Stock Market Prediction with Japanese Candlestick Pattern Recognition

Pritam Chandra, Vir Jhangiani, Kshitij Kapoor

Abstract: This report presents a new model that tries to predict the direction of returns of an instrument traded on electronic markets by recognizing patterns in Japanese Candlestick data by using a Random Forest Classifier. Rather than trying to predict the absolute magnitude, the model is trained to predict whether the market is going to make a significant upward move, a significant downward move, or is going to stay neutral in the next few bars. Hence the model is trained to classify input data to one of three possible classes, i.e. Buy, Sell or Hold. This report also presents a method for normalizing Japanese Candlestick data and labelling the training dataset into the three mentioned classes. Experimental results conducted on the model trained on actual data shows a win rate of above 50% for Buy and Sell calls, which might translate to positive excess returns for a trading strategy based on this model.

1. Introduction and Motivation

A model that can predict the movement of a tradeable instrument can help an investor time their entry and exit into the market. With the growing popularity of electronic markets, each and every trade can be recorded digitally and hence a huge amount of high frequency historical data is available for training and creating trading models which might use this data. The models can be first principles based or might be based on empirical estimators which try to minimize a given loss function. With the growing amount of computation power it also becomes easy to transform this data and train complicated empirical estimators like neural networks, decision trees or random forest classifiers on this data set of historical price. Hence, it is natural to look at how empirical models can be used to predict the direction of price returns of instruments on the stock market.

2. Background and Previous Work

2.1. Technical Analysis and Japanese Candlestick Patterns

Technical analysis is the practice of trying to predict future returns of a traded instrument by looking at recent price movement patterns. Technical analysis doesn't look at the fundamental factors like demand and supply for a commodity or financial statements for an equity share to figure out whether a tradable instrument is underpriced or overpriced. Rather than trying to figure out the instrument's intrinsic value, technical analysts observe recent price moves to figure out the direction of the future price moves. Technical analysts mostly either try to find statistical patterns in the price movements by using basic statistical tools like moving averages and variance or try to chart the price movements and find visually apparent patterns in price movements. Both these strategies are used immensely across the finance industry and are believed to provide excess positive returns.

The first form of technical analysis involves the use of statistical tools to predict the direction of price returns using well defined mathematical logic. Analysts perform mathematical transforms like moving average and moving standard deviation on the stock price time series and try to form a hypothesis around the values of these mathematical transforms and their relations. For example, the most common trading system involves looking at two moving averages, one of a shorter time frame and one of a longer time frame. A decision on the direction of the price of the instrument can be made on the basis of whether the absolute value of the shorter time frame moving average is above or below the longer time frame moving average.

The second form involves looking at price charts of various types and on the basis of that coming up with a signal on where the price of the instrument is going to be in the future. Japanese Candlesticks, a popular charting technique among such analysts, is used to define and identify recurring patterns. Since a human is

supposed to interpret these results and there are no well defined mathematical parameters for a pattern, it is very likely that there are two interpretations of the same chart. An example of such patterns can be seen in the following image. The image makes it pretty clear that given a chart, different inferences can be made.

An example of how Japanese Candlestick Pattern Recognition fails

The theoretical basis for technical analysis lies in the fact that it tries to establish how the market has reacted to a fundamental change by looking at how price moves. It is supposed to act as an aggregate of the psychology of the market participants weighted by the respective volume of the trades they make and hence the analysis of the same might act as an indicator of how the market moves. Another theory for why technical analysis might work is that if enough number of analysts make the same inference given a set of observations, the market might move in the same inferred direction, as the orders they place on the market also have an affect on the market. Hence, according to this hypothesis technical analysis might be a self fulfilling prophecy. The goal of this model is to capitalize on this theory, irrespective of whether technical analysis is an aggregate of the psychology of the market participants or a self fulfilling prophecy.

2.2. Neural Network for Adaptive Technical Analysis

In many situations because of the continuous change of the world around us, a true model of forecasting for a specific period of time does not make sense (Winkler, 1989). That is forecasting models encounter the challenge of adaptability seriously in the way that they must be updated continuously according to new conditions. This implies that a Technical analysis is not always applicable to all markets because unlike logic, sentiments are variable and not fixed for a long time. In fact if a typical technical analysis is beneficial now it may not in future and vice versa. Or if it is useful in market A, it may not in market B and vice versa. In a nutshell because of the sentimental foundation of technical analysis, the practitioners' reactions to the changes in the charts can be different depending on the place and time; so considering fixed outcomes for the specific changes in the charts is not always wise.

Therefore there is a need for an adaptive form of technical analysis that confirms old rules and continuously updates the rules of technical analysis based on new data. Here is where the Neural Network comes in. It plays the role of the analyzer. The abilities of a Neural network to decode nonlinear, uncertain, fuzzy of insufficient time series data that adequately describe the characteristics of the stock markets in very short periods of time, and also manage to do so with better ability than a human being(Yao, Tan, & Poh, 1999),(Schoeneburg, 1990) are what make it the best fit.

We look to a paper by Jasemi, Kimiagari and Memariani (2010) that presents a model to do stock market predictions based on a supervised feed-forward neural network and the technical analysis of the Japanese Candlestick charts. The approach they used is as follows: The data was separated into 3 classes: Buy, Sell and Neutral where if there was an upward or downward movement of the stock beyond a certain threshold in the next six days from the day in question then the stock for that day was classified into either a Buy or a Sell respectively. Another paper by Marshall et al. (2006) applies a ten-day period. Everything that did not cross this threshold was placed under the hold category. In the paper the threshold for positive returns had a lower bound of $\frac{1}{5}mp$ and negative returns had an upper bound of $\frac{1}{5}mn$ where mp and mn are the mean value of total

positive and negative returns for the year of the training data respectively.

The paper follows a raw data based approach where it feeds the neural network a vector of 15 inputs. It uses the data of the past 7 days. Where the signal to buy sell or hold is to be predicted for the next day or the 8th day. The first three values of the vector are $\frac{C_i}{C_1}$ where $i = 2, 3, 4$ and C_i is the closing price of the stock on the i^{th} day. The rest of the input features are $\frac{O_i}{C_1}$, $\frac{H_i}{C_1}$, $\frac{L_i}{C_1}$ and $\frac{C_i}{C_1}$ where $i = 5, 6, 7$ and O_i, H_i, L_i are the open, high and low prices of the stock on that day respectively. Here C_1: the closing price of the stock on the first day is used to normalise the data. The neural network has a single output node that assigns the day one of the three classes.

According to literature feeding the neural networks raw data is common since they are able to recognize the high-level features, such as the serial correlation, and are known as an appropriate classification and forecasting tool in business applications (Law & Au, 1999).

The Questions that the model is trying to answer is that:

- Is the model efficient in general?
- Does the quality of forecasting improve when a longer training period is used?
- Does the quality of forecasting decline when the distance(In time) between training and testing periods increases?

The model is trained and tested on 48 different train - test pairs of General Motors stocks on the NYSE for the years 2000 to 2008. Where it uses 3 different time periods of 1 year, 2 years and 3 years for training data and tests it on all the following years.

For example it will train the model with data from the year 2000 and then test it with data from the years 2001-2008. Then it will train it with data from the years 2000 and 2001 and test it with the training data for the years 2002-2008, then use 2000-2002 as training data and so on.

In a paper by Lee and Jo (1999) they believe that if the hit ratio defined by $\frac{\text{Number of successes}}{\text{Total number of signals}}$ is above 51%, then the model is regarded as useful and feasible. The method described in this paper produced a success rate between 70% to 81% on all train-test pairs.

Other inferences the paper makes is that increasing the length of training period shows small improvements with an increase of upto 9% accuracy in some train-test pairs and that the distance between training and testing data has no influence on the quality of results.

The paper concludes by stating that the accuracy achieved shows good forecasting efficiency and that Neural Networks serve as a good analyzer. Development of new networks from the perspectives of differing input layers or applying different forms of technical analysis and comparing them with other configurations is also a promising area to continue the study.

The paper fails to mention the loss function it uses for backpropagation, the activation function it uses for training and the number of hidden layers and nodes it uses. It also does not explain why the threshold of $\frac{1}{5}mp$ and $\frac{1}{5}mn$ are used in labelling the data. We would build upon this at a later point in the report.

3. Approach

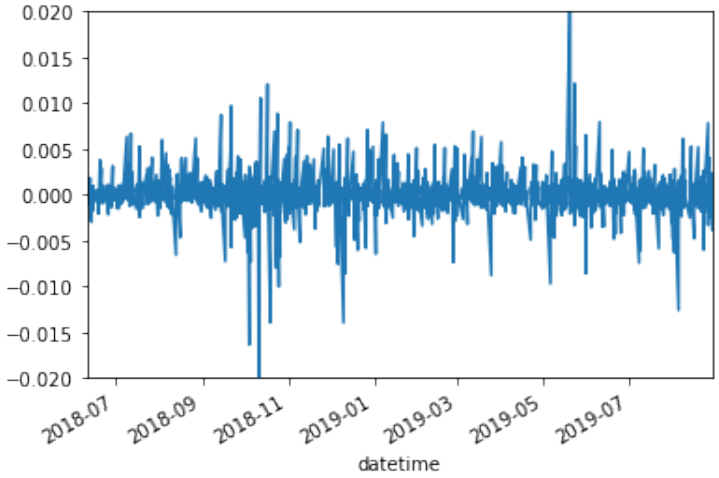
3.1. Data Labelling

The following variables are used in preprocessing the data. Their roles would be discussed in detail in the following sections.

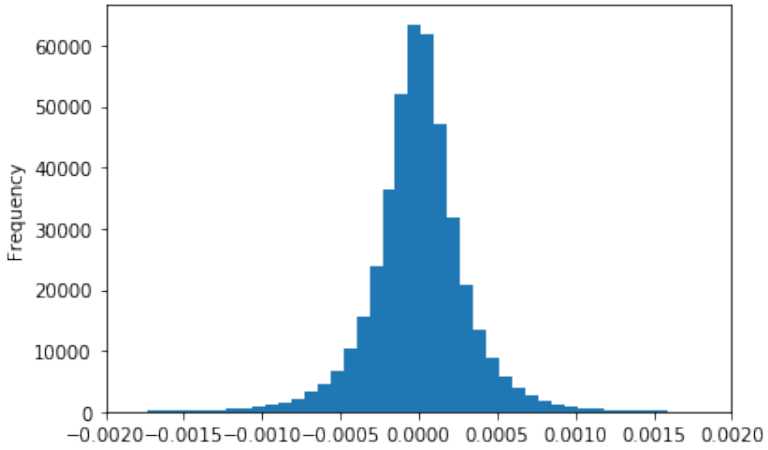
- **lookback**: refers to the length of historical data from a given point in time to be fed to the machine for prediction. This is precisely the input vector.

- **lookahead**: refers to the length of future data from a given point in time whose trends decide the class label of the input vector.
- **stddev_lookback**: refers to the length of historical data from a given point in time that determines the boundaries between different classes. This is taken to be a value much larger than *lookback*.

In this project the aspect of labelling data is grounded on the property of stock returns to be normally distributed with a mean close to zero. One can quickly plot the one minute returns of Nifty prices to observe graphs similar to Gaussian noise. Plotting a frequency distribution of these values over a period of considerable length shows the formation of the smooth bell curve.



A Plot of minute returns on Nifty data that looks like Gaussian Noise



A histogram of minute returns on Nifty data

In the coming section we would denote *lookahead* by k, *stddev_lookback* by l, and *lookback* by n. Ideally, the difference between l and n is large.

For the sake of clarity we would also introduce the following notations. For an array A and numbers i, j, we have,

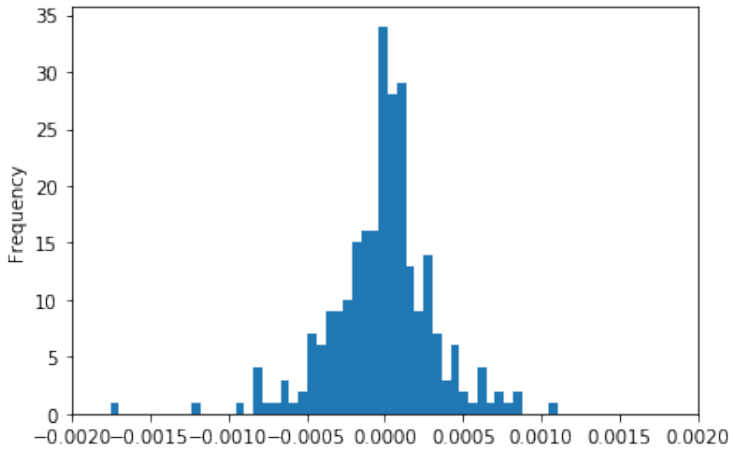
- $A_j^1(i) = [a_{i-j}, a_{i-j+1}, ..., a_i]$, which are all values of A from index $i-j$ to i. Similarly,
- $A_j^2(i) = [a_{i+1}, a_{i+2}, ..., a_{i+j}]$, which are all values of A from index $i+1$ to $i+j$.

To begin with, we first construct an array C comprising of all the minute-wise closing prices from the beginning to the end of the time period considered; and an array R consisting of the corresponding returns calculated as,

$$r_{i+1} = \frac{c_{i+1} - c_i}{c_i}, \text{ and for } i \geq 0.$$

We begin from index $i = l$ of C, and consider the n values preceding i as the potential input vector x. This vector $x = C_n^1(i)$. To label this input we analyse any observable trends in the following k values. This vector is represented by $C_k^2(i)$. Thus, $C_n^1(i)$ and $C_k^2(i)$ are vectors of length n and k respectively. We similarly define $R_l^1(i)$ and $R_k^2(i)$. Clearly, our ending index is k less than the length of the array C.

We have observed that the values in R are are normally distributed with a mean close to 0. Moreover, a particular subsection of considerable length that is drawn from R is also normally distributed. We have specified the length of such a subsection to be l. Our aim is construct a function of the input vector on which we would impose conditions to obtain the class of the input.



A histogram of minute returns for 250 minutes on Nifty data

To ensure uniformity, we would try to construct the thresholds in such a way that an input vector x is equally likely to fall in any of the three classes, i.e.,

$$P[x \in B] = P[x \in H] = P[x \in S] = 1/3 \ ...(I)$$

Where B, H, S represent classes Buy, Hold and Sell respectively. This, to some extent, would lead to a suitably representative dataset.

This would have been a simple task if we knew about the distribution of $R_k^2(i)$. Our primary job is to predict the trend in the k values starting from index i, using the values preceding index i. Since l is a considerably "large" number, we know that the subsections $R_l^1(i)$ and $R_{l+k}^1(i+k)$ are both normally distributed. The first array consists of the l values till i and the second consists of the first array appended to the k future values from index i.

Since $l >> k$, (and the observed variance in R is small) **we assume that the distributions of $R_l^1(i)$ and $R_{l+k}^1(i+k)$ are approximately similar.** This is because a normal distribution is uniquely defined by its mean and variance, and we expect that the introduction of a small number (k) of values into the system doesn't have much impact on the original mean and variance. This assumption is the key to the labelling technique used in this project.

Let $\mu = E[R_l^1(i)]$, $\sigma = \sqrt{V[R_l^1(i)]}$, which are usual mean and variance of the set $\{R_l^1(i)\}$ sampled from a normal population approximately following $N(\mu, \sigma)$, i.e., each $r \in R_l^1(i)$ is an instantiation of the Normal variable with the given mean and variance. It is usually observed that $\mu \approx 0$. In addition, we expect $E[R_{l+k}^1(i+k)] \approx \mu$, $\sqrt{V[R_{l+k}^1(i+k)]} \approx \sigma$. While in practise this σ is an overestimate of the actual standard deviation of the $R_{l+k}^1(i+k)$.

In theory we would want a critical value t, such that the class definitions are as follows.

- The input $x = C_n^1(i)$ belongs to class "hold", if for all $w \in [i+1, i+2, ..., i+k]$, $\mu - t\sigma \leq r_w \leq \mu + t\sigma$.
- The input $x = C_n^1(i)$ belongs to class "buy", if there exists $w \in [i+1, i+2, ..., i+k]$ such that

$r_w > \mu + t\sigma$, and $\mu - t\sigma \leq r_v \leq \mu + t\sigma,\ \forall\ v < w$.

- The input $x = C_n^1(i)$ belongs to class "sell", if there exists $w \in [i+1, i+2, ..., i+k]$ such that $r_w < \mu - t\sigma$, and $\mu - t\sigma \leq r_v \leq \mu + t\sigma,\ \forall\ v < w$.

Now substituting in (I) and assuming independence of the values in $R_k^2(i)$, we have,

$$\begin{aligned}
P[x \in H] = \frac{1}{3} &\implies P\left[\mu - t\sigma \leq r \leq \mu + t\sigma,\ \forall\ r \in R_k^2(i)\right] = \frac{1}{3} \\
&\implies \prod_{w=i+1}^{k+1} P[\mu - t\sigma \leq r_w \leq \mu + t\sigma] = \frac{1}{3} \\
&\implies \prod_{w=i+1}^{k+1} P[-t \leq \frac{r_w - \mu}{\sigma} \leq t] = \frac{1}{3} \\
&\implies \prod_{w=i+1}^{k+1} [2\phi(t) - 1] = \frac{1}{3},\ \because r_w \sim N(\mu, \sigma) \\
&\implies [2\phi(t) - 1]^k = \frac{1}{3} \implies t = \phi^{-1}\left[\frac{\frac{1}{3}^{\frac{1}{k}} + 1}{2}\right]
\end{aligned}$$

For $i + 1 \leq w \leq k + 1$, let B_w be the event such that $r_w > \mu + t\sigma$, and $\mu - t\sigma \leq r_v \leq \mu + t\sigma,\ \forall\ v < w$. We will keep note of the fact that r_w and r_v are all i.i.d copies of variable following $N(\mu, \sigma)$. Now,

$$\begin{aligned}
P\left[B_w\right] &= P\left[r_w > \mu + t\sigma,\ \ \mu - t\sigma \leq r_v \leq \mu + t\sigma,\ \forall\ v < w\right] \\
&= \left(\prod_{v=i+1}^{w-1} P[\mu - t\sigma \leq r_v \leq \mu + t\sigma]\right) P[r_w > \mu + t\sigma] \\
&= [1 - \phi(t)][2\phi(t) - 1]^{w-i-1} = \left(\frac{1-q}{2}\right) q^{w-i-1},\ \text{where } q = 2\phi(t) - 1
\end{aligned}$$

Now it can be easily verified that,

$$\begin{aligned}
P[x \in B] = P\left[\bigcup_{w=i+1}^{k+1} B_w\right] &= \sum_{w=i+1}^{k+1} \left(\frac{1-q}{2}\right) q^{w-i-1} \\
&= \left(\frac{1-q}{2}\right) \sum_{j=1}^{k} q^{j-1} = \frac{1-q}{2} \frac{1-q^k}{1-q} \\
&= \frac{1}{2}\left[1 - (2\phi(t) - 1)^k\right] = \frac{1}{2}\left(1 - \left[2 \cdot \frac{\frac{1}{3}^{\frac{1}{k}} + 1}{2} - 1\right]^k\right) \\
&= \frac{2/3}{2} = \frac{1}{3}
\end{aligned}$$

So, $P[x \in S] = 1 - P[x \in B] - P[x \in H] = 1 - \frac{2}{3} = \frac{1}{3} \implies (I)$ holds.

Or, for $t = \phi^{-1}\left[\frac{\frac{1}{3}^{\frac{1}{k}} + 1}{2}\right]$, $P[x \in S] = P[x \in B] = P[x \in H] = \frac{1}{3}$.

Similarly the other inputs can be obtained by iterating i to an index k less than the length of R. In practice, we iterate over the vector $C_k^2(i)$. For a fixed i if $|\left(\frac{c_{i+w} - c_i}{c_i}\right)| > t\sigma$ for some $w \in \{1, ..., k\}$, we label the class

as Buy or Sell depending on the sign of $\left(\frac{c_{i+w}-c_i}{c_i}\right)$ and stop the loop. If for all values $\forall\ w \in \{1, ..., k\}$ no such value is encountered, the input vector is labelled as Hold.

A quick review of the mentioned approach might kindle the immediate question, **why are n and l taken to be different values?** This can be answered in two parts.

- We need the l value large to ensure a normal distribution of $R^1_l(i)$. Also if k and l are approximately of equal magnitude, our assumption of $R^1_{l+k}(i+k)$ and $R^1_l(i)$ being "similarly" distributed would be severely weakened. Hence, we need a large value for l, $l >> k$.
- A comparatively smaller value for n can be attributed to two further reasons. First, we must not feed values which are too old to the machine, as dependency is likely to decrease as we go further back in time. We are still claiming that the large chunk of $l + k$ values appear to be similarly distributed as the chunk without the k values; because a normal distribution is uniquely determined by its mean and variance, and these quantities corresponding to both the chunks are approximately equal. Yet, in a highly zoomed in version, for a particular data point in the k future values to be dependent on values at instances which are l time units back is highly unlikely. So instead of looking back l units, we are good with looking back n units.
  Secondly, increasing the value of n by 1 increases the length of the input vector by 4. Considering the large amount of data the machine needs to be trained with, input vectors of large sizes would require strong computational capacity. Hence the convention used is $l >> n > k$.

3.2. Preprocessing

Once the data points are labelled, we divide the O,H,L,C values into vectors of length n moving from the index l and ending at $len(C) - k$. Each vector is two dimensional with n entries, with each entry being an O,H,L,C quadruple. Now, we take **logarithm** of each value in the input vectors; and to normalize the input, we subtract the log of the first opening price from each value. We drop the first input value since it is always 0. At last we flatten this two dimensional vector into a 1-D vector of length $4n$ and feed it to machine.

A neural network always constructs an "additive" model taking weighted combinations of inputs and passing them through the activation function. Such additive systems for predicting stock returns have been extensively modelled and tested. However in this project we have attempted to make use of the machine to unearth any "multiplicative" relation of raw inputs and their powers to the future prices. The preprocessing is based on the simple idea: $\log a + \log b = log(ab)$, or in extension,

$$\sum_{i=0}^{b} w_i \log(a_i) = log\left(\prod_{i=0}^{b} a_i^{w_i}\right)$$

3.3. Train Test Split

The temporal nature of the stock price data makes it necessary to make sure that there is no look ahead bias in terms of training and testing data. Look ahead bias is the use of data for creating a model, first principles based or empirical estimator based, that was trained on data that would not be available in real time scenario. An example of this would be training on data from time t_2 and testing on data from time t_1 when data from time t_2 would have only been available after t_1. This might give a model oracular knowledge which in the real life scenario it would not have. This condition is very specific to temporal systems in which it is quite possible that the model might be optimized in such a way that it could use this oracular knowledge and give a very high accuracy on test data but would fail to provide similar results on real time data.

Hence, special consideration was given to make sure that the data that model is being trained on does not have any lookahead bias. To recreate the real world scenario in which the model would be deployed, the available data was first sorted in ascending order of the time index. It was decided that about two third part of the available data would be used for training the model and the rest of the data would be used for testing the model. For the specific case of Nifty 1 Minute data, it meant that data from beginning of 2015 till end of first quarter of 2018 was used as training data whereas data from second quarter of 2018 to second quarter of 2019 was used to test the accuracy of the model. Essentially, the training data was for 9 quarters and the testing data was for 5 quarters. The fact that the model was not retrained on the data that became available as time passed by, should also be mentioned as this means that if the model was retrained on this newly available data, the model could have provided better results.

3.4. Empirical Estimator Models

3.4.1. Neural Network Models

We used a fully connected multi layered perceptron neural network which was implemented using the Dense Layer implementation from Keras Python Library. The model consisted of one input layer, 7 hidden layers and a final output layer. The size of input layer was dependent on the lookback. The input layer had $4n - 1$ inputs each of which were directly fed the preprocessed input vector with normalized log values. The number of perceptrons in the hidden layers was also dependent on value of lookback. The first and the second hidden layer had $100n$ perceptrons. The third, fourth, fifth, sixth and seventh had $60n$, $40n$, $20n$, $10n$ and n perceptrons respectively. The final hidden layer was followed by a output layer with 3 perceptrons. The activation function used for hidden layers was hyperbolic tan and the activation function used for the output layer was softmax. Hence this model adapted dynamically to changes in size of the lookback vector. The loss function used for this model was categorical crossentropy. It should also be mentioned that there was dropout layer after each hidden layer with 25% of the perceptrons being dropped in a given epoch.

3.4.2. Random Forest Model

For this model an ensemble of Decision Trees was used. This is commonly known as Random Forest Classifier. We used the Random Forest Classifier implementation from SciKit Learn python library. The number of base estimators for the Random Forest Classifier were 250. GINI impurity index was used as splitting criteria for the decision trees. The training data for each of the decision trees was sampled with replacement from the original training data set which did not have any lookahead bias as explained in section 3.3. An input vector to the random forest classifier was of size $4n - 1$. These $4n - 1$ values consisted of the normalized log values that were obtained in the preprocessing step.

4. Results

4.1. Neural Network Models

After 50 epochs of training the model with the training data, there was no reduction in the loss function and the accuracy being reported in training data was stuck at 33.334% which means that the model was as good as random guessing. We tried different values of *lookback*, *lookahead* and *std_dev_lookback* but none of the values we tried provided a different result.

4.2. Random Forest Classifier Model

After the failure of the Neural Network Model, we tried fitting the data to a random forest classifier and this lead to an accuracy of 39.8% on the testing dataset which suggests that it is somewhat better than random guessing a three class problem which would lead to an accuracy of 33%. It also lead to a win rate greater than 50% which makes us believe that a trading strategy based on this model could have positive expectation.

$$\text{Buy win rate} = \frac{c[\text{yh=buy} \mid \text{y=buy}]}{c[\text{yh=buy} \mid \text{y=buy}] + c[\text{yh=buy} \mid \text{y=sell}]}$$

$$\text{Sell win rate} = \frac{c[\text{yh=sell} \mid \text{sell}]}{c[\text{yh=sell} \mid \text{y=buy}] + c[\text{yh=sell} \mid \text{y=sell}]}$$

$$\text{Total win rate} = \frac{c[\text{yh=sell} \mid \text{sell}] + c[\text{yh=buy} \mid \text{y=buy}]}{c[\text{yh=sell} \mid \text{y=buy}] + c[\text{yh=sell} \mid \text{y=sell}] + c[\text{yh=buy} \mid \text{y=buy}] + c[\text{yh=buy} \mid \text{y=sell}]}$$

Quarter	*Buy Win Rate*	*Sell Win Rate*	*Total Win Rate*
April - June 2018	0.5430950997482084	0.5039613526570048	0.5235055136390018
July - September 2018	0.5609452736318408	0.5068600548804391	0.5331452750352609
October - December 2018	0.5492300513299113	0.4905541561712846	0.5200688522024881
January - March 2019	0.5204222914503288	0.4949584291526623	0.5078295862129297
April - June 2019	0.5200279867063144	0.516257225433526	0.5181729316626678

Win rate on test data with lookback = 30, lookahead = 10 and std_dev_lookback = 250

We believe the reason for the Random Forest Classifier's success is the fact that it produces weak estimators which have uncorrelated error. This is a good fit for the problem at hand because of the fact that market behaves differently in different market regimes and hence a technical analysis technique that works in one regime might not work in another. This different behaviour can be captured by different base estimators which constitute the random forest. We also believe that this is the reason why the neural network implementation might be failing.

5. Further Work

5.1. Backtesting a trading strategy built around the model

A win rate above 50% shows that a strategy built around the model might be profitable. To confirm this hypothesis an actual trading model with this model at it's core needs to be backtested. Backtesting is the process of testing a trading strategy against detailed historical data so that actual performance statistics like compounded yearly returns of the trading strategy can be calculated along with many other crucial statistics like maximum drawdown, volatility of the portfolio, etc. These statistics give us crucial insight into how the trading strategy would perform in real life.

Basically a trading strategy defines the rules of how the insights of the model get converted to positions that the investor takes. This is fairly complicated as it involves risk management and order execution. Risk management might put limits on how big of a position a single insight can lead to. This limitation might act as a double edged sword which might stop the investor from experiencing big losses but might also limit his ability to collect big profits when he is actually right.

Another thing that the trading strategy has to define is what kind of orders to place, market or limit. Market orders result in immediate execution of the order and the investor gets to enter the market immediately but this might lead to sub optimal pricing as the orders are executed at the best possible prices available at the time of order placement. It is quite possible that if the investor had waited for sometime, he would have achieved a more optimal pricing. Limit orders on the other hand allow the investor to publish the price that he is willing to buy or sell at and it is upto the other market participants whether they take this offer. This leads to optimal pricing but might result in failed execution if there were no market participants who were willing to take the other side of the trade at the price that the investor published.

Only when all of these principles are well defined can a strategy be backtested and actual numbers about the performance of the strategy be achieved.

5.2. Custom Loss Function

While training a neural network with more than two classes an usual approach used is one-hot-encoding. This is done when all the classes are independent of each other and all pairs of classes are "equidistant". However, in this specific project there is a scope of dropping this constraint. In fact, it is better to reconsider this constraint as in practice, the case of a data point belonging to "sell" class being predicted as "buy' is much worse a situation than "sell" being predicted as "hold", and the vice versa. In the latter we do lose out on an opportunity of a potential profit, but in the first we incur a loss. Thus, class buy and sell are farther apart than buy and hold, or sell and hold. So, we can define an encoding that captures these notions of distances between the classes, and subsequently define suitable loss and activation functions which are compatible with the encoding.

We will encode the classes $\{B, H, S\}$ as $\{-1, 0, 1\}$. Such an encoding preserves the desired notion of distances mentioned in the previous paragraph. We would like to assign importance to the sign of the output, so we would adapt the Hinge loss to make it compatible with the suggested encoding.

Let $\hat{f}$ be the predictor, and y be the desired output for input x. Let $\hat{y} = \hat{f}(x)$. Then the loss function L is defined as,

$$L(y, \hat{y}) = \max\left[0,\ \ |\hat{y}| \cdot (1 - |y|) + |y| \cdot (1 - y \cdot \hat{y})\right]$$

The above loss function simplifies to the case wise function,

$$L(y, \hat{y}) = \begin{cases} |\hat{y}|, \text{ if } y = 0 \\ \max\left(0,\ \ 1 - y \cdot \hat{y}\right), \text{ if } y = \pm 1 \end{cases}$$

We can easily observe that the loss function assigns a hinge loss if the classes are ± 1, thereby taking into the account the "sign" of the output - i.e. whether the output is on the buy-hold side, or the sell-hold side. The function assigns a absolute difference loss when the class is 0 (hold), thereby maintaining the equality between class-hold and sell-hold distances. We believe this might improve the performance of the neural network implementation and might lead to results similar to those reported by Jasemi, Kimiagari and Memariani.

6. Conclusion

The labelling method proposed tries to create a dataset with almost equal representation from all the classes and this provides us with an unbiased dataset which might help with better training of emperical estimator models. It is quite clear that the random forest model does show some promise in terms of win rate. To find out whether the model can provide consistent significant positive returns and beat the market, a strategy needs to be defined around the model. Further another try needs to be made with neural networks as neural network architecture seem like the perfect fit for such a dynamic non linear classification problem. The modified hinge loss function proposed in section 5.2 might help in the case of neural networks and might lead to an increase in the accuracy of the classification problem.

References

Law, R., & Au, N. (1999). A NN model to forecast Japanese demand for travel to Hong Kong. Tourism Management, 20, 8997.

Lee, K. H., & Jo, G. S. (1999). Expert system for predicting stock market timing using a candlestick chart. Expert Systems with Applications, 16, 357364.

Marshall, B. R., Young, M. R., & Rose, L. C. (2006). Candlestick technical trading strategies: Can they create value for investors? Journal of Banking and Finance, 30, 23032323.

Milad Jasemi, Ali M. Kimiagari , A. Memariani (2010). A modern neural network model to do stock market timing on the basis of the ancient investment technique of Japanese Candlestick. Expert Systems with Applications, 38, 3884-3890.

Schoeneburg, E. (1990). Stock price prediction using neural networks: A project report. Neurocomputing, 2, 1727.

Winkler, R. L. (1989). Combining forecasts: A philosophical basis and some current issues. International Journal of Forecasting, 5(4), 605609.

Yao, J., Tan, L. C., & Poh, H. (1999). Neural networks for technical analysis: A study on KLCI. International Journal of Theoretical and Applied Finance, 2, 221241.