Chapter 2. Introduction to Time Series Modeling

*Market behavior is examined using large amounts of past data, such as high-frequency bid-ask quotes of currencies or stock prices. It is the abundance of data that makes possible the empirical study of the market. Although it is not possible to run controlled experiments, it is possible to extensively test on historical data.*

 Sergio Focardi (1997)

Some models account better for some phenomena; certain approaches capture the characteristics of an event in a solid way. Time series modeling is a good example of this because the vast majority of financial data has a time dimension, which makes time series applications a necessary tool for finance. In simple terms, the ordering of the data and its correlation is important.

This chapter of the book will discuss classical time series models and compare the performance of these models. Deep learning–based time series analysis will be introduced in [Chapter 3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#chapter_3); this is an entirely different approach in terms of data preparation and model structure. The classical models include the moving average (MA), autoregressive (AR), and autoregressive integrated moving average (ARIMA) models. What is common across these models is the information carried by the historical observations. If these historical observations are obtained from error terms, we refer to this as a *moving average*; if these observations come out of time series itself, it is called *autoregressive*. The other model, ARIMA, is an extension of these models.

Here is a formal definition of *time series* from Brockwell and Davis (2016):

*A time series is a set of observations ��, each one being recorded at a specific time t. A discrete-time time series… is one in which the set �0 of times at which observations are made is a discrete set, as is the case, for example, when observations are made at fixed time intervals. Continuous time series are obtained when observations are recorded continuously over some time interval.*

Let’s observe what data with time dimension looks like. [Figure 2-1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#raw_oil_price) exhibits the oil prices for the period of 1980–2020, and the following Python code shows us a way of producing this plot:

In [1]: import quandl

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

plt.style.use('seaborn')

In [2]: oil = quandl.get("NSE/OIL", authtoken="insert you api token",

start\_date="1980-01-01",

end\_date="2020-01-01") [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO1-1)

In [3]: plt.figure(figsize=(10, 6))

plt.plot(oil.Close)

plt.ylabel('$')

plt.xlabel('Date')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO1-1)*

Extracting data from Quandl database



Figure 2-1. Oil prices between 1980 and 2020

**NOTE**

An API is a tool designed for retrieving data using code. We will make use of different APIs throughout the book. In the preceding practice, Quandl API is used.

Quandl API allows us to access financial, economic, and alternative data from the Quandl website. To get your Quandl API, please visit the [Quandl website](https://oreil.ly/1IFDc) first and follow the necessary steps to get your own API key.

As can be understood from the definition provided previously, time series models can be applicable to diverse areas such as:

* Health care
* Finance
* Economics
* Network analysis
* Astronomy
* Weather

The superiority of the time series approach comes from the idea that correlations of observations in time better explain the current value. Having data with a correlated structure in time implies a violation of the famous identically and independently distributed (IID) assumption, which is at the heart of many models.

**THE DEFINITION OF IID**

IID assumption enables us to model joint probability of data as the product of probability of observations. The process �� is said to be an IID with mean 0 and variance �2:

��∼���(0,�2)

So, due to the correlation in time, the dynamics of a contemporaneous stock price can be better understood by its own historical values. How can we comprehend the dynamics of the data? This is a question that we can address by elaborating the components of time series.

Time Series Components

Time series has four components: trend, seasonality, cyclicality, and residual. In Python, we can easily visualize the components of a time series with the seasonal\_decompose function:

In [4]: import yfinance as yf

import numpy as np

import pandas as pd

import datetime

import statsmodels.api as sm

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.seasonal import seasonal\_decompose

In [5]: ticker = '^GSPC' [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO2-1)

start = datetime.datetime(2015, 1, 1) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO2-2)

end = datetime.datetime(2021, 1, 1) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO2-2)

SP\_prices = yf.download(ticker, start=start, end=end, interval='1mo')\

.Close [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO2-3)

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

In [6]: seasonal\_decompose(SP\_prices, period=12).plot()

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO2-1)*

Denoting ticker of S&P 500

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO2-2)*

Identifying the start and end dates

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO2-4)*

Accessing the closing price of S&P 500

In the top panel of [Figure 2-2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#time_series_decompose), we see the plot of raw data, and in the second panel, trend can be observed showing upward movement. In the third panel, seasonality is exhibited, and finally residual is presented showing erratic fluctuations. You might wonder where the cyclicality component is; noise and the cyclical component are put together under the residual component.

Becoming familiar with time series components is important for further analysis so that we are able to understand characteristics of the data and propose a suitable model. Let’s start with the trend component.

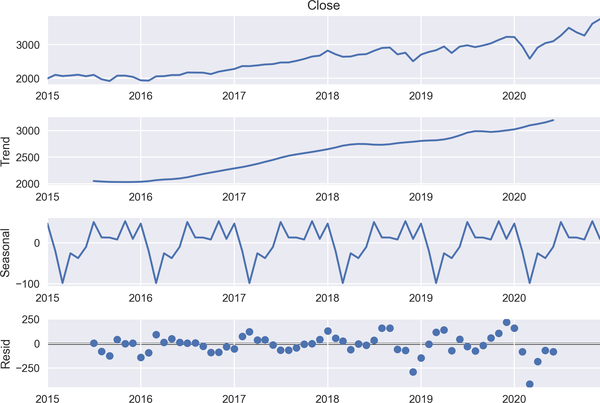


Figure 2-2. Time series decomposition of S&P 500

**Trend**

*Trend* indicates a general tendency of an increase or decrease during a given time period. Generally speaking, trend is present when the starting and ending points are different or have upward/downward slope in a time series. The following code shows what a trend looks like:

In [7]: plt.figure(figsize=(10, 6))

plt.plot(SP\_prices) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO3-1)

plt.title('S&P-500 Prices')

plt.ylabel('$')

plt.xlabel('Date')

plt.show()

Aside from the period in which the S&P 500 index price plunges, we see a clear upward trend in [Figure 2-3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#raw_sp_price) between 2010 and 2020.



Figure 2-3. S&P 500 price

A line plot is not the only option for understanding trend. Rather, we have some other strong tools for this task. So, at this point, it is worthwhile to talk about two important statistical concepts:

* Autocorrelation function
* Partial autocorrelation function

The autocorrelation function (ACF) is a statistical tool to analyze the relationship between the current value of a time series and its lagged values. Graphing ACF enables us to readily observe the serial dependence in a time series:

�^(ℎ)=Cov(��,��-ℎ)Var(��)

[Figure 2-4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#acf) denotes the ACF plot. The vertical lines represent the correlation coefficients; the first line denotes the correlation of the series with its 0 lag—that is, it is the correlation with itself. The second line indicates the correlation between series value at time *t* - 1 and *t*. In light of these, we can conclude that the S&P 500 shows a serial dependence. There appears to be a strong dependence between the current value and lagged values of S&P 500 data because the correlation coefficients, represented by lines in the ACF plot, decay in a slow fashion.

Here is how we can plot the ACF in Python:

In [8]: sm.graphics.tsa.plot\_acf(SP\_prices, lags=30) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO3-1)

plt.xlabel('Number of Lags')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO3-1)*

Plotting ACF

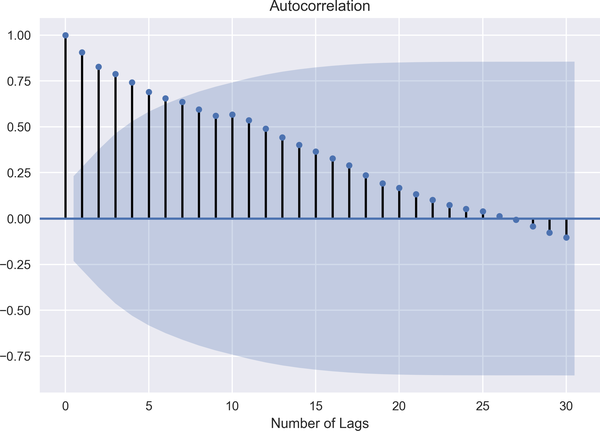


Figure 2-4. ACF plot of the S&P 500

Now the question is, what are the likely sources of autocorrelations? Here are some causes:

* The primary source of autocorrelation is “carryover,” meaning that the preceding observation has an impact on the current one.
* Model misspecification.
* Measurement error, which is basically the difference between observed and actual values.
* Dropping a variable, which has an explanatory power.

Partial autocorrelation function (PACF) is another method of examining the relationship between �� and ��-�,�∈ℤ. ACF is commonly considered as a useful tool in the MA(q) model simply because PACF does not decay fast but approaches toward 0. However, the pattern of ACF is more applicable to MA. PACF, on the other hand, works well with the AR(p) process.

PACF provides information on the correlation between the current value of a time series and its lagged values, controlling for the other correlations.

It is not easy to figure out what is going on at first glance. Let me give you an example. Suppose that we want to compute the partial correlation �� and ��-ℎ.

Put mathematically:

�^(ℎ)=Cov(��,��-ℎ|��-1,��-2...��-ℎ-1)Var(��|��-1,��-2,...,��-ℎ-1)Var(��-ℎ|��-1,��-2,...,��-ℎ-1)

where *h* is the lag. Take a look at the Python code for a PACF plot of the S&P 500 in the following snippet:

In [9]: sm.graphics.tsa.plot\_pacf(SP\_prices, lags=30) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO4-1)

plt.xlabel('Number of Lags')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO4-1)*

Plotting PACF

[Figure 2-5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#pacf) exhibits the PACF of raw S&P 500 data. In interpreting the PACF, we focus on the spikes outside the dark region representing confidence interval. [Figure 2-5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#pacf) exhibits some spikes at different lags, but lag 10 is outside the confidence interval. So it may be wise to select a model with 10 lags to include all the lags up to lag 10.

As discussed, PACF measures the correlation between current values of series and lagged values in a way to isolate in-between effects.

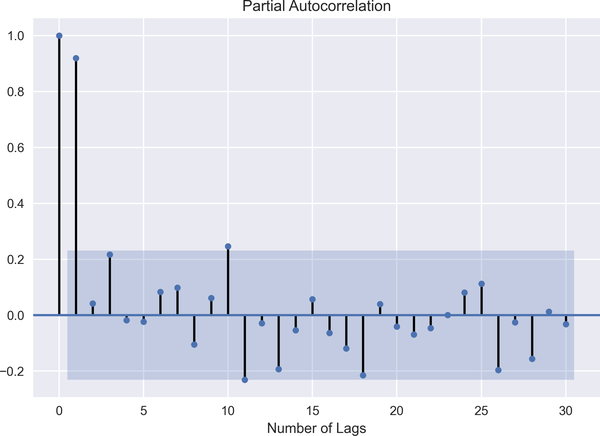


Figure 2-5. PACF plot of the S&P 500

**Seasonality**

Seasonality exists if there are regular fluctuations over a given period of time. For instance, energy usages can show a seasonality characteristic. To be more specific, energy usage goes up and down during certain periods over a year.

To show how we can detect the seasonality component, let’s use the Federal Reserve Economic Database (FRED), which includes more than 500,000 economic data series from over 80 sources covering many areas, such as banking, employment, exchange rates, gross domestic product, interest rates, trade and international transactions, and so on:

In [10]: from fredapi import Fred

import statsmodels.api as sm

In [11]: fred = Fred(api\_key='insert you api key')

In [12]: energy = fred.get\_series("CAPUTLG2211A2S",

observation\_start="2010-01-01",

observation\_end="2020-12-31") [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO5-1)

energy.head(12)

Out[12]: 2010-01-01 83.7028

2010-02-01 84.9324

2010-03-01 82.0379

2010-04-01 79.5073

2010-05-01 82.8055

2010-06-01 84.4108

2010-07-01 83.6338

2010-08-01 83.7961

2010-09-01 83.7459

2010-10-01 80.8892

2010-11-01 81.7758

2010-12-01 85.9894

dtype: float64

In [13]: plt.plot(energy)

plt.title('Energy Capacity Utilization')

plt.ylabel('$')

plt.xlabel('Date')

plt.show()

In [14]: sm.graphics.tsa.plot\_acf(energy, lags=30)

plt.xlabel('Number of Lags')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO5-1)*

Accessing the energy capacity utilization from the FRED for the period of 2010–2020

[Figure 2-6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#energy_seasonality) indicates periodic ups and downs over a nearly 10-year period with high-capacity utilization during the first months of every year and then going down toward the end of year, confirming that there is seasonality in energy-capacity utilization.

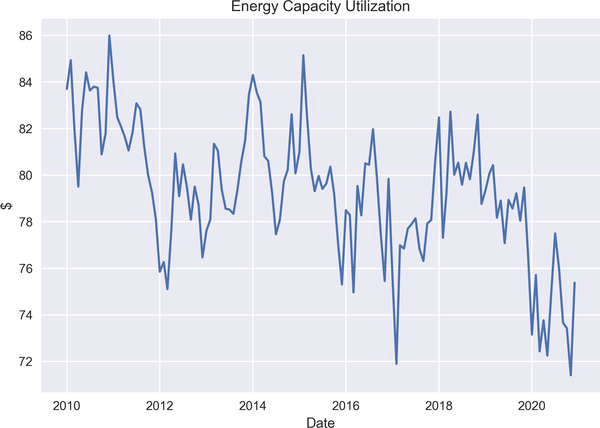


Figure 2-6. Seasonality in energy capacity utilization

An ACF plot can also provide information about the seasonality as the periodic ups and downs can be observable using ACF, too. [Figure 2-7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#energy_acf) shows the correlation structure in the presence of seasonality.

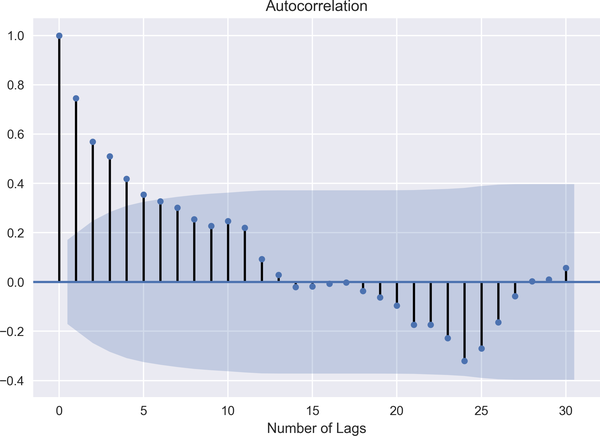


Figure 2-7. ACF of energy capacity utilization

**Cyclicality**

What if data does not show fixed period movements? At this point, cyclicality comes into the picture. It exists when higher periodic variation than the trend emerges. Some confuse cyclicality and seasonality in a sense that they both exhibit expansion and contraction. We can, however, think of cyclicality as business cycles, which take a long time to complete their cycles and the ups and downs are over a long horizon. So cyclicality is different from seasonality in the sense that there is no fluctuation in a fixed period. An example of cyclicality may be house purchases (or sales) depending on mortgage rate. That is, when a mortgage rate is cut (or raised), it leads to a boost for house purchases (or sales).

**Residual**

Residual is known as an irregular component of time series. Technically speaking, residual is equal to the difference between observations and related fitted values. We can think of it as a leftover from the model.

As we have discussed before, time series models lack some core assumptions, but this does not necessarily mean that time series models are free from assumptions. I would like to stress the most prominent one, which is called *stationarity*.

Stationarity means that statistical properties such as mean, variance, and covariance of the time series do not change over time.

There are two forms of stationarity:

*Weak stationarity*

Time series �� is said to be stationarity if:

* �� has finite variance, 𝔼(��2)<∞,∀�∈ℤ
* The mean value of �� is constant and does solely depend on time, 𝔼(��)=�,�∀∈ℤ
* Covariance structure, �(�,�+ℎ), depends on the time difference only:

�(ℎ)=�ℎ+�(�+ℎ,�)

In other words, time series should have finite variance with constant mean and a covariance structure that is a function of the time difference.

*Strong stationarity*

If the joint distribution of ��1,��2,...��� is the same with the shifted version of set ��1+ℎ,��2+ℎ,...���+ℎ, it is referred to as strong stationarity. Thus, strong stationarity implies that distribution of random variables of a random process is the same with a shifting time index.

The question is now why do we need stationarity? The reason is twofold.

First, in the estimation process, it is essential to have some distribution as time goes on. In other words, if distribution of a time series changes over time, it becomes unpredictable and cannot be modeled.

The ultimate aim of time series models is forecasting. To do that, we should estimate the coefficients first, which corresponds to learning in ML. Once we learn and conduct forecasting analysis, we assume that the distribution of the data in the estimation stays the same in a way that we have the same estimated coefficients. If this is not the case, we should reestimate the coefficients because we are unable to forecast with the previous estimated coefficients.

Having structural breaks, such as a financial crisis, generates a shift in distribution. We need to take care of this period cautiously and separately.

The other reason for having stationarity is, by assumption, some statistical models require stationary data, but that does not mean that some models requires stationary only. Instead, all models require stationarity but even if you feed the model with nonstationary data, some models, by design, turn it into stationary data and process it.

[Figure 2-4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#acf) showed the slow-decaying lags amounting to nonstationary because persistence of the high correlation between lags of the time series continues.

There are, by and large, two ways to detect nonstationarity: visualization and statistical methods. The latter, of course, is a better and more robust way of detecting the nonstationarity. However, to improve our understanding, let’s start with the ACF. Slow-decaying ACF implies that the data is nonstationary because it presents a strong correlation in time. That is what I observe in S&P 500 data.

We first need to check and see if the data is stationary or not. Visualization is a good but ultimately inadequate tool for this task. Instead, a more powerful statistical method is needed, and the augmented Dickey-Fuller (ADF) test provides this. Assuming that the confidence interval is set to 95%, the following result indicates that the data is not stationary:

In [15]: stat\_test = adfuller(SP\_prices)[0:2] [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO6-1)

print("The test statistic and p-value of ADF test are {}"

.format(stat\_test)) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO6-2)

The test statistic and p-value of ADF test are (0.030295120072926063,

0.9609669053518538)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO6-1)*

ADF test for stationarity

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO6-2)*

Test statistic and p-value of ADF test

Taking the difference is an efficient technique for removing the stationarity. This just means subtracting the current value of the series from its first lagged value, i.e., ��-��-1, and the following Python code presents how to apply this technique (and creates Figures [2-8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#detrendedSP500) and [2-9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#diff_acfSP500)):

In [16]: diff\_SP\_price = SP\_prices.diff() [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO7-1)

In [17]: plt.figure(figsize=(10, 6))

plt.plot(diff\_SP\_price)

plt.title('Differenced S&P-500 Price')

plt.ylabel('$')

plt.xlabel('Date')

plt.show()

In [18]: sm.graphics.tsa.plot\_acf(diff\_SP\_price.dropna(),lags=30)

plt.xlabel('Number of Lags')

plt.show()

In [19]: stat\_test2 = adfuller(diff\_SP\_price.dropna())[0:2] [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO7-2)

print("The test statistic and p-value of ADF test after differencing are {}"\

.format(stat\_test2))

The test statistic and p-value of ADF test after differencing are

(-7.0951058730170855, 4.3095548146405375e-10)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO7-1)*

Taking the difference of S&P 500 prices

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO7-2)*

ADF test result based on differenced S&P 500 data

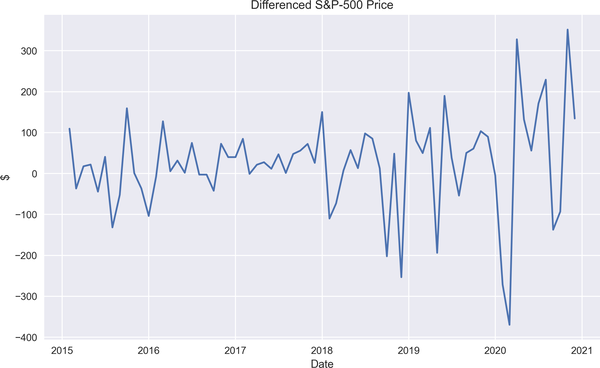


Figure 2-8. Detrended S&P 500 price

After taking the first difference, we rerun the ADF test to see if it worked, and yes, it does. The very low p-value of ADF tells me that S&P 500 data is stationary now.

This can be observed from the line plot provided in [Figure 2-8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#detrendedSP500). Unlike the raw S&P 500 plot, this plot exhibits fluctuations around the mean with similar volatility, meaning that we have a stationary series.

[Figure 2-9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#diff_acfSP500) shows that there is only one statistical significant correlation structure at lag 7.

Needless to say, trend is not the only indicator of nonstationarity. Seasonality is another source of it, and now we are about to learn a method to deal with it.

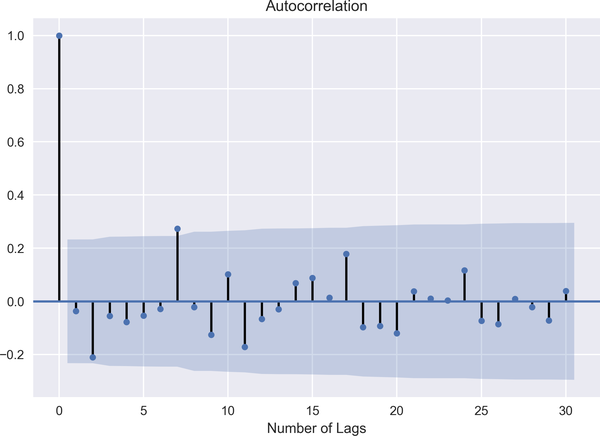


Figure 2-9. Detrended S&P 500 price

First, take a look at the ACF of energy capacity utilization in [Figure 2-7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#energy_acf), which shows periodic ups and downs, a sign of nonstationarity.

To get rid of seasonality, we first apply the *resample* method to calculate annual mean, which is used as the denominator in the following formula:

SeasonalIndex=ValueofaSeasonalTimeSeriesSeasonalAverage

Thus, the result of the application, *seasonal index*, gives us the deseasonalized time series. The following code shows us how we code this formula in Python:

In [20]: seasonal\_index = energy.resample('Q').mean() [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO8-1)

In [21]: dates = energy.index.year.unique() [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO8-2)

deseasonalized = []

for i in dates:

for j in range(1, 13):

deseasonalized.append((energy[str(i)][energy[str(i)]\

.index.month==j])) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO8-3)

concat\_deseasonalized = np.concatenate(deseasonalized) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO8-4)

In [22]: deseason\_energy = []

for i,s in zip(range(0, len(energy), 3), range(len(seasonal\_index))):

deseason\_energy.append(concat\_deseasonalized[i:i+3] /

seasonal\_index.iloc[s]) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO8-5)

concat\_deseason\_energy = np.concatenate(deseason\_energy)

deseason\_energy = pd.DataFrame(concat\_deseason\_energy,

index=energy.index)

deseason\_energy.columns = ['Deaseasonalized Energy']

deseason\_energy.head()

Out[22]: Deaseasonalized Energy

2010-01-01 1.001737

2010-02-01 1.016452

2010-03-01 0.981811

2010-04-01 0.966758

2010-05-01 1.006862

In [23]: sm.graphics.tsa.plot\_acf(deseason\_energy, lags=10)

plt.xlabel('Number of Lags')

plt.show()

In [24]: sm.graphics.tsa.plot\_pacf(deseason\_energy, lags=10)

plt.xlabel('Number of Lags')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO8-1)*

Calculating quarterly mean of energy utilization

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO8-2)*

Defining the years in which seasonality analysis is run

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO8-3)*

Computing the numerator of *Seasonal Index* formula

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO8-4)*

Concatenating the deseasonalized energy utilization

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO8-5)*

Computing *Seasonal Index* using the predefined formula

[Figure 2-10](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#deseasonalized_acf_energy) suggests that there is a statistically significant correlation at lag 1 and 2, but ACF does not show any periodic characteristics, which is another way of saying deseasonalization.

Similarly, in [Figure 2-11](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#deseasonalized_pacf_energy), although there is a spike at some lags, PACF does not show any periodic ups and downs. So we can say that the data is deseasonalized using the Seasonal Index Formula.

What we have now are the less periodic ups and down in energy-capacity utilization, meaning that the data turns out to be deseasonalized.

Finally, we are ready to move forward and discuss the time series models.

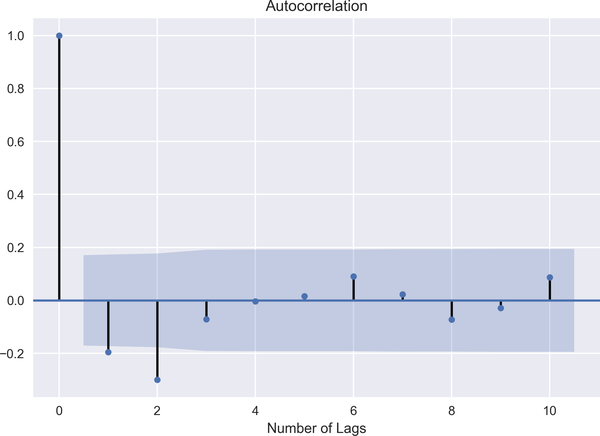


Figure 2-10. Deseasonalized ACF of energy utilization

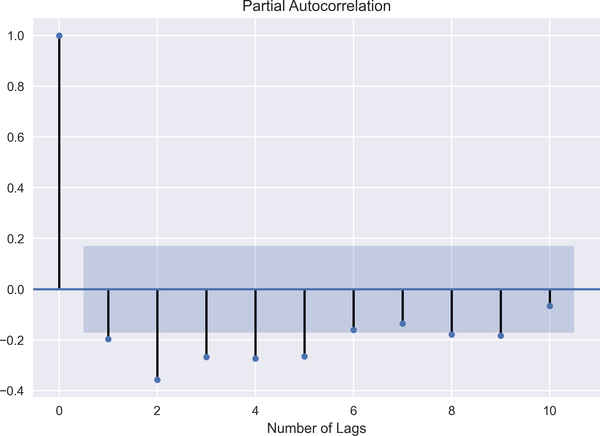


Figure 2-11. Deseasonalized PACF of energy utilization

Time Series Models

Traditional time series models are univariate models, and they follow these phases:

*Identification*

In this process, we explore the data using ACF and PACF, identifying patterns and conducting statistical tests.

*Estimation*

We estimate coefficients via the proper optimization technique.

*Diagnostics*

After estimation, we need to check if information criteria or ACF/PACF suggest that the model is valid. If so, we move on to the forecasting stage.

*Forecast*

This part is more about the performance of the model. In forecasting, we predict future values based on our estimation.

[Figure 2-12](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#modeling1) shows the modeling process. Accordingly, subsequent to identifying the variables and the estimation process, the model is run. Only after running proper diagnostics are we able to perform the forecast analysis.

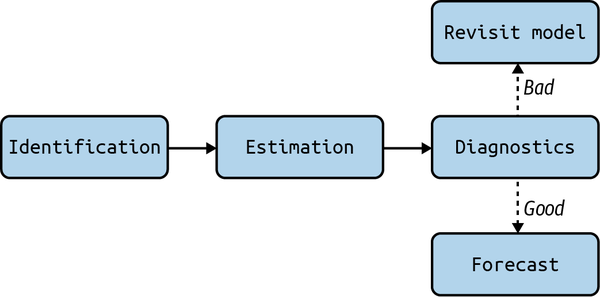


Figure 2-12. Modeling process

In modeling data with a time dimension, we should consider correlation in adjacent points in time. This consideration takes us to time series modeling. My aim in modeling time series is to fit a model and comprehend statistical character of a time series, which fluctuates randomly in time.

Recall the discussion about the IID process, which is the most basic time series model and is sometimes referred to as *white noise*. Let’s touch on the concept of white noise.

White Noise

The time series �� is said to be white noise if it satisfies the following:

��∼��(0,��2)

Corr(��,��)=0,∀�≠�

In other words, �� has mean of 0 and a constant variance. Moreover, there is no correlation between successive terms of ��. Well, it is easy to say that the white noise process is stationary and that the plot of white noise exhibits fluctuations around mean in a random fashion in time. However, as the white noise is formed by an uncorrelated sequence, it is not an appealing model from a forecasting standpoint. Uncorrelated sequences prevent us from forecasting future values.

As we can observe from the following code snippet and [Figure 2-13](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#WN), white noise oscillates around mean and is completely erratic:

In [25]: mu = 0

std = 1

WN = np.random.normal(mu, std, 1000)

plt.plot(WN)

plt.xlabel('Number of Simulations')

plt.show()

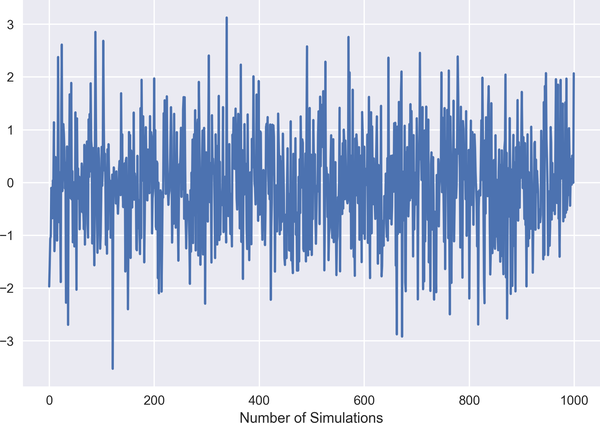


Figure 2-13. White noise process

From this point on, we need to identify the optimum number of lags before running the time series model. As you can imagine, deciding the optimal number of lags is a challenging task. The most widely used methods are ACF, PACF, and *information criteria*. ACF and PACF have already been discussed; see the following sidebar for more about information criteria, and specifically the Aikake information criterion (AIC).

**INFORMATION CRITERIA**

Determining the optimal number of lags is a cumbersome task. We need to have a criterion to decide which model fits best to the data as there may be numerous potentially good models. Cavanaugh and Neath (2019) describe the AIC as follows:

*AIC is introduced as an extension to the Maximum Likelihood Principle. Maximum likelihood is conventionally applied to estimate the parameters of a model once structure and dimension of the model have been formulated.*

AIC can be mathematically defined as:

���=-2��(�������������ℎ���)+2�

where *d* is the total number of parameters. The last term, 2*d*, aims at reducing the risk of overfitting. It is also called a *penalty term*, by which the unnecessary redundancy in the model can be filtered out.

The Bayesian information criterion (BIC) is the other information criterion used to select the best model. The penalty term in BIC is larger than that of AIC:

���=-2��(�������������ℎ���)+��(�)�

where *n* is the number of observations.

Please note that you need to treat the AIC with caution if the proposed model is finite dimensional. This fact is well put by Hurvich and Tsai (1989):

*If the true model is infinite dimensional, a case which seems most realistic in practice, AIC provides an asymptotically efficient selection of a finite dimensional approximating model. If the true model is finite dimensional, however, the asymptotically efficient methods, e.g., Akaike’s FPE (Akaike 1970), AIC, and Parzen’s CAT (Parzen 1977), do not provide consistent model order selections.*

Let’s get started visiting classical time series models with the moving average model.

**Moving Average Model**

MA and residuals are closely related models. MA can be considered a smoothing model, as it tends to take into account the lag values of residual. For the sake of simplicity, let us start with MA(1):

��=��+���-1

As long as �≠0, it has nontrivial correlation structure. Intuitively, MA(1) tells us that the time series has been affected by �� and ��-1 only.

In general form, MA(q) becomes:

��=��+�1��-1+�2��-2...+����-�

From this point on, to be consistent, we will model the data of two major tech companies, namely Apple and Microsoft. Yahoo Finance provides a convenient tool to access closing prices of the related stocks for the period between 01-01-2019 and 01-01-2021.

As a first step, we dropped the missing values and checked if the data is stationary, and it turns out neither Apple’s nor Microsoft’s stock prices have a stationary structure as expected. Thus, taking the first difference to make these data stationary and splitting the data as *train* and *test* are the steps to take at this point. The following code (which produces [Figure 2-14](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#acf_ma)) shows how we can do this in Python:

In [26]: ticker = ['AAPL', 'MSFT']

start = datetime.datetime(2019, 1, 1)

end = datetime.datetime(2021, 1, 1)

stock\_prices = yf.download(ticker, start, end, interval='1d')\

.Close [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO9-1)

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 2 of 2 completed

In [27]: stock\_prices = stock\_prices.dropna()

In [28]: for i in ticker:

stat\_test = adfuller(stock\_prices[i])[0:2]

print("The ADF test statistic and p-value of {} are {}"\

.format(i,stat\_test))

The ADF test statistic and p-value of AAPL are (0.29788764759932335,

0.9772473651259085)

The ADF test statistic and p-value of MSFT are (-0.8345360070598484,

0.8087663305296826)

In [29]: diff\_stock\_prices = stock\_prices.diff().dropna()

In [30]: split = int(len(diff\_stock\_prices['AAPL'].values) \* 0.95) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO9-2)

diff\_train\_aapl = diff\_stock\_prices['AAPL'].iloc[:split] [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO9-3)

diff\_test\_aapl = diff\_stock\_prices['AAPL'].iloc[split:] [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO9-4)

diff\_train\_msft = diff\_stock\_prices['MSFT'].iloc[:split] [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO9-5)

diff\_test\_msft = diff\_stock\_prices['MSFT'].iloc[split:] [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO9-6)

In [31]: diff\_train\_aapl.to\_csv('diff\_train\_aapl.csv') [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO9-7)

diff\_test\_aapl.to\_csv('diff\_test\_aapl.csv')

diff\_train\_msft.to\_csv('diff\_train\_msft.csv')

diff\_test\_msft.to\_csv('diff\_test\_msft.csv')

In [32]: fig, ax = plt.subplots(2, 1, figsize=(10, 6))

plt.tight\_layout()

sm.graphics.tsa.plot\_acf(diff\_train\_aapl,lags=30,

ax=ax[0], title='ACF - Apple')

sm.graphics.tsa.plot\_acf(diff\_train\_msft,lags=30,

ax=ax[1], title='ACF - Microsoft')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO9-1)*

Retrieving monthly closing stock prices

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO9-2)*

Splitting data as 95% and 5%

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO9-3)*

Assigning 95% of the Apple stock price data to the train set

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO9-4)*

Assigning 5% of the Apple stock price data to the test set

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO9-5)*

Assigning 95% of the Microsoft stock price data to the train set

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO9-6)*

Assigning 5% of the Microsoft stock price data to the test set

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO9-7)*

Saving the data for future use

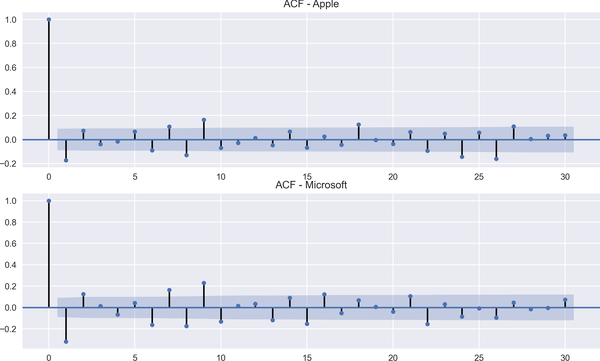


Figure 2-14. ACF after first difference

Looking at the top panel of [Figure 2-14](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#acf_ma), we can see that there are significant spikes at some lags and, therefore, we’ll choose lag 9 for the short MA model and 22 for the long MA for Apple. These imply that an order of 9 will be our short-term order and 22 will be our long-term order in modeling MA:

In [33]: short\_moving\_average\_appl = diff\_train\_aapl.rolling(window=9).mean() [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO10-1)

long\_moving\_average\_appl = diff\_train\_aapl.rolling(window=22).mean() [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO10-2)

In [34]: fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(diff\_train\_aapl.loc[start:end].index,

diff\_train\_aapl.loc[start:end],

label='Stock Price', linestyle='--') [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO10-3)

ax.plot(short\_moving\_average\_appl.loc[start:end].index,

short\_moving\_average\_appl.loc[start:end],

label = 'Short MA', linestyle='solid') [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO10-4)

ax.plot(long\_moving\_average\_appl.loc[start:end].index,

long\_moving\_average\_appl.loc[start:end],

label = 'Long MA', linestyle='solid') [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO10-5)

ax.legend(loc='best')

ax.set\_ylabel('Price in $')

ax.set\_title('Stock Prediction-Apple')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO10-1)*

Moving average with short window for Apple stock

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO10-2)*

Moving average with long window for Apple stock

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO10-3)*

Line plot of first differenced Apple stock prices

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO10-4)*

Visualization of short-window MA result for Apple

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO10-5)*

Visualization of long-window MA result for Apple

[Figure 2-15](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#ma_apple) exhibits the short-term MA model result with a solid line and the long-term MA model result with a dash-dot marker. As expected, it turns out that the short-term MA tends to be more responsive to daily changes in Apple’s stock price compared to the long-term MA. This makes sense because taking into account a long MA generates smoother predictions.

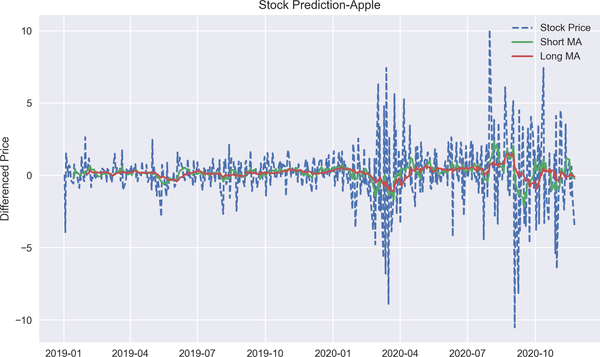


Figure 2-15. MA model prediction result for Apple

In the next step, we try to predict Microsoft’s stock price using an MA model with different window. But before proceeding, let me say that choosing the proper window for short and long MA analysis is key to good modeling. In the bottom panel of [Figure 2-14](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#acf_ma), there seem to be significant spikes at 2 and 22, so we’ll use these lags in our short and long MA analysis, respectively. After identifying the window length, we’ll fit data to the MA model with the following application:

In [35]: short\_moving\_average\_msft = diff\_train\_msft.rolling(window=2).mean()

long\_moving\_average\_msft = diff\_train\_msft.rolling(window=22).mean()

In [36]: fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(diff\_train\_msft.loc[start:end].index,

diff\_train\_msft.loc[start:end],

label='Stock Price', linestyle='--')

ax.plot(short\_moving\_average\_msft.loc[start:end].index,

short\_moving\_average\_msft.loc[start:end],

label = 'Short MA', linestyle='solid')

ax.plot(long\_moving\_average\_msft.loc[start:end].index,

long\_moving\_average\_msft.loc[start:end],

label = 'Long MA', linestyle='-.')

ax.legend(loc='best')

ax.set\_ylabel('$')

ax.set\_xlabel('Date')

ax.set\_title('Stock Prediction-Microsoft')

plt.show()

Similarly, predictions based on short MA analysis tend to be more responsive than those of the long MA model, as shown in [Figure 2-16](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#ma_msft). But in Microsoft’s case, the short-term MA prediction appears to be very close to the real data. This is something we expect in time series models in that a window with a short-term horizon is able to better capture the dynamics of the data, and this, in turn, helps us obtain better predictive performance.

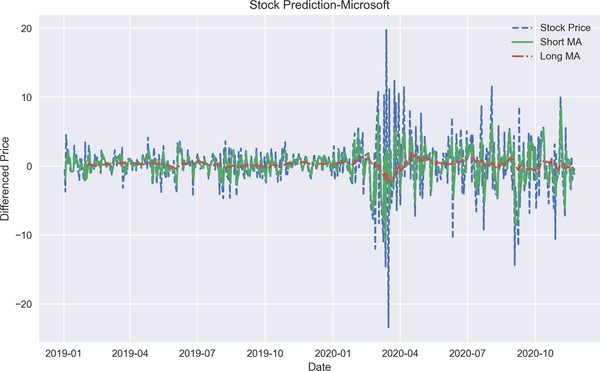


Figure 2-16. MA model prediction result for Microsoft

**Autoregressive Model**

The dependence structure of successive terms is the most distinctive feature of the AR model, in the sense that current value is regressed over its own lag values in this model. So we basically forecast the current value of the time series �� by using a linear combination of its past values. Mathematically, the general form of AR(p) can be written as:

��=�+�1��-1+�2��-2...+����-�+��

where �� denotes the residuals and *c* is the intercept term. The AR(p) model implies that past values up to order *p* have somewhat explanatory power on ��. If the relationship has shorter memory, then it is likely to model �� with a fewer number of lags.

We have discussed one of the main properties of time series, stationarity; the other important property is *invertibility*. After introducing the AR model, it is time to show the invertibility of the MA process. It is said to be invertible if it can be converted to an infinite AR model.

Under some circumstances, MA can be written as an infinite AR process. These circumstances are having stationary covariance structure, deterministic part, and invertible MA process. In doing so, we have another model called *infinite AR* thanks to the assumption of |�|<1.

��=��+���-1

=��+�(��-1-���-2)

=��+���-1-�2��-2

=��+���-1-�2(��-2+���-3)

=��+���-1-�2��-2+�3��-3)

=...

=���-1-�2��-2+�3��-3-�4��-4+...-(-�)���-�

After doing the necessary math, the equation gets the following form:

����-�=��-∑�=0�-1����-�

In this case, if |�|<1, then �→∞:

𝔼(��-∑�=0�-1����-�)2=𝔼(�2���-�2→∞)

Finally, the MA(1) process turns out to be:

��=∑�=0∞����-�

Due to the duality between the AR and MA processes, it is possible to represent AR(1) as infinite MA, MA(∞). In other words, the AR(1) process can be expressed as a function of past values of innovations:

��=��+���-1

=�(���-2+��-1)+��

=�2��-2+���-1+��

=�2(���-3+���-2)���-1+��

��=��+��-1+�2��-2+...+����

As �→∞, ��→0, so I can represent AR(1) as an infinite MA process.

In the following analysis, we run the AR model to predict Apple and Microsoft stock prices. Unlike MA, partial ACF is a useful tool to find out the optimum order in the AR model. This is because, in AR, we aim to find out the relationship of a time series between two different times, say �� and ��-�, and to do that we need to filter out the effect of other lags in between, resulting in Figures [2-17](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#pacf_appl) and [2-18](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#pacf_msft):

In [37]: sm.graphics.tsa.plot\_pacf(diff\_train\_aapl, lags=30)

plt.title('PACF of Apple')

plt.xlabel('Number of Lags')

plt.show()

In [38]: sm.graphics.tsa.plot\_pacf(diff\_train\_msft, lags=30)

plt.title('PACF of Microsoft')

plt.xlabel('Number of Lags')

plt.show()

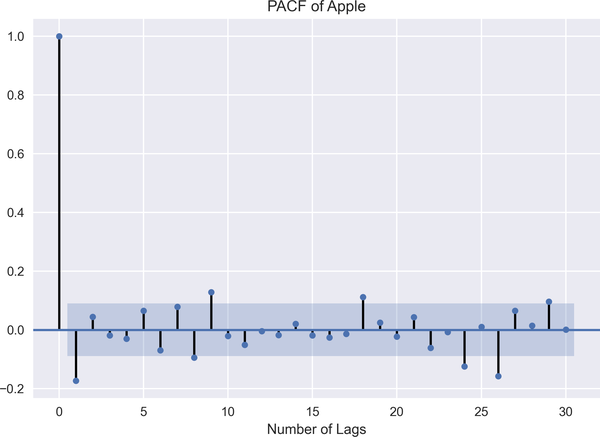


Figure 2-17. PACF for Apple

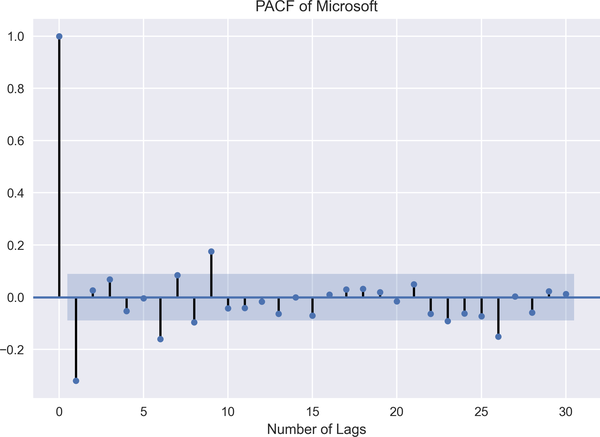


Figure 2-18. PACF for Microsoft

In [Figure 2-17](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#pacf_appl), obtained from the first differenced Apple stock price, we observe a significant spike at lag 29, and in [Figure 2-18](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#pacf_msft), we have a similar spike at lag 26 for Microsoft. Thus, 29 and 26 are the lags that we are going to use in modeling AR for Apple and Microsoft, respectively:

In [39]: from statsmodels.tsa.ar\_model import AutoReg

import warnings

warnings.filterwarnings('ignore')

In [40]: ar\_aapl = AutoReg(diff\_train\_aapl.values, lags=29)

ar\_fitted\_aapl = ar\_aapl.fit() [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-1)

In [41]: ar\_predictions\_aapl = ar\_fitted\_aapl.predict(start=len(diff\_train\_aapl),

end=len(diff\_train\_aapl)\

+ len(diff\_test\_aapl) - 1,

dynamic=False) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-2)

In [42]: for i in range(len(ar\_predictions\_aapl)):

print('==' \* 25)

print('predicted values:{:.4f} & actual values:{:.4f}'\

.format(ar\_predictions\_aapl[i], diff\_test\_aapl[i])) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-3)

==================================================

predicted values:1.6511 & actual values:1.3200

==================================================

predicted values:-0.8398 & actual values:0.8600

==================================================

predicted values:-0.9998 & actual values:0.5600

==================================================

predicted values:1.1379 & actual values:2.4600

==================================================

predicted values:-0.1123 & actual values:3.6700

==================================================

predicted values:1.7843 & actual values:0.3600

==================================================

predicted values:-0.9178 & actual values:-0.1400

==================================================

predicted values:1.7343 & actual values:-0.6900

==================================================

predicted values:-1.5103 & actual values:1.5000

==================================================

predicted values:1.8224 & actual values:0.6300

==================================================

predicted values:-1.2442 & actual values:-2.6000

==================================================

predicted values:-0.5438 & actual values:1.4600

==================================================

predicted values:-0.1075 & actual values:-0.8300

==================================================

predicted values:-0.6167 & actual values:-0.6300

==================================================

predicted values:1.3206 & actual values:6.1000

==================================================

predicted values:0.2464 & actual values:-0.0700

==================================================

predicted values:0.4489 & actual values:0.8900

==================================================

predicted values:-1.3101 & actual values:-2.0400

==================================================

predicted values:0.5863 & actual values:1.5700

==================================================

predicted values:0.2480 & actual values:3.6500

==================================================

predicted values:0.0181 & actual values:-0.9200

==================================================

predicted values:0.9913 & actual values:1.0100

==================================================

predicted values:0.2672 & actual values:4.7200

==================================================

predicted values:0.8258 & actual values:-1.8200

==================================================

predicted values:0.1502 & actual values:-1.1500

==================================================

predicted values:0.5560 & actual values:-1.0300

In [43]: ar\_predictions\_aapl = pd.DataFrame(ar\_predictions\_aapl) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-4)

ar\_predictions\_aapl.index = diff\_test\_aapl.index [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-5)

In [44]: ar\_msft = AutoReg(diff\_train\_msft.values, lags=26)

ar\_fitted\_msft = ar\_msft.fit() [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-6)

In [45]: ar\_predictions\_msft = ar\_fitted\_msft.predict(start=len(diff\_train\_msft),

end=len(diff\_train\_msft)\

+len(diff\_test\_msft) - 1,

dynamic=False) [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-7)

In [46]: ar\_predictions\_msft = pd.DataFrame(ar\_predictions\_msft) [8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-8)

ar\_predictions\_msft.index = diff\_test\_msft.index [9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO11-9)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-1)*

Fitting Apple stock data with AR model

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-2)*

Predicting the stock prices for Apple

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-3)*

Comparing the predicted and real observations

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-4)*

Turning array into dataframe to assign index

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-5)*

Assigning test data indices to predicted values

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-6)*

Fitting Microsoft stock data with AR model

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-7)*

Predicting the stock prices for Microsoft

*[8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-8)*

Turning the array into a dataframe to assign index

*[9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO11-9)*

Assigning test data indices to predicted values

The following code, resulting in [Figure 2-19](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#ar_all), shows the predictions based on the AR model. The solid lines represent the Apple and Microsoft stock price predictions, and the dashed lines denote the real data. The result reveals that the MA model outperforms the AR model in capturing the stock price:

In [47]: fig, ax = plt.subplots(2,1, figsize=(18, 15))

ax[0].plot(diff\_test\_aapl, label='Actual Stock Price', linestyle='--')

ax[0].plot(ar\_predictions\_aapl, linestyle='solid', label="Prediction")

ax[0].set\_title('Predicted Stock Price-Apple')

ax[0].legend(loc='best')

ax[1].plot(diff\_test\_msft, label='Actual Stock Price', linestyle='--')

ax[1].plot(ar\_predictions\_msft, linestyle='solid', label="Prediction")

ax[1].set\_title('Predicted Stock Price-Microsoft')

ax[1].legend(loc='best')

for ax in ax.flat:

ax.set(xlabel='Date', ylabel='$')

plt.show()

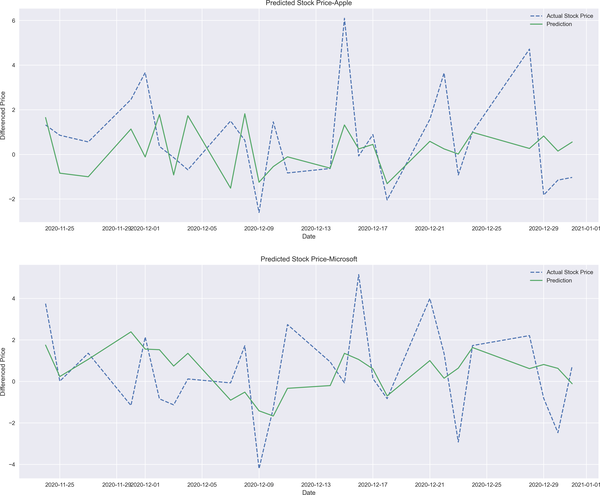


Figure 2-19. AR model prediction results

**Autoregressive Integrated Moving Average Model**

The ARIMA is a function of past values of a time series and white noise. ARIMA has been proposed as a generalization of AR and MA, but they do not have an integration parameter, which helps us to feed the model with the raw data. In this respect, even if we include nonstationary data, ARIMA makes it stationary by properly defining the integration parameter.

ARIMA has three parameters, namely *p*, *d*, and *q*. As should be familiar from previous time series models, *p* and *q* refer to the order of AR and MA, respectively. The *d* parameter controls for level difference. If *d* = 1, it amounts to first difference, and if it has a value of 0, that means that the model is ARIMA.

It is possible to have a *d* greater than 1, but it’s not as common as having a *d* of 1. The ARIMA (p, 1, q) equation has the following structure:

��=�1���-1+�2���-2...+�����-�+��+�1��-1+�2��-2...+����-�

where *d* refers to difference.

As it is a widely embraced and applicable model, let’s discuss the pros and cons of the ARIMA model to get more familiar with it.

*Pros*

* ARIMA allows us to work with raw data without considering if it is stationary.
* It performs well with high-frequency data.
* It is less sensitive to the fluctuation in the data compared to other models.

*Cons*

* ARIMA might fail in capturing seasonality.
* It works better with long series and short-term (daily, hourly) data.
* As no automatic updating occurs in ARIMA, no structural break during the analysis period should be observed.
* Having no adjustment in the ARIMA process leads to instability.

Now, let’s see how ARIMA works using the same stocks, namely Apple and Microsoft. But this time, a different short-term lag structure is used to compare the result with the AR and MA models:

In [48]: from statsmodels.tsa.arima\_model import ARIMA

In [49]: split = int(len(stock\_prices['AAPL'].values) \* 0.95)

train\_aapl = stock\_prices['AAPL'].iloc[:split]

test\_aapl = stock\_prices['AAPL'].iloc[split:]

train\_msft = stock\_prices['MSFT'].iloc[:split]

test\_msft = stock\_prices['MSFT'].iloc[split:]

In [50]: arima\_aapl = ARIMA(train\_aapl,order=(9, 1, 9)) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO12-1)

arima\_fit\_aapl = arima\_aapl.fit() [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO12-2)

In [51]: arima\_msft = ARIMA(train\_msft, order=(6, 1, 6)) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO12-3)

arima\_fit\_msft = arima\_msft.fit() [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO12-4)

In [52]: arima\_predict\_aapl = arima\_fit\_aapl.predict(start=len(train\_aapl),

end=len(train\_aapl)\

+ len(test\_aapl) - 1,

dynamic=False) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO12-5)

arima\_predict\_msft = arima\_fit\_msft.predict(start=len(train\_msft),

end=len(train\_msft)\

+ len(test\_msft) - 1,

dynamic=False) [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO12-6)

In [53]: arima\_predict\_aapl = pd.DataFrame(arima\_predict\_aapl)

arima\_predict\_aapl.index = diff\_test\_aapl.index

arima\_predict\_msft = pd.DataFrame(arima\_predict\_msft)

arima\_predict\_msft.index = diff\_test\_msft.index [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO12-7)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO12-1)*

Configuring the ARIMA model for Apple stock

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO12-2)*

Fitting the ARIMA model to Apple’s stock price

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO12-3)*

Configuring the ARIMA model for Microsoft stock

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO12-4)*

Fitting the ARIMA model to Microsoft’s stock price

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO12-5)*

Predicting the Apple stock prices based on ARIMA

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO12-6)*

Predicting the Microsoft stock prices based on ARIMA

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO12-7)*

Forming index for predictions

The next snippet, resulting in [Figure 2-20](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#arima_all), shows the result of the prediction based on Apple’s and Microsoft’s stock price, and as we employ the short-term orders from the AR and MA model, the result is not completely different:

In [54]: fig, ax = plt.subplots(2, 1, figsize=(18, 15))

ax[0].plot(diff\_test\_aapl, label='Actual Stock Price', linestyle='--')

ax[0].plot(arima\_predict\_aapl, linestyle='solid', label="Prediction")

ax[0].set\_title('Predicted Stock Price-Apple')

ax[0].legend(loc='best')

ax[1].plot(diff\_test\_msft, label='Actual Stock Price', linestyle='--')

ax[1].plot(arima\_predict\_msft, linestyle='solid', label="Prediction")

ax[1].set\_title('Predicted Stock Price-Microsoft')

ax[1].legend(loc='best')

for ax in ax.flat:

ax.set(xlabel='Date', ylabel='$')

plt.show()

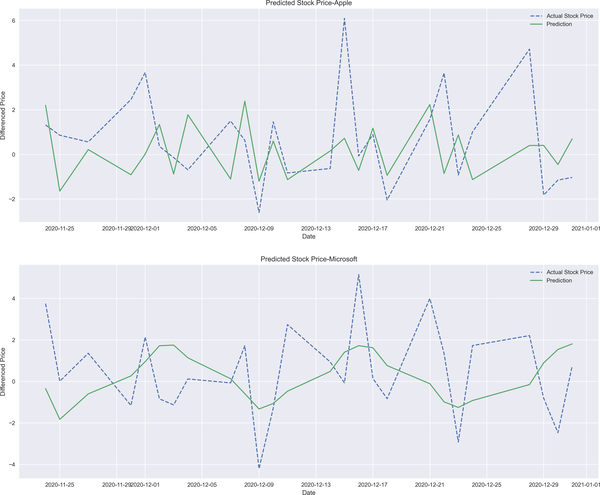


Figure 2-20. ARIMA prediction results

At this point, it is worthwhile to discuss an alternative method for optimum lag selection for time series models. AIC is the method that I apply here to select the proper number of lags. Please note that, even though the result of AIC suggests (4, 0, 4), the model does not converge with these orders. So, (4, 1, 4) is applied instead:

In [55]: import itertools

In [56]: p = q = range(0, 9) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-1)

d = range(0, 3) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-2)

pdq = list(itertools.product(p, d, q)) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-3)

arima\_results\_aapl = [] [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-4)

for param\_set in pdq:

try:

arima\_aapl = ARIMA(train\_aapl, order=param\_set) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-5)

arima\_fitted\_aapl = arima\_aapl.fit() [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-6)

arima\_results\_aapl.append(arima\_fitted\_aapl.aic) [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-7)

except:

continue

print('\*\*'\*25)

print('The Lowest AIC score is' + \

'{:.4f} and the corresponding parameters are {}'.format( \

pd.DataFrame(arima\_results\_aapl).where( \

pd.DataFrame(arima\_results\_aapl).T.notnull().all()).min()[0],

pdq[arima\_results\_aapl.index(min(arima\_results\_aapl))])) [8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-8)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The Lowest AIC score is 1951.9810 and the corresponding parameters are

(4, 0, 4)

In [57]: arima\_aapl = ARIMA(train\_aapl, order=(4, 1, 4)) [9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-9)

arima\_fit\_aapl = arima\_aapl.fit() [9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-9)

In [58]: p = q = range(0, 6)

d = range(0, 3)

pdq = list(itertools.product(p, d, q))

arima\_results\_msft = []

for param\_set in pdq:

try:

arima\_msft = ARIMA(stock\_prices['MSFT'], order=param\_set)

arima\_fitted\_msft = arima\_msft.fit()

arima\_results\_msft.append(arima\_fitted\_msft.aic)

except:

continue

print('\*\*' \* 25)

print('The lowest AIC score is {:.4f} and parameters are {}'

.format(pd.DataFrame(arima\_results\_msft)

.where(pd.DataFrame(arima\_results\_msft).T.notnull()\

.all()).min()[0],

pdq[arima\_results\_msft.index(min(arima\_results\_msft))])) [10](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-10)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The Lowest AIC score is 2640.6367 and the corresponding parameters are

(4, 2, 4)

In [59]: arima\_msft = ARIMA(stock\_prices['MSFT'], order=(4, 2 ,4)) [11](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-11)

arima\_fit\_msft= arima\_msft.fit() [11](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-11)

In [60]: arima\_predict\_aapl = arima\_fit\_aapl.predict(start=len(train\_aapl),

end=len(train\_aapl)\

+len(test\_aapl) - 1,

dynamic=False) [12](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-12)

arima\_predict\_msft = arima\_fit\_msft.predict(start=len(train\_msft),

end=len(train\_msft)\

+ len(test\_msft) - 1,

dynamic=False) [12](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#callout_introduction_to_time_series_modeling_CO13-12)

In [61]: arima\_predict\_aapl = pd.DataFrame(arima\_predict\_aapl)

arima\_predict\_aapl.index = diff\_test\_aapl.index

arima\_predict\_msft = pd.DataFrame(arima\_predict\_msft)

arima\_predict\_msft.index = diff\_test\_msft.index

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-1)*

Defining a range for AR and MA orders

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-2)*

Defining a range difference term

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-3)*

Applying iteration over *p*, *d*, and *q*

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-4)*

Creating an empty list to store AIC values

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-5)*

Configuring the ARIMA model to fit Apple data

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-6)*

Running the ARIMA model with all possible lags

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-7)*

Storing AIC values into a list

*[8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-8)*

Printing the lowest AIC value for Apple data

*[9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-9)*

Configuring and fitting the ARIMA model with optimum orders

*[10](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-11)*

Running the ARIMA model with all possible lags for Microsoft data

*[11](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-12)*

Fitting the ARIMA model to Microsoft data with optimum orders

*[12](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#co_introduction_to_time_series_modeling_CO13-14)*

Predicting Apple and Microsoft stock prices

Orders identified for Apple and Microsoft are (4, 1, 4) and (4, 2, 4), respectively. ARIMA does a good job in predicting the stock prices as shown below. However, please note that improper identification of the orders results in a poor fit, and this, in turn, produces predictions that are far from being satisfactory. The following code, resulting in [Figure 2-21](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch02.html#arima_all_2), shows these results:

In [62]: fig, ax = plt.subplots(2, 1, figsize=(18, 15))

ax[0].plot(diff\_test\_aapl, label='Actual Stock Price', linestyle='--')

ax[0].plot(arima\_predict\_aapl, linestyle='solid', label="Prediction")

ax[0].set\_title('Predicted Stock Price-Apple')

ax[0].legend(loc='best')

ax[1].plot(diff\_test\_msft, label='Actual Stock Price', linestyle='--')

ax[1].plot(arima\_predict\_msft, linestyle='solid', label="Prediction")

ax[1].set\_title('Predicted Stock Price-Microsoft')

ax[1].legend(loc='best')

for ax in ax.flat:

ax.set(xlabel='Date', ylabel='$')

plt.show()

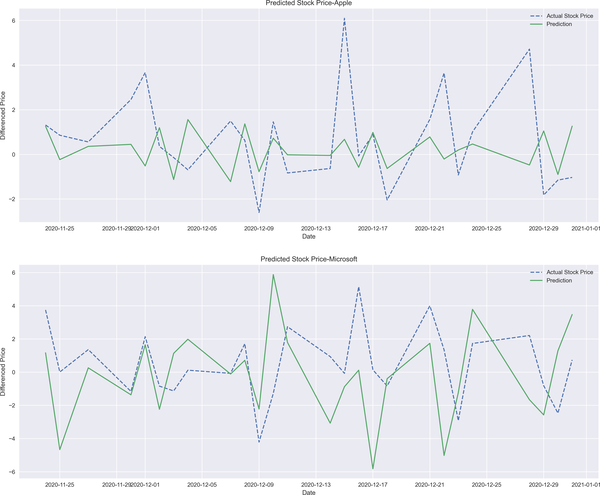


Figure 2-21. ARIMA prediction results

Conclusion

Time series analysis has a central role in financial analysis. This is simply because most financial data has a time dimension, and this type of data should be modeled cautiously. This chapter worked out a first attempt at modeling data with a time dimension, and to do so, we employed classical time series models, namely MA, AR, and finally, ARIMA. But do you think that’s the whole story? Absolutely not! In the next chapter, we will see how a time series can be modeled using deep learning models.

References

Articles cited in this chapter:

* Cavanaugh, J. E., and A. A. Neath. 2019. “The Akaike Information Criterion: Background, Derivation, Properties, Application, Interpretation, and Refinements.” *Wiley Interdisciplinary Reviews: Computational Statistics* 11 (3): e1460.
* Hurvich, Clifford M., and Chih-Ling Tsai. 1989. “Regression and Time Series Model Selection in Small Samples.” *Biometrika* 76 (2): 297-30.

Books cited in this chapter:

* Brockwell, Peter J., and Richard A. Davis. 2016. *Introduction to Time Series and Forecasting*. Springer.
* Focardi, Sergio M. 1997. *Modeling the Market: New Theories and Techniques*. The Frank J. Fabozzi Series, Vol. 14. New York: John Wiley and Sons.

Chapter 3. Deep Learning for Time Series Modeling

*...Yes, it is true that a Turing machine can compute any computable function given enough memory and enough time, but nature had to solve problems in real time. To do this, it made use of the brain’s neural networks that, like the most powerful computers on the planet, have massively parallel processors. Algorithms that run efficiently on them will eventually win out.*

 Terrence J. Sejnowski (2018)

*Deep learning* has recently become a buzzword for some good reasons, although recent attempts to improve deep learning practices are not the first of their kind. However, it is quite understandable why deep learning has been appreciated for nearly two decades. Deep learning is an abstract concept, which makes it hard to define in few of words. Unlike a neural network (NN), deep learning has a more complex structure, and hidden layers define the complexity. Therefore, some researchers use the number of hidden layers as a comparison benchmark to distinguish a neural network from deep learning, a useful but not particularly rigorous way to make this distinction. A better definition can clarify the difference.

At a high level, deep learning can be defined:

*Deep learning methods are representation-learning*[***1***](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#idm45737247845584)*methods with multiple levels of representation, obtained by composing simple but nonlinear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.*

Le Cunn et al. (2015)

Applications of deep learning date back to the 1940s, when *Cybernetics* by Norbert Wiener was published. Connectivist thinking then dominated between the 1980s and 1990s. Recent developments in deep learning, such as backpropagation and neural networks, have created the field as we know it. Basically, there have been three waves of deep learning, so we might wonder why deep learning is in its heyday *now*? Goodfellow et al. (2016) list some plausible reasons, including:

* Increasing data sizes
* Increasing model sizes
* Increasing accuracy, complexity, and real-world impact

It seems like modern technology and data availability have paved the way for an era of deep learning in which new data-driven methods are proposed so that we are able to model time series using unconventional models. This development has given rise to a new wave of deep learning. Two methods stand out in their ability to include longer time periods: the *recurrent neural network* (RNN) and *long short-term memory* (LSTM). In this section, we will concentrate on the practicality of these models in Python after briefly discussing the theoretical background.

Recurrent Neural Networks

An RNN has a neural network structure with at least one feedback connection so that the network can learn sequences. A feedback connection results in a loop, enabling us to unveil the nonlinear characteristics. This type of connection brings us a new and quite useful property: *memory*. Thus, an RNN can make use not only of the input data but also the previous outputs, which sounds compelling when it comes to time series modeling.

RNNs come in many forms, such as:

*One-to-one*

A one-to-one RNN consists of a single input and a single output, which makes it the most basic type of RNN.

*One-to-many*

In this form, an RNN produces multiple outputs for a single input.

*Many-to-one*

As opposed to the one-to-many structure, many-to-one has multiple inputs for a single output.

*Many-to-many*

This structure has multiple inputs and outputs and is known as the most complicated structure for an RNN.

A hidden unit in an RNN feeds itself back into the neural network so that the RNN has recurrent layers (unlike a feed-forward neural network) making it a suitable method for modeling time series data. Therefore, in RNNs, activation of a neuron comes from a previous time-step indication that the RNN represents as an accumulating state of the network instance (Buduma and Locascio 2017).

As summarized by Nielsen (2019):

* RNNs have time steps one at a time in an orderly fashion.
* The state of the network stays as it is from one time step to another.
* An RNN updates its state based on the time step.

These dimensions are illustrated in [Figure 3-1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#RNN_dimensions). As can be seen, the RNN structure on the right-hand side has a time step, which is the main difference between it and the feed-forward network.

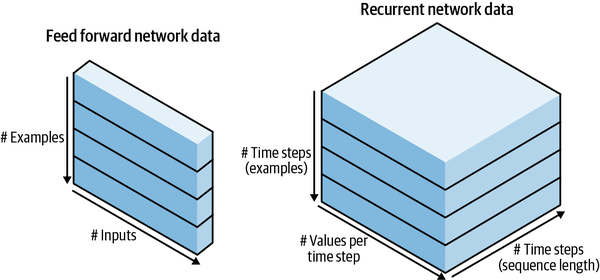


Figure 3-1. RNN structure[**2**](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#idm45737247426656)

RNNs have a three-dimensional input, comprised of:

* Batch size
* Time steps
* Number of features

*Batch size* denotes the number of observations or number of rows of data. *Time steps* are the number of times to feed the model. Finally, *number of features* is the number of columns of each sample.

We’ll start with the following code:

In [1]: import numpy as np

import pandas as pd

import math

import datetime

import yfinance as yf

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.layers import (Dense, Dropout,

Activation, Flatten,

MaxPooling2D, SimpleRNN)

from sklearn.model\_selection import train\_test\_split

In [2]: n\_steps = 13 [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-1)

n\_features = 1 [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-2)

In [3]: model = Sequential() [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-3)

model.add(SimpleRNN(512, activation='relu',

input\_shape=(n\_steps, n\_features),

return\_sequences=True)) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-4)

model.add(Dropout(0.2)) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-5)

model.add(Dense(256, activation = 'relu')) [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-6)

model.add(Flatten()) [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-7)

model.add(Dense(1, activation='linear')) [8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-8)

In [4]: model.compile(optimizer='rmsprop',

loss='mean\_squared\_error',

metrics=['mse']) [9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-9)

In [5]: def split\_sequence(sequence, n\_steps):

X, y = [], []

for i in range(len(sequence)):

end\_ix = i + n\_steps

if end\_ix > len(sequence) - 1:

break

seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]

X.append(seq\_x)

y.append(seq\_y)

return np.array(X), np.array(y) [10](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO1-10)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-1)*

Defining the number of steps for prediction

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-2)*

Defining the number of features as 1

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-3)*

Calling a sequential model to run the RNN

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-4)*

Identifying the number of hidden neurons, activation function, and input shape

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-5)*

Adding a dropout layer to prevent overfitting

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-6)*

Adding one more hidden layer with 256 neurons with the relu activation function

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-7)*

Flattening the model to transform the three-dimensional matrix into a vector

*[8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-8)*

Adding an output layer with linear activation function

*[9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-9)*

Compiling the RNN model

*[10](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO1-10)*

Creating a dependent variable y

**ACTIVATION FUNCTIONS**

Activation functions are mathematical equations that are used to determine the output in a neural network structure. These tools introduce nonlinearity in the hidden layers so that we are able to model the nonlinear issues.

The following are the most famous activation functions:

*Sigmoid*

This activation function allows us to incorporate a small amount of output as we introduce small changes in the model. It takes values between 0 and 1. The mathematical representation of sigmoid is:

sigmoid(�)=11+���(-∑�����-�)

where *w* is weight, *x* denotes data, *b* represents bias, and subscript *i* shows features.

*Tanh*

If you are handling negative numbers, tanh is your activation function. As opposed to the sigmoid function, it ranges between -1 and 1. The tanh formula is:

tanh(�)=���ℎ(�)���ℎ(�)

*Linear*

Using the linear activation function enables us to build linear relationships between independent and dependent variables. The linear activation function takes the inputs and multiplies by the weights to form the outputs proportional to the inputs. It is a convenient activation function for time-series models. Linear activation functions take the form of:

�(�)=��

*Rectified linear*

The rectified linear activation function, known as ReLu, can take 0 if the input is zero or below zero. If the input is greater than 0, it goes up in line with *x*. Mathematically:

ReLu(x)=max(0,�)

*Softmax*

Like sigmoid, this activation function is widely applicable to classification problems because softmax converts input into probabilistic distribution proportional to the exponential of the input numbers:

softmax(��)=���(��)∑����(��)

After configuring the model and generating a dependent variable, let’s extract the data and run the prediction for the stock prices for both Apple and Microsoft:

In [6]: ticker = ['AAPL', 'MSFT']

start = datetime.datetime(2019, 1, 1)

end = datetime.datetime(2020, 1 ,1)

stock\_prices = yf.download(ticker,start=start, end = end, interval='1d')\

.Close

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 2 of 2 completed

In [7]: diff\_stock\_prices = stock\_prices.diff().dropna()

In [8]: split = int(len(diff\_stock\_prices['AAPL'].values) \* 0.95)

diff\_train\_aapl = diff\_stock\_prices['AAPL'].iloc[:split]

diff\_test\_aapl = diff\_stock\_prices['AAPL'].iloc[split:]

diff\_train\_msft = diff\_stock\_prices['MSFT'].iloc[:split]

diff\_test\_msft = diff\_stock\_prices['MSFT'].iloc[split:]

In [9]: X\_aapl, y\_aapl = split\_sequence(diff\_train\_aapl, n\_steps) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-1)

X\_aapl = X\_aapl.reshape((X\_aapl.shape[0], X\_aapl.shape[1],

n\_features)) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-2)

In [10]: history = model.fit(X\_aapl, y\_aapl,

epochs=400, batch\_size=150, verbose=0,

validation\_split = 0.10) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-3)

In [11]: start = X\_aapl[X\_aapl.shape[0] - n\_steps] [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-4)

x\_input = start [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-5)

x\_input = x\_input.reshape((1, n\_steps, n\_features))

In [12]: tempList\_aapl = [] [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-6)

for i in range(len(diff\_test\_aapl)):

x\_input = x\_input.reshape((1, n\_steps, n\_features)) [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-7)

yhat = model.predict(x\_input, verbose=0) [8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-8)

x\_input = np.append(x\_input, yhat)

x\_input = x\_input[1:]

tempList\_aapl.append(yhat) [9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO2-9)

In [13]: X\_msft, y\_msft = split\_sequence(diff\_train\_msft, n\_steps)

X\_msft = X\_msft.reshape((X\_msft.shape[0], X\_msft.shape[1],

n\_features))

In [14]: history = model.fit(X\_msft, y\_msft,

epochs=400, batch\_size=150, verbose=0,

validation\_split = 0.10)

In [15]: start = X\_msft[X\_msft.shape[0] - n\_steps]

x\_input = start

x\_input = x\_input.reshape((1, n\_steps, n\_features))

In [16]: tempList\_msft = []

for i in range(len(diff\_test\_msft)):

x\_input = x\_input.reshape((1, n\_steps, n\_features))

yhat = model.predict(x\_input, verbose=0)

x\_input = np.append(x\_input, yhat)

x\_input = x\_input[1:]

tempList\_msft.append(yhat)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-1)*

Calling the split\_sequence function to define the lookback period

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-2)*

Reshaping training data into a three-dimensional case

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-3)*

Fitting the RNN model to Apple’s stock price

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-4)*

Defining the starting point of the prediction for Apple

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-5)*

Renaming the variable

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-6)*

Creating an empty list to store predictions

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-7)*

Reshaping the x\_input, which is used for prediction

*[8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-8)*

Running prediction for Apple stock

*[9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO2-9)*

Storing yhat into tempList\_aapl

For the sake of visualization, the following code block is used, resulting in [Figure 3-2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#rnn):

In [17]: fig, ax = plt.subplots(2,1, figsize=(18,15))

ax[0].plot(diff\_test\_aapl, label='Actual Stock Price', linestyle='--')

ax[0].plot(diff\_test\_aapl.index, np.array(tempList\_aapl).flatten(),

linestyle='solid', label="Prediction")

ax[0].set\_title('Predicted Stock Price-Apple')

ax[0].legend(loc='best')

ax[1].plot(diff\_test\_msft, label='Actual Stock Price', linestyle='--')

ax[1].plot(diff\_test\_msft.index,np.array(tempList\_msft).flatten(),

linestyle='solid', label="Prediction")

ax[1].set\_title('Predicted Stock Price-Microsoft')

ax[1].legend(loc='best')

for ax in ax.flat:

ax.set(xlabel='Date', ylabel='$')

plt.show()

[Figure 3-2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#rnn) shows the stock price prediction results for Apple and Microsoft. Simply eyeballing this, we can readily observe that there is room for improvement in terms of predictive performance of the model in both cases.

Even if we can have satisfactory predictive performance, the drawbacks of the RNN model should not be overlooked. The main drawbacks of the model are:

* The vanishing or exploding gradient problem (please see the following note for a detailed explanation).
* Training an RNN is a very difficult task as it requires a considerable amount of data.
* An RNN is unable to process very long sequences when the *tanh* activation function is used.

**NOTE**

A vanishing gradient is a commonplace problem in deep learning scenarios that are not properly designed. The vanishing gradient problem arises if the gradient tends to get smaller as we conduct the backpropagation. It implies that neurons are learning so slowly that optimization grinds to a halt.

Unlike the vanishing gradient problem, the exploding gradient problem occurs when small changes in the backpropagation results in huge updates to the weights during the optimization process.



Figure 3-2. RNN prediction results

The drawbacks of RNNs are well stated by Haviv et al. (2019):

*This is due to the dependency of the network on its past states, and through them on the entire input history. This ability comes with a cost—RNNs are known to be hard to train (Pascanu et al. 2013a). This difficulty is commonly associated with the vanishing gradient that appears when trying to propagate errors over long times (Hochreiter 1998). When training is successful, the network’s hidden state represents these memories. Understanding how such representation forms throughout training can open new avenues for improving learning of memory-related tasks.*

Long-Short Term Memory

The LSTM deep learning approach was developed by Hochreiter and Schmidhuber (1997) and is mainly based on the *gated recurrent unit* (GRU).

GRU was proposed to deal with the vanishing gradient problem, which is common in neural network structures and occurs when the weight update becomes too small to create a significant change in the network. GRU consists of two gates: *update* and *reset*. When an early observation is detected as highly important, then we do not update the hidden state. Similarly, when early observations are not significant, that leads to resetting the state.

As previously discussed, one of the most appealing features of an RNN is its ability to connect past and present information. However, this ability turns out to be a failure when *long-term dependencies* comes into the picture. Long-term dependencies mean that the model learns from early observations.

For instance, let’s examine the following sentence:

*Countries have their own currencies as in the USA, where people transact with dollars…*

In the case of short-term dependencies, it is known that the next predicted word is about a currency, but what if it is asked *which* currency it’s about? Things get complicated because we might have mentioned various currencies earlier on in the text, implying long-term dependencies. It is necessary to go way back to find something relevant about the countries using dollars.

LSTM tries to attack the weakness of RNN regarding long-term dependencies. LSTM has a quite useful tool to get rid of the unnecessary information so that it works more efficiently. LSTM works with gates, enabling it to forget irrelevant data. These gates are:

* Forget gates
* Input gates
* Output gates

Forget gates are created to sort out the necessary and unnecessary information so that LSTM performs more efficiently than RNN. In doing so, the value of the activation function, *sigmoid*, becomes zero if the information is irrelevant. Forget gates can be formulated as:

��=�(����+ℎ�-1��+��)

where � is the activation function, ℎ�-1 is the previous hidden state, �� and �� are weights, and finally, �� is the bias parameter in the forget cell.

Input gates are fed by the current timestep, ��, and the hidden state of the previous timestep, �-1. The goal of input gates is to determine the extent that information should be added to the long-term state. The input gate can be formulated like this:

��=�(����+ℎ�-1��+��)

Output gates basically determine the extent of the output that should be read, and work as follows:

��=�(����+ℎ�-1��+��)

These gates are not the sole components of LSTM. The other components are:

* Candidate memory cell
* Memory cell
* Hidden state

Candidate memory cell determines the extent to which information passes to the cell state. Differently, the activation function in the candidate cell is tanh and takes the following form:

��^=�(����+ℎ�-1��+��)

Memory cell allows LSTM to remember or to forget the information:

��=��⊙�+�-1+��⊙��^

where ⊙ is Hadamard product.

In this recurrent network, hidden state is a tool to circulate information. Memory cell relates output gate to hidden state:

ℎ�=�(��)⊙��

[Figure 3-3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#LSTM_structure) exhibits the LSTM structure.

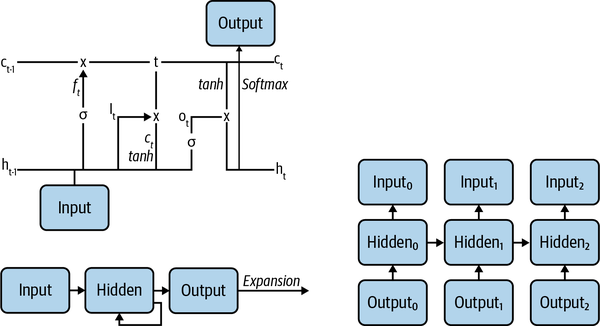


Figure 3-3. LSTM structure

Now, let’s predict the stock prices using LSTM:

In [18]: from tensorflow.keras.layers import LSTM

In [19]: n\_steps = 13 [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-1)

n\_features = 1 [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-2)

In [20]: model = Sequential()

model.add(LSTM(512, activation='relu',

input\_shape=(n\_steps, n\_features),

return\_sequences=True)) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-3)

model.add(Dropout(0.2)) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-4)

model.add(LSTM(256,activation='relu')) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-5)

model.add(Flatten())[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-6)

model.add(Dense(1, activation='linear')) [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-7)

In [21]: model.compile(optimizer='rmsprop', loss='mean\_squared\_error',

metrics=['mse']) [8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-8)

In [22]: history = model.fit(X\_aapl, y\_aapl,

epochs=400, batch\_size=150, verbose=0,

validation\_split = 0.10) [9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#callout_deep_learning_for_time_series_modeling_CO3-9)

In [23]: start = X\_aapl[X\_aapl.shape[0] - 13]

x\_input = start

x\_input = x\_input.reshape((1, n\_steps, n\_features))

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-1)*

Defining the number of steps for prediction

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-2)*

Defining the number of feature as 1

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-3)*

Identifying the number of hidden neurons, the activation function, which is relu, and input shape

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-4)*

Adding a dropout layer to prevent overfitting

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-5)*

Adding one more hidden layer with 256 neurons, with a relu activation function

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-6)*

Flattening the model to vectorize the three-dimensional matrix

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-7)*

Adding an output layer with a linear activation function

*[8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-8)*

Compiling LSTM with Root Mean Square Propagation, rmsprop, and mean squared error (MSE), mean\_squared\_error

*[9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#co_deep_learning_for_time_series_modeling_CO3-9)*

Fitting the LSTM model to Apple’s stock price

**NOTE**

Root Mean Square Propagation (RMSProp) is an optimization method in which we calculate the moving average of the squared gradients for each weight. We then find the difference of weight, which is to be used to compute the new weight:

��=���-1+1-���2

���=-��+���

��+1=��+���

Pursuing the same procedure and given the Microsoft stock price, a prediction analysis is carried out:

In [24]: tempList\_aapl = []

for i in range(len(diff\_test\_aapl)):

x\_input = x\_input.reshape((1, n\_steps, n\_features))

yhat = model.predict(x\_input, verbose=0)

x\_input = np.append(x\_input, yhat)

x\_input = x\_input[1:]

tempList\_aapl.append(yhat)

In [25]: history = model.fit(X\_msft, y\_msft,

epochs=400, batch\_size=150, verbose=0,

validation\_split = 0.10)

In [26]: start = X\_msft[X\_msft.shape[0] - 13]

x\_input = start

x\_input = x\_input.reshape((1, n\_steps, n\_features))

In [27]: tempList\_msft = []

for i in range(len(diff\_test\_msft)):

x\_input = x\_input.reshape((1, n\_steps, n\_features))

yhat = model.predict(x\_input, verbose=0)

x\_input = np.append(x\_input, yhat)

x\_input = x\_input[1:]

tempList\_msft.append(yhat)

The following code creates the plot ([Figure 3-4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#LSTM)) that shows the prediction results:

In [28]: fig, ax = plt.subplots(2, 1, figsize=(18, 15))

ax[0].plot(diff\_test\_aapl, label='Actual Stock Price', linestyle='--')

ax[0].plot(diff\_test\_aapl.index, np.array(tempList\_aapl).flatten(),

linestyle='solid', label="Prediction")

ax[0].set\_title('Predicted Stock Price-Apple')

ax[0].legend(loc='best')

ax[1].plot(diff\_test\_msft, label='Actual Stock Price', linestyle='--')

ax[1].plot(diff\_test\_msft.index, np.array(tempList\_msft).flatten(),

linestyle='solid', label="Prediction")

ax[1].set\_title('Predicted Stock Price-Microsoft')

ax[1].legend(loc='best')

for ax in ax.flat:

ax.set(xlabel='Date', ylabel='$')

plt.show()

LSTM seems to outperform the RNN, particularly in the way it captures the extreme values better.



Figure 3-4. LSTM prediction results

Conclusion

This chapter was about predicting stock prices based on deep learning. The models used are RNN and LSTM, which have the ability to process longer time periods. These models do not suggest remarkable improvement but still can be employed to model time series data. LSTM considers, in our case, a 13-step lookback period for prediction. For an extension, it would be a wise approach to include multiple features in the models based on deep learning, which is not allowed in parametric time series models.

In the next chapter, we will discuss volatility predictions based on parametric and ML models so that we can compare their performance.

References

Articles cited in this chapter:

* Ding, Daizong, et al. 2019. “Modeling Extreme Events in Time Series Prediction.” *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1114-1122.
* Haviv, Doron, Alexander Rivkind, and Omri Barak. 2019. “Understanding and Controlling Memory in Recurrent Neural Networks.” arXiv preprint. arXiv:1902.07275.
* Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. “Long Short-term Memory.” *Neural Computation* 9 (8): 1735-1780.
* LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. “Deep Learning.” *Nature* 521, (7553): 436-444.

Books cited in this chapter:

* Buduma, N., and N. Locascio. 2017. *Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms*. Sebastopol: O’Reilly.
* Goodfellow, I., Y. Bengio, and A. Courville. 2016. *Deep Learning*. Cambridge, MA: MIT Press.
* Nielsen, A. 2019. *Practical Time Series Analysis: Prediction with Statistics and Machine Learning*. Sebastopol: O’Reilly.
* Patterson, Josh, and Adam Gibson. 2017. *Deep Learning: A Practitioner’S Approach*. Sebastopol: O’Reilly.
* Sejnowski, Terrence J. 2018. *The Deep Learning Revolution*. Cambridge, MA: MIT Press.

[**1**](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#idm45737247845584-marker) Representation learning helps us define a concept in a unique way. For instance, if the task is to detect whether something is a circle, then edges play a key role, as a circle has no edge. So using color, shape, and size, we can create a representation for an object. In essence, this is how the human brain works, and we know that deep learning structures are inspired by the brain’s functioning.

[**2**](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch03.html#idm45737247426656-marker) Patterson et. al, 2017. “Deep learning: A practitioner’s approach.”

# Part II. Machine Learning for Market, Credit, Liquidity, and Operational Risks

Chapter 6. Credit Risk Estimation

*Although market risk is much better researched, the larger part of banks’ economic capital is generally used for credit risk. The sophistication of traditional standard methods of measurement, analysis, and management of credit risk might, therefore, not be in line with its significance.*

 Uwe Wehrspohn (2002)

The primary role of financial institutions is to create a channel by which funds move from entities with surplus into ones with deficit. Thereby, financial institutions ensure the capital allocation in the financial system as well as gain profit in exchange for these transactions.

However, there is an important risk for financial institutions to handle, which is credit risk. This is such a big risk that without it capital allocation might be less costly and more efficient. *Credit risk* is the risk that arises when a borrower is not able to honor their debt. In other words, when a borrower defaults, they fail to pay back their debt, which causes losses for financial institutions.

Credit risk and its goal can be defined in a more formal way (BCBS and BIS 2000):

*Credit risk is most simply defined as the potential that a bank borrower or counterparty will fail to meet its obligations in accordance with agreed terms. The goal of credit risk management is to maximise a bank’s risk-adjusted rate of return by maintaining credit risk exposure within acceptable parameters.*

Estimating credit risk is so formidable a task that a regulatory body, Basel, closely monitors recent developments in the financial markets and sets regulations to strengthen bank capital requirements. The importance of having strong capital requirements for a bank rests on the idea that banks should have a capital buffer in turbulent times.

There is a consensus among policy makers that financial institutions should have a minimum capital requirement to ensure the stability of the financial system because a series of defaults may result in a collapse in financial markets, as financial institutions provide collateral to one another. Those looking for a workaround for this capital requirement learned their lessons the hard way during the [2007—2008 mortgage crisis](https://oreil.ly/OjDw9).

Of course, ensuring at least a minimum capital requirement is a burden for financial institutions in the sense that capital is an asset they cannot channel to deficit entities to make a profit. Consequently, managing credit risk amounts to profitable and efficient transactions.

In this respect, this chapter shows how credit risk can be estimated using cutting-edge ML models. We start our discussion with a theoretical background of credit risk. Needless to say, there are many topics in credit risk analysis, but we confine our focus on probability of default and how we can introduce ML approaches for estimating it. For this purpose, customers are segmented via a clustering method so that models can be separately fitted to this data. This provides a better fit in the sense that the distribution of credit risk data changes across different customer segments. Given the clusters obtained, ML and deep learning models, including the Bayesian approach, are introduced to model the credit risk.

Estimating the Credit Risk

Aside from the probability of default (which is the likelihood that a borrower fails to cover their debt), credit risk has three defining characteristics:

*Exposure*

This refers to a party that may possibly default or suffer an adverse change in its ability to perform.

*Likelihood*

The likelihood that this party will default on its obligations.

*Recovery rate*

How much can be retrieved if a default takes place.

The BCBS put forth the global financial credit management standards, which are known as the *Basel Accord*. There are currently three Basel Accords. The most distinctive rule set by Basel I in 1988 was the requirement to hold capital equating to at least 8% of risk-weighted assets.

Basel I includes the very first capital measurement system, which was created following the onset of the [Latin American debt crisis](https://oreil.ly/KI5vs). In Basel I, assets are classified as follows:

* 0% for risk-free assets
* 20% for loans to other banks
* 50% for residential mortgages
* 100% for corporate debt

In 1999, Basel II issued a revision to Basel I based on three main pillars:

* Minimum capital requirements, which sought to develop and expand the standardized rules set out in the 1988 Accord
* Supervisory review of an institution’s capital adequacy and internal assessment process
* Effective use of disclosure as a lever to strengthen market discipline and encourage sound banking practices

The last accord, Basel III in 2010, was inevitable. as the 2007–2008 mortgage crisis heightened. It introduced a new set of measures to further strengthened liquidity and poor governance practices. For instance, equity requirements were introduced to prevent a serial failure in the financial system, known as *domino effect*, during times of financial turbulence and crises. Accordingly, Basel III requires the financial ratios for banks listed in [Table 6-1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#table6-1).

| **Financial ratio** | **Formula** |
| --- | --- |
| Tier 1 capital ratio | EquitycapitalRiskweightedassets>=4.5% |
| Leverage ratio | Tier1capitalAveragetotalassets>=3% |
| Liquidity coverage ratio | StockofhighqualityliquidassetsTotalnetcashoutflowsoverthenext30calendardays>=100% |
| Table 6-1. Financial ratios required by Basel III | |

Basel II suggests banks implement either a standardized approach or an internal ratings–based (IRB) approach to estimate the credit risk. The standardized approach is out of the scope of this book, but interested readers can refer to the “Standardized Approach to Credit Risk” [consultative document from the BIS](https://oreil.ly/0Mj7J).

Let’s now focus on the IRB approach; the key parameters of this internal assessment are:

Expectedloss=EAD×LGD×PD

where *PD* is the probability of default, *LGD* is the expected loss given default taking a value between 0 and 1, and *EAD* is the exposure at default.

The most important and challenging part of estimating credit risk is to model the probability of default, and the aim of this chapter is mainly to come up with an ML model to address this issue. Before moving forward, there is one more important issue in estimating credit risk that is sometimes neglected or overlooked: *risk* *bucketing*.

Risk Bucketing

Risk bucketing is nothing but grouping borrowers with similar creditworthiness. The behind-the-scenes story of risk bucketing is to obtain homogenous groups or clusters so that we can better estimate the credit risk. Treating different risky borrowers equally may result in poor predictions because the model cannot capture entirely different characteristics of the data at once. Thus, by dividing the borrowers into different groups based on riskiness, risk bucketing enables us to make accurate predictions.

Risk bucketing can be accomplished via different statistical methods, but we will apply a clustering technique to end up with homogeneous clusters using K-means.

We live in the age of data, but that does not necessarily mean that we always find the data we are searching for. Rather, it is rare to find it without applying data-wrangling and cleaning techniques.

Data with dependent variables is, of course, easy to work with and also helps us get more accurate results. However, sometimes we need to unveil the hidden characteristics of the data—that is, if the riskiness of the borrowers is not known, we are supposed to come up with a solution for grouping them based on their riskiness.

Clustering is the method proposed to create these groups or *clusters*. Optimal clustering has clusters located far away from one another spatially:

*Clustering groups data instances into subsets in such a manner that similar instances are grouped together, while different instances belong to different groups. The instances are thereby organized into an efficient representation that characterizes the population being sampled.*

 Rokach and Maimon (2005)

Different clustering methods are available, but the K-means algorithm serves our purpose, which is to create risk bucketing for credit risk analysis. In K-means, the distance of observations within the cluster is calculated based on the cluster center, the *centroid*. Depending on the distance to the centroid, observations are clustered. This distance can be measured via different methods. Of them, the following are the most well-known metrics:

*Euclidean*

∑�=1�(��-��)2

*Minkowski*

(∑�=1�|��-��|�)1/�

*Manhattan*

∑�=1�|��-��|

The aim in clustering is to minimize the distance between the centroid and observations so that similar observations will be on the same cluster. This logic rests on the intuition that the more similar observations are, the smaller the distance between them. So we are seeking to minimize the distance between observations and the centroid, which is another way of saying that we are minimizing the sum of the squared error between the centroid and the observations:

∑�=1�∑�∈��(��-�)2

where *x* is observation and �� is the centroid of ��ℎ cluster. However, considering the number of observations and the combinations of clusters, the search area might be too big to handle. It may sound intimidating, but don’t worry: we have the *expectation-maximization* *(E-M)* algorithm behind our clustering. As K-means does not have a closed-form solution, we are searching for an approximate one, and E-M provides this solution. In the E-M algorithm, *E* refers to assigning observations to the nearest centroid, and *M* denotes completion of the data generation process by updating the parameters.

In the E-M algorithm, the distances between observations and the centroid is iteratively minimized. The algorithm works as follows:

1. Pick *k* random points to be centroids.
2. Based on the distance metric chosen, calculate the distances between observations and *n* centroids. Based on these distances, assign each observation to the closest cluster.
3. Update cluster centers based on the assignment.
4. Repeat the process from step 2 until the centroid does not change.

Now, we apply risk bucketing using K-means clustering. To decide the optimal number of clusters, different techniques will be employed. First, we use the *elbow method*, which is based on the *inertia*.

Inertia is computed as the sum of the squared distances of observations to their closest centroid. Second, the *Silhouette score* is introduced as a tool to decide the optimal number of clusters. This takes a value between 1 and -1. A value of 1 indicates that an observation is close to the correct centroid and correctly classified. However, -1 shows that an observation is not correctly clustered. The strength of the Silhouette score rests on taking into account both the intracluster distance and the intercluster distance. The formula for Silhouette score is as follows:

Silhouettescore=�-�max(�,�)

where *x* is the average intercluster distance between clusters, and *y* is the mean intracluster distance.

The third method is *Calinski-Harabasz* *(CH)*, which is known as the *variance ratio criterion*. The formula for the CH method is as follows:

CH=������×�-��-1

where ��� denotes between-cluster variance, ��� represents within cluster variance, *N* is number of observations, and *k* is the number of clusters. Given this information, we are seeking a high CH score, as the larger (lower) the between-cluster variance (within cluster variance), the better it is for finding the optimal number of clusters.

The final approach is *gap analysis*. Tibshirani et al. (2001) came up with a unique idea by which we are able to find the optimal number of clusters based on reference distribution. Following the similar notations of Tibshirani et al., let ���� be a Euclidean distance between ��� and ���� and let �� be the ��ℎ cluster denoting the number of observations in cluster *r*:

∑�(���-����)2

The sum of pairwise distances for all observations in cluster *r* is:

��=∑�,��∈����,��

The within-cluster sum of squares, ��, is:

��=∑�=1�12����

where *n* is the sample size and expectation of �� is:

��=���(��/12)-(2/�)���(�)+��������

where *p* and *k* are dimension and centroids, respectively. Let’s create a practice exercise using German credit risk data. The data is gathered from the [Kaggle platform](https://oreil.ly/4NgIy), and the explanations of the variables are shown here:

* Age: Numerical
* Sex: Male, female
* Job: 0—unskilled and non-resident, 1—unskilled and resident, 2—skilled, 3—highly skilled
* Housing: Own, rent, free
* Saving accounts: Little, moderate, quite rich, rich
* Checking account: Numerical
* Credit amount: Numerical
* Duration: Numerical
* Purpose: Car, furniture/equipment, radio/TV, domestic appliances, repairs, education, business, vacation/others

The estimate of the optimal clusters will be the value that maximizes the gap statistic, as the gap statistic is the difference between the total within-intracluster variation for different values of *k* and their expected values under null reference distribution of the respective data. The decision is made when we get the highest gap value.

In the following code block, we import the German credit dataset and drop the unnecessary columns. The dataset includes both categorical and numerical values, which need to be treated differently, and we will do this soon:

In [1]: import pandas as pd

In [2]: credit = pd.read\_csv('credit\_data\_risk.csv')

In [3]: credit.head()

Out[3]: Unnamed: 0 Age Sex Job Housing Saving accounts Checking account \

0 0 67 male 2 own NaN little

1 1 22 female 2 own little moderate

2 2 49 male 1 own little NaN

3 3 45 male 2 free little little

4 4 53 male 2 free little little

Credit amount Duration Purpose Risk

0 1169 6 radio/TV good

1 5951 48 radio/TV bad

2 2096 12 education good

3 7882 42 furniture/equipment good

4 4870 24 car bad

In [4]: del credit['Unnamed: 0'] [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO1-1)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO1-1)*

Dropping unnecessary column named Unnamed: 0

The summary statistics are given in the following code. Accordingly, the average age of the customers is roughly 35, average job type is skilled, average credit amount and duration are nearly 3,271 and 21, respectively. Additionally, the summary statistics tell us that the credit amount variable shows a relatively high standard deviation as expected. The duration and age variables have a very similar standard deviation, but the duration moves within a narrower interval as its minimum and maximum values are 4 and 72, respectively. As job is a discrete variable, it is natural to expect low dispersion and we have it:

In [5]: credit.describe()

Out[5]: Age Job Credit amount Duration

count 1000.000000 1000.000000 1000.000000 1000.000000

mean 35.546000 1.904000 3271.258000 20.903000

std 11.375469 0.653614 2822.736876 12.058814

min 19.000000 0.000000 250.000000 4.000000

25% 27.000000 2.000000 1365.500000 12.000000

50% 33.000000 2.000000 2319.500000 18.000000

75% 42.000000 2.000000 3972.250000 24.000000

max 75.000000 3.000000 18424.000000 72.000000

In what follows, the distribution of the numerical variables in the dataset are examined via histogram and it turns out none of the variables follow a normal distribution. The age, credit amount, and duration variables are positively skewed as we can see in [Figure 6-1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#credit_risk_hist), generated by the following:

In [6]: import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

plt.rcParams["figure.figsize"] = (10,6) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO2-1)

In [7]: numerical\_credit = credit.select\_dtypes(exclude='O') [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO2-2)

In [8]: plt.figure(figsize=(10, 8))

k = 0

cols = numerical\_credit.columns

for i, j in zip(range(len(cols)), cols):

k +=1

plt.subplot(2, 2, k)

plt.hist(numerical\_credit.iloc[:, i])

plt.title(j)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO2-1)*

Setting a fix figure size

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO2-2)*

Dropping the object type variables to obtain all numerical variables

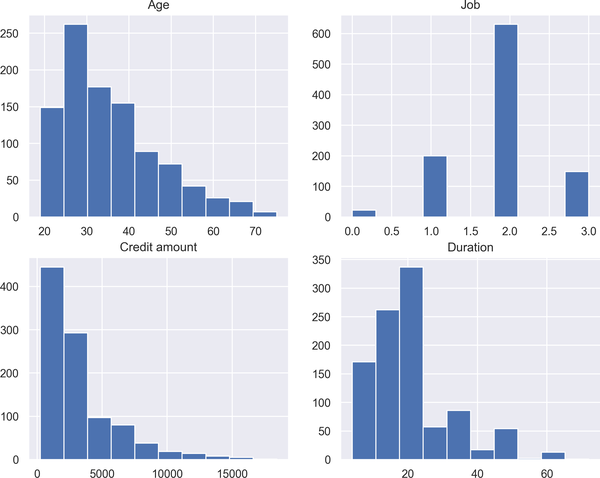


Figure 6-1. Credit risk data histogram

[Figure 6-1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html" \l "credit_risk_hist) shows the distribution of age, job, credit amount, and duration variables. Aside from the job variable, which is a discrete variable, all other variables have skewed distributions.

The elbow method, as a first method, is introduced in the following code snippet and the resulting [Figure 6-2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#elbow_kmeans). To find the optimal number of clusters, we observe the slope of the curve and decide the cut-off point at which the curve gets flatter—that is, the slope of the curve gets lower. As it gets flatter, the inertia, telling us how far away the points within a cluster are located, decreases, which is nice for the purpose of clustering. On the other hand, as we allow inertia to decrease, the number of clusters increases, which makes the analysis more complicated. Given that trade-off, the stopping criteria is the point where the curve gets flatter. In code:

In [9]: from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

import numpy as np

In [10]: scaler = StandardScaler()

scaled\_credit = scaler.fit\_transform(numerical\_credit) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO3-1)

In [11]: distance = []

for k in range(1, 10):

kmeans = KMeans(n\_clusters=k) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO3-2)

kmeans.fit(scaled\_credit)

distance.append(kmeans.inertia\_) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO3-3)

In [12]: plt.plot(range(1, 10), distance, 'bx-')

plt.xlabel('k')

plt.ylabel('Inertia')

plt.title('The Elbow Method')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO3-1)*

Applying standardization for scaling purpose

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO3-2)*

Running K-means algorithm

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO3-3)*

Calculating inertia and storing into a list named distance

[Figure 6-2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html" \l "elbow_kmeans) shows that the curve gets flatter after four clusters. Thus, the elbow method suggests that we stop at four clusters.

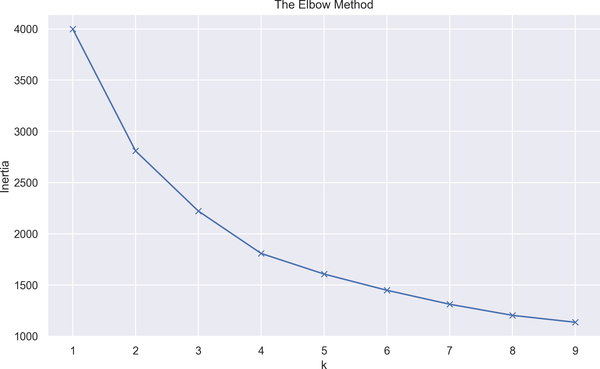


Figure 6-2. Elbow method

The following code, resulting in [Figure 6-3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#silhouette_kmeans), presents Silhouette scores on the x-axis for clusters 2 to 10. Given the average Silhouette score represented by the dashed line, the optimal number of clusters can be two:

In [13]: from sklearn.metrics import silhouette\_score [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO4-1)

from yellowbrick.cluster import SilhouetteVisualizer [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO4-2)

In [14]: fig, ax = plt.subplots(4, 2, figsize=(25, 20))

for i in range(2, 10):

km = KMeans(n\_clusters=i)

q, r = divmod(i, 2) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO4-3)

visualizer = SilhouetteVisualizer(km, colors='yellowbrick',

ax=ax[q - 1][r]) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO4-4)

visualizer.fit(scaled\_credit)

ax[q - 1][r].set\_title("For Cluster\_"+str(i))

ax[q - 1][r].set\_xlabel("Silhouette Score")

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO4-1)*

Importing the silhouette\_score module to calculate Silhouette score

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO4-2)*

Importing the SilhouetteVisualizer module to draw Silhouette plots

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO4-3)*

Using divmod for configuring labels, as it returns the quotient (q) and remainder (r)

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO4-4)*

Plotting the Silhouette scores

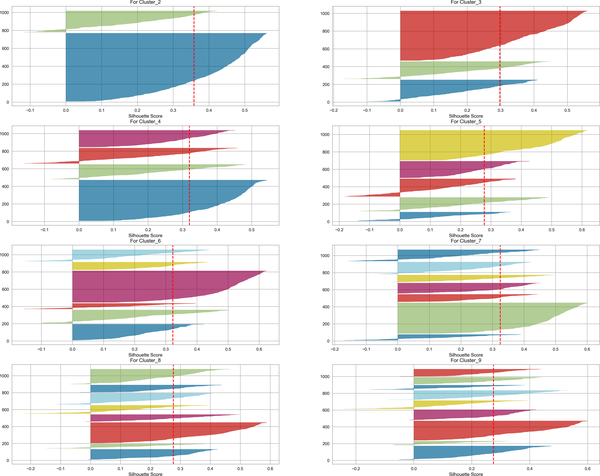


Figure 6-3. Silhouette score

As mentioned, the CH method is a convenient tool for finding optimal clustering, and the following code shows how we can use this method in Python, resulting in [Figure 6-4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#CH_analysis). We are looking for the highest CH score, and we’ll see that it is obtained at cluster 2:

In [15]: from yellowbrick.cluster import KElbowVisualizer [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO5-1)

model = KMeans()

visualizer = KElbowVisualizer(model, k=(2, 10),

metric='calinski\_harabasz',

timings=False) [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO5-2)

visualizer.fit(scaled\_credit)

visualizer.show()

Out[]: <Figure size 576x396 with 0 Axes>

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO5-1)*

Importing KElbowVisualizer to draw the CH score

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO5-2)*

Visualizing the CH metric

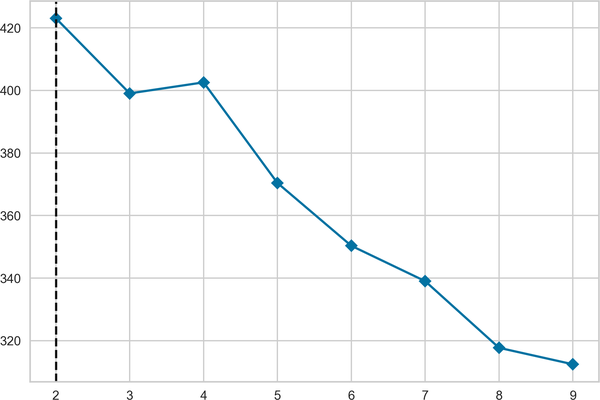


Figure 6-4. The CH method

[Figure 6-4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html" \l "CH_analysis) shows that the elbow occurs at the second cluster, indicating that stopping at two clusters is the optimum decision.

The last step for finding the optimal number of clusters is gap analysis, resulting in [Figure 6-5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#gap_cluster):

In [16]: from gap\_statistic.optimalK import OptimalK [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO6-1)

In [17]: optimalK = OptimalK(n\_jobs=8, parallel\_backend='joblib') [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO6-2)

n\_clusters = optimalK(scaled\_credit, cluster\_array=np.arange(1, 10)) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO6-3)

In [18]: gap\_result = optimalK.gap\_df [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO6-4)

gap\_result.head()

Out[18]: n\_clusters gap\_value gap\* ref\_dispersion\_std sk \

0 1.0 0.889755 5738.286952 54.033596 0.006408

1 2.0 0.968585 4599.736451 366.047394 0.056195

2 3.0 1.003974 3851.032471 65.026259 0.012381

3 4.0 1.044347 3555.819296 147.396138 0.031187

4 5.0 1.116450 3305.617917 27.894622 0.006559

sk\* diff diff\*

0 6626.296782 -0.022635 6466.660374

1 5328.109873 -0.023008 5196.127130

2 4447.423150 -0.009186 4404.645656

3 4109.432481 -0.065543 4067.336067

4 3817.134689 0.141622 3729.880829

In [19]: plt.plot(gap\_result.n\_clusters, gap\_result.gap\_value)

min\_ylim, max\_ylim = plt.ylim()

plt.axhline(np.max(gap\_result.gap\_value), color='r',

linestyle='dashed', linewidth=2)

plt.title('Gap Analysis')

plt.xlabel('Number of Cluster')

plt.ylabel('Gap Value')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO6-1)*

Importing the OptimalK module for calculating the gap statistic

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO6-2)*

Running gap statistic using parallelization

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO6-3)*

Identifying the number of clusters based on the gap statistic

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO6-4)*

Storing the result of gap analysis

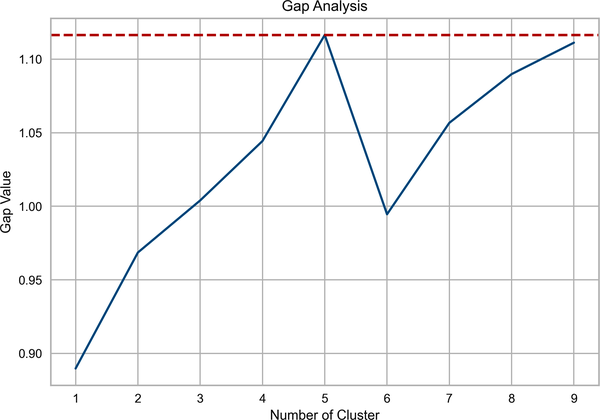


Figure 6-5. Gap analysis

What we observe in [Figure 6-5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#gap_cluster) is a sharp increase to the point at which the gap value reaches its peak, and the analysis suggests stopping at the maximum value at which we find the optimal number for clustering. In this case, we find the value at cluster 5, so this is the cut-off point.

In light of these discussions, two clusters are chosen to be the optimal number of clusters, and the K-means clustering analysis is conducted accordingly. To illustrate, given the clustering analysis, let us visualize 2-D clusters with the following, resulting in [Figure 6-6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#all_clusters):

In [20]: kmeans = KMeans(n\_clusters=2)

clusters = kmeans.fit\_predict(scaled\_credit)

In [21]: plt.figure(figsize=(10, 12))

plt.subplot(311)

plt.scatter(scaled\_credit[:, 0], scaled\_credit[:, 2],

c=kmeans.labels\_, cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 0],

kmeans.cluster\_centers\_[:, 2], s = 80,

marker= 'x', color = 'k')

plt.title('Age vs Credit')

plt.subplot(312)

plt.scatter(scaled\_credit[:, 0], scaled\_credit[:, 2],

c=kmeans.labels\_, cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 0],

kmeans.cluster\_centers\_[:, 2], s = 80,

marker= 'x', color = 'k')

plt.title('Credit vs Duration')

plt.subplot(313)

plt.scatter(scaled\_credit[:, 2], scaled\_credit[:, 3],

c=kmeans.labels\_, cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 2],

kmeans.cluster\_centers\_[:, 3], s = 120,

marker= 'x', color = 'k')

plt.title('Age vs Duration')

plt.show()

[Figure 6-6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html" \l "all_clusters) presents the behavior of the observations and cross sign x indicates the cluster center, i.e., the centroid. Age represents the more dispersed data, and the centroid of the age variable is located above the credit variable. Two continuous variables, namely credit and duration, are shown in the second subplot of [Figure 6-6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#all_clusters), where we observe clearly separated clusters. This figure suggests that the duration variable is more volatile compared to the credit variable. In the last subplot, the relationship between age and duration is examined via scatter analysis. It turns out that there are many overlapping observations across these two variables.



Figure 6-6. K-means clusters

Probability of Default Estimation with Logistic Regression

Having obtained the clusters, we are able to treat customers with similar characteristics the same way—that is, the model learns in an easier and more stable way if data with similar distributions is provided. Conversely, using all the customers for the entire sample might result in poor and unstable predictions.

This section is ultimately about calculating the probability of default with Bayesian estimation, but let’s first look at logistic regression for the sake of comparison.[**1**](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#idm45737223570832)

Logistic regression is a classification algorithm, widely applicable in the finance industry. In other words, it proposes a regression approach to the classification problem. Logistic regression seeks to predict discrete output, taking into account some independent variables.

Let *X* be the set of independent variables and *Y* be a binary (or multinomial) output. Then, the conditional probability becomes:

Pr(�=1|�=�)

This can be read as: given the values of *X*, what is the probability of having *Y* as 1? As the dependent variable of logistic regression is of the probabilistic type, we need to make sure the dependent variable cannot take on values other than between 0 and 1.

To this aim, a modification is applied known as *logistic (logit) transformation*, which is simply the log of the odds ratio (*p* / 1 - *p*):

���(�1-�)

And the logistic regression model takes the following form:

���(�1-�)=�0+�1�

Solving *p* results in:

�=��0+�1�1+��0+�1�

Let’s start off our application by preparing the data. First, we distinguish the clusters as 0 and 1. The credit data has a column named risk, suggesting the risk level of the customers. Next, the number of observations per risk in cluster 0 and cluster 1 are examined; it turns out we have 571 and 129 good customers in the cluster 0 and 1, respectively. In code:

In [22]: clusters, counts = np.unique(kmeans.labels\_, return\_counts=True) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO7-1)

In [23]: cluster\_dict = {}

for i in range(len(clusters)):

cluster\_dict[i] = scaled\_credit[np.where(kmeans.labels\_==i)] [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO7-2)

In [24]: credit['clusters'] = pd.DataFrame(kmeans.labels\_) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO7-3)

In [25]: df\_scaled = pd.DataFrame(scaled\_credit)

df\_scaled['clusters'] = credit['clusters']

In [26]: df\_scaled['Risk'] = credit['Risk']

df\_scaled.columns = ['Age', 'Job', 'Credit amount',

'Duration', 'Clusters', 'Risk']

In [27]: df\_scaled[df\_scaled.Clusters == 0]['Risk'].value\_counts() [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO7-4)

Out[27]: good 571

bad 193

Name: Risk, dtype: int64

In [28]: df\_scaled[df\_scaled.Clusters == 1]['Risk'].value\_counts() [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO7-5)

Out[28]: good 129

bad 107

Name: Risk, dtype: int64

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO7-1)*

Obtaining cluster numbers

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO7-2)*

Based on the cluster numbers, differentiating the clusters and storing them in a dictionary called cluster\_dict

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO7-3)*

Creating a clusters column using K-means labels

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO7-4)*

Observing the number of observations of categories within a cluster

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO7-5)*

Finding number of observations per category

Next, we draw a couple of bar plots to show the difference of the number of observations per risk level category (Figures [6-7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#risk_level1) and [6-8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#risk_level2)):

In [29]: df\_scaled[df\_scaled.Clusters == 0]['Risk'].value\_counts()\

.plot(kind='bar',

figsize=(10, 6),

title="Frequency of Risk Level");

In [30]: df\_scaled[df\_scaled.Clusters == 1]['Risk'].value\_counts()\

.plot(kind='bar',

figsize=(10, 6),

title="Frequency of Risk Level");

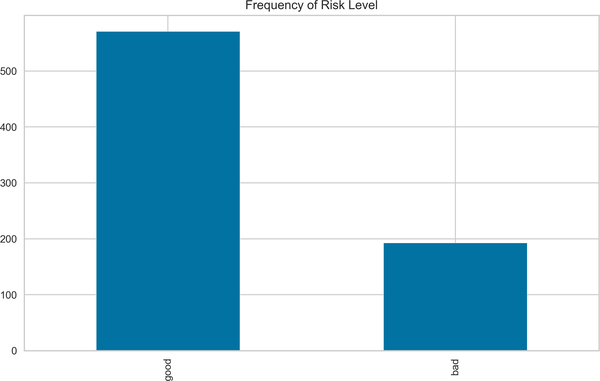


Figure 6-7. Frequency of risk level of the first cluster

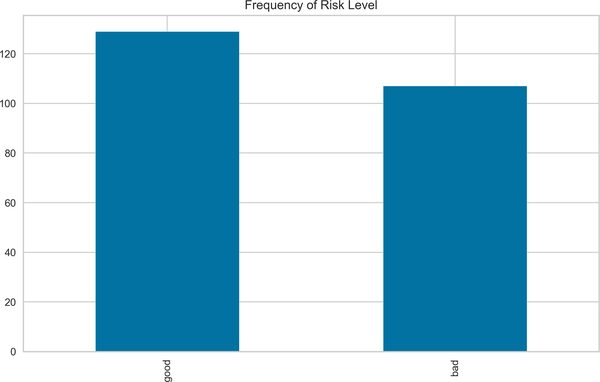


Figure 6-8. Frequency of risk level of the second cluster

Based on the clusters we defined previously, we can analyze the frequency of risk level by histogram. [Figure 6-7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html" \l "risk_level1) shows that there is an imbalance distribution across risk level in the first cluster, whereas the frequency of good and bad risk levels are more balanced, if not perfectly balanced, in [Figure 6-8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#risk_level2).

At this point, let’s take a step back and focus on an entirely different problem: *class imbalance*. In credit risk analysis, it is not uncommon to have a class imbalance problem. Class imbalance arises when one class dominates over another. To illustrate, in our case, given the data obtained from the first cluster, we have 571 customers with a good credit record and 193 customers with a bad one. As can be readily observed, customers with good credit records dominate over customers with bad records; that is basically what we refer to as a class imbalance.

There are numerous ways to handle this issue: up-sampling, down-sampling, the synthetic minority oversampling technique (SMOTE), and the edited nearest neighbor (ENN) rule. To take advantage of a hybrid approach, we’ll incorporate a combination of SMOTE and ENN so we can clean the unwanted overlapping observations between classes, which will help us detect the optimal balancing ratio and, in turn, boost the predictive performance (Tuong et al. 2018). Converting imbalanced data into balanced data will be our first step in predicting the probability of default, but please note that we will merely apply this technique to the data obtained from the first cluster.

Now, we next apply a train-test split. To do that, we need to convert the categorical variable Risk into a discrete variable. The category good takes a value of 1, and bad takes a value of 0. In a train-test split, 80% of the data is devoted to training samples and 20% of is allocated to the test sample:

In [31]: from sklearn.model\_selection import train\_test\_split

In [32]: df\_scaled['Risk'] = df\_scaled['Risk'].replace({'good': 1, 'bad': 0}) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO8-1)

In [33]: X = df\_scaled.drop('Risk', axis=1)

y = df\_scaled.loc[:, ['Risk', 'Clusters']]

In [34]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.2,

random\_state=42)

In [35]: first\_cluster\_train = X\_train[X\_train.Clusters == 0].iloc[:, :-1] [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO8-2)

second\_cluster\_train = X\_train[X\_train.Clusters == 1].iloc[:, :-1] [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO8-3)

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO8-1)*

Discretization of the variable

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO8-2)*

Creating data based on the first cluster and dropping last column from X\_train

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO8-3)*

Creating data based on the second cluster and dropping last column from X\_train

After these preparations, we are ready to move ahead and run the logistic regression to predict the probability of default. The library that we’ll make use of is called statsmodels, and it is allowed to have a summary table. The following result is based on the first cluster data. According to the result, the age, credit amount, and job variables are positively related with the creditworthiness of customer, while a negative association emerges between the dependent and duration variables. This finding suggests that all the estimated coefficients reveal statistically significant results at a 1% significance level. A general interpretation would be that a slide in duration and a surge in credit amount, age, and job imply a high probability of default:

In [36]: import statsmodels.api as sm

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_auc\_score, roc\_curve

from imblearn.combine import SMOTEENN [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO9-1)

import warnings

warnings.filterwarnings('ignore')

In [37]: X\_train1 = first\_cluster\_train

y\_train1 = y\_train[y\_train.Clusters == 0]['Risk'] [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO9-2)

smote = SMOTEENN(random\_state = 2) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO9-3)

X\_train1, y\_train1 = smote.fit\_resample(X\_train1, y\_train1.ravel()) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO9-4)

logit = sm.Logit(y\_train1, X\_train1) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO9-5)

logit\_fit1 = logit.fit() [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO9-6)

print(logit\_fit1.summary())

Optimization terminated successfully.

Current function value: 0.479511

Iterations 6

Logit Regression Results

==============================================================================

Dep. Variable: y No. Observations: 370

Model: Logit Df Residuals: 366

Method: MLE Df Model: 3

Date: Wed, 01 Dec 2021 Pseudo R-squ.: 0.2989

Time: 20:34:31 Log-Likelihood: -177.42

converged: True LL-Null: -253.08

Covariance Type: nonrobust LLR p-value: 1.372e-32

================================================================================

coef std err z P>|z| [0.025 0.975]

--------------------------------------------------------------------------------

Age 1.3677 0.164 8.348 0.000 1.047 1.689

Job 0.4393 0.153 2.873 0.004 0.140 0.739

Credit amount 1.3290 0.305 4.358 0.000 0.731 1.927

Duration -1.2709 0.246 -5.164 0.000 -1.753 -0.789

================================================================================

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO9-1)*

Importing SMOTEENN to deal with the class imbalance problem

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO9-2)*

Creating y\_train based on cluster 0 and risk level

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO9-3)*

Running the SMOTEENN method with a random state of 2

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO9-4)*

Turning the imbalanced data into balanced data

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO9-5)*

Configuring the logistic regression model

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO9-6)*

Running the logistic regression model

In what follows, prediction analysis is conducted by creating different datasets based on clusters. For the sake of testing, the following analysis is done with test data, and results in [Figure 6-9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#roc_auc_curve1_first):

In [38]: first\_cluster\_test = X\_test[X\_test.Clusters == 0].iloc[:, :-1] [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO10-1)

second\_cluster\_test = X\_test[X\_test.Clusters == 1].iloc[:, :-1] [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO10-2)

In [39]: X\_test1 = first\_cluster\_test

y\_test1 = y\_test[y\_test.Clusters == 0]['Risk']

pred\_prob1 = logit\_fit1.predict(X\_test1) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO10-3)

In [40]: false\_pos, true\_pos, \_ = roc\_curve(y\_test1.values, pred\_prob1) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO10-4)

auc = roc\_auc\_score(y\_test1, pred\_prob1) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO10-5)

plt.plot(false\_pos,true\_pos, label="AUC for cluster 1={:.4f} "

.format(auc))

plt.plot([0, 1], [0, 1], linestyle = '--', label='45 degree line')

plt.legend(loc='best')

plt.title('ROC-AUC Curve 1')

plt.show()

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO10-1)*

Creating first test data based on cluster 0

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO10-2)*

Creating second test data based on cluster 1

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO10-3)*

Running prediction using X\_test1

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO10-4)*

Obtaining false and true positives using roc\_curve function

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO10-5)*

Compute the roc-auc score

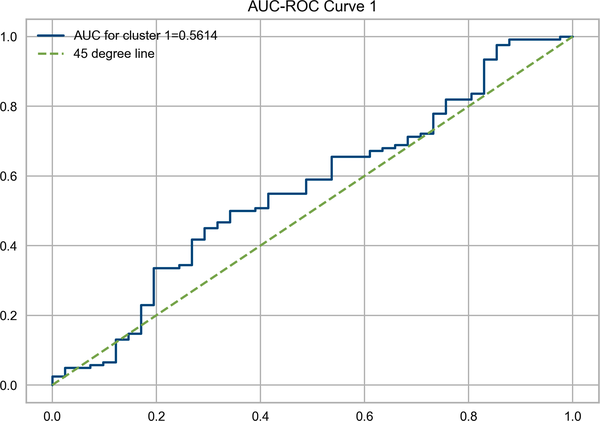


Figure 6-9. ROC-AUC curve of the first cluster

The ROC-AUC curve is a convenient tool in the presence of imbalanced data. The ROC-AUC curve in [Figure 6-9](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#roc_auc_curve1_first) suggests that the performance of the model is not very good, because it moves just above the 45-degree line. Generally speaking, given the test results, a good ROC-AUC curve should be close to 1, implying that there is a close-to-perfect separation.

Moving on to the second set of training samples obtained from the second cluster, the signs of the estimated coefficients of job, duration, and age are positive, suggesting that customers with job type of 1 and having larger duration tend to default, and the credit amount variable shows a negative relation with dependent variable. However, all the estimated coefficients are statistically insignificant at 95% confidence interval; therefore, it makes no sense to further interpret the findings.

Similar to what we did with the first set of test data, we create a second set of test data to run the prediction to draw the ROC-AUC curve, resulting in [Figure 6-10](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#roc_auc_curve1_second):

In [41]: X\_train2 = second\_cluster\_train

y\_train2 = y\_train[y\_train.Clusters == 1]['Risk']

logit = sm.Logit(y\_train2, X\_train2)

logit\_fit2 = logit.fit()

print(logit\_fit2.summary())

Optimization terminated successfully.

Current function value: 0.688152

Iterations 4

Logit Regression Results

==============================================================================

Dep. Variable: Risk No. Observations: 199

Model: Logit Df Residuals: 195

Method: MLE Df Model: 3

Date: Wed, 01 Dec 2021 Pseudo R-squ.: -0.0008478

Time: 20:34:33 Log-Likelihood: -136.94

converged: True LL-Null: -136.83

Covariance Type: nonrobust LLR p-value: 1.000

================================================================================

coef std err z P>|z| [0.025 0.975]

--------------------------------------------------------------------------------

Age 0.0281 0.146 0.192 0.848 -0.259 0.315

Job 0.1536 0.151 1.020 0.308 -0.142 0.449

Credit amount -0.1090 0.115 -0.945 0.345 -0.335 0.117

Duration 0.1046 0.126 0.833 0.405 -0.142 0.351

================================================================================

In [42]: X\_test2 = second\_cluster\_test

y\_test2 = y\_test[y\_test.Clusters == 1]['Risk']

pred\_prob2 = logit\_fit2.predict(X\_test2)

In [43]: false\_pos, true\_pos, \_ = roc\_curve(y\_test2.values, pred\_prob2)

auc = roc\_auc\_score(y\_test2, pred\_prob2)

plt.plot(false\_pos,true\_pos,label="AUC for cluster 2={:.4f} "

.format(auc))

plt.plot([0, 1], [0, 1], linestyle = '--', label='45 degree line')

plt.legend(loc='best')

plt.title('ROC-AUC Curve 2')

plt.show()

Given the test data, the result shown in [Figure 6-10](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#roc_auc_curve1_second) is worse than the previous application, as can be confirmed by the AUC score of 0.4064. Considering this data, we are far from saying that logistic regression is doing a good job of modeling probability of default using the German credit risk dataset.

We will now use different models to see how good the logistic regression is in modeling this type of problem relative to other methods. Thus, in the following part, we will take a look at Bayesian estimation with maximum a posteriori (MAP) probability and Markov Chain Monte Carlo (MCMC) approaches. We will then explore those approaches using a few well-known ML models—SVM, random forest, and neural networks using MLPRegressor—and we will test the deep learning model with TensorFlow. This application will show us which model works better in modeling the probability of default.

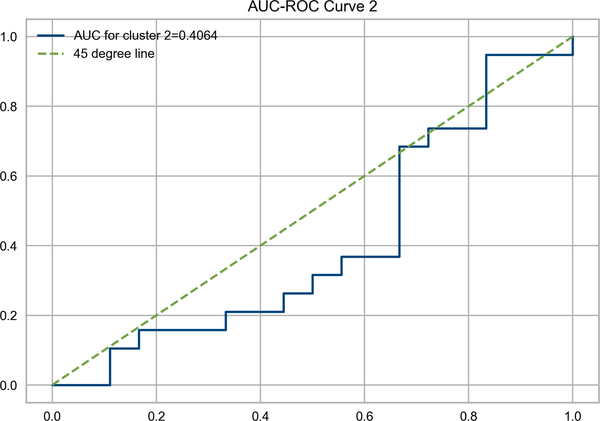


Figure 6-10. ROC-AUC curve of the second cluster

**Probability of Default Estimation with the Bayesian Model**

In this part, we’ll use the PYMC3 package, which is a Python package for Bayesian estimation, to predict the probability of default. However, there are several approaches for running Bayesian analysis using PYMC3, and for the first application, we’ll use the MAP distribution discussed in [Chapter 4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch04.html#chapter_4). As a quick reminder, given the representative posterior distribution, MAP becomes an efficient model in this case. Moreover, we select the Bayesian model with a deterministic variable (*p*) that is entirely determined by its parents—that is, age, job, credit amount, and duration.

Let’s compare the results obtained from Bayesian analysis with that of logistic regression:

In [44]: import pymc3 as pm [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-1)

import arviz as az [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-2)

In [45]: with pm.Model() as logistic\_model1: [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-3)

beta\_age = pm.Normal('coeff\_age', mu=0, sd=10) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-4)

beta\_job = pm.Normal('coeff\_job', mu=0, sd=10)

beta\_credit = pm.Normal('coeff\_credit\_amount', mu=0, sd=10)

beta\_dur = pm.Normal('coeff\_duration', mu=0, sd=10)

p = pm.Deterministic('p', pm.math.sigmoid(beta\_age \*

X\_train1['Age'] + beta\_job \*

X\_train1['Job'] + beta\_credit \*

X\_train1['Credit amount'] + beta\_dur \*

X\_train1['Duration'])) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-5)

with logistic\_model1:

observed = pm.Bernoulli("risk", p, observed=y\_train1) [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-6)

map\_estimate = pm.find\_MAP() [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-7)

Out[]: <IPython.core.display.HTML object>

In [46]: param\_list = ['coeff\_age', 'coeff\_job',

'coeff\_credit\_amount', 'coeff\_duration']

params = {}

for i in param\_list:

params[i] = [np.round(map\_estimate[i], 6)] [8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO11-8)

bayesian\_params = pd.DataFrame.from\_dict(params)

print('The result of Bayesian estimation:\n {}'.format(bayesian\_params))

The result of Bayesian estimation:

coeff\_age coeff\_job coeff\_credit\_amount coeff\_duration

0 1.367247 0.439128 1.32721 -1.269345

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-1)*

Importing PYMC3

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-2)*

Importing arviz for exploratory analysis of Bayesian models

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-3)*

Identifying Bayesian model as logistic\_model1

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-4)*

Identifying the assumed distributions of the variables as normal with defined mu and sigma parameters

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-5)*

Running a deterministic model using the first sample

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-6)*

Running a Bernoulli distribution to model the dependent variable

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-7)*

Fitting the MAP model to data

*[8](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO11-8)*

Storing all the results of the estimated coefficients into params with six decimals

The most striking observation is that the differences between estimated coefficients are so small that they can be ignored. The difference occurs in the decimals. Taking the estimated coefficient of the credit amount variable as an example, we have estimated the coefficient to be 1.3290 in logistic regression and 1.3272 in Bayesian analysis.

The story is more or less the same when it comes to comparing the analysis result based on the second cluster data:

In [47]: with pm.Model() as logistic\_model2:

beta\_age = pm.Normal('coeff\_age', mu=0, sd=10)

beta\_job = pm.Normal('coeff\_job', mu=0, sd=10)

beta\_credit = pm.Normal('coeff\_credit\_amount', mu=0, sd=10)

beta\_dur = pm.Normal('coeff\_duration', mu=0, sd=10)

p = pm.Deterministic('p', pm.math.sigmoid(beta\_age \*

second\_cluster\_train['Age'] +

beta\_job \* second\_cluster\_train['Job'] +

beta\_credit \*

second\_cluster\_train['Credit amount'] +

beta\_dur \*

second\_cluster\_train['Duration']))

with logistic\_model2:

observed = pm.Bernoulli("risk", p,

observed=y\_train[y\_train.Clusters == 1]

['Risk'])

map\_estimate = pm.find\_MAP()

Out[]: <IPython.core.display.HTML object>

In [48]: param\_list = [ 'coeff\_age', 'coeff\_job',

'coeff\_credit\_amount', 'coeff\_duration']

params = {}

for i in param\_list:

params[i] = [np.round(map\_estimate[i], 6)]

bayesian\_params = pd.DataFrame.from\_dict(params)

print('The result of Bayesian estimation:\n {}'.format(bayesian\_params))

The result of Bayesian estimation:

coeff\_age coeff\_job coeff\_credit\_amount coeff\_duration

0 0.028069 0.153599 -0.109003 0.104581

The most remarkable difference occurs in the duration variable. The estimated coefficients of this variable are 0.1046 and 0.1045 in logistic regression and Bayesian estimation, respectively.

Instead of finding the local maximum, which is sometimes difficult to get, we look for an approximate expectation based on the sampling procedure. This is referred to as MCMC in the Bayesian setting. As we discussed in [Chapter 4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch04.html#chapter_4), one of the most well known methods is the Metropolis-Hastings (M-H) algorithm.

The Python code that applies Bayesian estimation based on the M-H algorithm is shown in the following and results in [Figure 6-11](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#MCMC_risk_cluster1). Accordingly, we draw 10,000 posterior samples to simulate the posterior distribution for two independent Markov chains. The summary table for the estimated coefficients is provided in the code as well:

In [49]: import logging [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO12-1)

logger = logging.getLogger('pymc3') [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO12-2)

logger.setLevel(logging.ERROR) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO12-3)

In [50]: with logistic\_model1:

step = pm.Metropolis() [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO12-4)

trace = pm.sample(10000, step=step,progressbar = False) [5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO12-5)

az.plot\_trace(trace) [6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO12-6)

plt.show()

In [51]: with logistic\_model1:

display(az.summary(trace, round\_to=6)[:4]) [7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO12-7)

Out[]: mean sd hdi\_3% hdi\_97% mcse\_mean \

coeff\_age 1.392284 0.164607 1.086472 1.691713 0.003111

coeff\_job 0.448694 0.155060 0.138471 0.719332 0.002925

coeff\_credit\_amount 1.345549 0.308100 0.779578 1.928159 0.008017

coeff\_duration -1.290292 0.252505 -1.753565 -0.802707 0.006823

mcse\_sd ess\_bulk ess\_tail r\_hat

coeff\_age 0.002200 2787.022099 3536.314548 1.000542

coeff\_job 0.002090 2818.973167 3038.790307 1.001246

coeff\_credit\_amount 0.005670 1476.746667 2289.532062 1.001746

coeff\_duration 0.004826 1369.393339 2135.308468 1.001022

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO12-1)*

Importing the logging package to suppress the warning messages

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO12-2)*

Naming the package for logging

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO12-3)*

Suppressing errors without raising exceptions

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO12-4)*

Initiating the M-H model

*[5](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO12-5)*

Running the model with 10,000 samples and ignoring the progress bar

*[6](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO12-6)*

Creating a simple posterior plot using plot\_trace

*[7](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO12-7)*

Printing the first four rows of the summary result

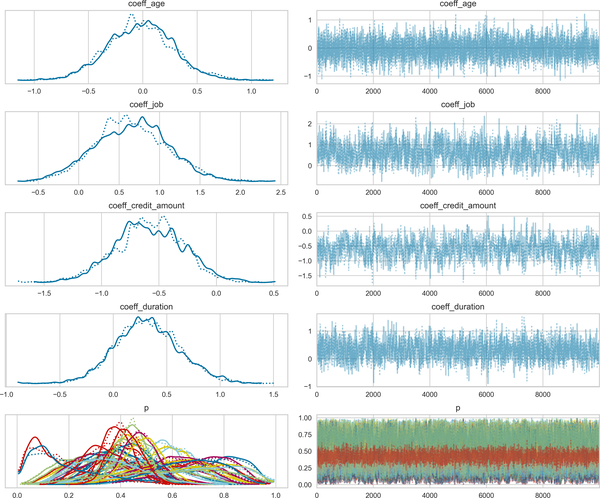


Figure 6-11. Bayesian estimation with M—H with first cluster

The result suggests that the predictive performances are supposed be very close to that of logistic regression, as the estimated coefficients of these two models are quite similar.

In [Figure 6-11](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#MCMC_risk_cluster1), we see the dashed and solid lines. Given the first cluster data, the plot located on the lefthand side of [Figure 6-11](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#MCMC_risk_cluster1) shows the sample values of the related parameters. Though it is not our present focus, we can observe the deterministic variable, *p*, located in the last plot.

In a similar vein, the result of Bayesian estimation with M-H based on the second cluster performs very closely to the logistic regression. However, the results obtained from MAP application are better, which is expected primarily because M-H works with random sampling. It is not, however, the only potential reason for this small deviation that we’ll discuss.

As for the data that we obtained from the second cluster, the result of Bayesian estimation with M-H can be seen in the following code, which also creates the plot shown in [Figure 6-12](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#MCMC_risk_cluster2):

In [52]: with logistic\_model2:

step = pm.Metropolis()

trace = pm.sample(10000, step=step,progressbar = False)

az.plot\_trace(trace)

plt.show()

In [53]: with logistic\_model2:

display(az.summary(trace, round\_to=6)[:4])

Out[]: mean sd hdi\_3% hdi\_97% mcse\_mean \

coeff\_age 0.029953 0.151466 -0.262319 0.309050 0.002855

coeff\_job 0.158140 0.153030 -0.125043 0.435734 0.003513

coeff\_credit\_amount -0.108844 0.116542 -0.328353 0.105858 0.003511

coeff\_duration 0.103149 0.128264 -0.142609 0.339575 0.003720

mcse\_sd ess\_bulk ess\_tail r\_hat

coeff\_age 0.002019 2823.255277 3195.005913 1.000905

coeff\_job 0.002485 1886.026245 2336.516309 1.000594

coeff\_credit\_amount 0.002483 1102.228318 1592.047959 1.002032

coeff\_duration 0.002631 1188.042552 1900.179695 1.000988

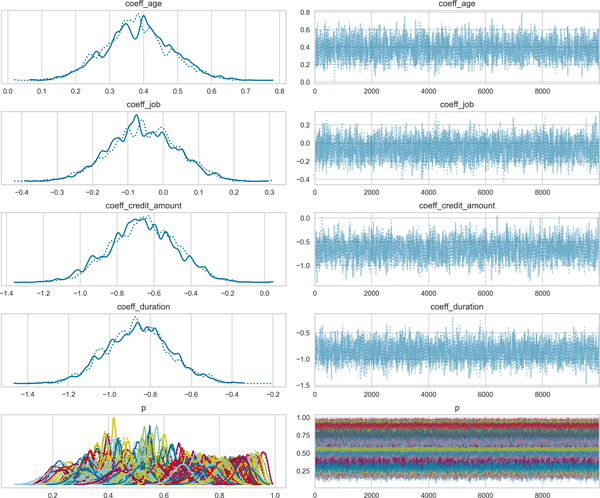


Figure 6-12. Bayesian estimation with M—H with second cluster

Let’s now discuss the limitations of the M-H model, which may shed some light on the discrepancies across the model results. One disadvantage of the M-H algorithm is its sensitivity to step size. Small steps hinder the convergence process. Conversely, big steps may cause a high rejection rate. Besides, M-H may suffer from rare events—as the probability of these events are low, requiring a large sample to obtain a reliable estimation—and that is our focus in this case.

Now, let’s consider what happens if we use SVM to predict probability of default and compare its performance with logistic regression.

**Probability of Default Estimation with Support Vector Machines**

SVM is thought to be a parametric model, and it works well with high-dimensional data. The probability of default case in a multivariate setting may provide fertile ground for running SVM. Before proceeding, it would be a good idea to briefly discuss a new approach that we will use to run hyperparameter tuning, namely HalvingRandomSearchCV.

HalvingRandomSearchCV works with iterative selection so that it uses fewer resources, thereby boosting performance and getting you some time back. HalvingRandomSearchCV tries to find the optimal parameters using successive halving to identify candidate parameters. The logic behind this process is as follows:

1. Evaluate all parameter combinations, exploiting a certain number of training samples at first iteration.
2. Use some of the selected parameters in the second iteration with a large number of training samples.
3. Only include the top-scoring candidates in the model until the last iteration.

Using the credit dataset, we predict the probability of default with support vector classification (SVC). Again, we use two different datasets based on the clustering we performed at the very first part of this chapter. The results are provided in the following:

In [54]: from sklearn.svm import SVC

from sklearn.experimental import enable\_halving\_search\_cv [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO13-1)

from sklearn.model\_selection import HalvingRandomSearchCV [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO13-2)

import time

In [55]: param\_svc = {'gamma': [1e-6, 1e-2],

'C':[0.001,.09,1,5,10],

'kernel':('linear','rbf')}

In [56]: svc = SVC(class\_weight='balanced')

halve\_SVC = HalvingRandomSearchCV(svc, param\_svc,

scoring = 'roc\_auc', n\_jobs=-1) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO13-3)

halve\_SVC.fit(X\_train1, y\_train1)

print('Best hyperparameters for first cluster in SVC {} with {}'.

format(halve\_SVC.best\_score\_, halve\_SVC.best\_params\_))

Best hyperparameters for first cluster in SVC 0.8273860106443562 with

{'kernel': 'rbf', 'gamma': 0.01, 'C': 1}

In [57]: y\_pred\_SVC1 = halve\_SVC.predict(X\_test1) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO13-4)

print('The ROC AUC score of SVC for first cluster is {:.4f}'.

format(roc\_auc\_score(y\_test1, y\_pred\_SVC1)))

The ROC AUC score of SVC for first cluster is 0.5179

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO13-1)*

Importing the library to enable successive halving search

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO13-2)*

Importing the library to run the halving search

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO13-3)*

Running the halving search using parallel processing

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO13-4)*

Running a prediction analysis

An important step to take in SVM is hyperparameter tuning. Using a halving search approach, we try to find out the best combination of kernel, gamma, and C. It turns out that the only difference across the two different samples occurs in the gamma and C hyperparameters. In the first cluster, the optimal C score is 1, whereas it is 0.001 in the second one. The higher C value indicates that we should choose a smaller margin to make a better classification. As for the gamma hyperparameter, both clusters take the same value. Having a lower gamma amounts to a larger influence of the support vector on the decision. The optimal kernel is Gaussian, and the gamma value is 0.01 for both clusters.

The AUC performance criteria indicates that the predictive performance of SVC is slightly below that of logistic regression. More precisely, AUC of the SVC is 0.5179, and that implies that SVC performs worse than logistic regression for the first cluster.

The second cluster shows that the performance of SVC is even slightly worse than that of the first cluster, and this indicates the SVC does not perform well on this data, as it is not clearly separable data, this implies that SVC does not work well with low-dimensional spaces:

In [58]: halve\_SVC.fit(X\_train2, y\_train2)

print('Best hyperparameters for second cluster in SVC {} with {}'.

format(halve\_SVC.best\_score\_, halve\_SVC.best\_params\_))

Best hyperparameters for second cluster in SVC 0.5350758636788049 with

{'kernel': 'rbf', 'gamma': 0.01, 'C': 0.001}

In [59]: y\_pred\_SVC2 = halve\_SVC.predict(X\_test2)

print('The ROC AUC score of SVC for first cluster is {:.4f}'.

format(roc\_auc\_score(y\_test2, y\_pred\_SVC2)))

The ROC AUC score of SVC for first cluster is 0.5000

Well, maybe we’ve had enough of parametric methods—let’s move on to nonparametric methods. Now, the word *nonparametric* may sound confusing, but it is nothing but a model with an infinite number of parameters, and one that becomes more complex as the number of observations increases. Random forest is one of the most applicable nonparametric models in ML, and we’ll discuss that next.

**Probability of Default Estimation with Random Forest**

The random forest classifier is another model we can employ to model the probability of default. Although random forest fails in high-dimensional cases, our data is not that complex, and the beauty of random forest lies in its good predictive performance in the presence of a large number of samples, so it’s plausible to think that the random forest model might outperform the SVC model.

Using halving search approach, we try to find out the best combination of n\_estimators, criterion, max\_features, max\_depth, min\_samples\_split. The result suggests that we use n\_estimators of 300, min\_samples\_split of 10, max\_depth of 6 with a gini criterion, and sqrt max\_features for the first cluster. As for the second cluster, we have two different optimal hyperparameters as can be seen in the following. Having larger depth in a tree-based model amounts to having a more complex model. With that said, the model proposed for the second cluster is a bit more complex. The max\_features hyperparameter seems to be different across samples; in the first cluster, the maximum number of features is picked via numberoffeatures.

Given the first cluster data, the AUC score of 0.5387 indicates that random forest has a better performance compared to the other models:

In [60]: from sklearn.ensemble import RandomForestClassifier

In [61]: rfc = RandomForestClassifier(random\_state=42)

In [62]: param\_rfc = {'n\_estimators': [100, 300],

'criterion' :['gini', 'entropy'],

'max\_features': ['auto', 'sqrt', 'log2'],

'max\_depth' : [3, 4, 5, 6],

'min\_samples\_split':[5, 10]}

In [63]: halve\_RF = HalvingRandomSearchCV(rfc, param\_rfc,

scoring = 'roc\_auc', n\_jobs=-1)

halve\_RF.fit(X\_train1, y\_train1)

print('Best hyperparameters for first cluster in RF {} with {}'.

format(halve\_RF.best\_score\_, halve\_RF.best\_params\_))

Best hyperparameters for first cluster in RF 0.8890871444218126 with

{'n\_estimators': 300, 'min\_samples\_split': 10, 'max\_features': 'sqrt',

'max\_depth': 6, 'criterion': 'gini'}

In [64]: y\_pred\_RF1 = halve\_RF.predict(X\_test1)

print('The ROC AUC score of RF for first cluster is {:.4f}'.

format(roc\_auc\_score(y\_test1, y\_pred\_RF1)))

The ROC AUC score of RF for first cluster is 0.5387

The following code shows a random forest run based on the second cluster:

In [65]: halve\_RF.fit(X\_train2, y\_train2)

print('Best hyperparameters for second cluster in RF {} with {}'.

format(halve\_RF.best\_score\_, halve\_RF.best\_params\_))

Best hyperparameters for second cluster in RF 0.6565 with

{'n\_estimators': 100, 'min\_samples\_split': 5, 'max\_features': 'auto',

'max\_depth': 5, 'criterion': 'entropy'}

In [66]: y\_pred\_RF2 = halve\_RF.predict(X\_test2)

print('The ROC AUC score of RF for first cluster is {:.4f}'.

format(roc\_auc\_score(y\_test2, y\_pred\_RF2)))

The ROC AUC score of RF for first cluster is 0.5906

Random forest has a much better predictive performance in the second cluster, with an AUC score of 0.5906. Given the predictive performance of random forest, we can conclude that random forest does a better job of fitting the data. This is partly because of the low-dimensional characteristics of the data, as random forest turns out to be a good choice when data has low dimensionality and a large number of observations.

**Probability of Default Estimation with Neural Network**

Given the complexity of the probability of default estimation, unveiling the hidden structure of the data is a tough task, but the NN structure does a good job handling this, so it would be an ideal candidate model for such tasks. In setting up the NN model, GridSearchCV is used to optimize the number of hidden layers, optimization technique, and learning rate.

In running the model, we first employ the MLP library, which allows us to control for many parameters, including hidden layer size, optimization technique (solver), and learning rate. Comparing the optimized hyperparameters of the two clusters indicates that the only difference is in the number of neurons in the hidden layer. Accordingly, we have larger number of neurons in the first hidden layer in cluster one. However, the neuron number is larger in the second hidden layer in the second cluster.

The following code suggests that data based on the first cluster is only a marginal improvement. In other words, the AUC moves to 0.5263, only slightly worse than random forest:

In [67]: from sklearn.neural\_network import MLPClassifier

In [68]: param\_NN = {"hidden\_layer\_sizes": [(100, 50), (50, 50), (10, 100)],

"solver": ["lbfgs", "sgd", "adam"],

"learning\_rate\_init": [0.001, 0.05]}

In [69]: MLP = MLPClassifier(random\_state=42)

In [70]: param\_halve\_NN = HalvingRandomSearchCV(MLP, param\_NN,

scoring = 'roc\_auc')

param\_halve\_NN.fit(X\_train1, y\_train1)

print('Best hyperparameters for first cluster in NN are {}'.

format(param\_halve\_NN.best\_params\_))

Best hyperparameters for first cluster in NN are {'solver': 'lbfgs',

'learning\_rate\_init': 0.05, 'hidden\_layer\_sizes': (100, 50)}

In [71]: y\_pred\_NN1 = param\_halve\_NN.predict(X\_test1)

print('The ROC AUC score of NN for first cluster is {:.4f}'.

format(roc\_auc\_score(y\_test1, y\_pred\_NN1)))

The ROC AUC score of NN for first cluster is 0.5263

The ROC-AUC score obtained from the second cluster is 0.6155, with two hidden layers endowed with 10 and 100 neurons, respectively. Moreover, the best optimization technique is adam, and optimum initial learning rate is 0.05. This is the highest AUC score we’ve obtained, implying that the NN is able to capture the dynamics of the complex and nonlinear data, as shown here:

In [72]: param\_halve\_NN.fit(X\_train2, y\_train2)

print('Best hyperparameters for first cluster in NN are {}'.

format(param\_halve\_NN.best\_params\_))

Best hyperparameters for first cluster in NN are {'solver': 'lbfgs',

'learning\_rate\_init': 0.05, 'hidden\_layer\_sizes': (10, 100)}

In [73]: y\_pred\_NN2 = param\_halve\_NN.predict(X\_test2)

print('The ROC AUC score of NN for first cluster is {:.4f}'.

format(roc\_auc\_score(y\_test2, y\_pred\_NN2)))

The ROC AUC score of NN for first cluster is 0.6155

**Probability of Default Estimation with Deep Learning**

Let’s now take a look at the performance of a deep learning model using TensorFlow via KerasClassifier, which enables us to control for the hyperparameters.

The hyperparameters that we tune in this model are batch size, epoch, and dropout rate. As probability of default is a classification problem, the sigmoid activation function appears to be the optimal function to use. Deep learning is based on the structure of NNs, but provides a more complex structure, so it is expected to better capture the dynamics of data in a way that enables us to have better predictive performance.

As we can see in the following code, the predictive performance of the second sample stumbles, however, with an AUC score of 0.5628:

In [74]: from tensorflow import keras

from tensorflow.keras.wrappers.scikit\_learn import KerasClassifier [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO14-1)

from tensorflow.keras.layers import Dense, Dropout

from sklearn.model\_selection import GridSearchCV

import tensorflow as tf

import logging [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO14-2)

tf.get\_logger().setLevel(logging.ERROR) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO14-3)

In [75]: def DL\_risk(dropout\_rate,verbose=0):

model = keras.Sequential()

model.add(Dense(128,kernel\_initializer='normal',

activation = 'relu', input\_dim=4))

model.add(Dense(64, kernel\_initializer='normal',

activation = 'relu'))

model.add(Dense(8,kernel\_initializer='normal',

activation = 'relu'))

model.add(Dropout(dropout\_rate))

model.add(Dense(1, activation="sigmoid"))

model.compile(loss='binary\_crossentropy', optimizer='rmsprop')

return model

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO14-1)*

Importing KerasClassifier to run grid search

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO14-2)*

Importing logging to suppress the warning messages

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO14-3)*

Naming TensorFlow for logging

Given the optimized hyperparameters of dropout, batch size, and epoch, the deep learning model produces the best performance among the models we have employed so far, with an AUC score of 0.5614. The difference between MLPClassifier and deep learning models used in this chapter is the number of neurons in the hidden layer. Technically, these two models are deep learning models with different structures.

In [76]: parameters = {'batch\_size': [10, 50, 100],

'epochs': [50, 100, 150],

'dropout\_rate':[0.2, 0.4]}

model = KerasClassifier(build\_fn = DL\_risk) [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO15-1)

gs = GridSearchCV(estimator = model,

param\_grid = parameters,

scoring = 'roc\_auc') [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO15-2)

In [77]: gs.fit(X\_train1, y\_train1, verbose=0)

print('Best hyperparameters for first cluster in DL are {}'.

format(gs.best\_params\_))

Best hyperparameters for first cluster in DL are {'batch\_size': 10,

'dropout\_rate': 0.2, 'epochs': 50}

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO15-1)*

Calling a predefined function named DL\_risk to run with optimized hyperparameters

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO15-2)*

Applying the grid search

In [78]: model = KerasClassifier(build\_fn = DL\_risk, [1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO16-1)

dropout\_rate = gs.best\_params\_['dropout\_rate'],

verbose = 0,

batch\_size = gs.best\_params\_['batch\_size'], [2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO16-2)

epochs = gs.best\_params\_['epochs']) [3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO16-3)

model.fit(X\_train1, y\_train1)

DL\_predict1 = model.predict(X\_test1) [4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#callout_credit_risk_estimation_CO16-4)

DL\_ROC\_AUC = roc\_auc\_score(y\_test1, pd.DataFrame(DL\_predict1.flatten()))

print('DL\_ROC\_AUC is {:.4f}'.format(DL\_ROC\_AUC))

DL\_ROC\_AUC is 0.5628

*[1](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO16-1)*

Running deep learning algorithm with optimum hyperparameter of dropout rate

*[2](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO16-2)*

Running deep learning algorithm with optimum hyperparameter of batch size

*[3](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO16-3)*

Running deep learning algorithm with optimum hyperparameter of epoch number

*[4](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#co_credit_risk_estimation_CO16-4)*

Computing the ROC-AUC score after flattening the prediction

In [79]: gs.fit(X\_train2.values, y\_train2.values, verbose=0)

print('Best parameters for second cluster in DL are {}'.

format(gs.best\_params\_))

Best parameters for second cluster in DL are {'batch\_size': 10,

'dropout\_rate': 0.2, 'epochs': 150}

In [80]: model = KerasClassifier(build\_fn = DL\_risk,

dropout\_rate= gs.best\_params\_['dropout\_rate'],

verbose = 0,

batch\_size = gs.best\_params\_['batch\_size'],

epochs = gs.best\_params\_['epochs'])

model.fit(X\_train2, y\_train2)

DL\_predict2 = model.predict(X\_test2)

DL\_ROC\_AUC = roc\_auc\_score(y\_test2, DL\_predict2.flatten())

print('DL\_ROC\_AUC is {:.4f}'.format(DL\_ROC\_AUC))

DL\_ROC\_AUC is 0.5614

This finding confirms that DL models have become increasingly popular in financial modeling. In the industry, however, due to the opaque nature of network structure, this method is suggested for use in conjunction with traditional models.

Conclusion

Credit risk analysis has a long tradition but is also still a challenging task to accomplish. This chapter attempted to present a brand new ML-based approach to tackling this problem and to getting better predictive performance. In the first part of the chapter, the main concepts related to credit risk were provided. Then, we applied a well-known parametric model, logistic regression, to German credit risk data. The performance of logistic regression was then compared with Bayesian estimation based on MAP and M-H. Finally, core machine learning models—namely SVC, random forest, and NNs with deep learning—were employed, and the performance of all models was compared.

In the next chapter, a neglected dimension risk will be introduced: liquidity risk. The appreciation of liquidity risk has grown considerably since the 2007–2008 financial crisis and has turned out to be an important part of risk management.

References

Articles cited in this chapter:

* Basel Committee on Banking Supervision, and Bank for International Settlements. 2000. “Principles for the Management of Credit Risk.” Bank for International Settlements.
* Le, Tuong, Mi Young Lee, Jun Ryeol Park, and Sung Wook Baik. 2018. “Oversampling Techniques for Bankruptcy Prediction: Novel Features from a Transaction Dataset.” *Symmetry* 10 (4): 79.
* Tibshirani, Robert, Guenther Walther, and Trevor Hastie. 2001. “Estimating the Number of Clusters in a Data Set via the Gap Statistic.” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63 (2): 411-423.

Books and PhD theses cited in this chapter:

* Rokach, Lior, and Oded Maimon. 2005. “Clustering methods.” In *Data Mining and Knowledge Discovery Handbook*, 321-352. Boston: Springer.
* Wehrspohn, Uwe. 2002. “Credit Risk Evaluation: Modeling-Analysis-Management.” PhD dissertation. Harvard.

[**1**](https://learning.oreilly.com/library/view/machine-learning-for/9781492085249/ch06.html#idm45737223570832-marker) It is useful to run logistic regression to initialize results for priors in Bayesian estimation.