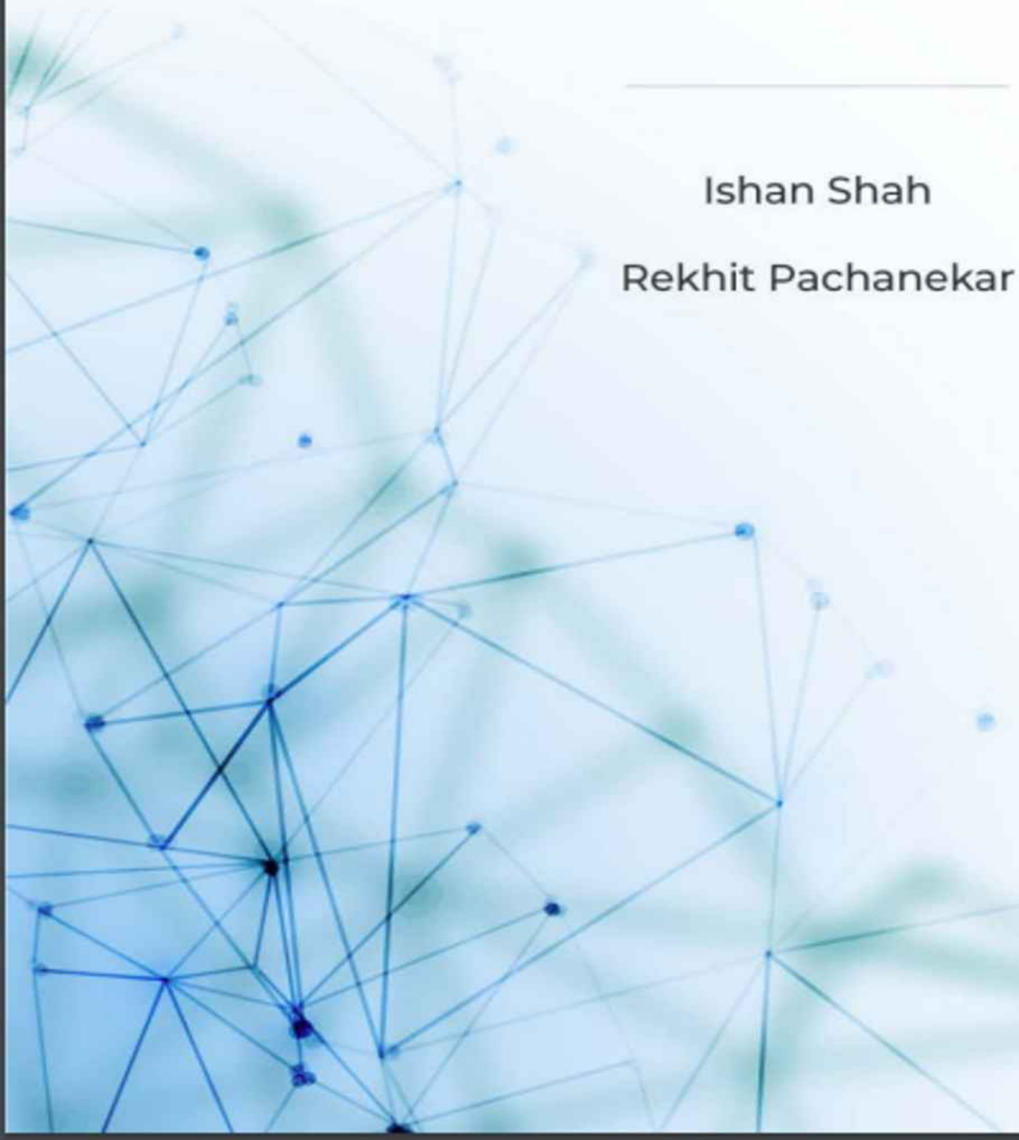
MACHINE LEARNING IN TRADING

Step by step implementation of Machine Learning models



Ishan Shah

Rekhit Pachanekar

QUANT INSTI®
GO ALGO

A QuantInsti® Publication

Machine Learning in Trading

Ishan Shah Rekhit Pachanekar

Machine Learning in Trading

Step by step implementation of machine learning models. QuantInsti

Contents

Part I
Introduction Preface

“If you invent a breakthrough in artificial intelligence, so machines can learn, that is worth 10 Microsofts.”

— Bill Gates Alan Turing had created a test, called the “Turing Test” in 1950. The test was to judge if a machine was ‘human’ enough. For it to be so, the machine had to interact with a real person. In such interactions, if the person found the machine’s behavior indistinguishable from a human, the machine passed the test.

Do you think we have such devices today?
Google recently debuted an AI voice assistant which could book appointments by calling the establishment and conversing with the receptionist.
How can a machine think like a human?

A conventional computer program has an algorithm, or a set of steps to execute. The program does this efficiently. But tell a normal calculator to find the gravitational force on Mars, and it won’t be able to, until you program the equation for gravity. Thus, was born this new field called machine learning, where the machines weren’t limited to their initial program. They could learn! And improve themselves.

Arthur Samuel wrote the first computer learning program to play checkers. It studied the previous moves and incorporated them in its moves when it played against an opponent. Today, Google’s DeepMind AI has defeated a human player in the board game Go! A feat that was earlier thought to be impossible. All this, thanks to machine learning.

Machine learning has varied applications. From agricultural forecasts, to stock price estimations, you can use machine learning almost anywhere where data has to be analysed. Some machine learning algorithms actually unearth information that you didn’t even know existed.

And in today’s world, we are generating data more than ever! According to the World Economic Forum, by 2025, we will generate almost 463 exabytes (that’s one billion GB) of data each day!

Obviously, a human will not be able to keep up with the data overload, and conventional computer programs might not be able to handle such large amounts of data.

Thus, we need machine learning, which harnesses the adaptive learning method of a human and combine it with the efficiency and speed of a computer program. This is more evident in the trading domain than in others.

If we focus on algorithmic trading, we routinely execute the following tasks:

- Download and manage price, fundamental and other alternative data from multiple data vendors and in different formats.
- Pre-processing the data and cleaning
- Forming hypothesis for a trading strategy and backtesting it.
- Automating the trading strategy to make sure emotions don’t get in the way of your trading strategy.

We found out that we can outsource most of these tasks to the machine. And this is the reason you, the reader, have a book which talks about machine learning and its applications in trading.

Why was this book written? Machine learning is a vast topic if you look at the various disciplines originating from it. You will also hear buzzwords such as AI, Neural Networks, Deep learning, AI Engineering being associated with machine learning.

Our aim in this book is to demystify these concepts and provide clarity on how machine learning is different from conventional programming. And further, how machine learning can be used to gain an edge in the trading domain.

We have structured the book in such a way that initially, you will learn about the various tasks carried out by a machine learning algorithm.

When it is appropriate, you will be introduced to the code which is required to run these tasks. If you are well versed with Python programming, you will be able to breeze through these sections and understand the concepts easily.

What if I have no experience in programming, or Python?

Don’t worry. Python itself is a relatively easier language to understand and program in. Moreover, we have tried to break down the code into bite-sized segments and explain what the code does in plain English. This not only helps you understand faster, but will also serve as a support later when you start to code and build your own ML algorithm.

Once the basics of ML tasks are covered, we move further and go through various ML algorithms, one chapter at a time. We will build upon the concepts and give you a step by step guide on how an idea can be converted to rules, and apply machine learning to test this idea on real-world data. And if you are satisfied with the results, move further and implement them in real life.

Who should read this? We think it should also be useful to:

- Anyone who heard about machine learning algorithms and is excited to know more.
- Programmers who would like to expand their horizons and see how machine learning can help them optimize their work.
- Algo traders who are continuously looking for an edge over the competition.

As said before, we do not want the readers of this book to be computer programmers, as we have tried to keep the text simple with lots of real world examples to illustrate various concepts.

What’s in this book? The material presented here is an elementary introduction to the world of machine learning. You can think of it as a book telling you about the foundations of machine learning and how it is applied in real life.

From the outset, we believe that only theory is not enough to retain knowledge. You need to know how you can apply this knowledge in the real world. Thus, our book contains lots of real world examples, especially in the field of trading. But rest assured that these concepts can be transferred to any

other discipline which requires data analysis.
How to read this? The ideal way to go through this book is one chapter at a time. But you can try other options as well, such as:

1. Skim through the book and stop at a chapter that catches your eye. And then go back if you want some preliminary knowledge.
2. If you have worked on machine learning algorithms before, then feel free to go to any random chapter you like and see if the text can be improved or not.

Where else can you find all this? Machine learning has been around for more than half a century now. In fact, the word “robota” indicating a human like machine, was first mentioned in a Czech play in 1920!

Thus, you can find a variety of information related to machine learning in the form of blogs, courses, videos and other mediums. And there are free resources too. The idea of this book was to present the core set and principles of machine learning in an easy to understand language and also, in a compact form.

Code and Data Used in the Book You can download all the code and data files from the Github link: ‘https://github.com/quantra-go-algo/ml-trading-ebook’. We will also update the codes and data files on the same link.

Suggestions and Errors We believe that none of us is perfect in this world. Everyone is constantly striving to improve themselves, and we are no different. Thus, suggestions and feedback from you are welcome. You can tell us anything we can imrpove or just leave a line at quantra@quantinsti.com.

1 Introduction to Machine Learning

Machine learning has been a hot topic for decades. However, research in this field had declined in the late 1900s. There were two things which served as a roadblock for machine learning progress.

1. The computing resources required for machine learning were prohibitively expensive, and hence, only large institutions were capable of investing in its research.

2. Machine learning requires data to learn. In the early days, this data was hard to acquire.

But with the revolution in the computer industry, computing resources became cheaper and the amount of data generated increased multifold. This was digital data, so even acquiring it became cheaper. All this contributed to a spurt in machine learning progress. This has resulted in numerous machine learning libraries able to process years of data in seconds. Even on your personal computer!

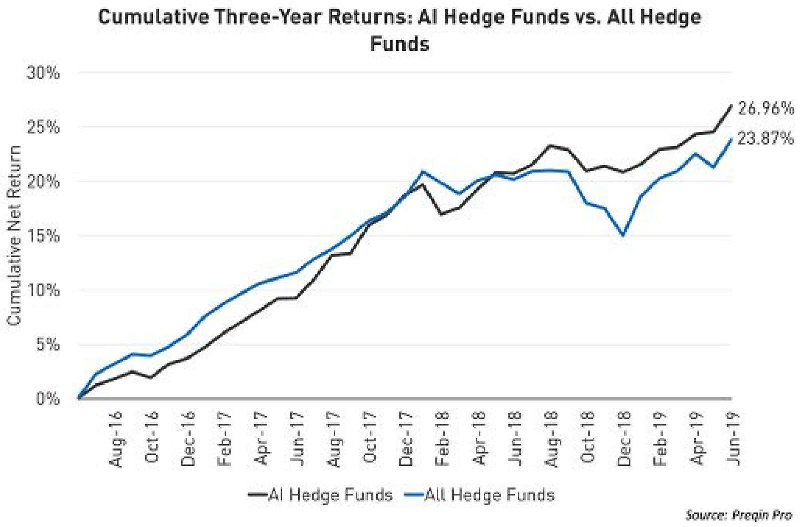
Why should you use machine learning? The world is moving towards automation and machine learning, and the finance industry is no different. Zest AI, which develops AI solutions for lenders, helps auto lenders screen potential clients and advises them on the correct lending rate to be charged. Underwrite.ai claims that after using their ML model to screen clients, they cut the first payment default rate from 32% to 8%.

What about machine learning in trading? Machine learning (ML) algorithms can create an investment portfolio for you. A human advisor has to study the markets, earn advisory certifications to be confident enough to suggest a portfolio to her clients. On the other hand, an ML algorithm can carry out the same task in seconds. Betterment and Wealthfront are companies who provide investment services through their machine learning platform.

How can an ML algorithm advise a person on investments?

Let’s take a simple example to illustrate this concept. First, the ML algorithm receives the details of all the clients and their respective portfolios as input. The ML algo then learns the relationship between the portfolio advised and the characteristics of the client (from our input data). After this, the model is ready to make recommendations for the portfolio of any client based on her characteristics (like age, income, employment status, etc.). For example, if the client is a 25- year-old female with a high risk appetite, the model, based on previous patterns learnt, would suggest a portfolio inclined towards technology stocks such as Apple and Facebook.

The ML algorithms are really fast and are likely to do this in a matter of seconds. The advantage of speed is harnessed by hedge funds that are using algorithmic trading. Renaissance Technologies is just one example from hundreds of AI-driven funds. Renaissance Technologies is known to use machine learning in its trading strategies. And it has about $55 billion worth of assets in its management. Various hedge funds and high-frequency trading firms rely on machine learning to create their trading strategies.



Source: https://www.preqin.com/insights/research/blogs/the-rise-of-themachines-ai-funds-are-outperforming-the-hedge-fund-benchmark

In a 2019 report, Preqin reported that it followed the performance of 152 ML and AIdriven hedge funds. Preqin compared the AI funds’ performance to their own benchmark of hedge funds. And saw that AI outperformed by 3 percentage points.

Is machine learning only for hedge funds and large institutions? That was the case for a long time but not anymore. With the advent of technology and open source data science resources, retail traders and individuals have also started to learn and use machine learning. Let’s take an example here. Assume that you have an understanding of the market trend. You have a simple trading strategy using a few technical indicators which help you to predict the market trend and trade accordingly. Meanwhile, another trader uses machine learning to implement the same trading strategy. Although, instead of a few old technical indicators, he will allow the machine to go through hundreds of technical indicators. He lets the machine decide which indicator performs best in predicting the correct market trend. While the regular trader might have settled for RSI or MA, the ML algo will be able to go through many more technical indicators and pick the best ones, in terms of prediction power.

This technique is known as feature selection, where machine learning chooses between different features of indicators, and chooses the most efficient ones for prediction. These features can be, to name a few, price change, buy/sell signals, volatility signals, volume weights, etc. It is obvious that the trader who has done more quantitative research, better backtesting on the historical data, and better optimisation, has a greater chance of performing better in live markets.

How do you implement machine learning? Machine learning can be implemented in any programming language. In fact, C++ is used by programmers to implement ML projects which are time-sensitive. If you are an HFT firm, you might choose C++. But an overwhelming majority use Python because it is easier to learn and it supports various ML libraries. The syntax is also cleaner and easy to understand. Depending on your requirements, you can choose any Python library, be it scikit-learn or TensorFlow.

Thus, you won’t need a computer science or programming degree to implement machine learning algorithms in your domain. You simply need to know the Python ML libraries and syntax. And with a few lines of code, implement your ML project in real-time.

This book gives you a detailed and step-by-step guide to create a machine learning trading strategy.

1. Use Python libraries, which will help you read the data.
2. Use machine learning to find patterns in the data.
3. Generate signals on whether you should buy or sell the asset.
4. Learn to analyse the performance of the model.

Let us answer a few questions related to machine learning before we move on to a real life machine learning model.

What is Machine Learning? A key difference between a regular algorithm (algo) and a machine learning algo is the “learning” model which allows the algorithm to learn from the data and make its own decisions. This allows machines to perform tasks, which otherwise are impossible for them to perform. Such tasks can be as simple as recognising human handwriting or as complex as self-driving cars!

For example, say an algorithm is supposed to correctly distinguish between a male and a female face from a group of ID-card photos. A machine learning (ML) algorithm would be trained on one part of the data to ‘learn’ how to recognize any face. Where a simple algorithm would not be capable of performing this task, an ML algo would not only be able to categorise the photos as trained, it would continuously learn from testing data and add to its “learning” to become more accurate in its predictions!

Recall how often Facebook (FB) prompts you to tag the person in the picture! Among billions of users, FB ML algos are able to correctly match different pictures of the same person and identify her! Tesla’s autopilot feature is another example.

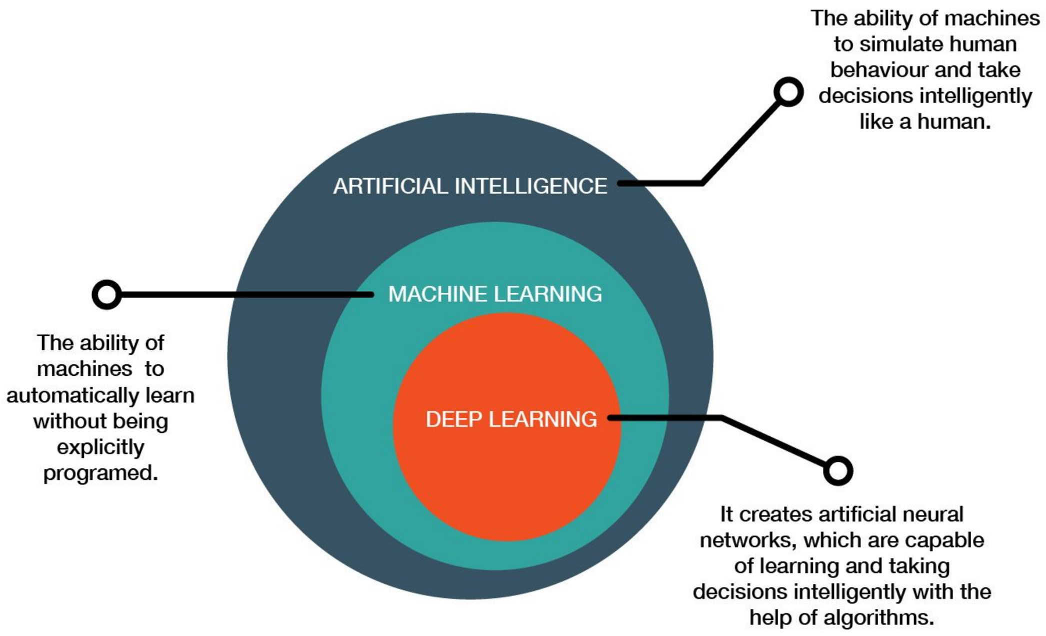
How do Machines learn? Well, the simple answer is, just like humans!

First, we receive information about a certain thing, which is kept in our mind. If we encounter the same thing in the future, our brain is able to identify it. Also, past experiences help us in making better decisions in the future. Our brain trains itself by identifying the features and patterns in knowledge/data received, thus enabling itself to successfully identify or differentiate between various things.

Machine learning is one of the most popular approaches in Artificial Intelligence. One of the key aspects of ML is the usage of new/continuous data to iterate and keep on learning.

There are many key industries where ML is making a huge impact: financial services, logistics, marketing and sales, health care, etc. It is expected that in a couple of decades, the mechanical repetitive tasks will be over. Machine learning and improvements in artificial intelligence techniques have made the impossible possible, from self-driving cars to cancer cell detection.

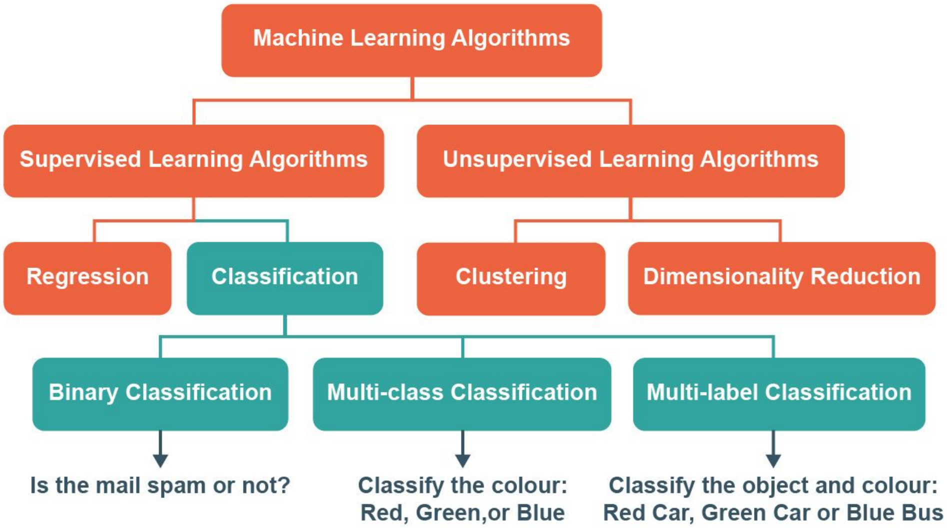
What is the difference between Machine Learning, Deep Learning and Artificial Intelligence? People often get confused between these terms, so we created an infographic, which will help you clear these concepts.


The ability of machines to simulate human behaviour and take decisions intelligently like a human.
ARTIFICIAL INTELLIGENCE
MACHINE LEARNING
The ability of machines to automatically learn without being explicitly programed.
DEEP LEARNING
It creates artificial neural networks, which are capable of learning and taking decisions intelligently with the help of algorithms.

Now, let us see how machine learning approaches different types of problems.

A machine learning algorithm can perform simple classification tasks and complex mathematical computations like regression. It involves the building of mathematical models that are used in classification or regression. To ‘train’ these mathematical models, you need a set of training data. This is the dataset over which the system builds the model.

The mathematical models are divided into two categories, depending on their training data: Supervised and unsupervised learning models.


Machine Learning Algorithms
Supervised Learning Algorithms
Unsupervised Learning Algorithms
Regression
Classification
Clustering
Dimensionality Reduction
Binary Classification
Multi-class Classification
Multi-label Classification
Is the mail spam or not?
Classify the colour: Red, Green,or Blue
Classify the object and colour: Red Car, Green Car or Blue Bus

1.1 Supervised Learning

Think of supervised learning as a kid learning multiplication tables. Tell the kid the multiplication table of 2, till 2 times 5. Then the kid deduces the logic that they have to add 2 to get the next answer in the multiplication table. Then, it says that 2 times 6 is 12. When building supervised learning models, the training data contains the required answers or the expected output. These required answers are called labels. For example, if the training data contains the technical indicators such as RSI and ADX, as well as the trading position to take such as buy or sell, then it is known as supervised learning approach.

With enough data points, the machine learning algorithm will be able to classify the trading signal correctly more often than not. Supervised learning models can also be used to predict continuous numeric values such as the share price of Disney. These models are known as regression models. In this case, the labels would be the share price of Disney.

Types of Supervised Models
Supervised models are trained on labeled dataset. It can either be a continuous label (regression) or categorical label (classification).

Regression Models Regression is used when one is dealing with continuous values such as the cost of a house when you are given features such as location, the area covered, historic prices etc. Popular regression models are:

- Linear Regression
- Lasso Regression
- Ridge Regression

Classification Models Classification is used for data that is separated into categories, with each category represented by a label. The training data must contain the labels and must have sufficient observations of each label. Some popular classification models include:

- Decision Trees Classifiers
- Random Forests Classifiers
- Neural Network Classifiers

There are various evaluation methods to find out the performance of these models. We will discuss these models and the evaluation methods in greater detail in later chapters.

Types of Classification There are mainly three types of classification.
Binary Classification

This type of classification has only two categories. Usually, they are Boolean values: 1 or 0 (sometimes known as True/False, or High/Low). Some examples where such a classification could be used are in cancer detection or email spam detection. In cancer detection, the labels would be positive or negative for cancer. Similarly, the labels for spam detection would be spam or not spam. You can also make trading a binary classification problem with labels such as buy and sell, or buy and no position.

Multi-class Classification
Multi-class classifiers or multinomial classifiers can distinguish between more than two classes. For example, you can have three labels such as buy, no position or sell. Multi-label Classification

This type of classification occurs when a single observation contains multiple labels. For example, a single image could contain a car, a truck and a human. The algorithm must be able to classify each of them separately. Thus, it has to be trained for many labels and should report ‘True’ for each of the objects i.e. a car, truck and human and ‘False’ for any other labels it has trained for.

Let us now look at the next type of machine learning approach.

1.2 Unsupervised Learning

In unsupervised learning, as the name suggests, the dataset used for training does not contain the required answers. Instead, the algorithm uses techniques such as clustering to group similar objects together.

The assumption in such a system is that the clusters discovered will match reasonably well with an intuitive classification. For example, the clustering of stocks based on historical data. This will result in clustering the stocks that belong to same sector or industry group together. There may also be some surprises, like Facebook and Twitter can be part of different group even though they belong to same sector.

Another application of unsupervised learning is anomaly detection. This uses a clustering algorithm to find out major outliers in a graph. These are used in credit card fraud detection and to detect market crash or black swan events. Apart from the two types of learning models mentioned earlier, Machine learning includes a third type which is known as reinforcement learning. Reinforcement learning involves methods that retro feed the model with rewards or punishment to improve performance. In order to improve performance, the model needs to be able to interpret the inputs correctly. Further, the model will decide on action and compare the outcome against a predefined reward system.

Reinforcement learning takes actions to maximise the rewards. This is not a supervised type of learning because it does not strictly depend on supervised or labelled data. And it is not unsupervised learning either since it modifies the model according to the final reward. The special feature of such models is that they try to learn from their mistakes and make better decisions in subsequent runs. A good example of this type of machine learning model can be the computer-controlled AlphaGo bot to play the board game Go. Similar machine learning agents can be used for trading an asset.

A few algorithms are a mix of supervised and unsupervised algorithms. Such algorithms combine a small amount of labelled data with a large amount of unlabeled data during training. For example, text-based classification (Natural Language Processing). The knowledge of the types of machine learning algorithms will help you appropriately select a model for a given problem statement.

To help you get started, we have created a step-by-step explanation of the different tasks carried out while creating and executing machine learning algorithms, in the next part of the book.

Part II
Machine Learning Implementation Framework

When you use machine learning, there are a few things you need to make sure before passing the information to an ML algorithm. In general, you will carry out the following tasks.

1. Defining the problem statement
2. Reading data
3. Data sanity checks
4. Setting target variable
5. Feature engineering
6. Test-train data split
7. Backtesting results and analyzing the performance of algorithm

Mr. Rob Read, a trader from Manhattan, is looking to use a machine learning algorithm that can guide him when to go long on J. P. Morgan stock. He goes to a data vendor to get the price data.

He then makes sure that the data quality is good. He does this by checking for missing and duplicate values in the data. He performs other analysis to ensure that the data can be used with the ML algorithm.

Now, Rob wants ML algo to predict when to go long on J. P. Morgan. He marks the days when the next day’s price is more than that day’s price as 1. And the rest of the days are marked as zero.
This is his target variable or what his ML algo will try to predict.

But what is required as input to predict the target variable? The inputs to the machine learning algorithm are called feature variables. He uses indicators such as RSI, MACD, and even momentum indicators such as ADX.

How does he evaluate if the machine learning model is effective? For that, he splits the data into train and test.
The train data is then used by the algorithm to learn the relationship between

the feature variables and the target. This relationship is then referred to as the ML model.

This way the algorithm will learn how the feature variables and target variables are related by using the train dataset. And he verifies the performance of the model on the test dataset.

Which metrics can be used to evaluate the model? Since he has created the target variable beforehand, he has the actual signals. He can simply compare the predicted signals of the ML model with the actual data.

For example, the ML model predicted a signal of 1 on 8th July 2021. This means that the ML model thinks that the price on 9th July 2021 would be higher than 8th July 2021.

He checked the price and sure enough, it had increased on 9th July. This means the ML program was correct in predicting the price moves for that day.
Finally, he generates signals through the machine learning model and backtests them. He plots the equity curve and drawdown to analyse the performance further.

This was a brief overview of the ML tasks. Let us deep dive into each of these steps and implement them in Python.

2 Defining the Problem Statement, Target and Feature

2.1 Defining the Problem Statement

A machine learning project or task should start by defining the problem statement. As the name suggests, a problem statement states the problem you want to solve. It is what you want as output from the ML algorithm.

But why do we need a well-defined problem statement?
A problem well defined is a problem half-solved.
— Charles Kettering

All your efforts would be devoted to solving this problem statement. In trading applications, the problem statement can be:

- Whether to buy or sell gold?
- What is the expected price of Tesla next day?
- How much capital should be allocated to different stocks in a portfolio?

For the purpose of this book, our problem statement will be, “Whether to buy J. P. Morgan’s stock at a given time or not?”
In technical jargon, the possible answers to these questions is the target variable.

2.2 Target Variable

Rob, a stock trader, has defined his problem statement as to whether he should buy or sell J. P. Morgan stocks? But how will the ML algorithm understand this logic? Rob defines buy when the expected returns next day is positive and sell if the expected returns next day is negative.
Rob takes the historical data of the returns and starts marking these days as buy and sell.

Next-Day Returns	Position
0.003	Buy
0.007	Buy
-0.002	Sell
0.02	Buy
-0.005	Sell

This column marked as buy and sell is called the target variable. In machine learning, you will denote a letter, “y” for the target variable.
Similarly, you can create a target variable based on the different problem statements you will be solving.

For instance, Mary wants to create an ML algorithm to predict J P Morgan’s stock price at the end of the next trading day. In this case, the target variable will be a series of close price data.

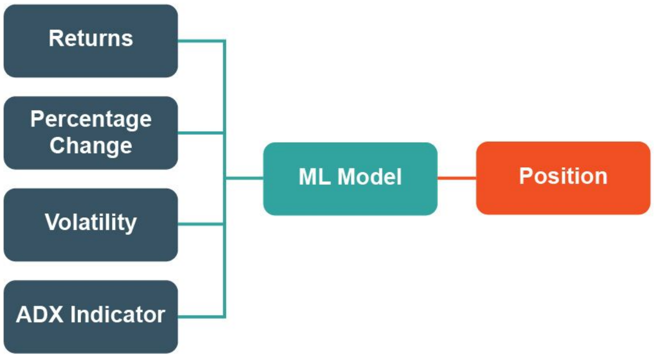
Alright! You have the target variable in mind. What next?
Let’s look at what is required by ML models to make this predictions.

2.3 Features

Let’s assume there is no machine learning model and Rob has to make this decision on his own.
What data points or information would Rob need in order to make this decision? Can you help Rob?
Probably Rob can look at the following things:

1. Trend in the price. Is it increasing or decreasing in the last few days?
2. Volatility of the stock
3. Values of technical indicators such as 14-day RSI and 50-day moving average

Let’s come back to the ML world. The information which Rob needs to predict to buy or sell J P Morgan’s stock, the same information is used by the ML model to make that decision.



Rob passes all this information to the ML model which in turn processes this information and tells Rob to buy or sell. The information, such as volatility, that Rob passed to the ML model are called features.

It is important to note that the information which Rob passed to the ML model needs to be of good quality. You would agree that poor quality of input will produce poor and misleading output. The common saying is, “garbage in garbage out”.

The original data may have incorrect data or missing values.
Rob has to clean the data and then feed it into the ML model.

Rob was excited and he started collecting all the data which he has, and prepared them to pass to ML model. On a whim, he also decides to collect the price history of apples for the same time period to use as a feature on his ML model.

How useful will this data be? It won’t be useful at all. The price of the fruit will have no predictive power for Apple Inc. the company’s stock price. Therefore, before passing any data to ML model, Rob needs to ensure that each feature is relevant and has some predictive power.

But even if the data is relevant, there is one important thing to check. Let’s see what this point is in the next section.

Most ML Models Are Based on Assumption of Stationary Input

Stationarity refers to the property that the observed value of the feature is independent of time. It means that the mean and the variance of the feature would be constant over time. For instance, a stock price, which has fluctuated between $8 and $9 ever since it started trading, is said to be stationary. Do note that this feature is not compulsory for all machine learning algorithms.

However, basic models cannot handle non-stationary features.

Let’s look at one more example. ML model learns that as S&P 500 index increases, the volatility index (VIX) decreases and vice versa. A basic machine learning algorithm, on seeing constant increase in SP500 price over the years, would start predicting VIX to crash into 0 and then go in the negative zone, whereas this will never be the case.

Rob has now understood this concept and has almost decided the features to be added. He finally wanted to include the stock’s volatility in his model. But what should he include in his model, weekly volatility or monthly volatility, or both? Since both these features, weekly and monthly volatility are highly correlated with each other, adding both of them will not add any extra information to the model.

Instead, keeping both of them together in the model might add more weight to the information carried by each of them individually. Hence, Rob decides to drop one of them. Thus, to make sure you get reliable results, you need to make sure that the above conditions are satisfied.

Let’s do a quick summary on features.

- Features are the input given to the ML model to make predictions.
- Each feature that is being passed to the ML model should be relevant and have some predictive power.
- Most ML models require stationary features in order to make predictions.
- Correlated features might add weight to the information carried by each of them individually. Hence, uncorrelated features are desired.

In the next chapter, we will see how you can create these targets and features in Python.

3 Target and Features in Python

In the previous chapter, you have seen how a machine learning algorithm requires a set of features to predict the target variable.
This chapter is specially created so that you can understand how to apply the knowledge you have learnt so far in a programming environment like Python.
Problem Statement

You have decided that you want to trade in J. P. Morgan. More specifically, you want to design an ML algorithm that will help you in deciding whether to go long in J. P. Morgan at a given point in time. Thus, the problem statement is:

Whether to buy J.P. Morgan’s stock at a given time or not?
To code the target and feature variables in Python, we will need some libraries or packages, and also the historical data for J P Morgan.

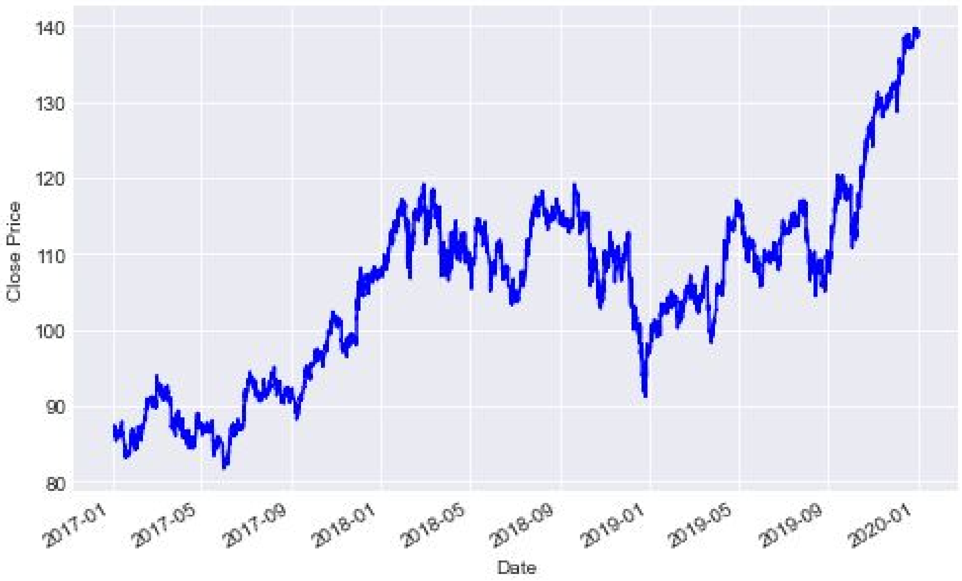
```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-darkgrid')
import talib as ta
from statsmodels.tsa.stattools import adfuller
```

Read the Data

All the data modules used in the book are uploaded on the following github link: https://github.com/quantra-go-algo/ml-trading-ebook. The 15-minute OHLCV data of J.P. Morgan stock price is stored in a CSV file JPM.csv in the data_modules directory. The data ranges from January 2017 to December 2019.

To read a CSV file, you can use the read_csv method of pandas. The syntax is shown below:
Syntax:
import pandas as pd
pd.read_csv(filename)
filename: Complete path of the file and file name in string format.

```
[2]: #The dataisstored inthedirectory 'data_modules'
path = "../data_modules/"

#Read thedata
data = pd.read_csv(path + 'JPM_2017_2019.csv',index_col=0) data.index =
pd.to_datetime(data.index)

data .close.plot(figsize=(8,5), color='b') plt.ylabel('Close Price')
plt.xlabel('Date')
plt.show()
```



Target Variable Target variable is what the machine learning model tries to predict in order to solve the problem statement. Remember that we had denoted it with the letter **y**.

Going back to our problem statement, **Whether to buy J.P. Morgan's stock or not?**, we will create a column, signal. The signal column will have two labels, 1 and 0.

Whenever the label is 1, the model indicates a **buy** signal. And whenever the label is 0, the model indicates a **do not buy** signal. We will assign 1 to the signal column whenever the future returns will be greater than 0.

The future_returns can be calculated using the pct_change method of pandas. The pct_change method will calculate the percentage change for the current time period. Remember how Rob at looked the next day's returns to form the target variables, buy or sell, for that day? It is the same concept over here.
Since we want future returns, we will shift the percentage change for the current period to the previous time period. This can be done using the shift method of pandas.

Syntax:
DataFrame[column].pct_change().shift(period)

Parameters: 1. **column:** The column for which the percentage change is to be calculated. 2. **period:** The period to shift the series. To shift to the current value to the previous time period, the period will be -1.

```
[3]: #Create acolumn 'future_returns' with the calculation of
#percentagechange
data['future_returns'] = data['close'].pct_change().shift(-1)

#Create thesignalcolumn
data['signal'] = np.where(data['future_returns'] > 0, 1, 0)
data.head()
```

```
[3]: open high low close volume \
2017-01-03 09:45:00+00:00 87.34 87.75 87.02 87.39 2184761.0
2017-01-03 10:00:00+00:00 87.39 87.44 86.95 87.19 1148228.0
2017-01-03 10:15:00+00:00 87.21 87.41 87.14 87.30 860609.0
2017-01-03 10:30:00+00:00 87.31 87.38 87.26 87.38 481605.0
2017-01-03 10:45:00+00:00 87.37 87.46 87.13 87.13 675950.0

future_returns signal
2017-01-03 09:45:00+00:00 -0.002289 0
2017-01-03 10:00:00+00:00 0.001262 1
2017-01-03 10:15:00+00:00 0.000916 1
2017-01-03 10:30:00+00:00 -0.002861 0
2017-01-03 10:45:00+00:00 -0.006312 0
```

As you can see in the above table, the close price from second row to third row is increased, and therefore the signal column in second row is marked as 1. In other words, if you buy when the signal column is 1, it is going to result in positive returns for you. Our aim is to develop a machine learning model which can accuratly forecast when to buy!

Features In order to predict the signal, we will create the input variables for the ML model. These input variables are called features. The features are referred to as **X**. You can create features in such a way that each feature in your dataset has some predictive power.

We will start by creating the 15-minute, 30-minute, and 75-minute prior percentage change columns.

```
[4]: #Create acolumn 'pct_change' with the 15-minute prior
#percentagechange
data['pct_change'] = data['close'].pct_change()

#Create acolumn 'pct_change2' with the 30-minute prior #percentagechange
data['pct_change2'] = data['close'].pct_change(2)

#Create acolumn 'pct_change5' with the 75-minute prior #percentagechange
data['pct_change5'] = data['close'].pct_change(5)
```

Next, we will calculate the technical indicators, RSI and ADX. These can be done by using the RSI and ADX method of the talib library.
Syntax:

```
import talib as ta
ta.RSI(data, timeperiod)
ta.ADX(data_high, data_low, data_open, timeperiod)
```

The parameters above are self-explanatory.
Since there are 6.5 trading hours in a day, and ours is a 15-minutes data, the time period will be 6.5*4.

```
[5]: #Create acolumnby thenameRSI,and assigntheRSI values toit data['rsi']
= ta.RSI(data['close'].values, timeperiod=int(6.5*4))
#Create acolumnby thenameADX,and assigntheADX values toit data['adx']
= ta.ADX(data['high'].values, data['low'].values, data['open'].values,
timeperiod=int(6.5*4))
```

We will now create the simple moving average and rolling correlation of the close price. This can be done by using the mean and the corr method of the pandas library. Syntax:

```
DataFrame[column].rolling(window).mean()
DataFrame[column].rolling(window).corr()
```

column: The column to perform the operation on. **window:** The span of the rolling window.
We will calculate the daily moving average and correlation.

```
[6]: #Create acolumnby thenamesma,and assignSMAvalues to it data['sma']
= data['close'].rolling(window=int(6.5*4)).mean()
```

```
#Create acolumnby thenamecorr,and assignthecorrelation #values toit
data['corr'] = data['close'].rolling(window=int(6.5*4))\

.corr(data['sma'])
```

Let us now calculate the volatility of the stock. This can be done by calculating the rolling standard deviation of the pct_change column.

```
[7]: #1-day and2-dayvolatility
data['volatility'] = data.rolling(
int(6.5*4), min_periods=int(6.5*4))['pct_change'].std()*100
data['volatility2'] = data.rolling(
int(6.5*8), min_periods=int(6.5*8))['pct_change'].std()*100
```

Create X and y

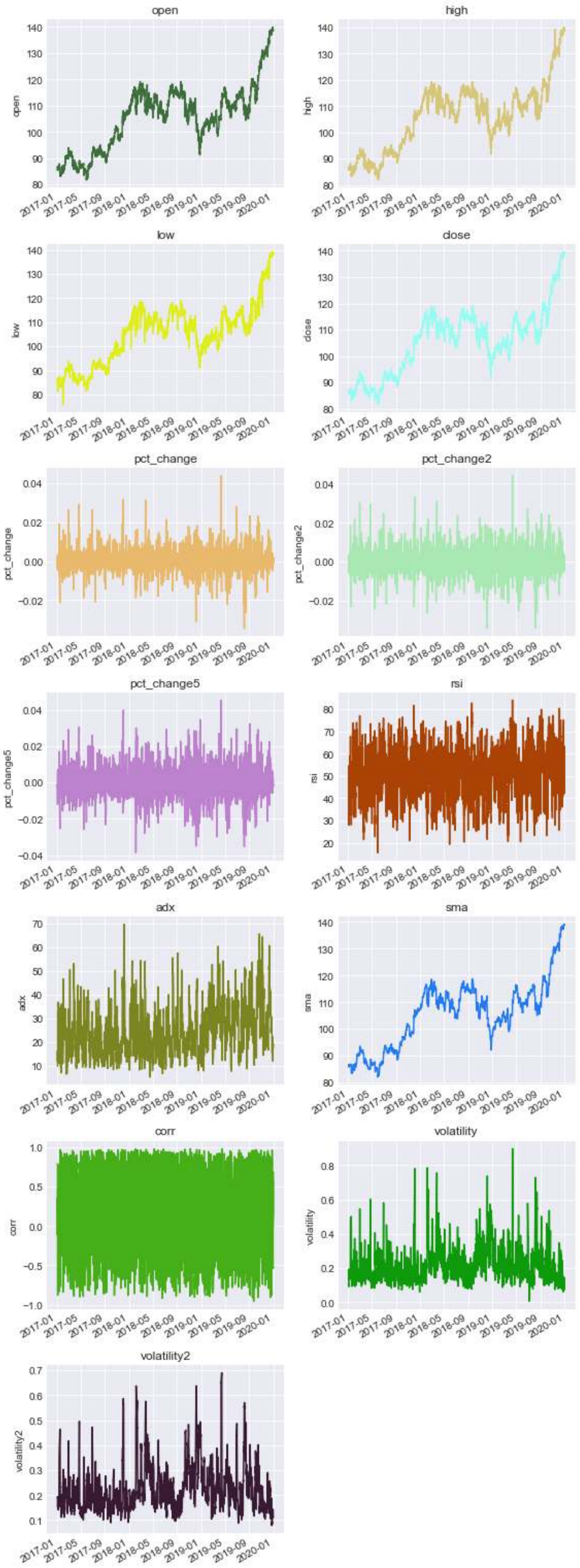
Before creating the features (X) and target(y), we will drop the rows with any missing values.

```
[8]: #Drop themissingvalues data.dropna(inplace=True)
```

Store the signal column in y and features in X. The columns in the variable X will be the input for the ML model and the signal column in y will be the output that the ML model will predict.

```
[9]: #Target
y = data[['signal']].copy()
#Features

X = data[['open','high','low','close','pct_change', 'pct_change2', 'pct_change5',
'rsi', 'adx', 'sma', 'corr', 'volatility', 'volatility2']].copy()

i=1

#Set numberofrows insubplot nrows = int(X.shape[1]+1/2) for feature in
X.columns:

plt.subplot(nrows, 2,i)
#Plot thefeature

X[feature] .plot(figsize=(8,3*X.shape[1]), color=np.random.rand(3,))
plt.ylabel(feature)
plt.title(feature)
i+=1
```

```
plt.tight_layout()
plt.show()
```



Stationarity Check

As you have seen that most ML algorithm requires stationary features, we will drop the non-stationary features from X.
You can use the adfuller method from the statsmodels library to perform this test in Python, and compare the p-value.

- If the p-value is less than or equal to 0.05, you reject H0
- If the p-value is greater than 0.05, you fail to reject H0

To use the adfuller method, you need to import it from the statsmodels library as shown below:

```
from statsmodels.tsa.stattools import adfuller
```

The adfuller method can be used as shown below:

```
result = adfuller(X)
```

The p-value can be accessed as result[1].

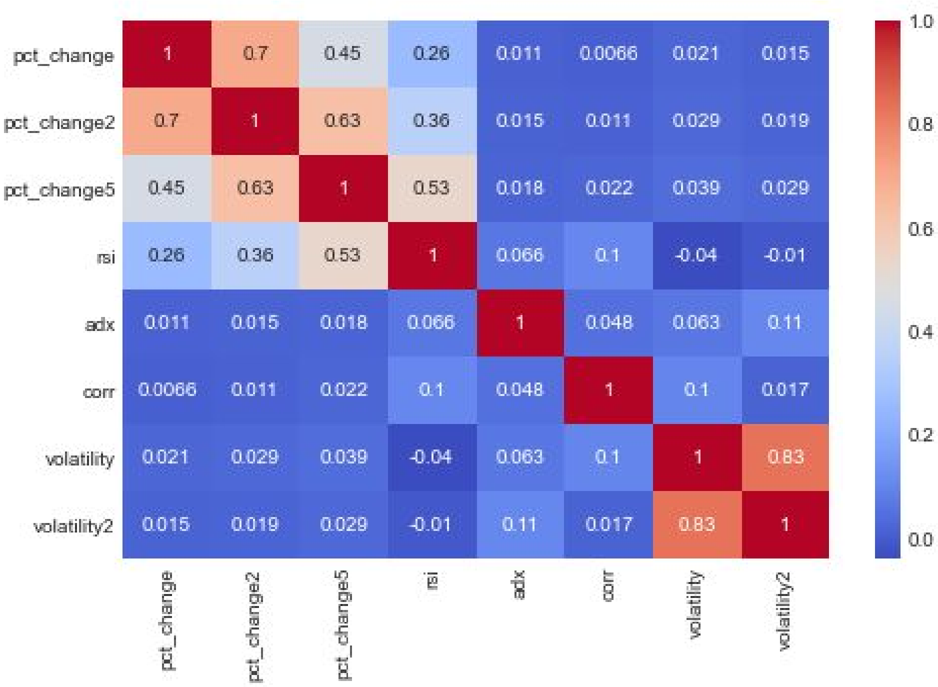
```
[10]: def stationary(series):
"""Function to check if the series is stationary or not. """

result = adfuller(series) if(result[1] < 0.05):
return 'stationary' else:
return 'not stationary'
#Check forstationarity
for col in X.columns:
if stationary(data[col]) == 'not stationary': print('%s is not stationary.
Dropping it.' % col) X.drop(columns=[col], axis=1,inplace=True)
```

```
open is not stationary. Dropping it.
high is not stationary. Dropping it.
low is not stationary. Dropping it.
close is not stationary. Dropping it.
sma is not stationary. Dropping it.
```

Thus, you can see that open, high, low, close, and sma are not stationary. They are dropped from the dataset.

Correlation You can check if two features have high correlation, then essentially one of the feature is redundant. We can remove that feature and improve learning of model.

```
[11]: import seaborn as sns
plt.figure(figsize=(8,5))
sns.heatmap(X.corr(), annot=True,cmap='coolwarm') plt.show()
```



In this exercise, we will define correlation to be high if it is greater than 0.7. The choice is subjective but it shouldn’t be very small, otherwise most of the features will look correlated. And at the same time not very high, otherwise none of the feature will be correlated. In the above output, you can see that the correlation between volatility and volatility2 is above threshold of 0.7. Hence, we should drop any one of the above columns. We will drop the volatility2 column. An automated code to get the correlated pair is below:

```
[12]: def get_pair_above_threshold(X, threshold):
"""Function to return the pairs with correlation above threshold.
"""
#Calculatethe correlation matrix
correl = X.corr()

#Unstack the matrix
correl = correl.abs().unstack()
```

```
#Recurring& redundantpair
pairs_to_drop = set()
cols = X.corr().columns
for i in range(0,X.corr().shape[1]):

for j in range(0,i+1):
pairs_to_drop.add((cols[i], cols[j]))
#Drop therecurring&redundant pair correl =
correl.drop(labels=pairs_to_drop) \ .sort_values(ascending=False)
return correl[correl > threshold].index
print(get_pair_above_threshold(X, 0.7))
MultiIndex([('volatility', 'volatility2')],
)
[13]: #Drop thehighlycorrelatedcolumn
X = X.drop(columns=['volatility2'], axis=1)
```

Once we have removed the features which are not stationary and are correlated, we will display the features which we have selected.

Display the Final Features

```
[14]: list(X.columns)

[14]: ['pct_change',
'pct_change2',
'pct_change5',
'rsi',
'adx',
'corr',
'volatility']
```

Great! We have the features and target variable with us. What next? In the next chapter, you will learn why it is important to split the data in test and train data set.

4 Train and Test Split

Ask anyone who has written exams involving math calculations before, and they will say that the questions seemed familiar. But they were not what they had studied.

This is because you want to test someone on their solving skills, and not memory. In a similar manner, you don’t train the machine learning algorithm by giving it the entire set of data. You keep a part for later, when you want to test if the ML algorithm has learned correctly.

In machine learning, the train-test split means splitting the data into two parts: 1. Training data (train_data)
2. Testing data (test_data)

The machine learning algorithm is trained on the train_data, and then it is applied to the test_data. The machine learning output is compared with the actual output for the test_data to evaluate how the model performs on ‘unseen’ data (data not known at the time of training).

Let us split our data into train and test data. In the first step, we will import the libraries and the dataset.

Import Libraries

```
[1]: #For datamanipulation import pandas as pd
#Import sklearn'strain-testsplit module from sklearn.model_selection import
train_test_split

#For plotting
import matplotlib.pyplot as plt %matplotlib inline
plt.style.use('seaborn-darkgrid')
```

Read the Data

For convenience, we saved the target (y) and features (X), prepared for J. P. Morgan in the previous sections, as csv files.

Let us first load these csv files using the read_csv method of the pandas library. [2]: #Read thefeatures

```
X = pd.read_csv('../data_modules/JPM_features_2017_2019.csv',
index_col=0,parse_dates=True)
#Read thetarget
y = pd.read_csv('../data_modules/JPM_target_2017_2019.csv',
index_col=0,parse_dates=True)
```

In the train-test split you divide the data into two parts.

But what proportion is the data split into?

Let us consider a student one day before university exams.

If the exam is for 10 chapters, and the student attempts the exam only after studying 2 chapters. This would probably result in a poor performance in the exam. In the machine learning context, this situation is analogous to under-learning. Under-learning is when the model is trained on a very small train_data.

For the university student to avoid under-learning, it can be recommended to study at least 7 or 8 out of the 10 chapters before appearing for the exam. Similarly in machine learning, you should select the train-test split proportion such that the training data is a fair representation of the whole data.

Say, if the train-test split is 80%-20%. It means 80% of the original data is the train_data and the remaining 20% is the test_data. The 80%-20% proportion is a popular proportion to split the data. But there is no rule of thumb that we always have to use the 80%-20% ratio.

You can also try other popular proportion choices like 90%-10%, 75%-25%. We will use a ready-made method called train_test_split from the sklearn module to perform the train-test split.

Syntax:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size, shuffle)
```

Parameters:

1. **X:** The features from the entire dataset.
2. **y:** The target values from the entire dataset.
3. **train_size:** The proportion of the training data.
4. **shuffle:** Parameter to specify shuffling of data.

Returns:

1. **X_train:** The features from the training dataset.
2. **X_test:** The features from the testing dataset.

3. **y_train:** The target values from the training dataset.
4. **y_test:** The target values from the testing dataset.

Let’s use the train_test_split to split the data in an 80% train and 20% test proportion.

```
[3]: #Obtain thefeaturesandtarget for the train_data and
# test_data
#The features (X)and the target(y)is passedalongwiththe #size
ofthetrain_dataas apercentageof thetotaldata
X_train, X_test, y_train, y_test = \

train_test_split(X, y, train_size=0.80)
#Print thedimensionsofthe variables

print (f"The shape of the X variable is {X.shape}.")
print(f"The shape of the y variable is {y.shape}. \n") print(f"The shape of the
X_train variable is {X_train.shape}.") print(f"The shape of the y_train
variable is {y_train.shape}.\n") print(f"The shape of the X_test variable is
{X_test.shape}.") print(f"The shape of the y_test variable is
{y_test.shape}.\n")
```

```
The shape of the X variable is (19317, 7).
The shape of the y variable is (19317, 1).
The shape of the X_train variable is (15453, 7). The shape of the y_train
variable is (15453, 1).
The shape of the X_test variable is (3864, 7). The shape of the y_test
variable is (3864, 1).
```

A few observations after the train-test split:

1. The dimensions of the original dataset show that there were 7 features and 19318 observations in the feature dataset (X). The target variable (y) has one column and the same number of observations as X.

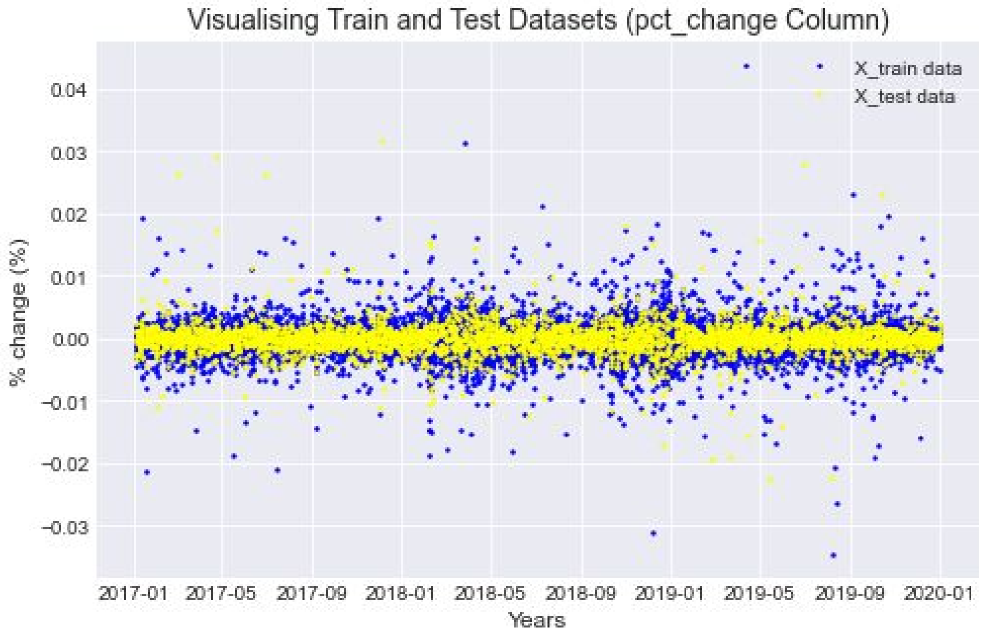
2. The dimensions of the train_data show that X_train has 7 features and 15454 observations. That is 80% of 19318, rounded down to the nearest integer. The target variable for the train data (y_train) has one column and the same number of observations as X_train.

3. The dimensions of the test_data show that X_test has 7 features and 3864 observations. That is the balance 20% of 19318. The target variable for the train data (y_test) has one column and the same number of observations as X_test.

Visualise the Data

Let’s plot one of the columns of the features to see how the data is split. [4]:

```
#Plot thedata
plt.figure(figsize=(8, 5))

plt .plot(X_train['pct_change'], linestyle='None',
marker='.',markersize=3.0,label='X_train data', color='blue')

plt .plot(X_test['pct_change'], linestyle='None',
marker='.',markersize=3.0,label='X_test data', color='yellow')

#Set thetitleand axislabel
plt.title("Visualising Train and Test Datasets (pct_change Column)",
fontsize=14)
plt.xlabel('Years',fontsize=12)
plt.ylabel('%change (%)',fontsize=12)

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



We can see that the train_data (blue points) and the test_data (orange points) are randomly shuffled.

Do you want randomly shuffled data for our train and test datasets? The answer depends on what type of data you are handling. If you are handling discrete observations, like the number of faulty products in a factory production line, then you can shuffle the indices for the train-test split.

But as seen in the above illustration, we are dealing with financial time-series data.

For time-series data, the order of indices matters and you cannot do random shuffling. This is because the indices in time series data are timestamps that occur one after the other (in sequence).

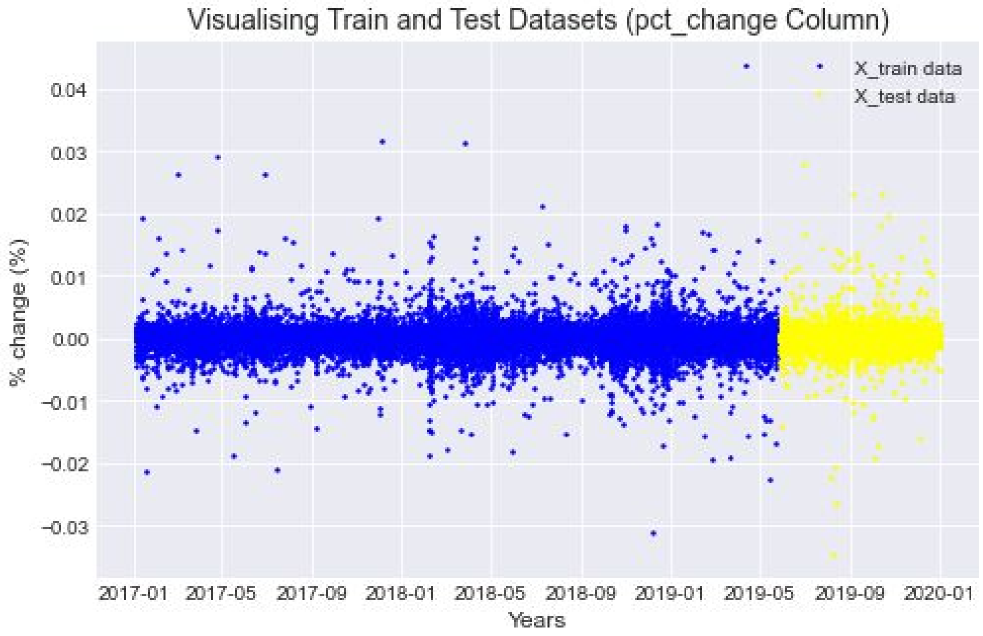
The data would make no sense if the timestamps are shuffled.

The reason for that is simple. You cannot use the data from 2021 to train your model, and then use the model to predict the prices in 2017. It is not possible in real life as we do not have access to future data.

Correct Way of Splitting Time-series Data To split the time-series data, we must not shuffle the datasets. We can specify the shuffle parameter to

False. It is set to True by default, so not specifying it in the method call results in a shuffled output.

```
[5]: #Obtain thefeaturesandtarget for the train_data and # test_data without
shuffling
X_train, X_test, y_train, y_test = train_test_split(

X, y, train_size=0.80,shuffle=False)
#Plot thedata
plt.figure(figsize=(8, 5))

plt .plot(X_train['pct_change'], linestyle='None',
marker='.',markersize=3.0,label='X_train data', color='blue')

plt .plot(X_test['pct_change'], linestyle='None',
marker='.',markersize=3.0,label='X_test data', color='yellow')

#Set thetitleand axislabel
plt.title("Visualising Train and Test Datasets (pct_change Column)",
fontsize=14)
plt.xlabel('Years',fontsize=12)
plt.ylabel('%change (%)',fontsize=12)

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



As seen on the plot, the train and test data points are not shuffled. The model is trained on train_data (blue part), and then the performance is evaluated for the test_data (orange part). The previous issue where we possibly were using future data to predict the past will not occur now. In this illustration, the model will be trained on data up to May 2019 (blue part), and then the model will be used to make predictions for the future.

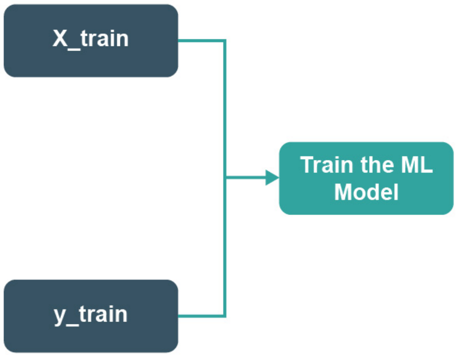
Now that you have learnt how to split the data, let’s try to train a model and use the trained model to make some predictions in the upcoming chapters.

5 Training and Forecasting using Classification Model

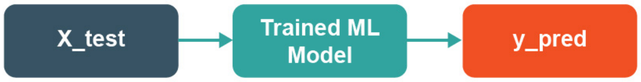
In the previous chapters, we learned about the features (X), target (y), and the train-test split.

As you have seen in countless movies, the hero always trains hard before he faces the competitor. Similarly, we have to train the ML algorithm too. So that when it goes in the real world, it will perform spectacularly.

We will use the X_train and y_train to train the machine learning model. The model training is also referred to as “fitting” the model.



After the model is fit, the X_test will be used with the trained machine learning model to get the predicted values (y_pred).



Import Libraries

```
[]: #For datamanipulation
import pandas as pd

#Import sklearn'sRandom Forest Classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

Read the Data

The target (y) and features (X) for the train and test dataset is read from the CSV files. Note that this data was prepared in the previous chapters.

```
[]: #Define thepathfor the datafiles path = "../data_modules/"
#Read thetargetand features ofthetrainingand testingdata X_train =
pd.read_csv(
path + "JPM_features_training_2017_2019.csv",index_col=0,
parse_dates=True)

X_test = pd.read_csv(
path + "JPM_features_testing_2017_2019.csv",index_col=0,
parse_dates=True)

y_train = pd.read_csv(
path + "JPM_target_training_2017_2019.csv",index_col=0,
parse_dates=True)

y_test = pd.read_csv(path + "JPM_target_testing_2017_2019.csv",
index_col=0,parse_dates=True)
```

Select a Classification Model

For illustration, we will use the RandomForestClassifier. Don’t worry if you are unfamiliar with this ML model.

It is not important to understand how the random forest classifier works at this time. We can use any other classification model in its place.

What is important here is to learn how the train_data and test_data are used along with the ML model.

The RandomForestClassifier model from the sklearn package is used to create the classification tree model. If you are very new to machine learning, you can skip the interpretation and understanding of these parameters for now.

Syntax:
RandomForestClassifier(n_estimators, max_features, max_depth, random_state) Parameters:

1. **n_estimators:** The number of trees in the forest.
2. **max_features:** The number of features to consider when looking for the best split.
3. **max_depth:** The maximum depth of a tree.
4. **random_state:** Seed value for the randomised bootstrapping and feature selection. This is set to replicate results for subsequent runs.

Returns:
A RandomForestClassifier type object that can be fit on the test data, and then used for making forecasts.

We have set the values for the parameters. These are for illustration and can be changed.

```
[]: #Create themachinelearningmodel
```

```
rf_model = RandomForestClassifier(
n_estimators=3,max_features=3,max_depth=2,random_state=4)
```

Train the Model

Now it is time for the model to learn from the X_train and y_train. We call the fit function of the model and pass the X_train and y_train datasets.

Syntax:
model.fit(X_train, y_train)
Parameters:

1. **model:** The model (RandomForestClassifier) object.
2. **X_train:** The features from the training dataset.
3. **y_train:** The target from the training dataset.

Returns:
The fit function trains the model using the data passed to it. The trained model is stored in the model object where the fit function was applied.

```
[]: #Fit themodelon thetrainingdata rf_model.fit(X_train, y_train['signal'])
[]: RandomForestClassifier(max_depth=2, max_features=3, n_estimators=3,
random_state=4)
```

Forecast Data

The model is now ready to make forecasts. We can now pass the unseen data (X_test) to the model, and obtain the model predicted values (y_pred). To make a forecast, the predict function is called and the unseen data is passed as a parameter.

Syntax:
model.predict(X_test)

Parameters: 1. **model:** The model (RandomForestClassifier) object. 2. **X_test:** The features from the testing dataset.
Returns: A numpy array of the predicted outputs is obtained.
Let’s make one prediction using the model. For illustration, we are using the first data point in the X_test.

```
[]: #Get asampleday ofthedata fromX_test unseen_data_single_day =
X_test.head(1)
#Preview the data unseen_data_single_day
```

[]: pct_change pct_change2 pct_change5 ↪rsi \
2019-05-28 12:00:00+00:00 0.0 -0.000091 0.001374 47. ↪746053
adx corr volatility 2019-05-28 12:00:00+00:00 26.139722 -0.515815
0.143024 This data is for 28th May, 2019. Let us pass this to the model and get the prediction. []: #Get thepredictionofa singleday

```
single_day_prediction = rf_model.predict(unseen_data_single_day)
#Preview the prediction
single_day_prediction
```

[]: array([0])

The predicted model output is 0. This means that the model is signaling to take no position on 28th May, 2019. Let's apply the model to all of the testing dataset. []: #Use themodeland predict thevaluesfor thetestdata

```
y_pred = rf_model.predict(X_test)
#Display the firstfivepredictions
print("The first five predicted values",y_pred[:5])
```

```
The first five predicted values [0 0 1 0 0]
```

The model predictions are stored in y_pred. 0 means no position and 1 means a long position. With the y_pred, we can now place trades using an ML model.

But how do we know that the ML model predictions are good?

As we can see, the model correctly predicts the first three values of the test_data. But how do we know the accuracy of the model prediction for the entire dataset? We need to learn some metric for measuring the model performance. This will be covered in the next chapter.

6 Metrics to Evaluate Classifier Model

In the previous chapter, you created a machine learning classifier model and asked yourself, **"How will I check if the machine learning model is effective and useful?"** And whether the machine learning model is able to learn the patterns from the data and make correct predictions?

One of the easiest ways to find out is how many times the predictions made by the ML model were correct. For example, if the ML model is right only 50% of the times, then it is not at all useful. As you can be right 50% of the times by making random decisions too.

For instance, the task is to predict the heads or tails in a coin toss. The model can predict heads every time and be right 50% of the time. Since there are only two choices, heads or tails, even a random guess will be right 50% of the time. Therefore, you need the model to be right more than 50% of the times.

On the other hand, if the ML model was right 8 out of 10 times, then it is right 80% of the times. And we can say that it is indeed able to find some patterns to map the input to output and not random. In technical jargons, this score of 80% is called accuracy.

What should be the desired threshold of the accuracy while using ML models for trading?

An accuracy score above 50% is satisfactory for an ML model. Aren't you interested in finding the accuracy of the machine learning model we had created in the previous chapter?

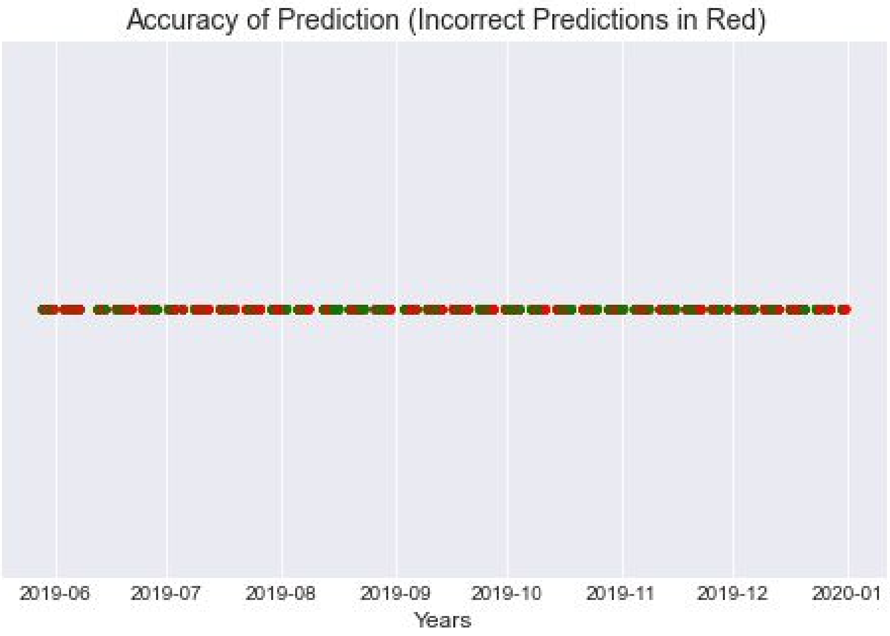
Let's find out now!
To do that, we will use the predicted output (y_pred) and the expected output (y_test).
We will import the libraries and read the model predicted values (y_pred) and the expected target values (y_test) from the test_data.

```
[1]: #For datamanipulation
import pandas as pd

#Librariesfor evaluating themodel
from sklearn.metrics import classification_report, \ confusion_matrix

#Librariesfor plotting
import matplotlib.pyplot as plt %matplotlib inline
plt.style.use('seaborn-darkgrid') import seaborn as sns
import matplotlib.colors as clrs

#Define thepathfor the datafiles path = "../data_modules/"
#Read themodelpredictedtarget values
y_pred = pd.read_csv(path + "JPM_predicted_2017_2019.csv",
index_col=0,parse_dates=True)['signal']
```

```
#Read thetargetvaluesof thetestingdataset
y_test = pd.read_csv(path + "JPM_target_testing_2017_2019.csv",
index_col=0,parse_dates=True)['signal']
```

Accuracy is nothing but the total correct predictions divided by the total predictions. We plot the data to see how the correct and incorrect predictions are distributed. The green points are where the predictions were correct and the red points are where the predictions were incorrect.

```
[2]: #Define theaccuracydata
accuracy_data = (y_pred == y_test)

#Accuracy percentage
accuracy_percentage = round(100 *
accuracy_data.sum()/len(accuracy_data),
2)

#Plot theaccuracydata plt.figure(figsize=(8, 5))
#Colour mapping forthecorrect and incorrectpredictions cmap =
clrs.ListedColormap(['green', 'red'])

plt .yticks([])
plt.scatter(x=y_test.index, y=[1]*len(y_test), c=(accuracy_data !=
True).astype(float),
marker='.',cmap=cmap)

#Set thetitleand axislabel
plt.title("Accuracy of Prediction (Incorrect Predictions in Red)",
fontsize=14)
plt.xlabel('Years',fontsize=12)

#Display the results
plt.show()
print(f"The accuracy is {accuracy_percentage}%.")
```



The accuracy is 51.55%.

The accuracy is calculated as seen above. These calculations for the accuracy and other performance metrics can be done using the ready-made classification_report method. You will learn about the classification_report method in the latter part of the chapter.

Is the accuracy score enough to conclude that the ML model is effective?

Probably not. Consider an ML model designed to predict whether you should buy or sell. This ML model has an accuracy of 73%. But while using this model, the sell signals are not great and making losses.

How is that possible for an ML model with 73% accuracy?

To get an answer to that question, you have to get the accuracy number label wise, or the action which the ML model will take. In this case, the accuracy for each buy and sell prediction. This will help you get a more granular view of how the model is performing.

It comes out that the ML model predicted 100 times that the price would go up, the price actually went up 90 times. And 10 times it actually fell.
So for predicting the “Buy” signal, the model is 90% accurate. That’s pretty

good.
ML Predicted Buy Actual price movement Price does not go up 10
ML Predicted Buy Price goes up 90

But for the sell signal. The model predicted 50 times that price would go down.The price went down only 20 times and 30 times it went up. The model is only 40% accurate in predicting the sell signal. Therefore, placing a sell order based on the model’s recommendation is bound to be disastrous.

ML Predicted Buy ML Predicted to Sell Actual price movement Price does not go up 10 20
Price goes up 90 30
The matrix shown here is called a confusion matrix.

Sometimes, there are jargons used to explain the matrix labels such as False Positive, True Positive, False Negative, and True Negative. In the above example, let’s say Buy is Positive and Sell is Negative.

On all the occasions where the ML model predicted buy correctly, we call that as True Positive. The number of times where the ML model was wrong in predicting Buy or positive, we call it as False Positive. False because the ML model was wrong in forecasting the positive label.

Similarly, can you tell what is a True Negative and False Negative?

The number of times ML model correctly predicted negative or the sell signal, is True Negative. And where the ML model was incorrect in the prediction of negative label or sell signal, is called a False Negative. This usage of terminology of positive and negative comes from medical science, where we run the test and the result is either positive or negative. But in some cases, there can be a misdiagnosis.

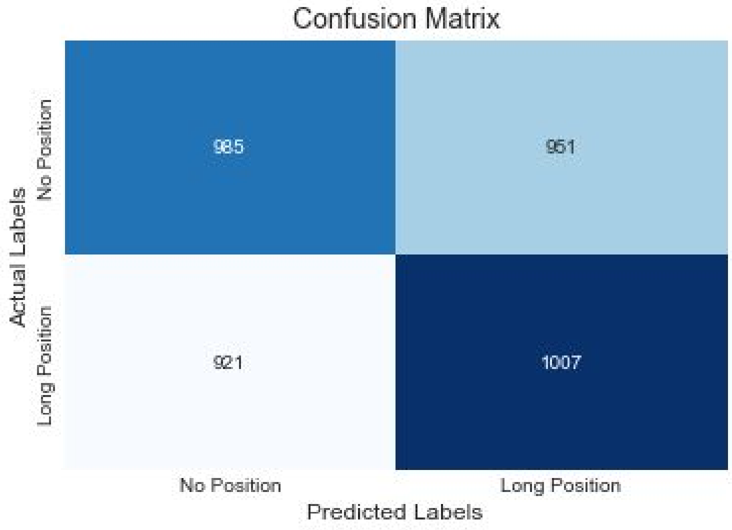
Let us see the confusion matrix of our machine learning model now. Syntax:
confusion_matrix(y_test, y_pred)
Parameters:

1. **y_test:** The observed target from the training dataset.
2. **y_pred:** The predicted target from the model.

Returns:

A numpy array of the confusion matrix.

```
[3]: #Define theconfusionmatrix
confusion_matrix_data = confusion_matrix(y_test.values,
y_pred.values)
#Plot thedata
fig, ax = plt.subplots(figsize=(6, 4))
sns.heatmap(confusion_matrix_data, fmt="d",
cmap='Blues',cbar=False,annot=True,ax=ax)

#Set theaxeslabels and thetitle
ax.set_xlabel('Predicted Labels',fontsize=12) ax.set_ylabel('Actual
Labels',fontsize=12)
ax.set_title('Confusion Matrix',fontsize=14)
ax.xaxis.set_ticklabels(['No Position', 'Long Position'])
ax.yaxis.set_ticklabels(['No Position', 'Long Position'])

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



The confusion matrix as seen above gives us the following information:

1. True Positive: 1007 correct predictions for taking a long position.
2. False Positive: 951 incorrect predictions for taking a long position when the expected action was no position.
3. True Negative: 985 correct predictions for taking no position.
4. False Negative: 921 incorrect predictions for taking no position when the expected action was to take a long position.

The confusion matrix helps us understand the effectiveness of the model, but it has its own limitation.
If you have more labels to classify, the confusion matrix grows. For example, 3 labels like buy, no position, and sell look like this.
Actual price movement ML Predicted Buy No position Sell

Price goes up 10 25 40
Price stays the same 90 25 20
Price goes down 80 40 30

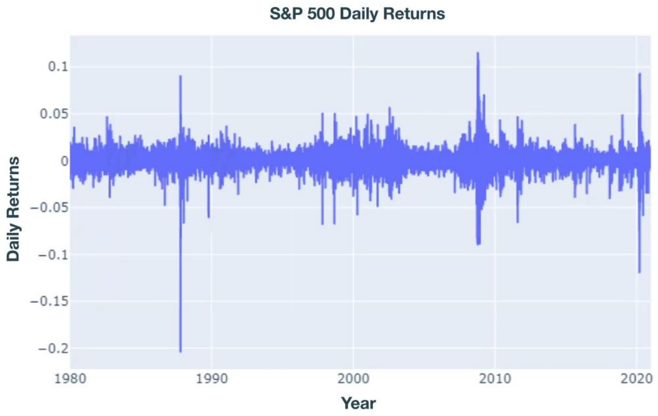
And if your labels are the quantity of shares, which can range from 0 to 10, to buy, then it will be a 11 by 11 table.
Thus, as the number of labels grows, it becomes difficult to interpret.
Actual Price Movement ML Predict

Buy 0 .. .. Buy 10 No increase 10 25 40 10 25 40 10 25 40 10 25 Increases 0.1% 25 40 10 25 40 10 25 40 10 25 10 Increases 0.2% 40 10 25 40 10 25 40 10 25 10 23 Increases 0.3% 25 40 10 25 40 10 25 10 45 5 15 Increases 0.4% 10 25 40 10 25 40 10 25 10 40 50 Increases 0.5% 25 40 10 25 40 10 25 10 45 5 15 Increases 0.6% 40 10 25 40 10 25 40 10 25 10 23 Increases 0.7% 10 25 40 10 25 40 10 25 10 40 50 Increases 0.8% 40 10 25 40 10 25 40 10 25 10 23 Increases 0.9% 25 40 10 25 40 10 25 10 45 5 15 Increases 1%+ 10 25 40 10 25 40 10 25 10 40 50

Are there other reasons we should not keep accuracy as the only metric?

Let’s take a slight diversion here and imagine that Rob wants to create an ML algorithm that would be helpful in times of stress or market fall, such as during the outbreak of the covid 19 pandemic.

He analysed the daily returns of the S&P 500 for the past 40 years.



Rob built an ML algorithm to predict when the S&P 500’s daily returns will be less than -5%. And based on the ML model prediction, Rob will short the SP500 futures. This ML model created by Rob showed accuracy of 99.8%. Whoa! He was on cloud 9!

But Mary, his friend and colleague, looked suspicious and took a deep dive into the data. The ML algorithm was run on 10,000 days from 1980 to 2021. The ML was correct 9980 times, resulting in accuracy of 99.8%.

She found that the S&P 500 went below 5% only 20 times.
And the algorithm predicted on all days including the 20 days to not short S&P 500 futures. On all 20 days where SP500 fell by 5%, the model was incorrect.

Mary explained the problem to Rob and advised him that there is a better metric to use here. It is to check how many times the model predicted that the market will fall by 5% correctly.

ML prediction

Price not below 5% Price below 5% Actual value Price not below 5% 9980 0
Price goes below 5% 20 0

In the example, it is zero out of 20 instances.
This is called recall metrics.
So Rob was delighted and said he will only use recall going forward. Mary

had to tell Rob that the recall metric as standalone is also not good. What if the model simply predicts to sell on all days?

ML prediction
Price not below 5% Price below 5% Actual value Price not below 5% 0 9980
Price goes below 5% 0 20
The recall for the market falling by 5% will be 20/20 or 100%, that is correct all the time. But you know this model is not useful.
Mary said you should also check the number of times the model gave the sell signal and it turned out right.
ML prediction

Price not below 5% Price below 5% Actual value Price not below 5% 0 9980
Price goes below 5% 0 20 Total 100

Rob realised that this value is 20 out of 10000 times, which is 0.002. This value is called precision.
Rob realised that both recall and precision are equally important.

He asked Mary if there’s a way to create a performance measure that can combine both?

Mary said Yes. It is called the f1 score. You can use the f1 score to understand the overall performance of the algorithm. The f1 score is the harmonic mean of precision and recall.

f
1
score
=
2
(precision$_{⇤}$recall) ⇤ (precision +recall)
Rob quickly calculated the f1 score for his model. It came out to 0.003.
f
1
score
=
2

$(1_{⇤}$ 0.002$^{)}$

⇤ (1 + 0.002) = $^{0.004\ =\ 0.003}$1.002

Let us look at the formulae for the different performance metrics:

Recall

=

Number of times the algorithm predicted an outcome correctly Total number of the actual outcomes

Precision

=

Number of times the algorithm predicted an outcome correctly Total number of said outcomes predicted by the algorithm

f1-score

=

$2_{⇤}$ (precision * recall) (precision + recall)

The scikit-learn library has a function called classification_report which provides measures like precision, recall, f1-score and support for each class. Precision and recall indicate the quality of our predictions. The f1-score gives the harmonic mean of precision and recall. The support values are used as weights to compute the average values of precision, recall and f1-score.

An f1-score above 0.5 is usually considered a good number.

You can simply use the following syntax to print the classification report.

Syntax:

classification_report(y_test, y_pred)

Parameters:

1. **y_test:** The observed target from the training dataset.
2. **y_pred:** The predicted target from the model.

Returns:

Classification Report containing precision, recall, f1-score and support. [4]:

```
#Classificationreport
classification_report_data = classification_report(y_test, y_pred) #Print
theclassification report print(classification_report_data)
precision recall f1-score support
00.52 0.51 0.51 1936 10.51 0.52 0.52 1928
```

```
accuracy 0.52 3864
macro avg 0.52 0.52 0.52 3864
weighted avg 0.52 0.52 0.52 3864
```

In the left-most column, you can see the values 0.0 and 1.0. These represent the position as follows:
1. 0 means no position.
2. 1 means a long position.

So from the table, you can say that the ML Model has an overall accuracy score of 0.52. The accuracy we calculated was 51.55% which is approximately 0.52. Apart from accuracy, you can identify the precision, recall, and f1-score for the signals as well.

Support is the number of actual occurrences of the class in the specified dataset. Thus, in the total signal, there were 1936 occurrences of 0, and 1928 occurrences of the 1 signal.

The accuracy score tells you how the ML model performed in total.
What are macro and weighted average?

Sometimes, the signal values might not be balanced. There could be instances where the number of occurrences for 0 is barely 50 while the number of occurrences for 1.0 is 500. In this scenario, the weighted average will give more weightage to the signal 1. In contrast, the macro average takes a simple average of all the occurrences.

Thus, the machine learning model’s performance can be analysed using the metrics you have learned in this notebook. Great! Now you know the metrics to analyse the performance, but what happens if we use this algorithm in the real world?

Let’s backtest this model and see how it would have performed!

7 Backtesting and Live Trading

So far you have seen different model evaluation metrics like accuracy, precision, recall and f1-score. After you are satisfied with the model performance, you can take the signals generated by them to trade and analyse the returns. Not only returns, but you should also analyse the risk associated with generating the returns.

This process is called backtesting.

Why are we backtesting the model? Think about it, before you buy anything, be it a mobile phone or a car, you would want to check the history of the brand, its features, etc. You check if it is worth your money. The same principle applies to trading, and backtesting helps you with it.

Do you know the majority of the traders in the market lose money?

They lose money not because they lack understanding of the market. But simply because their trading decisions are not based on sound research and tested trading methods.

They make decisions based on emotions, suggestions from friends, and take excessive risks in the hope to get rich quickly. If they remove emotions and instincts from the trading and backtest the ideas before trading, then the chance to trade profitability in the market is increased.

In backtesting, you are testing a trading hypothesis/strategy on the historical data.

In the previous chapters, you took the 15-minute, 30-minute, and 75-minute prior percentage change as your features and expected that these features will help you predict the future returns. This is your hypothesis.

How would you test this hypothesis? How would you know whether the strategy will work in the market or not?

By using historical data, you can backtest and see whether your hypothesis is true or not. It helps assess the feasibility of a trading strategy by discovering how it performs on the historical data.

If you backtest your strategy on the historical data and it gives good returns, you will be confident to trade using it. If the strategy is performing poorly on the historical data, you will discard or re-evaluate the hypothesis.

We will go through a few terms and concepts which will help us analyse our strategy. But we will do this simultaneously to see how our strategy performs. First, let us read the data files.
In the previous chapters, you have used a random forest classifier to generate the signal whether to buy J.P. Morgan's stock or not.
The signals are stored in a CSV JPM_predicted_2019.csv. We will use the read_csv method of pandas to read this CSV and store it in a dataframe strategy_data.

Also, we will read the close price of J.P. Morgan stored in a column close in the CSV file JPM_2017_2019.csv.
Further, we will store it in close column in strategy_data. While reading the close price data, we will slice the period to match the signal data.

```
[]: #For thedatamanipulation import numpy as np
import pandas as pd

#For plotting
import matplotlib.pyplot as plt %matplotlib inline
plt.style.use('seaborn-darkgrid')

#The dataisstored inthedirectory 'data_modules' path = '../data_modules/'
#Read theCSVfile using read_csvmethodof pandas
strategy_data = pd.read_csv(path + "JPM_predicted_2017_2019.csv",
index_col=0)

#Read thecloseprice
strategy_data['close'] = pd.read_csv(
path + "JPM_2017_2019.csv",index_col=0)\
.loc[strategy_data.index[0]:]['close']
strategy_data.index = pd.to_datetime(strategy_data.index)

#Preview the strategydata strategy_data.head()
```

```
[]: signal close
2019-05-28 12:00:00+00:00 0 109.29
2019-05-28 12:15:00+00:00 0 109.37
2019-05-28 12:30:00+00:00 1 109.33
```

```
2019-05-28 12:45:00+00:00 0 109.37
2019-05-28 13:00:00+00:00 0 109.38
```

The first thing you want to know from your strategy is what the returns are. Only then will you think if the trade made sense. So you will first calculate the strategy returns, as shown below.

Calculate Strategy Returns

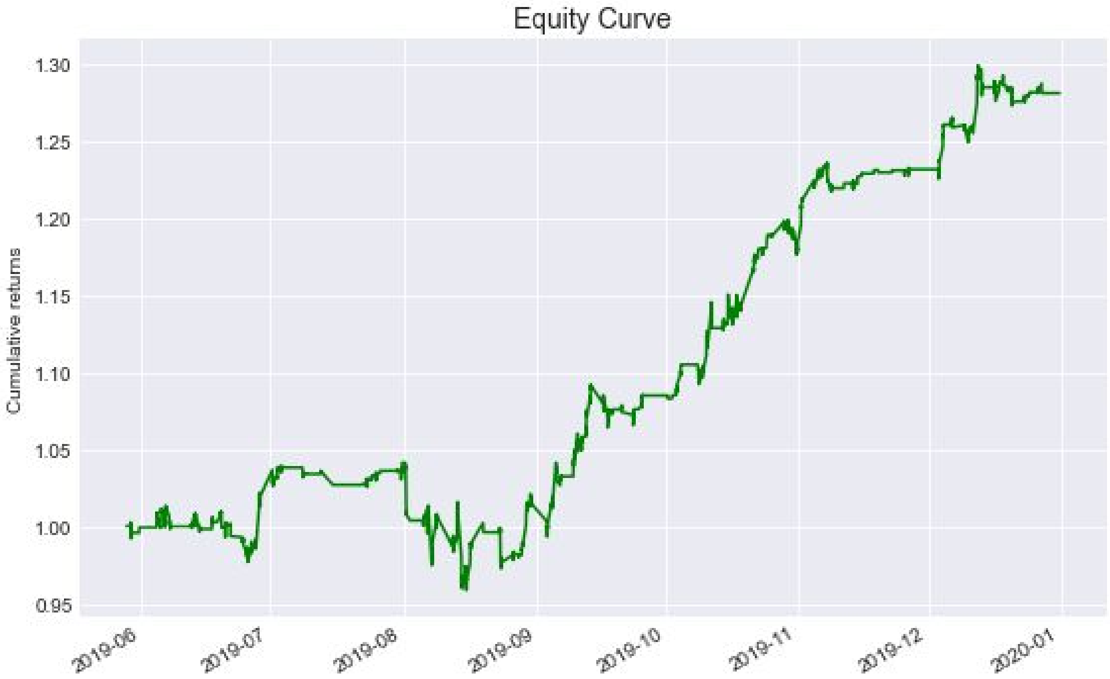
```
[]: #Calculatethe percentage change
strategy_data['pct_change'] = strategy_data['close'].pct_change()
#Calculatethe strategyreturns
strategy_data['strategy_returns'] = strategy_data['signal'].shift(1) * \
strategy_data['pct_change']
#Drop themissingvalues
strategy_data.dropna(inplace=True)
strategy_data.head()
```

```
[]: signal close pct_change strategy_returns
2019-05-28 12:15:00+00:00 0 109.37 0.000732 0.000000
2019-05-28 12:30:00+00:00 1 109.33 -0.000366 -0.000000
2019-05-28 12:45:00+00:00 0 109.37 0.000366 0.000366
2019-05-28 13:00:00+00:00 0 109.38 0.000091 0.000000
2019-05-28 13:15:00+00:00 0 109.37 -0.000091 -0.000000
```

The strategy returns help us understand the returns on a granular level. But now you want to visualise how the portfolio value has changed over a period of time. You can use the equity curve for this purpose.

Plot the Equity Curve You can plot the cumulative_returns columns of the strategy_data to obtain the equity curve.

```
[]: #Calculatethe cumulative returns
strategy_data['cumulative_returns'] = (
1+strategy_data['strategy_returns']).cumprod()

#--------------------Equity Curve-------------------#Plot cumulative strategy
returns
strategy_data['cumulative_returns'].plot(figsize=(8, 5),
color ='green') plt.title('Equity Curve',fontsize=14)
plt.ylabel('Cumulative returns')
plt.tight_layout()
plt.show()

cumulative_returns = (strategy_data['cumulative_returns'][-1] - 1)\ *100
print("The cumulative return is {0:.2f}%.".format(cumulative_returns))
```



```
The cumulative return is 28.10%.
```

From the above output, you can see that the strategy generated a cumulative returns of 28.10% in seven months. That is impressive. But this is not the only metric we need to see. Let us dive futher to understand the performance of your strategy in detail.

You can analyse the returns generated by the strategy and the risk associated with them using different performance metrics.

Annualised Returns The annualised return is the geometric average amount of money earned by an investment each year over a given time period. It shows what strategy would earn over a period of time if the annual return was compounded. It is calculated using the below formula:

Annualised Returns
=(

252

Cumulative Returns

no
.
of

15
⇤
6.5⇤4
minute datapoints
) 1

Note: There are approximately 252 trading days in a year, and 6.5 trading hours in a day. Since we are working with 15-minute data, the number of trading frequencies in a year is 252⇤ 6.5⇤ 4.

And the numerator in the exponent term is 252⇤ 6.5⇤ 4.

```
[]: #Calculatethe annualised returns
annualised_return = ((strategy_data['cumulative_returns'][-1]) \ **
(252*6.5*4/strategy_data.shape[0]) - 1)\ * 100

print("The annualised return is {0:.2f}%.".format(annualised_return))
```

```
The annualised return is 52.20%.
```

From the above output, you can see that the average annual return of the strategy is 52.20%. But how volatile is the asset. You can find that out using annual volatility.

Annualised Volatility Volatility is the measure of risk. It is defined as the standard deviation of the returns of the investment. Annualised volatility can be calculated by multiplying the daily volatility with the square root of the number of trading days in a year.

Annualised Volatility = Var(Returns)⇤p252⇤ 6.5⇤ 4

p

```
[]: #Calculatethe annualised volatility
annualised_volatility = strategy_data['strategy_returns']\
.std()*np.sqrt(252*6.5*4) * 100 print("The annualised volatility is
{0:.2f}%." \ .format(annualised_volatility))
```

The annualised volatility is 14.90%.
Annualised volatility of 14.90% means that for approximately 68% time in a year, the current time's price would differ by less than 14.90% from the previous time.

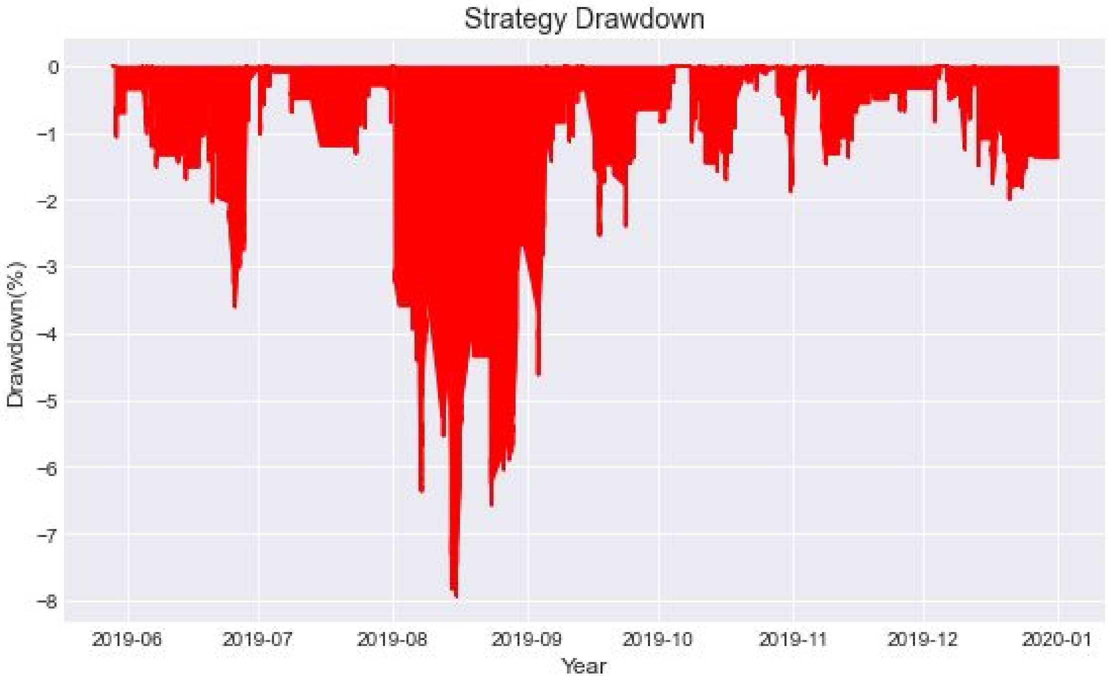
This is interesting. But volatility as a term treats positive and negative terms as the same. You would like to know how much the portfolio can go in the negative territory. You can check that using maximum drawdown.

Maximum Drawdown Maximum drawdown is the maximum value a portfolio lost from its peak. It is the maximum loss the strategy can make. Higher the value of the drawdown, higher would be the losses. It is calculated as below:

Maximum Drawdown
=
Trough Value Peak Value Peak Value

```
[]: #Calculatethe runningmaximum
running_max = np.maximum.accumulate(
strategy_data['cumulative_returns'].dropna())

#Ensure thevalueneverdrops below 1
running_max[running_max < 1] = 1
#Calculatethe percentage drawdown
drawdown = ((strategy_data['cumulative_returns'])/running_max - 1)\

* 100

#Calculatethe maximumdrawdown
max_dd = drawdown.min()
print("The maximum drawdown is {0:.2f}%.".format(max_dd))

#---------------------DD plot-------------------fig = plt.figure(figsize=(8, 5))

#Plot themaximumdrawdown
plt.plot(drawdown, color='red')
#Fill in-between thedrawdown
plt.fill_between(drawdown.index, drawdown.values, color='red')
plt.title('Strategy Drawdown',fontsize=14)
plt.ylabel('Drawdown(%)',fontsize=12)
plt.xlabel('Year',fontsize=12)

plt.tight_layout() plt.show()
```

```
The maximum drawdown is -7.94%.
```



From the above output, you can see that the maximum drawdown is 7.94%. This means that the maximum value that the portfolio lost from its peak was 7.94%. As with any investment, you can calculate the Sharpe ratio as well, to understand how well the strategy performs.

Sharpe Ratio Sharpe ratio measures the performance of a portfolio when compared to a risk-free asset. It is the ratio of the returns earned in excess of the risk-free rate to the volatility of the returns. It is calculated as below:

Sharpe ratio

=

R_p R_f

s_p

where,

• R_p is the return of the portfolio.

• R_f is the risk-free rate.

- s_p is the volatility.

A portfolio with a higher Sharpe ratio will be preferred over a portfolio with a lower Sharpe ratio.

```
[]: #Calculatethe Sharperatio
sharpe_ratio = round(strategy_data['strategy_returns']\
.mean()/strategy_data['strategy_returns']\ .std() * np.sqrt(252*6.5*4), 2)

print("The Sharpe ratio is {0:.2f}.".format(sharpe_ratio))
The Sharpe ratio is 2.89.
```

The Sharpe ratio of 2.89 indicates that the returns are pretty good when compared to the risk associated.

Note that to keep the chapter simple, the transaction cost and slippage were not considered while analysing the performance of the strategy. But this is an important concept too. Also, since the risk free rate depends on the region as well as the time period, we have preferred to keep it 0. You can of course plug it in the formula to calculate it yourself.

How should you define risk metrics for yourself?

Volatility and maximum drawdown are the standard measures of risk. If you are concerned about the maximum loss a strategy can incur over a period of time. Then you can use maximum drawdown.

Traders also use the Sharpe ratio as it provides information about the returns per unit risk. So, it is using both factors, risk and returns.

Backtesting vs Walk Forward Trading Testing Backtesting a strategy gives you a good understanding of what happened in the past, but it’s not a predictor of the future. Walk forward testing is a better approach which to some extent, can tell the future.

In the walk forward testing method, we divide the historical data in the training (insample) and testing (out-of-sample) dataset. On the training dataset, we optimise the trading parameters and check the performance of the strategy on the testing datasets.

Suppose you have ten years of data.
You take the first three years of data as train and keep the 4th year as test. You then assess the performance for the 4th year using various performance metrics.

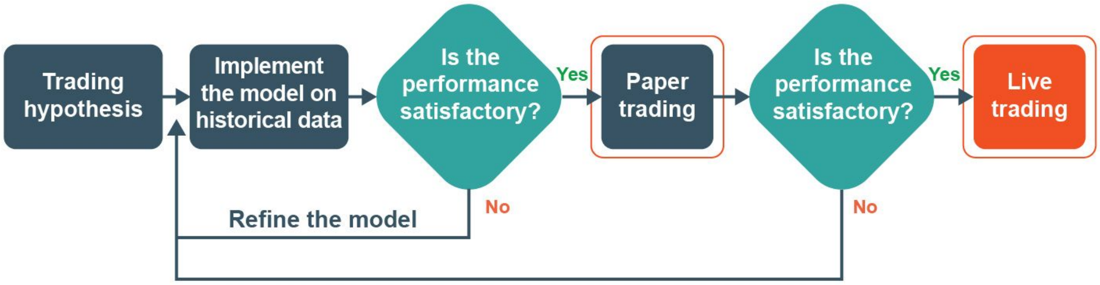
Next, you repeat the optimisation using data from years 2–4, and validate using month 5. You keep repeating this process until you've reached the end of the data. You collate the performances of all the out-of-sample data from year 4 to 10, which is your out-ofsample performance.

Paper Trading & Live Trading You created the strategy and analysed the performance of the strategy.

Can you directly start a paper or live trading?
When should you consider your strategy for paper trading or live trading?

If you are satisfied with the backtesting strategy performance, then you can start paper trading. If not, you should tweak the strategy until the performance is acceptable to you. And once the paper trading results are satisfactory, you can start live trading.

Process of Paper trading and Live trading


Trading hypothesis
Implement the model on historical data
Is the performance satisfactory?
Yes
Paper trading
Is the performance satisfactory?
Yes
Live trading
No
Refine the model
No

How many backtests should you do before taking a strategy live?

There is no fixed number. You can take your strategy live after backtesting once or it can be after multiple backtesting. As we mentioned in the previous question, once you are satisfied with the backtesting results, you can consider your trading strategy for paper trading and live trading.

A good backtester should be aware of certain drawbacks/biases which might drastically change your backtesting results. Let’s look at them one by one.

Overfitting/Optimisation Bias Backtesting, like any other model, is prone to overfitting. While testing the model on historical data, you inadvertently try to fit the parameters to get the best results.

You get the best result on the historical dataset, but when you deploy the same model on the unseen dataset, it might fail to give the same result.
The best way to avoid overfitting is:

- To divide the dataset into train and test datasets.
- You backtest your trading strategy on the training dataset and run your strategy on the test dataset with the same parameters that you used on the training dataset to ensure the effectiveness of the strategy.

Look-ahead Bias Look-ahead bias is the use of information in the analysis before the time it would have actually occurred.

While devising a strategy, you have access to the entire data. Thus, there might be situations where you include future data that was not able in the time period being tested.

A seemingly insignificant oversight, such as assuming that the earning report being available one day prior, can lead to skewed results during the backtesting. You need to make sure you are not using data that will only be available in the future to avoid look-ahead bias.

Survivorship Bias During backtesting the strategy, you often tend to backtest a strategy on the current stock universe rather than the historical stock universe. That is, you use the universe that has survived until today to backtest.

There is a famous example that is used to illustrate the survivorship bias.

If you were to use stocks of technology companies to formulate a strategy, but took the data after the dot com bubble burst, it would present a starkly different scenario than if you had included it before the bubble burst.

It’s a simple fact, after the year 2000, the companies which survived did well because their fundamentals were strong, and hence your strategy would not be including the whole universe. Thus, your backtesting result might not be able to give the whole picture.

Ignoring Trading Costs It is crucial to incorporate all kinds of commissions, taxes and slippages while backtesting. It is highly probable that the strategy performs well without these costs, but it drastically affects the appearance of a strategy’s profitability after the inclusion of these costs.

Backtesting Software There are platforms available that provide the functionality to perform backtesting on historical data. The important points to consider before selecting a backtesting platform are:

- Which asset classes does the platform support?
- Sources of the market data feeds it supports
- Which programming languages can be used to code the trading strategy which is to be tested?

Some of the common backtesting software and live trading software are:

1. Blueshift
2. MetaTrader
3. Amibroker
4. QuantConnect
5. Quanthouse, etc.

While this has already been said before, it is not necessary that the past is always representative of the future.

Hence, it is important to note that you should not over-rely on backtesting. For example, the COVID pandemic in 2020 was the first of it’s kind and had an impact on global markets.

The rules identified on the historical data might not have performed well during this pandemic.

Great! You have succesfully built your own machine learning algorithm and not only have you tested it on historical data, but also analysed its results.

Before you go ahead and start live trading, there might be some things you need to care of first. In the next section, we will answer some of the most common questions when it comes to applying the machine learning model in live markets.

8 Challenges in Live Trading

When you are trying to deploy a machine learning model in live trading, you will need to answer some practical questions such as:

- How do I save a model for later use?
- How do I use an already saved model?
- How to handle the data?
- When should I retrain the model?

Let us go through them one by one.

How to save a model? Training machine learning models takes a lot of time and resources. So, most traders train their models on weekends, if the model works on daily data. Or if the model works on intraday data, they train it after the close of market.

In both the cases, traders use the latest data available to train their models and then save it. These models are later retrieved and used to make predictions while trading. This process saves both time and resources. You can use the pickle library to save a model once it is trained.

The pickle module implements a fundamental, but powerful algorithm for saving and loading a Python object structure. The process of saving is known as “Serialization” and the process of loading is called “De-serialization”.

Saving Serialization Loading De-Serialization

When you save any object using pickle, that object will be converted into a byte stream of 1s and 0s, and then saved. When you want to load a pickled object then an inverse operation takes place, whereby a byte stream is converted back into an object. There are a couple of things that you need to remember when saving an object using pickle.

Things to remember while serialization or deserialization
Python Version Compatibility

When (de)serializing objects you need to use the same version of Python, as the process used is different in different versions and this might result in errors. Security
Pickle data can be altered to insert malicious code, so it is not recommended to restore data from unauthenticated sources.

Let us save the model. Before you save the model, you need to decide two parameters: What do you want to save? How do you want to save it?

In the code shown here, we have created a simple function called save_model which takes these two parameters as inputs and saves the model.

In a simulation, we will be saving the trained model multiple times. So we create a function called save_model to handle this repetitive step. This function takes the model name and model's saved name as its input, and saves the model in a binary file since the machine could be local, remote or a de-located file system.

Right now, we will assume we are saving the file on a local system.

The save_model function opens a file in the local machine using the variable model_pickled_name. Here the keyword 'wb', or 'write binary', implies that Python will overwrite the file, if it already exists or creates a new one if it doesn't. Then the dump command of the pickle library is used to write the model to the specified destination.

Apart from the pickle library you can also use joblib and JSON libraries to save your models. The joblib library is very efficient compared to the pickle library when saving objects containing large data. On the other hand, JSON saves a model in a string format which is easier for humans to read.

```
[]: def save_model(model_name, model_saved_name):
#Open afilewith thementionednameon thelocalmachine with
```

```
open(model_saved_name, 'wb') as model_save:

#Use thedumpcommand to saveit
pickle.dump(model_name, model_save)
```

In this example, we run this save_model function and it saves the model in the local machine with the name model_pickle as shown.

```
[]: model_saved_name = 'model_save.pkl' save_model(model,
model_saved_name)
```

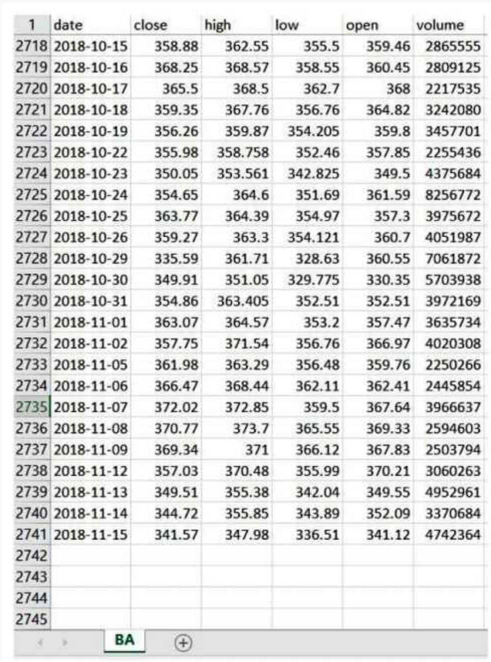
Load a Model Once you have saved the model, you can access the model on your local machine by using the load_model function. This function takes the name of the pickled model as its input and loads that model.

We will be loading the trained model at every data point.

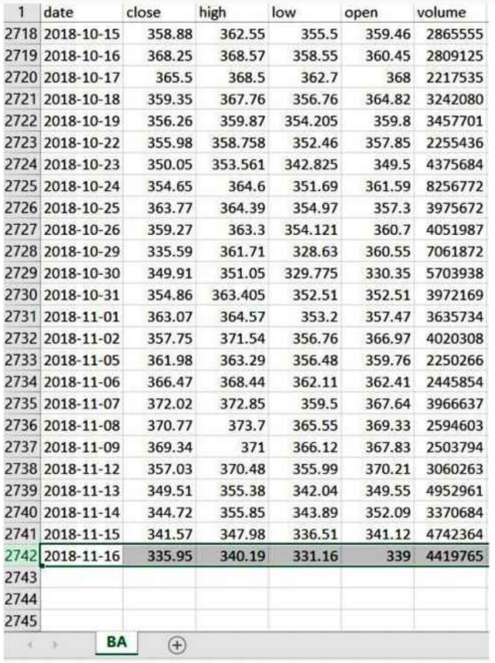
```
[]: def load_model(model_saved_name):
#Open thefilecontainingthe model withthementioned name #on
thelocalmachine
with open(model_saved_name, 'rb') as file:

#Load themodeland assign itto avariable model = pickle.load(file)
#Return themodel
return model
```

How to handle the data? To train machine learning based trading models we require a lot of data. Downloading data from an online source every time you want to train a model takes a lot of time. To avoid this, the old data that you used to train the initial model must be saved on your local machine, and the new data can be added to this file at the end of trading everyday. The new data can be appended to the existing data using the pandas append function.

```
[]: Updated_data = Old_data.append(current_day_OHLC ,
ignore_index=True) Updated_data.to_csv("Data.csv")
```



1	date	close	high	low	open	volume
2718	2018-10-15	358.88	362.55	355.5	359.46	2865555
2719	2018-10-16	368.25	368.57	358.55	360.45	2809125
2720	2018-10-17	365.5	368.5	362.7	368	2217535
2721	2018-10-18	359.35	367.76	356.76	364.82	3242080
2722	2018-10-19	356.26	359.87	354.205	359.8	3457701
2723	2018-10-22	355.98	358.758	352.46	357.85	2255436
2724	2018-10-23	350.05	353.561	342.825	349.5	4375684
2725	2018-10-24	354.65	364.6	351.69	361.59	8256772
2726	2018-10-25	363.77	364.39	354.97	357.3	3975672
2727	2018-10-26	359.27	363.3	354.121	360.7	4051987
2728	2018-10-29	335.59	361.71	328.63	360.55	7061872
2729	2018-10-30	349.91	351.05	329.775	330.35	5703938
2730	2018-10-31	354.86	363.405	352.51	352.51	3972169
2731	2018-11-01	363.07	364.57	353.2	357.47	3635734
2732	2018-11-02	357.75	371.54	356.76	366.97	4020308
2733	2018-11-05	361.98	363.29	356.48	359.76	2250266
2734	2018-11-06	366.47	368.44	362.11	362.41	2445854
2735	2018-11-07	372.02	372.85	359.5	367.64	3966637
2736	2018-11-08	370.77	373.7	365.55	369.33	2594603
2737	2018-11-09	369.34	371	366.12	367.83	2503794
2738	2018-11-12	357.03	370.48	355.99	370.21	3060263
2739	2018-11-13	349.51	355.38	342.04	349.55	4952961
2740	2018-11-14	344.72	355.85	343.89	352.09	3370684
2741	2018-11-15	341.57	347.98	336.51	341.12	4742364
2742						
2743						
2744						
2745						



1	date	close	high	low	open	volume
2718	2018-10-15	358.88	362.55	355.5	359.46	2865555
2719	2018-10-16	368.25	368.57	358.55	360.45	2809125
2720	2018-10-17	365.5	368.5	362.7	368	2217535
2721	2018-10-18	359.35	367.76	356.76	364.82	3242080
2722	2018-10-19	356.26	359.87	354.205	359.8	3457701
2723	2018-10-22	355.98	358.758	352.46	357.85	2255436
2724	2018-10-23	350.05	353.561	342.825	349.5	4375684
2725	2018-10-24	354.65	364.6	351.69	361.59	8256772
2726	2018-10-25	363.77	364.39	354.97	357.3	3975672
2727	2018-10-26	359.27	363.3	354.121	360.7	4051987
2728	2018-10-29	335.59	361.71	328.63	360.55	7061872
2729	2018-10-30	349.91	351.05	329.775	330.35	5703938
2730	2018-10-31	354.86	363.405	352.51	352.51	3972169
2731	2018-11-01	363.07	364.57	353.2	357.47	3635734
2732	2018-11-02	357.75	371.54	356.76	366.97	4020308
2733	2018-11-05	361.98	363.29	356.48	359.76	2250266
2734	2018-11-06	366.47	368.44	362.11	362.41	2445854
2735	2018-11-07	372.02	372.85	359.5	367.64	3966637
2736	2018-11-08	370.77	373.7	365.55	369.33	2594603
2737	2018-11-09	369.34	371	366.12	367.83	2503794
2738	2018-11-12	357.03	370.48	355.99	370.21	3060263
2739	2018-11-13	349.51	355.38	342.04	349.55	4952961
2740	2018-11-14	344.72	355.85	343.89	352.09	3370684
2741	2018-11-15	341.57	347.98	336.51	341.12	4742364
2742	2018-11-16	335.95	340.19	331.16	339	4419765
2743						
2744						
2745						

On the left you can see the old data file, and on the right you can see the updated data file after adding the new data.

When do you retrain the model? We need to retrain a model whenever its performance goes bad.

You can decide when to retrain a model based on its performance metrics such as:

1. Capital Loss Let us say that you want to retrain a model based on its capital loss. Then you need to track the profit and loss (or PnL) of the strategy at every time period, such as everyday or a minute.

If the PnL falls below a certain limit, then you will retrain it.

If the model has initially made a profit of 100 dollars, and then it has lost 5 dollars, which is the cutoff criteria in this case.

After the cutoff criterion is triggered, we will stop trading and then retrain the model. This cutoff criteria is decided by a trader, depending on his or her own risk appetite.

2. Accuracy This is another criteria that can used to decide whether to retrain a model or not.

Let us say that you have set 55% accuracy as the criterion for retraining a model. Whenever the model’s accuracy falls below the 55% mark, you retrain it.

In addition to these two approaches, you can retrain your model as often as possible, regardless of the model's performance. However, make sure your model is not overfitted.

This will create a model that is trained on the latest available data at all times. When you want to retrain a model, you need to perform many tasks such as creating the features, training the model and saving it.
To do these multiple tasks, we created a simple function called create_new_model. This function takes the raw data and the saved name of the model as input.

```
[]: def create_new_model(data, model_saved_name):
#Create afeaturefromthe rawdata
X, y = create_features(data)
#Train themodelon the featuresgenerated
model = train_model(X, y)
#Save themodelon thelocalmachine
save_model(model, model_saved_name)
```

In this way, you should take care that your machine learning model is performing according to your expectations. Remember, there might be occasions where your model's performance might start deteriorating. Do not hesitate in pausing your trading until you have modified the strategy to perform as per your expectations.

Great! we have finally implemented a machine learning model from the start to end.

So far, you have studied the classification based machine learning model. This is a type of supervised learning algorithm. This brings us to the end of the second part of the book. In the next part, you will see other types of machine learning algorithms.

Additional Reading

1. A Practical Guide to Feature Engineering in Python -
https://heartbeat.fritz.ai/a-practical-guide-to-feature-engineering-in-

python8326e40747c8

2. Data & Feature Engineering for Trading [Course] - https://quantra.quantinsti.com/course/data-and-feature-engineering-fortrading

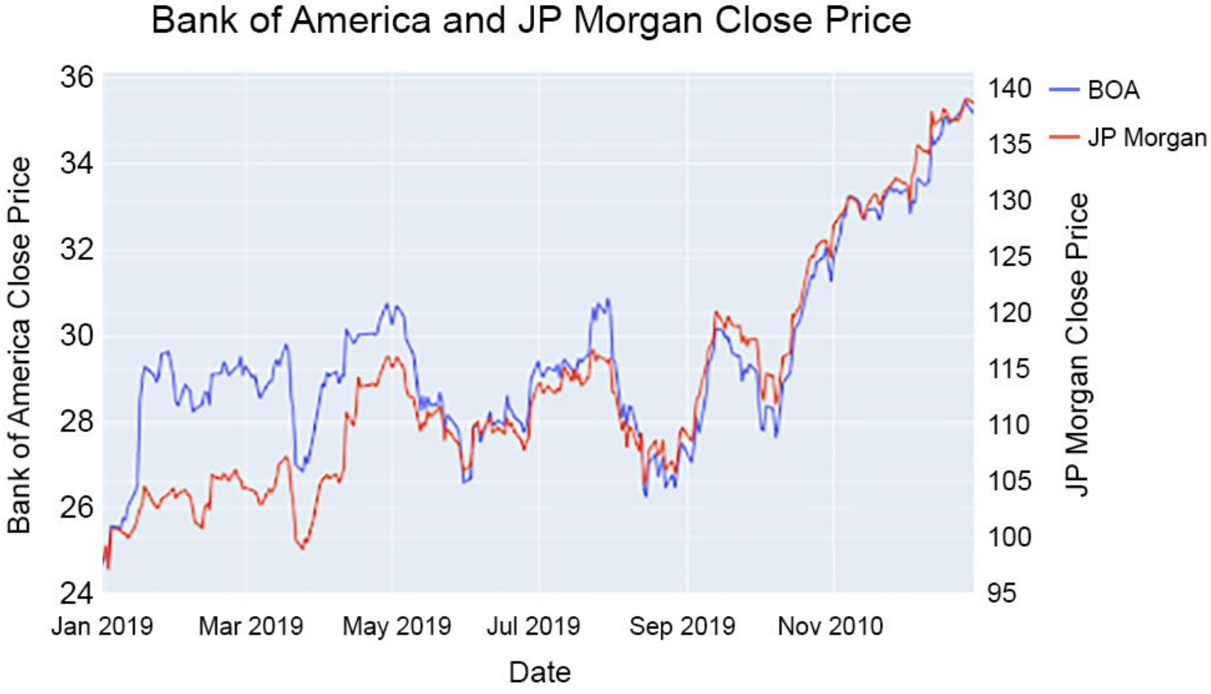
3. Best Input for Financial Models - https://davidzhao12.medium.com/advancesin-financial-machine-learning-for-dummies-part-1-7913aa7226f5
4. Top 9 Feature Engineering Techniques with Python - https://rubikscode.net/2021/06/29/top-9-feature-engineering-techniques/
5. Data Labelling: The Triple-barrier Method - https://towardsdatascience.com/the-triple-barrier-method-251268419dcd
6. How to Use StandardScaler and MinMaxScaler Transforms in Python - https://machinelearningmastery.com/standardscaler-and-minmaxscalertransforms-in-python/
7. What is the ideal ratio of in-sample length to out-of-sample length? - https://quant.stackexchange.com/questions/1480/what-is-the-ideal-ratio-ofin-sample-length-to-out-of-sample-length
8. Data normalization before or after train-test split? - https://datascience.stackexchange.com/questions/54908/data-normalizationbefore-or-after-train-test-split
9. Cross Validation In Machine Learning Trading Models - https://blog.quantinsti.com/cross-validation-machine-learning-trading models/
10. Cross Validation in Finance: Purging, Embargoing, Combination - https://blog.quantinsti.com/cross-validation-embargo-purgingcombinatorial/
11. How to Choose Right Metric for Evaluating ML Model - https://www.kaggle.com/vipulgandhi/how-to-choose-right-metric-for evaluating-ml-model
12. Choosing the Right Metric for Evaluating Machine Learning Models—Part 2 - https://www.kdnuggets.com/2018/06/right-metric-evaluating-machinelearning-models-2.html
13. How do I evaluate models that predict stock performance? - https://quant.stackexchange.com/questions/33074/strategy-for-backtesting
14. What is an acceptable Sharpe Ratio for a prop desk? - https://quant.stackexchange.com/questions/21120/what-is-an-acceptablesharpe-ratio-for-a-prop-desk/21123#21123

15. Doing opposite of what the model says - https://quant.stackexchange.com/questions/35905/doingopposite-of-what-the-model-says/35906#35906

16. Should a model be re-trained if new observations are available? - https://datascience.stackexchange.com/questions/12761/should-a-modelbe-re-trained-if-new-observations-are-available

Part III
Supervised Learning: Regression and Classification 9 The Linear Regression Model

If two stocks move in a set pattern, can you take advantage of it?
For example, the stock prices of J. P. Morgan and Bank of America move in tandem. How will you find the relationship between J. P. Morgan and Bank of America? Linear regression to the rescue!



This knowledge of the relationship will help you to predict J. P. Morgan’s stock price using Bank of America’s stock price. And you can generate trading signals when there is extreme deviation from the predicted price and the actual price.

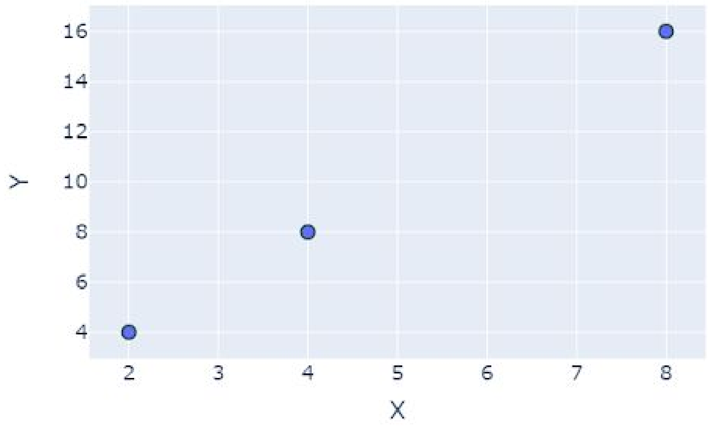
How does linear regression work or how to find the relationship? We will work with a simplified example to learn the workings of linear regression.

There are two time series x and y.
Date x y

124
248
3 8 16
43?

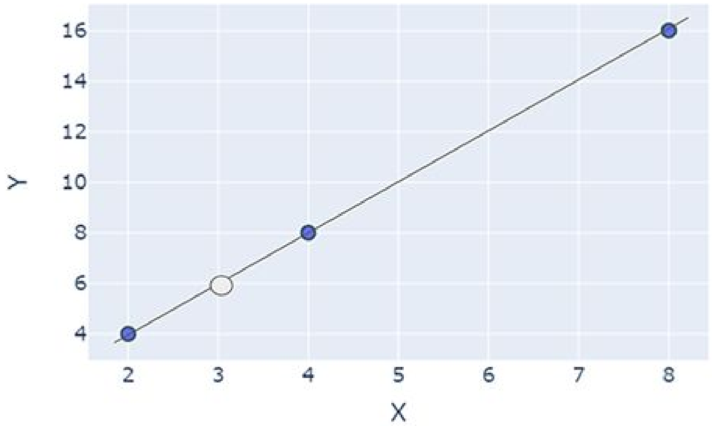
On day 1, the price of X is 2 and Y is 4 and so on.
What should the price of Y on day 4 given that the price of X is $3?

You can easily see here that y is double of the x. So you can say that the price of Y will be $6. In other words, the mathematical representation of the relationship is y = 2x. You can also plot the first three points on a graph.



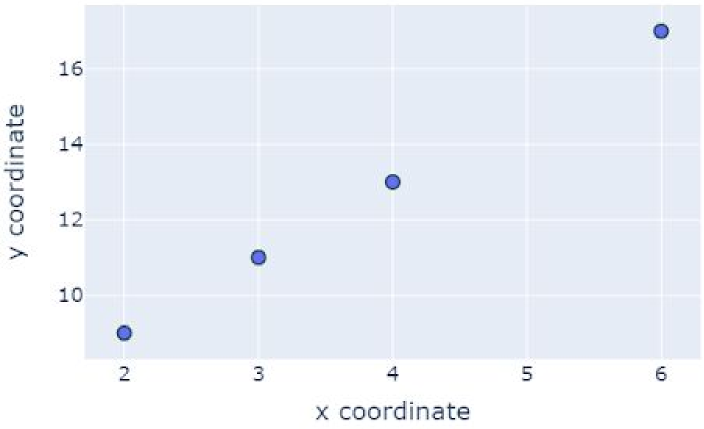
And draw a line passing through these points.

29
3 11



To predict the price of Y on day 4, given that the price of X is $3. You can draw the line as shown to get the price of $6 for Y.

This graph is also called scatter plot.



Let’s take another example. From the table or the scatter plot shown above, what will be the price of y on day 4 given that price of X is $6?

It is easy to find from the scatter plot that the price would be $17.
xy

29
3 11

4	13
6	17

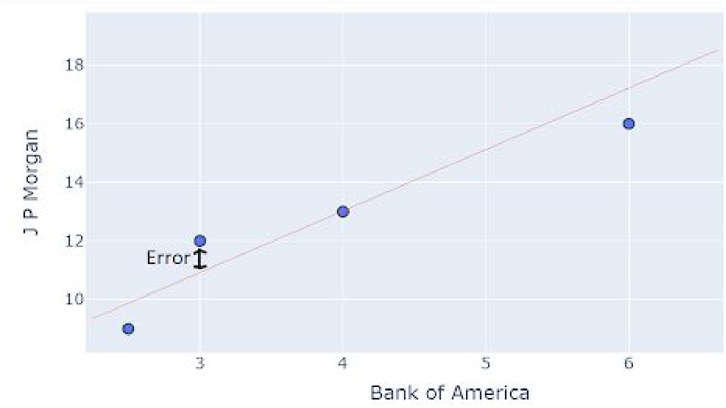
From the table, the price of y is double of x plus 5 dollars. (y = 2x + 5) That is double of 6 plus 5. The price of Y is 17.

This equation can also be used to describe the line on the scatter plot. Here, 2 is the slope of the line and 5 is the intercept or the value of y when x is 0.

This is the basic principle behind linear regression, where you fit a line to connect the data points. Further, you can use this line to predict the value of Y if you know the value of X.

In the above two examples, all the points were on the line.

But this is far from true for real-world data. If we change the x and y to J. P. Morgan and Bank of America, then the points would fall either above or below the line. The distance between the point and the line is called the forecasting error.

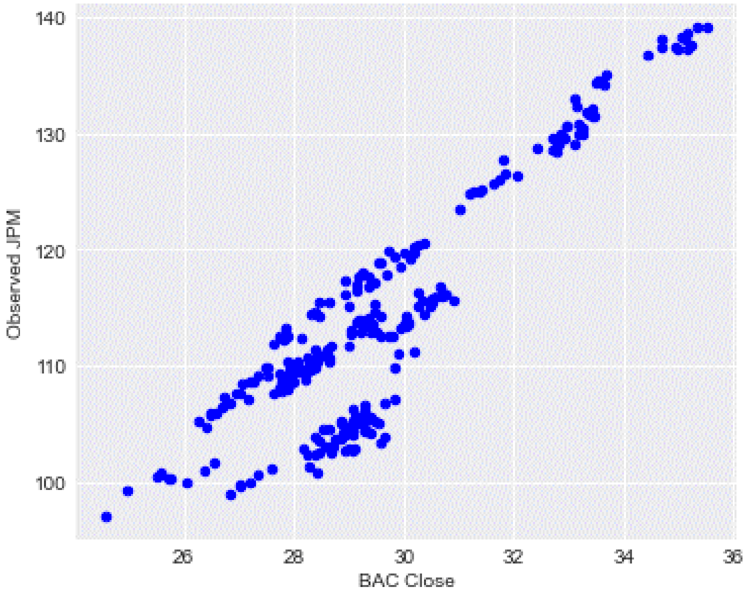


There are various regression models which work to minimise this error. Let us see how linear regression can be used on real world data and how to judge its performance.

9.1 Linear Regression in Trading

Let us now see how linear regression works on trading data. You will build a linear regression model using Bank of America and J.P. Morgan’s stock prices. Since you are using Bank of America’s stock price to predict J.P. Morgan’s price, Bank of America’s price will be your independent variable and x coordinate. J.P. Morgan will be your dependent variable and y coordinate. To read a CSV file, you can use the read_csv method of pandas.

```
[]: #Import thelibraries
import pandas as pd
import matplotlib.pyplot as plt %matplotlib inline
plt.style.use('seaborn-darkgrid')

#The dataisstored inthedirectory 'data_modules'
path = '../data_modules/'
#Read thecsvfile using theread_csvmethod of pandas
data = pd.read_csv(path + 'jpm_and_bac_price_2019.csv',

index_col=0)
data.plot.scatter('BAC Close','Observed JPM',color='blue', figsize=(6,5));
```



You can use the fit() method of statsmodels.api.OLS to compute the variables needed to develop a linear regression model. The syntax is shown below:

Syntax:

```
import statsmodels.api as sm
```

```
model = sm.OLS(y, X).fit()
```

1. **y**: y coordinate
2. **X**: x coordinate

The following methods/properties are used.
1. add_constant(): Compute the coefficient, as default is considered 0. 2. summary(): View the details of the regression model.

```
[]: import warnings
warnings.simplefilter('ignore')
import statsmodels.api as sm
Y = data['Observed JPM']
X = sm.add_constant(data['BAC Close'])
model = sm.OLS(Y,X).fit()
model.summary()
```

```
[]: <class 'statsmodels.iolib.summary.Summary'> """

OLS Regression Results
=====================================================
========================= Dep. Variable:

,! 0.816
Model:
,! 0.815
Method:
,! 1101.
Date:
,!17e-93
Time:
,!-728.85
No. Observations: ,! 1462.
Df Residuals:
,! 1469.
Df Model:
Covariance Type:
=====================================================
======================== coef std err t P>|t| [0.025 ,!0.975]
------------------------------------------------------------------------------const
-10.8576 3.764 -2.885 0.004 -18.270 ,!-3.445
BAC Close 4.2167 0.127 33.185 0.000 3.966 ,! 4.467
======================================================
========================

Observed JPM R-squared:
OLS Adj. R-squared:
Least Squares F-statistic:
Wed, 01 Sep 2021 Prob (F-statistic): 2.
21:11:28 Log-Likelihood:
251 AIC:
249 BIC:
1 nonrobust

Omnibus:
,! 0.045
Prob(Omnibus): ,!23.956
Skew:
,!28e-06
Kurtosis:
,! 399.
======================================================
======================== 37.420 Durbin-Watson:

0.000 Jarque-Bera (JB):
-0.626 Prob(JB): 6.
2.150 Cond. No.
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is

,!correctly
specified.
"""
```

```
[]: model.params
[]: const -10.857609
BAC Close 4.216694
dtype: float64
```

The linear regression equation can be computed as follows:

```
[]: print(f"y= {round(model.params[0],2)} + \
{round(model.params[1],2)} *x")
print(f"JPM = {round(model.params[0],2)} + \
{round(model.params[1],2)} *BAC")
y= -10.86+4.22 *x
JPM = -10.86 + 4.22 * BAC
```

But how well is Bank of America's price data able to predict the price of J. P. Morgan. For this, we will use the goodness of fit metric or R2, as it is commonly known. Let us also understand the math behind the metric and learn how to calculate it using the sklearn library in Python.

9.2 R-squared

R-squared is also known as the **Coefficient of Determination** and is denoted by R^2.

You have already seen that the R-squared explains the percentage of variation in the dependent variable, y, that is described by the independent variable, X. It is equal to the square of correlation. Mathematically,

R
2 = Variance Explained by the Model Total Variance
=

1
Unexplained Variance Total Variance

=
1
Sum Squared Regression Error Sum Squared Total Error
=
1

Â^n $(y_i\ y_i)^2{}_{i=1}$

Â

n

(

$y_i\ y)^2{}_{i=1}$

Where, y_i is the observed value for the ith row. - ˆ$_i$ is the predicted value for the ith row. - ¯ is the average of observed y’s.

Now, since you understand the math behind R^2, what is the range of R^2? The value of R^2 always lies between 0 and 1.

Refer back to the formula above, and you will be able to see that R^2 will be 1 if and only if all the y_i’s would be exactly equal to the respective ˆ$_i$’s. That means R^2 will be 1 when the model will be able to predict all the values precisely. And R^2 will be 0 when the model will not be able to capture any relationship between the X’s and the y’s.

Calculate R-squared Let us calculate the R-squared for the above data using the formula.

```
[]: r_sq = 1 ((data['Observed JPM']-data['Predicted JPM_BAC']) \ **
2).sum()/((data['Observed JPM']\
- data['Observed JPM'].mean()) \ ** 2).sum()
print('The R-squared is %.2f' % r_sq)
```

```
The R-squared is 0.82
```

The above value of R-squared means that 82% of the variance in the stock price of J.P. Morgan is explained by the variance in the stock price of Bank of America. You can also calculate the same using the sklearn library. The sklearn library has a r2_score function which can be imported as below:

from sklearn.metrics import r2_score

Using the r2_score function, the R-squared can be calculated as shown below: r2_score(y_true,y_predicted)

```
[]: from sklearn.metrics import r2_score
r_sq = r2_score(data['Observed JPM'], data['Predicted JPM_BAC'])
print('The R-squared is %.2f' % r_sq)
```

```
The R-squared is 0.82
```

Let us read another CSV file stored in the same directory and check the value of R^2. This file has observed and predicted stock price of J.P. Morgan. This time, the stock price of J.P. Morgan has been predicted using the stock price of Nestle.

```
[]: data_nestle = pd.read_csv(path \
+ 'predicted_jpm_and_nestle_price_2019.csv', index_col=0)

data_nestle.tail()
```

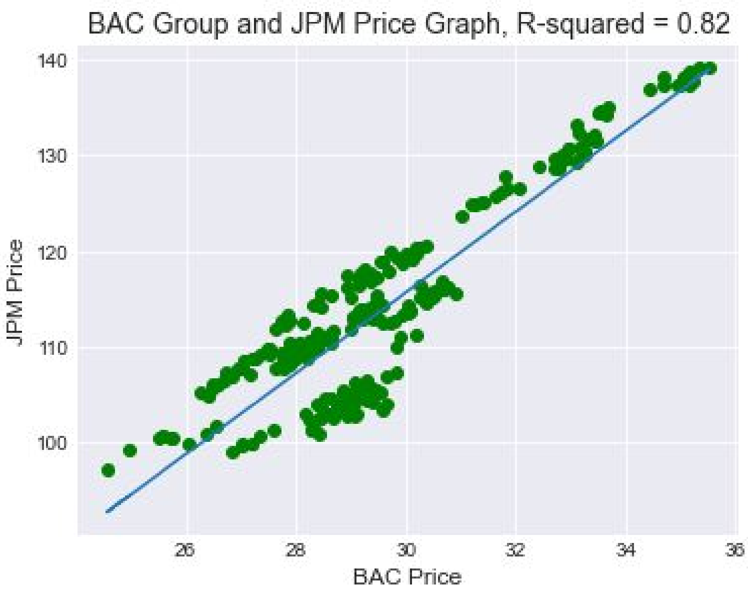
```
[]: Observed JPM Nestle Close Predicted JPM_Nestle Date
23-12-2019 137.199997 108.800003 120.207267 24-12-2019 137.580002
108.589996 120.048491 26-12-2019 139.039993 108.709999 120.139220
27-12-2019 139.139999 108.980003 120.343358 30-12-2019 138.630005
107.849998 119.489012
```

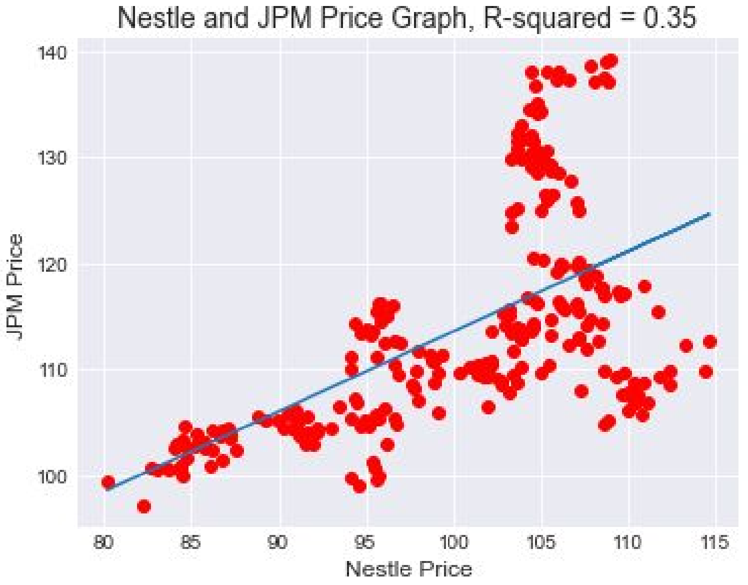
```
[]: r_sq = r2_score(data_nestle['Observed JPM'], data_nestle['Predicted
JPM_Nestle']) print('The R-squared is %.2f' % r_sq)
The R-squared is 0.35
```

Can you interpret the value of R^2 above? The above value of R-squared, 0.35, means that only 35% of the variance in the stock price of J.P. Morgan is explained by the variance in the stock price of Nestle.

Let us have a look at the plot of the JPM Price vs BAC Price and JPM Price vs Nestle Price.

```
[]: fig, (ax1, ax2) = plt.subplots(2, 1,figsize=(6, 5*2))
#Plot ofJPMPrice vsBACPrice
ax1.scatter(data['BAC Close'], data['Observed JPM'], color="green")

ax1 .plot(data['BAC Close'], data['Predicted JPM_BAC']) #Set thetitleand
labels fortheplot
ax1.set_title('BAC Group and JPM Price Graph, R-squared = %.2f' \

% r2_score(data['Observed JPM'],
data['Predicted JPM_BAC']), fontsize=14) ax1.set_xlabel('BAC
Price',fontsize=12)
ax1.set_ylabel('JPM Price',fontsize=12)

#Plot ofJPMPrice vsNestlePrice
ax2.scatter(data_nestle['Nestle Close'],
data_nestle['Observed JPM'], color="red")

ax2 .plot(data_nestle['Nestle Close'],
data_nestle['Predicted JPM_Nestle'])
#Set thetitleand labels fortheplot

ax2.set_title('Nestle and JPM Price Graph, R-squared = %.2f' \

% r2_score(data_nestle['Observed JPM'], data_nestle['Predicted
JPM_Nestle']),
fontsize=14)
ax2.set_xlabel('Nestle Price',fontsize=12)
ax2.set_ylabel('JPM Price',fontsize=12)

plt .show()
```





Of the two values of the R^2 that we have seen, which one do you think is better?

Yes, the first value (82%) is better as it explains more variance. In the first graph, you can see that the points are closer to the line of best fit and scattered in the second graph. This explains the difference between the two values of the R^2.

Intuitively, the change in the stock price of J.P. Morgan would be highly correlated to the change in the stock price of Bank of America, and it would not be correlated to the change in the stock price of Nestle. This is because both J.P. Morgan and Bank of America belong to the same sector of Banking, and are very sensitive to factors like interest rates. Nestle belongs to a different sector, Fast Moving Consumer Goods. The dynamics of this sector are different and are affected by different factors.

9.2.1 Limitations of R-squared

R-squared does not allow us to see if the predictions are biased. This can be done by the analysis of the residuals. Any pattern in the residual plot will help us identify the bias in our model if any. Hence, a high R^2 alone will always not be a good statistic.

Conclusion R-squared is one of the most popular metrics to measure the goodness of fit of a linear regression model.

Using R-squared, you concluded how the stock price of Bank of America was able to explain better the variance in the stock price of J.P. Morgan. Can you think of any other stock price that can also help you explain this variance? The stock price of any other bank or investment institute might do the job for you.

Now it’s your turn. You can download any such data from finance.yahoo.com and apply what you learned.

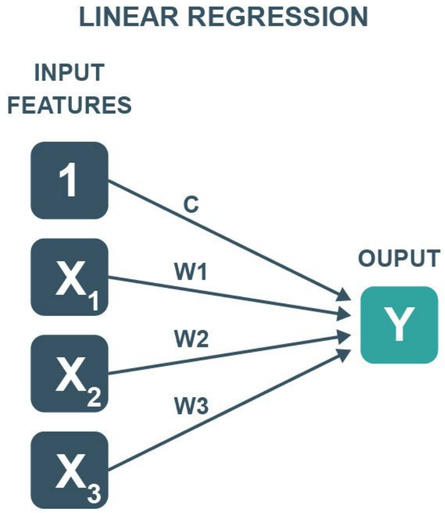
The linear regression is a simple way to model a relation between the independent and dependent variable. Another similar sounding name is logistic regression, which we will see in the next chapter.

10 Logistic Regression

Logistic regression falls under the category of supervised learning. It measures the relationship between the categorical dependent variable, and one or more independent variables by estimating probabilities using a logistic/sigmoid function.

In spite of the name ‘logistic regression’, this is not used for machine learning regression problem where the task is to predict the real-valued output. It is a classification problem which is used to predict a binary outcome (1/0, -1/1, True/False) given a set of independent variables.

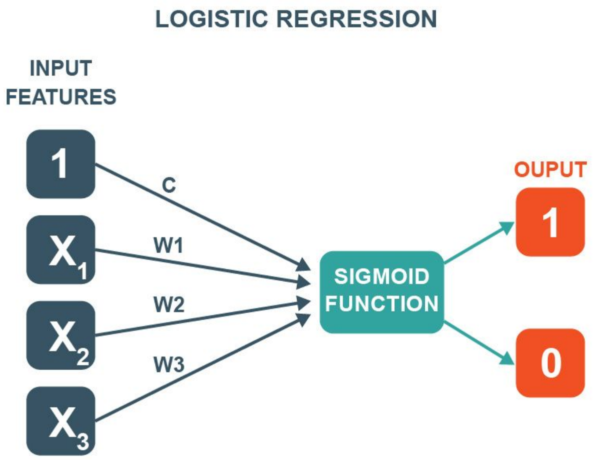
Logistic regression is a bit similar to the linear regression, or we can say it as a generalised linear model. In linear regression, we predict a real-valued output ‘y’ based on a weighted sum of input variables.



$y = c + x_{1\barleftarrow}\ w_1 x_{2\barleftarrow}\ w_2 + ...x_{n\barleftarrow}\ w_n$

The aim of linear regression is to estimate values for the model coefficients $c, w_1, w_2, w_3 .... w_n$ and fit the training data with minimal squared error and predict the output y.

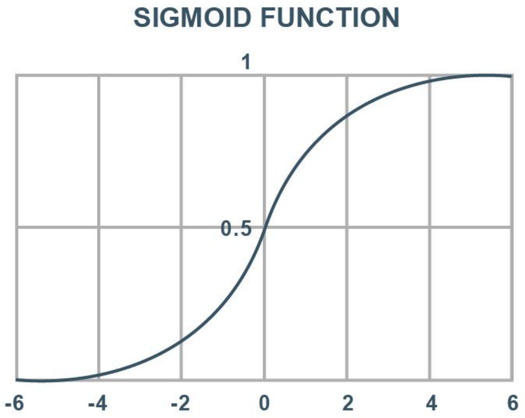
Logistic regression does the same thing, but with one addition. The logistic regression model computes a weighted sum of the input variables similar to the linear regression, but it runs the result through a special non-linear function, the logistic function or sigmoid function to produce the output y. Here, the output is binary or in the form of 0/1 or -1/1.



$y = \text{logistic}(c + x_{1⇤}\ w_1 x_{2⇤}\ w_2 + ...x_{n⇤}\ w_n)$

$y = 1/(1 + e[\ (c + x_{1⇤}\ w_1 x_{2⇤}\ w_2 + ...x_{n⇤}\ w_n)]$
The sigmoid/logistic function is given by the following equation:
$y = 1/1 + e^x$

As you can see in the graph, it is an S-shaped curve that gets closer to 1 as the value of input variable increases above 0, and gets closer to 0 as the input variable decreases below 0. The output of the sigmoid function is 0.5 when the input variable is 0.



Thus, if the output is more than 0.5, we can classify the outcome as 1 (or positive) and if it is less than 0.5, we can classify it as 0 (or negative).

Now, we have a basic intuition behind the logistic regression and the sigmoid function. We will learn how to implement logistic regression in Python and predict the stock price movement using the above condition.

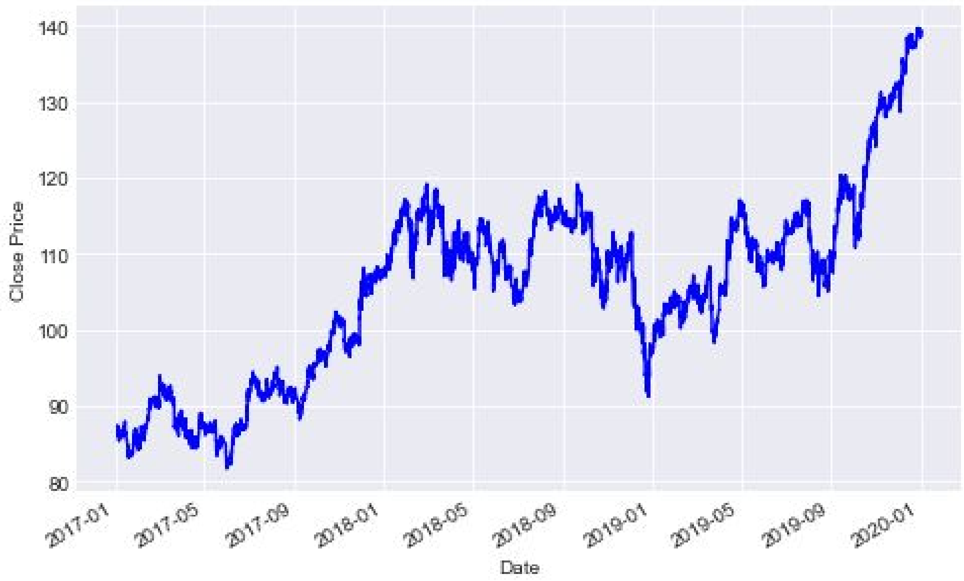
10.1 Logistic Regression in Python

We will start by importing the necessary libraries.

```
[]: #Data manipulation
import numpy as np
import pandas as pd

#Technicalindicators
import talib as ta

#Plotting the graphs
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-darkgrid')

#Machine learning
from sklearn.linear_model import LogisticRegression from sklearn import
metrics
```

Import Dataset We will use the same data which we used in the second part of the book.

```
[]: #The dataisstored inthedirectory 'data_modules' path = "../data_modules/"

#Read thedata
data = pd.read_csv(path + 'JPM_2017_2019.csv',index_col=0) data.index =
pd.to_datetime(data.index)

data .close.plot(figsize=(8,5), color='b') plt.ylabel('Close Price')
plt.xlabel('Date')
plt.show()
```



Define Target, Features and Split the Data We have created functions for the standard tasks such as creating the target and features. This makes it easy for you to import the module and carry out the tasks without typing the code everytime. You will find the utility module as well as its related information on the github page.

Note that this is not available as a standard ‘pip install’.

```
[]: import sys
sys.path.append("..")
from data_modules.utility import get_target_features
y, X = get_target_features(data)
split = int(0.8*len(X))
X_train, X_test, y_train, y_test = X[:split], X[split:],
y[:split], y[split:]
```

Feature Scaling Another important step in data preprocessing is to standardise the dataset. This process makes the mean of all the input features equal to zero and also converts their variance to 1. This ensures that there is no bias while training the model due to the different scales of all input features. If this is not done the neural network might get confused, and give a higher weight to those features which have a higher average value than others.

We implement this step by importing the StandardScaler method from the sklearn.preprocessing library. We instantiate the variable sc with the StandardScaler() function. After which we use the fittransform function for implementing these changes on the Xtrain and Xtest datasets. The ytrain and y_test sets need not be standardised.

```
[]: from sklearn.preprocessing import StandardScaler sc = StandardScaler()
X_test_original = X_test.copy()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Instantiate the Logistic Regression We will instantiate the logistic regression in Python using 'LogisticRegression' function and fit the model on the training dataset using 'fit' function.

```
[]: model = LogisticRegression() model = model.fit(X_train,y_train)
```

Examine the Coefficients

```
[]: pd.DataFrame(zip(X.columns, np.transpose(model.coef_)))
```

```
[]: 01
0pct_change[0.01538636081786934]
1pct_change2 [-0.08026259453839449]
2pct_change5 [0.024335207683013584]
3rsi[0.0052866122780786074]
4adx[0.013464920611383466]
5corr[-0.001639525259172929]
6volatility[0.016321785785599726]
```

Calculate Class Probabilities We will calculate the probabilities of the class for the test dataset using 'predict_proba' function.

```
[]: probability = model.predict_proba(X_test) print(probability)
```

```
[[0.50948029 0.49051971] [0.51218059 0.48781941] [0.51603258
0.48396742] ...
[0.51860144 0.48139856] [0.52032489 0.47967511] [0.52745957
0.47254043]]
```

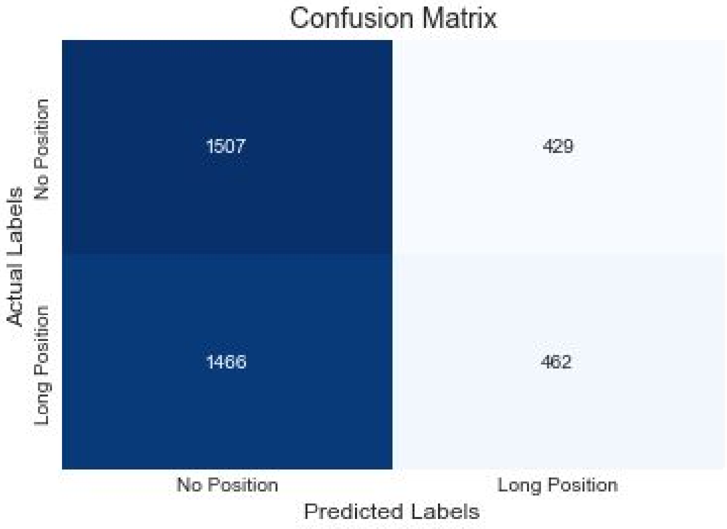
Predict Class Labels Next, we will predict the class labels using predict function for the test dataset.

If you print ‘predicted’ variable, you will observe that the classifier is predicting 1, when the probability in the second column of variable ‘probability’ is greater than 0.5. When the probability in the second column is less than 0.5, then the classifier is predicting -1.

```
[]: predicted = model.predict(X_test)
```

Evaluate the Model We will by printing the confusion matrix, as well as the classification report and the score.

```
[]: from data_modules.utility import get_metrics
get_metrics(y_test, predicted)
```



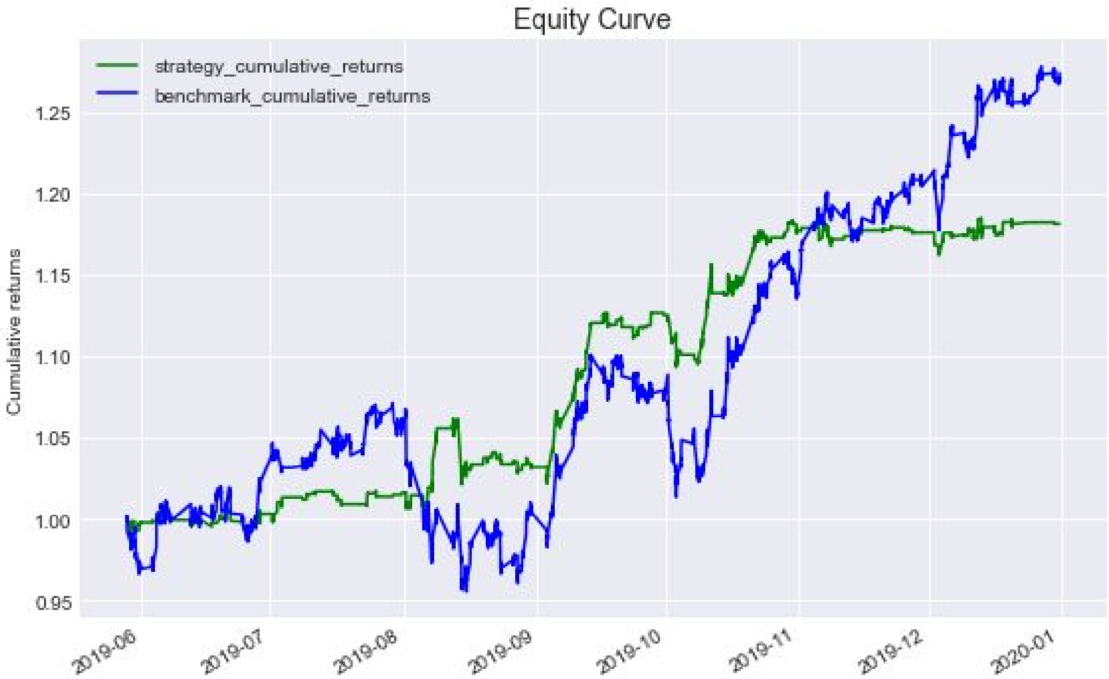
```
precision recall f1-score
support
00.51 0.78 0.61 1936
10.52 0.24 0.33 1928

accuracy 0.51 3864
macro avg 0.51 0.51 0.47 3864
weighted avg 0.51 0.51 0.47 3864
```

Create Trading Strategy Using the Model We will predict the signal on the test dataset. We will calculate the cumulative strategy return based on the signal predicted by the model in the test dataset. We will also plot the cumulative returns.

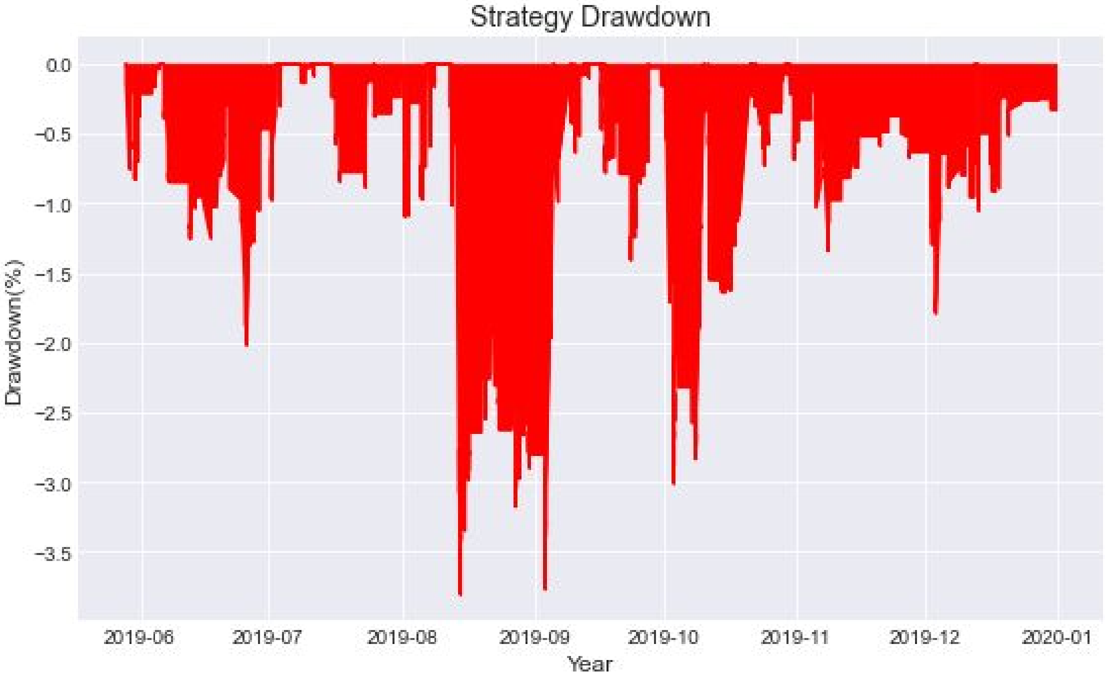
```
[]: #Calculatethe percentage change
strategy_data = X_test_original[['pct_change']].copy()
#Predict the signals
strategy_data['predicted_signal'] = model.predict(X_test)

#Calculatethe strategyreturns
strategy_data['strategy_returns'] = \
strategy_data['predicted_signal'].shift(1) * \
strategy_data['pct_change']

#Drop themissingvalues
strategy_data.dropna(inplace=True)
strategy_data.head()
```

```
[]: pct_change predicted_signal ,!strategy_returns

2019-05-28 12:15:00+00:00 0.000732 0 0. ,!0
2019-05-28 12:30:00+00:00 -0.000366 0 -0. ,!0
2019-05-28 12:45:00+00:00 0.000366 0 0. ,!0
2019-05-28 13:00:00+00:00 0.000091 0 0. ,!0
2019-05-28 13:15:00+00:00 -0.000091 0 -0. ,!0
```

```
[]: from data_modules.utility import get_performance
get_performance(strategy_data)
```



The maximum drawdown is -3.80%.



The Sharpe ratio is 2.75.

It can be observed that the Logistic Regression model in Python generates decent returns. Now it’s your turn to play with the code by changing parameters and create a trading strategy based on it.
Probability plays a key role in creating a trading strategy. In the next chapter,

we will look at Naive Bayes theorem and how it can be used in machine learning.

11 Naive Bayes Model

Have you heard about the Occam’s razor?
To put it simply, William of Ockham stated that, “The simplest solutions are almost always the best solutions.”

But in a post about Naive Bayes, why are we talking about razors? Actually, Naive Bayes implicitly incorporates this belief, because it really is a simple model. Let’s see how a simple model like the Naive Bayes model can be used in trading.

What is Naive Bayes? Let’s take a short detour and understand what the “Bayes” in Naive Bayes means. There are basically two schools of thought when it comes to probabilities. One school suggests that the probability of an event can be deduced by calculating the probabilities of all the probable events, and then calculating the probability of the event you are interested in.

For example, in the case of coin toss experiment, you know that probability of heads is ½ because there are only two possibilities here, heads or tails.

The other school of thought suggests that probability is more dependent on prior information as well as other factors too. For example, if the probability of a person saying red is their favourite colour is 30%, but 50% if they are in love marriage, then your result will be different based on their marital status.

This is known as Bayesian inference, where you try to calculate the probability depending on a certain condition.
And how do you calculate this conditional probability? Let’s see in the next section.
Equation for Bayes Theorem
P
(
A
|
B

)=
P(B|A)⇤ P(A) P(B)
Let’s say A is the event when a person says red is their favourite colour.
Now, let B be the event when the person is married.
Thus, P(A | B) is the likelihood of A saying red is their favourite colour when the person is married.
This is the conditional probability we have to find.
In a similar sense P(B | A) is the likelihood of a person being married when the person says their favourite colour is red.

P(A) and P(B) is the respective probability.
How does this help us in trading?
Let’s assume we know the RSI value of a stock.

Now, what if you want to find the probability that the price rises the next day after the RSI goes below 40. Think about it. If the RSI goes below 40 on Tuesday, you will want to buy on Wednesday hoping that price will rise.

This can found by using the Bayes theorem.
Let P(A) be the probability the the price will increase and P(B) be the probability that the RSI was below 40 the previous day.

Now, we will find the probability that the price rises the next day if the RSI is below 40 by the same formula.
Here, B is similar to the feature we define in machine learning. It can also be called as the evidence.
But hold on! What if we want to check when the RSI is below 40, as well as the “slow k” of stochastic oscillator is more than its “slow d”?

Technically, we can have multiple conditions in Bayes theorem to improve our probability model. If P(C) is the probability that “slow k” passes “slow d”, then the Bayes theorem would be written as:

P
(
A
|
B

,
C
)=
$P(B|A)P(C|A)P(A)\ P(B)P(C)$

While this looks simple enough to compute, if you add more features to the model, the complexity increases. Here is where the Naive part of the Naive Bayes model comes into the picture.

Assumption of Naive Bayes Model The Naive Bayes model assumes that both B and C are independent events. Further the denominator is also dropped. This simplifies the model to a great extent and we can simply write the equation as:
$P(A|B,C)= P(B|A)_{⇤}\ P(C|A)_{⇤}\ P(A)$

You must remember that this assumption can be incorrect in real life. Logically speaking, both the RSI and stochastic indicators are calculated using the same variable, i.e. price data. Thus, they are not exactly independent.

However, the beauty of Naive Bayes model is that even though this assumption is not true, the model still performs well in various scenarios. Wait, is there just one type of Naive Bayes model? Actually there are three. Let’s find out in the next section.
Types of Naive Bayes models Depending on the requirement, you can pick the model accordingly. These models are based on the input data you are working on:

Multinomial: This model is used when we have discrete data and working on its classification. A simple example is we can have the weather (cloudy, sunny, raining) as our input and we want to see in which weather is a tennis match played.

Gaussian: As the name suggests, in this model we work on continuous data which follows a Gaussian distribution. An example would be the temperature of the stadium where the match is played.
Binomial: What if we have the input data as simply yes or no, i.e. a Boolean value. In this instance, we will use the binomial model.

The great thing about Python is that the sklearn library incorporates all these models. Shall we try to use it for building our own Naive Bayes model? Why not try it out.

11.1 Naive Bayes Model in Python

We will start our strategy by first importing the libraries and the dataset.

```
[]: #Data manipulation
import numpy as np
import pandas as pd

#Technicalindicators
import talib as ta

#Plotting the graphs
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-darkgrid')

from sklearn.naive_bayes import BernoulliNB
[]: #The dataisstored inthedirectory 'data_modules' path = "../data_modules/"

#Read thedata
data = pd.read_csv(path + 'JPM_2017_2019.csv',index_col=0) data.index =
pd.to_datetime(data.index)

data .close.plot(figsize=(8,5), color='b')
plt.ylabel('Close Price')
plt.xlabel('Date')
plt.show()
```

We will calculate the indicators as well as their signal values.

```
[]: import sys
sys.path.append("..")
from data_modules.utility import get_target_features y, X =
get_target_features(data)
```

```
split = int(0.8*len(X))
X_train, X_test, y_train, y_test = X[:split], X[split:], \ y[:split], y[split:] And
now we use the Bernoulli Naive Bayes model for binomial analysis.
```

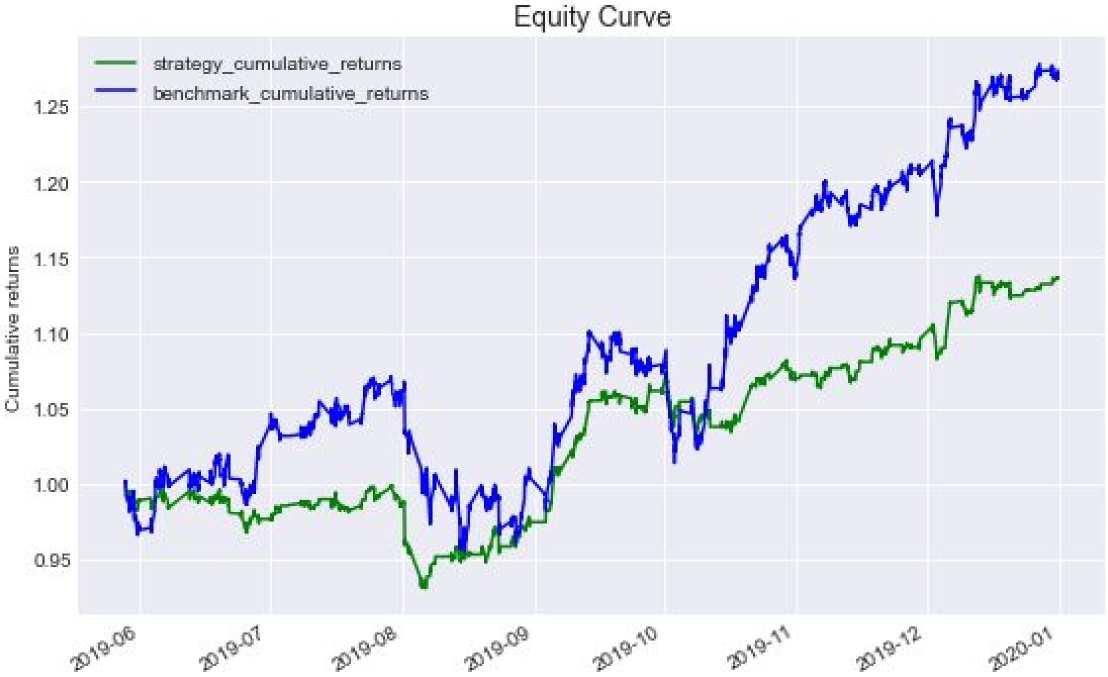
```
[]: model = BernoulliNB().fit(X_train, y_train) #Fit themodelon traindataset
model.fit(X_train, y_train)
predicted = model.predict(X_test)
```

```
How was the accuracy of our model? Let’s find out. []: from
data_modules.utility import get_metrics get_metrics(y_test, predicted)
precision recall f1-score support 00.51 0.64 0.56 1936 10.51 0.38 0.44 1928

accuracy 0.51 3864
macro avg 0.51 0.51 0.50 3864
weighted avg 0.51 0.51 0.50 3864
```

```
[]: #Calculatethe percentage change
strategy_data = X_test[['pct_change']].copy()
#Predict the signals
strategy_data['predicted_signal'] = model.predict(X_test)

#Calculatethe strategyreturns
strategy_data['strategy_returns'] = \
strategy_data['predicted_signal'].shift(1) * \
strategy_data['pct_change']

#Drop themissingvalues
strategy_data.dropna(inplace=True)
strategy_data.head() []: pct_change predicted_signal

‚!strategy_returns
2019-05-28 12:15:00+00:00 0.000732 0 0.
‚!000732
2019-05-28 12:30:00+00:00 -0.000366 0 -0.
‚!000000
2019-05-28 12:45:00+00:00 0.000366 0 0.
‚!000000
2019-05-28 13:00:00+00:00 0.000091 0 0.
```

```
↪000000
2019-05-28 13:15:00+00:00 -0.000091 1 -0.
↪000000
```

```
[]: from data_modules.utility import get_performance
get_performance(strategy_data)
```



The maximum drawdown is -7.04%.
The Sharpe ratio is 1.91.

There is obviously room for improvement here, but this was just a demonstration of how a Naive Bayes model works. But are there special occasions when the model should be used? Let's find out in the next section.

Advantages of the Naive Bayes Model

- The main advantage of the Naive Bayes model is its simplicity and fast computation time. This is mainly due to its strong assumption that all events are independent of each other.
- They can work on limited data as well.
- Their fast computation is leveraged in real time analysis when quick responses are required.

Although this speed comes at a price. Let’s find out how in the next section.
Disadvantages of Naive Bayes Model

- Since Naive Bayes assumes that all events are independent of each other, it cannot compute the relationship between the two events
- The Naive Bayes Model is fast but it comes at the cost of accuracy. Naive Bayes is sometimes called bad estimator.
- The equation for Naive Bayes shows that we are multiplying the various probabilities. Thus, if one feature returned 0 probability, it could turn the whole result as 0. There are, however, various methods to overcome this instance. One of the more famous ones is called Laplace correction. In this method, the features are combined or the probability is set to 1 which ensures that we do not get zero probability.

Conclusion The Naive Bayes model, despite the fact that it is naive, is pretty simple and effective in a large number of use cases in real life. While it is mostly used for text analysis, it has been used as a verification tool in the field of trading.

The Naive Bayes model can also be used as a stepping stone towards more precise and complex classification based machine learning models.

12 Decision Trees

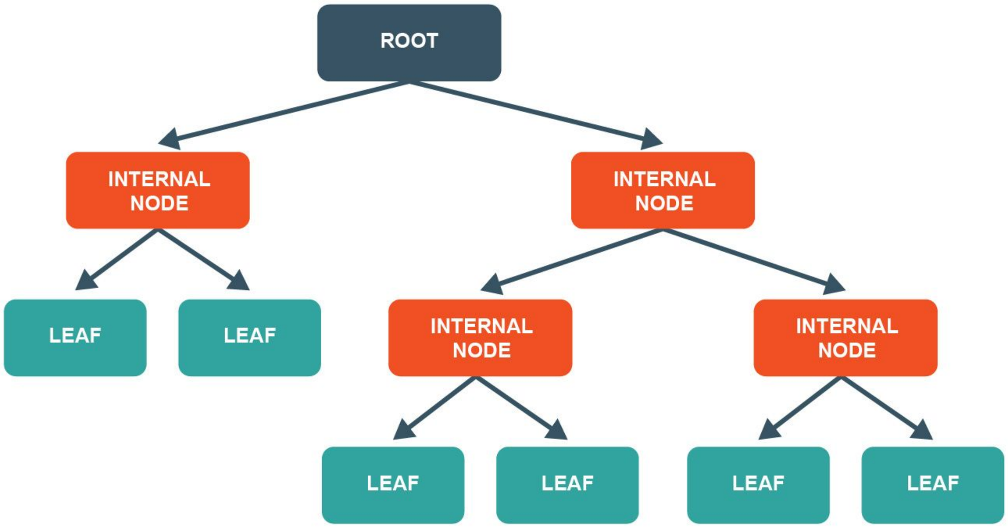
Decision Trees are a Supervised Machine Learning method used in Classification and Regression problems, also known as CART.

Remember that a classification problem tries to classify unknown elements into a class or category; the outputs are always categorical variables (i.e. yes/no, up/down, red/blue/yellow, etc.).

A regression problem tries to forecast a number such as the return for the next day. Although the classification and regression problems have different objectives, the trees have the same structure:

- The Root node is at the top and has no incoming pathways.
- Internal nodes or test nodes are at the middle and can be at different levels or sub-spaces, and have incoming and outgoing pathways.

- Leaf nodes or decision nodes are at the bottom, have incoming pathways but no outgoing pathways and here we can find the expected outputs.



Thanks to Python’s Sklearn library, the tree is automatically created for us. It takes a starting point as the predictor variables that we hypothetically think are responsible for the output we are looking for.

We will create a classification decision tree in Python to forecasts whether the financial instrument we are going to analyse will go up or down the next day.
We will also create a decision tree to forecast the concrete return of the asset the next day.

Building a Decision Tree Building a classification decision tree or a regression decision tree is very similar when it comes to the way we organise the input data and predictor variables. Then, by calling the corresponding functions, the classification decision tree or regression decision tree will be automatically created for us. Of course, we have to specify some criteria which are simple conditions.

The main steps to build a decision tree are:
1. Retrieve the market data for a financial instrument.
2. Introduce the predictor variables (i.e. Technical indicators, Sentiment

indicators, Breadth indicators, etc.).
3. Setup the target variable or the desired output.
4. Split the data between training and test data.
5. Generate the decision tree after training the model.
6. Test and analyse the model.

If we look at the first four steps, they are common operations for data processing.

Steps 5 and 6 are related to the ML algorithms for the decision trees specifically. As we will see, the implementation in Python will be quite simple. However, it is fundamental to understand well the parameterisation and the analysis of the results.

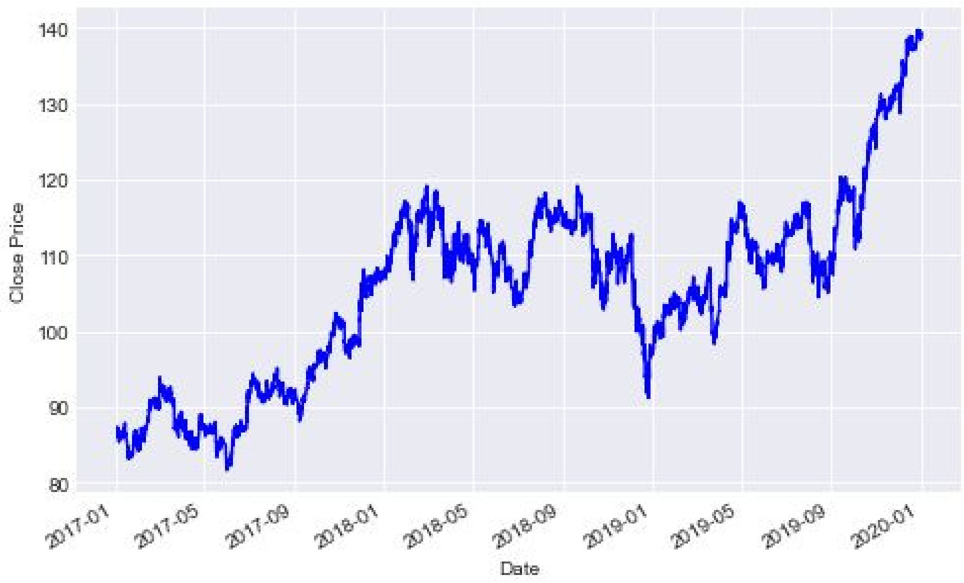
Getting the Data The raw material for any algorithm is data. In our case, it would be the time series of financial instruments, such as indices, stocks, etc. and it usually contains details like the opening price, maximum, minimum, closing price and volume.

There are multiple data sources to download the data, free and premium. The most common sources for free daily data are Quandl, Yahoo, Google, or any other data source we trust.

For the sake of uniformity, we are going to use the same dataset which we used in the previous part of the book. This helps us because we don't need to spend time on getting the data, but rather, we can focus on the machine learning algorithm and its working.

```
[]: #Data manipulation
import numpy as np
import pandas as pd

#Technicalindicators import talib as ta

#Plotting graphs
import matplotlib.pyplot as plt import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-darkgrid')
```

```
[]: #The dataisstored inthedirectory 'data_modules' path = "../data_modules/"

#Read thedata
data = pd.read_csv(path + 'JPM_2017_2019.csv',index_col=0) data.index =
pd.to_datetime(data.index)

data .close.plot(figsize=(8,5), color='b') plt.ylabel('Close Price')
plt.xlabel('Date')
plt.show()
```



```
[]: import sys
sys.path.append("..")
from data_modules.utility import get_target_features
y, X = get_target_features(data)

split = int(0.8*len(X))
X_train, X_test, y_train, y_test = X[:split], X[split:], \ y[:split], y[split:]
```

12.0.1 Decision Trees for Classification

Now let’s create the classification decision tree using the DecisionTreeClassifier function from the sklearn.tree library.

Although the DecisionTreeClassifier function has many parameters that we invite you to know and experiment with (help(DecisionTreeClassifier)), here we will see the basics to create the classification decision tree.

```
DecisionTreeClassifier(class_weight =None,criterion='gini',max_depth=3,
max_features=None,max_leaf_nodes=None,
min_samples_leaf=5,min_samples_split=2,
min_weight_fraction_leaf=0.0,presort=False,
random_state=None,splitter='best')
```

Basically, refer to the parameters with which the algorithm must build the tree because it follows a recursive approach to build the tree, so we must set some limits to create it.

criterion: For the classification decision trees, we can choose Gini or Entropy and Information Gain, these criteria refer to the loss function to evaluate the performance of a machine learning algorithm, and are the most used when it comes to the classification algorithms. Although it is beyond the scope of this chapter, it serves us to adjust the accuracy of the model, and the algorithm to build the tree. It also stops evaluating the branches in which no improvement is obtained according to the loss function.

max_depth: Maximum number of levels the tree will have.
min_samples_leaf: This parameter is optimisable and indicates the minimum number of samples that we want to have in the leaves.

```
[]: from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='gini',max_depth=3,
min_samples_leaf=5)
model = model.fit(X_train, y_train)
```

Now we need to make forecasts with the model on unknown data. For this, we will use 30% of the data that we had kept reserved for testing, and finally, evaluate the performance of the model. But first, let’s take a graphical look at the classification decision tree that the ML algorithm has automatically created for us.

Visualise Decision Trees for Classification We have at our disposal a very powerful tool that will help us to analyse graphically the tree that the ML algorithm has created automatically. The graph shows the most significant nodes that maximise the output and will help us determine, if applicable, some useful trading rules.

The graphviz library allows us to graph the tree that the DecisionTreeClassifier function has automatically created with the training data. You can use the conda forge command in the anaconda prompt to install it in your local system. Link: https://anaconda.org/conda-forge/python-graphviz

```
[]: from sklearn import tree
import graphviz
dot_data = tree.export_graphviz(model,

out_file =None,
filled=True,
feature_names=X_train.columns)

#To createthegraph, you canuncomment
#the belowlineof codeandrun it
#graphviz.Source(dot_data)
```

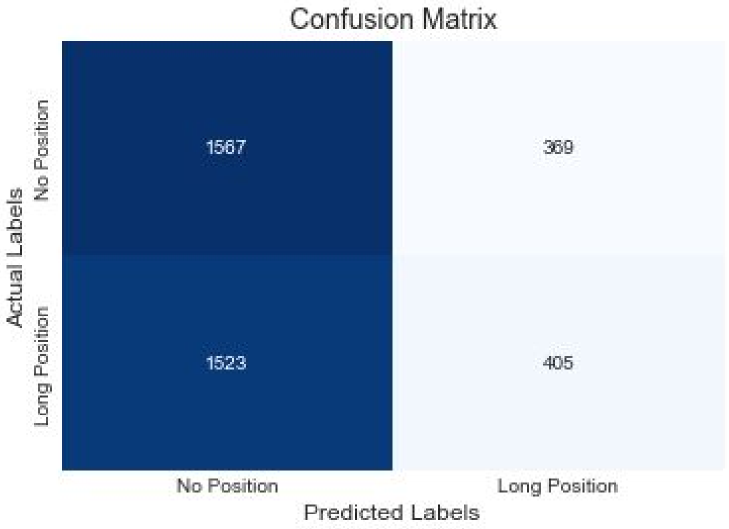
Note that the graph only shows the most significant nodes. In this graph, we can see all the relevant information in each node:

1. The predictor variable used to split the data set.
2. The value of Gini impurity.
3. The number of data points available at that node.
4. The number of target variable data points belonging to each class, 1 and 0.

We can observe a pair of pure nodes that allows us to deduce possible trading rules. For example, you can see how the nodes are split depending on the feature values which the algorithm has given more preference to.

Forecast Now let’s make predictions with data sets that were reserved for testing, this is the part that will let us know if the algorithm is reliable with unknown data in training.

```
[]: predicted = model.predict(X_test)
[]: from data_modules.utility import get_metrics
get_metrics(y_test, predicted)
```



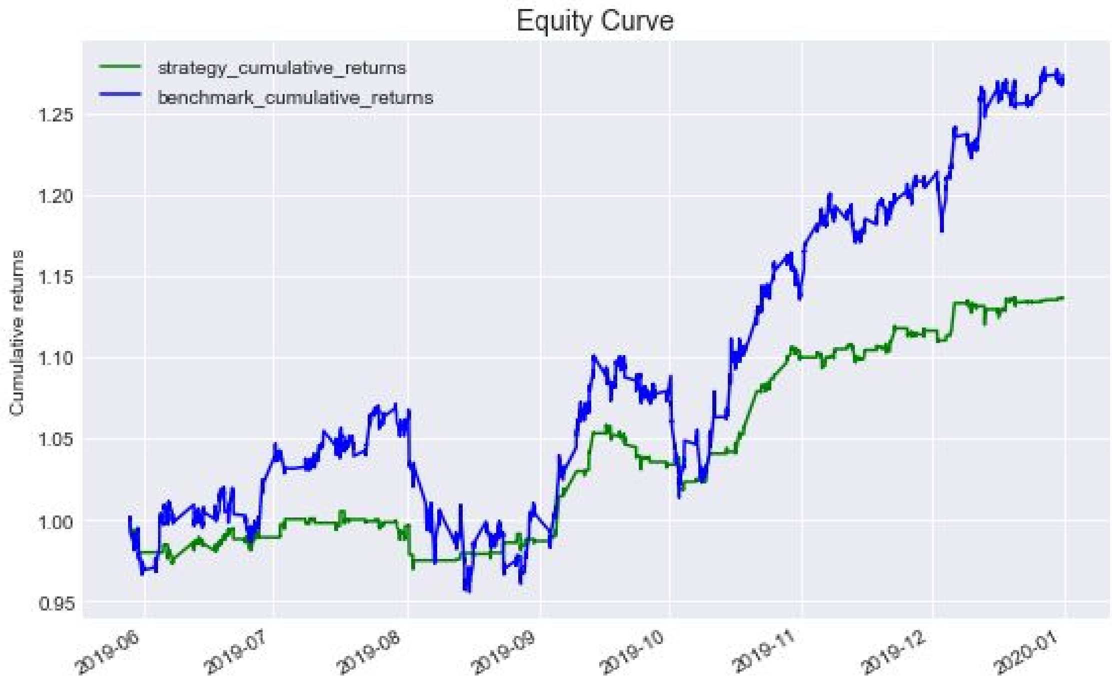
```
precision recall f1-score
support
00.51 0.81 0.62 1936 10.52 0.21 0.30 1928

accuracy 0.51 3864
macro avg 0.52 0.51 0.46 3864
weighted avg 0.52 0.51 0.46 3864
```

Performance Analysis Finally, we can only evaluate the performance of the algorithm on unknown data by comparing it with the result obtained in the training process.

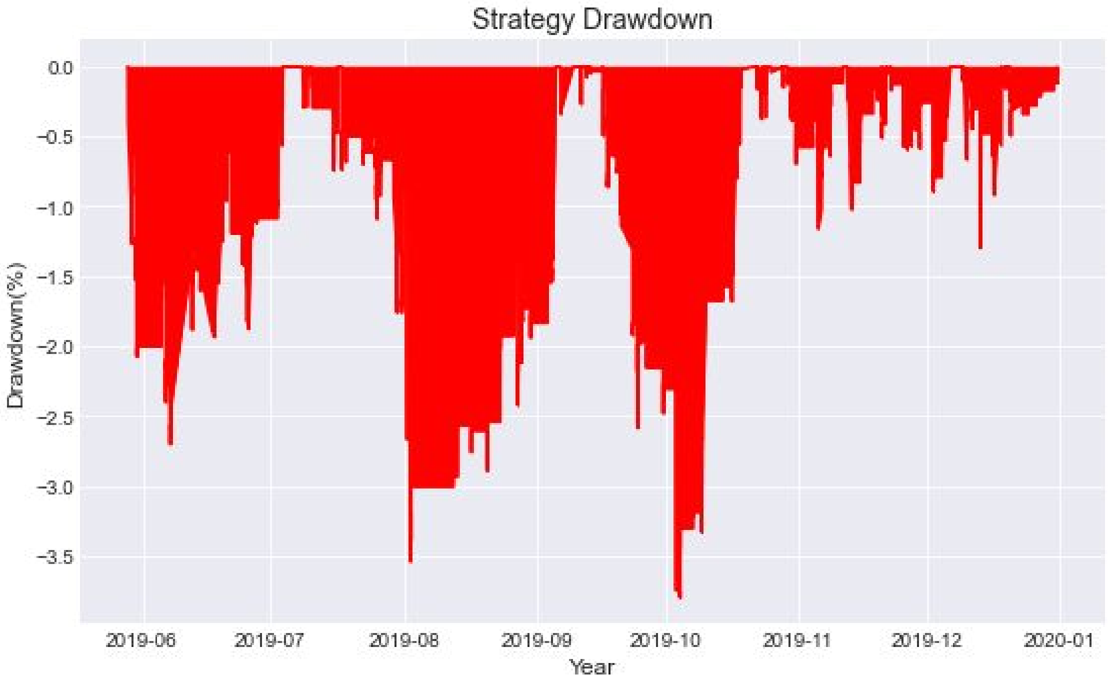
```
[]: #Calculatethe percentage change
strategy_data = X_test[['pct_change']].copy()
#Predict the signals
strategy_data['predicted_signal'] = model.predict(X_test)

#Calculatethe strategyreturns
strategy_data['strategy_returns'] = \
strategy_data['predicted_signal'].shift(1) * \
strategy_data['pct_change']

#Drop themissingvalues
strategy_data.dropna(inplace=True)
strategy_data.head()
[]: pct_change predicted_signal ,!strategy_returns
```

```
2019-05-28 12:15:00+00:00 0.000732 0 0. ,!0
2019-05-28 12:30:00+00:00 -0.000366 0 -0. ,!0
2019-05-28 12:45:00+00:00 0.000366 0 0. ,!0
2019-05-28 13:00:00+00:00 0.000091 0 0. ,!0
2019-05-28 13:15:00+00:00 -0.000091 0 -0. ,!0
```

```
[]: from data_modules.utility import get_performance
get_performance(strategy_data)
```



The maximum drawdown is -3.80%.



The Sharpe ratio is 2.46.

Decision Trees for Regression Now let’s create the regression decision tree using the DecisionTreeRegressor function from the sklearn.tree library. Although the DecisionTreeRegressor function has many parameters that we invite you to know and experiment with (help(DecisionTreeRegressor)), here we will see the basics to create the regression decision tree.

```
DecisionTreeRegressor(criterion
='mse',max_depth=None,min_impurity_decrease=0.0,
min_impurity_split=None min_samples_leaf=400,
min_samples_split=2,min_weight_fraction_leaf=0.0,
presort=False,random_state=None,splitter='best')
```

Basically, refer to the parameters with which the algorithm must build the tree, because it follows a recursive approach to build the tree, we must set some limits to create it.

Criterion: For the classification decision trees, we can choose Mean Absolute Error (MAE) or Mean Square Error (MSE). These criteria are related with the loss function to evaluate the performance of a learning machine algorithm and are the most used for the regression algorithms. It basically serves us to adjust the accuracy of the model, the algorithm to build the tree, and stops evaluating the branches in which no improvement is obtained according to the loss function. Here, we left the default parameter to Mean Square Error (MSE).

max_depth: Maximum number of levels the tree will have. We have left it to the default parameter of ‘none’.
min_samples_leaf: This parameter is optimisable and indicates the minimum number of leaves that we want the tree to have.

```
[]: #Regressiontree model
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(min_samples_leaf = 200)
```

Now we are going to train the model with the training datasets.

```
[]: y = X['pct_change'].shift(-1)
split = int(0.8*len(X))
X_train, X_test, y_train, y_test = X[:split], X[split:], \ y[:split], y[split:]
model = dtr.fit(X_train, y_train)
```

Visualise the Model To visualise the tree, we again use the graphviz library that gives us an overview of the regression decision tree for analysis.

```
[]: from sklearn import tree
import graphviz
dot_data = tree.export_graphviz(dtr,

out_file =None,
filled=True,
feature_names=X_train.columns)

#To createthegraph, you canuncomment #the belowlineof codeandrun it
#graphviz.Source(dot_data)
```

In the graph, we can see all the relevant information in each node:

1. The predictor variable used to split the data set.
2. The value of MSE.
3. The number of data points available at that node

Conclusion Decision trees are easy to create and it’s easy to extract some rules that promise to be useful, but the truth is that to create decision trees, they need to be parametrised and these parameters can and must be optimised.

Sometimes, you can combine different models together to create an ensemble algorithm. There are two types of ensemble methods, mainly:

Parallel Ensemble Methods or Averaging Methods Several models are created by one algorithm and the forecast is the average of the overall models. Some examples include:

1. Bagging
2. Random Subspace
3. Random Forest

Sequential Ensemble Methods or Boosting Methods
The algorithm creates sequential models refining on each new model to reduce the bias of the previous one. The examples include
1. AdaBoosting
2. Gradient Boosting
Let’s look at the random forest algorithm in the next chapter.

13 Random Forest Algorithm

Decision trees have a hierarchical or tree-like structure with branches which act as nodes. We can arrive at a certain decision by traversing through these nodes which are based on the responses garnered from the parameters related to the nodes.

However, decision trees suffer from a problem of overfitting. Overfitting is basically increasing the specificity within the tree to reach a certain conclusion by adding more and more nodes in the tree, and thereafter increasing the depth of the tree and making it more complex.

One of the solutions to overcome this issue is to use Random Forest. Let us see how.

What is a Random Forest? Random forest is a supervised classification machine learning algorithm which uses ensemble method. Simply put, a random forest is made up of numerous decision trees and helps to tackle the problem of overfitting in decision trees. These decision trees are randomly constructed by selecting random features from the given dataset.

Random forest arrives at a decision or prediction based on the maximum number of votes received from the decision trees. The outcome which is arrived, for a maximum number of times through the numerous decision trees, is considered as the final outcome by the random forest.

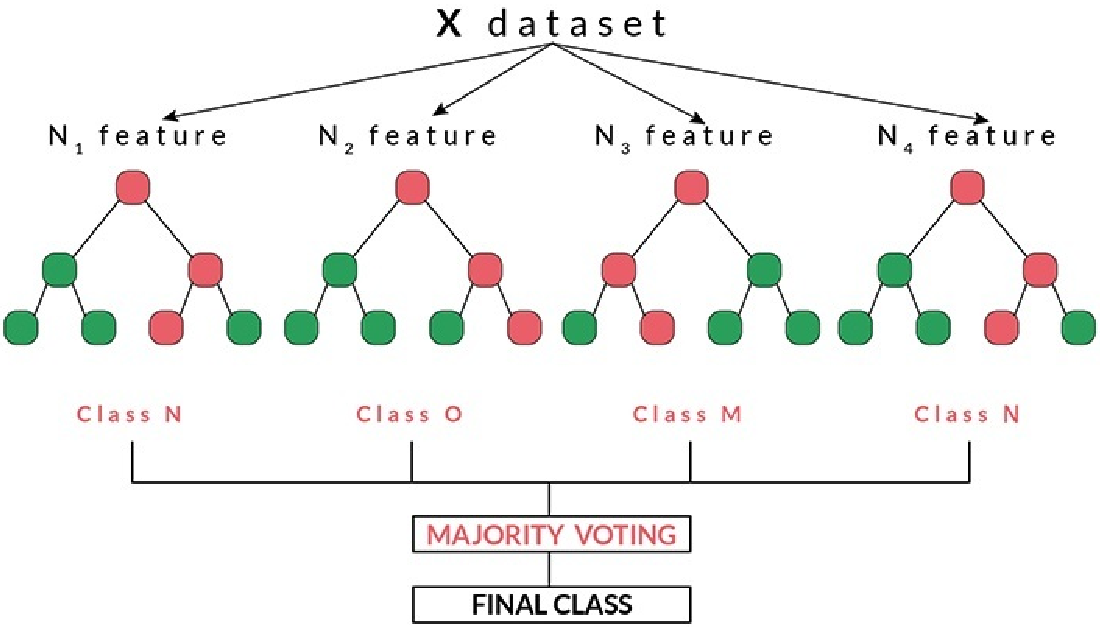
Working of Random Forest Random forests are based on ensemble learning techniques. Ensemble, simply means a group or a collection, which in this case is a collection of decision trees, together called a random forest. The accuracy of ensemble models is better than the accuracy of individual models due to the fact that it compiles the results from the individual models and provides a final outcome.

How to select features from the dataset to construct decision trees for the Random Forest?

Features are selected randomly using a method known as bootstrap aggregating or bagging. From the set of features available in the dataset, a number of training subsets are created by choosing random features with replacement. What this means is that one feature may be repeated in different training subsets at the same time.

For example, if a dataset contains 20 features and subsets of 5 features are to be selected to construct different decision trees, then these 5 features will be selected randomly and any feature can be a part of more than one subset. This ensures randomness, making the correlation between the trees less, thus overcoming the problem of overfitting.

Once the features are selected, the trees are constructed based on the best split. Each tree gives an output which is considered as a ‘vote’ from that tree to the given output. The output which receives the maximum ‘votes’ is chosen by the random forest as the final output/result or in case of continuous variables, the average of all the outputs is considered as the final output.



For example, in the above diagram, we can observe that each decision tree has voted or predicted a specific class. The final output or class selected by the Random Forest will be the Class N, as it has majority votes or is the predicted output by two out of the four decision trees.

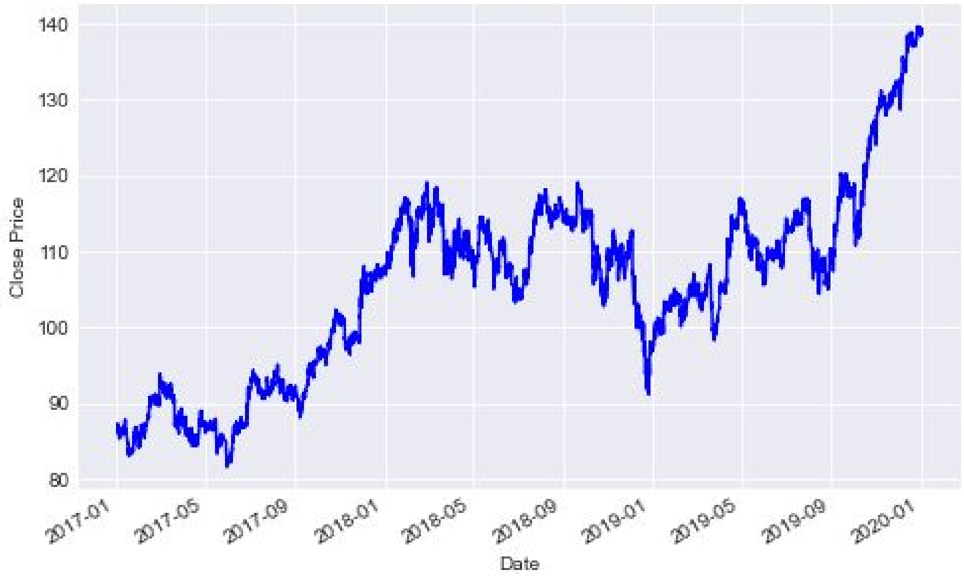
13.1 Random Forest in Trading

In this code, we will create a Random Forest Classifier and train it to give the daily returns.

Importing the Libraries

```
[]: #Data manipulation
import numpy as np
import pandas as pd

#Technicalindicators
import talib as ta
```

```
#Plotting graphs
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-darkgrid')

from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
[]: #The dataisstored inthedirectory 'data_modules'
path = "../data_modules/"

#Read thedata
data = pd.read_csv(path + 'JPM_2017_2019.csv',index_col=0) data.index =
pd.to_datetime(data.index)

data .close.plot(figsize=(8,5), color='b')
plt.ylabel('Close Price')
plt.xlabel('Date')
plt.show()
```



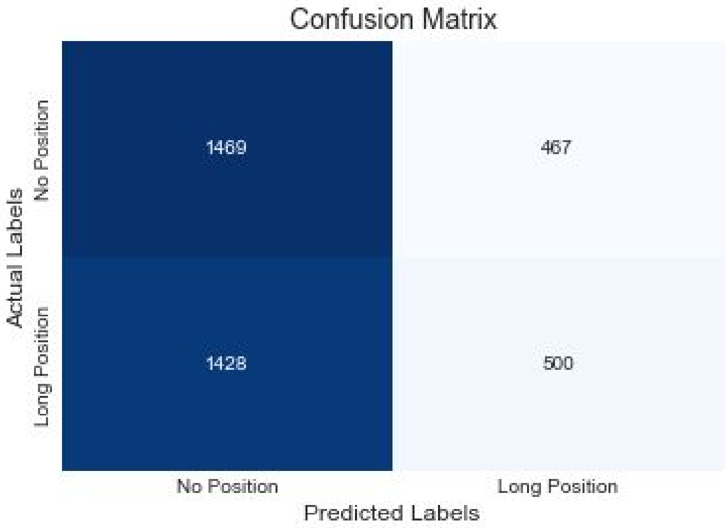
Creating Input and Output Dataset In this step, we will use the same target and feature variable as we have taken in the previous chapters.

```
[]: import sys
sys.path.append("..")
from data_modules.utility import get_target_features
y, X = get_target_features(data)
split = int(0.8*len(X))
X_train, X_test, y_train, y_test = X[:split], X[split:], \ y[:split], y[split:]
```

Training the Machine Learning Model All set with the data! Let’s train a decision tree classifier model. The RandomForestClassifier function from tree is stored in variable clf, and then a fit method is called on it with X_train and y_train dataset as the parameters so that the classifier model can learn the relationship between input and output.

```
[]: clf = RandomForestClassifier(random_state=5,max_depth=3,
max_features=3)
#Create themodelon train dataset
model = clf.fit(X_train, y_train)
predicted = model.predict(X_test)
```

```
[]: confusion_matrix_data = metrics.confusion_matrix(y_test,
predicted)
#Plot thedata
fig, ax = plt.subplots(figsize=(6, 4))
sns.heatmap(confusion_matrix_data, fmt="d",
cmap='Blues',cbar=False,annot=True,ax=ax)

#Set theaxeslabels and thetitle
ax.set_xlabel('Predicted Labels',fontsize=12) ax.set_ylabel('Actual
Labels',fontsize=12)
ax.set_title('Confusion Matrix',fontsize=14)
ax.xaxis.set_ticklabels(['No Position', 'Long Position'])
ax.yaxis.set_ticklabels(['No Position', 'Long Position'])

#Display the plot plt.show()
print(metrics.classification_report(y_test, predicted))
```



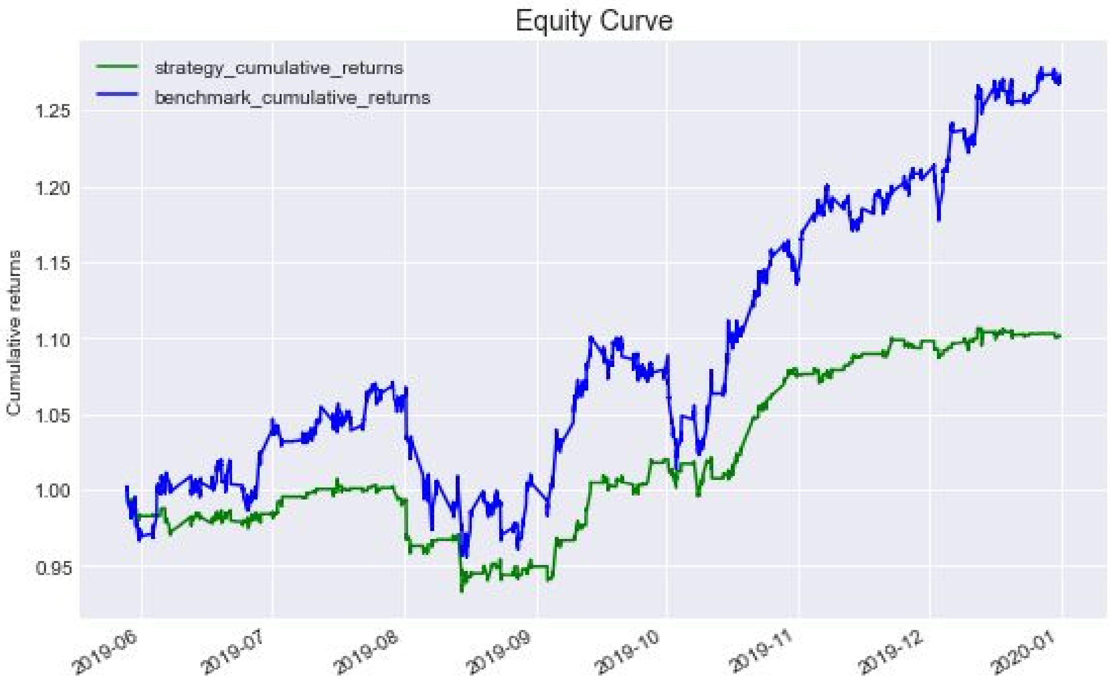
```
precision recall f1-score
support
00.51 0.76 0.61 1936
10.52 0.26 0.35 1928

accuracy 0.51 3864
macro avg 0.51 0.51 0.48 3864
weighted avg 0.51 0.51 0.48 3864
```

Strategy Returns

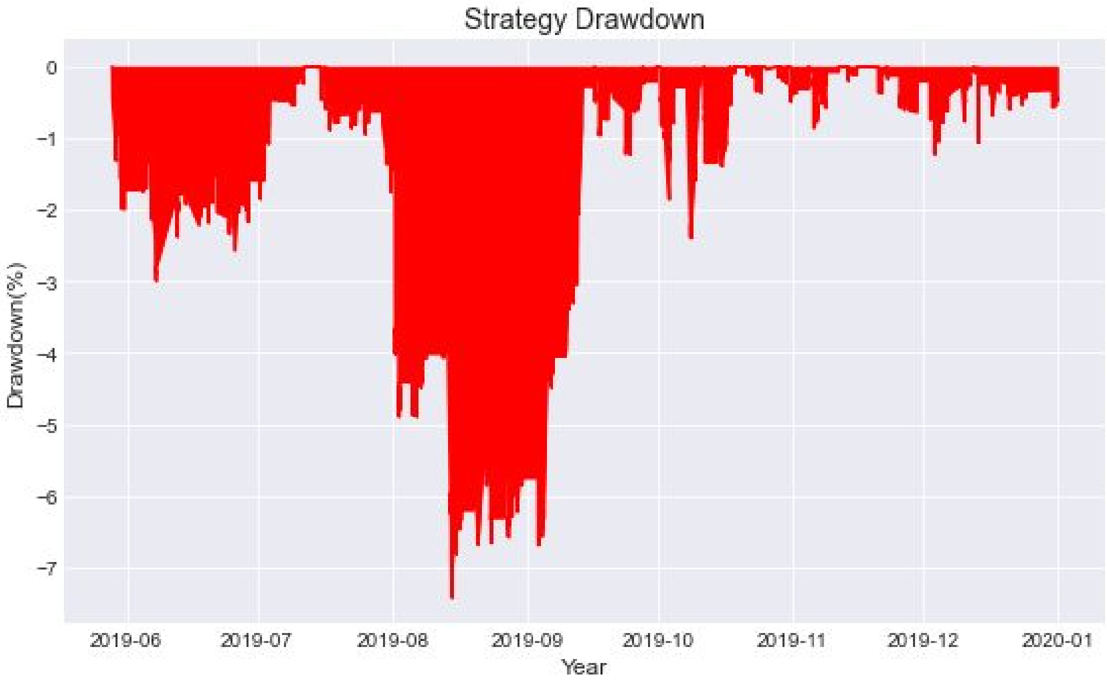
```
[]: #Calculatethe percentage change
strategy_data = X_test[['pct_change']].copy()
#Predict the signals
strategy_data['predicted_signal'] = model.predict(X_test)

#Calculatethe strategyreturns
strategy_data['strategy_returns'] = \
strategy_data['predicted_signal'].shift(1) * \
strategy_data['pct_change']

#Drop themissingvalues
strategy_data.dropna(inplace=True)
strategy_data.head()
[]: pct_change predicted_signal ,!strategy_returns
```

```
2019-05-28 12:15:00+00:00 0.000732 0 0. ,!0
2019-05-28 12:30:00+00:00 -0.000366 0 -0. ,!0
2019-05-28 12:45:00+00:00 0.000366 0 0. ,!0
2019-05-28 13:00:00+00:00 0.000091 0 0. ,!0
2019-05-28 13:15:00+00:00 -0.000091 0 -0. ,!0
```

```
[]: from data_modules.utility import get_performance
get_performance(strategy_data)
```



The maximum drawdown is -7.42%.



The Sharpe ratio is 1.63.

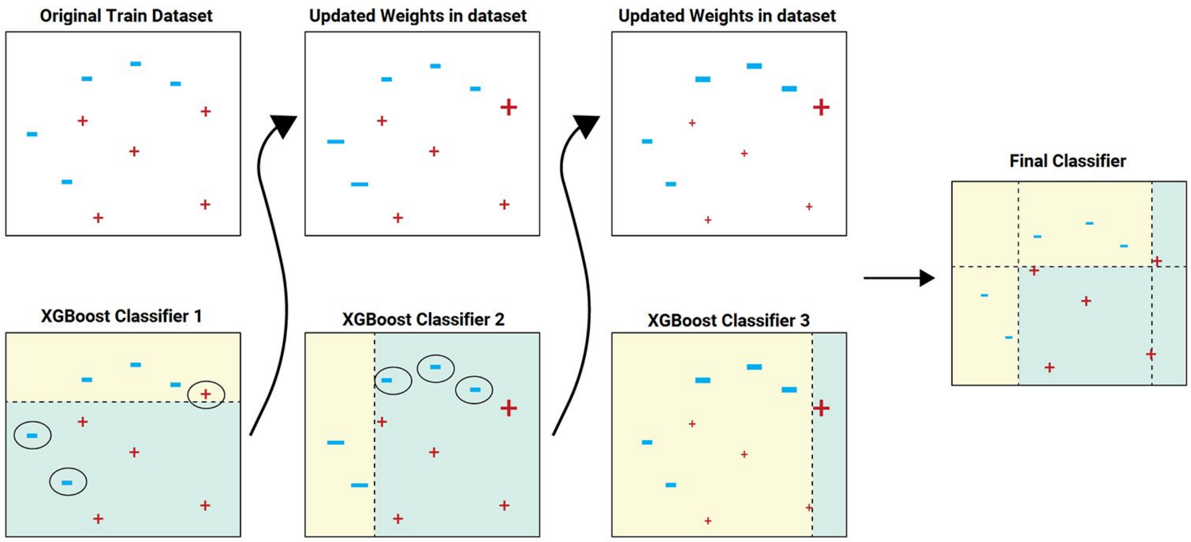
The output displays the strategy returns and daily returns according to the code for the Random Forest Classifier. In the next chapter, we will look at the XGBoost algorithm, which is the weapon of choice for most machine learning enthusiasts and competition winners alike.

14 XGBoost Algorithm

XGBoost stands for eXtreme Gradient Boosting and is developed on the framework of gradient boosting. We like the sound of that, Extreme! Sounds more like a supercar than an ML model, actually.

But that is exactly what it does, boosts the performance of a regular gradient boosting model.
“XGBoost used a more regularized model formalization to control overfitting, which gives it better performance.”
-Tianqi Chen, the author of XGBoost
Let’s break down the name to understand what XGBoost does.

What is boosting? The sequential ensemble method, also known as ‘boosting’, creates a sequence of models that attempts to correct the mistakes of the models before them in the sequence. The first model is built on training data, the second model improves the first model, the third model improves the second, and so on.



In the above image example, the train dataset is passed to classifier 1. The yellow background indicates that the classifier predicted hyphen, and the blue background indicates that it predicted plus. The classifier 1 model incorrectly predicts two hyphens and one plus. These are highlighted with a circle. The weights of these incorrectly predicted data points are increased and sent to the next classifier. That is to classifier 2.

Classifier 2 correctly predicts the two hyphens, which classifier 1 was not able to. But classifier 2 also makes some other errors. This process continues and we have a combined final classifier which predicts all the data points correctly.

The classifier models can be added until all the items in the training dataset is predicted correctly or a maximum number of classifier models are added. The optimal maximum number of classifier models to train can be determined using hyperparameter tuning.
Let’s take baby steps here and understand where does XGBoost fit in the bigger scheme of things.

In a nutshell With decision tree models, Bayesian and the like, we hit a roadblock. The prediction rate for certain problem statements was dismal when we used only one model. Apart from that, for decision trees, we realised that we had to live with bias, variance, as well as noise in the models. This led to the idea of combining models. This was and is called ‘ensemble learning’. But here, we can use much more than one model to create an ensemble. Gradient boosting was one such method of ensemble learning.

What is gradient boosting? In gradient boosting while combining the model, the loss function is minimised using gradient descent. Technically speaking, a loss function can be said as an error, i.e. the difference between the predicted value and the actual value. Of course, lesser the error, better is the machine learning model.

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models, and then added together to make the final prediction.

The objective of the XGBoost model is given as:
Obj = L + W

Where, L is the loss function which controls the predictive power, and W is regularization component which controls simplicity and overfitting.
The loss function (L) which needs to be optimised can be Root Mean Squared Error for regression, Logloss for binary classification, or mlogloss for multi-class classification. The regularization component (W) is dependent on the number of leaves and the prediction score assigned to the leaves in the tree ensemble model.

It is called gradient boosting because it uses a gradient descent algorithm to minimise the loss when adding new models. The Gradient boosting algorithm supports both regression and classification predictive modelling problems.

Alright, we have understood how machine learning evolved from simple models to a combination of models. Somehow, humans cannot be satisfied for long, and as problem statements became more complex and the data set larger, we realised that we should go one step further. This leads us to XGBoost.

Why is XGBoost so good? XGBoost was written in C++, which when you think about it, is really quick when it comes to the computation time. The great thing about XGBoost is that it can easily be imported in Python, and thanks to the sklearn wrapper, we can use the same parameter names which are used in Python packages as well.

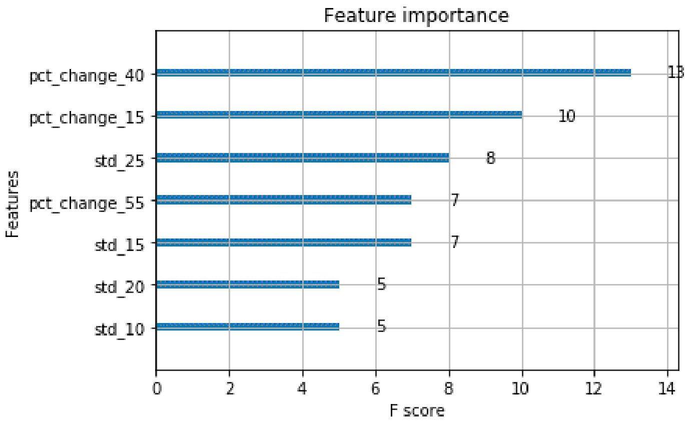
While the actual logic is somewhat lengthy to explain, one of the main things about XGBoost is that it has been able to parallelise the tree building component of the boosting algorithm. This leads to a dramatic gain in terms of processing time as we can use more cores of a CPU, or even go on and utilise cloud computing as well. While machine learning algorithms have support for tuning and can work with external programs, XGBoost has built-in parameters for regularization and cross-validation to make sure both bias and variance is kept at a minimal. The advantage of in-built parameters is that it leads to faster implementation.

Let’s discuss one such instance in the next section.

XGBoost Feature Importance Sometimes, we are not satisfied with just knowing how good our machine learning model is. We would like to know which feature has more predictive power. There are various reasons for why knowing feature importance can help us. Let us list down a few below:

1. If you know that a certain feature is more important than others, you would put more attention to it and try to see if you can improve your model further.
2. After you have run the model, you will see if dropping a few features improves the model.
3. Initially, if the dataset is small, the time taken to run a model is not a significant factor while we are designing a system. But if the strategy is complex and requires a large dataset to run, then the computing resources and the time taken to run the model becomes an important factor.
4. The good thing about XGBoost is that it contains an inbuilt function to compute the feature importance and we don’t have to worry about coding it in the model.

An example of how XGBoost depicts the feature importance is shown below:



All right, before we move on to the code, let’s make sure we all have XGBoost on our system.

How to install XGBoost in Anaconda? You can simply open the jupyter notebook and input the following: !pip install XGBoost

That’s all there is to it. Awesome! Now we move to the real thing, i.e. the XGBoost Python code.

XGBoost in Python Let’s give a summary of the XGBoost machine learning model before we dive into it. We are using the price data of US tech stocks in the US such as Apple, Amazon, Netflix, Nvidia, and Microsoft for the last sixteen years, and train the XGBoost model to predict if the next day’s returns are positive or negative.

Import Libraries

```
[]: import warnings
warnings.simplefilter('ignore')
#Import XGBoost import xgboost
#XGBoost Classifier
from xgboost import XGBClassifier

#ClassificationReportand Confusion Matrix from sklearn.metrics import
classification_report from sklearn.metrics import confusion_matrix

#Yahoo finance toget the data import yfinance as yf
```

```
#To plotthegraphs
import matplotlib.pyplot as plt import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-darkgrid')

#For datamanipulation import pandas as pd import numpy as np
```

Define Parameters We have defined the list of stock, start date, and the end date which we will be working with in this chapter.

```
[]: #Set thestocklist
stock_list = ['AAPL', 'AMZN', 'NFLX', 'WMT', 'MSFT']

#Set thestartdate andtheend date start_date = '2004-1-1'
end_date = '2021-9-1'
```

Get the Data, Create Features and Target Variable We define a list of features from which the model will pick the best ones. Here, we have the percentage change and the standard deviation with different time periods as the features.

The target variable is the next day's return. If the next day's return is positive we label it as 1, and if it is negative then we label it as -1. You can also try to create the target variables with three labels such as 1, 0, and -1 for long, no position and short respectively.
Let's see the code now.

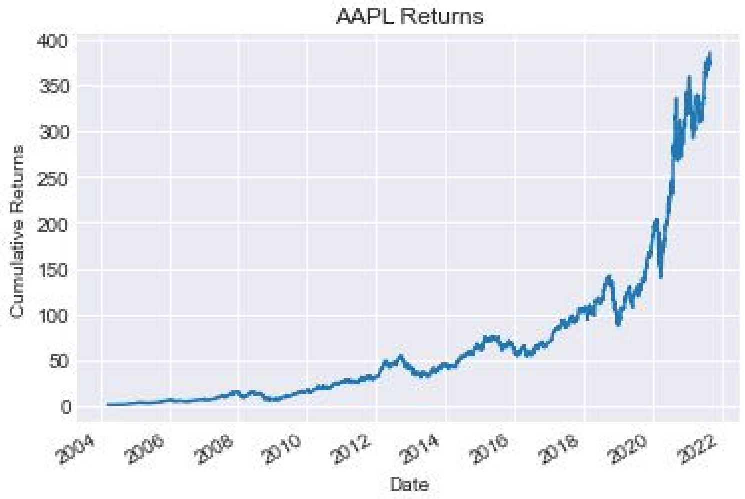
```
[]: #Create aplaceholdertostore thestockdata stock_data_dictionary = {}
for stock_name in stock_list:
#Get thedata
df = yf.download(stock_name, start_date, end_date)
#Calculatethe dailypercentchange
df['daily_pct_change'] = df['Adj Close'].pct_change()

#Create thefeatures
predictor_list = []
for r in range(10, 60, 5):

df[ 'pct_change_'+str(r)] = \
df.daily_pct_change.rolling(r).sum()
```

```
df['std_'+str(r)] = df.daily_pct_change.rolling(r).std()
predictor_list.append('pct_change_'+str(r))
predictor_list.append('std_'+str(r))

#Target Variable
df['return_next_day'] = df.daily_pct_change.shift(-1) df['actual_signal'] =
np.where(df.return_next_day > 0, 1, -1)

df = df.dropna()
#Add thedatato dictionary
stock_data_dictionary.update({stock_name: df})
```

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

Before we move on to the implementation of the XGBoost Python model, let’s first plot the daily returns of Apple stored in the dictionary to see if everything is working fine.

```
[]: #Access thedataframeofAAPL fromthedictionary
#and thencomputeand plot thereturns
(stock_data_dictionary['AAPL'].daily_pct_change+1).cumprod().plot()

#Set thetitle,axis lables, andplotgrid plt.title('AAPL Returns')
plt.ylabel('Cumulative Returns')
plt.show()
```



Split the Data into Train and Test Since XGBoost is after all a machine learning model, we will split the data set into test and train set.

```
[]: #Create aplaceholderforthe train andtest split data
X_train = pd.DataFrame()
X_test = pd.DataFrame()
y_train = pd.Series()
y_test = pd.Series()

for stock_name in stock_list:
#Get features
X = stock_data_dictionary[stock_name][predictor_list]

#Get thetargetvariable
y = stock_data_dictionary[stock_name].actual_signal

#Divide thedatasetintotrain andtest
train_length = int(len(X)*0.80)
X_train = X_train.append(X[:train_length])
X_test = X_test.append(X[train_length:])
y_train = y_train.append(y[:train_length])
y_test = y_test.append(y[train_length:])
```

Initialising the XGBoost Machine Learning Model We will initialise the classifier model. We will set two hyperparameters, namely max_depth and n_estimators. These are set on the lower side to reduce overfitting.

```
[]: #Initialisethe modelandset thehyperparameter values model =
XGBClassifier(max_depth=2,n_estimators=30) model
```

```
[]: XGBClassifier(base_score=None, booster=None,
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, gamma=None, gpu_id=None,
importance_type='gain',

↪interaction_constraints=None,
learning_rate=None, max_delta_step=None, max_depth=2,
min_child_weight=None, missing=nan,

↪monotone_constraints=None,
n_estimators=30, n_jobs=None, num_parallel_tree=None,
random_state=None, reg_alpha=None, reg_lambda=None,
scale_pos_weight=None, subsample=None, tree_method=None,
validate_parameters=None, verbosity=None)
```

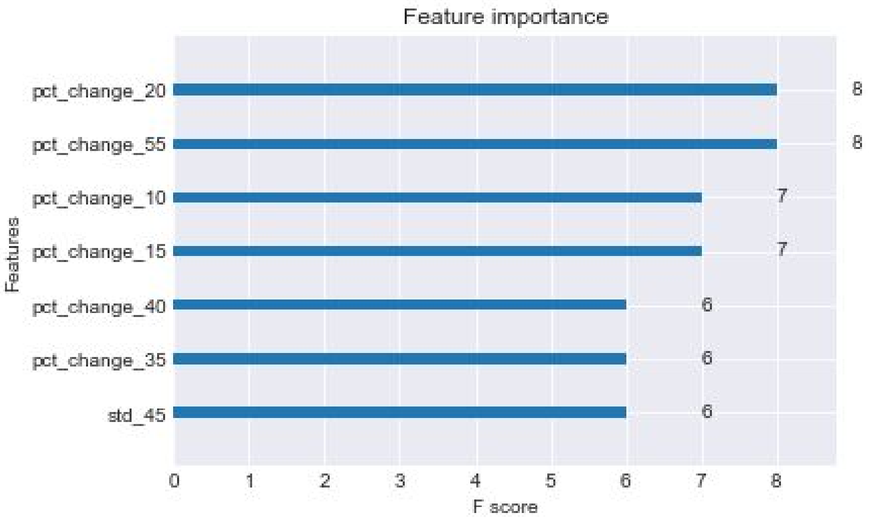
Create the Model We will train the XGBoost classifier using the fit method.

```
[]: #Fit themodel
model.fit(X_train, y_train) [05:44:56] WARNING:
C:/Users/Administrator/workspace/xgboostwin64_release_1.4.0/src/learner.c
c:1095: Starting in XGBoost 1.3.0, the

↪default
evaluation metric used with the objective 'binary:logistic' was changed
↪from
'error' to 'logloss'.Explicitlyset eval_metric ifyou'dlike to
↪restore the
old behavior.
```

```
[]: XGBClassifier(base_score=0.5, booster='gbtree',colsample_bylevel=1,
colsample_bynode=1, colsample_bytree=1, gamma=0,

↪gpu_id=-1,
importance_type='gain',interaction_constraints='',
learning_rate=0.300000012, max_delta_step=0, max_depth=2,
min_child_weight=1, missing=nan,

,!monotone_constraints='()',
n_estimators=30, n_jobs=8, num_parallel_tree=1,
,!random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
,!subsample=1,
tree_method='exact',validate_parameters=1,
,!verbosity=None)
```

Since we are using most of the default parameters, you might get certain warnings depending on updates in the XGBoost library.
Feature Importance We have plotted the top 7 features and sorted them based on their importance.

```
[]: xgboost.plot_importance(model, max_num_features=7)
#Show theplot
plt.show()
```



That’s interesting. The XGBoost Python model tells us that the pct_change_40 is the most important feature than others. Since we had mentioned that we only need 7 features, we received this list. Here’s an interesting idea, why don’t you increase the number and see how the other features stack up, when it comes to their f-score. You can also remove the unimportant features and then retrain the model. Would this increase the model accuracy? I leave that for you to verify.

Anyway, onwards we go!

Predict the Signal

```
[]: #Predict the tradingsignalon thetestdatset
y_pred = model.predict(X_test)
```

Classification Report

```
[]: #Get theclassificationreport
print(classification_report(y_test, y_pred))
precision recall f1-score support
-1 0.49 0.15 0.23 2044
10.54 0.86 0.66 2351

accuracy 0.53 4395
macro avg 0.52 0.51 0.45 4395
weighted avg 0.52 0.53 0.46 4395
```

Hold on! We are almost there. Let’s see what XGBoost tells us right now. That’s interesting. The f1-score for the long side is much more powerful compared to the short side. We can modify the model and make it a long-only strategy. Let’s try another way to formulate how well XGBoost performed.

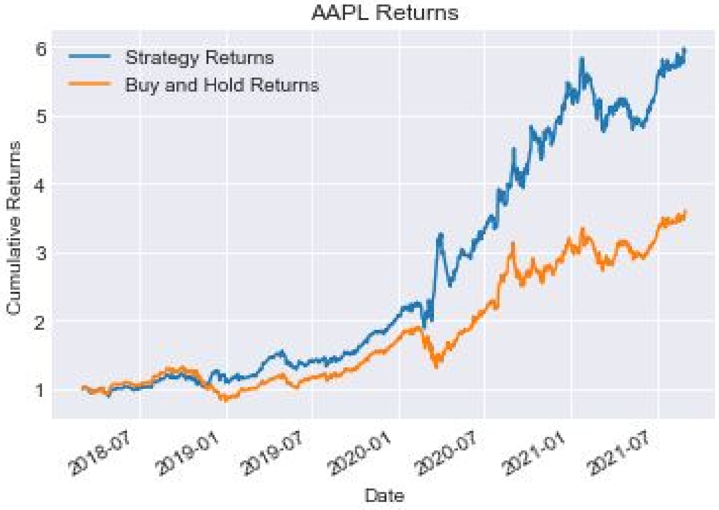
Confusion Matrix

```
[]: confusion_matrix_data = confusion_matrix(y_test, y_pred)

#Plot thedata
fig, ax = plt.subplots(figsize=(6, 4))
sns.heatmap(confusion_matrix_data, fmt="d",
cmap='Blues',cbar=False,annot=True,ax=ax)
#Set theaxeslabels and thetitle
ax.set_xlabel('Predicted Labels',fontsize=12) ax.set_ylabel('Actual
Labels',fontsize=12)
ax.set_title('Confusion Matrix',fontsize=14)
ax.xaxis.set_ticklabels(['No Position', 'Long Position'])
ax.yaxis.set_ticklabels(['No Position', 'Long Position']) #Display the plot
plt.show()
```

Individual Stock Performance Let’s see how the XGBoost based strategy returns held up against the normal daily returns, i.e. the buy and hold strategy. We will plot a comparison graph between the strategy returns and the daily returns for all the companies we had mentioned before. The code is as follows:

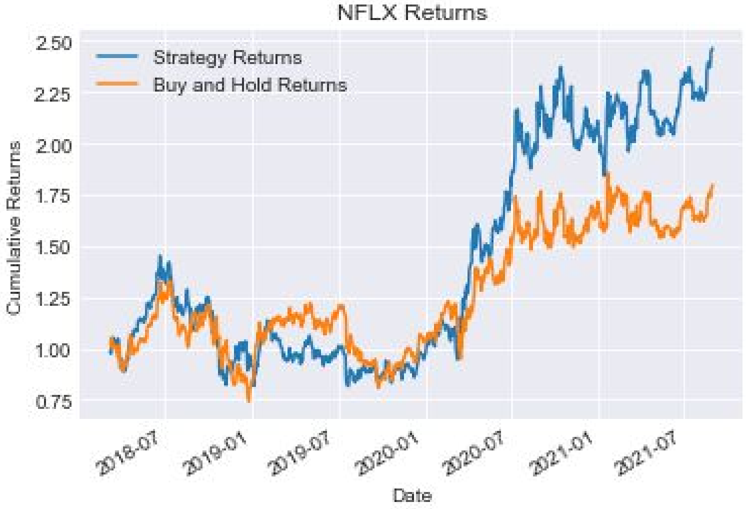
```
[]: #Create anemptydataframeto store thestrategyreturnsof #individualstocks
portfolio = pd.DataFrame(columns=stock_list)

#For eachstockin thestocklist,plot thestrategyreturns #and buyandhold
returns
for stock_name in stock_list:

#Get thedata
df = stock_data_dictionary[stock_name] #Store thefeaturesinX
X = df[predictor_list]

#Define thetrainandtest dataset train_length = int(len(X)*0.80)
#Predict the signalandstore inpredictedsignalcolumn df['predicted_signal'] =
model.predict(X)
#Calculatethe strategy returns
df['strategy_returns'] = df.return_next_day \ * df.predicted_signal
#Add thestrategyreturnsto theportfoliodataframe portfolio[stock_name] =
df.strategy_returns[train_length:]
#Plot thestockstrategyand buyandhold returns print(stock_name)
#Calculatethe cumulative strategyreturnsandplot
(df.strategy_returns[train_length:]+1).cumprod().plot()
#Calculatethe cumulative buyandhold strategy returns
(stock_data_dictionary[stock_name][train_length:].
,!daily_pct_change+1).cumprod().plot()

#Set thetitle,labeland grid
plt.title(stock_name + ' Returns')
plt.ylabel('Cumulative Returns')
plt.legend(labels=['Strategy Returns', 'Buy and Hold Returns']) plt.show()
```

AAPL

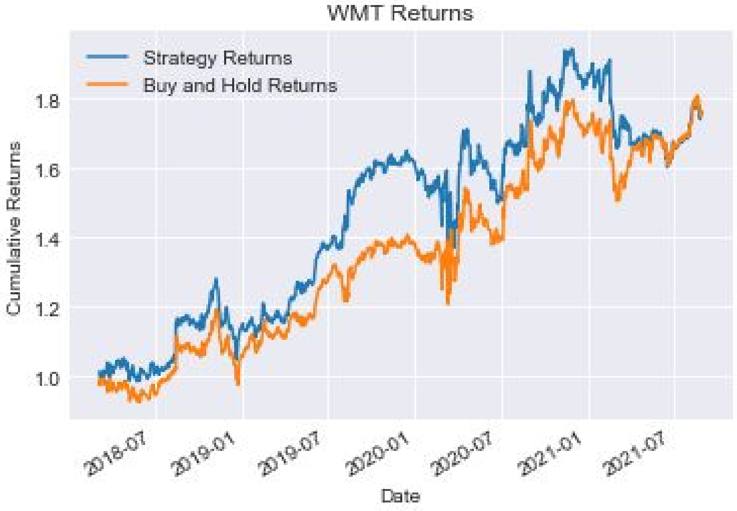


AMZN

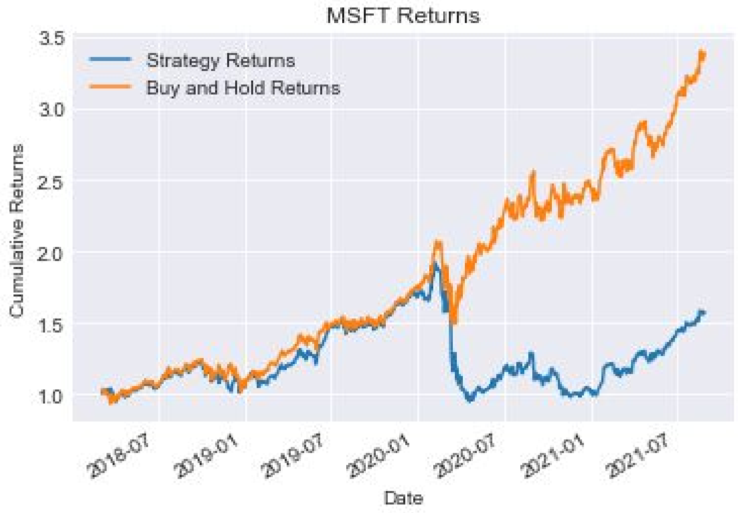
NFLX



WMT

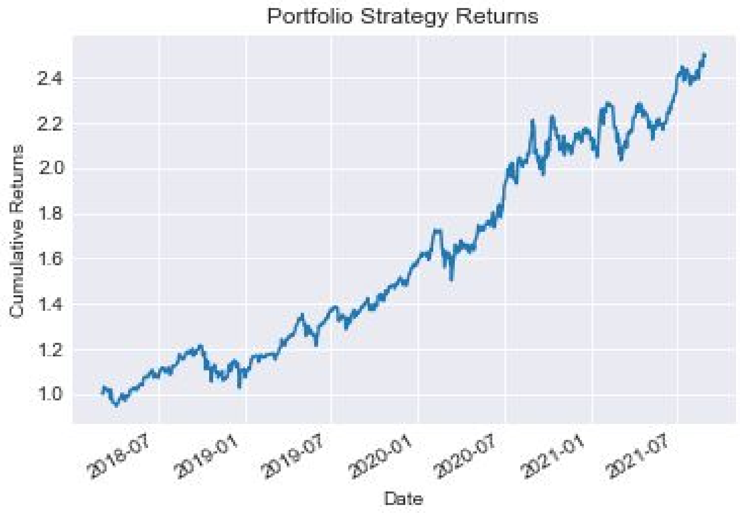


MSFT



Performance of Portfolio

```
[]: #Drop themissingvalues
portfolio.dropna(inplace=True)

#Calculatethe cumulative portfolio returnsbyassuming #equal allocation
tothestocks
(portfolio.mean(axis=1)+1).cumprod().plot()

#Set thetitleand label ofthe chart
plt.title('Portfolio Strategy Returns')
plt.ylabel('Cumulative Returns')
plt.show()
```



Conclusion We started from the emergence of machine learning algorithms and moved to ensemble learning. We learnt about boosted trees and how they help us in making better predictions. We finally came to XGBoost machine learning model and how it is better than a regular boosted algorithm. We then went through a simple XGBoost Python code and created a portfolio based on the trading signals created by the code. In between, we also listed down feature importance as well as certain parameters included in XGBoost.

Now let us see another interesting concept, called the neural networks in the next chapter.

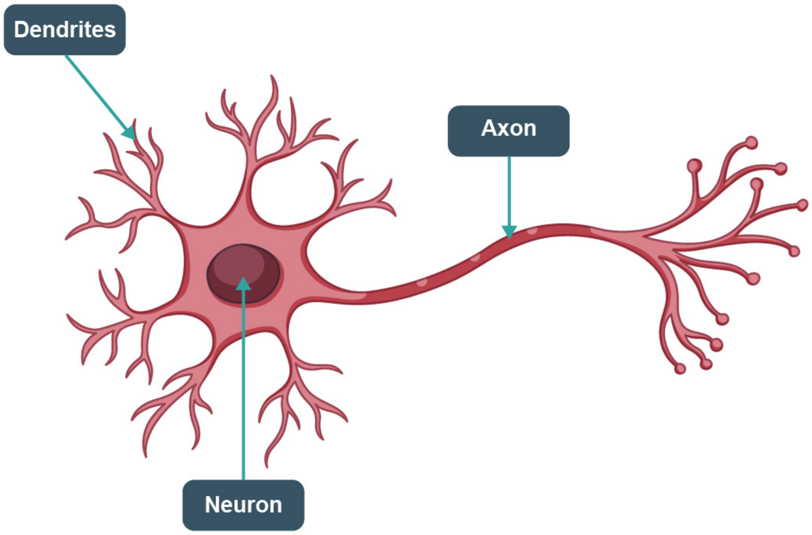
15 Neural Networks

Neural network studies were started in an effort to map the human brain and understand how humans take decisions. But algorithmic trading tries to remove human emotions altogether from the trading aspect. Why should we learn about neural networks then?

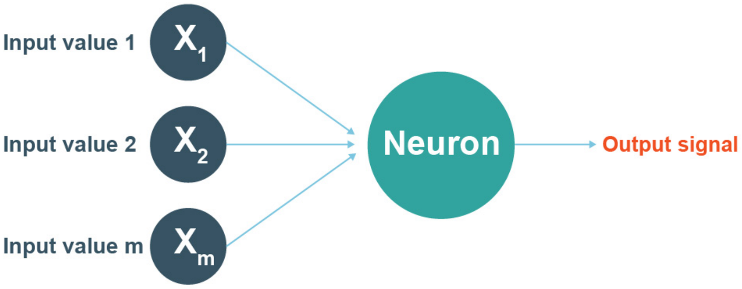
We sometimes fail to realise that the human brain is quite possibly the most complex machine in this world and has been known to be quite effective at coming to conclusions in record time.

Think about it. If we could harness the way our brain works and apply it in the machine learning domain (Neural networks are after all a subset of machine learning), we could possibly take a giant leap in terms of processing power and computing resources.

Before we dive deep into the nitty-gritty of neural network trading, we should understand the working of the principal component, i.e. the neuron.



There are three components to a neuron: The dendrites, axon and the main body of the neuron. The dendrites are the receivers of the signal and the axon is the transmitter. Alone, a neuron is not of much use, but when it is connected to other neurons, it does several complicated computations and helps operate the most complicated machine on our planet, the human body.



Perceptron: the Computer Neuron A perceptron, i.e. a computer neuron is designed in a similar manner, as shown in the diagram.

There are inputs to the neuron marked with blue lines, and the neuron emits an output signal after some computation.
The input layer resembles the dendrites of the neuron and the output signal is the axon.
Each input signal is assigned a weight, wi. This weight is multiplied by the input value and the neuron stores the weighted sum of all the input variables.

These weights are computed in the training phase of the neural network learning through concepts called gradient descent and backpropagation, we will cover these topics later on.

An activation function is then applied to the weighted sum, which results in the output signal of the neuron.
The input signals are generated by other neurons, i.e. the output of other neurons, and the network is built to make predictions/computations in this manner. This is the basic idea of a neural network.
Understanding a Neural Network Let’s take an example to understand the working of neural networks.

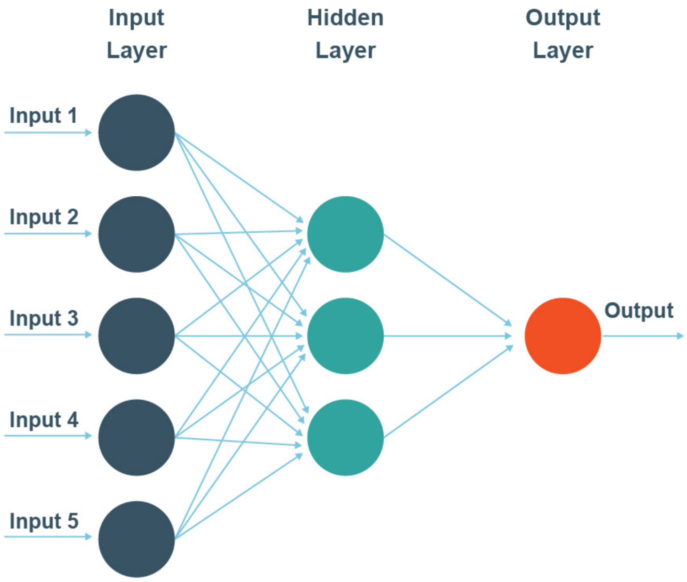
The input layer consists of the parameters that will help us arrive at an output value or make a prediction. Our brains essentially have five basic input parameters, which are our senses to touch, hear, see, smell and taste.

The neurons in our brain create more complicated parameters such as emotions and feelings, from these basic input parameters. And our emotions and feelings, make us act or take decisions which is basically the output of the neural network of our brains.

Therefore, there are two layers of computations in this case before making a decision.

The first layer takes in the five senses as inputs and results in emotions and feelings, which are the inputs to the next layer of computations, where the output is a decision or an action.
Hence, in this extremely simplistic model of the working of the human brain, we have one input layer, two hidden layers, and one output layer. Of course from our experiences, we all know that the brain is much more complicated than this, but essentially this is how the computations are done in our brain.

Neural Network In Trading: An Example To understand the working of a neural network in trading, let us consider a simple stock price prediction example, where the OHLCV (Open-High-Low-Close-Volume) values are the input parameters, there is one hidden layer and the output consists of the prediction of the stock price.



In the diagram shown above, there are five input parameters as shown in the diagram. The hidden layer consists of 3 neurons and the resultant in the output layer is the prediction for the stock price.

The 3 neurons in the hidden layer will have different weights for each of the five input parameters and might have different activation functions, which will activate the input parameters according to various combinations of the inputs.

For example, the first neuron might be looking at the volume and the difference between the Close and the Open price, and might be ignoring the High and Low prices as well. In this case, the weights for High and Low prices will be zero.

Based on the weights that the model has trained itself to attain, an activation function will be applied to the weighted sum in the neuron, this will result in an output value for that particular neuron.

Similarly, the other two neurons will result in an output value based on their individual activation functions and weights. Finally, the output value or the predicted value of the stock price will be the sum of the three output values of each neuron. This is how the neural network will work to predict stock prices.

Now that you understand the working of a neural network, we will move to the heart of the matter of this chapter, and that is learning how the Artificial Neural Network will train itself to predict the movement of a stock price.
Training the Neural Network To simplify things in neural networks, we can say that there are two ways to code a program for performing a specific task.

1. Define all the rules required by the program to compute the result given some input to the program.

2. Develop the framework upon which the code will learn to perform the specific task. This task is carried out by training itself on a dataset, and adjusting the result it computes to be as close to the actual results which have been observed.

The second process is called training the model which is what we will be focussing on. Let’s look at how our neural network will train itself to predict stock prices.

The neural network will be given the dataset, which consists of the OHLCV data as the input, as well as the output, which is the close price of the next day. This output variable is the value that we want our model to learn to predict. The actual value of the output will be represented by ‘y’ and the predicted value will be represented by ˆy.

The training of the model involves adjusting the weights of the variables for all the different neurons present in the neural network. This is done by minimizing the ‘Cost Function’. The cost function, as the name suggests is the cost of making a prediction using the neural network. It is a measure of how far off the predicted value, ˆy, is from the actual or observed value, y.

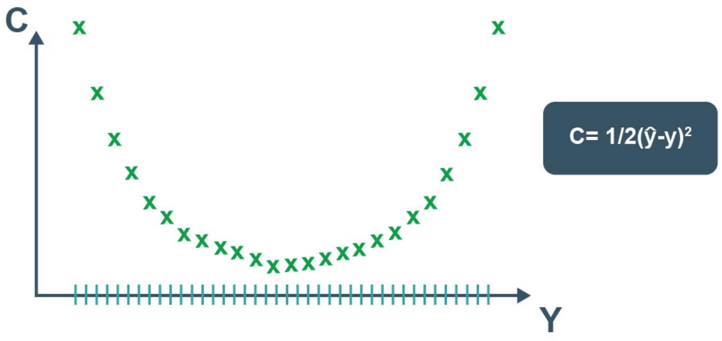
There are many cost functions that are used in practice, the most popular one is computed as half of the sum of squared differences between the actual and predicted values for the training dataset.

C =Â1/2(ˆy y)^2

The way the neural network trains itself is by first computing the cost function for the training dataset for a given set of weights for the neurons. Then it goes back and adjusts the weights, followed by computing the cost function for the training dataset based on the new weights.

The process of sending the errors back to the network for adjusting the weights is called backpropagation.
This is repeated several times till the cost function has been minimised. We will look at how the weights are adjusted and the cost function is minimised in more detail next. The weights are adjusted to minimise the cost function. One way to do this is through brute force.
Suppose we take 1000 values for the weights, and evaluate the cost function for these values.
When we plot the graph of the cost function, we will arrive at a graph as shown below.

The best value for weights would be the cost function corresponding to the minima of this graph.



This approach could be successful for a neural network involving a single weight which needs to be optimised.

However, as the number of weights to be adjusted and the number of hidden layers increases, the number of computations required will increase drastically.

The time it will require to train such a model will be extremely large even on the world’s fastest supercomputer. For this reason, it is essential to develop a better, faster methodology for computing the weights of the neural network.

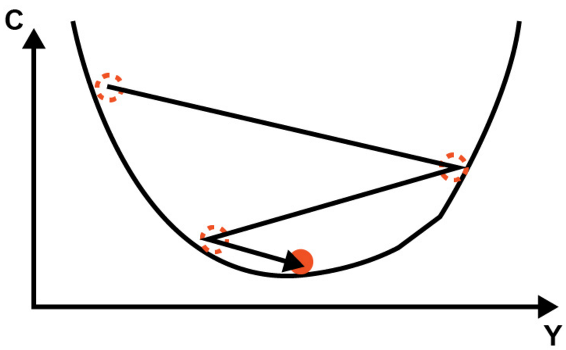
This process is called Gradient Descent. We will look into this concept in the next part.

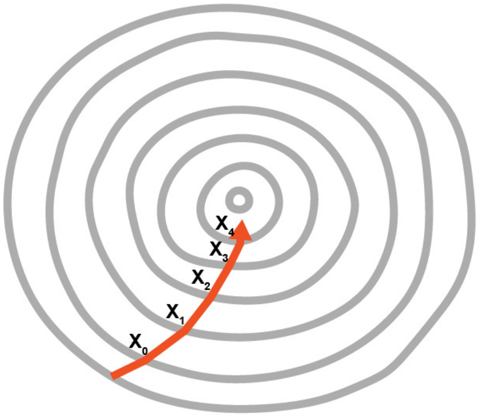
Gradient Descent Gradient descent involves analysing the slope of the curve of the cost function. Based on the slope we adjust the weights, to minimise the cost function in steps rather than computing the values for all possible combinations.

The visualization of Gradient descent is shown in the diagrams below. The first plot is a single value of weights and hence is two dimensional. It can be seen that the red ball moves in a zig-zag pattern to arrive at the minimum of the cost function.

In the second diagram, we have to adjust two weights in order to minimise the cost function.

Therefore, we can visualise it as a contour, as shown in the graph, where we are moving in the direction of the steepest slope, in order to reach the minima in the shortest duration. With this approach, we do not have to do many computations and as a result, the computations do not take very long, making the training of the model a feasible task.





Gradient descent can be done in three possible ways:

1. Batch Gradient Descent: In batch gradient descent, the cost function is computed by summing all the individual cost functions in the training dataset and then computing the slope and adjusting the weights.

2. Stochastic Gradient Descent: The slope of the cost function and the adjustments of weights are done after each data entry in the training dataset. This is extremely useful to avoid getting stuck at a local minima if the curve of the cost function is not strictly convex. Each time you run the stochastic gradient descent, the process to arrive at the global minima will be different. Batch gradient descent may result in getting stuck with a suboptimal result if it stops at local minima.

3. Mini-batch Gradient Descent: It is a combination of the batch and stochastic methods. Here, we create different batches by clubbing together multiple data entries in one batch. This essentially results in implementing the stochastic gradient descent on bigger batches of data entries in the training dataset.

While we can dive deep into Gradient descent, we are afraid it will be outside the scope of this chapter. Hence, let us move forward and understand how backpropagation works to adjust the weights according to the error which had been generated.

Backpropagation Backpropagation is an advanced algorithm which enables us to update all the weights in the neural network simultaneously.
This drastically reduces the complexity of the process to adjust weights. If we were not using this algorithm, we would have to adjust each weight individually by figuring out what impact that particular weight has on the error in the prediction. Let us look at the steps involved in training the neural network with Stochastic Gradient Descent:

1. Initialise the weights to small numbers very close to 0 (but not 0).
2. Forward propagation: The neurons are activated from left to right, by using the first data entry in our training dataset, until we arrive at the predicted result y.
3. Measure the error which will be generated.
4. Backpropagation: The error generated will be backpropagated from right to left, and the weights will be adjusted according to the learning rate.
5. Repeat the previous three steps, forward propagation, error computation, and backpropagation on the entire training dataset.

This would mark the end of the first epoch, the successive epochs will begin with the weight values of the previous epochs, we can stop this process when the cost function converges within a certain acceptable limit.

We will now learn how to develop our own Artificial Neural Network to predict the movement of a stock price.
You will understand how to code a strategy using the predictions from a neural network that we will build from scratch.

15.1 Neural Networks in Trading

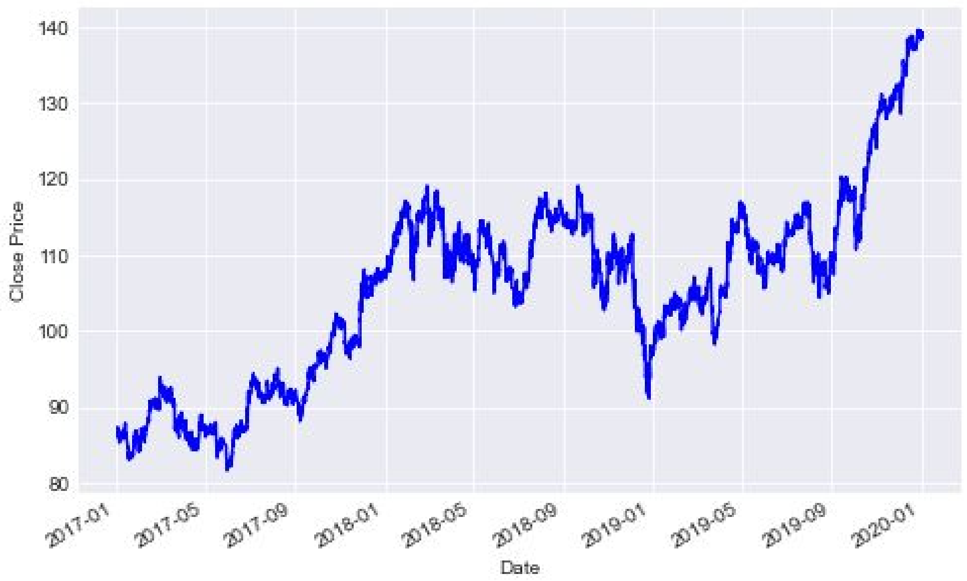
Importing Libraries We will start by importing a few libraries, the others will be imported as when they are used in the program at different stages. For now, we will import the libraries which will help us in importing and preparing the dataset for training and testing the model.

```
[]: #Data manipulation
import numpy as np
import pandas as pd

#Technicalindicators import talib as ta

#Plotting graphs
import matplotlib.pyplot as plt import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-darkgrid')

from sklearn import metrics

#Setting the randomseedto afixednumber import random
random.seed(42)
```

Random will be used to initialise the seed to a fixed number so that every time we run the code we start with the same seed.

Import dataset

```
[]: #The dataisstored inthedirectory 'data_modules' path = "../data_modules/"
#Read thedata
data = pd.read_csv(path + 'JPM_2017_2019.csv',index_col=0) data.index =
pd.to_datetime(data.index)

data .close.plot(figsize=(8,5), color='b')
plt.ylabel('Close Price')
plt.xlabel('Date')
plt.show()
```



Define Target, Features and Split the Data

```
[]: import sys
sys.path.append("..")
from data_modules.utility import get_target_features
y, X = get_target_features(data)
split = int(0.8*len(X))
X_train, X_test, y_train, y_test = X[:split], X[split:], \ y[:split], y[split:]
```

Feature Scaling

```
[]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_test_original = X_test.copy()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Another important step in data preprocessing is to standardise the dataset. This process makes the mean of all the input features equal to zero and also converts their variance to 1. This ensures that there is no bias while training the model due to the different scales of all input features. If this is not done, the neural network might get confused and give a higher weight to those features which have a higher average value than others.

We implement this step by importing sklearn.preprocessing library. We StandardScaler() function. After which we use the fit_transform function for implementing these changes on the X_train and X_test datasets. The y_train

and y_test sets contain binary values, hence they need not be standardised. Now that the datasets are ready, we may proceed with building the Artificial Neural Network using the Keras library.
the StandardScaler method from the instantiate the variable sc with the

Building the Artificial Neural Network

```
[]: #Building the Artificial NeuralNetwork
from keras.models import Sequential

from keras.layers import Dense
from keras.layers import Dropout
```

Now we will import the functions which will be used to build the Artificial Neural Network. We import the Sequential method from the keras.models library. This will be used to sequentially build the layers of the neural networks learning. The next method that we import will be the Dense function from the keras.layers library.

This method will be used to build the layers of our Artificial Neural Network. []: classifier = Sequential()

We instantiate the Sequential() function into the variable classifier. This variable will then be used to build the layers of the Artificial Neural Network learning in Python.

```
[]: classifier.add(Dense(
units = 128,
kernel_initializer = 'uniform',
activation = 'relu',
input_dim = X.shape[1]
))
```

To add layers into our Classifier, we make use of the add() function. The argument of the add function is the Dense() function, which in turn has the following arguments: **Units:** This defines the number of nodes or neurons in that particular layer. We have set this value to 128, meaning there will be 128 neurons in our hidden layer.

Kernel_initializer: This defines the starting values for the weights of the different neurons in the hidden layer. We have defined this to be ‘uniform’, which means that the weights will be initialised with values from a uniform distribution.

Activation: This is the activation function for the neurons in the particular hidden layer. Here we define the function as the rectified Linear Unit function or ‘relu’. **Input_dim:** This defines the number of inputs to the hidden layer, we have defined this value to be equal to the number of columns of our input feature dataframe. This argument will not be required in the subsequent layers, as the model will know how many outputs the previous layer produced.

```
[]: classifier.add(Dense(
units = 128,
kernel_initializer = 'uniform',
activation = 'relu'
))
```

We then add a second layer, with 128 neurons, with a uniform kernel initialiser and ‘relu’ as its activation function. We are only building two hidden layers in this neural network.

```
[]: classifier.add(Dense(
units = 1,
kernel_initializer = 'uniform',
activation = 'sigmoid'
))
```

The next layer that we build will be the output layer, from which we require a single output. Therefore, the units passed are 1, and the activation function is chosen to be the Sigmoid function because we would want the prediction to be a probability of market moving upwards.

```
[]: classifier.compile(
optimizer = 'adam',
loss = 'mean_squared_error', metrics = ['accuracy']
)
```

Finally, we compile the classifier by passing the following arguments:
Optimizer: The optimizer is chosen to be ‘adam’, which is an extension of the stochastic gradient descent.
Loss: This defines the loss to be optimised during the training period. We define this loss to be the mean squared error.
Metrics: This defines the list of metrics to be evaluated by the model during the testing and training phase. We have chosen accuracy as our evaluation metric. []: classifier.fit(X_train, y_train, batch_size = 20,epochs = 7)

```
Epoch 1/7
773/773 [==============================] - 1s 1ms/step - loss:
0.2487 accuracy: 0.5323
Epoch 2/7
773/773 [==============================] - 1s 1ms/step - loss:
0.2483 accuracy: 0.5354
Epoch 3/7
773/773 [==============================] - 1s 1ms/step - loss:
0.2482 accuracy: 0.5319
Epoch 4/7
773/773 [==============================] - 1s 1ms/step - loss:
0.2479 accuracy: 0.5374
Epoch 5/7
773/773 [==============================] - 1s 1ms/step - loss:
0.2477 accuracy: 0.5422
Epoch 6/7
773/773 [==============================] - 1s 1ms/step - loss:
0.2474 accuracy: 0.5412
Epoch 7/7
773/773 [==============================] - 1s 1ms/step - loss:
0.2473 accuracy: 0.5416
```

```
[]: <keras.callbacks.History at 0x242305bc400>
```

Now we need to fit the neural network that we have created to our train datasets. This is done by passing X_train, y_train, batch size and the number of epochs in the fit() function.
The batch size refers to the number of data points that the model uses to compute the error before backpropagating the errors and making modifications to the weights. The number of epochs represents the number

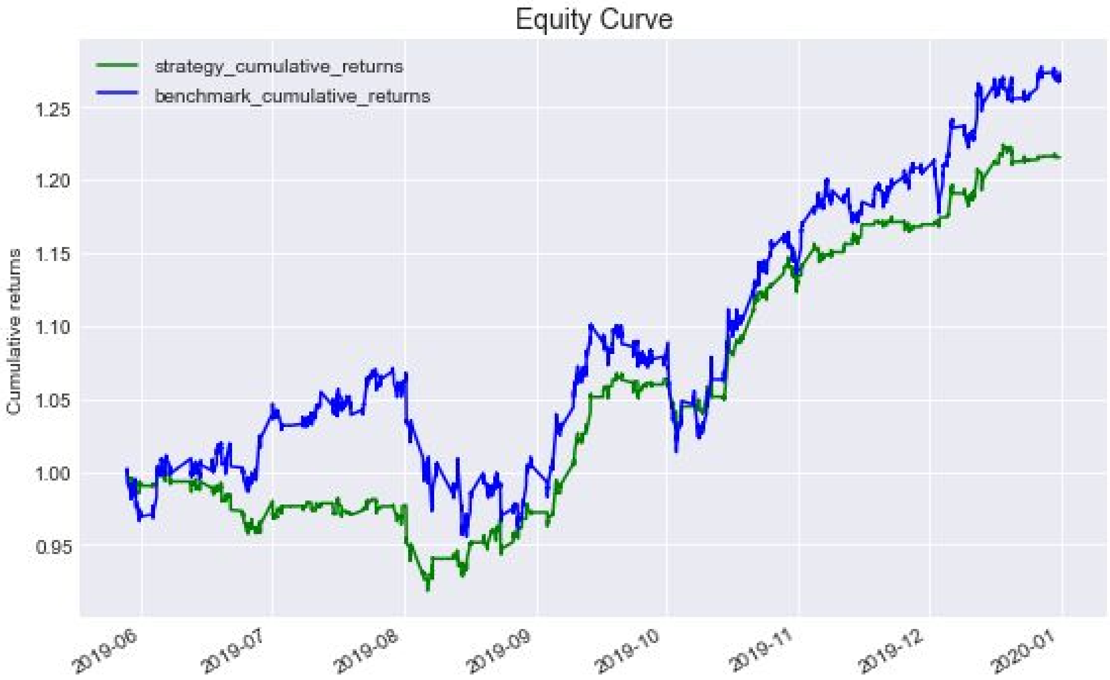
of times the training of the model will be performed on the train dataset. With this, our Artificial Neural Network in Python has been compiled and is ready to make predictions.

Predicting the Movement of the Stock []: predicted = classifier.predict(X_test) predicted = np.where(predicted>0.5,1,0)

```
[]: from data_modules.utility import get_metrics get_metrics(y_test,
predicted)
precision recall f1-score support 00.51 0.58 0.54 1936 10.51 0.44 0.47 1928

accuracy 0.51 3864
macro avg 0.51 0.51 0.51 3864
weighted avg 0.51 0.51 0.51 3864
```

Computing Strategy Returns

```
[]: #Calculatethe percentage change
strategy_data = X_test_original[['pct_change']].copy()
#Predict the signals
strategy_data['predicted_signal'] = predicted

#Calculatethe strategyreturns
strategy_data['strategy_returns'] = \
strategy_data['predicted_signal'].shift(1) * \
strategy_data['pct_change']

#Drop themissingvalues
strategy_data.dropna(inplace=True) strategy_data.head() []: pct_change
predicted_signal ,!strategy_returns

2019-05-28 12:15:00+00:00 0.000732 0 0. ,!000000
2019-05-28 12:30:00+00:00 -0.000366 1 -0. ,!000000
2019-05-28 12:45:00+00:00 0.000366 0 0. ,!000366
2019-05-28 13:00:00+00:00 0.000091 0 0. ,!000000
2019-05-28 13:15:00+00:00 -0.000091 0 -0. ,!000000

[]: from data_modules.utility import get_performance
get_performance(strategy_data)
```



The maximum drawdown is -8.57%.
The Sharpe ratio is 2.72.

This is interesting! While the strategy returns were less than the benchmark returns, you should note that the drawdown percentage is relatively lower. Nevertheless, the neural network strategy can be improved by letting the algorithm process more data to improve its performance scores.

So far we looked at supervised learning algorithms. In the next part, we will look at unsupervised learning and its algorithms.

Additional Reading

1. Grokking Linear Regression Analysis in Finance - https://blog.quantinsti.com/linear-regression/
2. Linear regression on market data - Implemented from scratch in Python and R https://blog.quantinsti.com/linear-regression-market-data-python-r/
3. Gold Price Prediction Using Machine Learning In Python - https://blog.quantinsti.com/gold-price-prediction-using-machine-learningpython/
4. Trading with Machine Learning: Regression [Course] - https://quantra.quantinsti.com/course/trading-with-machine-

learningregression
5. Linear Regression vs Logistic Regression - https://www.javatpoint.com/linearregression-vs-logistic-regression-in-machine-learning
6. Uses Logistic Regression Analysis to Explore Between the Financial Crises and Financial Statement Restatements Relationship Under the Digital Transformation - https://ieeexplore.ieee.org/document/9382566
7. Prediction of Stock Performance in the Indian Stock Market Using Logistic Regression - https://www.semanticscholar.org/paper/Prediction-of-StockPerformance-in-the-Indian-Stock-Dutta-Bandopadhyay/082826e9adf1c3ce3ff9f70491566719772fdc8a
8. Predicting future trends in stock market by decision tree rough-set based hybrid system with HHMM - https://www.researchgate.net/publication/267988069_Predicting_future_trends_in_stock_market_by_decision_tree_roughset_based_hybrid_system_with_HHMM
9. Understanding Naive Bayes - https://stats.stackexchange.com/questions/21822/understandingnaive-bayes
10. Decision Trees in Trading [Course] - https://quantra.quantinsti.com/course/decisiontrees-analysis-trading-ernest-chan
11. Understanding LSTM Understanding-LSTMs/ 12. A Gentle Introduction Networks - -

to Exploding Gradients in Neural Networks
- https://machinelearningmastery.com/exploding-gradients-in-neuralnetworks/

13. HOW THE KAGGLE WINNERS ALGORITHM XGBOOST WORKS - https://dataaspirant.com/xgboost-algorithm/
14. Ensemble Learning to Improve Machine Learning https://blog.statsbot.co/ensemble-learning-d1dcd548e936
15. Quantitative Tactical Asset Allocation Using Ensemble Machine Learning Methods - https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2438522 ALGORITHM

Results -

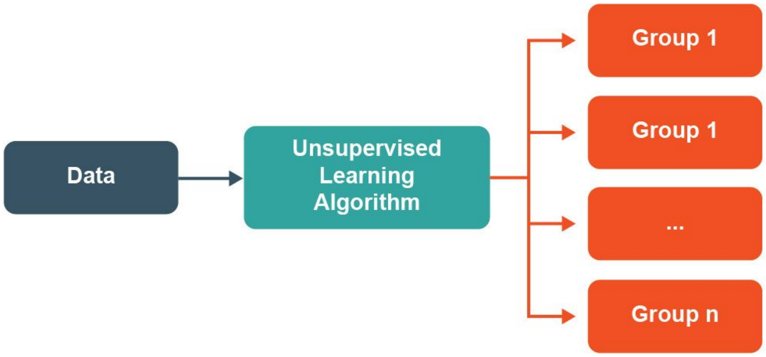
Part IV
Unsupervised Learning, Natural Language Processing, and Reinforcement Learning 16 Unsupervised Learning

In the previous chapters, we examined supervised learning algorithms like classification and regression in detail. The common element of these algorithms is that you know the labels and target variable in supervised learning models.

But if you think about it, humans have another approach towards learning too. A baby starts with zero knowledge of the surroundings and is introduced to his parents who are humans. Even though the baby might not be able to speak, he knows how to differentiate between things. When he sees a pet dog, he will first think if it is some other being, which is different from his parents. When he sees another pet dog, he sees that this pet dog has more things in common, such as a tail, with the first dog than a human. In this way, while he does not know that he is seeing a ‘dog’, he has learnt to put dogs in a different box than humans.

He repeats the same process when he sees a cat. He knows it one is different from both dogs and humans. This process, when replicated in machine learning is called clustering, which is a part of unsupervised learning.

Unsupervised learning algorithms can also help to discover hidden patterns in the dataset. It can help you to group or cluster similar data points together. You can then analyse these groups and extract insights from them.



Let us see how this can be applied in trading.

Suppose, you want to analyse the stocks that constitute the S&P 500 index and find two similar stocks. This will help you apply a pairs trading strategy on the similar stocks identified. But analysing the 500 stocks fundamental and price data, and finding similarities is a very tedious and time-consuming task.

You can use an unsupervised learning algorithm here. Just pass the relevant data for 500 stocks to the algorithm. In return, it will cluster similar stocks together. For example, it will create a cluster of Google and Facebook, and another cluster of Citigroup and Bank of America, and so on. You can then pick a pair of Citigroup and Bank of America, and create a statistical arbitrage trading strategy on it.

This can work not only on 500 stocks, but also on stocks from different geographies or even from different asset classes such as FX, commodities or bonds. The unsupervised algorithm will group them together in a matter of seconds.
This is useful not only for pairs trading but you can use these groups to create diversified portfolios. Remember, the securities which are similar to each other are placed together in a single cluster. And security in one cluster is dissimilar to security in another cluster.

You can pick top-performing security from each cluster and create a diversified portfolio. For example, you can pick Bank of America, Facebook, Gold ETF, EURUSD, and US Treasury and create a diversified portfolio.

Let’s take a small example here to drive home the point.
Name of the Company Sector

Facebook Tech Bank of America Bank Amazon Tech Google Tech Microsoft Tech Ford Auto J.P. Morgan Bank Wells Fargo Bank Intel Tech Tesla Auto

You want to divide these companies into two groups. One of the many possible ways to do this can be as shown below:

- Group 1: Facebook, Amazon, Google, Microsoft, Intel, Apple
- Group 2: Bank of America, J.P. Morgan, Wells Fargo, Ford, Tesla Can you think of how this grouping was done?

Group 1 comprises all the tech companies in the list. And group 2 comprises all the non-tech companies.
Similarly, if you had to divide these companies into three groups, it can be done as shown:

- Group 1: Amazon, Google, Microsoft, Facebook, Apple
- Group 2: Bank of America, J.P. Morgan, Wells Fargo
- Group 3: Ford, Tesla

Here, group 1 comprises the tech companies, group 2 is banks, and group 3 is automobiles. From these groupings, you can conclude that the stocks within a group are similar to each other. And they differ across the different groups. Each group is called a cluster.

The next question that arises is what if instead of ten, you have a list of thousand companies? What if, not only the sector, but you have many other features in the dataset?

You can read the data row-by-row and create the groups. But as the size of the data increases, creating the groups with human intervention will become impossible. You can utilise the clustering algorithm to do this task. Clustering is a technique to divide the data into similar groups. You simply need to pass meaningful data for all the companies, and the clustering algorithm will create groups for you.

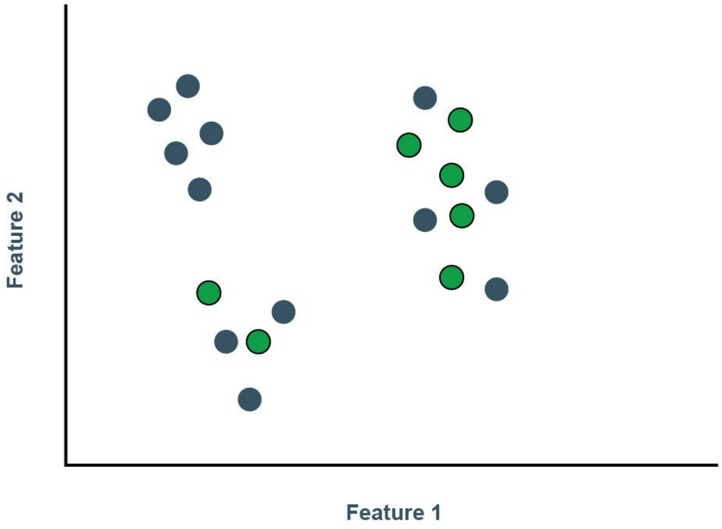
In the given dataset, the feature is the sector. Hence, the clustering algorithm creates the groups based on the sectors. You can also pass the historical data, fundamental data, etc. for a more detailed grouping. You don’t have to specify any rules to create the groups, the groups are created based on similarity. The similarity between the points within a group is not known, you have to analyse each group and add a meaningful label to them.

For example, we analysed the cluster above and found that the clusters were made based on the sectors. Also note, the only feature passed to the clustering algorithm was the sector. That means that the output generated by the algorithm is based on the input provided to the algorithm.

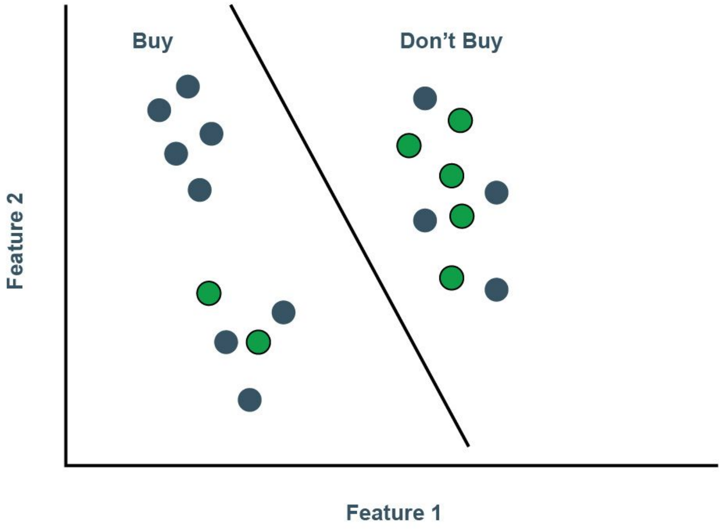
One thing must have come to your mind here. In the previous chapters, we had looked at supervised learning which included classification. Is it similar to clustering? In a way, yes, but there are certain differences here.

Let us consider a scenario to understand these differences in detail. You have decided to use a machine learning algorithm that will predict whether to buy Apple stock or not. One option is to use a classification algorithm. Let us look at the steps involved in doing so.

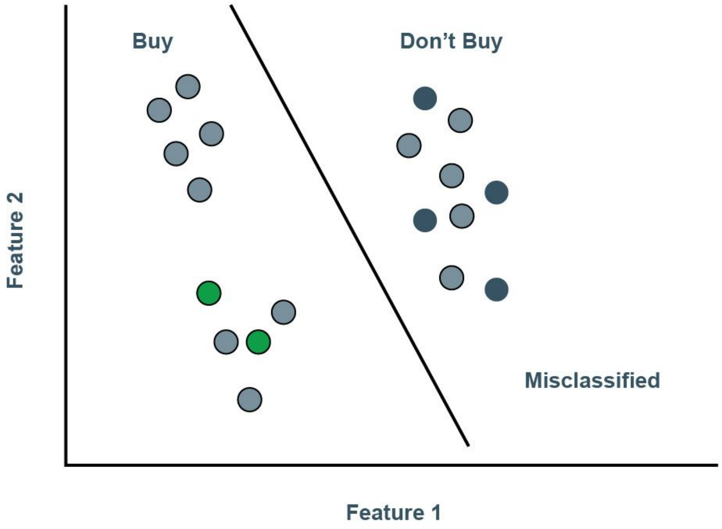
You will get the historical data. In order to train a classification model, you have to label the historical data. For days, when the next day returns are positive, you label it as buy. And days where the next day returns are negative, you label it as no position. You also use two features as input to the machine learning model. The image shows a scatter plot of these two features. The blue dots correspond to a buy signal and green to no position.



The classification model will create a boundary to separate blue dots from green dots. It will then use this boundary to predict the Buy or Don'tbuy position on the unseen data or future data points. If a data point falls in the left region, it will be classified as a Buy signal. And if it falls in the right region, it will be classified as Don'tBuy, which means no position is taken.

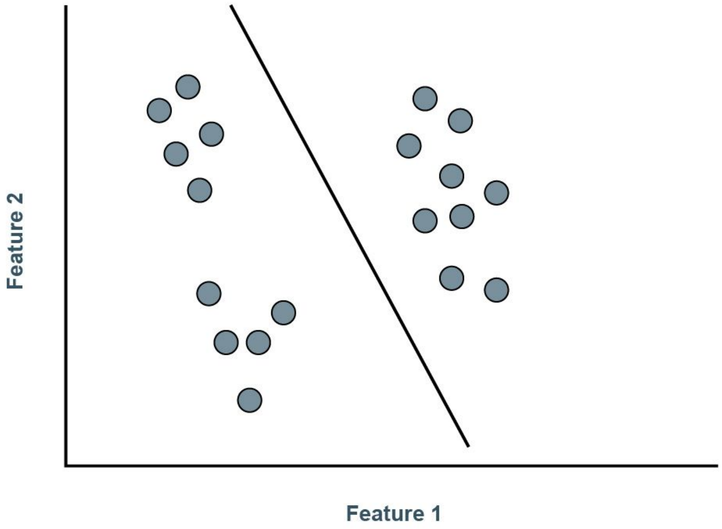


Here, you can see that few points are falling in the incorrect region. This means that these points are misclassified.

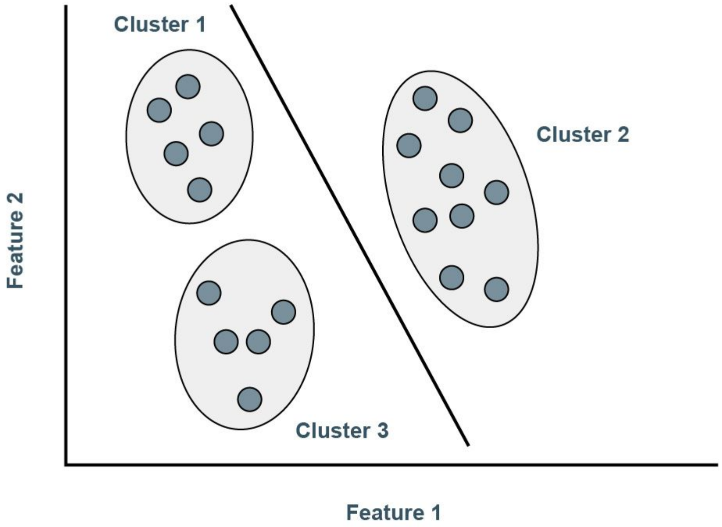


Also, it is very difficult to create the boundary if the classes are not well separated. Let us now see how we can approach the same problem with a clustering algorithm.

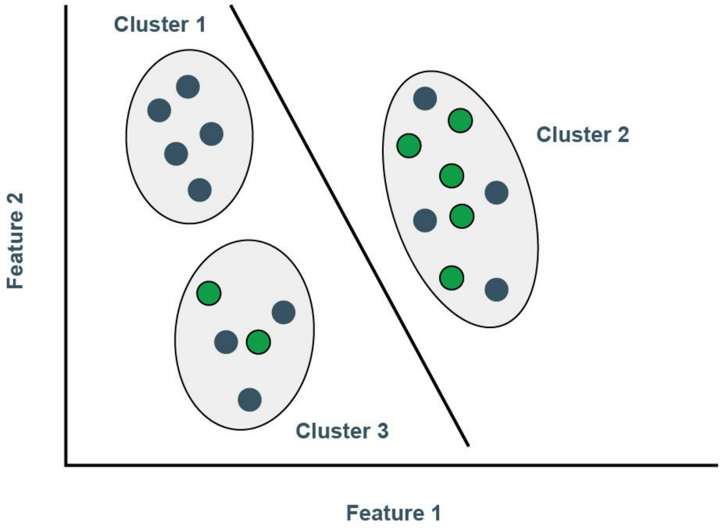
In clustering, you will simply pass the features to the algorithm without any expected outcome such as buy or no position.



That is, the clustering algorithm has access only to the features and does not know the groups in which it has to classify the data. Since there are no separate labels for buy, the algorithm will not create any boundary to separate buy and don’t buy. But it will create groups of similar data points.



You can analyse the groups and identify in which group you will buy the asset. Based on your analysis, you can buy Apple stock if the data points fall in cluster 1.



As you can see, all the data points falling in cluster 1 are blue in colour indicating that the next day returns are positive.

A limitation of clustering is that you don’t have any control over the clusters created by the algorithm. Some of the clusters created might not be useful to us, whereas some might be. For example, cluster 2 and cluster 3 have a combination of buy and no position. Thus, making it difficult to add a meaningful label to it.

That was all about the key differences between classification and clustering. Let us summarise them.
Classification Clustering

Nature of class/label Known Not known
Nature of classification Based on labels Based on data points similarity Type Classify market regimes Grouping similar stocks Control over grouping Yes No

When do we use unsupervised algorithms? Unsupervised learning is utilised under the following conditions:

- You do not have the output/target data.
- You don’t exactly know what you are looking for and want the machine to discover patterns/insights in the data.
- You want to keep only the essential information from a large dataset.

There are different clustering algorithms such as k-means clustering, Hierarchical clustering, DBSCAN, OPTICS, etc., which group the data according to their own definitions of similarity between the data points.

Let’s look at a clustering algorithm, called the K-means clustering algorithm in the next chapter.

17 K-Means Clustering

K-means clustering is used when we have unlabeled data, i.e. data without defined categories or groups. This algorithm finds groups/clusters in the data, with the number of groups represented by the variable ‘k’(hence the name).

The great thing about k-means is that we simply give the data to the algorithm, and the number of clusters that need to be formed. And then tell it to go play! That’s all the algorithm gets.

The algorithm works iteratively to assign each observation to one of the k groups based on the similarity in provided features.
The inputs to the k-means algorithm are the data/features(X) and the value of ‘k’(number of clusters to be formed).
The steps can be summarised as:

1. The algorithm starts with randomly choosing ‘K’ data points as ‘centroids’, where each centroid defines a cluster.
2. Each data point is assigned to a cluster defined by a centroid such that the distance between that data point and the cluster’s centroid is minimum.
3. The centroids are recalculated by taking the mean of all data points that were assigned to that cluster in the previous step. The algorithm iterates between steps (ii) and (iii) until a stopping criterion is met, such as a pre-defined maximum number of iterations are reached, or datapoints stop changing the clusters.

Let us see this in action now. Often, traders and investors want to group stocks based on similarities in certain features.

For example, a trader who wishes to trade a pair-trading strategy, where she simultaneously takes a long and shorts position in two similar stocks, would ideally want to scan through all the stocks and find those which are similar to each other in terms of industry, sector, market-capitalisation, volatility, or any other features.

Now consider a scenario where a trader wants to cluster stocks of 12 American companies based on two features:
1. Return on equity (ROE) = Net Income/Total shareholder’s equity, and 2. The beta of the stock

Investors and traders use ROE to measure the profitability of a company in relation to the stockholders’ equity. A high ROE is, of course, preferred to invest in a company. Beta, on the other hand, represents stock’s volatility in relation to the overall market(represented by the index such as S&P 500 or DJIA).

Going through each and every stock manually and then forming groups is a tedious and time-consuming process. Instead, one could use a clustering algorithm such as the k-means clustering algorithm to group/cluster stocks based on a given set of features.

Let’s implement a k-means algorithm for clustering these stocks in Python.

17.1 K-Means in Trading

We start with importing the necessary libraries and fetching the required data using the following commands.

Import Dataset We will use the dataset which contains the data for 12 companies and is stored in the sample_stocks.csv file. You can download this file from the github link provided in “About The Book” section of this book.

```
[]: #Importingthe necessary libraries import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-darkgrid') import warnings
warnings.filterwarnings('ignore')

#The dataisstored inthedirectory 'data_modules' path = "../data_modules/"

#Read thedata
DF = pd.read_csv(path + 'sample_stocks.csv',index_col=0) DF
```

```
[]: ROE(%) Beta
ADBE 28.84 0.96
AEP 10.27 0.26
```

```
CSCO 22.53 0.89
EXC 8.57 0.43
FB 22.18 1.29
GOOGL 15.19 1.00
INTC 23.77 0.59
LNT 10.90 0.33
MSFT 34.74 0.78
STLD 21.34 1.45
TMUS 12.13 0.56
XEL 10.20 0.30
```

As seen above, we have downloaded the data for the 12 stocks successfully.

We will now create a copy (df) of the original data and work with it. The first step is to pre-process the data so that it can be fed to a k-means clustering algorithm. This involves converting the data in a NumPy array format and scaling it.

Scaling amounts to subtracting the column mean and dividing by the column standard deviation from each data point in that column.
For scaling, we use the StandardScaler class of scikit-learn library as follows:

```
[]: #Making acopyto workwith df = DF.copy()
#Scaling the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_values = scaler.fit_transform(df.values)
#Printing pre-processeddata print(df_values)
```

```
[[ 1.29583307 0.59517359]
[-1.00653849 -1.27029587]
[0.51349787 0.40862664]
[-1.21731025 -0.81725329]
[0.47010369 1.4746092]
[-0.39654021 0.70177185]
[0.66723728-0.39086027]
[-0.92842895 -1.08374893]
[2.02733507 0.11548144]
[0.36595764 1.90100222]
[-0.77592938 -0.47080896]
[-1.01521733 -1.16369762]]
```

The next step is to import the ‘KMeans’ class from scikit-learn and fit a model with the value of hyperparameter ‘K’ (which is called n_clusters in scikit-learn) set to 2(randomly chosen) to which we fit our pre-processed data ‘df_values’.

```
[]: from sklearn.cluster import KMeans
km_model = KMeans(n_clusters=2).fit(df_values)
```

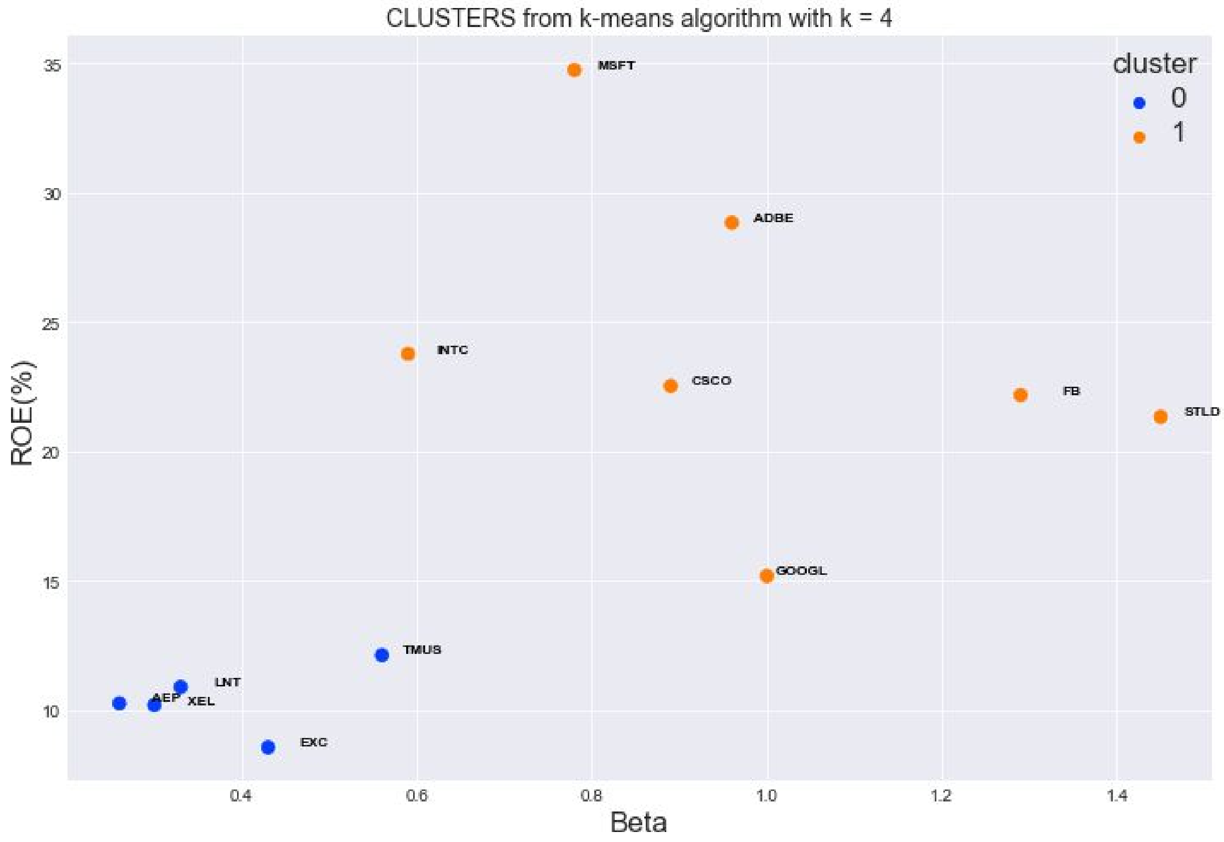
Thats it! ‘km_model’ is now trained and we can extract the cluster it has assigned to each stock as follows:

```
[]: clusters = km_model.labels_
df['cluster']=clusters
df
```

```
[]: ROE(%) Beta cluster ADBE 28.84 0.96 1
AEP 10.27 0.26 0
CSCO 22.53 0.89 1
EXC 8.57 0.43 0
FB 22.18 1.29 1
GOOGL 15.19 1.00 1
INTC 23.77 0.59 1
LNT 10.90 0.33 0
MSFT 34.74 0.78 1
STLD 21.34 1.45 1
TMUS 12.13 0.56 0
XEL 10.20 0.30 0
```

Now that we have the assigned clusters, we will visualise them using the matplotlib and seaborn libraries as follows:

```
[]: #Set graphsize
plt.figure(figsize=(12, 8))

#Set xandy axislabels
ax = sns.scatterplot(y="ROE(%)",x="Beta",edgecolor='face',
hue="cluster",data=df, palette='bright',
s=60)

#Plot thegraph
plt.xlabel('Beta',size=17)
plt.ylabel('ROE(%)',size=17)
plt.setp(ax.get_legend().get_texts(), fontsize='17')
plt.setp(ax.get_legend().get_title(), fontsize='17') plt.title('CLUSTERS from
k-means algorithm with k = 4',

fontsize='x-large')
#Label individual elements
for i in range(0,df.shape[0]):
plt.text(df.Beta[i]+0.07,df['ROE(%)'][i]+0.01,df.index[i],

horizontalalignment ='right',
verticalalignment='bottom',size='small', color='black',weight='semibold')
```



We can clearly see the difference between the two clusters in the graph above.

Can you make an analysis on this cluster? Cluster 1 largely consists of all the public utility companies which have a low ROE and low beta compared to high growth tech companies in Cluster 0.

Although we did not tell the k-means algorithm about the industry sectors to which the stocks belonged, it was able to discover that structure in the data itself. Therein lies the power and appeal of unsupervised learning.

The next question that arises is how to decide the value of hyperparameter k before fitting the model?

We passed the value of hyperparameter k=2 on a random basis while fitting the model. What would happen if we had increased the number of clusters?

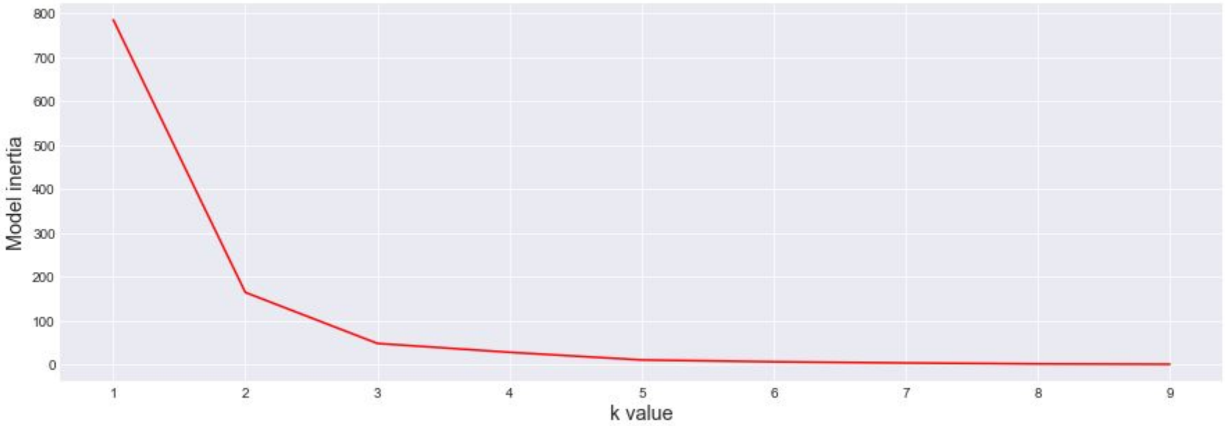
Let's say you took k = 8. Now, k-means algorithm will cumpulsorily try to create 8 clusters. But you know that your data set contains 12 companies, which means some clusters will have only one company.

How to find an optimum number of clusters?

One of the ways of doing this is to check the model's 'inertia', which represents the distance of points in a cluster from its centroid. As more and more clusters are added, the inertia keeps on decreasing, creating what is called an 'elbow curve'. We select the value of k beyond which we do not see much benefit (i.e., decrement) in the value of inertia. Think about it, if there were 8 clusters, then in the cluster with one company, the inertia would be 0 and there would be no point in having that cluster. Below we plot the inertia values for k-mean models with different values of 'k':

```
[]: #Calculatinginertia for k-meansmodelswith different values #of 'k'
inertia = []
k_range = range(1,10)
for k in k_range:

model = KMeans(n_clusters=k)
model.fit(df)
inertia.append(model.inertia_)
```

```
#Plotting the 'elbow curve'
plt.figure(figsize=(15,5))
plt.xlabel('kvalue',fontsize='x-large')
plt.ylabel('Model inertia',fontsize='x-large')
plt.plot(k_range,inertia,color='r')
plt.show()
```



As we can see that the inertia value shows marginal decrement after k= 3, a k-means model with k=3(three clusters) is the most suitable for this task. You can try running the same algorithm but change the parameter to 3 and see how it changes.

In the next chapter, we will look at another clustering algorithm, i.e. the hierarchical clustering algorithm.

18 Hierarchical Clustering

We looked at the K-means clustering in the previous chapter, but did you realise one limitation of K-means?

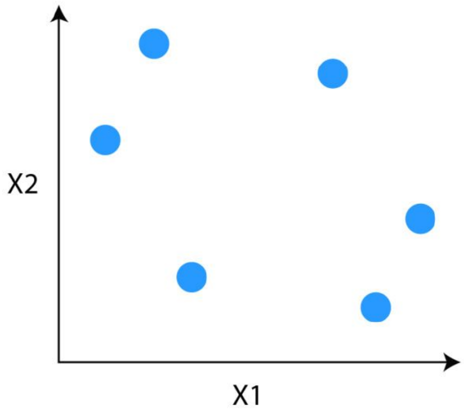
At the beginning of the algorithm, we need to decide the number of clusters. However, we wouldn’t know how many clusters we need in the start. It is true that we found out the optimum number of clusters using the concept of inertia, but what if we don’t need to define the number of clusters at the beginning?

Hierarchical clustering is one such algorithm which helps us in this regard. Let’s see how it works.
In essence, there are two types of hierarchical clustering:

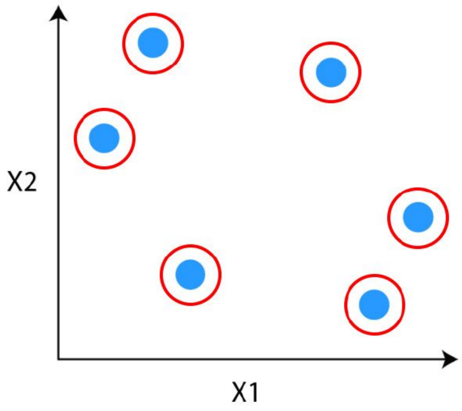
- Agglomerative Hierarchical Clustering
- Divisive Hierarchical Clustering

Agglomerative Hierarchical Clustering The agglomerative hierarchical clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It’s a bottom-up approach where each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

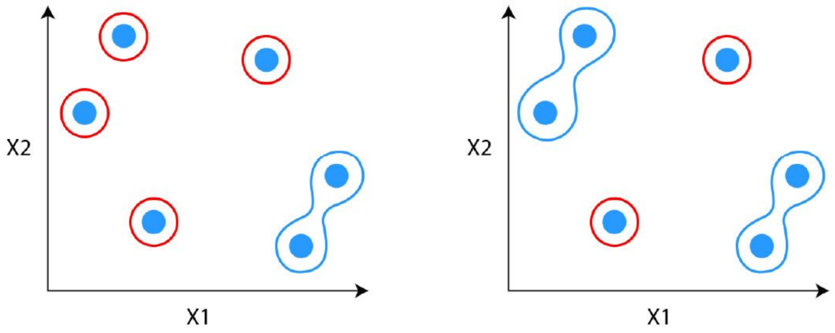
How does Agglomerative Hierarchical Clustering work? Suppose you have data points which you want to group in similar clusters.



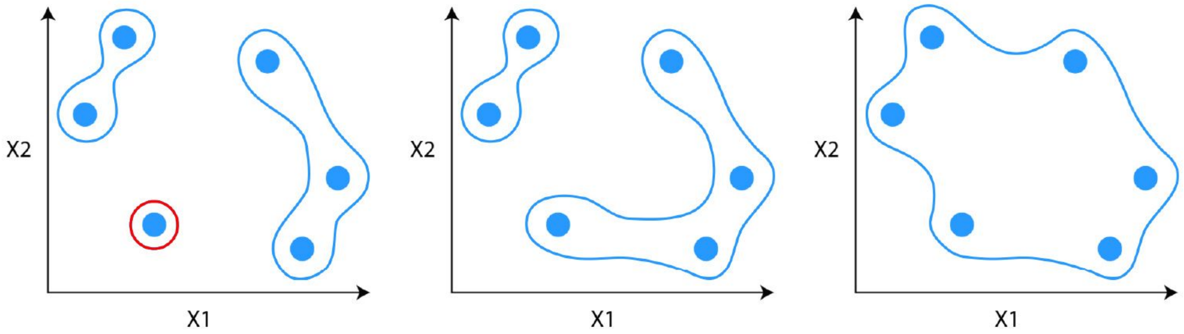
Step 1: The first step is to consider each data point to be a cluster.



Step 2: Identify two clusters that are similar and make them into one cluster.



Step 3: Repeat the process until only single cluster remains.



How to identify if two clusters are similar? One of the ways to do so is to find the distance between clusters.

Measure of Distance (Similarity) The distance between two clusters can be computed based on the length of the straight line drawn from one cluster to another. This is commonly known as the Euclidean distance.
The Euclidean distance between two points in either the plane or 3-dimensional space measures the length of a segment connecting the two points. It is the most obvious way of representing distance between two points.

If the (x_1,y_1) and (x_2,y_2) are points in 2-dimensional space, then the Euclidean distance between is:
$(x_2\ x_1)^2\ (y_2\ y_1)^2$

Other than Euclidean distance, several other metrics have been developed to measure distance such as Hamming distance, Manhattan distance (Taxicab or City Block), Minkowski distance.

The choice of distance metrics should be based on the field of study, or the problem that you are trying to solve. For example, if you are trying to measure distance between objects on a uniform grid, like a chessboard or city blocks. Then Manhattan distance would be an apt choice.

Linkage Criterion After selecting a distance metric, it is necessary to determine from where distance is computed. Some of the common linkage methods are: Single-Linkage: Single linkage or nearest linkage is the shortest distance between a pair of observations in two clusters.
Complete-linkage: Complete linkage or farthest linkage is the farthest distance between a pair of observations in two clusters.
Average-linkage: Average linkage is the average of the distance between each observation in one cluster to every observation in the other cluster.

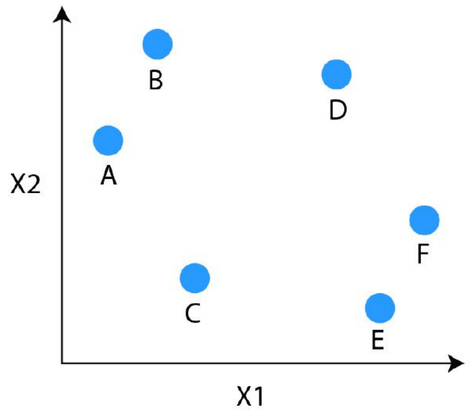
Centroid-linkage: Centroid linkage is the distance between the centroids of two clusters. In this, you need to find the centroid of two clusters and then calculate the distance between them before merging.

Ward’s-linkage: Ward’s method or minimum variance method, or Ward’s minimum variance clustering method, calculates the distance between two clusters when there’s increase in the error sum of squares after merging two clusters into a single cluster. This method seeks to choose the successive clustering steps so as to minimise the increase in sum of squares error at each step.

The choice of linkage criterion is based on the domain application. Average-linkage and complete-linkage are the two most popular distance metrics in hierarchical clustering. However, when there are no clear theoretical justifications for the choice of linkage criteria, Ward’s method is the default option.

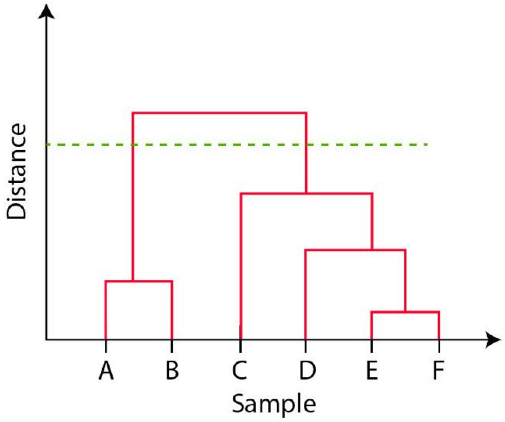
How to choose the number of clusters? To choose the number of clusters in hierarchical clustering, we make use of concept called dendrogram.

What is a Dendrogram? Dendrogram is a tree like diagram that shows the hierarchical relationship between the observations. It contains the memory of hierarchical clustering algorithms.
Just by looking at the dendrogram, you can tell how the cluster is formed.
Let see how to form the dendrogram for the below data points.



The observations E and F are closest to each other by any other points. So, they are combined into one cluster, and also the height of the link that joins them together is the smallest. The next observations that are closest to each other are A and B which are combined together.

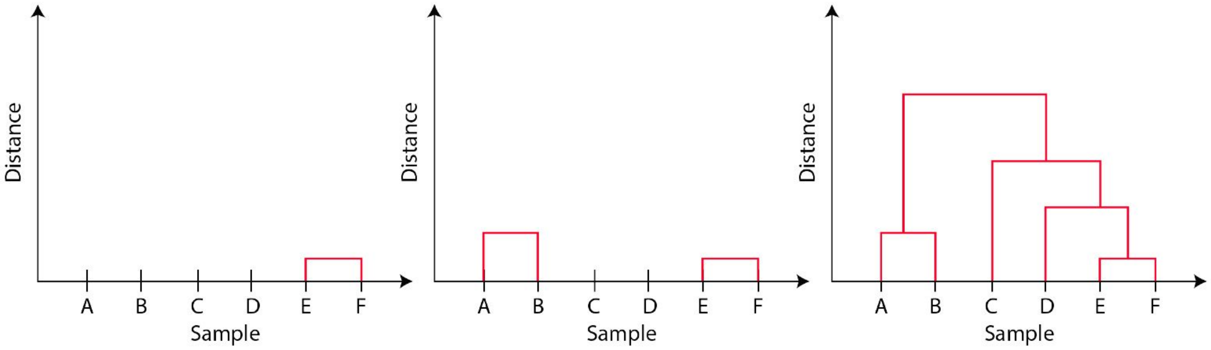
This can also be observed in the dendrogram as the height of the block between A and B is slightly bigger than E and F. Similarly, D can be merged into E and F clusters and then C can be combined to that. Finally A and B combined to C, D, E, and F to form a single cluster.



Important points to note while reading dendrogram is that:
1. Height of the blocks represents the distance between clusters.
2. Distance between observations represents dissimilarities.
But the question still remains the same. How do we find the number of clusters using a dendrogram or where should we stop merging the clusters?

If you think about it, you can see that while A and B can be in one cluster, they are far from the other cluster made of C, D, E and F. This is seen in the dendrogram as the height of the line joining these two clusters is the longest. Thus, you could divide these points into two clusters.

Observations are allocated to clusters by drawing a horizontal line through the dendrogram.



Divisive Hierarchical Clustering Divisive hierarchical clustering is not used much in solving real-world problems. It works in the opposite way of agglomerative clustering. In this, we start with all the data points as a single cluster.

At each iteration, we separate the farthest points or clusters which are not similar until each data point is considered as an individual cluster. Here we are dividing the single clusters into n clusters, therefore the name divisive clustering.

18.1 Hierarchical Clustering in Trading

We will start our strategy by first importing the libraries and the dataset.

```
[]: #Data manipulation
import numpy as np
import pandas as pd

#Technicalindicators
import talib as ta

#Plotting graphs
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-darkgrid')

#Machine learning
#Scaling the data
from sklearn.preprocessing import StandardScaler

from sklearn import metrics
import scipy.cluster.hierarchy as sc
from sklearn.cluster import AgglomerativeClustering
#The dataisstored inthedirectory 'data_modules' path = "../data_modules/"

#Read thedata
DF = pd.read_csv(path + 'sample_stocks.csv',index_col=0) DF
```

```
[]: ROE(%) Beta
ADBE 28.84 0.96
AEP 10.27 0.26
CSCO 22.53 0.89
EXC 8.57 0.43
FB 22.18 1.29
GOOGL 15.19 1.00
INTC 23.77 0.59
LNT 10.90 0.33
MSFT 34.74 0.78
STLD 21.34 1.45
TMUS 12.13 0.56
XEL 10.20 0.30
```

We will scale the data as well.

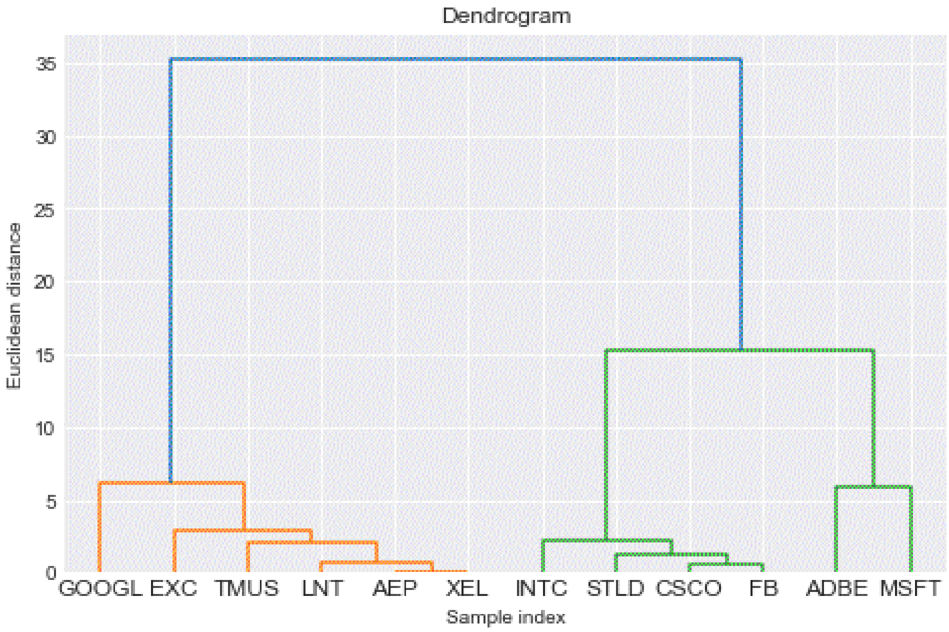
```
[]: #Making acopyto workwith df = DF.copy()
scaler = StandardScaler()
df_values = scaler.fit_transform(df.values)
#Printing pre-processeddata print(df_values)
```

```
[[ 1.29583307 0.59517359] [-1.00653849 -1.27029587] [0.51349787
0.40862664] [-1.21731025 -0.81725329] [0.47010369 1.4746092]
[-0.39654021 0.70177185] [0.66723728-0.39086027] [-0.92842895
-1.08374893] [2.02733507 0.11548144] [0.36595764 1.90100222]
[-0.77592938 -0.47080896] [-1.01521733 -1.16369762]]
```

Create a Dendrogram We start by importing the library that will help to create dendrograms. Dendrogram helps to give a rough idea of the number of clusters.

```
[]: #Plot adendrogram
plt.figure(figsize=(8, 5)) plt.title("Dendrograms")

#Create adendrogram
sc.dendrogram(sc.linkage(df, method='ward'),labels=df.index)

plt .title('Dendrogram')
plt.xlabel('Sample index') plt.ylabel('Euclidean distance') plt.show()
```

```
[]: Text(0, 0.5, 'Euclidean distance')
```



By looking at the above dendrogram, we divide the data into two clusters.

Fit the Model We instantiate the AgglomerativeClustering. We will then pass the Euclidean distance as the measure of the distance between points and Ward linkage to calculate clusters’ proximity.Then we fit the model on our data points. Finally, we return an array of integers where the values correspond to the distinct categories using lables_ property.

```
[]: #Instantiatethe clustering algorithm
cluster = AgglomerativeClustering(
n_clusters=2,affinity='euclidean',linkage='ward')

#Using thefitfunction cluster.fit(df_values)
DF['labels']=cluster.labels_ DF
```

```
[]: ROE(%) Beta labels ADBE 28.84 0.96 0
AEP 10.27 0.26 1
CSCO 22.53 0.89 0
EXC 8.57 0.43 1
FB 22.18 1.29 0
GOOGL 15.19 1.00 0
INTC 23.77 0.59 0
LNT 10.90 0.33 1
```

MSFT	34.74	0.78	0
STLD	21.34	1.45	0
TMUS	12.13	0.56	1
XEL	10.20	0.30	1

Pros and Cons of Hierarchical Clustering
1. Like K-means clustering, we don’t need to specify the number of clusters required for the algorithm.

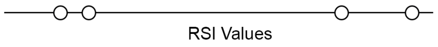
2. It doesn’t work well on a large dataset. It is generally applicable to a smaller data. If you have a large dataset, it can become difficult to determine the correct number of clusters by the dendrogram.

3. In comparison to K-means, hierarchical clustering is computationally heavy and takes longer time to run.

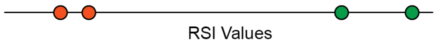
Conclusion Despite the limitations of hierarchical clustering when it comes to large datasets, it is still a great tool to deal with small to medium dataset and find patterns in them. In the next chapter, we will look at the concept of dimensionality reduction.

19 Principal Component Analysis

You know that a clustering algorithm calculates the distance between the points. And forms clusters of points which are closer to each other. For instance, let’s take the RSI values of J. P. Morgan.



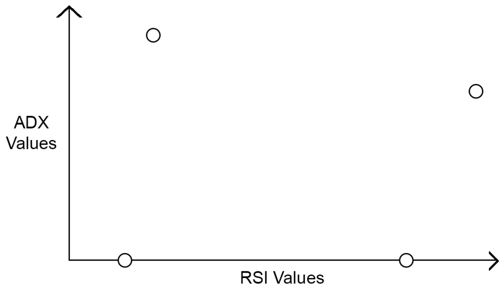
You can easily see that you can form two clusters here.



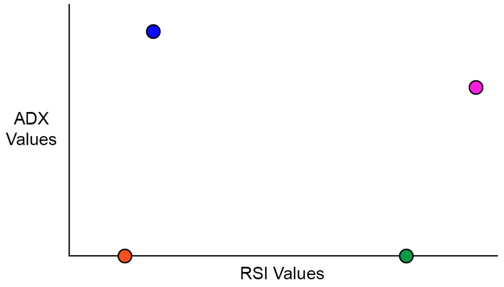
What happens if you add another feature, say ADX?
Date RSI ADX

9 August 2021 40 25
10 August 2021 42 62
11 August 2021 60 25
12 August 2021 65 50

A new axis is added to capture the ADX values. The graph becomes a 2D graph with RSI as the x-axis and ADX as the y-axis.



You can see that due to the addition of ADX, the data points moved away. These data points are difficult to cluster in two groups. Instead, they are clustered into four different groups.



Why did the number of clusters increase? At first, we had calculated only one feature value, which was the RSI. Then, we added more information in the form of the ADX values. Thus, the points which were similar and clustered together became distant due to the addition of new features. Similarly, when we add more features, the information increases and the points may move farther away.

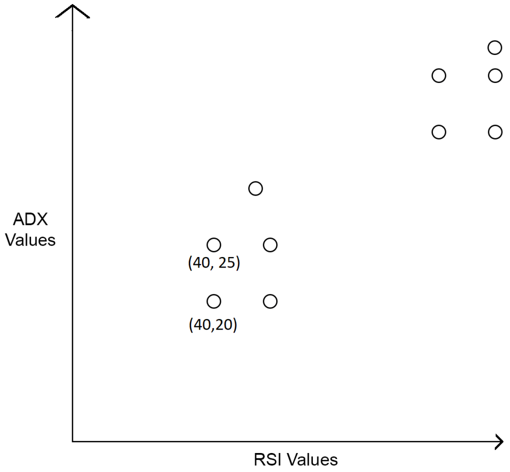
But what is the problem in adding more information and creating more clusters?

In the RSI and ADX example, we ended up creating 4 clusters. Each cluster had only 1 data point. In the essence, we ended up with what we started. We started with 4 points and ended with 4 clusters. There was no grouping of these 4 points. To generalise, too many dimensions or features causes each point to appear equidistant from other points. If the distances are all approximately equal, then all the observations appear equally alike (as well as equally different), and no meaningful clusters can be formed.

This is called the curse of dimensionality.
How do you overcome the curse of dimensionality?
One way is to eliminate the dimensions, or is it the features?
But that leads to loss of information! Let’s take another example here. The RSI and ADX values for an asset is given below.

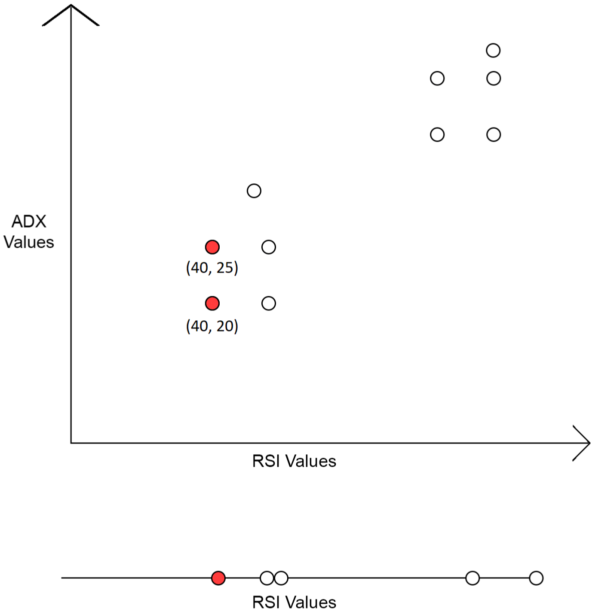
Date	RSI	ADX
9 August 2021	40	20
10 August 2021	45	25
11 August 2021	60	45
12 August 2021	65	50
13 August 2021	40	25
14 August 2021	45	20
15 August 2021	60	50
16 August 2021	65	45
17 August 2021	44	30
18 August 2021	65	55

If you draw a scatter plot for these values, it looks as shown below:

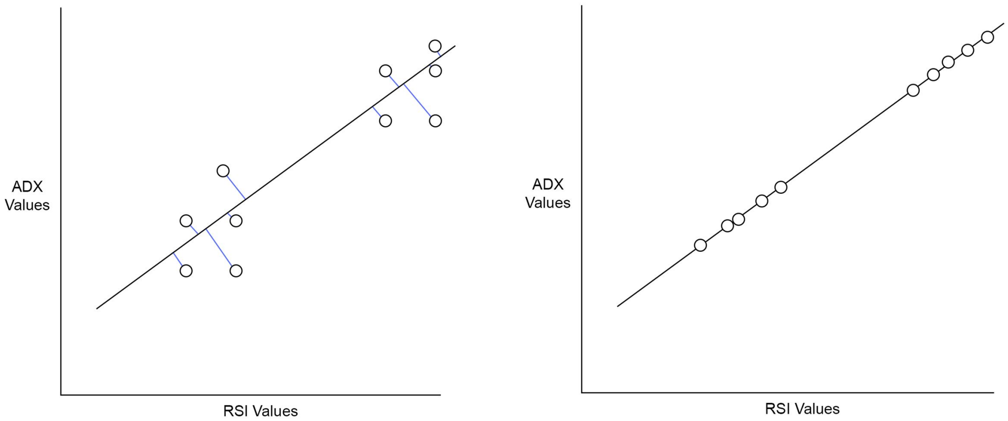


The easiest way to reduce the dimensions is to remove one of them. For example, either remove ADX or remove RSI values. But that would lead to loss of information.

If only RSI was kept, the data point (40, 20) and (40, 25) will come together. Earlier, the two points were at a distance of 5 units from each other. But now it seems like they are the same point. Thus, the information that these two points are different and not the same is lost.



Is there a better way to reduce the dimension while keeping the loss in information at a minimum? Yes. Draw a line that is in between the two axes. And then bring all the points on this line.



You can see that even though the points are closer on the new axis, they can still be seen distinctly. You simply rotate or straighten this line to convert the data points into a single dimension. And you managed to preserve some information of both dimensions. This is essentially what the principal component analysis does, i.e. dimensionality reduction.

Principal Component Analysis creates a new variable which contains most of the information in the original variables. An example would be that if we are given 5 years of closing price data for 10 companies, i.e. approximately (1265 * 10) data points. We would seek to reduce this number in such a way that the information is preserved.

Of course, in the real world, there would be some information loss, and thus we use the principal component analysis to make sure that it is kept to a minimum. That sounds really interesting, but how do we do this exactly?

The answer is EigenValues. Well, that’s not the whole answer but in a way it is. Before we get into the mathematics of it, let us refresh our concepts about the matrix (no, not the movie!).

19.1 Eigen Vectors and Covariance Matrix

One of the major applications of Eigenvectors and Eigenvalues is in the image transformations. Remember how we can tilt an image in any of the photo editing apps. Well, Eigenvalues and Eigenvectors helps us in this regard.

Let’s assume we have two features, shown in two dimensions, x and y. Basically, we are creating new axes, u and v, and then project the points on one of the axis, where we get maximum spread. This is done by multiplying a vector, with the matrix of features. So formally, this is what happens, for transforming matrix A and a vector v:

$Av = lv$

The entire set of such vectors which do not change direction when multiplied with a matrix are called Eigen Vectors. The corresponding change in magnitude is called the Eigen Value. It is represented with *l* here. Remember how we said that the ultimate aim of the principal component analysis is to reduce the number of variables. Well, Eigenvectors and Eigenvalues help in this regard. One thing to follow is that Eigenvectors are for a square matrix i.e. a matrix with an equal number of rows and columns. But they can also be calculated for rectangular matrices.

Let’s say we have a matrix A, which consists of the following elements:
Matrix A
28 6 10
Now, we have to find the Eigenvectors and Eigenvalues in such a manner such that:
$Av = lv$
Where,

- v is a matrix consisting of Eigenvectors.
- *l* is the Eigenvalue.

While it may sound daunting, but there is a simple method to find the Eigenvectors and Eigenvalues.

Let us modify the equation to the following:
$Av = lIv$ (We know that AI = A)
If we bring the values from the R.H.S to the L.H.S, then,

$Av\ lIv = 0$
Now, we will assume that the Eigenvector matrix is non-zero. Thus, we can find *l* by finding the determinant.

Thus, $_{|}A\ lI_{|} = 0$
$(2\ l)(10\ l)\ (8_{\leftarrow}\ 6)= 0$
$20\ 2l\ 10l\ +l^2\ 48 = 0$

$28\ 12l\ +l^2 = 0$
$l^2\ 12l\ 28 = 0$
$(l\ 14)(l + 2)= 0$
$l = 14, 2$
Hence, we can say that the Eigenvalues are 14, -2.
Taking the *l* value as -2. We will now find the Eigenvector matrix.
Recall that, $Av = lv$.

$_{A\ =}$ 26 8 10
$v =^{x}$

y

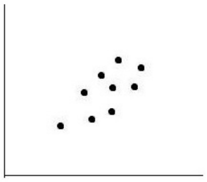
Therefore,

(2x + 6y) = -2x
(8x + 10y) = -2y
(4x + 6y) = 0
(8x + 12y) = 0
Since they are similar, solving them to get:
2x + 3y = 0, x = (-3/2)y
x = -3, y = 2
Thus, the Eigenvector matrix is:

3
2

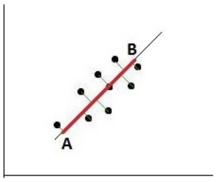
Similarly, for *l* = 14, the Eigenvectors are x =1, y = 2.
That’s awesome! We have understood how Eigenvectors and Eigenvalues are formed. Now that we know how to find the Eigenvalues and Eigenvectors, let us talk about their properties.

In our earlier example, we had a 2 x 2 matrix. This can be considered as 2 dimensions. One of the great things about Eigenvectors and Eigenvalues is that they are used for “Orthogonal Transformations” which is a fancy word for saying that the variables are shifted in another axis without changing the direction. Let us take a visual example of this.



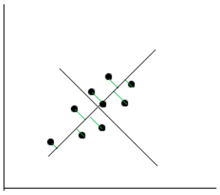
Let’s say we have a graph with certain points mapped on the x and y axis. Now you can locate them by giving the (x, y) coordinates of a point, but we realise that there can be another efficient way to display them meaningfully.

What if we draw a line in such a way that we can locate them using just the distance between a point on this new line and the points. Let’s try that now.

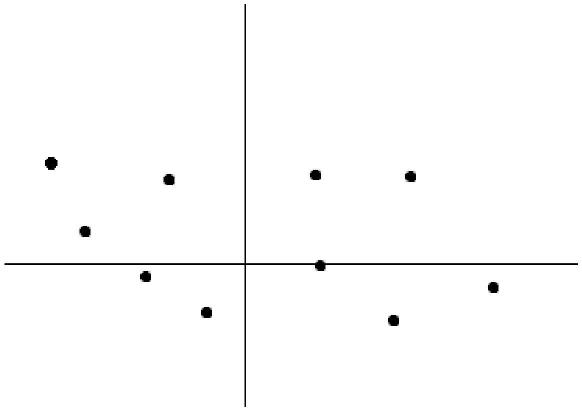


Let us say that the green lines represent the distance from each point to the new line. Furthermore, the distance from the first point on the line projected, i.e. A, to the last point's projection on the new line, i.e. B, is referred to as the spread here. In the above figure, it is denoted as the red line. The importance of spread is that it gives us information on how varied is the data set. The larger the value, the easier it is for us to differentiate between two individual data points.

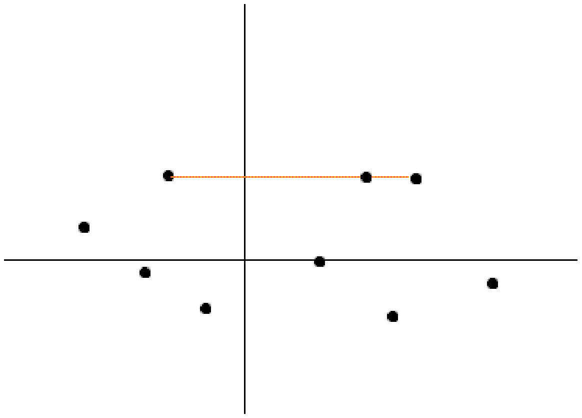
We should note that if we had variables in two dimensions, then we will get two sets of Eigenvalue and Eigenvectors. The other line would actually be perpendicular to the first line.



Now, we have a new coordinate axis which is much more suited to the points, right? Since we are used to seeing the axes in a particular manner, we will rotate them and keep it as shown in the below diagram.



If we had to explain Eigenvectors and Eigenvalues diagrammatically, we can say that the Eigenvectors gives us the direction in which it is stretched or transformed and the Eigenvalues is the value by which it is stretched. In that sense, the points projected on the horizontal line gives us more information when it comes to the spread than the points being projected on the vertical line.



But why is this important to understand?

When we said the Eigenvalues gives us the value by which the points are stretched, we should add further that the Eigenvalues with the corresponding larger number gives us more information than the one with the lesser valued Eigenvalue.

Oh, wait! This chapter talks about Principal Component Analysis. Well, the Eigenvalues and Eigenvectors are actually the components of the data.

But there is one missing part of the puzzle left. As we have seen the illustration, the data points were bunched together, and hence the distance between the points was not that significant.

But what if we were trying to compare stock prices, one whose value was less than $10 and another which was more than \$500? Obviously, the variation in the price itself would lead to a different result. Hence, we try to find a way to standardise the data. And since we are trying to find the difference, we can use the covariance matrix.

Before we go to covariance, let us see what is variance.

19.1.1 What is variance?

Variance tells us how much the values are far from the mean or average value. The formula for variance is given as:

$$\text{Variance}(s^2) = \frac{\sum_{i}^{n} (x_i - \bar{x})^2}{N}$$

For example, there are three points, 155, 146, 152.

The average of these three points is 151. Using the formula, the variance is:

Variance
(
s
$_2$)= (155 151)^2+(146 151)^2+(152 151)^2 $_3$ = 14

Why did we square the difference? The first reason is that if we had simply taken the difference, it would lead to the numerator becoming 0. Also, squaring the difference helps us give more weightage to the points which are farther from the mean. Thus, the point 146 is 5 points away from 151 and that leads to the value of 25. Whereas, 152 is merely one point away and thus, its square is only 1.

Another reason is that squaring helps us give same weightage to points which are above or below the mean. A data point that is 2 units away from the mean should have the same weightage irrespective of whether it is above or below the mean price. Squaring the difference helps us achieve that.

19.1.2 Covariance

Covariance is calculated using the below formula:
Â$_i$(x$_i$ Xmean)(y$_i$ Ymean)$_{\text{Cov(X,Y)= N}}$

Cov (X, Y) = Covariance between X and Y
Xi = Various observations of X
Xmean = Average value of X
Yi = Various observations of Y
Ymean = Average value of Y
N = Number of observations

In the resulting covariance matrix, the diagonal elements represent the variance of the stocks.
Also, the covariance matrix is symmetric along the diagonal, meaning:
$s_{21} = s_{12}$
Thus, in this manner, we can find the variance as well as the covariance of a dataset. Let us assume that we had two features, RSI and ADX. The covariance matrix would be represented as follows:
RSI ADX RSI 120 142 ADX 142 190

All right. So we know that Principal component analysis acts as a compression tool and seeks to reduce the dimensions of data in order to make computing easier. We also saw how Eigenvectors and Eigenvalues make the original matrix simpler. Further, we know that covariance simplifies the whole aspect of finding Eigenvalues and Eigenvectors to a great deal.
Now recall that initially, we talked about how Principal component analysis seeks to reduce the dimensions by converting two variables into one, by creating a new feature altogether. Do you see a picture forming in your head about how we can use it in trading?

Let’s say we find the Eigenvalues and Eigenvectors of the covariance matrix, we will say that the largest Eigenvalue has the most information about the covariance matrix. Thus, we can keep majority of the Eigenvalues and their corresponding Eigenvectors and still be able to contain most of the information which was present in the covariance matrix.

If we had selected 4 Eigenvalues and their corresponding Eigenvectors, we will say that we have chosen 4 Principal components.

That’s all there is to it. Now to take the earlier example, if we had chosen the Eigenvalue 14, we would lose some information but it simplifies the computation by a great deal and in that sense, we have reduced the dimension of the variables.

Whew! We have actually understood how the Principal Component Analysis works. If we have to add another note here and try to reinforce the use of Eigenvectors in PCA, we can see that the sum of the Eigenvalues of the covariance matrix is approximately equal to the sum of the total variance in our original matrix.

But there is still one question in mind.

When to use Principal Component Analysis? Suppose you are dealing with a large dataset which is computationally heavy, then you can use Principal component Analysis. In fact, it could help us a great deal in the strategy for pairs trading. Let’s try that, shall we?

19.2 Principal Component Analysis in Trading

Luckily for us, we don’t have to code the whole logic of the Principal Component Analysis in Python. We will simply import the sklearn library and use the PCA function already defined.

Since we are looking for global implementation, we will use the stocks listed on the NYSE.

Since we will be working on the data for a lot of companies, we have created a csv file which contains tickers for 20 companies listed on the NYSE. For the time being, we will import their Adjusted Close Price. The code is as follows:

```
[]: #Data manipulation
import numpy as np
import pandas as pd

#Technicalindicators import talib as ta

#Plotting the graphs
import matplotlib.pyplot as plt import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-darkgrid')

#Importingthe machinelearninglibraries from sklearn.decomposition import
PCA from sklearn.cluster import KMeans, DBSCAN from sklearn.manifold
import TSNE
from sklearn import preprocessing
from statsmodels.tsa.stattools import coint

#Import thedataset
df = pd.read_csv("pca.csv",index_col=0) df.head()
```

```
[]: GOOG GOOGL AMZN MA ,!BA \
Date
2018-01-02 1065.000000 1073.209961 1189.010010 148.906738 282.
```

```
,!886383
2018-01-03 1082.479980 1091.520020 1204.199951 150.778992 283.
,!801239
2018-01-04 1086.400024 1095.760010 1209.589966 152.729645 282.
,!724396
2018-01-05 1102.229980 1110.290039 1229.140015 155.895767 294.
,!322296
2018-01-08 1106.939941 1114.209961 1246.869995 156.367020 295.
,!570740

CABT CRMCOST ACN\ Date
2018-01-02 66.456902 55.270180 104.410004 177.262619 145.510391
2018-01-03 66.662445 55.392387 105.290001 179.389923 146.181931
2018-01-04 67.484688 55.298374 106.680000 177.996857 147.912842
2018-01-05 67.395294 55.458187 108.099998 176.726089 149.132996
2018-01-08 66.608818 55.298374 108.860001 177.413223 150.324783

AVGO MMM CVS FIS SYK \ Date
2018-01-02 231.258133 209.198364 65.703430 90.138031 151.780838
2018-01-03 233.787170 209.189468 65.417442 90.667137 152.636368
2018-01-04 233.865082 211.923889 67.142265 91.340523 152.588287
2018-01-05 235.250839 213.575165 70.109261 91.859985 154.818405
2018-01-08 235.813828 212.882675 69.501572 92.090866 156.933121
MDLZ CI CME ISRG COP

Date
2018-01-02 39.286922 200.766891 131.867477 375.250000 49.867218
2018-01-03 39.444405 204.136826 133.980453 383.820007 50.786686
2018-01-04 39.537041 205.207260 135.410294 376.920013 51.372623
2018-01-05 40.046532 208.686172 136.293716 379.010010 51.273472
2018-01-08 39.842739 206.376816 138.188080 391.859985 51.796291
```

Since we will be working on daily returns, we write the code as follows: []:

```
data_daily_returns = df.pct_change()
data_daily_returns.dropna(inplace=True)
```

If we have to check the number of rows and columns of the array, we use the following code:

```
[]: data_daily_returns.shape
[]: (501, 20)
```

Here, we understand that there are 20 columns corresponding to the number of companies we have selected and 501 is the data points we have of each company. Moving ahead, we will now use the Principal Component Analysis code. Since we are trying to reduce the variables, let's keep the number of Principal components as 18.

```
[]: N_PRIN_COMPONENTS = 18
pca = PCA(n_components=N_PRIN_COMPONENTS)
pca.fit(data_daily_returns)

[]: PCA(n_components=18)
```

You can check the number of rows and columns in the array with the "shape" command.

```
[]: pre_process = pca.components_.T
pre_process.shape
[]: (20, 18)
```

You can see that we have gone from 501 to 20. Now, we have to use this for trading. First we will scale the data.

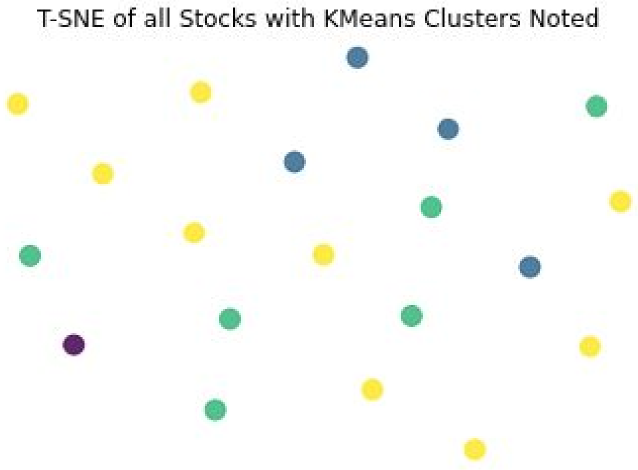
```
[]: #Using transpose onthedataset
X = pca.components_.T
X.shape

#Scaling the dataset
X = preprocessing.StandardScaler().fit_transform(X)
print(X.shape)

(20, 18)
[]: #Using k-means algorithm ondataset
clf = KMeans(n_clusters=4,init='k-means++',max_iter=30,
n_init=10,random_state=7)
print(clf)

#Using thefitfunction
clf.fit(X)
```

```
labels = clf.labels_
n_clusters_ = len(set(labels)) (1 if -1 in labels else 0) print("\nClusters
discovered: %d" % n_clusters_)

clustered = clf.labels_ KMeans(max_iter=30, n_clusters=4, random_state=7)
Clusters discovered: 4
```

To visualise it, we would use the t-SNE tool which is used to visualise high dimensional data into a 2D data.

The exact code is as follows:

```
[]: clustered_series = pd.Series(index=data_daily_returns.columns,
data=clustered.flatten())
clustered_series_all = pd.Series(
index=data_daily_returns.columns, data=clustered.flatten())
clustered_series = clustered_series[clustered_series != -1] X_tsne =
TSNE(learning_rate=1000,perplexity=25,
random_state=1337).fit_transform(X)

plt .figure(1,facecolor='white') plt.clf()
plt.axis('off')

plt .scatter(
X_tsne[(labels!=-1), 0], X_tsne[(labels!=-1), 1], s=100,
alpha=0.85,
c=labels[labels!=-1]

)

plt .scatter(
X_tsne[(clustered_series_all==-1).values, 0],
X_tsne[(clustered_series_all==-1).values, 1], s=100,
alpha=0.05

)
plt.title('T-SNE of all Stocks with KMeans Clusters Noted');
```



You can see that there are 4 clusters formed. While it may look like they are spread out, the t-SNE tool has visualised the clusters in a 2 dimensional space, and hence you can’t see the clusters grouped together.

Once these clusters are formed, you can use them further for analysis or your own trading strategy.

Great! We have not only seen two unsupervised algorithms, but we have also seen how to overcome the curse of dimensionality. In the next chapters, we lean back a bit and try to understand the concepts of natural language processing and reinforcement learning.

20 Natural Language Processong

Natural Language Processing or NLP has a wide variety of applications. Let us look at them now.

1. Machine Translation: As the amount of information available online is growing, the need to access it becomes increasingly important. Machine translation helps us conquer language barriers by translating technical manuals, support content, or catalogues at a significantly reduced cost.

2. Automatic Summarization: The second subset of NLP is Automatic Summarization. Information overload is a problem when we need to access a specific piece of information from a huge knowledge base. Automatic summarization is relevant not only for summarising the meaning of

documents and information but also for understanding the emotional meanings inside the information.

3. Sentiment Analysis: The goal of sentiment analysis is to identify sentiment among several posts or even in the same post where emotion is not explicitly expressed.

4. Text Classification: It makes it possible to assign predefined categories to a document and organise it to help you find the information you need or simplify some activities.

5. Question Answering: A question answering application is capable of coherently answering a human request. As speech-understanding technology and voiceinput applications improve, the need for NLP will increase. QA is becoming popular thanks to applications such as Siri, OK Google, Alexa, chat boxes, and virtual assistants. It may be used as a text-only interface or as a spoken dialogue system.

How can you use NLP in trading? NLP in trading is mainly used to gauge the sentiment of the market through Twitter feeds, newspaper articles, RSS feeds, and Press releases. In this chapter, we will cover the basic structure needed to solve the NLP problem from a trader’s perspective.

News and NLP Anyone who has traded some sort of a financial instrument knows that the markets constantly factor in all the news that is pouring in through various sources.

The cause and effect relationship between impactful news and market movements can be directly observed when one tries to trade the market during the release of big news such as the non-farm payrolls data.

Before social media became one of the main sources of information, traders used to depend on the Radio or TV announcements for the latest information.

But since Twitter became a source of market-moving news (thanks to political leaders), traders are finding it difficult to manually track all the information originating from different twitter handles. To circumvent this problem, traders can use NLP packages to read multiple news sources in a short amount of time and make a quick decision.

20.1 NLP in Trading

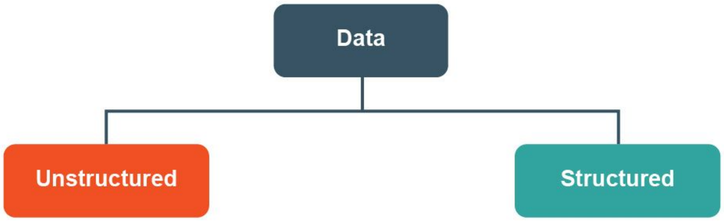
Following are the steps that one needs to follow for using NLP for Trading:
- Get the Data
- Pre-process the Data
- Convert the Text to a Sentiment Score
- Generate a Trading Model
- Backtest the Model

Get the Data To build an NLP model for trading, you need to have a reliable source of data. There are multiple vendors for this purpose.
How to get news headlines for a specific company?

In the news headline text, you can search for the company ticker or company name to get the news headline related to it. For example, to get the news headline specific to Apple Inc, you can search for AAPL or Apple. You can also consider to include news headlines which mention Apple products such as the iPhone or Mac. However, there are chances that you might miss out on news which indirectly references Apple. For example, “The biggest price drop seen in the best selling smartphone in the US”.

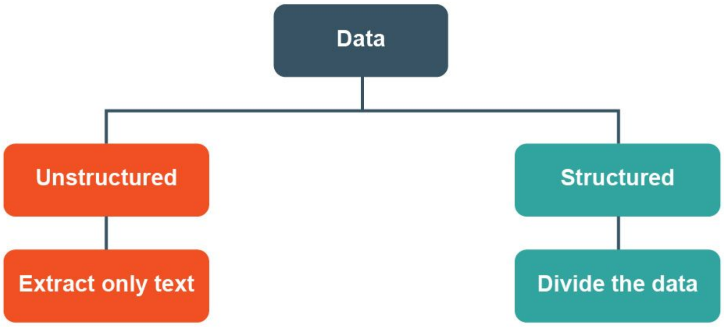
How to get the latest news headline data and sentiment score for them? You can get the latest news headline data from webhose.io. It provides a Python API to extract the news headline.

Let us divide the data into two types and try to approach each of them differently.



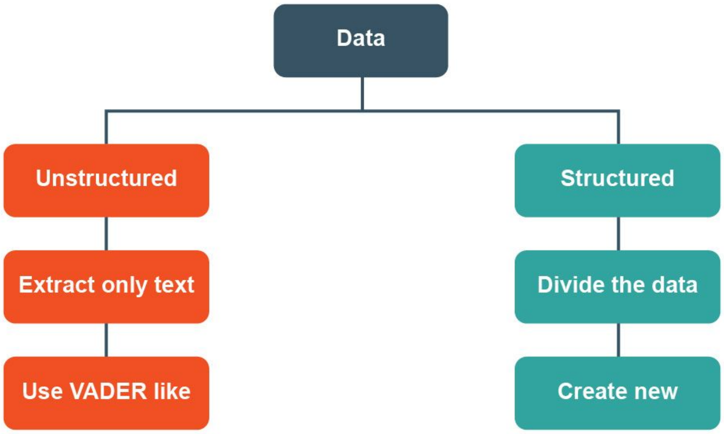
Structured data is one that is published in a predetermined or consistent format. The language is also very consistent.
For example, the press release of fed minutes or a company’s earnings can be considered as structured data. Here, the length of the text is usually very huge.

On the contrary, unstructured data is one where neither the language or format is consistent. For example, twitter feed, blogs and articles can be counted as a part of this. These texts are usually limited in size.



Pre-process the Data Unstructured data like Twitter feeds consists of many nontextual data, such as hashtags and mentions. These need to be removed before measuring the text’s sentiment.

For structured data, the size of the text can easily cloud its essence. To solve this, you need to break the text down to individual sentences or apply techniques such as term frequency-inverse document frequency (tf-idf) to estimate the importance of words.



Convert the Text to a Sentiment Score To convert the text data to a numerical score is a challenging task. For unstructured text, you can use pre-existing packages such as VADER to estimate the sentiment of the news. If the text is a blog or an article then you can try breaking it down for VADER to make sense of it.

For structured text, you don't have any pre-existing libraries that can help you convert the text to a positive or a negative score. So, you will have to create a library of your own.

When building such a library of relevant structured data, care should be taken to consider texts from similar sources and the corresponding market reactions to this text data.

For example, if the Fed releases a statement saying that "the inflation expectations are firmly anchored" and changes it to "the inflation expectations are stable", then libraries like VADER won't be able to tell the difference apart, but the market will react significantly.

To understand score the sentiment of such text you need to develop a word-to-vector model or a decision tree model using the tf-idf array.

Generate a Trading Model Once you have the sentiment scores of the text, you can combine this with some kind of technical indicators to filter the noise and generate the buy and sell signals.

What role does machine learning play in this step?

The essential part of natural language processing is to convert text or information to a objective number which can be used to create trading signals. Once you have the sentiment scores, you can use an ML model to generate trading signals.

Backtest the Model Once the model is ready, you need to backtest it on the past data to check whether your model's performance is within the risk limitations. While backtesting, make sure that you don't use the same data that is used to train the decision tree model.

If the model confirms to your risk management criterion then you can deploy the model in live trading.

Machine learning has varied applications in trading and can be used according to your needs. In the next chapter, let us look at something which the world calls the future of machine learning, ie reinforcement learning.

21 Reinforcement Learning

Initially, we were using machine learning and AI to simulate how humans think, only a thousand times faster! The human brain is complicated but is limited in capacity. This simulation was the early driving force of AI research. But we have reached a point today where humans are amazed at how AI "thinks".

A quote sums it up perfectly, "AlphaZero, a reinforcement learning algorithm developed by Google's DeepMind AI, taught us that we were playing chess wrong!".

While most chess players know that the ultimate objective of chess is to win, they still try to keep most of the chess pieces on the board. But AlphaZero understood that it didn't need all its chess pieces as long as it was able to take the opponent's king. Thus, its moves are perceived to be quite risky but ultimately they would pay off handsomely.

AlphaZero understood that to fulfil the long term objective of checkmate, it would have to suffer losses in the game. We call this delayed gratification. What’s impressive is that before AlphaZero, few people thought of playing in this manner. Ever since various experts in a variety of disciplines have been working on ways to adapt reinforcement learning in their research. This exciting achievement of AlphaZero started our interest in exploring the usage of reinforcement learning for trading.

The focus is to describe the applications of reinforcement learning in trading and discuss the problem that RL can solve, which might be impossible through a traditional machine learning approach.

What is reinforcement learning? Reinforcement learning might sound exotic and advanced, but the underlying concept of this technique is quite simple. In fact, everyone knows about it since childhood!

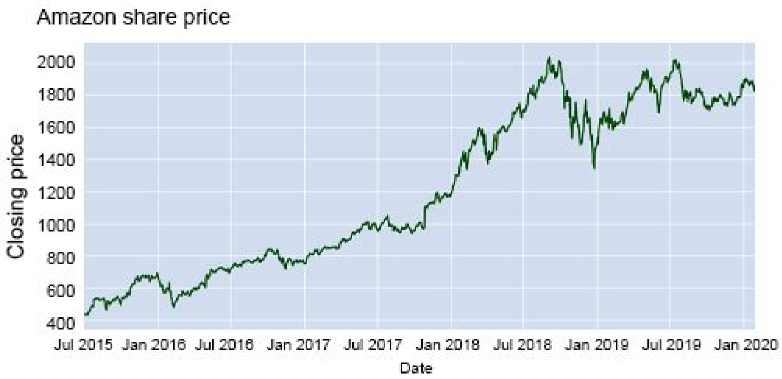
As a kid, you were always given a reward for excelling in sports or studies. Also, you were reprimanded or scolded for doing something mischievous like breaking a vase. This was a way to change your behaviour. Suppose you would get a bicycle or PlayStation for coming first, you would practice a lot to come first. And since you knew that breaking a vase meant trouble, you would be careful around it. This is called reinforcement learning. The reward served as positive reinforcement while the punishment served as negative reinforcement. In this manner, your elders shaped your learning.

Similarly, the RL algorithm can learn to trade in financial markets on its own by looking at the rewards or punishments received for the actions.

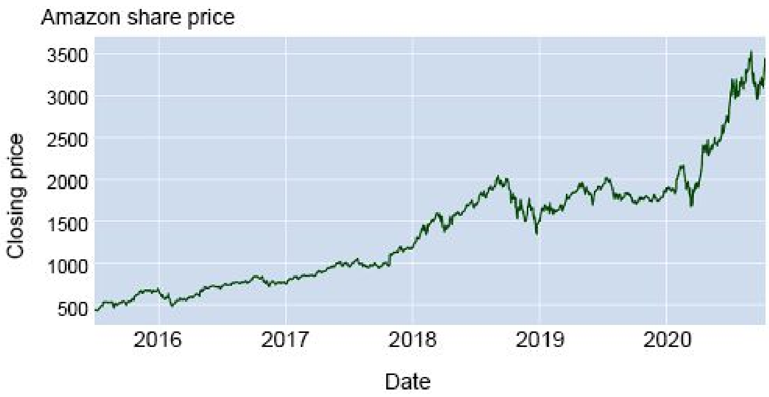
Like a human, our agents learn for themselves to achieve successful strategies that lead to the greatest long-term rewards. This paradigm of learning by trial-and-error, solely from rewards or punishments, is known as reinforcement learning (RL).

-Google Deepmind

How to apply reinforcement learning in trading? In the realm of trading, the problem can be stated in multiple ways such as to maximise profit, reduce drawdowns, or portfolio allocation. The RL algorithm will learn the strategy to maximise long-term rewards.



For example, the share price of Amazon was almost flat from late 2018 to the start of 2020. Most of us would think a mean-reverting strategy would work better here.



But if you see from early 2020, the price picked up and started trending. Thus from the start of 2020, deploying a mean-reverting strategy would have resulted in a loss. Looking at the mean-reverting market conditions in the prior year, most of the traders would have exited the market when it started to trend.

But if you had gone long and held the stock, it would have helped you in the long run. In this case, foregoing your present reward for future long-term gains. This behaviour is similar to the concept of delayed gratification which was talked about at the beginning of the article.

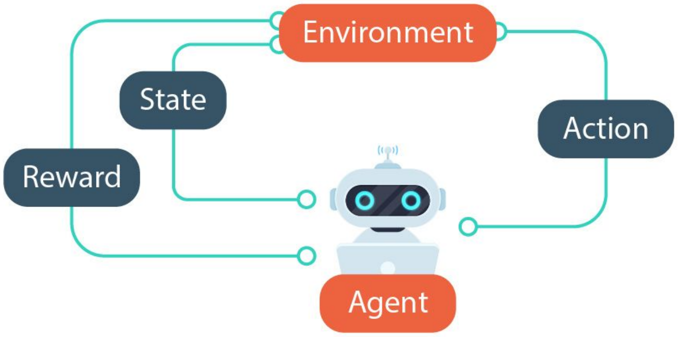
The RL model can pick up price patterns from the year 2017 and 2018, and with a bigger picture in mind, the model can continue to hold on to a stock for outsize profits later on. How is reinforcement learning different from traditional machine learning algorithms?

As you can see in the above example, you don't have to provide labels at each time step to the RL algorithm. The RL algorithm initially learns to trade through trial and error, and receives a reward when the trade is closed. And later optimises the strategy to maximise the rewards. This is different from traditional ML algorithms which require labels at each time step or at a certain frequency.

For example, the target label can be percentage change after every hour. The traditional ML algorithms try to classify the data. Therefore, the delayed gratification problem would be difficult to solve through conventional ML algorithms.

Components of Reinforcement Learning

With the bigger picture in mind on what the RL algorithm tries to solve, let us learn the building blocks or components of the reinforcement learning model.



Actions The actions can be thought of what problem is the RL algo solving. If the RL algo is solving the problem of trading then the actions would be

Buy, Sell and Hold. If the problem is portfolio management then the actions would be capital allocations to each of the asset classes. How does the RL model decide which action to take?

Policy There are two methods or policies which help the RL model take the actions. Initially, when the RL agent knows nothing about the game, the RL agent can decide actions randomly and learn from it. This is called an exploration policy. Later, the RL agent can use past experiences to map state to action that maximises the long-term rewards. This is called an exploitation policy.

State The RL model needs meaningful information to take actions. This meaningful information is the state. For example, you have to decide whether to buy Apple stock or not. For that, what information would be useful to you? Well, you can say I need some technical indicators, historical price data, sentiments data and fundamental data. All this information collected together becomes the state. It is up to the designer on what data should make up the state.

But for proper analysis and execution, the data should be weakly predictive and weakly stationary. The data should be weakly predictive is simple enough to understand, but what do you mean by weakly stationary? Weakly stationary means that the data should have a constant mean and variance. But why is this important? The short answer is that machine learning algorithms work well on stationary data. Alright! How does the RL model learn to map state to action to take?

Rewards A reward can be thought of as the end objective which you want to achieve from your RL system. For example, the end objective would be to create a profitable trading system. Then, your reward becomes profit. Or it can be the best risk-adjusted returns then your reward becomes a Sharpe ratio.
Defining a reward function is critical to the performance of an RL model. The following metrics can be used for defining the reward.

- Profit per Tick
- Sharpe Ratio
- Profit per Trade

Environment The environment is the world that allows the RL agent to observe State. When the RL agent applies the action, the environment acts on that action, calculates rewards and transitions to the next state. For example, the environment can be thought of as a chess game or trading Apple stock.

RL Agent The agent is the RL model which takes the input features/state and decides the action to take. For example, the RL agent takes RSI and past 10 minutes returns as input and tells us whether we should go long on Apple stock or square off the long position if we are already in a long position.

Let’s put everything together and see how it works.
Step 1

- State & Action: Suppose the Closing price of Apple was $92 on July 24, 2020. Based on the state (RSI and 10-days returns), the agent gave a buy signal.

- Environment: For simplicity, we say that the order was placed at the open of the next trading day, which is July 27. The order was filled at $92. Thus, the environment tells us that you are long one share of Apple at \$92.

- Reward: And no reward is given as we are still in the trade.

Step 2

- State & Action: You get the next state of the system created using the latest price data which is available. On the close of July 27, the price had reached \$94. The agent would analyse the state and give the next action, say Sell to environment.

- Environment: A sell order will be placed which will square off the long position.
- Reward: A reward of 2.1% is given to the agent.

Date Closing price Action Reward July 24 $92 Buy Na July 27 $94 Sell 2.1

Great! We have understood how the different components of the RL model come together. Let us now try to understand the intuition of how the RL agent takes the action.

Q Table and Q Learning

Q table and Q learning might sound fancy, but it is a very simple concept.

At each time step, the RL agent needs to decide which action to take. What if the RL agent had a table which would tell it which action will give the maximum reward. Then simply select that action. This table is Q-table.

In the Q-table, the rows are the states (in this case, the days) and the actions are the columns (in this case, hold and sell). The values in this table are called the Q-values.

Date	Sell	Hold
23-07-2020	0.954	0.966
24-07-2020	0.954	0.985
27-07-2020	0.954	1.005
28-07-2020	0.954	1.026
29-07-2020	0.954	1.047
30-07-2020	0.954	1.068
31-07-2020	0.954	1.090

From the above Q-table, on 23 July, which action would RL agent take? Yes, that’s right. A “hold” action would be taken as it has a q-value of 0.966 which is greater than q-value of 0.954 for Sell action.

But how to create the Q-table? Let’s create a Q-table with the help of an example. For simplicity sake, let us take the same example of price data from July 22 to July 31, 2020. We have added the percentage returns and cumulative returns as shown below:

Date	Closing Price	Percentage returns	Cumulative Returns
22-07-2020	97.2		
23-07-2020	92.8	-4.53%	0.95
24-07-2020	92.6	-0.22%	0.95
27-07-2020	94.8	2.38%	0.98
28-07-2020	93.3	-1.58%	0.96
29-07-2020	95	1.82%	0.98

30-07-2020	96.2	1.26%	0.99
31-07-2020	106.3	10.50%	1.09

You have bought one stock of Apple a few days back and you have no more capital left. The only two choices for you are “hold” or “sell”. As a first step, you need to create a simple reward table.

If we decide to hold, then we will get no reward till 31 July and at the end, we get a reward of 1.09. And if we decide to sell on any day then the reward will be cumulative returns up to that day. The reward table (R-table) looks like below. If we let the RL model choose from the reward table, the RL model will sell the stock and gets a reward of 0.95.

State/Action	Sell	Hold
22-07-2020	0	0
23-07-2020	0.95	0
24-07-2020	0.95	0
27-07-2020	0.98	0
28-07-2020	0.96	0
29-07-2020	0.98	0
30-07-2020	0.99	0
31-07-2020	1.09	1.09

But the price is expected to increase to $106 on July 31 resulting in a gain of 9%. Therefore, you should hold on to the stock till then. We have to represent this information in such a way that the RL agent can Hold rather than Sell. How to go about it? To help us with this, we need to create a Q table. You can start by copying the reward table into the Q table and then calculate the implied reward using the Bellman equation on each day for Hold action.

The Bellman Equation

$Q(s_t,a^t)= R(s_t,a^t)+g_{\kappa\leftarrow}\ \mathrm{Max}[Q(s_{t^+1},a_{t^+1})]_{i\ i}$

In this equation, s is the state, a is a set of actions at time t and a_i is a specific action from the set. R is the reward table. Q is the state action table but it is constantly updated as we learn more about our system by experience. *g* is the learning rate.

We will first start with the q-value for the Hold action on July 30. The first part is the reward for taking that action. As seen in the R-table, it is 0. Let us assume that g = 0.98. The maximum Q-value for sell and hold actions on the next day, i.e. 31 July, is 1.09. Thus q-value for Hold action on 30 July is 0 + 0.98 (1.09) = 1.06.

In this way, we will fill the values for the other rows of the Hold column to complete the Q table.

State/Action	Sell	Hold
23-07-2020	0.95	0.966
24-07-2020	0.95	0.985
27-07-2020	0.98	1.005
28-07-2020	0.96	1.026
29-07-2020	0.98	1.047
30-07-2020	0.99	1.068
31-07-2020	1.09	1.090

The RL model will now select the hold action to maximise the Q value. This was the intuition behind the Q table. This process of updating the Q table is called Q learning. Of course, we had taken a scenario with limited actions and states. In reality, we have a large state space and thus, building a q-table will be time-consuming as well as a resource constraint.

To overcome this problem, you can use deep neural networks. They are also called Deep Q networks or DQN. The deep Q networks learn the Q table from past experiences and when given state as input, they can provide the Q-value for each of the actions. We can select the action to take with the maximum Q value.

How to train artificial neural networks? We will use the concept of experience replay. You can store the past experiences of the agent in a replay buffer or replay memory. In layman language, this will store the state, action taken and reward received from it. And use this combination to train the neural network.

Issues in Reinforcement Learning There are mainly two issues which you have to consider while building the RL model. They are as follows:

Type 2 Chaos This might feel like a science fiction concept but it is very real. While we are training the RL model, we are working in isolation. Here, the RL model is not interacting with the market. But once it is deployed, we don’t know how it will affect the market. Type 2 chaos is essentially when the observer of a situation has the ability to influence the situation. This effect is difficult to quantify while training the RL model itself. However, it can be reasonably assumed that the RL model is still learning even when it is deployed and thus will be able to correct itself accordingly.

Noise in Financial Data There are situations where the RL model could pick up random noise which is usually present in financial data, and consider it as input which should be acted upon. This could lead to inaccurate trading signals. While there are ways to remove noise, we have to be careful of the tradeoff between removing noise and losing important data.

While these issues are definitely not something to be ignored, there are various solutions available to reduce them and create a better RL model in trading.

Conclusion

We have only touched the surface of reinforcement learning with the introduction of the components which make up the reinforcement learning system. As you delve deep, you will be able to understand the right parameters to be used and create a model with greater accuracy and precision.

This leads us to the not so happening part of this book, i.e. its conclusion. We have covered a long way together, haven’t we?

From understanding the basic tasks of machine learning to examining how different machine learning algorithms work. Humans have this propensity to try to create something which will explain everything. If you looked at physics, you will always see scientists trying to come up with one eqution to explain everything. In a similar manner, it was expected that there would be one ML or AI which would be able to do everything. That is a good dream to have. And reinforcement learning could be the answer.

But right now, we have different ML algorithms which are used for different purposes, each one with their own strengths and weakness. We hope you enjoyed this ride and gained some knowledge from this book. We also hope it made you curious on what more is out there in the ML world and its application in trading.

So keep learning and gaining knowledge.
As a parting note, do let us know how you liked it by leaving a comment at contact@quantinsti.com. It would help us improve the book further.

Additional Reading

1. The Hierarchical Risk Parity Algorithm: An Introduction - https://hudsonthames.org/an-introduction-to-the-hierarchical-risk-parityalgorithm/

2. K-Means Clustering Algorithm For Pair Selection In Python - https://blog.quantinsti.com/k-means-clustering-pair-selection-python/
3. Hierarchical cluster analysis [Chapter] - http://www.econ.upf.edu/~michael/stanford/maeb7.pdf
4. Eigen Portfolio Selection: A Ratio Maximization - programs/departments/finance/actuarial-science/seminar series/documents/WengPaper.pdf
5. Natural Language Processing in Trading [Course] - https://quantra.quantinsti.com/course/natural-language-processing-trading
6. Deep Reinforcement Learning in Trading [Course] - https://quantra.quantinsti.com/course/deep-reinforcement-learning-trading
Robust Approach to Sharpe https://business.unl.edu/academic

QuantInsti® is one of the pioneer algorithmic trading research and training institutes across the globe. With its educational initiatives, QuantInsti is preparing financial market professionals for the contemporary field of algorithmic and quantitative trading. QuantInsti has also designed education modules and conducted knowledge sessions for/with various exchanges in South and South-East Asia and for leading educational and financial institutions.

QuantInsti's flagship programme 'Executive Programme in Algorithmic Trading' (EPAT®) is designed for professionals looking to grow in the field algorithmic and quantitative Trading. It inspires individuals towards a successful career by focusing on derivatives, quantitative trading, electronic market-making, financial computing and risk management. This comprehensive certificate offers unparalleled insights into the world of algorithms, financial technology and changing market microstructure with its exhaustive course curriculum designed by leading industry experts and market practitioners.

Quantra® is an e-learning portal by QuantInsti that specializes in short self-paced courses on algorithmic and quantitative trading. Quantra offers an interactive environment which supports 'learning by doing' through guided coding exercises, videos and presentations in a highly interactive fashion through machine enabled learning.

Blueshift® is a comprehensive trading and strategy development platform that lets you focus more on the strategy and less on coding and data. It is fast (real-time), flexible and reliable. It is asset-class and instruments agnostic - we support multiple asset classes and instruments like FX, Equities, Futures. It is also style-agnostic. Whether you use factor strategies, technical indicators or advanced machine-learning, it can do it all. This makes developing complex (and simple) strategies easy, and moving a strategy from back-testing to live trading seamless.

QuantInsti Quantitative Learning Pvt. Ltd.

A-309, Boomerang, Chandivali Farm Road, Powai, Mumbai – 400 072

Email: contact@quantinsti.com