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

Hello, and welcome to the exciting world of machine learning. 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. Fortunately, there is very little truth in that idea of machine learning. For starters, it is not a recent development; computer scientists have been researching for decades ways to make computers more intelligent by attempting to find ways to teach computers to make generalizations and predictions much like humans do. However, intelligence is not just about recognizing things, and even the best machine learning systems today are not capable of reasoning like human beings. The machine learning systems that exist today are essentially pattern recognizers and can, for instance, examine a picture and detect a cup of tea close to the edge of a table. They cannot, however, reason that the cup could accidentally fall off the table, as it is too close to the edge.

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. Machine learning algorithms require a lot of processing and storage space and until recently were only possible to deploy in large companies or in academic institutions. Recent advances in storage, processor, and GPU technology have provided the processing power required to build and deploy machine learning systems at scale and get results in real time.

In the past, lack of quality data was also a factor that prevented widespread adoption of machine learning. With the advent of social media and analytics applications, developers have access to lot more data about their customers than they did in the past.

Another factor that has contributed to the recent increase in machine learning applications is the availability of excellent tools and frameworks such as Core ML, Create ML, Pandas, Matplotlib, TensorFlow, Scikit-learn, PyTorch, and Jupyter Notebooks, 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. 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 that deals with creating algorithms that learn from data. Machine learning traces its roots to a computer program created in 1959 by a computer scientist Arthur Samuel while 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

As an iOS developer, Core ML is likely to be your framework of choice when it comes to deploying a machine learning model in your app. Training a machine learning model, on the other hand, involves several steps and, except for the simplest of cases, is performed offline and using a different set of tools. Apple provides a number of tools to train Core ML models and convert pre-trained models built using other frameworks such as Scikit-learn and Keras to the Core ML format. The reason you may want to use a non-Apple framework like Scikit-learn or Keras to train a model is because the library may provide an implementation of a model that is not possible to train using Apple's toolset or may simply be updated more frequently.

Depending on the type of model you are trying to build, there may be a lot of steps involved even before you get to the point when you can start training—such as data preprocessing, feature engineering, and data visualization. Data scientists frequently use a set of tools to assist them with these steps. In this section, you will learn about some of the tools commonly used by data scientists to build machine learning solutions.

Although R has historically been the language of choice for statisticians, most data scientists and machine learning engineers today work in Python. The popularity of Python in the machine learning space is due to the abundance of machine learning–specific libraries that assist with all steps of the machine learning process from preparing data, feature engineering, information visualization, to training models with the latest algorithms. Where Python code is presented, this book will use Python 3.6.5. The following are the most popular Python machine learning tools:

- **Jupyter Notebook:** This is a 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 Notebooks at `http://jupyter.org`. Appendix A contains instructions on installing Anaconda Navigator and setting up Jupyter Notebooks 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 libraries and quickly switch between different sets of libraries and Python versions. It also includes popular tools such as Jupyter Notebooks and Spyder Python IDE. 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`.
- **Google TensorFlow:** This is a Python library for numerical computation. It was developed by Google and eventually released as an open source project in 2015. Google 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 out more information on TensorFlow at `https://www.tensorflow.org`.
- **Keras:** This is another popular Python library for training and using deep learning networks. It was designed to facilitate rapid experimentation with neural networks and acts like a higher-level abstraction to popular deep learning frameworks like Google TensorFlow. As of TensorFlow 2.0, Keras is included with TensorFlow.

Common Terminology

In this section, we 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; these parameters are determined by a process known as *training*.
- **Data acquisition:** This is 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 is 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:** This is 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:** This is a set of data that contains all the input features (including engineering 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:** This is 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:** This is the task of using an algorithm to assign observations to a label from a fixed set of predefined labels.
- **Linear regression:** This is a statistical technique that attempts to fit a straight line to a set of data points. Linear regression is commonly used to create machine learning models that can be used to predict continuous values (such as height, width, age, etc.).
- **Logistic regression:** This is 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. An example is 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:** This is 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:** This is a mathematical function that takes as input the predicted and actual values and returns a numerical measure that captures the error in prediction. The goal of the training function is to minimize the error function.
- **Neural networks:** This is 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:** This branch of machine learning utilizes multilayer neural networks with a large number of neurons 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 a customer when they apply to purchase an insurance product.
- **Behavioral biometrics:** Machine learning systems can be trained to build a profile of a user 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 from where they access the platform, at what times of the day, where they click a page, how long they stay on a page, how quickly they move their mouse across a 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 recommendation:** Machine learning systems are commonly used by online retailers (such as Amazon) to provide a list of recommendations to a customer based on their purchase history. These systems can even predict when a customer is likely to run out of groceries and send them reminders to order items.
- **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.
- **Text prediction:** Smart keyboards like the one on your iPhone use machine learning models to predict the next word you are likely to type, and the top three predictions are surfaced as suggestions on top of the iPhone keyboard as you type.
- **Conversational interfaces:** Machine learning models are used with chatbots to map natural language inputs from users into an input that the chatbot can understand. For example, a chatbot that allows users to access banking functions will use machine learning to interpret slightly differing natural language instructions such as "I want to pay somebody," "I want to make a payment to my mother," "Make a payment," and "Transfer some money" to all map to the same internal banking process, perhaps called "Make payments."
- **Text analysis:** Machine learning models can be used to analyze text documents to extract information on entities referenced in the document and the overall sentiment of the document. The document in this context does not have to be a lengthy Microsoft Word document; it could be a short Twitter message. Such a system could be used to monitor a Twitter feed and inform you about trending topics or the overall sentiment of your followers' responses to a post you made.
- **Scene analysis and recognition:** Machine learning models (particularly deep learning models) are commonly used for scene analysis and object recognition in digital images and video feeds. Scene analysis involves feeding in an image (or a video) to the machine learning model and the model providing a description of the contents of the image—such as *man riding a green bicycle on a sunny day*. Object recognition involves feeding an image to the machine learning model and getting a list of objects detected in the image along with bounding boxes that represent the location of the objects in the image.

Types of Machine Learning Systems

There are several different ways in which you can go about classifying machine learning systems; one of the most common approaches is to base the classification on the manner in which the system is trained and the manner in which the system can make predictions. Based on this approach, machine learning systems can be classified as follows:

- Supervised learning
- Unsupervised learning

- Semisupervised 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 (see Figure 1.1).

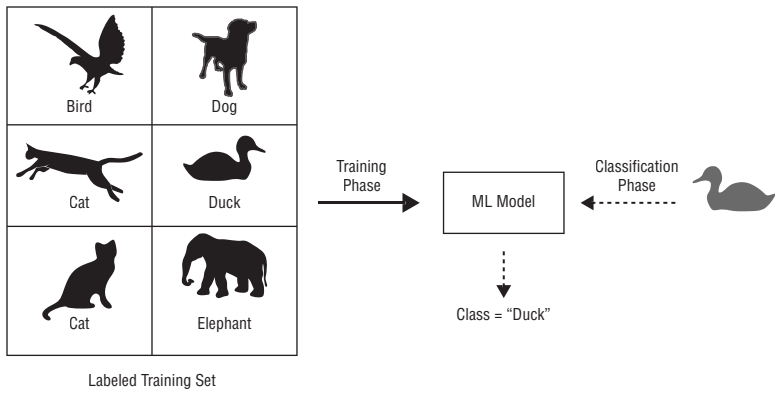


Figure 1.1
Supervised learning

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 to 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 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 the following:

- Finding objects in digital images
- Spam filtering

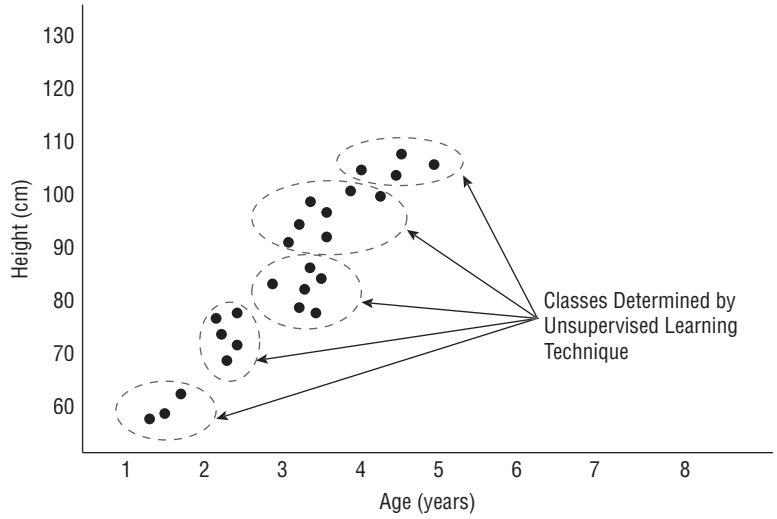
- Predicting the possibility of developing a medical condition based on lifestyle factors
- Predicting the likelihood of a financial transaction being fraudulent
- Predicting the price of property
- Recommending a product to a customer based on historical purchasing data
- A music streaming servicing 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 six 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

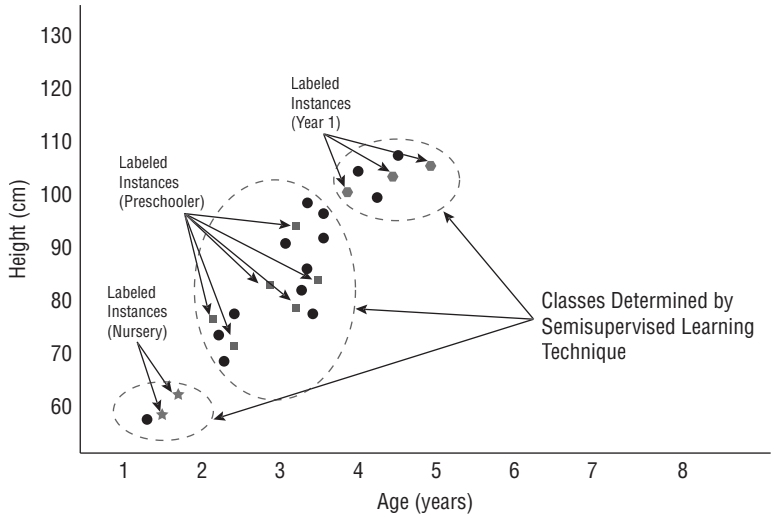


- **Dimensionality reduction:** These algorithms are used to combine a large number of input features into a smaller number of features without losing too much information, Typically the features that are combined have a high degree of correlation with each other. Dimensionality reduction reduces the 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.

Semisupervised Learning

Semisupervised 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 semisupervised learning to the same data on the heights and ages of children under six years of age, with some of the samples labeled to indicate the level of education.

Figure 1.3
Semisupervised 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 semisupervised learning is that it is less prone to labeling bias than supervised learning. This is because a data scientist is labeling only 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.

Semisupervised 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. With each action is an associated reward or penalty. The goal of the system is to maximize 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 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 a 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, also known as *offline 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 new additional data for use in training a batch learning system, you need to create a new model on all the previous data plus the new data. You must then 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 retail 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, also known as *memory-based learning*, attempts to make predictions 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 into 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.

Common Machine Learning Algorithms

Machine learning systems are also sometimes classified on the basis of the type of algorithm that is used to implement the model, and by now it should come as no surprise that there are several different types of machine learning algorithms, and this is an area of active research. You may have also encountered the term *classical machine learning models* to encompass all the algorithms that are not deep learning based. This may lead you to think that deep learning models are the only kind that matter and are the only kind that should be used to implement new solutions.

This is not true; non-deep-learning models have several use cases and are often the first choice for many tasks where the need to explain the result is just as important as its accuracy. In this section, you will learn about some of the different types of machine learning algorithms. This list is not exhaustive.

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: X1, X2, and X3, and a known target value Y. Linear regression will assume that the relationship between target variable Y and input features X1, X2, and X3 is linear and can be expressed by the following equation:

$$Y_i = \alpha X1_i + \beta X2_i + \gamma X3_i + \varepsilon.$$

where:

- Y_i is the predicted value of the i^{th} target variable.
- $X1_i$ is the i^{th} value of feature X1.
- $X2_i$ is the i^{th} value of feature X2.
- $X3_i$ is the i^{th} value of feature X3.
- α, β, γ are the coefficients (or the slopes) of the features X1, X2, X3.
- 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 follows:

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

where Y' is the predicted value and Y is the reference or actual value.

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.

Linear regression with a large number of variables can be difficult to visualize. Figure 1.4 is a scatter plot of two variables: the feature variable X and the target variable Y.

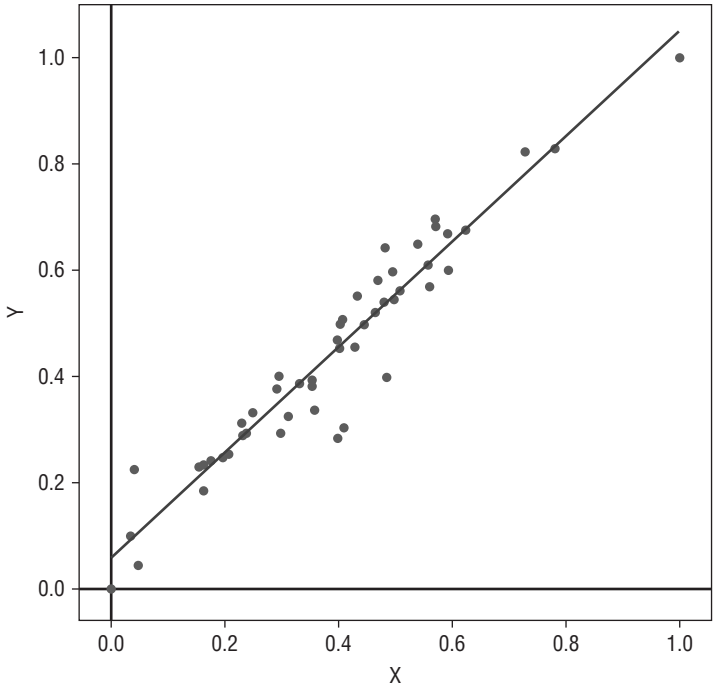


Figure 1.4
A simple linear regression model

The values of both variables have been normalized to ensure they lie between 0.0 and 1.0. The regression line is also depicted in the plot, and as you can see, although it does not fit every single point perfectly, it appears to be the best-fitting overall line. The equation of this line can be represented as follows:

$$Y_i = \alpha X1_i + \varepsilon$$

The coefficient α and the intercept term ε of the regression line are values that are determined by the training process, and once you know those values, you can fit them into the equation to calculate the predicted Y value for any X value.

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 Vladmir 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, with each point belonging to one of two classes. Let's also assume that the data is linearly separable—that is, one can draw a line that can separate them.

Figure 1.5 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. Figure 1.5 also presents three possible linear decision boundaries, each capable of separating the observations into two different sets.

The red decision boundary 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 if they are only slightly different from the training set. The green decision boundary has a similar problem in that it is too close to the second set of observations. The decision boundary in blue is optimal because it is as far away as possible from both classes and at the same time clearly separates both classes. SVM models aim to find this blue line 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 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 1.5 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 1.5. 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 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.

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 1.6 shows a scatter plot of another fictional dataset with two features per data point, with 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.

If, however, you were to add an extra dimension (z-axis) to the data, and compute `z` values for each point using the equation $z = x^2 + y^2$, then the data becomes linearly separable along the z-axis using a plane at `z = 0.3`. This is depicted in Figure 1.7.

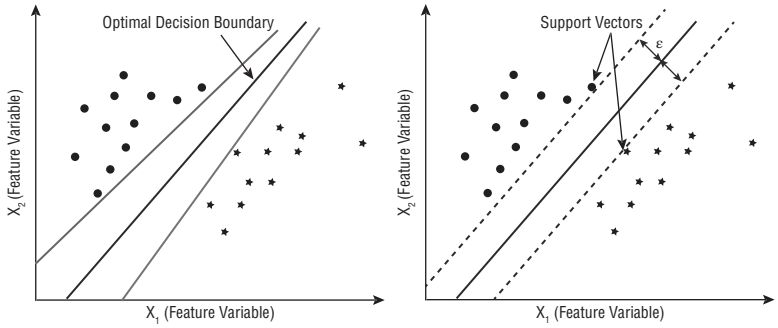


Figure 1.5
Three potential decision boundaries

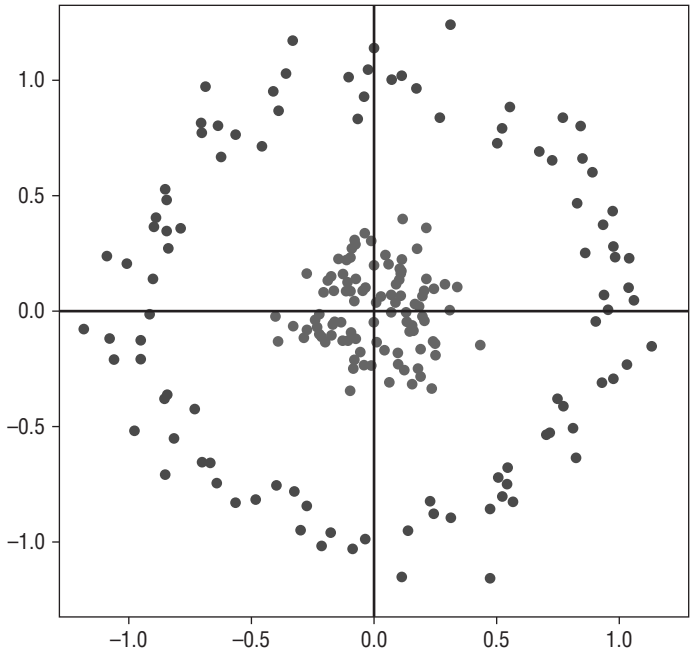


Figure 1.6
Data that cannot be classified using a linear decision boundary in two-dimensional space

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 1.8.

This is an over-simplification 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 at `https://www.cambridge.org/core/books/an-introduction-to-support-vector-machines-and-other-kernelbased-learning-methods/A6A6F4084056A4B23F88648DDBFDD6FC`.

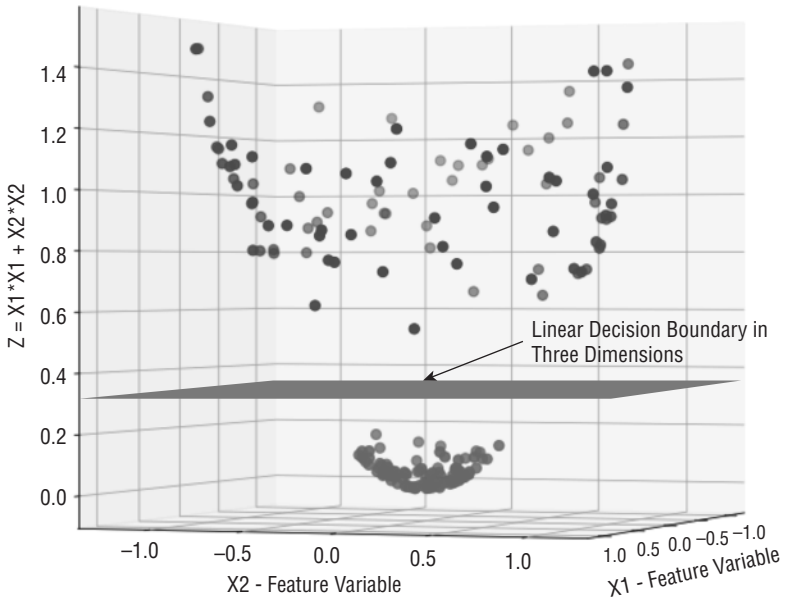


FIGURE 1.7
Data that cannot be classified using a linear decision boundary in two-dimensional space can be classified in three-dimensional space.

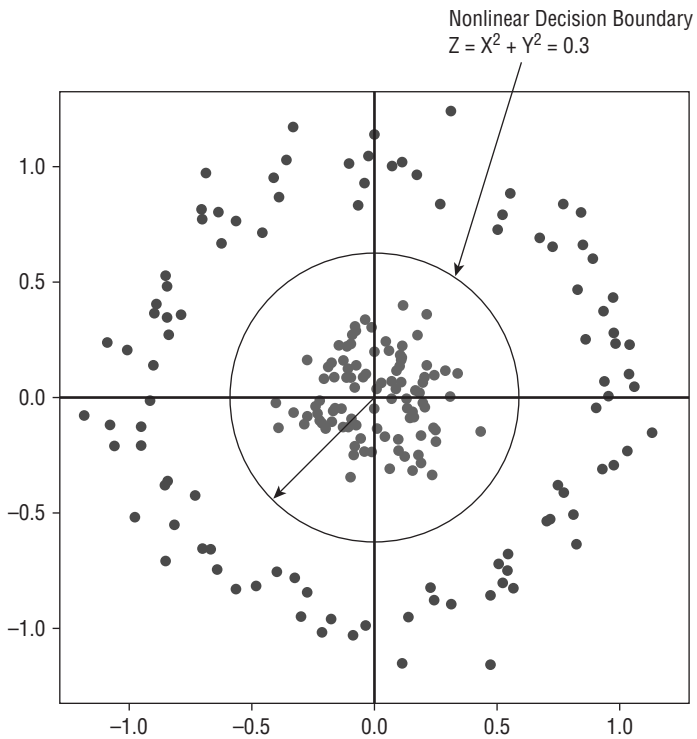


FIGURE 1.8
Nonlinear decision boundary in two-dimensional space

Logistic Regression

Logistic regression, despite having the word *regression* in its name, is a technique that can be used to build binary and multiclass 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.

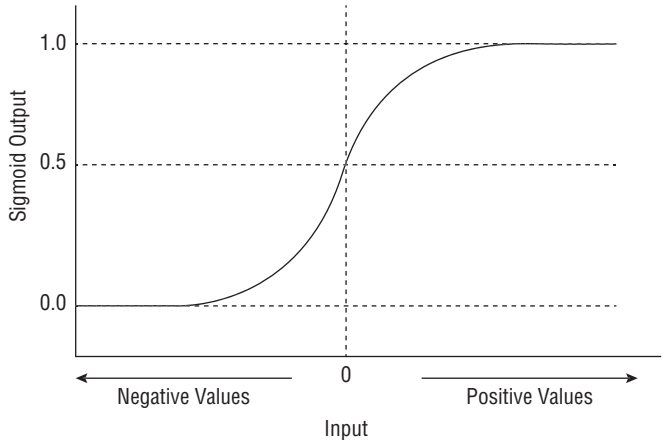
To use a continuous value for binary classification, logistic regression converts it into a probability value between 0.0 and 1.0 by feeding the output of linear regression into a logistic function. In statistics, a logistic function is a type of function that converts values from [-infinity, + infinity]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

In the case of logistic regression, the logistic function is the sigmoid function, which is defined as follows:

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

The graph of the sigmoid function is presented in Figure 1.9. The output of the sigmoid function will never go below 0.0 or above 1.0, regardless of the value of the input.

Figure 1.9
The sigmoid function



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 (see Figure 1.10).

Logistic regression is inherently a binary classifier, but it can be used as a multiclass classifier for datasets where the target variable can belong to more than two classes. There are two fundamental approaches that can be used with a binary classifier for multiclass 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 multiclass 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 output 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

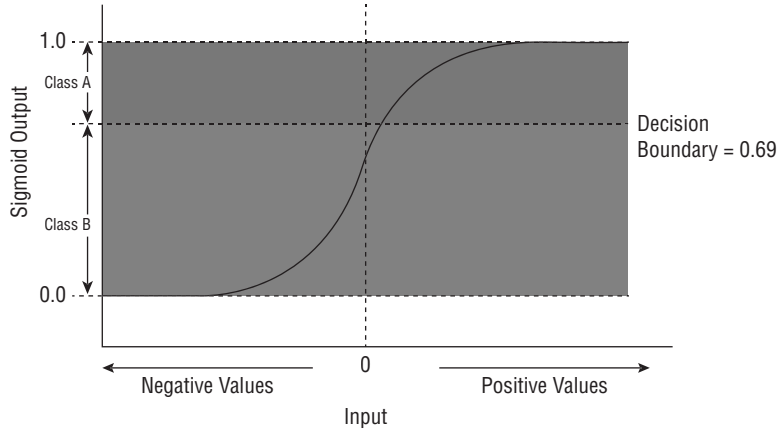


FIGURE 1.10
Using the sigmoid function for binary classification

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

While training an ensemble of binary models is one way to build models capable of multiclass classification, some algorithms like logistic regression can be modified to inherently support multiclass 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 1.11.

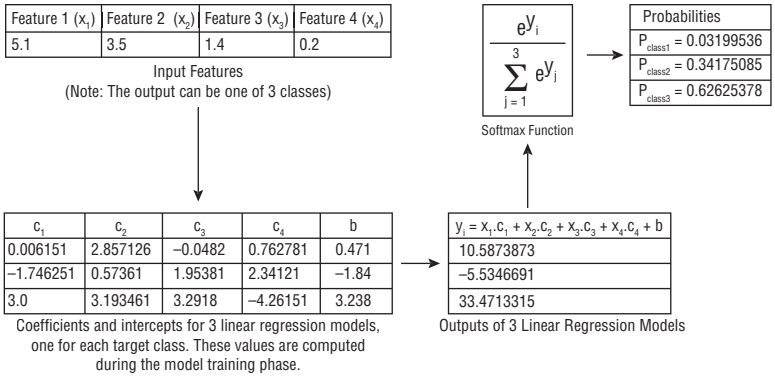


FIGURE 1.11 Softmax logistic regression

To understand how this function works, consider a dataset where each row contains four continuous numeric attributes and a multiclass 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.

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 is met.

Decision trees are versatile and can be used for both classification and regression tasks. When used for classification tasks, they are inherently capable of handling multiclass 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, which 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*.

Figure 1.12 depicts a decision tree trained on the popular Iris flowers dataset. The dataset consists of 150 rows, and each row contains four feature variables and one target variable. The four feature variables are the sepal length, sepal width, petal length, and petal width in cm. The target variable is a string that can take three possible values—setosa, virginica, and versicolor—and indicate the species of the Iris flower. You will learn more about decision trees in Chapter 8.

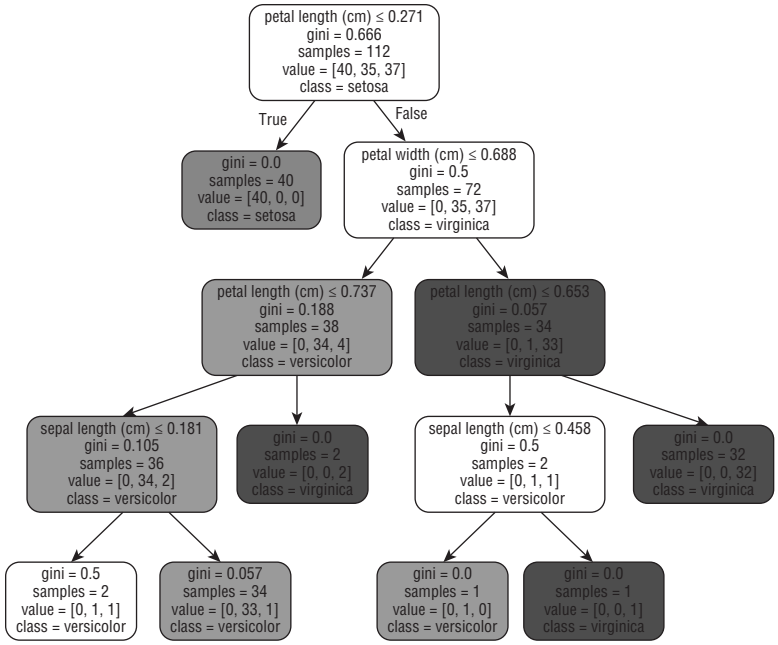


Figure 1.12 Decision tree visualization

Table 1.1 contains a sample of five rows from the Iris flowers dataset that was used to train the decision tree.

Table 1.1: Type and Range of Data Across 100 Sample Application

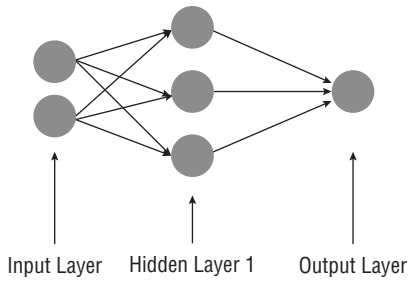
Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

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. A decision tree can also be used for regression problems, with the key difference being that each node predicts a numeric value instead of a class.

Artificial Neural Networks

Artificial neural networks (ANNs) are computing tools that were developed by Warren McCulloch and Walter Pitts in 1943, and their design is inspired by biological neural networks. Figure 1.13 depicts the structure of a simple artificial neural network.

Figure 1.13
Structure of an ANN



ANNs consist of units called *neurons* and are organized into a series of layers. There are three types of layers.

- **Input layer:** Directly receives inputs for the computation. There is only one input layer in an artificial neural network.
- **Output layer:** Provides the output of the computation. There can be one or more neurons in this layer depending on the type of problem the network is used to solve.
- **Hidden layer:** Sits between the input and the output layer. Neurons in the input layer are connected to neurons in the hidden layer, and neurons in the hidden layer are connected to the neurons in the output layer. When all of the neurons in one layer are connected to every neuron in the previous layer, the network is called a *fully connected network*. A simple neural network may not necessarily have a hidden layer, and complex neural networks such as deep learning networks have several hidden layers each with a large number of neurons in them.

Each of the connections between neurons has a weight value associated with it; the weight multiplies the value of the neuron the connection originated from. Each neuron works by computing the sum of its inputs and passing the sum through a nonlinear activation function. The output value of the neuron is the result of the activation function. Figure 1.14 depicts a simple neural network with two neurons in the input layer and one neuron in the output layer.

Figure 1.14
A simple neural network

If x_1, x_2 are the values loaded into the neurons in the input layer, if w_1, w_2 are the connection weights between the input layer and the output layer, and if f() is the activation function of the neuron in the output layer, then the output value of the neural network in Figure 1.14 is $f(w_1.x_1 + w_2.x_2)$.

There are many different types of activation functions with their own advantages and disadvantages. The reason to have an activation function is to ensure that the network is not just one big linear model. When a neural network is instantiated, the weights are set to random values. The process of training the neural network involves finding out the values of the weights.

There are many different types of neural network architectures; the field of deep learning deals with neural networks that have several hidden layers, and each layer can have several dozens, if not hundreds of neurons. Convolutional neural networks (CNNs) are a class of deep neural networks that contain a series of hidden layers called *convolution* and *pooling* layers. CNNs are commonly used for image-based tasks such as object detection and scene analysis. Convolution layers perform a filtering operation known in the image processing community as a *convolution*, the purpose of which is to transform the input data in such a way to find useful features. Pooling layers perform downsampling, and the purpose is to make the resulting model robust to minor variations in input.

Neural-network-based models are trained using libraries that allow you to quickly create a network by combining different types of layers. These libraries often provide efficient means to train the network. You can either create a CNN architecture from scratch or, more typically, use a well-known CNN architecture such as LeNet, AlexNet, VGGNet, and ResNet50. Since training a neural network is a time-consuming task and requires a lot of data, Apple provides several pre-trained models at `https://developer.apple.com/machine-learning/models/` based on popular CNN architectures.

The advantage of neural network-based models is their ability to handle complex problem spaces such as speech recognition, image understanding, etc. The disadvantage of neural network models is that they require a large amount of data to train, and it is not possible to determine why a neural-network-based model produces a certain output.

Sources of 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 downsampled versions of popular machine learning datasets such as the Iris, Boston, and Digits datasets. These datasets are often referred to as *toy* datasets, and they are helpful when you are learning to use existing machine learning algorithms or are building a new algorithm and need a small well-known dataset to work with.

Scikit-learn provides functions to load a 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 process of data exploration and feature engineering is typically performed using Jupyter Notebooks and Python. The following Python 3.6 snippet loads the toy version of the popular Iris dataset and explores the attributes of the dataset. The corresponding notebook file `Iris_dataset_exploration.ipynb` is included with the resources that accompany this chapter.

```
#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)
(150,)

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

The following are the toy datasets included with Scikit-learn:

- **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. The full version of this dataset can be found 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. The full version of this dataset can be found 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. The full version of this dataset can be found 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. The full version of this dataset can be found 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. The full version of this dataset can be found 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. The full version of this dataset can be found 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. The full version of this dataset can be found at `https://goo.gl/U2Uwz2`.

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 more than 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`.

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

You can download the code files for this chapter from Wrox.com or from GitHub using the following URL:

`https://github.com/asmtechnology/iosmlbook-chapter1.git`

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 semisupervised 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.