Noname manuscript No.
(will be inserted by the editor)

Predicting the Direction of Stock Market Price Using Tree Based Classifiers

Suryoday Basak · Snehanshu Saha · Saibal Kar · Luckyson Khaidem · Sudeepa Roy Dey

Received: date / Accepted: date

Abstract Predicting trends in stock market prices has been an area of interest for researchers for many years due to its complex and dynamic nature. Intrinsic volatility in stock market across the globe makes the task of prediction challenging. Forecasting and diffusion modeling, though effective, cannot be the panacea for the diverse range of problems encountered in predicting trends in the stock market, short-term or otherwise. Market risk, strongly correlated with forecasting errors, needs to be minimized to ensure minimal risk in investment. This paper is the outcome of experiments with a different approach: instead of defining this problem as a traditional forecasting-style problem, we just try to predict whether prices will increase or decrease. The problem is posed as a classification problem, where the class labels may be ± 1, indicating an increase or a decrease in the price of a stock with respect to n days back. For this purpose, the potential of Random Forests and XGBoosted trees is explored. Random Forests use an ensemble of Decision Trees to improve the accuracy of classification. XGBoost is an engineering solution which aims to speed up the process of growing Gradient Boosted Decision Trees (GBDT). Technical indicators such as Relative Strength Index (RSI), Stochastic Oscillator, etc. are used as features to train the model. The algorithms are shown to outperform the algorithms used in the existing literature.

Keywords stock direction prediction · machine learning · xgboost · decision trees

1 Introduction and Motivation

For a long time, it was believed that changes in the price of stocks is not forecastable. The well known Random Walk hypothesis (Malkiel and Fama 1970; Malkiel and Burton 2003), and the Efficient Market hypothesis (Jensen 1978), which states that a market is efficient with respect to a current information set $I(t)$ if it is impossible to make economic gains in this market, led to this belief. In other words, if it is impossible to outperform the market owing to the randomness in stock prices, then unless a different (often excessive) type of risk is considered, economic profits

Suryoday Basak, Snehanshu Saha, Luckyson Khaidem, Sudeepa Roy Dey
Department of Computer Science and Engineering, and Center for Applied Mathematical Modeling and Simulation (CAMMS), PESIT South Campus, Bangalore, India
Saibal Kar (Corresponding Author)
Tel.: +91 (0)33 2462 7252 / 5794 / 5795 / 2436 8313 / 7794 / 95 / 97
E-mail: saibal@cssscal.org
Centre for Studies in Social Sciences, Calcutta, India and IZA Institute of Labor Economics, Bonn

cannot rise. It is unclear, however, how such a risk would be measured (see (Timmermann and Granger 2004), where stock prices are treated as a martingale). Therefore, it should be of little doubt that predicting the trends in stock market prices is a challenging task. On the contrary, the Wisdom of Crown hypothesis, which emerges from the theory of collaborative filtering, states that many individuals, each with limited information, can provide very accurate assessments if their information is elicited in an appropriate fashion. It is, however, not known to be useful for predicting stock market returns; nonetheless, some individual, as well as institutional investors are able to beat the market to make profits (Avery et al. 2016). The inefficiency of prediction gets accentuated due to various uncertainties involved and owing to presence of multiple variables all of which can potentially influence the market value on a particular day. Over time, a number of explanatory variables have been added to this enormous literature (see a history of EMH in (Sewell 2011; Beechey et al. 2000) etc.): these include country-specific economic conditions, investors' sentiments towards a particular company, political events, etc. Consequently, stock markets are susceptible to quick changes, which may seem to be random fluctuations in the stock prices.

Stock market series are generally dynamic, non-parametric, chaotic and noisy in nature making investments intrinsically risky. Stock market price movement is considered to be a random process with fluctuations, that are more prominent in the short-run. However, some stocks tend to develop linear trends in the long-run. It is needless to mention that advanced knowledge of stock price movement in the future should help to minimize this risk. Traders are more likely to buy a stock whose value is expected to increase in the future and conversely for falling prices. So there is a need for accurately predicting the trends in stock market prices in order to maximize capital gain and minimize loss. Regardless, it had best be admitted that adding value to this complex and deeply researched topic is not easy. To this end, this paper presents the use of techniques of machine learning (ML) to predict stock prices at the level of a firm to get better insights into the accuracy of price movements. The ML models of choice are Random Forests (RF) and forests of Gradient Boosted Trees (GBDT). RF works by finding the best threshold based on which the feature space is recursively split, whereas GBDTs approximate regressors to the training samples and find the best split of the aggregate of the regressor functions. XGBoost is a fairly new invention, which is a tool that reduces the time taken to construct GBDTs by reducing the time for training of a complete model.

Application of ML models in stock market behavior is a rather recent phenomenon (Khaidem et al. 2016). The approach is a departure from traditional forecasting and diffusion type methods. Early models used in stock forecasting involved statistical methods such as time series modeling and multivariate analysis (Gencay 1999; Timmermann and Granger 2004; Bao and Yang 2008). The stock price movements are usually treated as a function of time and solved as a regression problem. Here, however, the problem is treated as a classification problem, where the class label of each sample is determined by measuring the change in price of a stock compared to it's price t days back. In our analysis, we have conducted experiments on $t = 3, 5, 10, 15, 30, 60,$ and90 days. The goal is to design an intelligent model that learns from the market data using machine learning techniques and predicts the *direction* in which a stock price will move.

2 Related Work

As already stated briefly, the use of prediction algorithms to determine future trends in stock market prices (Widom 1995; Hellstrom and Holmstromm 1998; Gencay 1999; Li et al. 2014; Dai and Zhang 2013; Timmermann and Granger 2004; Bao and Yang 2008) contradicts a basic rule in finance known as the Efficient Market Hypothesis (EMH) (Malkiel and Fama 1970). EMH states

that current stock prices fully reflect all the relevant information and implies that if someone were to gain an advantage by analyzing historical stock data, the entire market will become aware of this advantage. As a result, the price of the share will be corrected. Diffusion models (Saha et al. 2014) tend to support this idea. Although it is generally accepted, there are many researchers who have tried to ascertain its folly by demonstrating algorithms that can model more complex dynamics of the financial system (Malkiel and Burton 2003), thus fortifying the controversial nature of this hypothesis.

Consequently, several algorithms have been used in stock prediction such as support vector machine (SVM), artificial neural networks (ANN), linear discriminant analysis (LDA), linear regression, K-NN, and naïve Bayesian classifier (Khan et al. 2014). The relevant literature survey revealed that SVM has been used most of the time in stock prediction research. The sensitivity of stock prices to external conditions have been considered by Li et al. (2014): the external conditions taken into consideration include daily quotes of commodity prices such as gold, crude oil, natural gas, corn, and cotton in 2 foreign currencies (EUR, JPY). In addition to that, they collected daily trading data of 2666 U.S stocks trading (or once traded) at NYSE or NASDAQ from 1^{st} January 2000 to 10^{th} November 2014. This dataset includes the opening price, closing price, highest price, lowest price, and trading volume of every stock of every day for which the data was collected. Features were derived using the information from the historical stock data as well as external variables which were mentioned earlier in this section. It was found that logistic regression turned out to be the best model with a success rate of 55.65%. In the paper by Dai and Zhang (2013), the data used for the analysis were stock (closing) prices of the company 3M. The data contained daily stock information ranging from 1^{st} September 2008 to 11^{th} August 2013 (1471 data points). Multiple algorithms were chosen to train the prediction system. The algorithms used were logistic regression, quadratic discriminant analysis, and SVM. These algorithms were used to predict the direction of the stock on the successive day corresponding to a given data sample; it also predicted the price after the next n days. The accuracy of the successive-day prediction model ranged from 44.52% to 58.2%. The results reported by Dai and Zhang (2013) were justified on the grounds that the US stock market is semi-strong efficient, meaning that neither fundamental nor technical analysis can be used to achieve superior gains. However, the long-run prediction model produced better results which peaked when the time window was 44 days. SVM reported the highest accuracy of 79.3%. Di (2014) has used three stocks (AAPL, MSFT, AMZN) that span between 4^{th} January 2010 and 10^{th} December 2014. Various technical indicators such as RSI, On Balance Volume, Williams %R, etc. are used as features. Out of 84 features, an extremely randomized tree algorithm, as described by Geurts and Louppe (2014), was implemented for the selection of the most relevant features. These features were then fed to an SVM with RBF kernel for training. Devi et al. (2015) has proposed a model which uses a hybrid cuckoo search with support vector machine (with Gaussian kernel): the cuckoo search method is an optimizes the parameters of support vector machine. The proposed model used technical indicators: RSI, Money Flow Index, EMA, Stochastic Oscillator, and MACD as features. The data used in the proposed system consists of daily closing prices of BSE-Sensex and CNX - Nifty from Yahoo finance between January 2013 and July 2014. Giacomel et al. (2015) proposed a trading agent based on a neural network ensemble that predicts if a certain stock is going to rise or fall. They evaluated their model using two datasets: the North American and the Brazilian stock markets and achieved hit rates of greater than 56%. They even performed a simulation based on the predictions of their classifier, whose results were promising. Boonpeng and Jeatrakul (2015) implemented a one-against-all (OAA-NN) and one-against-one neural network (OAO-NN) to classify buy, hold, or sell data and compared their performance with a traditional neural network. Historical data of Stock Exchange of Thailand (SET) for seven years (3^{rd} January 2007 to 29^{th} August 2014) was selected. It was found that OAA-NN performed better than OAO-NN and traditional NN

models, producing an average accuracy of 72.50%. Qiu and Song (2016) have tried an optimized ANN using genetic algorithms (GA) to predict the direction of the stock market in a similar fashion of the current work; the hit ratio achieved here for two types of data, based on different sets of technical indicators, for the Nikkei 225 index (Tokyo Stock Exchange) are 61.87% and 81.27%.

The literature survey helps us to conclude that tree-ensemble learning algorithms have remained unexploited in the problem of stock market predictions. The focus of the current paper is therefore to implement Random Forests and XGBoost and to discuss its advantages over non-ensemble techniques. These models have been trained on time intervals of 3, 5, 10, 15, 30, 60, and 90 days days and the results are impressive. Moreover, majority of the related work focused on the time window of 10 to 44 days on an average as most of the previous studies preferred to use metric classifiers on time series data which was not smoothened. Therefore, these models are unable to learn from the data set when it comes to predicting for a long run window.

In the method we propose, the data is first smoothened using exponential smoothing, such that the more recent data gets more weight as compared to data which is comparatively very old. Overall, the paper will highlight certain critical aspects of data analysis which have been largely ignored in the previous literature (these include exploring the non-linearity in features and the futility of employing linear classifiers). We also offer an extension of the time window to 90 days over which predictions are made. This almost doubles the time adopted in previous studies (44 days). Significant improvement in accuracy obtained by our approach clearly establishes the superiority of the model developed. It should be instructive to point out that forecasting techniques are rather insensitive to the crests and troughs in the time-series data. In the event of data point insufficiency or improper data conditioning, incorrect forecasting is common. The forecasting may generate some random value(s) which turns out to be an out-lier. For example, forecasting the price of a commodity as zero is one such instance. From an economic point of view, unless a commodity is a free good, usually publicly provided, the commodity price is unlikely to be zero. On the contrary, the classification model provides a probabilistic view of the predictive analysis and hence it plays a safer role as it predicts the *direction* of the trend. It uses the likelihood of the situation and hence the results are more trustworthy. A possible way to avoid the inherent problems in forecasting methods is therefore by recasting the problem as one that implements tree-based classifiers in ensemble learning as we adopt here. Finally, the growing popularity of stock market trend prediction both in the field of economics and machine learning had been clear from the literature survey in the paper. However, to efficiently compute the results with minimal errors, the data needs to be sufficiently large. Consequently, the proposed application of machine learning to a dedicated problem in financial decision making via use of efficient algorithms and faster computation seems to be the most efficient path.

The remainder of the paper is organized as follows. In Section 3, we discuss the data used in this research, and in Section 4, the operations implemented on the data that include cleaning, preprocessing, feature extraction and characterizations such as Relative Strength Index, Stochastic Oscillator, etc. and testing for linear separability and learning the data via random forest and XGBoost ensemble. Sections 5 and 6 offer the descriptions about the random forests and XGBoost algorithms respectively, and provides an exposition of their working in the context of stock markets. Section 7 presents the results obtained by implementing the models developed. In addition to this, a specific case study on stocks of pharmaceutical stocks is presented, which is substantiated by past studies on price changes in pharmaceutical companies (Behner at al. 1998). We conclude by summarizing the results and the cornerstones of our research in Section 8.

3 Data Set:

The results are reported for ten companies (see Appendix B). For these ten companies, all of the data available have been used, starting from the day they went public till 3^{rd} February 2017. These companies were chosen at random, without any rigorous consideration of their background or the kind of economic impact they have on society. Some of these companies are software companies (FB, TWTR), electronics companies (AMS), automobile (TATA), sports (NKE), etc. We like to emphasize that the diversity of the background of companies thus chosen for analysis of stock prices is crucial for ensuring the efficacy of the algorithms. Further justification is provided in the analysis of four pharmaceutical companies' stocks.

The raw values considered from the data include (that are acquired at the date of entry) the closing price, the volume, etc. Based on these values, the remaining technical indicators (used as features for the learning algorithms) are determined. The data sets do not contain categorical and ordinal variables: all the feature values are continuous. The trends generally observed among most features is non-linear. This makes tree-based classifiers an attractive suite of algorithms for exploration.

4 Methodology and Analysis

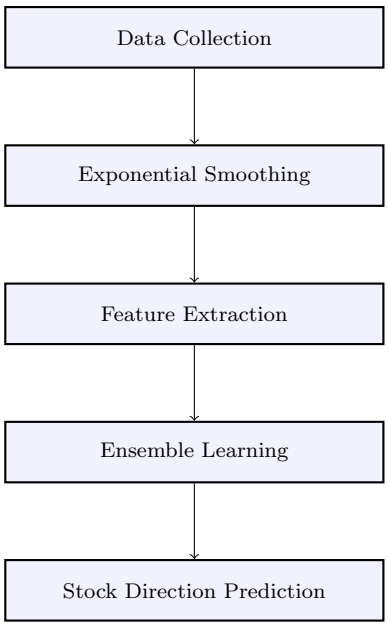

Fig. 1: Framework of supervised learning in the current work

In our experiments, the time series data acquired is first exponentially smoothed. Then the technical indicators are extracted. Technical indicators provide insights to the expected stock price behavior in future. These technical indicators are then used as features to train the classifiers. The indicators used in the current work will be discussed in this section.

4.1 Data Preprocessing

Exponential smoothing grants larger weights to the recent observations and exponentially decreases weights of the past observations. The exponentially smoothed statistic of a series Y can be recursively calculated as:

$$\begin{aligned} S_0 &= Y_0 \\ \text{for } t > 0,\ S_t &= \alpha * Y_t + (1 - \alpha) * S_{t-1} \end{aligned} \tag{1}$$

where α is the smoothing factor and $0 < \alpha < 1$. Larger values of α reduce the level of smoothing. When $\alpha = 1$, the smoothed statistic becomes equal to the actual observation. The smoothed statistic S_t can be calculated as soon as consecutive observations are available. This smoothing removes random variation or noise from the historical data, allowing the model to easily identify the long-term price trend in the stock price behavior. Technical indicators are then calculated from the exponentially smoothed time series data which are later organized into a feature matrix. The target to be predicted in the i^{th} day is calculated as follows:

$$target_i = sign(close_{i+d} - close_i) \tag{2}$$

where d is the number of days after which the prediction is to be made. When the value of $target_i$ is +1, it indicates that there is a positive shift in the price after d days; -1 indicates that there is a negative shift after d days, giving us an idea of the direction of the prices for the respective stock. The $target_i$ values are assigned as labels to the i^{th} row in the feature matrix.

4.2 Feature Extraction

Stock data is usually acquired as time-series data. In our solution, we consider only the closing price of a stock and we collect these values for many years. Hence, our solution can be considered to be of the form $(date, price_{closing})$. A short-term plot of this data should generate a graph such as this:

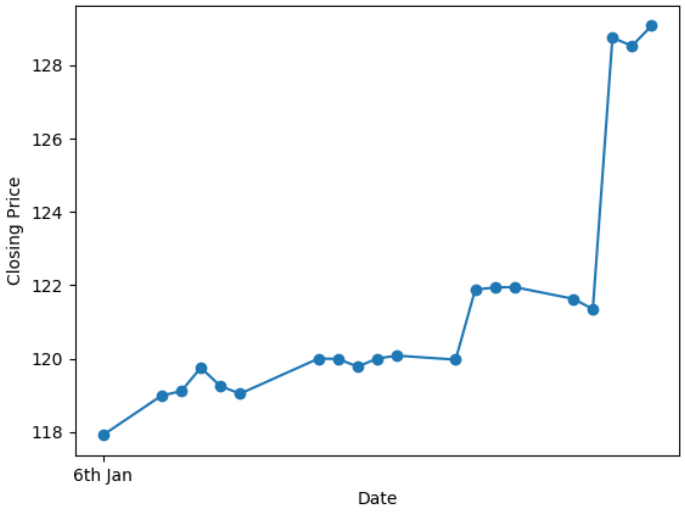

Fig. 2: A sample plot of a few days of a stock (AAPL), representative of the fluctuations that are observed.

A keen observer might visually inspect this graph and make an intelligent prediction regarding the closing price on a particular day to rise or fall below what was observed the day before. However, such methods of subjective examination requires a lot of experience in order to predict if an investor can profit by investing in a particular stock and hence the interest to automate the prediction of the outcome of an investment is considerably appealing. When this data is fed into a computer, it needs to be filtered and preprocessed through a series of stages in order to be suitable for analysis. As we are dealing with time-series data, this preprocessing eventually leads to an extraction of useful features.

A *feature* in the data can be any useful characteristic that can be used to correctly identify the class a data sample belongs to (when we are performing a classification). As an example, consider the closing prices of an arbitrary stock as shown in Table 1. Here, we calculate the *relative strength index* (RSI) (Section 4.2.1) as a feature.

Using the sample closing prices in Table 1, one can show that the monetary gain or loss of day n is calculated with respect to day $n-1$. For example, there is a loss on day 6, where the closing price is 53.0, since the closing price of day 5 is 54.0. There is neither a gain nor a loss on day 1 as it is the first day under consideration. As we are calculating RSI over a 14-day period, the average gain and average loss for day 15 is calculated over the period of days 1-14. Similarly, the average gain and average loss for day 16 is calculated over the period of days 2-15, and so on. Accordingly, the RS and RSI are calculated for days 15, 16, ..., till the last day for which the closing price is observed.

Hence, in this example, RSI is considered as a characteristic of the data and we calculate it for day 15 onwards. This process of determining and calculating relevant and important characteristics is known as *feature extraction*. Since the acquired data is time-series in nature, feature extraction gives us a matrix of the important characteristics as defined by us. In cases where tabulated data already exists (with a lot of features, some of which are noisy), algorithms like principal component analysis (PCA), independent component analysis (ICA), factor analysis, etc. helps to further reduce the number of important features for analysis or classification.

Day	Price ($)	Gain ($)	Loss ($)	Average $Gain_{14}$ ($)	Average $Loss_{14}$ ($)	RS_{14}	RSI_{14}
1	50.0	-	-	-	-	-	-
2	52.0	2	-	-	-	-	-
3	51.0	-	1	-	-	-	-
4	55.0	4	-	-	-	-	-
5	54.0	-	1	-	-	-	-
6	53.0	-	1	-	-	-	-
7	56.0	3	-	-	-	-	-
8	57.0	1	-	-	-	-	-
9	56.0	-	1	-	-	-	-
10	57.0	1	-	-	-	-	-
11	55.0	-	2	-	-	-	-
12	58.0	3	-	-	-	-	-
13	56.0	-	2	-	-	-	-
14	59.0	3	-	-	-	-	-
15	58.0	-	1	1.21	0.57	2.12	67.95
16	60.0	2	-	1.21	0.64	1.89	65.38

Table 1: Sample Closing Prices

Feature selection is how a certain classification algorithm handles data which results in the best classification. The *Gini impurity criteria* is a method used to select the best features for classification (see Section 5 for details on the use of Gini impurity criteria) in decision trees and in forests of trees.

4.2.1 Relative Strength Index (RSI)

RSI (Wilder Jr 1978) is a popular momentum indicator which determines whether the stock is over-purchased or over-sold. A stock is said to be overbought when the demand unjustifiably pushes the price upwards. This condition is generally interpreted as a sign that the stock is overvalued and the price is likely to go down. A stock is said to be oversold when the price goes down sharply to a level below its true value. This is a result caused due to panic selling. RSI ranges from 0 to 100 and generally, when RSI is above 70, it may indicate that the stock is overbought and when RSI is below 30, it may indicate the stock is oversold.

The formula for calculating RSI is:

$$RSI = 100 - \frac{100}{1 + RS} \tag{3}$$

$$RS = \frac{Average\ Gain\ Over\ past\ 14\ days}{Average\ Loss\ Over\ past\ 14\ days} \tag{4}$$

4.2.2 Stochastic Oscillator

Stochastic Oscillator (Lane 1984) follows the momentum of the price. As a rule, momentum changes before the price changes. It measures the level of the closing price relative to low-high range over a period of time.

The formula for calculating Stochastic Oscillator is:

$$\%K = 100 \times \frac{(C - L_{14})}{(H_{14} - L_{14})} \tag{5}$$

where,

C = current closing price
L_{14} = lowest price over the past 14 days
H_{14} = highest price over the past 14 days

4.2.3 Williams Percentage Range

Williams Percentage Range or Williams %R is another momentum indicator, similar in idea to stochastic oscillator. The Williams %R indicates the level of a market's closing price in relation to the highest price for the look-back period, which is 14 days. It's value ranges from -100 to 0. When its value is above -20, it indicates a *sell signal* and when its value is below -80, it indicates a *buy signal*.

Williams %R is calculated as follows:

$$\%R = -100 \times \frac{(H_{14} - C)}{(H_{14} - L_{14})} \tag{6}$$

where,

C = current closing price
L_{14} = lowest price over the past 14 days
H_{14} = highest price over the past 14 days

4.2.4 Moving Average Convergence Divergence

The moving average convergence-divergence (MACD) (Appel 2005) is a momentum indicator which compares two moving averages of prices. The first moving average is a 26-day exponential moving average (EMA) and the second moving average is a 12-day EMA. The 26-day EMA is subtracted from the 12-day EMA. A 9-day EMA of the MACD is considered as the *signal line*, which serves as the threshold for the *buy* or *sell* signals.

The formula for calculating MACD is:

$$MACD = EMA_{12}(C) - EMA_{26}(C) \tag{7}$$

$$SignalLine = EMA_9(MACD) \tag{8}$$

where,

C = closing price
EMA_n = n-day exponential moving average

When the MACD goes below the signal line, it indicates a sell signal. When it goes above the signal line, it indicates a buy signal.

4.2.5 Price Rate of Change

The Price Rate of Change (PROC) is a technical indicator which reflects the percentage change in price between the current price and the price over the window that we consider to be the time period of observation.

It is calculated as follows:

$$PROC_t = \frac{C_t - C_{t-n}}{C_{t-n}} \tag{9}$$

where,

$PROC_t$ = price rate of change at time t
C_t = closing price at time t

4.2.6 On Balance Volume

On balance volume (OBV) (Granville 1976) utilizes changes in volume to estimate changes in stock prices. This technical indicator is used to find buying and selling trends of a stock, by considering the cumulative volume: it cumulatively adds the volumes on days when the prices go up, and subtracts the volume on the days when prices go down, compared to the prices of the previous day. The formula for calculating OBV is:

$$OBV(t) = \begin{cases} OBV(t-1) + Vol(t) & \text{if } C(t) > C(t-1) \\ OBV(t-1) - Vol(t) & \text{if } C(t) < C(t-1) \\ OBV(t-1) & \text{if } C(t) = C(t-1) \end{cases} \tag{10}$$

where,

$OBV(t)$ = on balance volume at time t
$Vol(t)$ = trading volume at time t
$C(t)$ = closing price at time t

5 Random Forest

Decision trees (Geurts and Louppe 2014) and random forests (Breiman 2001) are popular machine learning approaches which can be used to solve a wide range of problems in classification. The basic training principle of decision trees is the recursive partitioning of the feature space using a tree structure, where each child node is split until pure nodes, i.e nodes which contain samples of a single class, are achieved. The splitting is done by the means of a criteria which tries to maximize the purity of the child nodes relative to their respective parent nodes. As maximum purity is ensured in child nodes, subsequently, pure nodes are arrived at. These pure nodes are not split further and constitute the leaf nodes. When a decision tree is used for the classification of a test sample, it is traced all the way down to a leaf node of the tree; as the leaf nodes of a decision tree are pure, the respective test sample is assigned the class label of the training samples of leaf node it arrives at. Random forests use an ensemble of many decision trees to reduce the effects of over-fitting. In a random forest, each tree is grown on a subset of the feature space. Usually, if each sample in the data set has M features, $m = \sqrt{M}$ features are selected to grow each tree.

5.1 Decision Trees: Background

Decision trees (Geurts and Louppe 2014; Breiman 2001) can be used for various machine learning applications. But trees that are grown really deep to learn highly irregular patterns tend to over-fit the training sets. Noise in the data may cause the tree to grow in a completely unexpected manner. Random Forests overcome this problem by training multiple decision trees on different subspaces of the feature space at the cost of slightly increased bias. This means that none of the trees in the forest sees the entire training data. The data is recursively split into partitions. At a particular node, the split is done by asking a question on an attribute. The choice for the splitting criterion is based on some impurity measures such as Gini impurity or Shannon Entropy.

Gini impurity is used as the function to measure the quality of split in each node. Gini impurity at node N is given by:

$$G(N) = 1 - (P_1)^2 - (P_{-1})^2 \tag{11}$$

where P_i is the proportion of the population with class label i. Another function which can be used to judge the quality of a split is Shannon Entropy. It measures the disorder in the information content. In Decision trees, Shannon entropy is used to measure the unpredictability in the information contained in a particular node of a tree (in this context, it measures how mixed the population in a node is). The entropy in a node N can be calculated as follows:

$$S(N) = -P_1 log(P_1) - P_{-1} log(P_{-1}) \tag{12}$$

where d is number of classes considered and $P(\omega_i)$ is the proportion of the population labeled as i. Entropy is the highest when all the classes are contained in equal proportion in the node. It is the lowest when there is only one class present in a node (when the node is pure).

The obvious heuristic approach to choose the best splitting decision at a node is the one that reduces the impurity as much as possible. In order words, the best split is characterized by the highest gain in information or the highest reduction in impurity. The information gain due to a split can be calculated as follows:

$$Gain = I(N) - p_L I(N_L) - p_R I(N_R) \tag{13}$$

where $I(N)$ is the impurity measure (Gini or Shannon Entropy) of node N, $I(N_L)$ is the impurity in the left child of node N after the split and similarly, $I(N_R)$ is is the impurity in the right child of node N after the split; N_L and N_R are the left and right children of N respectively' p_L and p_R are the proportions of the samples in the left and right children nodes with respect to the parent node.

Note that Equations 11, 12 and 13 can be used when there are only two splits at each node. However, depending on the algorithm used, there may be more than two splits and hence these formulae may be generalized as:

$$\begin{aligned} G(N) &= 1 - \sum_{j=1}^{n} P_j \\ S(N) &= -\sum_{j=1}^{n} P_j log(P_j) \\ Gain &= I(N) - \sum_{j=1}^{n} p_j I(N_j) \end{aligned} \tag{14}$$

However, in our work, we have used the CART (Classification and Regression Trees) algorithm, which splits every node into only two children nodes.

Bootstrap aggregating, also known as *bagging*, is an ensemble learning method that improves the stability and accuracy of learning algorithms while reducing variance and over-fitting, which are common problems while constructing decision trees. Given a sample dataset D of shape $n \times m$, bagging generates B new sets of shape $n' \times m'$ by sampling uniformly from D with replacement. With this knowledge, we can now summarize the algorithm of random forest classifier as follows:

Some of the main advantages of using random forests of CART are:

1. Tree based learning methods are *non-metric*. This means that there are no inherent assumptions of distributions in data, and neither are there any parameters which need to be tweaked in order to obtain optimal performance.
2. Random Forests are recommended to be used with *bagging*, i.e., test data is sampled with replacement. Bagging ensures that with the increase in the number of tree estimators in a random forest, the chance of error decreases. A simple assertion of this is made (using

S.Id	Class Label	RSI	SCH	W%R	MACD	PROC60	OBV
1	-1	91.02	74.0	-54.07	1.12	0.03	259336600.0
2	1	11.09	17.12	-35.42	-0.8	-0.05	23149700.0
3	1	75.95	77.76	-54.6	0.64	-0.02	263929400.0
4	-1	88.25	85.8	-50.08	1.02	0.01	369776000.0
5	-1	16.71	14.19	-35.08	-0.67	0.04	1434446700.0
6	1	9.55	11.48	-35.28	-1.5	-0.02	1264042100.0
7	1	66.21	77.38	-48.78	-0.23	-0.02	198021100.0
8	-1	67.82	77.98	-50.25	0.36	0.02	276163600.0
9	-1	71.88	49.71	-50.82	0.58	0.08	560679400.0
10	1	4.66	11.6	-44.17	-1.56	-0.06	135643900.0
11	1	1.52	16.01	45.27	-0.33	-0.13	-2263609600.0
12	-1	17.41	24.08	-44.97	-0.12	0.02	-3417917300.0
13	1	22.27	31.57	-40.62	-0.22	-0.03	-3265559400.0
14	-1	70.79	77.25	-45.93	0.09	0.02	-2846701500.0
15	1	77.74	84.71	-72.5	0.36	-0.08	-3516514200.0
16	-1	57.5	73.5	-44.46	-0.12	0.09	-2926120100.0
17	1	42.81	54.22	-52.63	-0.26	-0.02	-3213555400.0
18	1	61.83	72.97	-66.66	-0.09	-0.1	-3904360800.0
19	-1	18.86	16.0	-36.27	-0.13	0.03	-3645587700.0
20	-1	63.08	52.02	-50.11	-0.03	0.02	-2182828500.0

Table 2: Sample Training Set

the binomial distribution stuff) and is proved using Chebyshev's Inequality in (enter section number).

3. CART can handle categorical and continuous values naturally.
4. It is easy to understand and quick to fit, even if the data is cluttered and/or if the problem is inherently large.
5. The accuracy of random forests compares well with traditional methods in ML, such as Naïve Bayes', etc.
6. Random forests are *stable*. A slight change in the input data may affect individual trees, but the characteristics of the forest remains largely unchanged.

5.2 An Example

In random forests, decision tree learners are constructed by randomly selecting m out of M features and n out of N samples. Here, we illustrate the working of random forests by randomly considering 20 samples from the data set as the training set and 5 samples as the test set; the training and test sets are mutually exclusive. They are shown in Tables 2 and 3 respectively. Let us take $m = \sqrt{M}$, so that in each tree constructed, $\sqrt{M}$ features are randomly considered. In the data set, since there are 6 features, $\sqrt{M} = \sqrt{6} \approx 2.45$. Hence, in each tree, 2 features will be considered. We shall take the subsample size to be equal to the sample size, i.e., 20 in each tree constructed here.

The best *split* for a node in a decision tree is determined by sorting the samples in a node with respect to a feature (considered by the respective tree) and then partitioning it into two parts: the left and right children of that node. The partitioning is done by selecting the best threshold (hence, the sorting) for the corresponding feature and calculating the Gini gain (insert equation number). For example, let us consider a tree constructed by using the first two features in the training set, RSI and Stochastic Oscillator (SCH), the training set as seen by the decision tree, after being sorted based on RSI, looks as shown in Table 4.

S.Id	Class Label	RSI	SCH	W%R	MACD	PROC60	OBV
1	-1	24.97	15.54	-43.74	-0.57	0.15	-1373395100.0
2	-1	85.06	75.77	-59.75	0.37	0.08	-1711049700.0
3	-1	86.88	70.44	-66.76	0.78	0.03	-1910292300.0
4	1	0.69	13.22	-36.48	-1.35	-0.19	-2573216900.0
5	1	78.9	80.95	-47.66	1.71	-0.01	-2371874000.0

Table 3: Sample Test Set

S.Id	Class Label	RSI	SCH
11.0	1	1.52	16.01
10.0	1	4.66	11.6
6.0	1	9.55	11.48
2.0	1	11.09	17.12
5.0	-1	16.71	14.19
12.0	-1	17.41	24.08
19.0	-1	18.86	16.0
13.0	1	22.27	31.57
17.0	1	42.81	54.22
16.0	-1	57.5	73.5
18.0	1	61.83	72.97
20.0	-1	63.08	52.02
7.0	1	66.21	77.38
8.0	-1	67.82	77.98
14.0	-1	70.79	77.25
9.0	-1	71.88	49.71
3.0	1	75.95	77.76
15.0	1	77.74	84.71
4.0	-1	88.25	85.8
1.0	-1	91.02	74.0

Table 4: Training set sorted in increasing order of RSI.

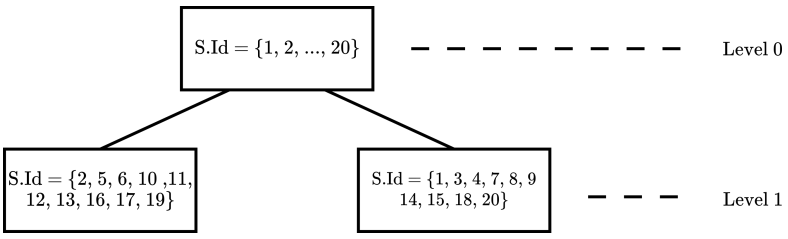

Fig. 3: Tree creation by arbitrarily using the first 10 samples from Table 4 as the left child and the last 10 samples as the right child

The root node of the decision tree will perform the first partition of the data in Table 4. Hence, the constituents of the root node of the tree (or any tree) will have all the samples in the training set with 2 features selected at random. Let us consider arbitrarily the first 10 samples in Table 4 to constitute the *left child* and the last 10 samples to constitute the *right child* of the root node. The tree, after just the first split, will look as shown in Figure 3.

Here, the impurity of the parent node (in this case, at level 0; the root node) is 0.5. This is easily understandable as there are 10 samples belonging to Class 1 and 10 samples belonging to Class -1. So the Gini Impurity is calculated as shown in Equation 15.

$$\begin{aligned} G(N) &= 1 - (P_1)^2 - (P_{-1})^2 \\ &= 1 - \left(\frac{10}{20}\right)^2 - \left(\frac{10}{20}\right)^2 \\ &= 1 - 0.25 - 0.25 \\ &= 0.5 \end{aligned} \tag{15}$$

However, as the quality of the split depends on the *GiniGain*, the Gini impurities of the left and right children also need to be calculated. They are calculated in a manner similar to Equation 15.

$$\begin{aligned} G(N_L) &= 0.48 \\ G(N_R) &= 0.48 \end{aligned} \tag{16}$$

Note that the left child has 6 samples belonging to Class 1 and 4 samples belonging to Class -1, and the right child has 6 samples belonging to Class -1 and 4 samples belonging to Class 1. Hence, in this case, $G(N_L)$ and $G(N_R)$ are equal. It is not always necessary for them to be equal! The *Gain*, hence, is calculated as in Equation 17.

$$Gain = 0.5 - \frac{10}{20} \times 0.48 - \frac{10}{20} \times 0.48 = 0.02 \tag{17}$$

This is how a node in a tree is split. At the next level of the tree, in each child node, the best split is determined again in a similar way. But now the partitions of the training sample space belonging to the children nodes are considered for partitioning each respective child node. This kind of partitioning is done recursively till *pure* nodes are reached. In decision trees, all leaf nodes need to be pure nodes (a leaf node is a node that is not split further; see A for more definitions and details). A node is said to be *pure* if all the samples in that node belong to the same class such that it does not require any further splitting. When an element needs to be classified, through a series of partitioning criteria, it reaches a leaf node and is assigned the class label of all the elements in the respective leaf node.

Returning to the illustration, it should be noted that a 50:50 split on the root node, with the first 10 samples from Table 4 constituting the left child node and the last 10 samples constituting the right child node is not the best split for that node. In fact, the best split is when the left child node has the first 4 samples and the right child node has the last 16 samples, such that the split as shown in Figure 4. For this split, the *GiniGain* is 0.125, which is greater than that for the split shown in Figure 3.

Now, how is this number 11.09, as the splitting threshold, determined? Looking at Table 4, we see that it is the value of the RSI attribute of the fourth sample. Reiterating, the samples in this table are sorted in the increasing order of RSI. As the partition till the 4^{th} sample for the left child node and the last 16 samples for the right child node results in the best GiniGain, the RSI value of the last sample comprising the left partition is used as a boundary value, or the split value (notice how the first four samples all belong to Class 1: this should help develop the readers' intuition on the working of *GiniGain* for determining the best split). Thus, the split point for the root node becomes $(RSI, 11.09)$. When an element is being classified using this tree,

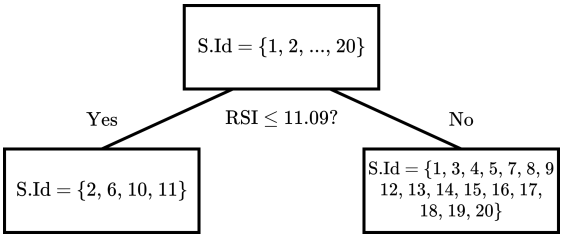

Fig. 4: The best split for the root node when considering RSI and SCH.

S.Id	RSI	SCH	Class Label	Predicted Label
1	24.97	15.54	-1	1
2	85.06	75.77	-1	-1
3	86.88	70.44	-1	-1
4	0.69	13.22	1	1
5	78.9	80.95	1	-1

Table 5: Results of classification the sample test set using **Tree 1**

it essentially needs to reach a leaf node by going through a series of splits. Hence, the first split will be done using $(RSI, 11.09)$: if the value of RSI is ≤ 11.09) for an element to be classified, it will traverse the left branch of the root node (which, in this case, is a pure node), and if the value of RSI > 11.09, it will traverse the right branch, and consequently, a series of more splits, till it reaches a leaf node. The complete tree, built recursively such that all leaf nodes are pure nodes, is shown in Figure 5.

The geometric interpretation of a decision tree is that of partitions in the feature space corresponding to each class. Using the thresholds determined using the *Gain* for each split, the feature space can be considered to be divided into partitions or *pockets*, with each partition corresponding to one class. The partitions of the feature space for Tree 1 are shown in Figure 6. In this graph, all the partitions in blue color correspond to Class 1, and all the partitions in red color correspond to Class 2. On this graph, if we were to plot each sample in the test set (Table 3) by considering only (RSI, SCH), then the *color* of the partition a sample belonged to would indicate which class the respective sample is a part of (as the color of the partition represents the Class label)! Thus, random forests can be used to effectively handle non-linear trends in data or separability of data.

Now that a tree is constructed using the training set, the obvious task at hand is to classify the samples in the test set using this tree. Reiterating, in order to classify a test sample, we need to trace a test sample down the tree until a leaf node is reached. As leaf nodes are pure, the class of the leaf node will be assigned to the test sample being classified.

From Table 5, we can see that **Tree T1** misclassifies 2 out of 5 test samples, samples 1 and 5. Hence, it can be said that **T1** has a classification accuracy of 60%. Generally, a 60% accuracy for a classifier is not considered to be good enough. So in order to have a complete understanding of how a random forest might work, we shall construct two more trees and consider the majority vote for each test sample.

Let us construct the next tree in the forest by considering the features W%R and SCH. Notice that we're resampling the values of SCH. This is the principle of bootstrap aggregation, more commonly known as *bagging*: to sample data with replacement. Bagging has certain advantages. While growing a tree, a feature may or may not be chosen to be a feature using which the tree

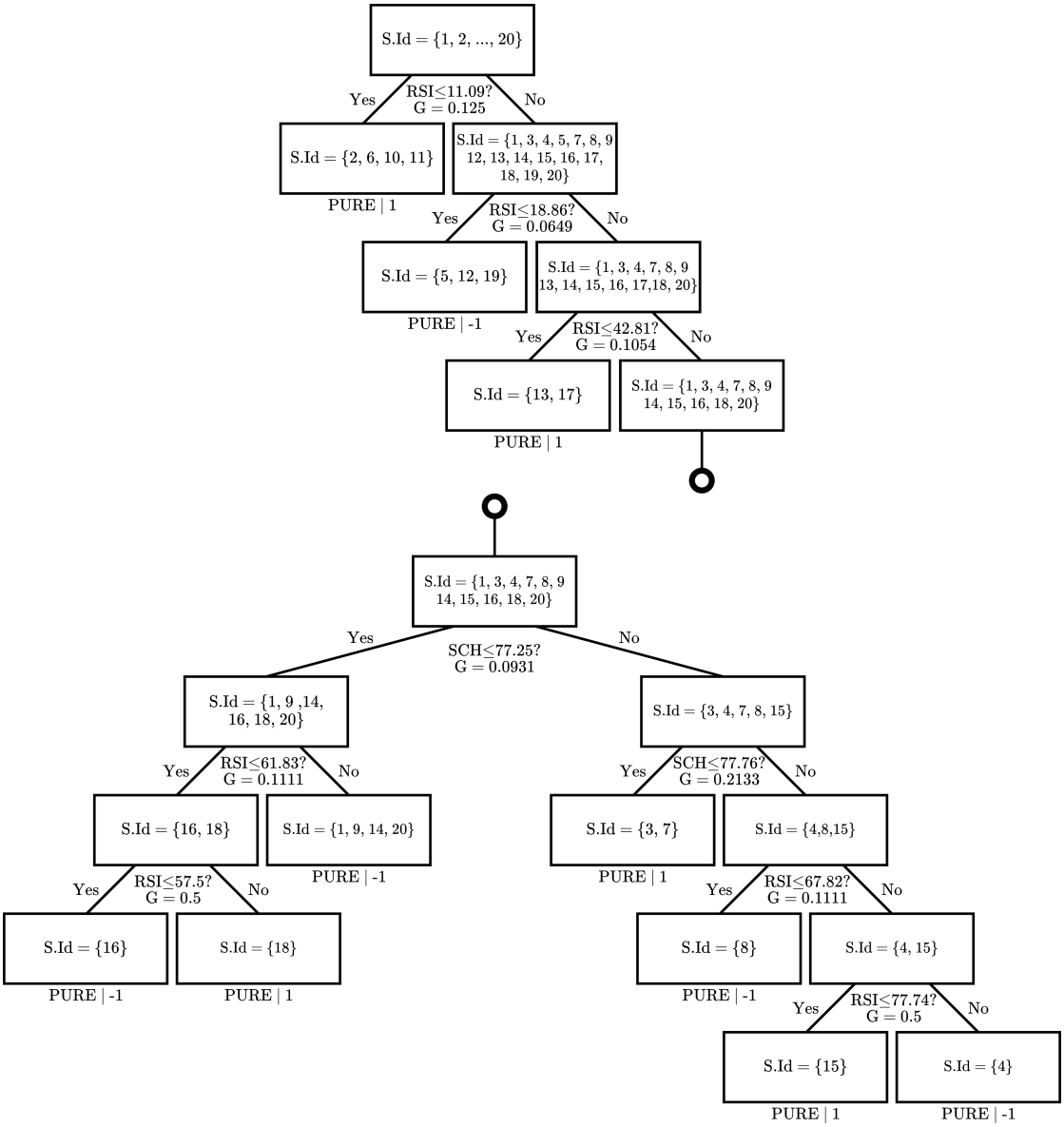

Fig. 5: **Tree 1**: Random tree constructed considering RSI and SCH

grows. This is especially relevant for high-dimensional datasets. Since each feature has a certain probability of success or failure of being chosen when a tree is being grown, it can be said that the probability of a feature being used in a random forest depends on the number of trees being constructed, with the probability being *binomially distributed*.

In the dataset used for the current work, as we have already mentioned, there are 6 features and for each tree being constructed, 2 features are considered. This implies that the probability of *success* (success being the condition that a feature is chosen to grow a decision tree) is $P(success) = 2/6 = 1/3 = 0.33$. The probability of *failure* (failure being the condition that a

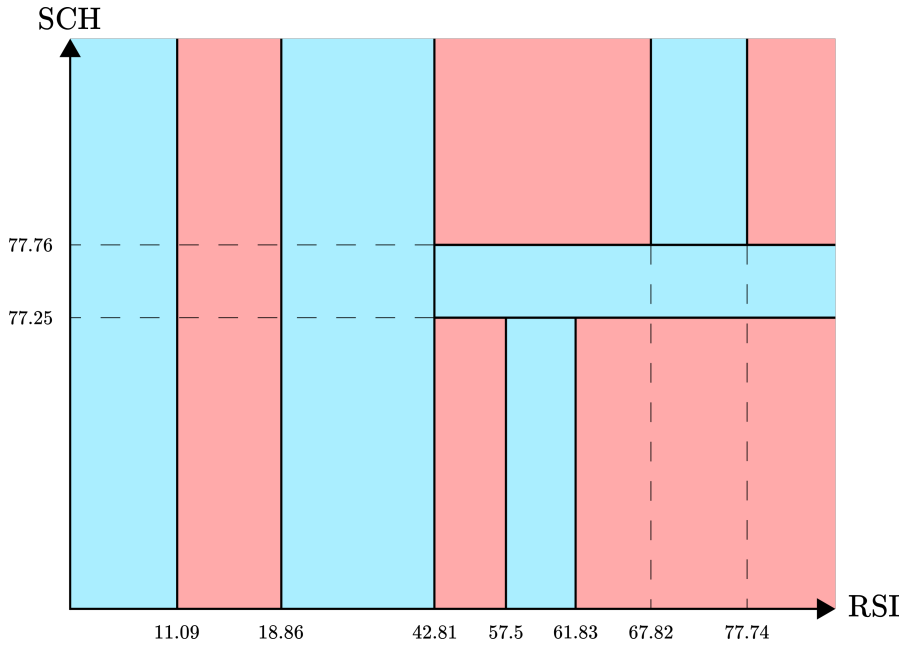

Fig. 6: Partitions of sample space of Tree 1 (graph not drawn to scale)

S.Id	W%R	SCH	Class Label	Predicted Label
1	-43.74	15.54	-1	-1
2	-59.75	75.77	-1	1
3	-66.76	70.44	-1	1
4	-36.48	13.22	1	-1
5	-47.66	80.95	1	1

Table 6: Results of classification the sample test set using **Tree 2**

feature is not chosen to grow a decision tree) is $P(failure) = 1 - P(success) = 0.67$. Let us represent the probability of success as p and the probability of failure as q. Then the probability that a feature x_a is used in growing k out of n trees is given by Equation 18.

$$P(k\ successes) = \binom{n}{k} p^k q^{n-k} \tag{18}$$

It is easy to see that the LHS of Equation 18 increases as the value of n increases. This can be used to understand the idea of *stability* of a random forest: that upon increasing the number of trees in a random forest, the expected results are stabilized by continuous re-sampling of data so that the forest as a whole is representative of the class to which a sample to be classified belongs.

From Table 6, we can see that 2 out of 5 samples are classified correctly. This is performance, as an individual tree, is not good. Upon comparing the predicted results from **Tree 1** and **Tree 2**, we see that the results from both trees combined are accurate only 50% of the time. The system as a whole is not accurate enough to predict the class of a previously unobserved sample accurately. To move a step closer towards an accurate system, let us construct another tree. We shall construct **Tree 3** using SCH and PROC60.

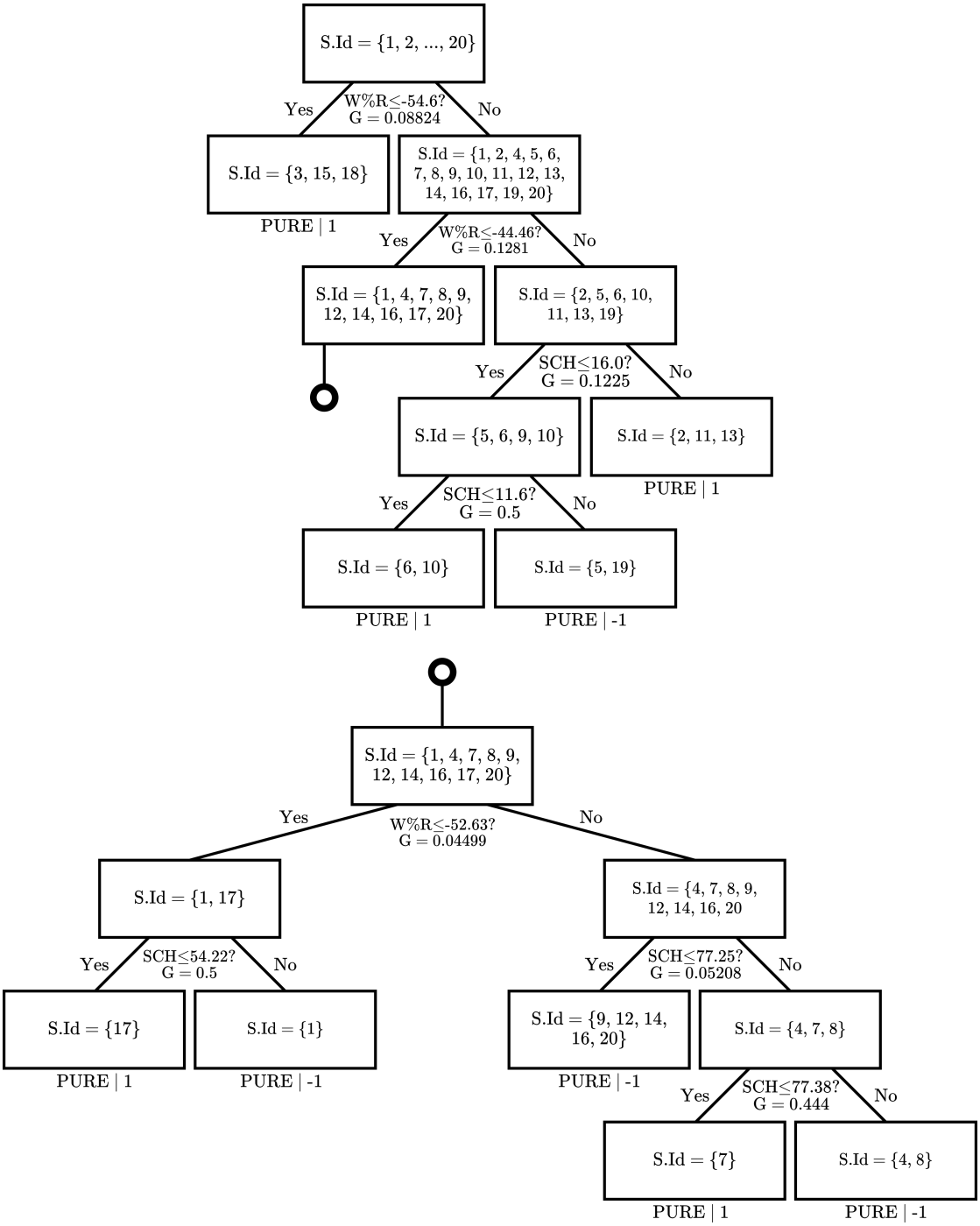

Fig. 7: **Tree 2**: Random tree constructed considering W%R and SCH

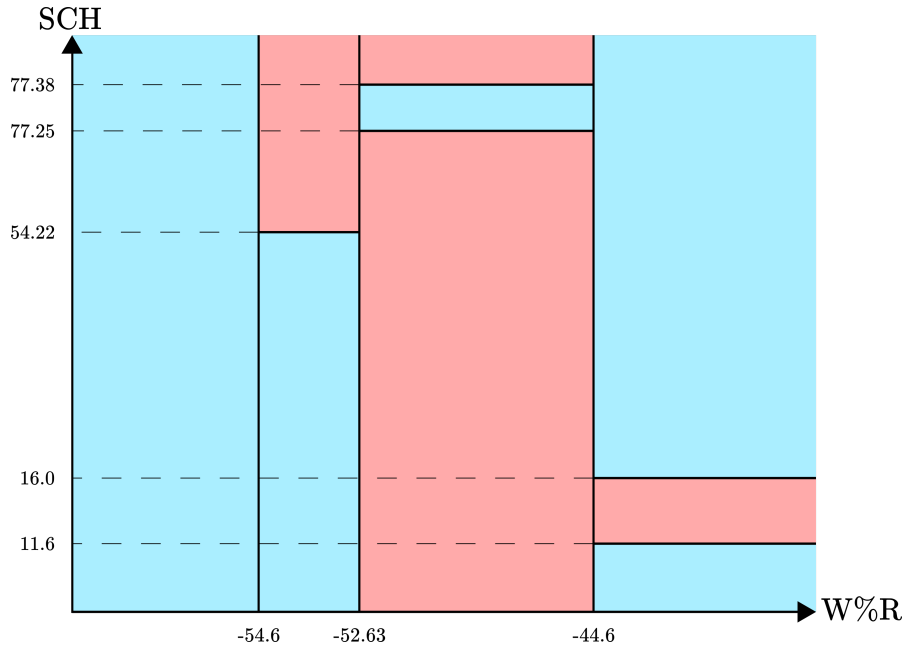

Fig. 8: Partitions of sample space of Tree 2 (graph not drawn to scale)

S.Id	SCH	PROC60	Class Label	Predicted Label
1	15.54	0.15	-1	-1
2	75.77	0.18	-1	-1
3	70.44	0.03	-1	-1
4	13.22	-0.19	1	1
5	80.95	-0.10	1	1

Table 7: Results of classification the sample test set using **Tree 3**

The decision tree constructed using SCH and PROC60 is a rather interesting one: there's only one split in the tree and the children of the root nodes are pure! The results of classification using just this tree is given in Table 7. This single tree gives us a 100% classification accuracy! The question arises: *why not use just this tree to classify the sample?*. The reasons go back to the notions of stability: that if there's some change in the data, some of the trees might not perform well, but the results of the forest should largely remain unchanged. In general, a forest classifier can be used to reduce the effect of muddy data. As this is an illustrative example, we are dealing with only 20 samples. However, the trend in millions of samples may not be as easy to deal with a single threshold. Besides, more recent innovations in tree based classifiers (such as AdaBoost, XGBoost) can discern or decide how to grow trees based on the performance of previously grown trees in the forest.

Now that there are three trees in our forest, we can perform a majority voting. The result of the entire tree using **Tree 1**, **Tree 2** and **Tree 3** is given in Table 8. By combining the performance of all the trees in the forest, the results are perfect, and also stable.

The algorithm used to build decision trees is described in Algorithms 1 and 2.

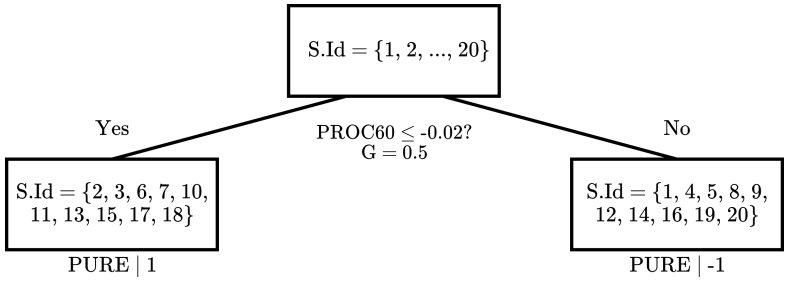

Fig. 9: **Tree 3**: Random tree constructed considering SCH and PROC60

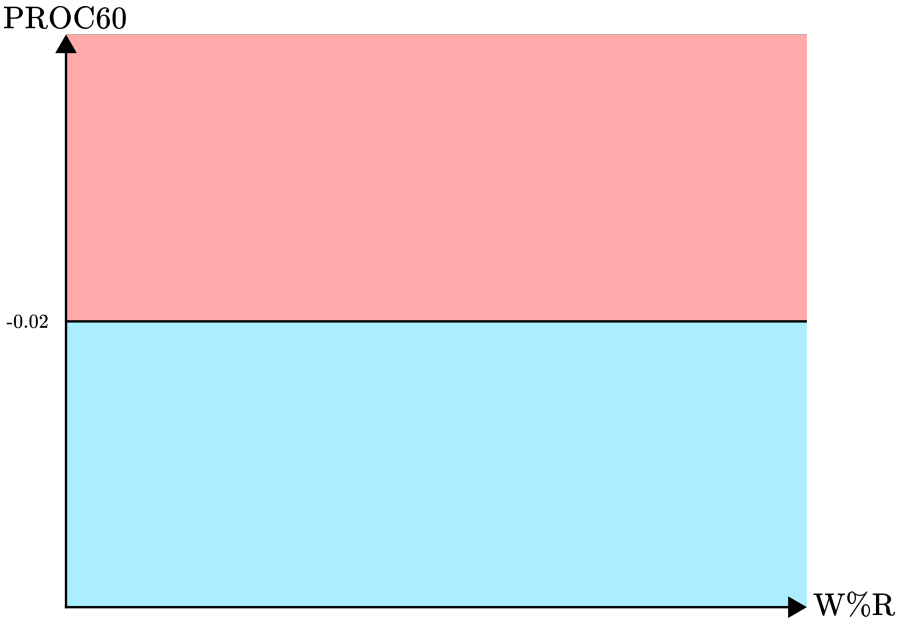

Fig. 10: Partitions of sample space of Tree 3 (graph not drawn to scale)

S.Id	Class Label	Tree 1	Tree 2	Tree 3	Predicted Label
1	-1	1	-1	-1	-1
2	-1	-1	1	-1	-1
3	-1	-1	1	-1	-1
4	1	1	-1	1	1
5	1	-1	1	1	1

Table 8: Results of classification of the entire forest using majority voting

Algorithm 1: DecisionTree

input : $X = (x_i, y_i)_1^n$ is the labeled training data
l is the current level of the tree
M is the set of features used to grow a tree

output: A tree which is configured to predict the class label of a test sample

---

$l \leftarrow l + 1$;
$C_L \leftarrow$ null;
$C_R \leftarrow$ null;
; /* C_L, C_R are the left and right children of this node respectively */
$MaxGain \leftarrow 0$;
; /* $MaxGain$ stores the value of the maximum possible gain that can be achieved from splitting a node - based on this, the final split is determined */
for j *in* M **do**
  Sort $(x_i, y_i)_1^n$ in the increasing order of the j^{th} feature
  for $i := 1$ *to* n **do**
    $c_L \leftarrow X[0 : i]$;
    $c_R \leftarrow X[i + 1 : n]$;
    $Gain \leftarrow$ **GiniGain**$((x_i, y_i)_1^n[j], c_L, c_R)$;
    if $Gain > MaxGain$ **then**
      $MaxGain \leftarrow Gain$;
      $C_L, C_R \leftarrow c_L, c_R$;
  end
end
if C_L ***does not satisfy the desired level of purity*** **then**
  DecisionTree(C_L, l, M);
if C_R ***does not satisfy the desired level of purity*** **then**
  DecisionTree(C_R, l, M);
; /* A node is not pure when it has samples belonging to more than one class. In such a case, it needs to be split in turn. In this way, through successive function calls, the feature space is recursively partitioned. */

Algorithm 2: GiniGain

input : $(x_i, y_i)_1^n$ is the labeled training data
c_L is the left child node
c_R is the right child node

output: The Gini gain of the current split

$G_N \leftarrow$ Gini impurity of root node;
$G_L \leftarrow$ Gini impurity of left child;
$G_R \leftarrow$ Gini impurity of right child;
$L_n \leftarrow$ number of samples in c_L;
$R_n \leftarrow$ number of samples in c_R;
$P_L \leftarrow L_n/n$;
$P_R \leftarrow R_n/n$;
return $G_N - (P_L \times G_L) - (P_R \times G_R)$;

5.3 OOB error and Convergence of the Random Forest

Given an ensemble of decision trees $h_1(X), h_2(x), h_3(x), ..., h_k(x)$, as in (Breiman 2001) we define margin function as

$$mg(X, Y) = av_k I(h_k(X) = Y) - max_{j \neq Y} av_k I(h_k(X) = j) \tag{19}$$

where X, Y are randomly distributed vectors from which the training set is drawn. Here, $I(.)$ is the indicator function. The generalization error is given by:

$$PE^* = P_{X,Y}(mg(X,Y) < 0) \tag{20}$$

The X and Y subscripts indicate that probability is calculated over X, Y space. In random forests, the k^{th} decision tree $h_k(x)$ can be represented as $h(x, \theta_k)$ where x is the input vector and θ_k is the bootstrapped dataset which is used to train the k^{th} tree. For a sequence of bootstrapped sample sets $\theta_1, \theta_2, ..., \theta_k$ which are generated from the original dataset θ, it is found that PE^* converges to:

$$P_{X,Y}(P_\theta(h(X,\theta) = Y) - max_{j \neq Y} P_\theta(h(X,) = j) < 0) \tag{21}$$

The proof can be found in Appendix I in (Breiman 2001). To practically prove this theorem with respect to our dataset, the generalization error is estimated using out of bags estimates (Bylander and Hanzlik 1999). The out of Bag (OOB) error measures the prediction error of Random forests algorithm and other machine learning algorithms which are based on Bootstrap aggregation.
Note: The average margin of the ensemble of classifiers is the extent to which the average vote count for the correct class flag exceeds the count for the next best class flag.

5.4 Random Forest as Ensembles: An Analytical Exploration

As defined earlier, a Random Forest model specifies θ as classification tree marker for $h(X|\theta)$ and a fixed probability distribution for θ for diversity determination in trees is known.

The margin function of an RF is:

$$margin_{RF}(x,y) = P_\theta(h(x|\theta) = y) - max_{j \neq y} P_\theta(h(x|\theta) = j) \tag{22}$$

The strength of the forest is defined as the expected value of the margin:

$$s = E_{x,y}(margin_{RF}(x,y)) \tag{23}$$

The generalization error is bounded above by Chebyshev's inequality and is given as:

$$Error = P_{x,y}(margin_{RF}(x,y) < 0) \leq P_{x,y}(|margin_{RF}(x,y) - s| \geq s) \leq \frac{var(margin_{RF}(x,y))}{s^2} \tag{24}$$

Remark: We know that the average margin of the ensemble of classifiers is the extent to which the average vote count for the correct class flag exceeds the count for the next best class flag. The strength of the forest is the expected value of this margin. When the margin function gives a negative value, it means that an error has been made in classification. The generalization error is the probability that the margin is a negative value. Since margin itself is a random variable, equation (20) shows that it is bounded above by its variance divided by the square of the threshold. As the strength of the forest grows, error in classification decreases.

We present below, the Chebyshev's inequality as the inspiration for the error bound.

5.5 Chebyshev's Inequality

Let X be any random variable (not necessarily non-negative) and C > 0. Then,

$$P(|X - E(X)| \geq c) \leq \frac{var(x)}{c^2} \tag{25}$$

It is easy to relate the inequality to the error bound of the Random Forest learner.

5.6 Proof of Chebyshev's Inequality:

We require a few definitions before the formal proof.

A) Indicator Random Variable

$$I(X \geq c) = \begin{cases} 1 & \text{if } X \geq c \\ 0 & \text{Otherwise} \end{cases} \tag{26}$$

B) Measurable space

$$A = \{x \in \Omega | X(x) \geq c\} \tag{27}$$

$$E(X) = \sum_{x \in A} P(x)X(x) = \mu \tag{28}$$

Proof:

Define $A = \{x \in \Omega | X(x) - E(x) \geq c\}$

Thus,

$$\begin{aligned} var(X) &= \sum_{x \in \Omega} P(X = x)(X(x) - E(x))^2 \\ &= \sum_{x \in A} P(X = x)(X(x) - E(x))^2 + \sum_{x \notin A} P(X = x)(X(x) - E(x))^2 \geq 0 \\ &\geq \sum_{x \in A} P(x = x)(X(x) - E(x))^2 \\ &\geq P(X = x)c^2 \quad \text{since,} X(x) - E(x) \geq C; \forall x \in A \\ &= c^2 P(A) = c^2 P(|X - E(X)| \geq c) \\ \Rightarrow \frac{var(X)}{c^2} &\geq P(|X - E(X)| \geq c) \end{aligned} \tag{29}$$

Remark: This means that the probability of the deviation of a data point from its expected value being greater than c, a threshold, is bounded above by the variance of the data points divided by the square of the threshold, c. As c increases, the upper bound decreases which implies the probability of a large deviation of a data point from its expected value is less likely.

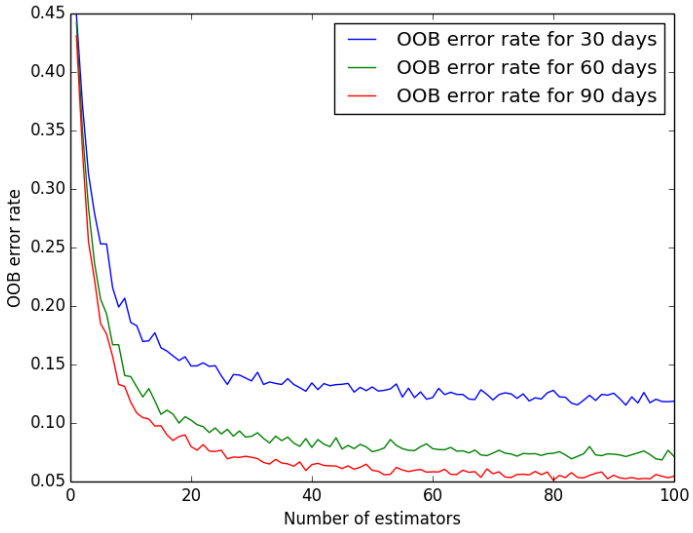

Fig. 11: OOB error rate vs Number of estimators

5.7 OOB Error Visualization

After creating all the decision trees in the forest, for each training sample $Z_i = (X_i, Y_i)$ in the original training set T, we select all bagged sets T_k which does not contain Z_i. This set contains bootstrap datasets which do not contain a particular training sample from the original training dataset. These sets are called out of bags examples. There are n such sets for each n data samples in the original training dataset. OOB error is the average error for each Z_i calculated using predictions from the trees that do not contain z_i in their respective bootstrap sample. OOB error is an estimate of generalization error which measures how accurately the random forest predicts previously unseen data. We plotted the OOB error rate for our random forest classifier using the AAPL dataset.

From Figure 11 OOB plot, we can see that the OOB error rate decreases dramatically as more number of trees are added in the forest. However, a limiting value of the OOB error rate is reached eventually. The plot shows that the Random Forest converges as more number of trees are added in the forest. This result also explains why random forests do not over-fit as more number of trees are added into the ensemble.

$L(y, \gamma)$ is the differentiable loss function and $h_m(x)$ is the base learner, connected by the following relation: $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$.

6 eXtreme Gradient Boosting (XGBoost)

6.1 Boosting Approaches

Boosting a classifier means combining the results of many weak predictors to make a strong prediction. Boosting has many variants but all of them work by optimally selecting or building successive weak classifiers such that the prediction resulting from many weak classifiers is strong.

Gradient boosting is an improvement on decision trees, where each tree is approximated as an aggregate of many regressor functions $f_i(x)$. This is in contrast to the traditional idea of decision trees, wherein a Gini impurity based splitting is directly used to find the best split in the feature space. Each successive function f_i is built such that the misclassification rate successively decreases. This is done by trying to better classify the *residuals*, or the misclassified samples of the i^{th} iteration in the $i+1^{th}$ iteration. Hence, the error in classification successively decreases. This step-wise aggregation of functions approximates a node in a tree, and eventually, the entire tree is approximated. Once each tree has been optimally approximated, the *structure scores* and *gain* are calculated, based on which the best split is determined. There are obvious advantages of doing this: since each tree is built carefully, a lot of random trees need not be used in classifying a random sample, substantially decreasing the number of trees required in the forest of classifiers. XGBoost is a framework and library which parallelizes the growth of gradient boosted trees in a forest.

The idea of gradient boosting (Friedman 2000) comes from the principle of gradient descent: a greater number of the misclassified samples of the i^{th} learner should be classified correctly by the $i+1^{th}$ learner, and so on. This implies that the error in classification reduces as more number of regressors are constructed in each node. More specifically, in the case of gradient boosted decision trees (GBDT), the $i+1^{th}$ regression function is expected to rectify the mistakes of the i^{th} function. Since the error goes on decreasing as a tree is approximated by a larger number of functions, gradient boosting is considered to be a convex optimization problem. XGBoost (Chen and Guestrin 2016) aims to minimize the time required to grow trees. This makes GBDTs more practical to use. Here, too, a subset of the features is used to build each tree. However, in the algorithmic description of XGBoost, the set of features being used to grow the tree is deliberately excluded: the assumption is that a subset of features is drawn out and fed into the algorithm.

6.2 Gradient Boosted Decision Trees: An Analytic Exploration

The tree approximation by aggregating many functions is done by *additive learning*. Each node is hence built sequentially, with each successive approximated function trying to better classify the residuals of the previous learner. XGBoost employs an additive strategy wherein every subsequent approximated function optimizes an objective function. Hence, a series of functions are built such that the node is ultimately approximated using an aggregate of all the functions and is gradually optimized. The objective function may be represented as:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^t) + \sum_{k=1}^{t} \Omega(f_k) \tag{30}$$

where l is a loss function and Ω is the regularization term, which is used to measure the complexity of the model. Ω is of the form: $\Omega(f) = \gamma T + \lambda \|w\|^2$, thus making use of an L2 regularization.

Additive learning over t iterations for the i^{th} tree happens as follows:

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
\hat{y}_i^{(3)} &= f_1(x_i) + f_2(x_i) + f_3(x_i) = \hat{y}_i^{(2)} + f_3(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= f_1(x_i) + ... + f_n(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned} \tag{31}$$

Algorithm 3: GradientBoostedTree

input : $(x_i, y_i)_1^n$ is the labeled training data
γ is the minimum required structure score
L is the maximum number of levels in the tree
l is the current level of the tree
λ is the learning rate
N is the number of training steps

output: A tree which is configured to predict the class label of a test sample

```
l ← l + 1;
if l ≤ L then
    t ← 0;
    f ← 0;
    while t < N do
        estimate f_t as a regressor function, density, or distribution;
        f ← f + f_i;
    end
    initialize array of scores S[:];
    for j := 1 to n do
        G_j ← 0;
        H_j ← 0;
        for i := 1 to N do
            G_j ← G_j + g_ji;
            H_j ← H_j + h_ji;
            /* g_ji and h_ji are the gradient statistics of the j^th sample for the i^th function
               */
        end
        S[j] ← −0.5 · G_j^2 ÷ (H_j + λ);
    end
    C_L, C_R ← MaxGain((x_i, y_i)_1^n, S) ;    // C_L, C_R are the left and right children of this node
     respectively
    if C_L does not satisfy the desired level of purity then
        GradientBoostedTree(C_L, γ, L, l, λ, N);
    if C_L does not satisfy the desired level of purity then
        GradientBoostedTree(C_R, γ, L, l, λ, N);
```

And hence, the expression:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{32}$$

Hence, the objective function may be expanded as:

$$\begin{aligned} \mathcal{L}^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constants \end{aligned} \tag{33}$$

The constants arise because the regularization till step $t-1$ is considered to be a constant at step t. Hence, Equation 33 represents the loss function in it's general form. This loss function can be approximated using the Taylor Series approximation till the n^{th} order term. The approximation till the 2^{nd} order is considered. The Taylor Series expansion is done as follows.

Algorithm 4: MaxGain

input : $(x_i, y_i)_1^m$ is the labeled training data
$S[:]$ is the list of structure scores

output: Left and right child nodes, ensuring maximum purity after split

Sort $(x_i, y_i)_1^m$ and $S[:]$ in the increasing order of $S[:]$;
$Gain_{split} \leftarrow -\infty$ /* $Gain_{split}$ stores the value of the gain after each split */
$Gain_N \leftarrow \sum S[:]$ /* the overall score of the parent node */
$SplitPoint \leftarrow -1$ /* stores the best split point */
$f \leftarrow 0$;
for $j := 1$ *to* $m - 1$ **do**
  $N_L \leftarrow S[0 : j]$ /* list of scores of left child */
  $N_R \leftarrow S[j + 1 : m]$ /* list of scores of right child */
  $G_L \leftarrow \sum N_L$ /* overall gain of left child */
  $G_R \leftarrow \sum N_R$ /* overall gain of right child */
  $Gain \leftarrow G_L + G_R - G_N$ /* gain calculated in the j^{th} iteration */
  /* the best gain is selected iteratively */
  if $Gain > Gain_{split}$ ***and*** $Gain > \gamma$ **then**
    $Gain_{split} \leftarrow Gain$;
    $SplitPoint \leftarrow j$;
end
return $(x_i, y_i)_1^j, (x_i, y_i)_{j+1}^m$;

Let us consider a function $F(u) = l(X(u), Y(u)) = l(X + u\Delta X, Y + u\Delta Y)$. At $u = 1$, the function becomes: $F(1) = l(X + \Delta X, Y + \Delta Y)$. Hence, in the current context, at $u = 1$, the following considerations can be made: $X \equiv y_i$, $\Delta X = 0$, $Y \equiv \hat{y}_i^{(t-1)}$, $\Delta Y \equiv f_t(x_i)$. Applying the chain rule, we can find the n^{th} order derivatives of $F(u)$. The expression for the first order derivative is derived in Equation 34.

$$
\begin{aligned}
&\frac{d}{du}\big[F(u)\big] = \frac{\partial}{\partial X}\big[l(X(u), Y(u))\big]\frac{d}{du}(X(u)) + \frac{\partial}{\partial Y}\big[l(X(u), Y(u))\big]\frac{d}{du}(Y(u)) \\
\Rightarrow &\frac{d}{du}\big[F(u)\big] = \frac{\partial}{\partial X}\big[l(X(u), Y(u))\big]\Delta X + \frac{\partial}{\partial Y}\big[l(X(u), Y(u))\big]\Delta Y \\
\Rightarrow &\frac{d}{du}\big[F(u)\big] = \frac{\partial}{\partial X}\big[l(X(u), Y(u))\big] \cdot 0 + \frac{\partial}{\partial Y}\big[l(X(u), Y(u))\big] \cdot f_t(x_i) \\
\Rightarrow &\frac{d}{du}\big[F(u)\big] = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(\hat{y}_i, \hat{y}_i^{(t-1)}) \cdot f_t(x_i) \\
&\Rightarrow F'(u) = g_i f_t(x_i)
\end{aligned}
\tag{34}
$$

where

$$
g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(\hat{y}_i, \hat{y}_i^{(t-1)})
$$

The expression for the second order derivative is derived in Equation 35:

$$
\begin{aligned}
&\frac{d^2}{du^2}\big[F(u)\big] = \left\{ \frac{\partial}{\partial X}\Big[\frac{\partial}{\partial Y}\big[l(X(u),Y(u))\big]\Big] + \frac{\partial}{\partial Y}\Big[\frac{\partial}{\partial Y}\big[l(X(u),Y(u))\big]\Big] \right\} \cdot \frac{d}{du}(Y(u)) \\
\Rightarrow &\frac{d^2}{du^2}\big[F(u)\big] = \left\{ \frac{\partial^2}{\partial X \partial Y}\big[l(X(u),Y(u))\big] \cdot \frac{d}{du}(X(u)) + \frac{\partial^2}{\partial Y^2}\big[l(X(u),Y(u))\big] \cdot \frac{d}{du}(Y(u)) \right\} \cdot \frac{d}{du}(Y(u)) \\
\Rightarrow &\frac{d^2}{du^2}\big[F(u)\big] = \left\{ \frac{\partial^2}{\partial X \partial Y}\big[l(X(u),Y(u))\big] \cdot \Delta X + \frac{\partial^2}{\partial Y^2}\big[l(X(u),Y(u))\big] \cdot \Delta Y \right\} \cdot \frac{d}{du}(Y(u)) \\
\Rightarrow &\frac{d^2}{du^2}\big[F(u)\big] = \left\{ \frac{\partial^2}{\partial X \partial Y}\big[l(X(u),Y(u))\big] \cdot 0 + \frac{\partial^2}{\partial Y^2}\big[l(X(u),Y(u))\big] \cdot \Delta Y \right\} \cdot \Delta Y \\
\Rightarrow &\frac{d^2}{du^2}\big[F(u)\big] = \left\{ \frac{\partial^2}{\partial Y^2}\big[l(X(u),Y(u))\big] \cdot \Delta Y \right\} \cdot \Delta Y \\
\Rightarrow &\frac{d^2}{du^2}\big[F(u)\big] = \frac{\partial^2}{\partial \big(\hat{y}_i^{(t-1)}\big)^2} l(y_i, \hat{y}_i^{(t-1)}) \cdot \Delta Y \cdot \Delta Y \\
\Rightarrow &\frac{d^2}{du^2}\big[F(u)\big] = \frac{\partial^2}{\partial \big(\hat{y}_i^{(t-1)}\big)^2} l(y_i, \hat{y}_i^{(t-1)}) \cdot f_t^2(x_i) \\
&\Rightarrow F''(u) = h_i f_t^2(x_i)
\end{aligned}
\tag{35}
$$

where

$$
h_i = \frac{\partial^2}{\partial \big(\hat{y}_i^{(t-1)}\big)^2} l(y_i, \hat{y}_i^{(t-1)})
$$

The Taylor series approximation till the second order is then given as:

$$
\begin{aligned}
l\big[y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\big] &= F(0) + F'(0) + \frac{1}{2}F''(0) \\
&= l(\hat{y}_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)
\end{aligned}
\tag{36}
$$

Thus the objective function from Equation 30 can be approximated as:

$$
\begin{aligned}
\mathcal{L}^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k) \\
&= \sum_{i=1}^{n} \big[l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))\big] + \Omega(f_t) + constants \\
&= \sum_{i=1}^{n} \big[l(\hat{y}_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\big] + \Omega(f_t) + constants
\end{aligned}
\tag{37}
$$

As constants are not required in optimization, all the constant terms (note that the term $l(\hat{y}_i, \hat{y}_i^{(t-1)})$ can be considered to be constant in the t^{th} iteration) may be removed, thus consolidating the objective function further to:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{38}$$

The optimization, hence, only depends on g_i and h_i. XGBoost can support any loss function by taking g_i and h_i as input. These are known as the *gradient statistics* of the structure.

Each tree $f(x)$ is defined as:

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \to 1, 2, ..., T \tag{39}$$

Here, w is the vector of scores on leaves q is a function assigning each data point to the corresponding leaf T is the number of leaves.

The regularization function Ω is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \tag{40}$$

Here, γ is the minimum gain a branch needs to contribute to the entire learner in order to be added to the overall structure. The idea of γ becomes clearer later in the derivation. There are multiple ways of defining the regularization function, but the one used here is an L2 regularization.

From equations (previous two), the objective value with the t^{th} iteration is as follows:

$$\begin{aligned} Obj^{(t)} &\approx \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[\Big(\sum_{i \in I_j} g_i \Big) w_j + \frac{1}{2} \Big(\sum_{i \in I_j} h_i + \lambda \Big) w_j^2 \right] + \gamma T \end{aligned} \tag{41}$$

where $I_j = i | q(x_i) = j$ is the set of indices of data points assigned to the j^{th} leaf, and λ is the *learning rate*. If $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$, then Equation 41 becomes:

$$Obj^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} \big(H_j + \lambda \big) w_j^2 \right] + \gamma T \tag{42}$$

The part of Equation 42 within the summation is in a quadratic form. The optimal condition of a second degree objective function is at the point where the first derivative is zero. Hence, substituting the value of w_j at the point where the slope of the objective function is zero gives the optimal value of the objective function.

$$\begin{aligned} &\frac{d}{dw_j} \left[G_j w_j^* + \frac{1}{2} \big(H_j + \lambda \big) (w_j^2)^* \right] = 0 \\ &\Rightarrow G_j + 2 \cdot \frac{1}{2} \big(H_j + \lambda \big) w_j^* = 0 \\ &\Rightarrow \big(H_j + \lambda \big) w_j^* = -G_j \\ &\Rightarrow w_j^* = \frac{-G_j}{\big(H_j + \lambda \big)} \end{aligned} \tag{43}$$

Thus, substituting w_j^* in the objective function, we get the optimal value of the objective (from Equation 42):

$$\begin{aligned} Obj^* &= \sum_{j=1}^{T} \left[G_j \cdot \left(\frac{-G_j}{H_j + \lambda} \right) + \frac{1}{2}(H_j + \lambda) \left(\frac{-G_j}{H_j + \lambda} \right)^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[\frac{-G_j^2}{H_j + \lambda} + \frac{1}{2} \cdot \frac{G_j^2}{H_j + \lambda} \right] + \gamma T \\ &= -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \end{aligned} \tag{44}$$

Now we have an objective function which can tell us about the goodness of a tree. But how do we decide what is the best split? Analogous to the GiniGain measure in the case of random forests, the Gain in XGBoost is used to split a node. The form of the gain function is similar to GiniGain; the best split at a node results in the best Gain.

$$Gain = -\frac{1}{2} \cdot [Gain_L + Gain_R + Gain_{Root}] - \gamma \tag{45}$$

The loss function commonly used in XGBoost is the squared loss function, which is given by:

$$\mathcal{L} = \left[y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right]^2 \tag{46}$$

For which the gradient scores are calculated as:

$$\begin{aligned} g_i &= \frac{\partial}{\partial \hat{y}_i^{(t-1)}} \left[y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right]^2 \\ &= 2 \cdot \left[y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right] \cdot \left[0 - (1 + 0) \right] \\ &= -2 \left[y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right] \\ \Rightarrow h_i &= \frac{\partial}{\partial \hat{y}_i^{(t-1)}} \Big\{ -2 \left[y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right] \Big\} \\ &= -2 \left[0 - (1 + 0) \right] \\ &= 2 \end{aligned} \tag{47}$$

The algorithm of GBDTs are explained in Algorithms 3 and 4.

6.3 An Example

Consider the split made in the root node in Tree 1 (Figure 4). Utilizing the definitions laid down of g_i, h_i, Gain, λ, and γ, it can be considered from the perspective of gradient boosted trees that the left child node of the root node estimates $\hat{y}_i$ as Class 1 and the right node estimates it as -1, as the majority of the samples used for building the tree belong to classes $+1$ and -1 in the left and right children nodes, respectively. Thus, the structure score of the left branch may be calculated as (from Table 9):

From Table 9, G_j and H_j may be calculated as: $G_j = \sum g_i = 0$ and $H_j = \sum h_i = 8$. All the entities in this node truly belong to Class 1, and by the definition of g_i, the value of G_i sums

S. Id	g_i	h_i	$\hat{y}_i^{(t-1)}$	y_i
2	0	2	0	1
6	0	2	0	1
10	0	2	0	1
11	0	2	0	1

Table 9: Gradient statistics for left child node of root node of Tree 1.

S. Id	g_i	h_i	$\hat{y}_i^{(t-1)}$	y_i
1	0	2	0	-1
3	-2	2	0	-1
4	0	2	0	-1
5	0	2	0	-1
7	-2	2	0	-1
8	0	2	0	-1
9	0	2	0	-1
12	0	2	0	-1
13	-2	2	0	-1
14	0	2	0	-1
15	-2	2	0	-1
16	0	2	0	-1
17	-2	2	0	-1
18	-2	2	0	-1
19	0	2	0	-1
20	0	2	0	-1

Table 10: Gradient statistics for right child node of root node of Tree 1.

up to 0. As the root node is the first in the series, it does not classify any of the entities and hence $\hat{y}_i^{(t-1)}$ for this split is 0 (note that the class labels are $+1$ and -1, signifying an increase and decrease in prices respectively). Consider the value of learning rate, λ, to be 1, arbitrarily. The structure score can thence be calculated as:

$$obj^* = -\frac{1}{2} \cdot \frac{G_L^2}{H_L + \lambda} = -\frac{1}{2} \cdot \frac{0}{8+1} = 0 \tag{48}$$

Similarly, for the right child node, the gradient statistics may be calculated using Table 10, in the following manner:

Here, $G_j = -12$ and $H_j = 32$. The structure score then becomes:

$$obj^* = -\frac{1}{2} \cdot \frac{G_R^2}{H_R + \lambda} = -\frac{1}{2} \cdot \frac{-12^2}{32+1} = -4.36 \tag{49}$$

Next, we shall see how the Gain of the split is calculated. For this, let us consider the value of γ to be 0.3, arbitrarily:

$$\begin{aligned} Gain &= -\frac{1}{2}\cdot\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{G_{Root}^2}{H_{Root}+\lambda}\right]-\gamma \\ &= -\frac{1}{2}\cdot\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right]-\gamma \\ &= -\frac{1}{2}\cdot\left[0+\frac{144}{32+1}-\frac{(0-12)^2}{8+32+1}\right]-0.3 \\ &= 0.8514-0.3 \\ &= 0.5514 \end{aligned} \tag{50}$$

This is how the gain is calculated. In a similar fashion, the remaining levels of this tree are constructed. The parameters λ and γ can be tuned for optimal performance. Once this tree is complete, the next tree is built by bagging. Thus, with every level and every tree in sequence, the forest of trees is able to reduce the error in classification.

7 Results For Random Forest and XGBoost

Great care has been taken to collect data sets. Diversity, reputation and fiscal health in company profiles have been considered before choosing the stock data. We have considered stocks from Information Technology and enabled services, social media, Electronics and Instrumentation, Manufacturing and Pharmaceutical behemoths.

One of the essential steps include evaluating the model performance or in other words, the model should be evaluated for its robustness. The parameters that are used to evaluate the robustness of a binary classifier are accuracy, precision, recall (also known as sensitivity) and specificity. The formula to calculate these parameters are given below:

$$\begin{aligned} Accuracy &= \frac{tp+tn}{tp+tn+fp+fn} \\ Precision &= \frac{tp}{tp+fp} \\ Recall &= \frac{tp}{tp+fn} \\ Specificity &= \frac{tn}{tn+fp} \\ Fscore &= 2\cdot\frac{precision\cdot recall}{precision+recall} \end{aligned} \tag{51}$$

where,

tp = number of true positive values
tn = number of true negative values
fp = number of false positive values
fn = number of false negative values

Here, we consider the results of two stocks, for the sake of brevity, to justify the efficacy of the algorithms: those of Apple and Toyota. The entire experiment was implemented on 14 different stocks, the results of which are elaborated in B.

Table 11: Results of classification using Random Forests

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
AAPL	3	64.93	0.71	0.65	0.58	0.68
	5	72.49	0.77	0.73	0.67	0.75
	10	79.07	0.81	0.81	0.77	0.81
	15	82.39	0.86	0.83	0.79	0.84
	30	85.58	0.89	0.86	0.82	0.87
	60	90.68	0.94	0.90	0.87	0.92
	90	93.02	0.95	0.93	0.90	0.94
TYO	3	67.13	0.53	0.64	0.77	0.58
	5	72.69	0.61	0.65	0.80	0.63
	10	78.36	0.60	0.80	0.90	0.69
	15	84.91	0.73	0.85	0.92	0.79
	30	90.21	0.78	0.90	0.96	0.83
	60	92.06	0.71	0.94	0.99	0.81
	90	93.99	0.75	0.95	0.99	0.84

Table 12: Results of classification using XGBoost

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
AAPL	3	64.57	0.73	0.64	0.56	0.68
	5	71.57	0.77	0.72	0.65	0.74
	10	79.74	0.82	0.81	0.77	0.82
	15	81.67	0.85	0.82	0.78	0.83
	30	86.13	0.89	0.87	0.82	0.88
	60	89.68	0.93	0.90	0.86	0.91
	90	93.06	0.95	0.93	0.90	0.94
TYO	3	64.67	0.51	0.60	0.75	0.55
	5	72.11	0.59	0.64	0.80	0.62
	10	82.40	0.69	0.84	0.91	0.76
	15	85.86	0.75	0.86	0.93	0.80
	30	89.71	0.81	0.86	0.94	0.83
	60	91.97	0.69	0.96	0.99	0.80
	90	95.29	0.81	0.96	0.99	0.88

7.1 Random Forests

The results of classification of the samples from the dataset of stocks of Toyota and Apple are given in Table 11. In general, the accuracy increases as the width of the window is increased. Another important observation that is to be made is that the F-score also increases with the increase in window-width. This definitely confirms the efficacy of the learning algorithms on the dataset used.

7.2 XGBoost

The results of classification as well as the trends observed regarding the change in the classification accuracy and other metrics with the increase in the window-width in the case of XGBoost is similar to the trends observed in the case of Random Forests. Here, too, the classification accuracy and F-score increase with the increase in the window width. Moreover, the goodness of classification observed for a certain window-width of in the case of XGBoost is comparable to the goodness of classification for the same window-width in the case of Random Forests.

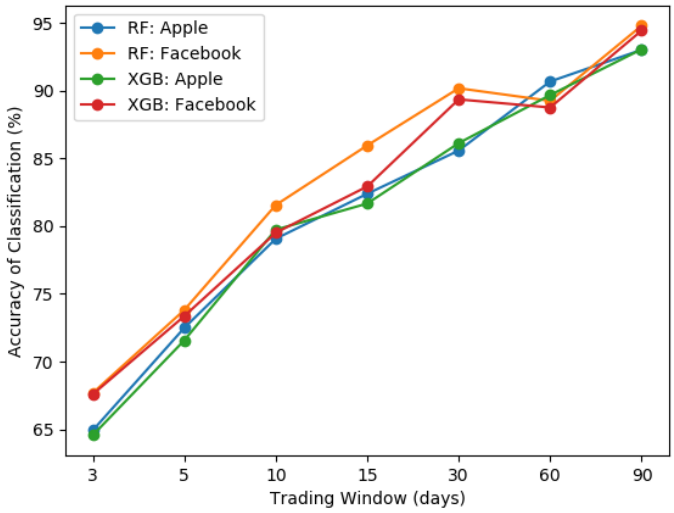

Fig. 12: The trend of accuracy against the trading width considered. The accuracy of classification generally increases as the trading window increases for both Random Forests and XGBoost, used over the two datasets. The values are given in Tables 11 and 12.

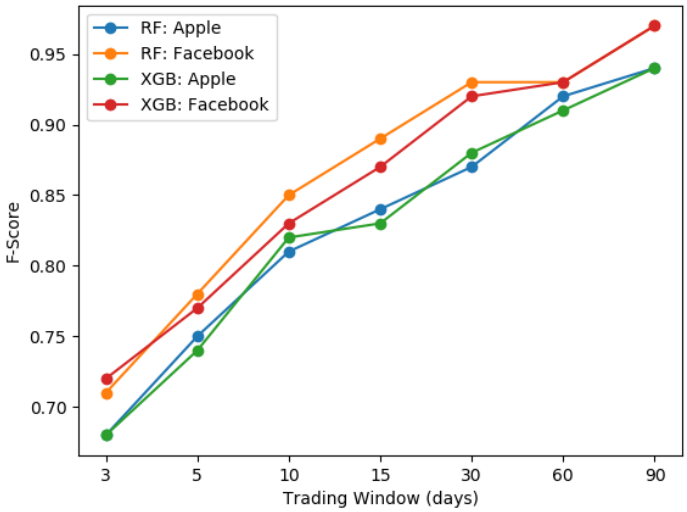

Fig. 13: The trend of F-score against the trading width considered. The F-score increases as the trading window increases for both Random Forests and XGBoost, used over the two datasets. The values are given in Tables 11 and 12.

7.3 Analysis of Pharmaceutical Stocks:

It may be an interesting exercise to investigate why certain stocks did not succumb to the aggravated economic crisis, even during the last round of global meltdown. The relevant candidates could easily be found among the pharmaceutical companies. Generally speaking, the health care industry comprising of the life science sectors such as pharmaceutical, biotechnology, and medical devices do not respond significantly to crisis or boom. These are well-known providers of inelastic commodities and services, that do not undergo visible changes in demand when subjected to income and price effects. These products are less exposed to fluctuations typically because health care requirements continue to remain unchanged regardless of economic prosperity or downturns. Indeed, during economic crisis, it is quite possible that people may be even more susceptible to stress and depression, leading to other forms of ailments. However, it cannot be neglected that during economic crisis, the internal readjustments within such companies generating purely supply side effects could lead to loss of production. It is also expected that research and development, which is at the core of innovations and inventions for the pharmaceutical industry get negatively affected. This may have considerable implications for profit and the outcome of stock prices. The principal conjecture is that the randomness or volatility does not affect the prediction accuracy significantly when the time window is expanded. Pharmaceutical stocks may indeed be resilient to shocks (Behner at al. 1998). It is therefore an empirically open question that we attempt to answer through the general procedures followed above.

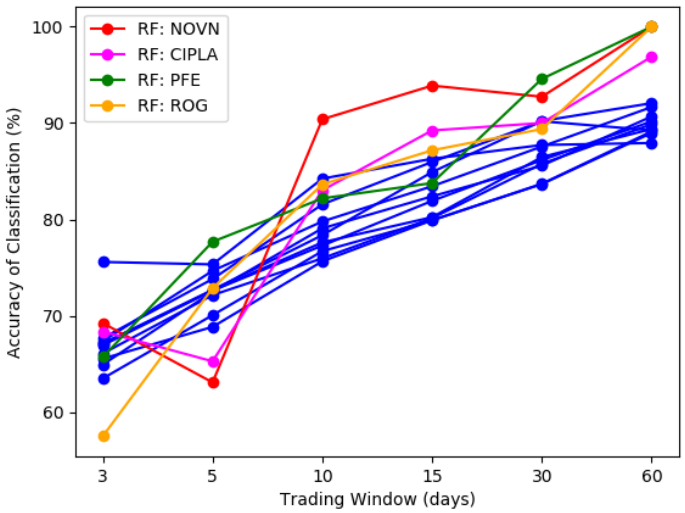

Fig. 14: Comparison of the performance of the algorithm on stock data of pharmaceutical companies. The plots in blue represent the accuracies of ten other companies (the results of whose classification are presented in B. Prediction accuracy in pharma stocks is distinctively better.

From Figure 15, it is observed that in the trading window between 10 and 60 day period, the method exhibits better accuracy. This is noteworthy as it establishes the resistance of pharmaceutical companies to market fluctuations. More specifically, the prediction accuracy achieved for the non-pharma organizations (10 data sets considered in our analysis) requires at least 60-day

window to match the accuracy in prediction, as compared to the pharmaceutical companies that take only 10-15 day window to match the prediction. Typically, the exercise of predictive analysis in pharmaceutical stocks does not seem to require wide training windows. This is clearly in agreement with our conjecture.

As reported by (Behner at al. 1998) (only a few such studies are available since the crisis), unless major policy changes are implemented or completely exogenous factors are in play (such as budget cuts in health care), the pharmaceutical stock prices are less exposed to financial crisis or elasticity. Therefore, understanding and accounting for volatility in pharmaceutical stocks should be easier as compared to other stocks we considered in this paper. Consequently, it is not too hard to comprehend the reason for the remarkable prediction accuracy achieved using shorter time windows.

7.4 Comparison of Performance

In this paper, we have performed exponential smoothing which is a rule of thumb technique for smoothing time series data. Exponential smoothing removes random variation in the data and makes the learning process easier. To our surprise, very few papers found in the literature survey exploited the technique of smoothing. Another important reason could be the inherent non linearity in data. This fact discourages the use of linear classifiers. In (Di 2014), the authors have used a linear classifier as the supervised learning algorithm which yielded a highest accuracy of 55.65%. The prediction by using SVM classifiers were limited to a maximum of 44 day time window which qualifies our learning model to surpass all these metric classifiers in terms of long term prediction. An important practical question that is not entirely solved, is the criteria for the selection of the kernel function parameters - for Gaussian kernels the width parameter σ - and the value of ϵ in the ϵ loss insensitive function.

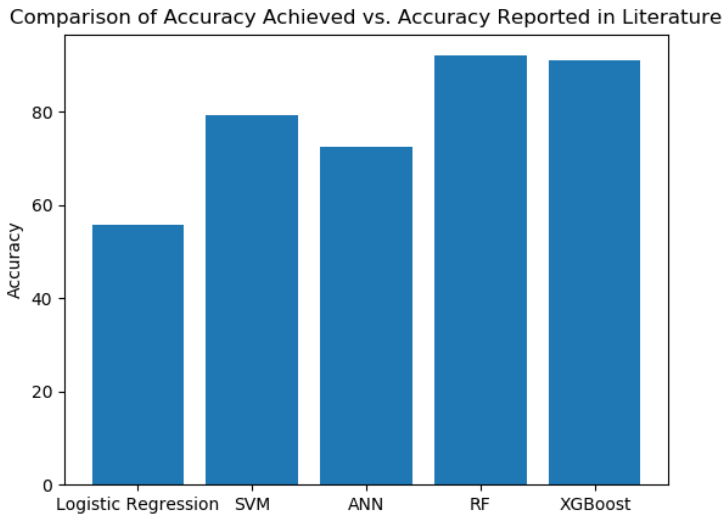

Fig. 15: Comparison of the accuracy achieved in this work to the accuracies achieved in available literature. From the literature survey, it may be concluded that Logistic Regression performs poorly as compared to SVM, RF, and XGBoost. Also, from the bar graph, it is evident that SVM performs well with an accuracy of nearly 80%. However, RF and XGBoost outperforms SVM in terms of accuracy (close to 92% on an average, across different stocks).

8 Discussion and Conclusion

Authors opine that application of machine learning techniques in stock price forecasting needs to be a well thought process and demands painstakingly detailed execution. The proposed approach is a paradigm shift in this class of problems by reformulating a traditional forecasting model as a classification problem. Moreover, knowledge discovery from the analysis should create new frontiers or applications such as a trading strategy based on the strengths of the classification accuracy, investigating the behavior of certain classes of stocks. We achieved this as derivatives of the elaborate machine learning exercise.

Fig. 16: Trading Strategy suggested by the model on AAPL data

– First, let us look at a subset of the trading strategy suggested by the Random Forest model. In Figure 16, the colored dots represent the trading decisions suggested by the model at data points with 90 days time interval. It can be seen from the graph that the model suggests to buy if it is predicted that price is going to rise after 90 days and the model suggests to sell if the price is predicted to fall after 90 days. It is an interesting question if the payoff function has a trend suggesting a trading model in time series.
– The nonlinearity and randomness, usually associated with the movement of stock prices are critical but oversold, sometimes. We believe, certain stock classes may not behave as randomly as conceived. This may happen due to the nature of the products those stocks are affiliated to. We tested this by considering large pharmaceutical companies. Our conjecture appears well placed. In fact, analysis of stock price movements in pharmaceutical companies deserve a dedicated manuscript.

The remarkably high accuracy of prediction could be a matter of concern. Natural suspicion about inherent bias in training data could arise. However, we have checked the data set and confirm the non-existence of heavy bias of data set. The proportion of positive and negative data are in range of 45:55. The absence of the effect of bias in the data can be ascertained by examining the values of the F-scores in comparison to the accuracy (see B).

Predicting stock market due to its non-linear, dynamic and complex nature is difficult. However in the recent years, machine learning techniques have proved effective in stock forecasting. Many algorithms such as SVM, ANN (Boonpeng and Jeatrakul 2015) etc. have been studied for robustness in predicting stock market. However, ensemble learning methods have remained unexploited in this field. In this paper, we have used Random Forests and XGBoost classifiers to build our predictive model and our model has produced impressive results. The model is found to be robust in predicting the direction of stock movement. The robustness of our model has been evaluated by calculating various parameters such as accuracy, precision, recall, specificity, and F-score. For all the datasets we have used, we were able to achieve high accuracies for long-term

predictions. The comparative analysis testifies the efficacy of our model as it outperforms the models discussed in the literature survey. We believe that this is due to the lack of proper data processing (Das and Sundaresan 2012; Chauhan et al. 2014; Mayankkumar and Patel 2014). In addition to that, a major part of the novelty of the current work lies in the careful selection of technical indicators and their usage as features. As the type of the problem that we're trying to solve is primarily that of financial analysis, we had the advantage of flexibility of the usage of various different features, each with it's own interpretation.

Our model can be used for devising new strategies for trading or to perform stock portfolio management, changing stocks according to trends prediction. The proposed model is indeed a novel way to minimize the risk of investment in stock market by predicting the returns of a stock more accurately than existing algorithms applied so far. In future, we could build boosted tree models to predict trends for short time windows in terms of hours or minutes. Ensembles of different machine learning algorithms can also be checked for its robustness in stock prediction. We also recommend exploration of the application of Deep Learning practices in Stock Forecasting involving learning weight coefficients on large, directed and layered graphs.

We have studied white papers and technical reports trying to use ML methods with 2-4 days' training window (historical data). We maintain that a sizable training window is necessary for reasonable prediction (60, 90 or 120 days' training window) since the efficacy of ML methods lie in the transition of short term to long term training window. However, we assumed the role of the devil's advocate and analyzed the stock prices based on short term training window. The prediction of stock prices yielded accuracy slightly better than a fair coin toss (for really short windows, 3 days or 5 days). This is expected since using historical data during such short term window fortifies the principle of efficient market hypothesis. The prediction exercise using short training windows is therefore futile and no different from the principle of efficient market hypothesis. Neural networks with short term memory is not appropriate for the same reasons (Nelson et al. 2017).

The codebase can be found at `https://github.com/LuckysonKhaidem/StockMarketPedicition` and `https://github.com/SuryodayBasak/StockMarketPedicition`. All the code for this research is written in Python. In the process of developing the code and performing the experiments, the authors have made extensive use of the libraries scikit-learn (Pedregosa et al. 2011), XGBoost (Chen and Guestrin 2016), and matplotlib (Hunter 2007).

9 References

References

Appel, G. (2005). Technical Analysis Power Tools for Active Investors. ISBN:0-13-147902-4.

Avery, C. N., Chevalier, J. A. & Zeckhauser, R. J. (2016). The CAPS Prediction System and Stock Market Returns. Review of Finance. 20 (4), 1363-1381.

Bao D. & Yang Z. (2008). Intelligent stock trading system by turning point confirming and probabilistic reasoning, Expert Systems with Applications, 34 (1), 620-627.

Beechey M., Gruen D. & Vickery J. (2000). The Efficient Market Hypothesis: A Survey, Research Discussion Paper. Economic Research Department. Reserve Bank of Australia.

Behner, P., Schwarting, D., Vallerien, S., Ehrhardt, M., Beever, C. & Rollmann, D. (2009). Pharmaceutical Companies in the Economic Storm Navigating from a Position of Strength. Technical Report: BooZ& Co Analysis.

Boonpeng, S., & Jeatrakul, P. (2016). Decision Support System for Investing in Stock Market by using OAA-Neural Network. 8th International Conference on Advanced Computational Intelligence Chiang Mai, Thailand.

Breiman, L. (2001), Statistics Department, University of California Berkeley, CA 94720. Random Forests.

Bylander, T. & Hanzlik, D. (1999) Estimating Generalization Error Using Out-of-Bag Estimates. AAAI-99 Proceedings.

Chauhan, B., Umesh, B., Ajit, G., Sachin, K. (2014). Stock Market Prediction Using Artificial Neural Networks. International Journal of Computer Science and Information Technology. 5 (1), 904-907.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System, Proceedings of the 22^{nd} ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD'16. doi:10.1145/2939672.2939785

Dai, Y. & Zhang Y. (2013). Machine Learning in Stock Price Trend Forecasting. Stanford University, http://cs229.stanford.edu/proj2013/DaiZhang-MachineLearningInStockPriceTrendForecasting.pdf

Das, S. P. & Sudersan, P., Support Vector Machines for Prediction of Future Prices in Indian Stock Market. International Journal of Computer Applications. 41 (3), 22-26, 2012.

Devi, K. N., Bhaskaran, V. M. & Kumar, G. P. (2015). Cuckoo Optimized SVM for Stock Market Prediction, IEEE Sponsored 2nd International Conference on Innovations in Information, Embedded and Communication systems (ICJJECS).

Di, X. (2014), Stock Trend Prediction With Technical Indicators using SVM. Stanford University.

Friedman, J. H. (2000). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics. Vol. 29, p.1189–1232.

Gencay R. (1999). Linear, non-linear and essential foreign exchange rate prediction with simple technical trading rules. Journal of International Economics. Vol. 47, no.1, pp. 91-107.

Geurts, P., & Louppe, G. (2011). Learning to rank with extremely randomized tree. JMLR: Workshop and Conference Proceedings, 14, 4961.

Giacomel ,F. , Galante, R. & Pareira, A. (2015). An Algorithmic Trading Agent based on a Neural Network Ensemble: a Case of Study in North American and Brazilian Stock Markets. IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology.

Granville, J. E. (1976). Granville's New Strategy of Daily Stock Market Timing for Maximum Profit. ISBN: 0-13-363432-9.

Hellstrom, T. & Holmstromm, K. (1998). Predictable Patterns in Stock Returns. Technical Report Series IMA-TOM-1997-09.

Hunter, J. D. (2007) Matplotlib: A 2D graphics environment. Computing In Science & Engineering. 9(3), 90–95, DOI:10.1109/MCSE.2007.55.

Imandoust, S. B. & Bolandraftar, M., (2014). Forecasting the direction of stock market index movement using three data mining techniques: the case of Tehran Stock Exchange. International Journal of Engineering Research and Applications. 4(6), 106-117.

Jensen M. C. (1978). Some Anomalous Evidence Regarding Market Efficiency. Journal of Financial Economics. 6 (2), 95-101.

Khaidem L., Saha S. & Roy Dey S. (2016). Predicting the direction of stock market prices using random forest. arXiv preprint arXiv:1605.00003.

Khan, W., Ghazanfar, M.A., Asam, M., Iqbal, A., Ahmed, S. & Khan, J. A. (2016). Predicting Trend In Stock Market Exchange Using Machine Learning Classifiers. Sci.Int, 28 (2), 1363-1367.

Lane, G. (1984) "Lanes Stochastics. Second issue of Technical Analysis of Stocks and Commodities magazine. pp 87-90.

Li, H., Yang, Z. & Li, T. (2014). Algorithmic Trading Strategy Based On Massive Data Mining. Stanford University. http://cs229.stanford.edu/proj2014/Haoming.

Malkiel, B. G. & Burton G. (2003). The efficient market hypothesis and its critics. The Journal of Economic Perspectives. 17 (1), 59-82.
Malkiel, B. G. & Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. The Journal of Finance. 25 (2), 383-417, 1970.
Mayankkumar, S. R. Y. & Patel, B. (2014). Stock prediction using artificial neural network. International Journal of Innovative Research in Science, Engineering and Technology. 3 (6), 76-84.
Nelson, D. M. Q., Pereira, A. C. M. & de Oliveira, R. A. (2017). Stock market's price movement prediction with LSTM neural networks. International Joint Conference on Neural Networks (IJCNN), 1419-1426.
Pedregosa F. et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830, 2011.
Qiu, M. & Song, U. (2016). Predicting the Direction of Stock Market Index Movement Using an Optimized Artificial Neural Network Model. PLOS ONE 11 (5).
Saha, S., Routh, S. & Goswami, B. (2014). Modeling Vanilla Option prices: A simulation study by an implicit method. Journal of Advances in Mathematics. 6 (1), 834-848.
Sewell M. (2011). History of the Efficient Market Hypothesis. Research Note RN/11/04- UCL Dept. of Computer Science.
Timmermann, A. & Granger, C. W. (2004). Efficient market hypothesis and forecasting. Interational Journal of Forecasting. 20 (1), 15-27
Widom, J. (1995). Research problems in data warehousing. In Proceedings of the fourth international conference on information and knowledge management. CIKM '95 (pp. 25- 30). New York, NY, USA: ACM. 10.1145/221270.221319.
Wilder Jr., J. W. (1978). New Concepts in Technical Trading Systems. ISBN 978-0-89459-027-6.

A Key Definitions

In this appendix, we introduce some key terms which have been used frequently through the paper. A basic understanding of the key concepts will bolster the reader's understanding of the methods used for predictive analysis.

1. **Data Structure**: In computer science, a *data structure* refers to the means of organizing and storing data for analysis and manipulation. Knowledge of data structures is fundamental for a computer scientist to judge how data should be stored for a particular application or task. Some popular data structures include stacks, queues, linked-lists, and trees.
2. **Node**: A node in a data structure is essentially an instance of that structure that contains data. A data structure is made of multiple nodes. It is the way in which nodes are connected in a data structure that defines it's characteristics.
3. **Tree Data Structure**: In a tree data structure, every node is the child of a single node (with the exception of the root node) and can have one or more children nodes. The name is inspired from how this data structure looks as subsequent levels of nodes *branch* or *spread out* like in a tree (Figure 3; notice how the subsequent levels of nodes branch out).
4. **Root Node**: In a tree data structure (henceforth referred to as just *tree*), the first node from which other nodes branch out is called the *root node*.
5. **Child Node**: A node in a tree can have one or more subordinate nodes or hierarchically lower nodes. These are called as child nodes of the respective node. In any tree, only the root node is not a child node.
6. **Level**: The level of a set of nodes in a tree is an index of how far away a node is from the root node. The level of the root node is 0. The level of the immediate children of the root node is 1. The level of the children nodes of the nodes at level 1 is 2, and so on.
7. **Parent Node**: If a node exists in a tree, it must be the subordinate of another node (with the exception of the root node) called it's parent node. A parent node of a child is the node that is immediately higher in hierarchy to the respective child node. The level of a node's parent is always one less than the level of the respective child.

8. **Leaf Node**: A leaf node is a node that does not have any children. All finite trees end with leaf nodes. In a classification tree, all leaf nodes need to be *pure*, i.e., they should have entities belonging to any one class only (exceptions to this may be incorporated by *pruning* a tree to prevent overfitting; pruning, in a general sense, is the removal of some leaf nodes with very few entities in them, before or after the tree has been built, so as to prevent overfitting).
9. **Probability Distribution**: $X = (X_1, ..., X_d)$ is an array of random variables defined on a probability space called as random vectors. The joint distribution of $X_1, ..., X_d$ is a measure on μ on R^d, $\mu(A) = P(X \in A)$, $A \in R^d$ where $d = 1, ..., m$. For example, Let $x = (x_i, ..., x_d)$ be an array of data points. Each feature x_i is defined as a random variable with some distribution. Then the random vector X has joint distribution identical to the data points, x.
10. **Classification Tree**: We define a classification tree where each node is endowed with a binary decision: **whether** $x_i \leq k$ **or not** ; where x_i is a feature in the data set, and k is some threshold. The topmost node in the classification tree contains all the data points and the set of data is subdivided among the children of each node as defined by the classification. The process of subdivision continues until pure leaf nodes are achieved. Each node is characterized by the feature x_i and threshold k chosen in such a way that minimizes diversity among the children nodes. This is often referred to as gini impurity.
11. **Random Forest**: Let us represent $h_k(x) = h(x|\theta_k)$ implying decision tree k leading to a classifier $h_k(x)$. Thus, a random forest is a classifier based on a family of classifiers $h(x|\theta_1), ..., h(x|\theta_k)$, which is an ensemble of classification trees with model parameters θ_k randomly chosen from model random vector θ. Each classifier, $h_k(x) = h(x|\theta_k)$ is a predictor of the number of training samples. $y = \pm 1$ is the outcome associated with input data, x for the final classification function, $f(x)$, which can result by a majority voting mechanism of the individual classifiers.
12. **Linear Separability**: Before feeding the training data to the Random Forest Classifier, the two classes of data are tested for linear separability by finding their convex hulls. Linear Separability is a property of two sets of data points where the two sets are said to be linearly separable if there exists a hyperplane such that all the points in one set lies on one side of the hyperplane and all the points in other set lies on the other side of the hyperplane.
    Mathematically, two sets of points X_0 and X_1 in n dimensional Euclidean space are said to be linearly separable if there exists an n dimensional normal vector W of a hyperplane and a scalar k, such that every point $x \in X_0$ gives $W^T x > k$ and every point $x \in X_1$ gives $W^T x < k$. Two sets can be checked for linearly separability by constructing their convex hulls.
13. **ROC**: In statistics, a *receiver operating characteristic* (ROC) curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection[1] in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm[1] and can be calculated as (1− specificity). The ROC curve is thus the sensitivity as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from -∞ to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability in x-axis.
    There are four possible outcomes from a binary classifier. If the outcome from a prediction is positive (p) and the actual value is also p, then it is called a *true positive* (TP); however if the actual value is negative (n) then it is said to be a false positive (FP). Conversely, a true negative (TN) has occurred when both the prediction outcome and the actual value are n, and false negative (FN) is when the prediction outcome is n while the actual value is p. To draw an ROC curve, only the true positive rate (TPR) and false positive rate (FPR) are needed (as functions of some classifier parameter). The TPR defines how many correct positive results occur among all positive samples available during the test. FPR, on the other hand, defines how many incorrect positive results occur among all negative samples available during the test.
    An ROC space is defined by FPR and TPR as x and y axes respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Since TPR is equivalent to sensitivity and FPR is equal to 1− specificity, the ROC graph is sometimes called the sensitivity vs (1− specificity) plot. Each prediction result or instance of a confusion matrix represents one point in the ROC space. The diagonal divides the ROC space. Points above the diagonal represent good classification results (better than random), points below the line represent poor results (worse than random).

B Results

In this appendix, we elaborate the experimental results achieved by performing the experiments.

Table 13: RF Results Table 1: results of random forests implemented on stocks of AAPL, AMS, and AMZN

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
AAPL	3	64.93	0.71	0.65	0.58	0.68
	5	72.49	0.77	0.73	0.67	0.75
	10	79.07	0.81	0.81	0.77	0.81
	15	82.39	0.86	0.83	0.79	0.84
	30	85.58	0.89	0.86	0.82	0.87
	60	90.68	0.94	0.90	0.87	0.92
	90	93.02	0.95	0.93	0.90	0.94
AMS	3	65.54	0.73	0.67	0.57	0.70
	5	68.84	0.74	0.68	0.63	0.71
	10	75.65	0.76	0.75	0.75	0.76
	15	79.91	0.80	0.78	0.80	0.79
	30	83.67	0.82	0.85	0.86	0.83
	60	88.97	0.89	0.89	0.89	0.89
	90	90.44	0.89	0.91	0.91	0.90
AMZN	3	66.93	0.75	0.66	0.58	0.70
	5	72.63	0.75	0.74	0.70	0.74
	10	77.61	0.81	0.77	0.74	0.79
	15	80.23	0.86	0.79	0.74	0.82
	30	86.47	0.92	0.86	0.79	0.89
	60	89.38	0.94	0.89	0.82	0.91
	90	94.85	0.96	0.96	0.93	0.96

Table 14: RF Results Table 2: results of random forests implemented on stocks of FB, MSFT, and NKE

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
FB	3	67.69	0.73	0.69	0.62	0.71
	5	73.81	0.83	0.73	0.62	0.78
	10	81.56	0.90	0.81	0.70	0.85
	15	85.96	0.89	0.90	0.79	0.89
	30	90.18	0.96	0.90	0.80	0.93
	60	89.25	0.98	0.89	0.60	0.93
	90	94.80	0.98	0.96	0.72	0.97
MSFT	3	67.06	0.72	0.67	0.61	0.70
	5	72.61	0.76	0.74	0.69	0.75
	10	77.28	0.85	0.76	0.68	0.80
	15	81.92	0.87	0.82	0.75	0.85
	30	86.16	0.88	0.88	0.84	0.88
	60	90.17	0.92	0.92	0.88	0.92
	90	92.34	0.94	0.93	0.90	0.93
NKE	3	66.08	0.72	0.68	0.59	0.70
	5	72.14	0.79	0.72	0.64	0.76
	10	75.96	0.83	0.76	0.68	0.79
	15	80.13	0.87	0.80	0.70	0.84
	30	85.77	0.93	0.85	0.74	0.89
	60	89.87	0.93	0.90	0.84	0.92
	90	93.35	0.97	0.93	0.88	0.95

Table 15: RF Results Table 3: results of random forests implemented on stocks of SNE, TATA, TWTR, and TYO

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
SNE	3	63.49	0.67	0.63	0.60	0.65
	5	70.06	0.71	0.71	0.70	0.71
	10	76.70	0.78	0.77	0.75	0.78
	15	79.90	0.83	0.79	0.77	0.81
	30	83.69	0.84	0.84	0.83	0.84
	60	89.13	0.91	0.89	0.87	0.90
	90	89.71	0.92	0.88	0.87	0.90
TATA	3	67.35	0.63	0.66	0.71	0.64
	5	74.66	0.69	0.75	0.80	0.72
	10	79.78	0.72	0.81	0.86	0.76
	15	83.44	0.75	0.86	0.90	0.80
	30	87.56	0.83	0.89	0.92	0.86
	60	91.66	0.87	0.92	0.95	0.89
	90	95.34	0.94	0.94	0.96	0.94
TWTR	3	75.58	0.73	0.75	0.78	0.74
	5	75.33	0.78	0.69	0.73	0.73
	10	84.23	0.83	0.82	0.86	0.82
	15	86.29	0.89	0.80	0.84	0.84
	30	87.72	0.81	0.85	0.92	0.83
	60	87.93	0.77	0.84	0.93	0.80
	90	86.16	0.83	0.83	0.88	0.83
TYO	3	67.13	0.53	0.64	0.77	0.58
	5	72.69	0.61	0.65	0.80	0.63
	10	78.36	0.60	0.80	0.90	0.69
	15	84.91	0.73	0.85	0.92	0.79
	30	90.21	0.78	0.90	0.96	0.83
	60	92.06	0.71	0.94	0.99	0.81
	90	93.99	0.75	0.95	0.99	0.84

Table 16: XGBoost Results Table 1: results of XGBoost implemented on stocks of AAPL, AMS, and AMZN.

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
AAPL	3	64.57	0.73	0.64	0.56	0.68
	5	71.57	0.77	0.72	0.65	0.74
	10	79.74	0.82	0.81	0.77	0.82
	15	81.67	0.85	0.82	0.78	0.83
	30	86.13	0.89	0.87	0.82	0.88
	60	89.68	0.93	0.90	0.86	0.91
	90	93.06	0.95	0.93	0.90	0.94
AMS	3	63.21	0.73	0.64	0.52	0.68
	5	68.61	0.75	0.68	0.62	0.71
	10	75.27	0.76	0.74	0.75	0.75
	15	78.41	0.78	0.77	0.79	0.77
	30	83.58	0.82	0.85	0.85	0.83
	60	88.67	0.90	0.88	0.88	0.89
	90	89.84	0.90	0.90	0.90	0.90
AMZN	3	66.61	0.76	0.66	0.57	0.70
	5	72.12	0.76	0.72	0.68	0.74
	10	79.06	0.84	0.78	0.74	0.81
	15	80.02	0.86	0.78	0.73	0.82
	30	85.73	0.91	0.85	0.78	0.88
	60	87.53	0.92	0.87	0.80	0.90
	90	94.06	0.96	0.94	0.91	0.95

Table 17: XGBoost Results Table 2: results of XGBoost implemented on stocks of FB, MSFT, and NKE.

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
FB	3	67.59	0.78	0.68	0.55	0.72
	5	73.36	0.82	0.73	0.62	0.77
	10	79.51	0.87	0.80	0.69	0.83
	15	82.93	0.88	0.87	0.73	0.87
	30	89.36	0.95	0.89	0.79	0.92
	60	88.76	0.98	0.89	0.57	0.93
	90	94.44	0.98	0.96	0.72	0.97
MSFT	3	65.58	0.72	0.65	0.58	0.69
	5	72.11	0.77	0.73	0.67	0.75
	10	76.95	0.86	0.75	0.66	0.80
	15	81.37	0.86	0.82	0.75	0.84
	30	84.79	0.86	0.87	0.83	0.87
	60	89.99	0.91	0.92	0.89	0.91
	90	91.22	0.94	0.91	0.88	0.92
NKE	3	65.05	0.74	0.66	0.54	0.70
	5	70.76	0.80	0.70	0.60	0.75
	10	76.81	0.85	0.76	0.67	0.80
	15	80.68	0.89	0.80	0.69	0.84
	30	86.15	0.93	0.86	0.75	0.89
	60	90.01	0.93	0.91	0.86	0.92
	90	92.62	0.96	0.92	0.87	0.94

Table 18: XGBoost Results Table 3: results of XGBoost implemented on stocks of SNE, TATA, TWTR, and TYO

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score
SNE	3	62.21	0.67	0.62	0.57	0.64
	5	70.06	0.73	0.70	0.67	0.72
	10	76.58	0.81	0.75	0.72	0.78
	15	79.19	0.84	0.77	0.74	0.80
	30	82.64	0.83	0.84	0.83	0.83
	60	87.93	0.91	0.87	0.85	0.89
	90	88.89	0.90	0.88	0.87	0.89
TATA	3	65.90	0.59	0.65	0.72	0.62
	5	74.12	0.69	0.74	0.79	0.71
	10	78.68	0.72	0.79	0.84	0.75
	15	85.93	0.79	0.87	0.91	0.83
	30	86.65	0.81	0.88	0.91	0.85
	60	91.24	0.87	0.90	0.94	0.89
	90	94.03	0.94	0.91	0.94	0.93
TWTR	3	72.08	0.73	0.70	0.71	0.72
	5	76.14	0.78	0.70	0.75	0.74
	10	80.61	0.83	0.76	0.79	0.79
	15	86.60	0.88	0.82	0.86	0.85
	30	87.30	0.81	0.84	0.91	0.83
	60	87.36	0.75	0.84	0.93	0.79
	90	84.91	0.77	0.85	0.90	0.81
TYO	3	64.67	0.51	0.60	0.75	0.55
	5	72.11	0.59	0.64	0.80	0.62
	10	82.40	0.69	0.84	0.91	0.76
	15	85.86	0.75	0.86	0.93	0.80
	30	89.71	0.81	0.86	0.94	0.83
	60	91.97	0.69	0.96	0.99	0.80
	90	95.29	0.81	0.96	0.99	0.88