you are charged for the combined space occupied by all versions of a document.

7. If you want to delete a version of the document that you do not need, click the trash can icon beside a document version in the versions drop-down menu. When you delete a version of a document (and not the entire document itself), the version you are deleting is permanently lost.

8. If instead you want to delete the document, select the document, and use the Delete menu item under the Actions menu.

When versioning is enabled on a bucket, you will see an additional selector that allows you to view all versions of the objects in your bucket (Figure 9.27).

FIGURE 9.27
Version selector switch



When the selector switch is set up to show versioned objects, you can see not only object versions, but also delete markers, which are special entries used to indicate that an object has been deleted. Restoring a deleted object is simply a matter of deleting the delete marker.

Accessing Amazon S3 Using the AWS CLI

You can use the AWS CLI to interact with Amazon S3 over the command line. Setup and configuration of the CLI client for Mac OS X and Windows is covered in Appendix C.

The general syntax of the `aws` command follows:

```
$ aws <service identifier > <service instructions>
```

The service identifier is a string that identifies an AWS service you want to interact with. The service identifier for Amazon S3 is `s3` (in lowercase). Each service supports a different list of instructions. For a complete list of s3 instructions that are available within the CLI, visit `http://docs.aws.amazon.com/cli/latest/userguide/using-s3-commands.html`.

As an example, the `ls` instruction retrieves a list of all buckets in the user account that have been configured into the CLI. If you type the following instruction at the command prompt:

```
$ aws s3 ls
```

you receive a list of buckets:

```
Abhisheks-MacBook:~ abhishekmishra$ aws s3 ls
2017-01-15 16:52:59 com.asmtechnology.awsbook.testbucket1
Abhisheks-MacBook:~ abhishekmishra$
```

In addition to the high-level operations that can be performed using the `s3` service identifier, Amazon also provides access to lower-level operations using the `s3api` service identifier. For more information on lower-level operations that can be performed on Amazon S3 buckets using the `s3api` service identifier, visit `http://docs.aws.amazon.com/cli/latest/userguide/using-s3api-commands.html`.

Summary

- Amazon S3 is a key-value, object-based storage service.
- Amazon S3 organizes objects into buckets; bucket names must be globally unique.
- Objects can be uploaded to Amazon S3 buckets using the AWS management console or the AWS CLI.
- You can control access to both buckets and the objects in buckets.
- Each object in Amazon S3 has a storage class associated with it. The storage class determines how Amazon S3 stores the data for the object and if you will be charged additional costs to read the data.
- Amazon S3 versioning allows you to save multiple versions of an object. You are charged for the combined space occupied by all versions of a document.