quantra

HOW TO TRADE USING MACHINE LEARNING

A COMPLETE GUIDE

INDEX

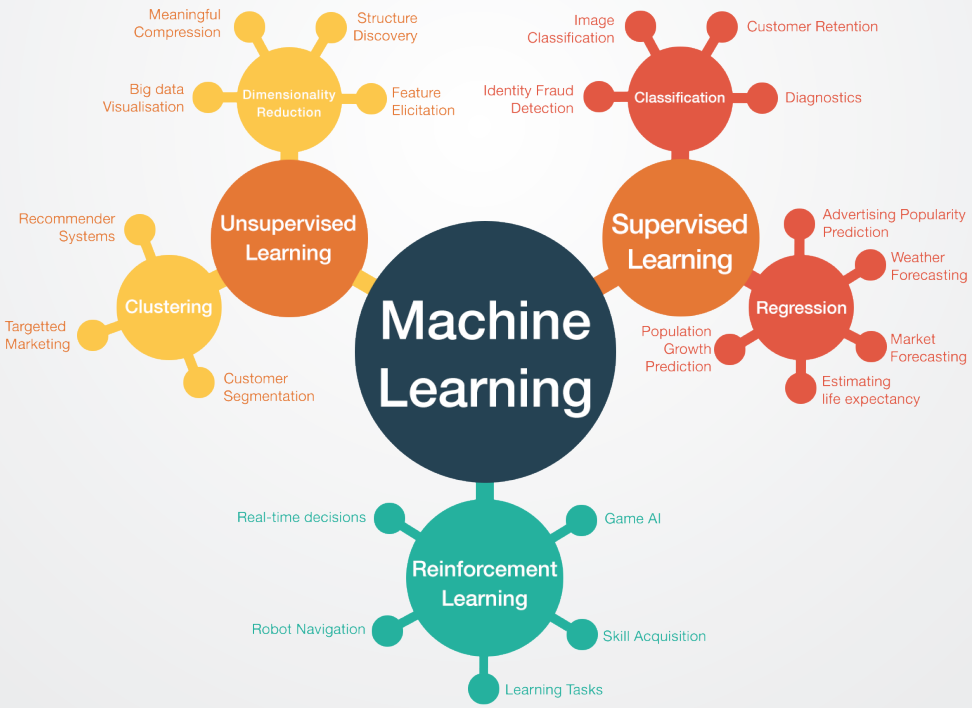
01 INTRODUCTION TO MACHINE LEARNING

What is Machine Learning ?

What do you think is Machine Learning? Machine Learning is everything on the list!

- Considered to be a subfield of Artificial Intelligence
- Involves learning models which allow the program to make predictions on data
- More than just a list of instructions which clearly define what the algorithm should do
- Closely linked to computational statistics which uses computers to make predictions

The third option here probably needs some explaining. 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 to perform, such tasks can be as simple as recognizing 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 ID-card photos. A Machine Learning (ML) algorithm would be trained on a training data to 'learn' to recognize any face. Where a simple algorithm would not be capable of performing this task, a ML algo would not only be able to categorize the photos as trained, but also learn from the testing data and add to its "learning" and become more accurate in its predictions. Recall how often Facebook prompts you to tag the person in the picture? Among billions of users, Facebook ML algos are able to correctly match different pictures of the same person and identify him/her!

02 MACHINE LEARNING IN FINANCIAL MARKETS

However, here we are to discuss the implementation and usage of machine learning in trading. Why have traders started to learn and use Machine Learning? In simple words, to get an advantage of complex mathematical/statistical computations which are difficult, if possible, to carry on in any other way. For instance, assume that you have an understanding of the market trend. You have a simple trading strategy using a few technical indicators which help you in predicting the market trend and trade accordingly.

Another trader, Machine Learning equipped, will follow the same approach but armed with ML algos, he/she will allow the machine to go through hundreds of technical indicators, instead of a few old preferred ones and let the machine decide which indicator performs the best in predicting the correct market trend. While the regular trader might have stuck to say RSI or MA, the ML algo would have referred many more technical indicators.

This technique is known as feature selection, where Machine Learning analyses different features of indicators and chooses the most effective ones for prediction. These features can be a price change, buy/sell signals, volatility signals, volume weights to name a few.

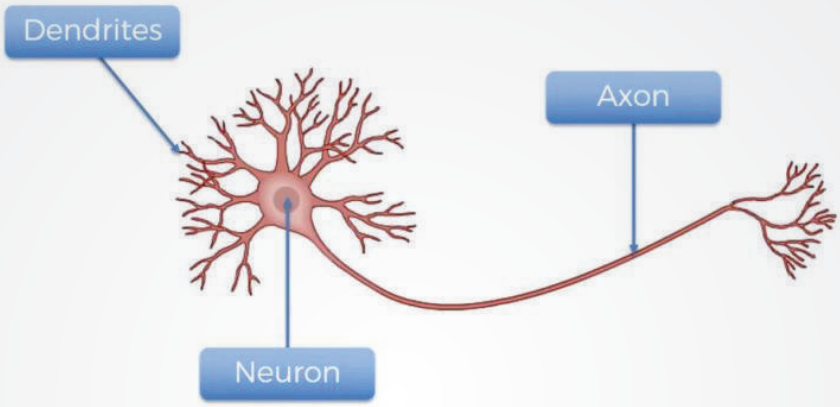
It is obvious that the trader who has done more quantitative research, backtesting on historical data, and better optimization has a good chance of performing better in live markets. However, the true effectiveness of a strategy depends on what happens in live markets!

03 ARTIFICIAL NEURAL NETWORK

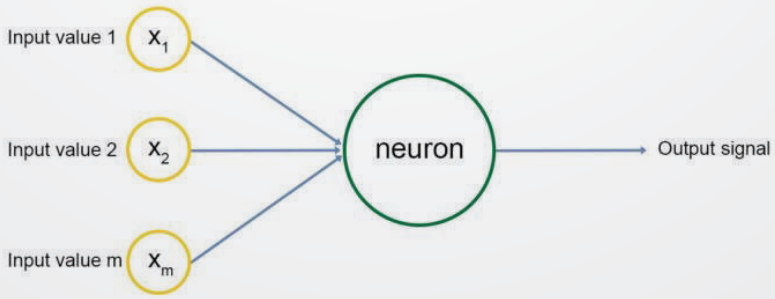
3.1 What is Artificial Neural Network?

Artificial Neural Network is an information processing paradigm which is used to study the behavior of a complex system by computer simulation. It is inspired by the biological way of processing of the information by human brain. The key element of the paradigm is the novel structure of the information processing system. The goal of the artificial neural network is to solve any specific problem in the same way as a human brain would.

Let us start by understanding what a neuron is.



This is the neuron that you must be familiar with, well if you aren't you should now be grateful that you can understand this because there are billions of neurons in your brain. There are three components to a neuron, the dendrites, the axon and the main body of the neuron. The dendrites are 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.



A computer neuron is built in a similar manner, as shown in the diagram. There are inputs to the neuron marked with yellow circles, 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 through concepts called gradient descent and backpropagation. An activation function is then applied to the weighted sum, which results in the output signal of the neuron.

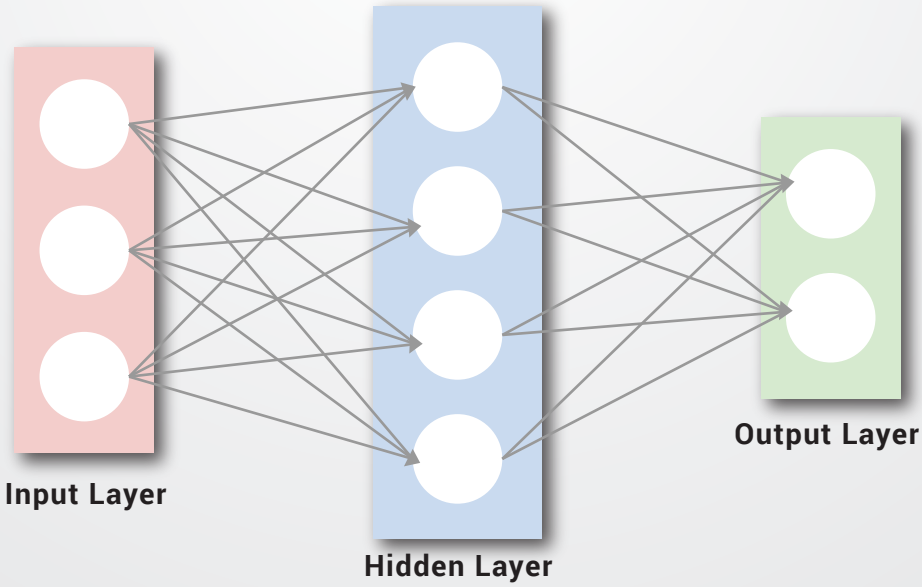
03

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.

Artificial Neural Network(ANN) is an information processing paradigm which is used to study the behaviour of a complex system by computer simulation. It is inspired by the biological way of processing of information by human brain. The key element of the paradigm is the novel structure of the information processing system. The goal of artificial neural network is to solve any specific problem in the same way as a human brain would.

ANN consist of multiple nodes which mimic the biological neurons of a human brain. As they are connected through links, they interact by taking the data and performing operations on it and then passing it over to the other connected node. Multi-Layer Perceptron classifier (MLP) is a classifier which consists of multiple layers of nodes. Each layer is fully connected to the next layer in the network.

Each link between the nodes is associated with a certain weight, an integer number which controls the signal between two nodes and interestingly, they have the power to alter the weight according to their previous learning of an event. If a network generates a "good or desired" output, there is no need to adjust the weights. However, if the network generates a "poor or undesired" output, then the system alters the weights in order to improve subsequent results. In short, they learn through examples and previous experiences. Artificial Neural Network computations can be carried out in parallel. Additionally, they can create their own organization or representation of information which they receive during the learning time.



Each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another

Artificial neural networking, with its phenomenal ability to derive results from complex data makes it an effective tool to solve a huge variety of tasks, for example, computer vision, handwritten digits recognition and speech recognition.

03

3.2 Artificial Neural Networks in Machine Learning

The ability of learning has made Artificial Neural Networks, a topic of interest. Given a specific task to solve, and a class of functions F, learning means using a set of observations to find, f*F which solves the task in some optimal sense.

Class MLP Classifier implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.
MLP trains on two arrays: array X of size (n_samples, n_features), which holds the training samples represented as floating point feature vectors; and array y of size (n_samples), which holds the target values (class labels) for the training samples:

Code:

```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False,
       epsilon=1e-08, hidden_layer_sizes=(5, 2), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

After fitting (training), the model can predict labels for new samples:

```
>>> clf.predict([[2., 2.], [-1., -2.]])
array([1, 0])
```

MLP can fit a non-linear model to the training data. clf.coefs_ contains the weight matrices that constitute the model parameters:

```
>>> [coef.shape for coef in clf.coefs_]
[(2, 5), (5, 2), (2, 1)]
```

03

Currently, MLP Classifier supports only the Cross-Entropy loss function, which allows probability estimates by running the predict_proba method.

MLP trains using Backpropagation. More precisely, it trains using some form of a gradient descent and the gradients are calculated using Backpropagation. For classification, it minimizes the Cross-Entropy loss function, giving a vector of probability estimates $P(y|x)$ per sample x:

```
>>> clf.predict_proba([[2., 2.], [1., 2.]])
array([[  1.967...e-04,   9.998...-01],
       [  1.967...e-04,   9.998...-01]])
```

MLP Classifier supports multi-class classification by applying Softmax as the output function.

Further, the model supports multi-label classification in which a sample can belong to more than one class. For each class, the raw output passes through the logistic function. Values larger or equal to 0.5 are rounded to 1, otherwise to 0. For a predicted output of a sample, the indices where the value is 1 represents the assigned classes of that sample:

```
>>> X = [[0., 0.], [1., 1.]]
>>> y = [[0, 1], [1, 1]]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(15,), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False,
       epsilon=1e-08, hidden_layer_sizes=(15,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
>>> clf.predict([[1., 2.]])
array([[1, 1]])
>>> clf.predict([[0., 0.]])
array([[0, 1]])
```

04 IMPLEMENTING ANN TO PREDICT STOCK MOVEMENTS

4.1 Coding the Strategy

4.1.1 Importing Libraries

We will start by importing a few libraries, the others will be imported as and 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.

```
import numpy as np
import pandas as pd
import tal ib
```

Numpy is a fundamental package for scientific computing, we will be using this library for computations on our dataset. The library is imported using the alias np.

Pandas will help us in using the powerful dataframe object, which will be used throughout the code for building the artificial neural network in Python.

Talib is a technical analysis library, which will be used to compute the RSI and Williams %R. These will be used as features for training our artificial neural network. We could add more features using this library.

4.1.2 Setting the random seed to a fixed number

```
import random
random.seed(42)
```

Random will be used to initialize the seed to a fixed number so that every time we run the code we start with the same seed.

4.1.3 Importing the dataset

```
dataset = pd.read_csv('RELIANCE.NS.csv')
dataset = dataset.dropna()
dataset = dataset[['Open', 'High', 'Low', 'Close']]
```

We then import our dataset, which is stored in the .csv file named 'RELIANCE.NS.csv'. This is done using the pandas library, and the data is stored in a dataframe named dataset. We then drop the missing values in the dataset using the dropna() function.

04

The csv file contains daily OHLC data for the stock of Reliance trading on NSE for the time period from 1st January 1996 to 15th January 2018. We choose only the OHLC data from this dataset, which would also contain the date, Adjusted Close and Volume data. We will be building our input features by using only the OHLC values.

4.2 Preparing the dataset

```
dataset['H-L'] = dataset['High'] - dataset['Low']
dataset['O-C'] = dataset['Close'] - dataset['Open']
dataset['3day MA'] = dataset['Close'].shift(1).rolling(window = 3).mean()
dataset['10day MA'] = dataset['Close'].shift(1).rolling(window = 10).mean()
dataset['30day MA'] = dataset['Close'].shift(1).rolling(window = 30).mean()
dataset['Std_dev']= dataset['Close'].rolling(5).std()
dataset['RSI'] = talib.RSI(dataset['Close'].values, timeperiod = 9)
dataset['Williams %R'] = talib.WILLR(dataset['High'].values, dataset['Low'].values,
dataset['Close'].values, 7)
```

We then prepare the various input features which will be used by the artificial neural network to train itself for making the predictions. We define the following input features:

- High minus low price
- Close minus open price
- Three day moving average
- Ten day moving average
- 30 day moving average
- Standard deviation for a period of 5 days
- Relative Strength Index (RSI)
- Williams %R

We then define the output value as price rise, which is a binary variable storing 1 when the closing price of tomorrow is greater than the closing price of today.

```
dataset = dataset.dropna()
```

Next, we drop all the rows storing NaN values by using the dropna() function.

```
X = dataset.iloc[:, 4:-1]
y = dataset.iloc[:, -1]
```

We then create two data frames storing the input and the output variables. The dataframe 'X' stores the input features, the columns starting from the fifth column (or index 4) of the dataset till the second last column. The last column will be stored in the dataframe y, which is the value we want to predict, i.e. the price rise.

4.3 Splitting the Dataset

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

In this part of the code, we will split our input and output variables to create the test and train datasets. This is done by creating a variable called split, which is defined to be the integer value of 0.8 times the length of the dataset.

We then slice the X and Y variables into four separate dataframes: X_train, X_test, y_train, and y_test. This is an essential part of any machine learning algorithm, the training data is used by the model to arrive at the weights of the model. The test dataset is used to see how the model will perform on new data which would be fed into the model. The test dataset also has the actual value for the output, which helps us in understanding how efficient the model is. We will look at the confusion matrix later in the code, which essentially is a measure of how accurate the predictions made by the model are.

4.4 Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Another important step in data preprocessing is to standardize 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 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 standardized. Now that the datasets are ready, we may proceed with building the Artificial Neural Network using the Keras library.

04

4.5 Building the Artificial Neural Network

```
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. 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 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:

1. 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.

2. 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 initialized with values from a uniform distribution.

3. 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'.

4. 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.

04

```
classifier.add(Dense(units = 128, kernel_initializer = 'uniform', activation =
'relu'))
```

We then add a second layer, with 128 neurons, with a uniform kernel initializer 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 = 'sig-
moid'))
```

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 = ['ac-
curacy'])
```

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 optimized 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 = 10, epochs = 100)
```

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.

04

4.6 Predicting the Movement of a Stock

```
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

Now that the neural network has been compiled, we can use the predict() method for making the prediction. We pass X_test as its argument and store the result in a variable named y_pred. We then convert y_pred to store binary values by storing the condition y_pred > 5. Now, the variable y_pred stores either True or False depending on whether the predicted value was greater or less than 0.5.

```
dataset['y_pred'] = np.NaN
dataset.iloc[(len(dataset) - len(y_pred)):,-1:] = y_pred
trade_dataset = dataset.dropna()
```

Next, we create a new column in the dataframe dataset with the column header 'y_pred' and store NaN values in the column. We then store the values of y_pred into this new column, starting from the rows of the test dataset. This is done by slicing the dataframe using the iloc method as shown in the code above. We then drop all the NaN values from dataset and store them in a new dataframe named trade_dataset.

4.7 Computing Strategy Returns

```
trade_dataset['Tomorrows Returns'] = 0.
trade_dataset['Tomorrows Returns'] = np.log(trade_dataset['Close']/trade_data-
set['Close'].shift(1))
trade_dataset['Tomorrows Returns'] = trade_dataset['Tomorrows Returns'].shift(-1)
```

Now that we have the predicted values of the stock movement. We can compute the returns of the strategy. We will be taking a long position when the predicted value of y is true and will take a short position when the predicted signal is false.

We first compute the returns that the strategy will earn if a long position is taken at the end of today, and squared off at the end of the next day. We start by creating a new column named 'Tomorrows Returns' in the trade_dataset and store in it a value of 0.

We use the decimal notation to indicate that floating point values will be stored in this new column. Next, we store in it the log returns of today, i.e. logarithm of the closing price of today divided by the closing price of yesterday. Next, we shift these values upwards by one element so that tomorrow's returns are stored against the prices of today.

```
trade_dataset['Strategy Returns'] = 0.
trade_dataset['Strategy Returns'] = np.where(trade_dataset['y_pred'] == True,
trade_dataset['Tomorrows Returns'], - trade_dataset['Tomorrows Returns'])
```

Next, we will compute the Strategy Returns. We create a new column under the header 'Strategy_Returns' and initialize it with a value of 0 to indicate storing floating point values. By using the np.where() function, we then store the value in the column 'Tomorrows Returns' if the value in the 'y_pred' column stores True (a long position), else we would store negative of the value in the column 'Tomorrows Returns' (a short position); into the 'Strategy Returns' column.

```
trade_dataset['Cumulative Market Returns'] = np.cumsum(trade_dataset['Tomorrows
Returns'])
trade_dataset['Cumulative Strategy Returns'] = np.cumsum(trade_dataset['Strategy
Returns'])
```

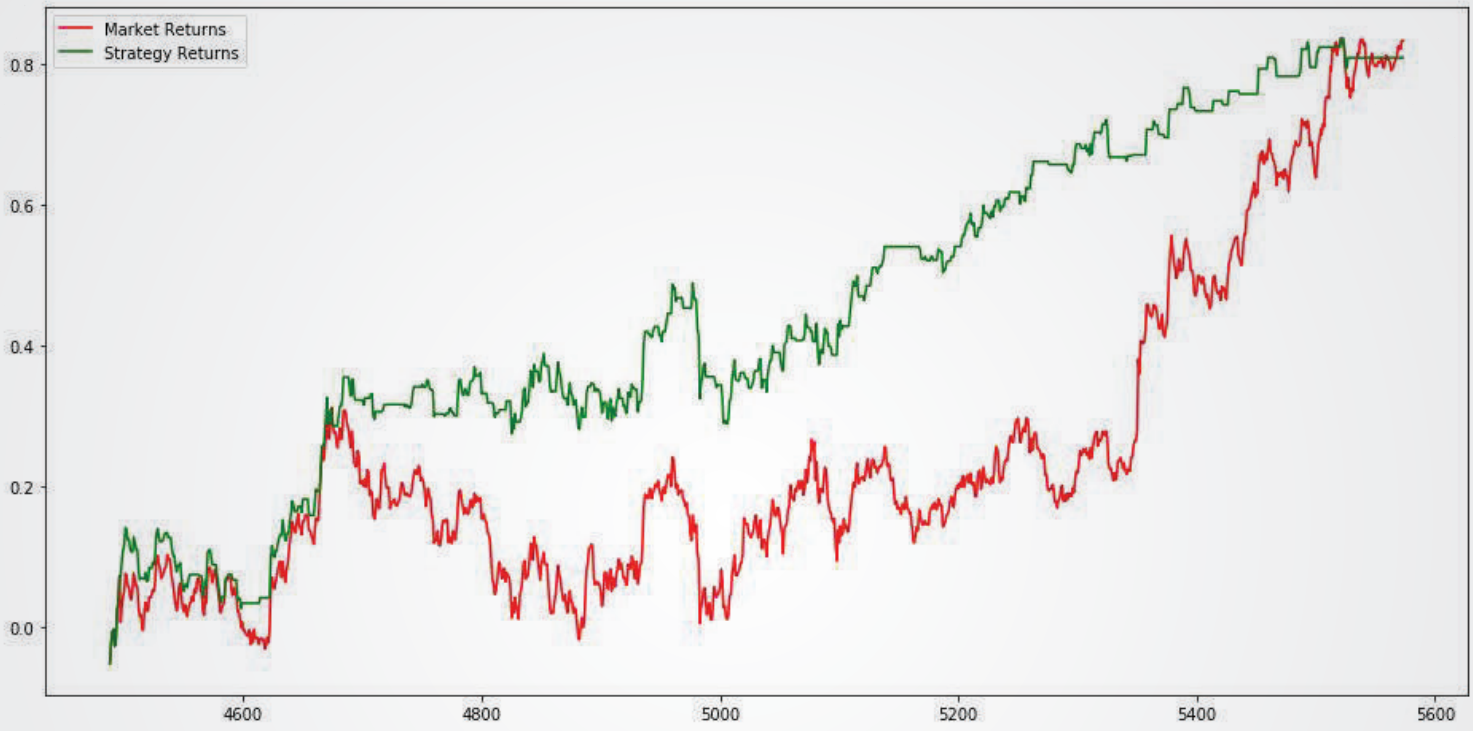
We now compute the cumulative returns for both the market and the strategy. These values are computed using the cumsum() function. We will use the cumulative sum to plot the graph of market and strategy returns in the last step.

4.8 Plotting the graph of returns

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.plot(trade_dataset['Cumulative Market Returns'], color='r', label='Market
Returns')
plt.plot(trade_dataset['Cumulative Strategy Returns'], color='g', label='Strategy
Returns')
plt.legend()
plt.show()
```

04

We will now plot the market returns and our strategy returns to visualize how our strategy is performing against the market. For this, we will import matplotlib.pyplot. We then use the plot function to plot the graphs of Market Returns and Strategy Returns using the cumulative values stored in the dataframe trade_dataset. We then create the legend and show the plot using the legend() and show() functions respectively. The plot shown below is the output of the code. The green line represents the returns generated using the strategy and the red line represents the market returns.



05 CONCLUSION

Now you can build your own Artificial Neural Network in Python and start trading using the power and intelligence of your machines. The Artificial Neural Network or any other Deep Learning model will be most effective when you have more than 100,000 data points for training the model. This model was developed on daily prices to make you understand how to build the model. It is advisable to use the minute or tick data for training the model, which will give you enough data for an effective training.

Predicting stock movements using ANN is just small part of Machine Learning implementation in Trading. We have a set of 3 courses that cover various ML techniques (Regression, Classification, SVM and many more) and how you can implement them in your day to day trading.