Listing 4.29 displays the contents of `weather_data.py` that illustrates how to read a CSV file, initialize a `Pandas DataFrame` with the contents of that CSV file, and display various subsets of the data in the `Pandas DataFrames`.

**Listing 4.29: weather_data.py**

```
import pandas as pd

df = pd.read_csv("weather_data.csv")

print(df)
print(df.shape)  # rows, columns
print(df.head()) # df.head(3)
print(df.tail())
print(df[1:3])
print(df.columns)
print(type(df['day']))
print(df[['day','temperature']])
print(df['temperature'].max())
```

Listing 4.29 invokes the `Pandas read_csv()` function to read the contents of the CSV file `weather_data.csv`, followed by a set of Python `print()` statements that display various portions of the CSV file. The output from Listing 4.29 is here:

```
day,temperature,windspeed,event
7/1/2018,42,16,Rain
7/2/2018,45,3,Sunny
7/3/2018,78,12,Snow
7/4/2018,74,9,Snow
7/5/2018,42,24,Rain
7/6/2018,51,32,Sunny
```

In some situations you might need to apply Boolean conditional logic to "filter out" some rows of data, based on a conditional condition that's applied to a column value.

Listing 4.30 displays the contents of the CSV file `people.csv` and Listing 4.31 displays the contents of `people_pandas.py` that illustrates how to define a `Pandas DataFrame` that reads the CSV file and manipulates the data.

**Listing 4.30: people.csv**

```
fname,lname,age,gender,country
```

```
john,smith,30,m,usa
jane,smith,31,f,france
jack,jones,32,f,france
dave,stone,33,f,france
sara,stein,34,f,france
eddy,bower,35,f,france
```

**Listing 4.31: people_pandas.py**

```
import pandas as pd

df = pd.read_csv('people.csv')
df.info()
print('fname:')
print(df['fname'])
print('_____________')
print('age over 33:')
print(df['age'] > 33)
print('_____________')
print('age over 33:')
myfilter = df['age'] >  33
print(df[myfilter])
```

Listing 4.31 populate the `Pandas` dataframe `df` with the contents of the CSV file people.csv. The next portion of Listing 4.12 displays the structure of `df`, followed by the first names of all the people. The next portion of Listing 4.12 displays a tabular list of six rows containing either True or False depending on whether a person is over 33 or at most 33, respectively. The final portion of Listing 4.31 displays a tabular list of two rows containing all the details of the people who are over 33. The output from Listing 4.31 is here:

```
myfilter = df['age'] >  33
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
fname      6 non_null object
lname      6 non_null object
age        6 non_null int64
gender     6 non_null object
country    6 non_null object
dtypes: int64(1), object(4)
memory usage: 320.0+ bytes
fname:
```

```
0    john
1    jane
2    jack
3    dave
4    sara
5    eddy
Name: fname, dtype: object

_____________
age over 33:
0    False
1    False
2    False
3    False
4     True
5     True
Name: age, dtype: bool

_____________
age over 33:
  fname  lname   age   gender  country
4  sara  stein   34      f     france
5  eddy  bower   35      m     france
```

## 4.33 Pandas DataFrames and Excel Spreadsheets (1)

Listing 4.32 displays the contents of people_xslx.py that illustrates how to read data from an Excel spreadsheet and create a Pandas DataFrame with that data.

**Listing 4.32: people_xslx.py**

```
import pandas as pd

df = pd.read_excel("people.xlsx")
print("Contents of Excel spreadsheet:")
print(df)
```

Listing 4.32 is straightforward: the Pandas dataframe df is initialized with the contents of the spreadsheet people.xlsx (whose contents are the same as people.csv) via the Pandas function read_excel(). The output from Listing 4.32 is here:

```
  fname  lname  age gender country
0  john  smith   30      m      usa
```

```
1  jane  smith   31      f   france
2  jack  jones   32      f   france
3  dave  stone   33      f   france
4  sara  stein   34      f   france
5  eddy  bower   35      f   france
```

## 4.34 Select, Add, and Delete Columns in `DataFrames`

This section contains short code blocks that illustrate how to perform operations on a `DataFrame` that resemble the operations on a Python dictionary. For example, getting, setting, and deleting columns works with the same syntax as the analogous Python `dict` operations, as shown here:

```
df = pd.DataFrame.from_dict(dict([('A',[1,2,3]),
( 'B',[4,5,6])]),
orient='index', columns=['one', 'two', 'three'])

print(df)
```

The output from the preceding code snippet is here:

```
   one  two  three
A    1    2      3
B    4    5      6
```

Now look at the following sequence of operations on the contents of the dataframe df:

```
df['three'] = df['one'] * df['two']
df['flag'] = df['one'] > 2
print(df)
```

The output from the preceding code block is here:

```
   one  two   three    flag
a  1.0  1.0    1.0   False
b  2.0  2.0    4.0   False
c  3.0  3.0    9.0    True
d  NaN  4.0    NaN   False
```

Columns can be deleted or popped like with a Python `dict`, as shown in following code snippet:

```
del df['two']
three = df.pop('three')
print(df)
```

The output from the preceding code block is here:

```
   one   flag
a  1.0  False
b  2.0  False
c  3.0   True
d  NaN  False
```

When inserting a scalar value, it will naturally be propagated to fill the column:

```
df['foo'] = 'bar'
print(df)
```

The output from the preceding code snippet is here:

```
   one   flag   foo
a  1.0  False  bar
b  2.0  False  bar
c  3.0   True  bar
d  NaN  False  bar
```

When inserting a `Series` that does not have the same index as the `DataFrame`, it will be "conformed" to the index of the `DataFrame`:

```
df['one_trunc'] = df['one'][:2]
print(df)
```

The output from the preceding code snippet is here:

```
   one   flag   foo   one_trunc
a  1.0  False  bar         1.0
b  2.0  False  bar         2.0
c  3.0   True  bar         NaN
d  NaN  False  bar         NaN
```

You can insert raw `ndarrays` but their length must match the length of the index of the `DataFrame`.

## 4.35 Pandas DataFrames and Scatterplots

Listing 4.33 displays the contents of `pandas_scatter_df.py` that illustrates how to generate a scatterplot from a `Pandas DataFrame`.

**Listing 4.33: pandas_scatter_df.py**

```
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from pandas import read_csv
from pandas.plotting import scatter_matrix

myarray = np.array([[10,30,20],
[50,40,60],[1000,2000,3000]])

rownames = ['apples', 'oranges', 'beer']
colnames = ['January', 'February', 'March']

mydf = pd.DataFrame(myarray, index=rownames,
columns=colnames)

print(mydf)
print(mydf.describe())

scatter_matrix(mydf)
plt.show()
```

Listing 4.33 starts with various import statements, followed by the definition of the `NumPy` array `myarray`. Next, the variables `myarray` and `colnames` are initialized with values for the rows and columns, respectively. The next portion of Listing 4.33 initializes the `Pandas DataFrame mydf` so that the rows and columns are labeled in the output, as shown here:

```
           January   February  March
apples        10        30      20
oranges       50        40      60
beer        1000      2000    3000
           January     February        March
count     3.000000     3.000000     3.000000
mean    353.333333   690.000000  1026.666667
std     560.386771  1134.504297  1709.073823
min      10.000000    30.000000    20.000000
25%      30.000000    35.000000    40.000000
50%      50.000000    40.000000    60.000000
75%     525.000000  1020.000000  1530.000000
max    1000.000000  2000.000000  3000.0000000
```

## 4.36 Pandas DataFrames and Simple Statistics

Listing 4.34 displays the contents of `housing_stats.py` that illustrates how to gather basic statistics from data in a `Pandas DataFrame`.

**Listing 4.34: housing_stats.py**

```
import pandas as pd

df = pd.read_csv("Housing.csv")

minimum_bdrms = df["bedrooms"].min()
median_bdrms  = df["bedrooms"].median()
maximum_bdrms = df["bedrooms"].max()

print("minimum # of bedrooms:",minimum_bdrms)
print("median  # of bedrooms:",median_bdrms)
print("maximum # of bedrooms:",maximum_bdrms)
print("")

print("median values:",df.median().values)
print("")

prices = df["price"]
print("first 5 prices:")
print(prices.head())
print("")

median_price = df["price"].median()
print("median price:",median_price)
print("")

corr_matrix = df.corr()
print("correlation matrix:")
print(corr_matrix["price"].sort_values(ascending=False))
```

Listing 4.34 initializes the `Pandas DataFrame df` with the contents of the CSV file `Housing.csv`. The next three variables are initialized with the minimum, median, and maximum number of bedrooms, respectively, and then these values are displayed.

The next portion of Listing 4.34 initializes the variable `prices` with the contents of the prices column of the `Pandas DataFrame df`. Next, the first five rows are printed via the `prices.head()` statement, followed by the median value of the prices.

The final portion of Listing 4.34 initializes the variable `corr_matrix` with the contents of the correlation matrix for the `Pandas DataFrame df`, and then displays its contents. The output from Listing 4.34 is here:

```
Apples
10
```

## 4.37 Useful `One_line` Commands in Pandas

This section contains an eclectic mix of one-line commands in `Pandas` (some of which you have already seen in this chapter) that are useful to know:

Save a dataframe to a CSV file (comma separated and without indices):

```
df.to_csv("data.csv", sep=",", index=False)
```

List the column names of a `DataFrame`:

```
df.columns
```

Drop missing data from a `DataFrame`:

```
df.dropna(axis=0, how='any')
```

Replace missing data in a `DataFrame`:

```
df.replace(to_replace=None, value=None)
```

Check for NANs in a `DataFrame`:

```
pd.isnull(object)
```

Drop a feature in a `DataFrame`:

```
df.drop('feature_variable_name', axis=1)
```

Convert object type to float in a `DataFrame`:

```
pd.to_numeric(df["feature_name"], errors='coerce')
```

Convert data in a `DataFrame` to `NumpPy` array:

```
df.as_matrix()
```

Display the first n rows of a dataframe:

```
df.head(n)
```

Get data by feature name in a `DataFrame`:

```
df.loc[feature_name]
```

Apply a function to a `DataFrame`: multiply all values in the "height" column of the dataframe by 3:

```
df["height"].apply(lambda height: 3 * height)
```

OR:

```
def multiply(x):
    return x * 3
```

```
df["height"].apply(multiply)
```

Rename the fourth column of the dataframe as "height":

```
df.rename(columns = {df.columns[3]:'height'},
inplace=True)
```

Get the unique entries of the column "first" in a `DataFrame`:

```
df[""first"].unique()
```

Create a dataframe with columns "first" and "last" from an existing `DataFrame`:

```
new_df = df[["name", "size"]]
```

Sort the data in a `DataFrame`:

```
df.sort_values(ascending = False)
```

Filter the data column named "size" to display only values equal to 7:

```
df[df["size"] == 7]
```

Select the first row of the "height" column in a `DataFrame`:

```
df.loc([0], ['height'])
```

This concludes the `Pandas` related portion of the chapter. The next section contains a brief introduction to `Jupyter`, which is a Flask-based Python application that enables you to execute Python code in a browser. Instead of Python scripts, you will use `Jupyter` "notebooks," which support various interactive features for executing Python code. In addition, your knowledge of `Jupyter` will be very useful when you decide to use Google Colaboratory (discussed later) that also supports `Jupyter` notebooks in a browser.

## 4.38 Summary

This chapter introduced you to `Pandas` for creating labeled `Dataframes` and displaying metadata of `Pandas Dataframes`. Then you learned how to create Pandas `Dataframes` from various sources of data, such as random numbers and hard_coded data values.

You also learned how to read Excel spreadsheets and perform numeric calculations on that data, such as the min, mean, and max values in numeric columns. Then you saw how to create `Pandas Dataframes` from

data stored in CSV files. Then you learned how to invoke a Web service to retrieve data and populate a `Pandas Dataframe` with that data. In addition, you learned how to generate a scatterplot from data in a `Pandas Dataframe`. Finally, you saw how to use `Jupyter`, which is a Python-based application for displaying and executing Python code in a browser.

CHAPTER 5

# INTRODUCTION TO MACHINE LEARNING

- What is Machine Learning?
- Types of Machine Learning Algorithms
- Feature Engineering, Selection, and Extraction
- Dimensionality Reduction
- Working with Datasets
- What is Regularization?
- The Bias-Variance Tradeoff
- Metrics for Measuring Models
- Other Useful Statistical Terms
- What is Linear Regression?
- Other Types of Regression
- Working with Lines in the Plane (optional)
- Scatter Plots with `Numpy` and Matplotlib (1)
- Scatter Plots with `Numpy` and Matplotlib (2)
- A Quadratic Scatterplot with `Numpy` and matplotlib
- The MSE Formula
- Calculating the MSE Manually
- Approximating Linear Data with `np.linspace()`
- Calculating MSE with np.linspace() API

- Linear Regression with Keras
- Summary

This chapter introduces numerous concepts in machine learning, such as feature selection, feature engineering, data cleaning, training sets, and test sets.

The first part of this chapter briefly discusses machine learning and the sequence of steps that are typically required *in order to prepare a dataset. These steps include feature selection or feature extraction* that can be performed using various algorithms.

The second section describes the types of data that you can encounter, issues that can arise with the data in datasets, and how to rectify them. You will also learn about the difference between *hold out* and *k-fold* when you perform the training step.

The third part of this chapter briefly discusses the basic concepts involved in linear regression. Although linear regression was developed more than 200 years ago, this technique is still one of the core techniques for solving (albeit simple) problems in statistics and machine learning. In fact, the technique known as mean squared error (MSE) for finding a best-fitting line for data points in a 2D plane (or a hyperplane for higher dimensions) is implemented in Python and `TensorFlow` in order to minimize so-called loss functions that are discussed later.

The fourth section in this chapter contains additional code samples involving linear regression tasks using standard techniques in `NumPy`. Hence, if you are comfortable with this topic, you can probably skim quickly through the first two sections of this chapter. The third section shows you how to solve linear regression using `Keras`.

One point to keep in mind is that some algorithms are mentioned without delving into details about them. For instance, the section pertaining to supervised learning contains a list of algorithms that appear later in the chapter in the section that pertains to classification algorithms. The algorithms that are displayed in bold in a list are the algorithms that are of greater interest for this book. In some cases the algorithms are discussed in greater detail in the next chapter; otherwise, you can perform an online search for additional information about the algorithms that are not discussed in detail in this book.

## 5.1 What is Machine Learning?

In high-level terms, machine learning is a subset of AI that can solve tasks that are infeasible or too cumbersome with more traditional programming languages. A spam filter for email is an early example of machine learning. Machine learning generally supersedes the accuracy of older algorithms.

Despite the variety of machine learning algorithms, the data is arguably more important than the selected algorithm. Many issues can arise with data, such as insufficient data, poor quality of data, incorrect data, missing data, irrelevant data, duplicate data values, and so forth. Later in this chapter you will see techniques that address many of these data-related issues.

If you are unfamiliar with machine learning terminology, a dataset is a collection of data values, which can be in the form of a CSV file or a spreadsheet. Each column is called a feature, and each row is a datapoint that contains a set of specific values for each feature. If a dataset contains information about customers, then each row pertains to a specific customer.

### 5.1.1 Types of Machine Learning

There are three main types of machine learning (combinations of these are also possible) that you will encounter:

- supervised learning
- unsupervised learning
- semisupervised learning

*Supervised learning* means that the datapoints in a dataset have a label that identifies its contents. For example, the `MNIST` dataset contains 28x28 PNG files, each of which contains a single hand-drawn digit (i.e. 0 through 9 inclusive). Every image with the digit 0 has the label 0; every image with the digit 1 has the label 1; all other images are labeled according to the digit that is displayed in those images.

As another example, the columns in the Titanic dataset are features about passengers, such as their gender, the cabin class, the price of their ticket, whether or not the passenger survived, and so forth. Each row contains information about a single passenger, including the value 1 if the passenger survived. The MNIST dataset and the Titanic dataset involve

*classification* tasks: the goal is to train a model based on a training dataset and then predict the class of each row in a test dataset.

In general, the datasets for classification tasks have a small number of possible values: one of nine digits in the range of 0 through 9, one of four animals (dog, cat, horse, giraffe), one of two values (survived versus perished, purchased versus not purchased). As a rule of thumb, if the number of outcomes can be displayed in a relatively small number of values (which is subjective number) in a drop-down list, then it's probably a classification task.

In the case of a dataset that contains real estate data, each row contains information about a specific house, such as the number of bedrooms, the square feet of the house, the number of bathrooms, the price of the house, and so forth. In this dataset the price of the house is the label for each row. Notice that the range of possible prices is too large to fit reasonably well in a drop-down list. A real estate dataset involves a *regression* task: the goal is to train a model based on a training dataset and then predict the price of each house in a test dataset.

*Unsupervised learning* involves unlabeled data, which is typically the case for clustering algorithms (discussed later). Some important unsupervised learning algorithms that involve *clustering* are as follows:

- k-Means
- hierarchical cluster analysis (HCA)
- expectation maximization

Some important unsupervised learning algorithms that involve *dimensionality reduction* (discussed in more detail later) are as follows:

- principal component analysis (PCA)
- kernel PCA
- locally linear embedding (LLE)
- t-distributed stochastic neighbor embedding (t-SNE)

There is one more very important unsupervised task called anomaly detection. This task is relevant for fraud detection and detecting outliers (discussed later in more detail).

*Semisupervised* learning is a combination of supervised and unsupervised learning: some datapoints are labeled and some are unlabeled. One

technique involves using the labeled data in order to classify (i.e., label) the unlabeled data, after which you can apply a classification algorithm.

## 5.2 Types of Machine Learning Algorithms

There are three main types of machine learning algorithms:

- regression (ex., linear regression)
- classification (ex., k-nearest-neighbor)
- clustering (ex., kMeans)

*Regression* is a supervised learning technique to predict numerical quantities. An example of a regression task is predicting the value of a particular stock. Note that this task is different from predicting whether the value of a particular stock will increase or decrease tomorrow (or some other future time period). Another example of a regression task involves predicting the cost of a house in a real estate dataset. Both of these tasks are examples of a regression task.

Regression algorithms in machine learning include linear regression and generalized linear regression (also called multivariate analysis in traditional statistics).

*Classification* is also a supervised learning technique, for predicting numeric or categorical quantities. An example of a classification task is detecting the occurrence of spam, fraud, or determining the digit in a PNG file (such as the `MNIST` dataset). In this case, the data is already labeled, so you can compare the prediction with the label that was assigned to the given PNG.

Classification algorithms in machine learning include the following list of algorithms (they are discussed in greater detail in the next chapter):

- decision trees (a single tree)
- random forests (multiple trees)
- kNN (k nearest neighbor)
- logistic regression (despite its name)
- naïve Bayes
- support vector machines (SVM)

Some machine learning algorithms (such as SVMs, random forests, and kNN) support regression as well as classification. In the case of SVMs, the scikit-learn implementation of this algorithm provides two APIs: SVC for classification and SVR for regression.

Each of the preceding algorithms involves a model that is trained on a dataset, after which the model is used to make a prediction. By contrast, a random forest consists of *multiple* independent trees (the number is specified by you), and each tree makes a prediction regarding the value of a feature. If the feature is numeric, take the mean or the mode (or perform some other calculation) in order to determine the final prediction. If the feature is categorical, use the mode (i.e., the most frequently occurring class) as the result; in the case of a tie you can select one of them in a random fashion.

Incidentally, the following link contains more information regarding the kNN algorithm for classification as well as regression:

*http://saedsayad.com/k_nearest_neighbors_reg.htm*

*Clustering* is an unsupervised learning technique for grouping similar data together. Clustering algorithms put data points in different clusters without knowing the nature of the data points. After the data has been separated into different clusters, you can use the SVM (Support Vector Machine) algorithm to perform classification.

Clustering algorithms in machine learning include the following (some of which are variations of each other):

- k-Means
- meanshift
- hierarchical cluster analysis (HCA)
- expectation maximization

Keep in mind the following points. First, the value of `k` in `k-Means` is a hyperparameter, and it's usually an odd number to avoid ties between two classes. Next, the `meanshift` algorithm is a variation of the `k-Means` algorithm that does *not* require you to specify a value for k. In fact, the `meanshift` algorithm determines the optimal number of clusters. However, this algorithm does not scale well for large datasets.

### 5.2.1 Machine Learning Tasks

Unless you have a dataset that has already been sanitized, you need to examine the data in a dataset to make sure that it's in a suitable condition.

The data preparation phase involves (1) examining the rows (data cleaning) to ensure that they contain valid data (which might require domain-specific knowledge), and (2) examining the columns (feature selection or feature extraction) to determine if you can retain only the most important columns.

A high-level list of the sequence of machine learning tasks (some of which might not be required) is shown here:

- obtain a dataset
- data cleaning
- feature selection
- dimensionality reduction
- algorithm selection
- train-versus-test data
- training a model
- testing a model
- fine-tuning a model
- obtain metrics for the model

First, you obviously need to obtain a dataset for your task. In the ideal scenario, this dataset already exists; otherwise, you need to cull the data from one or more data sources (e.g., a CSV file, a relational database, a no-SQL database, a Web service, and so forth).

Second, you need to perform *data cleaning*, which you can do via the following techniques:

- missing value ratio
- low variance filter
- high correlation filter

In general, data cleaning involves checking the data values in a dataset in order to resolve one or more of the following:

- Fix incorrect values.
- Resolve duplicate values.
- Resolve missing values.
- Decide what to do with outliers.

Use the missing value ratio technique if the dataset has too many missing values. In extreme cases, you might be able to drop features with a large number of missing values. Use the low variance filter technique to identify and drop features with constant values from the dataset. Use the high correlation filter technique to find highly correlated features, which increase multicollinearity in the dataset: such features can be removed from a dataset (but check with your domain expert before doing so).

Depending on your background and the nature of the dataset, you might need to work with a domain expert, which is a person who has a deep understanding of the contents of the dataset.

For example, you can use a statistical value (mean, mode, and so forth) to replace incorrect values with suitable values. Duplicate values can be handled in a similar fashion. You can replace missing numeric values with zero, the minimum, the mean, the mode, or the maximum value in a numeric column. You can replace missing categorical values with the mode of the categorical column.

If a row in a dataset contains a value that is an outlier, you have three choices:

- Delete the row.
- Keep the row.
- Replace the outlier with some other value (mean?).

*When a dataset contains an outlier, you need to make a decision based on domain knowledge that is specific to the given dataset.*

Suppose that a dataset contains stock-related information. As you know, there was a stock market crash in 1929, which you can view as an outlier. Such an occurrence is rare, but it can contain meaningful information. Incidentally, the source of wealth for some families in the 20th century was based on buying massive amounts of stock are very low prices during the Great Depression.

## 5.3 Feature Engineering, Selection, and Extraction

In addition to creating a dataset and cleaning its values, you also need to examine the features in that dataset to determine whether or not you can

reduce the dimensionality (i.e., the number of columns) of the dataset. The process for doing so involves three main techniques:

- feature engineering
- feature selection
- feature extraction (aka feature projection)

*Feature engineering* is the process of determining a new set of features that are based on a combination of existing features in order to create a meaningful dataset for a given task. Domain expertise is often required for this process, even in cases of relatively simple datasets. Feature engineering can be tedious and expensive, and in some cases you might consider using automated feature learning. After you have created a dataset, it's a good idea to perform feature selection or feature extraction (or both) to ensure that you have a high quality dataset.

*Feature selection* is also called variable selection, attribute selection or variable subset selection. Feature selection involves of selecting a subset of relevant features in a dataset. In essence, feature selection involves selecting the most significant features in a dataset, which provides these advantages:

- reduced training time
- simpler models are easier to interpret
- avoidance of the curse of dimensionality
- better generalization due to a reduction in overfitting (*reduction of variance*)

Feature selection techniques are often used in domains where there are many features and comparatively few samples (or data points). Keep in mind that a low-value feature can be redundant or irrelevant, which are two different concepts. For instance, a relevant feature might be redundant when it's combined with another strongly correlated feature.

Feature selection can use three strategies: the filter strategy (e.g., information gain), the wrapper strategy (e.g., search guided by accuracy), and the embedded strategy (prediction errors are used to determine whether features are included or excluded while developing a model). One other interesting point is that feature selection can also be useful for regression as well as for classification tasks.

*Feature extraction* creates new features from functions that produce combinations of the original features. By contrast, feature selection involves determining a subset of the existing features.

Feature selection and feature extraction both result in *dimensionality reduction* for a given dataset, which is the topic of the next section.

## 5.4 Dimensionality Reduction

Dimensionality Reduction refers to algorithms that reduce the number of features in a dataset: hence the term *dimensionality reduction*. As you will see, there are many techniques available, and they involve either feature selection or feature extraction.

Algorithms that use feature selection to perform dimensionality reduction are listed here:

- backward feature elimination
- forward feature selection
- factor analysis
- independent component analysis

Algorithms that use feature extraction to perform dimensionality reduction are listed here:

- principal component analysis (PCA)
- nonnegative matrix factorization (NMF)
- kernel PCA
- graph-based kernel PCA
- linear discriminant analysis (LDA)
- generalized discriminant analysis (GDA)
- autoencoder

The following algorithms combine feature extraction and dimensionality reduction:

- principal component analysis (PCA)
- linear discriminant analysis (LDA)

- canonical correlation analysis (CCA)
- nonnegative matrix factorization (NMF)

These algorithms can be used during a preprocessing step before using clustering or some other algorithm (such as kNN) on a dataset.

One other group of algorithms involves methods based on projections, which includes t-distributed stochastic neighbor embedding (t-SNE) as well as UMAP (Uniform Manifold Approximation and Projection).

This chapter discusses PCA, and you can perform an online search to find more information about the other algorithms.

### 5.4.1 PCA

Principal components are new components that are linear combinations of the initial variables in a dataset. In addition, these components are uncorrelated and the most meaningful or important information is contained in these new components.

There are two advantages to PCA: (1) reduced computation time due to far fewer features, and (2) the ability to graph the components when there are at most three components. If you have four or five components, you won't be able to display them visually, but you could select subsets of three components for visualization, and perhaps gain some additional insight into the dataset.

PCA uses the variance as a measure of information: the higher the variance, the more important the component. In fact, just to jump ahead slightly: PCA determines the eigenvalues and eigenvectors of a covariance matrix (discussed later), and constructs a new matrix whose columns are eigenvectors, ordered from left-to-right based on the maximum eigenvalue in the left-most column, decreasing until the right-most eigenvector also has the smallest eigenvalue.

### 5.4.2 Covariance Matrix

As a reminder, the statistical quantity called the variance of a random variable `X` is defined as follows:

```
variance(x) = [SUM (x - xbar)*(x-xbar)]/n
```

A covariance matrix `C` is an nxn matrix whose values on the main diagonal are the variance of the variables `X1`, `X2`, . . ., `Xn`. The other values of `C` are the covariance values of each pair of variables `Xi` and `Xj`.

The formula for the covariance of the variables X and Y is a generalization of the variance of a variable, and the formula is shown here:

```
covariance(X, Y) = [SUM (x - xbar)*(y-ybar)]/n
```

Notice that you can reverse the order of the product of terms (multiplication is commutative), and therefore the covariance matrix C is a symmetric matrix:

```
covariance(X, Y) = covariance(Y,X)
```

*PCA calculates the eigenvalues and the eigenvectors of the covariance matrix A.*

## 5.5 Working with Datasets

In addition to data cleaning, there are several other steps that you need to perform, such as selecting training data versus test data, and deciding whether to use hold out or cross-validation during the training process. More details are provided in the subsequent sections.

### 5.5.1 Training Data versus Test Data

After you have performed the tasks described earlier in this chapter (i.e., data cleaning and perhaps dimensionality reduction), you are ready to split the dataset into two parts. The first part is the *training set*, which is used to train a model, and the second part is the *test set*, which is used for *inferencing* (another term for making predictions). Make sure that you conform to the following guidelines for your test sets:

1. The set is large enough to yield statistically meaningful results.
2. It's representative of the dataset as a whole.
3. Never train on test data.
4. Never test on training data.

### 5.5.2 What is Cross-validation?

The purpose of cross-validation is to test a model with nonoverlapping test sets, and is performed in the following manner:

1. Split the data into k subsets of equal size.
2. Select one subset for testing and the others for training.
3. Repeat step 2 for the other k-1 subsets.

This process is called *k-fold cross-validation*, and the overall error estimate is the average of the error estimates. A standard method for evaluation involves ten-fold cross-validation. Extensive experiments have shown that 10 subsets is the best choice to obtain an accurate estimate. In fact, you can repeat ten-fold cross-validation ten times and compute the average of the results, which helps to reduce the variance.

The next section discusses regularization, which is an important yet optional topic if you are primarily interested in TF 2 code. If you plan to become proficient in machine learning, you will need to learn about regularization.

## 5.6 What is Regularization?

Regularization helps to solve overfitting problem, which occurs when a model performs well on training data but poorly on validation or test data.

Regularization solves this problem by adding a penalty term to the cost function, thereby controlling the model complexity with this penalty term. Regularization is generally useful for:

- large number of variables
- low ration of (# observations)/(# of variables
- high multicollinearity

There are two main types of regularization: L1 regularization (which is related to MAE, or the absolute value of differences) and L2 regularization (which is related to MSE, or the square of differences). In general, L2 performs better than L1 and L2 is efficient in terms of computation.

### 5.6.1 ML and Feature Scaling

Feature scaling standardizes the range of features of data. This step is performed during the data preprocessing step, in part because gradient descent benefits from feature scaling.

The assumption is that the data conforms to a standard normal distribution, and standardization involves subtracting the mean and divide by the standard deviation for every data point, which results in a N(0,1) normal distribution.

### 5.6.2 Data Normalization versus Standardization

Data normalization is a linear scaling technique. Let's assume that a dataset has the values `{X1, X2, . . . , Xn}` along with the following terms:

```
Minx = minimum of Xi values
```

```
Maxx = maximum of Xi values
```

Now calculate a set of new `Xi` values as follows:

```
Xi = (Xi – Minx)/[Maxx – Minx]
```

The new `Xi` values are now scaled so that they are between 0 and 1.

## 5.7 The Bias-Variance Tradeoff

*Bias* in machine learning can be due to an error from wrong assumptions in a learning algorithm. High bias might cause an algorithm to miss relevant relations between features and target outputs (underfitting). Prediction bias can occur because of "noisy" data, an incomplete feature set, or a biased training sample.

Error due to bias is the difference between the expected (or average) prediction of your model and the correct value that you want to predict. Repeat the model building process multiple times, and gather new data each time, and also perform an analysis to produce a new model. The resulting models have a range of predictions because the underlying datasets have a degree of randomness. Bias measures the extent to the predictions for these models are from the correct value.

*Variance* in machine learning is the expected value of the squared deviation from the mean. High variance can/might cause an algorithm to model the random noise in the training data, rather than the intended outputs (aka overfitting).

Adding parameters to a model increases its complexity, increases the variance, and decreases the bias. Dealing with bias and variance is dealing with underfitting and overfitting.

Error due to variance is the variability of a model prediction for a given data point. As before, repeat the entire model building process, and the variance is the extent to which predictions for a given point vary among different instances of the model.

## 5.8 Metrics for Measuring Models

One of the most frequently used metrics is R-squared, which measures how close the data is to the fitted regression line (regression coefficient). The R-squared value is always a percentage between 0 and 100%. The value

0% indicates that the model explains none of the variability of the response data around its mean. The value 100% indicates that the model explains all the variability of the response data around its mean. In general, a higher R-squared value indicates a better model.

### 5.8.1 Limitations of R-Squared

Although high R-squared values are preferred, they are not necessarily always good values. Similarly, low R-squared values are not always bad. For example, an R-squared value for predicting human behavior is often less than 50%. Moreover, R-squared cannot determine whether the coefficient estimates and predictions are biased. In addition, an R-squared value does not indicate whether a regression model is adequate. Thus, it's possible to have a low R-squared value for a good model, or a high R-squared value for a poorly fitting model. Evaluate R-squared values in conjunction with residual plots, other model statistics, and subject area knowledge.

### 5.8.2 Confusion Matrix

In its simplest form, a confusion matrix (also called an error matrix) is a type of contingency table with two rows and two columns that contains the # of false positives, false negatives, true positives, and true negatives. The four entries in a 2x2 confusion matrix can be labeled as follows:

```
TP: True Positive
FP: False Positive
TN: True Negative
FN: False Negative
```

The diagonal values of the confusion matrix are correct, whereas the off-diagonal values are incorrect predictions. In general a lower FP value is better than a FN value. For example, an FP indicates that a healthy person was incorrectly diagnosed with a disease, whereas an FN indicates that an unhealthy person was incorrectly diagnosed as healthy.

### 5.8.3 Accuracy versus Precision versus Recall

A 2x2 confusion matrix has four entries that that represent the various combinations of correct and incorrect classifications. Given the definitions in the preceding section, the definitions of precision, accuracy, and recall are given by the following formulas:

```
precision = TP/(TN + FP)
accuracy  = (TP + TN)/[P + N]
```