Udacity

Machine Learning Engineer Nanodegree

Capstone Project

Stock Price Trend Prediction

Author:
Aman Danish

Contents

1 Project Definition

1.1 Project Overview

Stock price predictions are imperative to the process of investment and risk management. It has always been a topic of interest for researchers which has lead to multiple discoveries with the advent of technology.

According to Efficient Market Hypothesis, it is impossible to achieve consistent risk adjusted returns above the benchmark in the long term. This is based on the assumption that all the underlying information of stock is available to all investors, which is immediately reflect in its price. However, the notion has been challenged since the beginning and with the developments in predictive modelling, has been revisited in recent literature.

Rajshree and Pradipta Dash designed novel decision support sysstems using artificial neural networks to propose set of rules for efficient trading [1]. Jigar Patel and Sahil shah employed Artificial Neural Networks, SVM , Random Forest and Naive Bayes to predict the direction of movement of stocks of Indian capital markets [2]. Jan Ivan used domain knowledge, machine learning and a money management strategy to create substantial profits through simulations on the Oslo Stock Exchange[3]. Lamartine Almeida and Adriano proposed a method of automatic stock trading through technical analysis and nearest neighbor classification[4].

In this project, we will be applying similar machine learning techniques to explore their effects in predicting trends and will try to determine the their efficacy in short-medium timeframe. Also, a comparison of techniques will be performed to evaluate the most accurate approach.

Me, being from financial background was most fascinated about this topic because I was always perplexed in choosing between so many Technical indicators for stock trading, never understanding which of them was most effective. This project gives me a golden opportunity to analyze this problem through empirical evidence.

1.2 Problem Statement

The main aim of the project is to build a model which at any time instance, determines the price trend in upcoming short-medium term of 10 days trough technical indicators of price data. The trend signal will be in form of labels:

- **1** - Reversal trend from bear to bull
- **0** - No change in price trend
- **-1** - Reversal of Trend from bull to bear

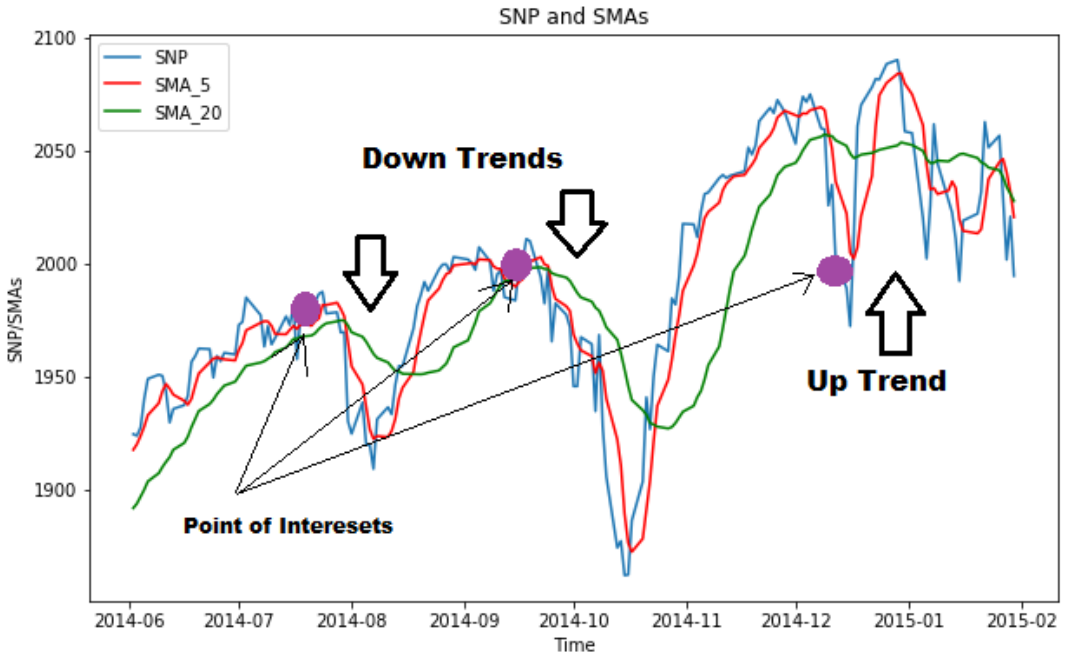


Figure 1: Example illustrating price trends and their points of interest

The model will use daily trading data of S&P 500 with the following attributes:

- Open - The price at the opening of market
- Close Price - The price at the close of market
- Adjusted Close price - Closing price adjusted for stock splits and dividends
- Volume - how many stocks were traded

In order to identify trends, we will be using price trend indicators of Moving Averages, calculated through TA-lib library. Furthermore, we will visualise relationships between price trend and other technical indicators attributing to trade momentum, volatility and strength. This will convert our problem from a time-series to a cross sectional classification problem. Hence, the solution will be formulated as :-

$$T_i = f(Momentum_i, Volume_i, Volatility_i)$$

where:
i = a point in time
T = {1,0,-1} Price trends as described in Section 1.1
f = the Classifier function
Momentum,Volume,Volatility are the respective technical indicators

1.3 Metrics

Our model solves a Classification problem on the basis of continuous numeric variables. Also, the number of labels/classes to be predicted is more than 2, so it's a multi-classification problem. This rules out the possibility of metrics such as AUC and AUCPR which are most robust for imbalanced binary classifications.

Also, it is worthy to consider that the distribution of classes in our data is heavily imbalanced

Label	Frequency
0	4480
1	171
-1	136

Table 1: Frequency table for labels of last 30 years data

So, we will need to explore other metrics such as average f1 score,accuracy etc. which mitigate our problems of imbalanced classes. For this problem, I have chosen these metrics for evaluation of our models:

- Micro f1 score(on positive labels): f1 Scores are best reflection of trade-off between precision and recall. For a subset of labels, micro score is equivalent to

$$\frac{2 * Precision_{micro} * Recall_{micro}}{Precision_{micro} + Recall_{micro}}$$

- Accuracy : Accuracy will give us an overall picture of our model. Note that in all our models, we will introduce sample weights for our observations, so losses will be calculated considering that non-zero labels contribute more towards losses

$$\frac{True\,Positives_i + True\,Negatives_i}{\Sigma(All\,values\,of\,Confusion\,Matrix)}$$

  where i stands for a label

- Micro-Precision : A measure of ow many labels predicted by our classifier are true positive. For subset of labels, micro - precision is :

$$\frac{TP_i + TN_i}{\Sigma(FP_i + TP_i\,and\,TN_i)}$$

2 Analysis

2.1 Data Exploration

Our input data consists of the daily price values (Open-High-Low-Close) along with Volume and Adjusted Close. We have chosen the granularity of data to be daily as we are interested in short-medium trends of data. Also, we didn't venture for the intra-day patterns because there is a significant level of noise and the analysis will be most suitable for High frequency trading. A brief description of these features has been delineated in section 1.2. A snapshot of the data is illustrated below:

```
In [6]: SNP_data.tail()
```

Out[6]:

Date	Open	High	Low	Close	Adj Close	Volume
2020-03-23	2290.709961	2300.729980	2191.860107	2237.399902	2237.399902	7402180000
2020-03-24	2344.439941	2449.709961	2344.439941	2447.330078	2447.330078	7547350000
2020-03-25	2457.770020	2571.419922	2407.530029	2475.560059	2475.560059	8285670000
2020-03-26	2501.290039	2637.010010	2500.719971	2630.070068	2630.070068	7753160000
2020-03-27	2555.870117	2615.909912	2520.020020	2541.469971	2541.469971	6194330000

Figure 2: Snapshot of input data

We have chosen the date range of our working dataset to be from 1950 to 2020 (70 years data). Our model prototyping will be done on 30 years of data and final evaluation will be done on 60 years of daily data.

```
In [8]: SNP_data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 17613 entries, 1950-01-03 to 2019-12-31
Data columns (total 6 columns):
Open         17613 non-null float64
High         17613 non-null float64
Low          17613 non-null float64
Close        17613 non-null float64
Adj Close    17613 non-null float64
Volume       17613 non-null int64
dtypes: float64(5), int64(1)
memory usage: 963.2 KB
```

Figure 3: Data type of input data

From above, we can see that the dates have been parsed and stored as Date-time index. All the features of Open, High, Low, Close are float objects. For further calculations, we will be considering 'Adjusted Close' as 'Close' as the former is adjusted for stock splits and dividends, giving a more accurate picture of Closing Price.

To explore more properties, we will be calculating daily returns of the Closing Price. This feature will be denoted as '1_day_return'

$$1\_day\_return = \frac{Close_t - Close_{t-1}}{Close_{t-1}}$$

Basic statistics of data are as follows:

```
In [13]: SNP_data.describe()
```

Out[13]:

	Open	High	Low	Close	Volume	1_day_return
count	17613.000000	17613.000000	17613.000000	17613.000000	1.761300e+04	17612.000000
mean	602.189355	605.676118	598.493530	602.321780	9.582886e+08	0.034546
std	727.082742	730.631133	723.220135	727.195771	1.569436e+09	0.960247
min	16.660000	16.660000	16.660000	16.660000	6.800000e+05	-20.466931
25%	86.370003	87.099998	85.690002	86.379997	9.170000e+06	-0.401151
50%	175.479996	177.300003	175.149994	176.529999	1.008500e+08	0.047659
75%	1113.859985	1120.270020	1106.420044	1113.859985	1.294900e+09	0.496257
max	3247.229980	3247.929932	3234.370117	3240.020020	1.145623e+10	11.580037

Figure 4: Statistics of features

```
In [14]: #Free view of daily returns
         SNP_data['1_day_return'].plot(kind = 'line',figsize = (10,6))
```

```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x25ea39cc0c8>
```

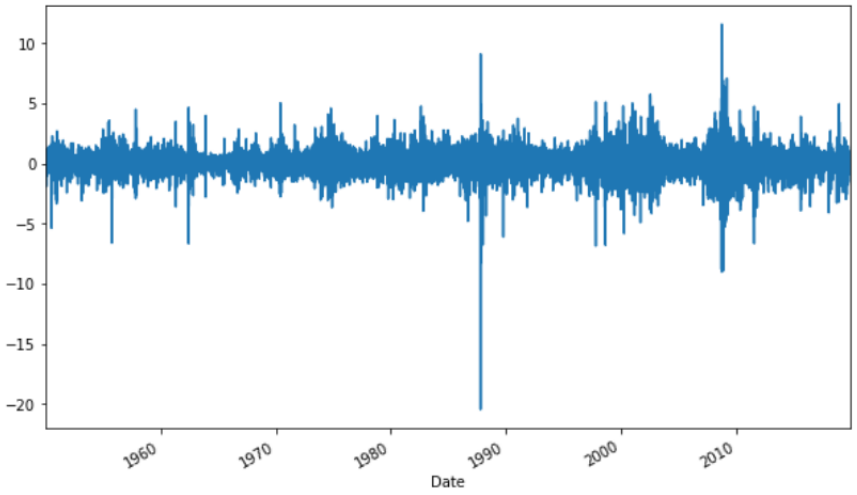


Figure 5: 1_day_return vs time

We can notice from the plot that there are 2 outliers of -20% and 11%. These will not be removed in order to retain the realism of data. Also, our features of concern will be dependant on price trend reversal and technical indicators, so the magnitude of returns won't affect our models much. To ensure the outliers do not affect the model, we will also employ regularisation in our classification algorithms.

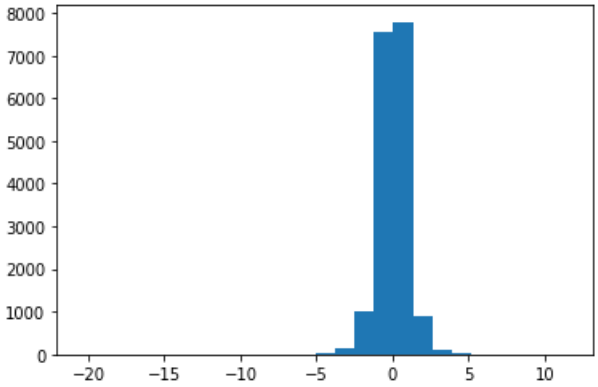


Figure 6: 1_day_return Histogram

```
Mean of daily return : 0.0345
Median of daily return : 0.0477
Standard Deviation of daily return : 0.9602
Kurtosis of daily return : 20.5865
Skewness of daily return : -0.6416
```

Figure 7: Properties of return distribution

Return range		Frequency
-20.499	-19.185	1
-19.185	-17.903	0
-17.903	-16.621	0
-16.621	-15.339	0
-15.339	-14.058	0
-14.058	-12.776	0
-12.776	-11.494	0
-11.494	-10.212	0
-10.212	-8.93	1
-8.93	-7.648	3
-7.648	-6.366	8
-6.366	-5.084	10
-5.084	-3.803	29
-3.803	-2.521	139
-2.521	-1.239	1014
-1.239	0.0431	7548
0.0431	1.325	7792
1.325	2.607	906
2.607	3.889	107
3.889	5.171	42
5.171	6.453	6
6.453	7.734	3
7.734	9.016	0
9.016	10.298	1
10.298	11.58	2

Table 2: Histogram table for returns

If we observe the histogram of daily returns,we can observe that most of the returns are in range -1.2% to 1.3% . Though the mean and standard deviation are close to 0 and 1, the skewness and kurtosis denote that the distribution is leptokurtic with negative skewness.

2.2 Additional features - Technical Indicators

Major goal of this project is to determine the relationship between the technical indicators and price trends, which will be done through technical analysis. The main assumption of TA is that the prices of a stock are dependant on the trades which have occurred. For liquid and efficient markets, it's an excellent regulator of prices which reflects the effects of news and inherent characteristics in real-time.

For our current project, we will be using following indicators on the basis of their characteristics. These are chosen on the basis of their popularity and effectiveness in short-medium time frame. We will not look into the calculation of these indicators as it's out of scope of project.

1. **Relative Strength Index** - It gauges the current and historical strength or weakness of a stock or market, based on the closing prices of recent trading period. It is classified as a momentum oscillator and computes momentum as the ratio of higher closes to lower closes. We chose the time period parameter as 14 as it's the most industry-wide used value.

2. **Simple Moving Average** - They are mainly used to smoothen the fluctuations of a price chart. Larger the value of time period, more smooth will be moving averages. We will be using two moving averages to determine price moves in medium term. SMA_5 will be Simple moving average for 5 days. SMA_20 will be simple moving average for 20 days. The price trend can be determined by faster average crossing the slower moving average.

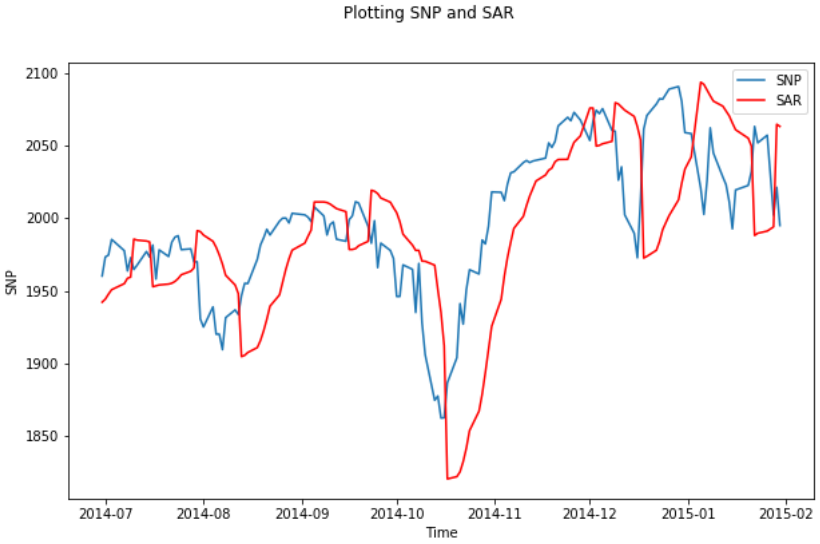
3. **Bollinger Bands** - These bands are constructed based on standard deviation of closing prices of last n periods. They are typically drawn 2 standard deviations above and below the n period moving average. To be consistent with RSI, we chose the value of n as 14. When the closing prices reach one of the extremes of bollinger band, it bounces in opposite direction. The periods with thinner range of bands denote low volatility and thicker denote high volatility.

4. **Parabolic SAR** - Parabolic Stop and Reverse is a method designed to find potential price trend reversals. When the price is in uptrend, the SAR emerges from the below and converges above. Similarly, in a downtrend, a SAR emerges above the price and converges downwards. The α value is called the acceleration factor which is usually set to 0.02.

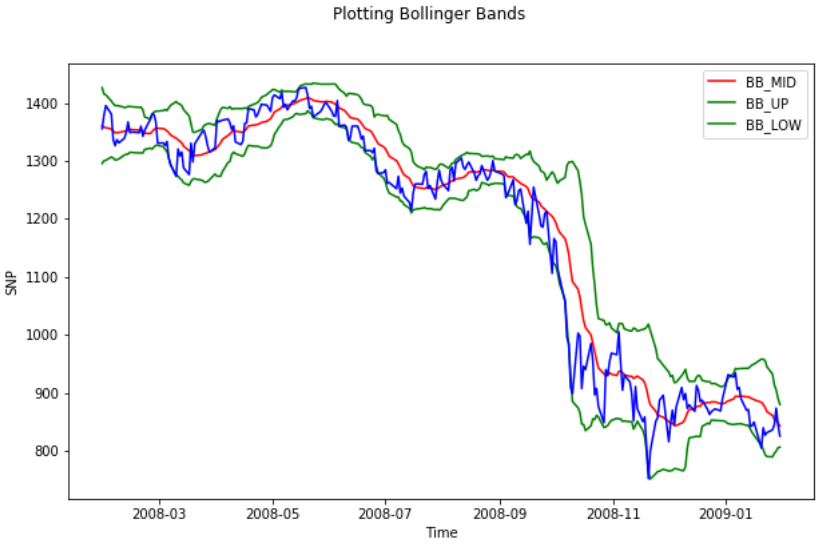
5. **Average directional moving index** - It's an indicator of trend strength and usually determines if the trend will persist. As a convention, values above 70 are referred as high strength trend.

2.3 Exploratory Visualization

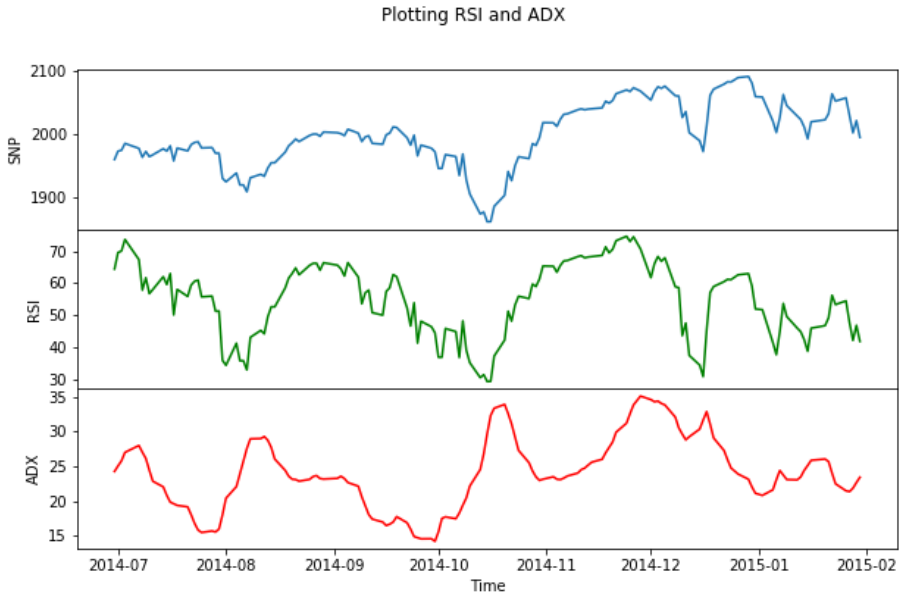
In this section, we will be visualising the plots of the indicators we discussed before and will analyse their relationship between the Closing price.



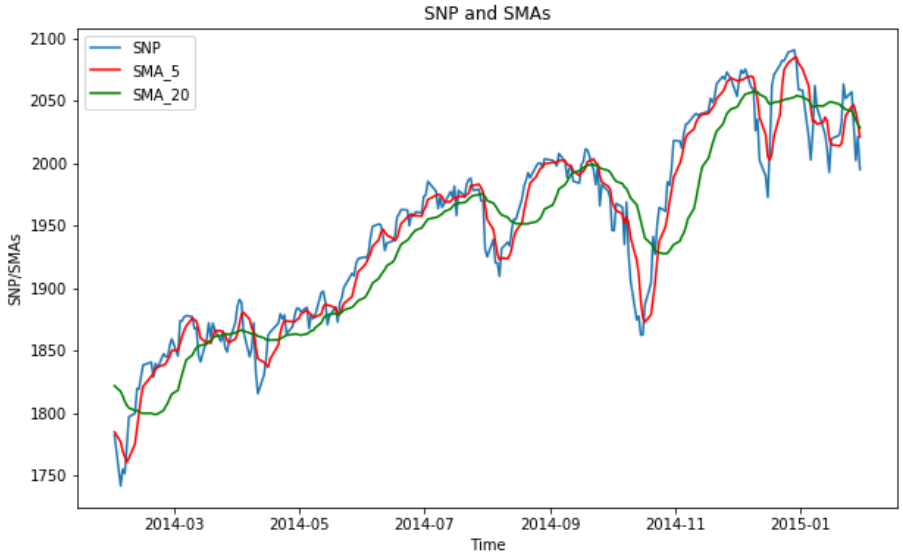
If we observe the trend, we will notice that the SAR graph is reverting post the trend has been materialised. However, we require an indicator which provides a signal before the trend. So, we will exclude this indicator from our list of features.



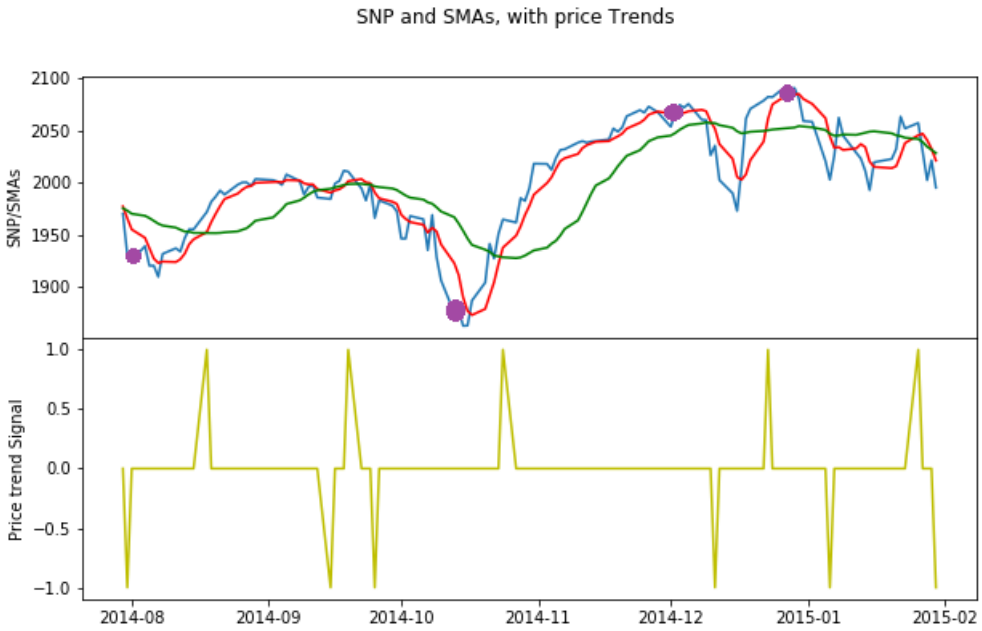
In above figure, we can see the relationship between Closing price(blue) volatility and Bollinger Band width. Also, when the Closing Price reaches any one of the extremes of the bands, it bounces back in the opposite direction in upcoming days.



Above figure illustrates the simultaneous relationship between Price, RSI and ADX. Price trends usually reverse when the RSI values cross 60-70 mark. Similarly, higher the value of ADX, higher will be the strength of trend.



Lastly, we will analyse the relation of price movements with Smooth moving averages. In order to gauge the short-medium trend, we have defined trend reversals in terms of SMAs crossing each other. When the faster moving SMA (5 day) crosses from above of slower SMA (20), it leads to a bearish trend labelled as -1. Similarly, when the faster SMA crosses the slower one from below, it leads to bullish trend labelled as +1.



The above plot illustrates the price trend labels (yellow) plotted against the closing prices and SMAs (red and green). Our points of interest are marked in violet. One of the most difficult problem we encountered here was to determine the time lag between the trend reversal and points of interest. Upon further scrutiny and visualisation, it was found that the price trend reversals most commonly took place within the time frame of 3-10 days, varying upon the steepness and strength of reversal. The main aim of the project is to predict the trend reversal based on the technical indicators of price action, rather than determining the time lag. So, we generated new features based on rolling 10 day maximum and minimum. This will be further discussed in next section.

3 Methodology

3.1 Data Preprocessing

3.1.1 Target Feature

Our target feature of Price Trend is a result of intersection of the Simple moving averages discussed before. Its detailed definition used for feature generation is as follows:

- +1 : If SMA_5-SMA_20 is negative for last three days and positive for today.
- -1 : If SMA_5-SMA_20 is positive for last three days and negative for today.
- 0 : Otherwise.

We added a rule of 3 days just to remove the associated noise.

3.1.2 Predictor Features

We figured out in the previous section that it will be quite challenging to find the right lag between the Trend and the point of interest. Also, as our problem is a classification problem, we leave the generation of the right time lag as an extension of this project.

To determine the most likely indicator properties contributing to a price trend, we filter-in the properties of indicator and generate the features in the following manner:

1. For a date where Price trend is non zero, find the 10 day rolling metrics of the individual indicators:
   - Max RSI if Price Trend +1, Min if it is -1
   - Max Volume in both cases of Price Trend
   - Max ADX in both cases of Price Trend
   - Min of $\frac{Bollinger\_band\_UP - Bollinger\_band\_MID}{Bollinger\_band\_MID}$ if Price Trend -1
   - Min of $\frac{Bollinger\_band\_LOW - Bollinger\_band\_MID}{Bollinger\_band\_MID}$ if Price trend is +1
2. Replace the last 10 days of data with the engineered feature data.
3. Don't change the values which have no trend (Price Trend 0).

Above steps will refine our data and will lead to the features with most favourable characteristics for a price trend. (Discussed more in implementation section 3.2.4)

3.1.3 Miscellaneous

After every step of return calculation and rolling feature calculation, rows with any NAs were removed. We removed the rows rather than imputing them with mean/std deviation because we had sufficient data of 30 years at disposal.

Also, for classification algorithms like Support Vector Machines and K Nearest Neighbours, the input features were scaled using SKlearn's Standard Scaler. This method centers the data with respect to its mean and standardizes it by dividing values with their standard deviation. This ensures that features with large magnitude do not dominate the classification.

Test-Train splitting was achieved through SKlearn's train-test split functionality with proportion of 3:7 and stratify parameter the target feature, to bring class imbalances into the account.

```
X_train, X_test, y_train, y_test = train_test_split(SNP_data_preprocessed.loc[:,'Volume_Rolling_10':'BB_LOW_MID_Rolling_10'],
                                                    SNP_data_preprocessed.loc[:,'Price_trend'],
                                                    test_size = 0.3, random_state=42,
                                                    stratify = SNP_data_preprocessed.loc[:,'Price_trend'])
```

3.2 Implementation

In this section, we will be discussing the major parts of data processing and the challenging parts of implementation

3.2.1 Return Calculation

To calculate daily returns, we used pandas .pct_change() method

```
In [10]: #Make a returns column
         SNP_data[['1_day_return']] = SNP_data[['Adj Close']].pct_change() * 100
```

3.2.2 TA-lib

For calculating Technical indicators, TA-lib was used

```
In [23]: #Momentum indcator
         SNP_data['RSI'] = ta.RSI(SNP_data['Close'],timeperiod = 14)

In [24]: #Trend indicator
         SNP_data['ADX'] = ta.ADX(SNP_data['High'], SNP_data['Low'], SNP_data['Close'], timeperiod=14)

In [25]: #Volatility Indicator
         SNP_data['BB_UP'], SNP_data['BB_MID'], SNP_data['BB_LOW'] = ta.BBANDS(SNP_data['Close'], timeperiod=14, nbdevup=2, nbdevdn=2, ma
         SNP_data['BB_UP_MID'] = (SNP_data['BB_UP']-SNP_data['BB_MID'])/SNP_data['BB_MID']
         SNP_data['BB_LOW_MID'] = (SNP_data['BB_LOW']-SNP_data['BB_MID'])/SNP_data['BB_MID']

In [26]: #Trend indicator
         SNP_data['SAR'] = ta.SAR(SNP_data['High'], SNP_data['Low'], acceleration=0.02, maximum=0.2)
```

3.2.3 Price Trend Feature Generation

To implement feature trend generation, we looped through the existing dataset.

```
In [22]: SNP_data.dropna(axis = 0,inplace = True)
col_pos = SNP_data.columns.get_loc('SMA_diff_5_20')
SNP_data['Price_trend'] = 0
col_trend = SNP_data.columns.get_loc('Price_trend')

for ctr in range(3,len(SNP_data)):
    signal_long = ((SNP_data.iloc[ctr,col_pos]>0) & (SNP_data.iloc[ctr-1,col_pos]<0) & (SNP_data.iloc[ctr-2,col_pos]<0) & (SNP_d
    if signal_long:
        SNP_data.iloc[ctr,col_trend] = 1
    signal_short = ((SNP_data.iloc[ctr,col_pos]<0) & (SNP_data.iloc[ctr-1,col_pos]>0) & (SNP_data.iloc[ctr-2,col_pos]>0) & (SNP_
    if signal_short:
        SNP_data.iloc[ctr,col_trend] = -1
```

3.2.4 Rolling features generation

This was a challenging part of implementation. We first generated the 10 day rolling values of features. Then, we ran a loop from latest to earliest time frame to pick up the rolling features for non-zero labels, remove the prior 10 days of data and replace it with one rolling observation. Issues were detected when the non zero label was in the first 10 observations, for which we just added additional conditions.

```
def add_rolling_feature(data,start_date,end_date):
    data = data.loc[start_date:end_date]
    data.dropna(axis = 0,inplace = True)
    data.reset_index(inplace = True,drop=True)


    #Rolling Strategy
    new_cols = ['Volume','RSI','ADX','BB_UP_MID','BB_LOW_MID']
    for ctr in new_cols:
        data[ctr+'_Rolling_10_max'] = data[ctr].rolling(10).max()
        data[ctr+'_Rolling_10_min'] = data[ctr].rolling(10).min()
        data[ctr+'_Rolling_10'] = 0

    data.dropna(axis = 0,inplace = True)
    data.reset_index(inplace = True,drop=True)


    i = data.shape[0]-1

    while i>=0 :
        if data.loc[i,'Price_trend'] == 1:
            data.loc[i,'Volume_Rolling_10'] = data.loc[i,'Volume_Rolling_10_max']
            data.loc[i,'RSI_Rolling_10'] = data.loc[i,'RSI_Rolling_10_min']
            data.loc[i,'ADX_Rolling_10'] = data.loc[i,'ADX_Rolling_10_max']
            data.loc[i,'BB_UP_MID_Rolling_10'] = data.loc[i,'BB_UP_MID_Rolling_10_min']
            data.loc[i,'BB_LOW_MID_Rolling_10'] = data.loc[i,'BB_LOW_MID']
            if (i-10) >= -1:
                data.drop(range(i-1,i-10,-1),inplace = True)
                i = i-10
            else:
                data.drop(range(i-1,-1,-1),inplace = True)
                i = -1
        elif data.loc[i,'Price_trend'] == -1:
            data.loc[i,'Volume_Rolling_10'] = data.loc[i,'Volume_Rolling_10_max']
            data.loc[i,'RSI_Rolling_10'] = data.loc[i,'RSI_Rolling_10_max']
            data.loc[i,'ADX_Rolling_10'] = data.loc[i,'ADX_Rolling_10_max']
            data.loc[i,'BB_UP_MID_Rolling_10'] = data.loc[i,'BB_UP_MID']
            data.loc[i,'BB_LOW_MID_Rolling_10'] = data.loc[i,'BB_LOW_MID_Rolling_10_max']
            if (i-10) >= -1:
                data.drop(range(i-1,i-10,-1),inplace = True)
                i = i-10
            else:
                data.drop(range(i-1,-1,-1),inplace = True)
                i = -1
        else :
            data.loc[i,'Volume_Rolling_10'] = data.loc[i,'Volume']
            data.loc[i,'RSI_Rolling_10'] = data.loc[i,'RSI']
            data.loc[i,'ADX_Rolling_10'] = data.loc[i,'ADX']
            data.loc[i,'BB_UP_MID_Rolling_10'] = data.loc[i,'BB_UP_MID']
            data.loc[i,'BB_LOW_MID_Rolling_10'] = data.loc[i,'BB_LOW_MID']
            i = i-1


    data = data[['Price_trend','Volume_Rolling_10','RSI_Rolling_10', 'ADX_Rolling_10', 'BB_UP_MID_Rolling_10','BB_LOW_MID_Rollir
    data.dropna(axis = 0,inplace = True)
    data.reset_index(inplace = True,drop=True)
    return data
```

Figure 8: Implementing Rolling feature generation

3.3 Algorithms, Techniques and Refinement

Even after refining our data through rolling feature generation, we still had a significant class imbalance.

Label	Frequency
0	4480
1	171
-1	136

Table 3: Frequency table for labels of last 30 years data

3.3.1 Class Weight generation

To address this, we used SKlearn's class weight method. This method gives a weight array inversely proportional to class frequency.

$$Class\_Weight = \frac{Total\_number\_Samples}{2 * Total\_Classes * Class\_Frequency}$$

This gave following weights to our labels which were assigned to each observation separately. The weight array was then passed to fit methods of respective classifiers as an argument

Label	Weight
-1	11.75
0	0.35
1	9.30

Table 4: Weights for classes

3.3.2 Simultaneous Hyperparameter Tuning

Note that we implemented Hyper-parameter Tuning at the same time of model fit as our data wasn't huge. This gave the best fit of each model and comparison was made on their best performances.

SKlearn's GridSearchCV was used to tune the hyperparameters of all classifier with micro-f1_score of non-zero labels as the evaluation metric, which was passed as a scoring argument wherever possible.

As large number of parameters are tested, we have mentioned results of only the best parameters. For more results, refer to Section 4.

3.3.3 Algo-1: Support Vector Machine Classifier

The goal of SVM is to learn a hyperplane or decision boundary which draws the separation between the two class labels in a way such that the distance of the hyperplane from them is maximum, i.e, the hyperplane is at an equal distance from them. It makes use of maximal margin classifier which is the optimal hyperplane separating the classes and does not suffer from overfitting.

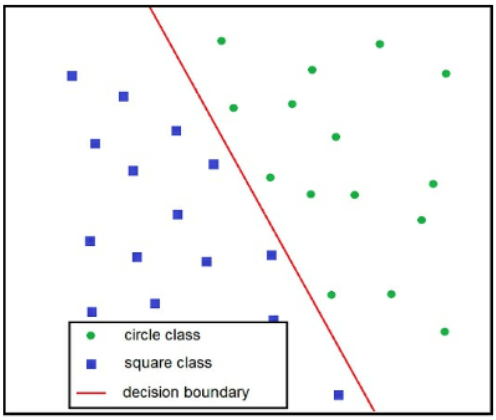


Figure 9: SVM Classifier

Sklearn's SVM Classifier has following parameters which are usually tuned:

- Kernel : A kernel is a mapping function which maps the 2 input vector spaces into a (higher dimension) space.
- C : This is a regularization parameter. The strength of the regularization is inversely proportional to C.

```
my_scorer = make_scorer(f1_score,average="micro",labels = [-1,1],greater_is_better=True )

param_grid= {'kernel':('linear', 'rbf','poly'),'C' : [0.1,1,10,100]}

grid_searcher = GridSearchCV( SVC(),param_grid,scoring=my_scorer,cv = 5,n_jobs = -1,verbose = 5 )
```

The best parameters for above Tuning were :

```
# Tuned hyper-parameters for micro f1_score
Best parameters set found on development set:

{'C': 100, 'kernel': 'linear'}
With micro f1_score : 0.6284
```

3.3.4 Algo-2 :K Nearest Neighbors Classifier

K-Nearest Neighbors (k-NN) is a technique which predicts the output label of the test example based on the labels of the closest k nearest neighbors from the training dataset. It first computes its distance from all the examples in the training dataset, then selects the k training examples with the lowest distances, and finally predicts the output label of the test example by either choosing the mode label.

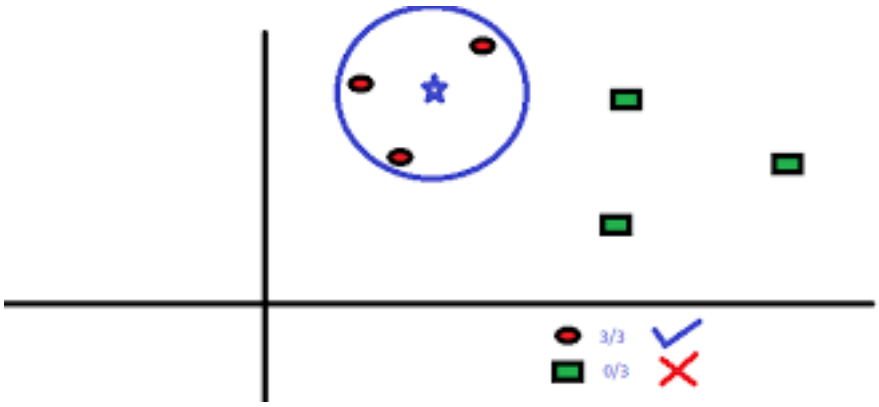


Figure 10: KNN Classifier

Following parameters were tuned for the classifier:

- n_neighbors : number of neighbors to get output from
- leaf_size : leaf size passed to BallTree or KDTree algorithm
- weights : How the points in neighborhood are weighted
- p : Power paramter for Minkowski Metric
- algorithm : Algorithm used to compute nearest neighbors

```
#Hyperparameter tuning KNN Classification

params = {'n_neighbors':[2,3,4,5, 6, 7, 8, 9,10],'leaf_size':[1,2,3,5],
          'weights': ['distance', 'uniform'], 'p':[2, 3],
         'algorithm':['auto', 'ball_tree','kd_tree','brute']}


my_scorer = make_scorer(f1_score,average="micro",labels = [-1,1],greater_is_better=True )

grid_searcher = GridSearchCV(KNeighborsClassifier(), scoring = my_scorer,cv = 5, param_grid = params, n_jobs = 4,verbose = 2)
```

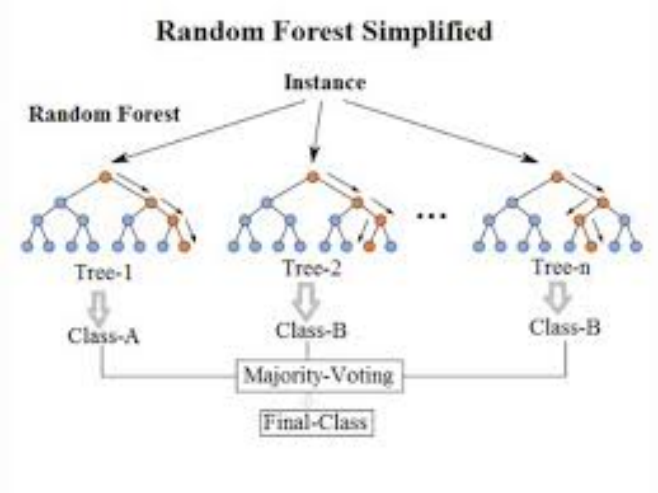
The best parameters for above Tuning were :

```
# Tuned hyper-parameters for micro f1_score
Best parameters set found on development set:

{'algorithm': 'auto', 'leaf_size': 1, 'n_neighbors': 2, 'p': 3, 'weights': 'distance'}
With micro f1_score : 0.2968
```

3.3.5 Algo- 3 : Random Forest Classifier

Random Forest is an Ensemble Learning method which operates by constructing a multitude of Decision Trees at the training time and outputting the class label that is the mode (most commonly occurring value) of the classes.



Following Parameters were tuned for Random Forest Classifier:

- Criterion : The function to measure the quality of split
- n_estimators : Number of trees in the forest.
- min_sample_leaf : The minimum number of samples needed at a leaf node.
- min_sample_split : The minimum number of samples to split an internal node
- random_state : Randomness of the bootstrapping
- class_weight : Computes weights associated with the classes.

```
#HyperParameter Tuning

params = {'criterion':['gini','entropy'],
          'n_estimators':[10,15,20,25,100,500,700],
          'min_samples_leaf':[1,2,3],
          'min_samples_split':[3,4,5,6,7],
          'random_state':[123],
         'class_weight': ['balanced_subsample']}


my_scorer = make_scorer(f1_score,average="micro",labels = [-1,1],greater_is_better=True )


grid_searcher = GridSearchCV(RandomForestClassifier(), scoring = my_scorer,cv = 5, param_grid = params, n_jobs = -1,verbose = 5)
```

The best parameters for above Tuning were :

```
# Tuned hyper-parameters for micro f1_score
Best parameters set found on development set:

{'class_weight': 'balanced_subsample', 'criterion': 'gini', 'min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 100,
'random_state': 123}
With micro f1_score : 0.5273
```

3.3.6 Algo - 4 : Decision Tree Classifier

A Decision Tree uses a tree-like structure, as a predictive model, to explicitly represent the decision and decision making. Each internal node of the Decision Tree is a feature and each outgoing edge from that node represents the value that the feature can take.Each leaf node represents a class label. Feature at each node is chosen based on the Information Gain.

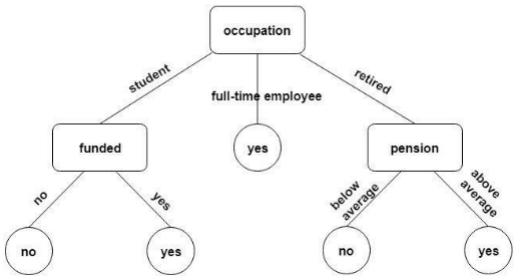


Figure 11: Decision Tree Classifier

Following parameters were tuned for decision Tree Classifier:

- Criterion : The function to measure the quality of a split. 'Gini and 'Entropy'
- max_features : The number of features to consider when looking for the best split
- min_samples_split : The minimum number of samples required to split an internal node
- min_samples_leaf : The minimum number of samples required to be at a leaf node
- random_state : Controls both the randomness of the bootstrapping of the samples and features.

```
params = {'criterion':['gini','entropy'],
          'max_features': ['auto', 'sqrt', 'log2'],
          'min_samples_split': [2,3,4,5,6,7,8,9,10,11,12,13,14,15],
          'min_samples_leaf':[1,2,3,4,5,6,7,8,9,10,11],
          'random_state':[123],'class_weight': ['balanced']}

my_scorer = make_scorer(precision_score,average="micro",labels = [-1,1],greater_is_better=True )


grid_searcher = GridSearchCV(DecisionTreeClassifier(), scoring = my_scorer,cv = 5, param_grid = params, n_jobs = -1,verbose = 5)
```

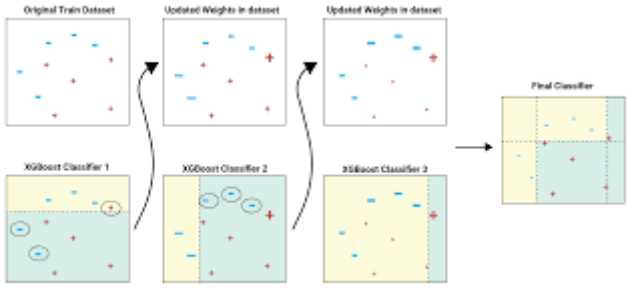
The best parameters for above Tuning are :

```
# Tuned hyper-parameters for micro f1_score
Best parameters set found on development set:

{'class_weight': 'balanced', 'criterion': 'entropy', 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'ra
ndom_state': 123}
With micro f1_score : 0.4699
```

3.3.7 Algo 5 : XGBoost Classifier

Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. 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. XGBoost is one of the popular algorithm for Gradient boosting.



XGBoost has a vast parameters which allows us to customize the model according to the input dataset. This is one of the reasons why it has superior speed and performance. For our project, we tuned following parameters :

- Max_depth : Maximum tree depth for base learners
- Min_child_weight : Minimum sum of instance weight(hessian) needed in a child
- subsample : Subsample ratio of the training instance
- colsample_bytree : Subsample ratio of columns when constructing each tree
- learning_rate : Boosting learning rate (eta)
- n_estimators : Number of gradient boosted trees

XGBoost only recieves merror and mlogloss as scoring metric for multiclass classification. So, we used merror with sample weights to train and tune our model. We will still be comparing the final result with other models through micro precision and accuracy.

```
params_grid = {'max_depth': [3,5,7],
'min_child_weight' : [0.2,0.6,1],
'subsample' : [0.8, 1],
'colsample_bytree' : [0.8,1],
'learning_rate' : [0.1],
'n_estimators': [500,600]
        }
|
gsearcher = GridSearchCV(estimator = XGBClassifier(objective= 'multi:softmax', eval_metric = 'merror', num_class = 3 ,nthread=4,
param_grid = params_grid, n_jobs=4,cv=5,verbose= 10)
```

The best parameters for above Tuning were :

```
Best parameters set found on development set:

{'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 0.6, 'n_estimators': 500, 'subsample':
1}
With  merror: 0.9624
```

3.4 Benchmark

Usually, benchmarking of a sophisticated model is carried out with a model which is primitive and widely used. For classification problems, Logistic regression is the most widely used model for predictive modelling. Hence, we will be using a hypertuned Logistic Regression model for benchmarking our classifier models.It is a statistical method for analyzing a dataset in which one or more independent variables determine the outcome, that can have only a limited number of values.Given the input data x, weight vector w (coefficients of the independent variable x) and the probability of the output label y as P(), If P(y) more than 0.5, you predict the output as 1, otherwise 0.
The parameters on which the Logistic regression was tuned are as follows :

- penalty : Used to specify the norm used in the penalization
- C : Inverse of regularization strength
- solver : Algorithm to use in the optimization problem
- Multi_class : with 'multinomial', the loss minimised is the multinomial loss fit across the entire probability distribution

```
params = {'penalty' : ['l2'],
          'C' : [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
          'solver' : ['lbfgs', 'sag', 'saga','newton-cg'],
          'multi_class' : ['multinomial']}


my_scorer = make_scorer(f1_score,average="micro",labels = [-1,1],greater_is_better=True )

grid_searcher = GridSearchCV(LogisticRegression(), scoring = my_scorer,cv = 5, param_grid = params, n_jobs = -1,verbose = 10)
```

The best parameters for above Tuning were :

```
# Tuned hyper-parameters for micro f1_score
Best parameters set found on development set:

{'C': 1000, 'multi_class': 'multinomial', 'penalty': 'l2', 'solver': 'lbfgs'}
With micro f1_score : 0.5964
```

With Classification Report

```
[[  35    6    0]
 [   6 1246   93]
 [   0    3   48]]

 Precision Score of positive labels: 0.4560

              precision    recall  f1-score   support

          -1  0.853658536585 0.853658536585 0.853658536585        41
           0  0.992828685259 0.926394052045 0.958461538462      1345
           1  0.340425531915 0.941176470588 0.500000000000        51

    accuracy                      0.924843423800      1437
   macro avg  0.728970917920 0.907076353073 0.770706691682      1437
weighted avg  0.965703746556 0.924843423800 0.939200256946      1437
```

So, we will benchmark the models with the Logistic Regression of accuracy 92.4% , Micro Precision of 45.6% and micro f1-score of 59.6%. In case a model has one of these parameters lesser, we will make a judgement on the basis of its confusion matrix.

4 Results

4.1 Model Evaluation and Justification

The summary of the Classification report in (%) of all models is as follows :

			Non-Zero Labels	
S.No.	Model	Accuracy	Micro Precision	MicroF1 Score
1	Logistic Regression	92.48	45.6	59.64
2	SVM Classifier	93.11	47.95	62.84
3	KNN Classifier	93.11	44.93	29.68
4	Random Forest	95.47	80	52.73
5	DecisionTree	93.45	48.65	46.99
6	XG Boost	96.93	84.29	72.83

Table 5: Figures in % for the metrics

From above table, we can observe that nearly all of the above models outperformed our Benchmark model in one metric or other. Out of them, XGBoost has generated the best results in all metric scores. It also shows that Ensemble Learners such as XGBoost and Random Forest outperformed its inferior version such as Decision Tree. The confusion Matrices of all models are as follows (with predicted in horizontal and actual in vertical) :

SVM		
35	6	0
0	1256	89
0	4	47

KNN		
12	29	0
20	1307	18
0	32	19

Random Forest		
10	31	0
1	1336	8
0	25	26

Decision Tree		
12	29	0
19	1307	19
0	27	24

XGBoost		
26	15	0
3	1334	8
0	18	33

Logistic Regression		
35	6	0
6	1246	93
0	3	48

Table 6: Confusion Matrices of every model

The confusion matrices portray interesting classification characteristics of models. SVM and Logistic Regression performed well on label 1 classifier but performed poorly on precision of -1 label. Though they have tremendous recall for non-zero labels, they still add a bit noise to the 0 label classification.

XGBoost sacrificed some of the Recall but it generated the best trade-off for all the labels. As a result, we choose XGBoost as the best model which produced far better results in every aspect with respect to Benchmark and other models.

5 Final testing on 60 years of data

Finally, the XGBoost model we chose was tested on a larger dataset. The parameters were similar to the tuned Hyperparameters, except for some minor details which were tuned manually. The dataset received the same preprocessing, feature engineering and scaling procedure. The test-train split was done in the same manner. The final results of the run are as follows-

```
[[  62   26    0]
 [   2 2680   11]
 [   0   31   72]]

 Precision Score of positive labels: 0.9116

              precision    recall  f1-score   support

          -1  0.968750000000 0.704545454545 0.815789473684        88
           0  0.979174278407 0.995172669885 0.987108655617      2693
           1  0.867469879518 0.699029126214 0.774193548387       103

    accuracy                      0.975728155340      2884
   macro avg  0.938464719308 0.799582416881 0.859030559229      2884
weighted avg  0.974866757746 0.975728155340 0.974277052269      2884
```

Figure 12: Results on 60 years data

Micro Precision score of 91% on $\tilde{2}.5\%$ error seems to be a good prediction. The model also generated fine trade-off between Precision and Recall with micro f1 score of 79.2% Unfortunately, we cannot plot an individual decision tree in XGBoost which can provide us thresholds as there are multiple estimators which contribute to the classification and examining one will not do the justice. However, we can check the the importance plot of features according to gain parameter:

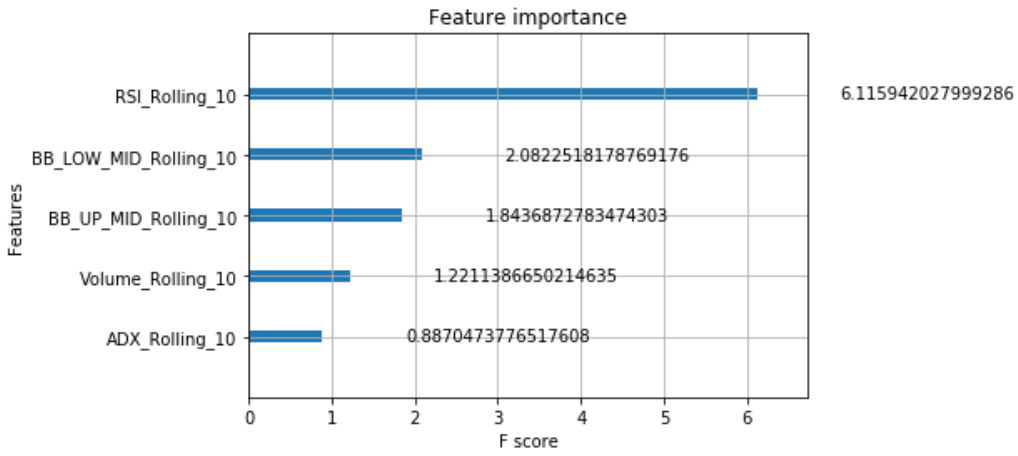


Figure 13: Feature importance

We can clearly see that RSI was the top most significant parameter in splitting the nodes. In fact, it was 3x significant in terms of Fscore in comparison to other parameters.

6 Conclusion

In this project, we tried to determine the model which produces the most accurate Price trend of S&P500 Index in upcoming 10 days, on the basis of technical indicators generated from its price data. A variety of classification algorithms were trained and tuned on 30 years of daily trading prices. With respect to benchmark model as Logistic regression, we found that XGBoost algorithm produced most accurate performance of prediction in terms of micro precision, f1 score and accuracy. This model did sacrifice some recall but provided best balance of evaluation scores.

Ultimately, The chosen XGBoost model was trained on 60 years of data and evaluated. It gave 97.5% Accuracy, 91.1% micro precision of non zero labels and 79.2% of micro-f1 score. Also, a feature importance graph was plotted with information gain as the criteria parameter. We found that in short-medium time frame, RSI seems to be the best indicator of predicting price trend. For other features, it might be the case that Bollinger Bands are effectve in short time frame of 3-5 days for very sharp trends, and ADX could be a lagging indicator. However, more research needs to be done on this finding and the current analysis can be extended to generate the time lag between technical indicators and Price trends in future projects.

References

[1] *"A hybrid stock trading framework integrating technical analysis with machine learning techniques"* Rajashree Dash, Pradipta Kishore Dash.

[2] *"Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques"*, Jigar Patel, Sahil Shah, Priyank Thakkar , K Kotecha

[3] *"Predicting Stock Prices Using Technical Analysis and Machine Learning" "*, Jan Ivar Larsen

[4] *"A method for automatic stock trading combining technical analysis and nearest neighbor classification"*, Lamartine Almeida Teixeira a, Adriano Lorena Inácio de Oliveira

[5] *"Cracking The Machine Learning Interview"* Nitin Suri

[6] Introduction to Data Science, Udacity

[7] Wikipedia : https://www.wikipedia.org/

[8] Wikipedia : https://www.wikipedia.org/

[9] Wikipedia : https://www.wikipedia.org/

[10] Investopedia : https://www.investopedia.com/

[11] Sklearn Documentation : https://scikit-learn.org/

[12] Analytics Vidhya : www.analyticsvidhya.com

[13] Machine Learning Mastery : machinelearningmastery.com