IMPROVING DIRECTIONAL PREDICTION OF IBM STOCK USING STACKED ENSEMBLING

by

NIKHIL RANJAN

Under the Direction of Yuan Ke

ABSTRACT

Machine Learning is paving its way into every domain of our lives and especially in finance and economics. In this Thesis, we used commonly available ensemble methods and compared it to stacked ensembles which involve creating a deep neural network for a meta learner. This meta learner will take inputs and learn from our regressive models to try to eliminate any innate bias. We have 9 models which are made from Linear Regression, Lasso, Multi-Layer Perceptron, Long-Term Short-Term Memory (LSTM) and Bidirectional LSTM which feed into a single meta-learner. Upon testing with IBM stock, we have found out that using a well-tuned stacked ensemble upon these models has 1.1% better directional accuracy while compared to other ensemble methods.

INDEX WORDS: Stacked Ensembling, Long-Term Short Term Memory (LSTM), Deep Neural Networks (DNN), Stock Market Prediction, IBM

IMPROVING DIRECTIONAL PREDICTION OF IBM STOCK USING STACKED ENSEMBLING

by

NIKHIL RANJAN

B.Tech., Manipal University, July 2018

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

MASTER OF SCIENCE

ATHENS, GEORGIA

2022

IMPROVING DIRECTIONAL PREDICTION OF IBM STOCK USING STACKED ENSEMBLING

by

NIKHIL RANJAN

Major Professor: Yuan Ke

Committee: Frederick Maier
Ismailcem Budak Arpinar

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
August 2022

DEDICATION

I dedicate this to my loving parents who have supported me unconditionally throughout my life.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Yuan Ke, for his constant support and for being patient with me. This thesis would not have been possible without his inputs at crucial moments, and his ability to steer me into the right path. I would like to also thank Dr. Frederick Maier and Dr. Budak Arpinar for agreeing to be on my committee. Additionally, I would like to that Dr. Khaled Rasheed who motivated me to keep going when I approached him with doubts related to the feasibility of my thesis. I would also like to thank Dr. Carolina Salge for giving me the push to start my thesis and always helping me balance my research assistant work and thesis. Last but not least, I must thank all my friends and family for the amazing memories and supporting me through the journey of my life. A special thanks to my friend Ciara H. Page for proofreading my thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The stock market is one of the primary ways to obtain fractional ownership of a publicly traded corporation. The word "stock" in stock markets refers to a financial security which represents partial ownership. Stocks are mostly bought and sold on stock exchanges at a dynamic rate. Since a stock represents partial ownership of a company, its price is determined by demand and supply. Stock markets are an integral part of any free economy because they give individuals the freedom to own and invest in a corporation. This allows the corporation to raise capital while also allowing the individual to increase their personal wealth.

A stock price prediction is an estimate of the stock price in the near future. The problem here is that the stock market is affected not only by the performance of the company, but by an uncountable number of factors in the world. There is also the theory of the efficient market hypothesis, which should be taken into consideration (Fama, 1965a). It implies that the price of stocks is the reflection of all publicly known information and expectations (Fama, 1965a). This is a well-established theory, and it entails that no one can beat the market consistently using the same strategy as the market follows a "random walk". A human decision maker cannot effectively work with complex information patterns and convert them into profitable investment decisions. Therefore, better investment decisions can be made by machines that can handle complex decision making (Felsen, 1976).

To create more robust regressors/classifiers, a common method used to remove inherent assumptions/biases of models is to average out or take a vote between multiple models. This is called as ensembling and it generally utilizes a latent space to allow for better predictions than just a single model (Dietterich, 2000). Primarily, we will use the ability of neural networks to be 'universal approximaters' (Hornik et al., 1989) to create an ensemble that takes input from other regressors and makes better predictions. Tradi-

tional statistically based methods unfortunately need to make assumptions about time series data and are mostly linear in their approach to predicting the stock market (Hill et al., 1996).

In this thesis, we will be using IBM as an example stock to extensively analyze and test multiple deep learning and statistical based models and ultimately evaluate the ensemble methods used. This paper focuses on showcasing the improvement attained by combining stacked ensembling in comparison to more popular regressive ensembling methods like averaging or bagging. The main objective of this paper is to popularize and showcase the performance of stacked ensembling compared to its more popular counterparts i.e. Bagging and Boosting.

This thesis is organized into 4 sections with an appendix. Chapter 2 contains all the necessary background information to understand the results generated in chapter 3 and 4. It contains information related to Neural Networks, Stock Markets and Metrics used to evaluate our results/models. We will be discussing the fundamental and technical approaches to stock market predictions. While this thesis focuses completely on the technical approach , it acknowledges that a combination of fundamental and technical approach would be the best approach for a real world scenario. This thesis purely focuses on improving the technical approach and leaves the incorporation of fundamental approaches to future research. This thesis utilizes the past daily log returns and uses that knowledge along with volume to predict 1-day ahead stock prices. For this purpose, we will be using statistical and deep neural networks which can learn the trends for 1-day ahead prediction and then combine them with different ensembling methods. Chapter 3 covers the details about the setup of the experiment and hyper-parameter tuning. The data was collected from yahoo finance for the dates 1970-Jan-1 to 31-Dec-2021, a total of 5283 trading days. Before jumping into model creation, let us see the three metrics used for evaluating the models and end result. RMSE (Root Mean Square Error), MAPE(Mean Absolute Percentage Error and MDA (Mean Directional Accuracy. Together, all three of the metrics allows us to get a complete picture of the model's performance. With the evaluation metrics ready, we will be able to tune hyper-parameters of our models after pre-processing our dataset. The models that were used are Linear Regression, Lasso, Multi-Layer Neural Network, LSTM and Bi-LSTM. Apart from their own hyper-parameters, window size was also considered as part of that list. After getting the best models using a grid search, a multi-layer perceptron

was trained to be a stacked ensemble. The best configuration for the same was found through validation. The experiment results were obtained from 2018-2021 for 1-day ahead prediction for a duration of 4 years. Chapter 4 contains the results of the experiments. We analyze the performance of every model separately and we found that Linear Regression with a window size of 4 had the best RMSE and MDA even beating the performance of complex Bi-LSTM models. After Ensembling, the stacked ensemble had an MDA of 76.86% which is 1.1% better than other bagging and boosting methods. In Chapter 5, we conclude the findings that showcase the improvement in directional accuracy obtained by using Stacked Ensembling. In our literature study, we couldn't find many projects which used anything else apart from bagging and boosting. We hope this project will encourage more people to use stacked ensembling for combining heterogeneous models. Additionally, we will discuss a few ways in which this paper can be improved upon.

CHAPTER 2

LITERATURE REVIEW AND BACKGROUND

§ 2.1 Market Prediction Approaches

Stock market prediction techniques vary greatly in theory and in their basis, but most of them can be classified in either of these 2 categories (*The encyclopedia of stock market techniques.* 1963):

1. Fundamental Approach

   This approach relies on finding the intrinsic value of a company/asset to determine its optimal price. We know that since markets follow a 'random walk' (Fama, 1965b), the fundamental approach does not make any statistical prediction based on previous stock price but rather focuses on finding the expected intrinsic value (Felsen, 1976). After finding this, it is just a matter of finding the appropriate methods/rules to compare the current and predicted intrinsic value to create an investment strategy. Intrinsic value is an important concept for financial securities analysis. Securities that meet stringent safety tests and appear to be selling well below their intrinsic value can often be profitable table investments. (Graham et al., 1934)

2. Technical Approach

   Technical approaches, as opposed to fundamental approaches, pay more attention to the state and value of the market than the company itself. The focus here is on finding the momentum or patterns in the market to predict the price. Decisions to buy and sell stocks are based on current data related to the stock, such as past prices and volume. This paper relies on the technical approach to predict stock markets and, as expected, technical approaches do not put a lot of faith in the "Random Walk Hypothesis". Lo and MacKinlay (2015) have rejected the random walk hypothesis based on a simple volatility-based specification test for weekly returns during September 6, 1962, to December 26,

1985. This was also later supported by Jegadeesh, 1990 where they showed that stocks show strong evidence of predictive behaviour.

The reality is that each of the above methods have their pros and cons. Although this paper focuses on the technical approach, advancements in either domain should not discourage further research. Both of these methods while opposing, have proven to be good indicators of the stock market. Purely technical analysis misses the real-world context while a purely fundamental approach will fail to capture important data related to market momentum and indicators. The best and perfect model prediction methods would incorporate both methods, but that is extremely complex and beyond the scope of this thesis. In this section, we will discuss the model of the stock market, a few statistical methods and then deep neural networks.

§ 2.2 IBM-Past Research

IBM as a financial security has existed since 1911, and the daily data for the same is available starting from 1/1/1970. It has lived through multiple major market events like the dot-com bubble, multiple global wars, financial crisis in 2007-2008, COVID-19 etc. Therefore it contains a wealth of knowledge ready to be extracted because it is not a volatile stock either. Due to this and many more advantages, price returns of IBM has been extensively studied by many researches throughout the years. White (1988) was one of the first researchers to look into this problem. He termed his neural network at the time as "not a money making machine" but he did offer deeper insights into the problem. Gorenc Novak and Velušček (2016) showcased Linear SVM being used to predict 'daily high' prices with a directional accuracy of $61.16\% \pm 10.68\%$. This work highlighted multiple statistical models and gives us a good baseline to work against. Khan et al. (2020) in their paper showcased a deep learning and fundamental approach to predict IBM stock on a 10-day ahead prediction. They got a 75.16% and 80.53% directional accuracy while using financial news and social media sentiment respectively. Lu et al. (2021) used a CNN-BiLSTM-AM network to predict daily IBM stock prices and showed that deep neural networks have a very high value R^2, thus making such models much better for predictions, as their mean error would always decrease as the RMSE is reduced. Ding et al. (2015) developed an event-driven stock prediction system using

a fundamental approach. They solidified the idea of using deep learning in modern stock prediction systems, be it for fundamental or technical approach.

§ 2.3 Stock Market Model

One of the most popular models for stock market modelling is the capital asset price model made by Sharpe (1964). It entails that returns can be modelled as a linear model of past returns. 2.1 represent current return r_t as a linear function of past returns $r_{t-1}, r_{t-2} \dots r_p$. $w = (w_0, w_1 \dots w_p$ are the weights associated to the returns and ϵ_t is the error term.

$$r_p = w_0 + w_1 r_{t-1} + \dots + w_p r_{t-p} + \epsilon_t \tag{2.1}$$

White (1988) in their paper showed that stock returns can be modeled as a linear relationship of past log returns and an error term. We can extend this formula from returns as shown in 2.1 to price predictions and an error term using 2.2 (White, 1988).

$$\begin{aligned} &Y_t = w_0 + w_1 * Y_{t-1} + ... + w_p * Y_{t-N} + \epsilon_t \\ &t = 1, 2, 3 \dots N \end{aligned} \tag{2.2}$$

The terms $w_0, w_1 ...$ are the set of weights and ϵ_t is the error / randomness in time t. Y_t refers to the price of the stock at time t. All predictive/regression methods should optimize on this and do their best to reduce ϵ_t while adhering to the formula.

§ 2.4 Linear Regression

LR (Linear Regression) or OLS (Ordinary Least Squares) regression is one of the most popular statistical methods for regression which is on par with modern state-of-the-art methods while still being simple to interpret and explain. A linear relationship between a set of inputs and output is defined by 2.3 where X is the input point of size $n * 1$ and n is the number of dimensions of that input. β_0 is the bias constant, and β is an $1 * n$ matrix of constants.

$$Y = \beta_0 + \beta X \tag{2.3}$$

LR involves minimizing the least squares error of approximating inputs and outputs as a linear relationship. Assuming Y is the actual output and $\hat{Y}$ is our prediction obtained from 2.3, the least squares error can be defined as shown in 2.4. The variables we need to optimize are β_0 and β to reduce value of OLS.

$$OLS = argmin_{\beta_0,\beta} \sum_i (\epsilon_i)^2 = argmin_{\beta_0,\beta} \sum_i (y_i - (\beta_0 + \beta x_i))^2 \tag{2.4}$$

2.3 can be rewritten in matrix form for a set of points with inputs written in matrix X with size nxp where p is the number of dimensions.Outputs in matrix Y with size $nx1$ if we add a unit column matrix to X and combine β_0 and β to a single term $\overrightarrow{\beta}$. Upon solving it, we will get 2.5 which is the generally accepted equation to solve Linear Regression.

$$\overrightarrow{\beta} = (X^T X)^{-1} X^T Y \tag{2.5}$$

We can see that 2.4 is similar to 2.2 however, linear regression works on 4 simple but difficult to follow assumptions.

1. Linearity: It assumes a linear relationship between the input and output.
2. Independence: Assumes that there is no correlation between residuals of consecutive points.
3. Homoscedasticity: Residuals have constant variance.
4. Normality: Residuals are normally distributed.

Violation of either one of those rules can lead to linear regression failing for the data and we know that in the real world it is easy to violate either of these assumptions. However, despite this, linear regression is an excellent baseline to compare other models because of its simplicity and interpretability.

§ 2.5 Lasso

While Linear Regression is effective, it suffers from the absence of regularization penalties. This means that the coefficients can get very close to zero or explode depending on the data. This introduces problems in the form of noisy predictions and thus Hoerl and Kennard (1970) created ridge regression which added a penalty on the coefficients. This was later improved by Tibshirani (1996) with the creation of Lasso regression. Lasso reduces the coefficients to zero if it falls below a certain threshold, allowing for feature selection while removing noise from the model. The package Scikit-Learn (Pedregosa et al., 2011) which forms a major part of our thesis also optimizes and implements lasso. It uses 2.6 for the same.

$$argmin \frac{1}{(2 * n)} * ||Y - X\overrightarrow{\beta}||_2^2 + \lambda * ||\overrightarrow{\beta}||_1 \tag{2.6}$$

§ 2.6 Neurons and Neural Networks

Neural net theory is based on the model created by McCulloch and Pitts, 1943. It consists of imagining the neuron with multiple inputs X_i and weights W_i. They are multiplied element-wise, and the sum of their products is passed to an activation function called $g(X)$ where X is the sum of products of the inputs and weights (each input has a weight associated with it). Refer to 2.7 for evaluation X, x_i refers to the inputs of the neuron, and w_i is the weight of each input.

$$X = \sum_{i=0}^{N} x_i * w_i \tag{2.7}$$

Given an N-dimensional space, a single neuron can create a decision boundary which resembles a hyperplane. This can be easily visualized in a smaller dimension and generalized to a higher dimension. Assume that a neuron has a step activation function and we combine it with the formula in 2.7 which has 2 inputs. The step activation function is defined by 2.8. The final equation upon combining two inputs ($N = 2$) with a step-activation function will be written in the form shown in 2.9.

$$f(x) = \begin{cases} 0 & x \leq 0 \\ 1 & \geq 0 \end{cases} \tag{2.8}$$

$$y = \begin{cases} 0 & w_1 * x_1 + w_2 * x_2 \leq 0 \\ 1 & w_1 * x_1 + w_2 * x_2 \geq 0 \end{cases} \tag{2.9}$$

Equation $w_1 * x_1 + w_2 * x_2$ represents a line in a 2-D space (the independent components of such a space are x_1 and x_2). The line is a hyperplane of a 2-D space and we can see that extending the same logic to the N-dimensional space will give us the equation $w_1 * x_1 + w_2 * x_2 \cdots + w_N * x_N$, which would be a hyperplane in that space.

This basic idea of creating a trainable decision boundary by adjusting the weights of a neuron allows us to stack multiple neurons and create densely connected layers to yield complex decision boundaries.

The concept of Neural Networks comes from biology and the concept of neurons. An artificial neuron is a combination of input, learning weights, and an activation function. McCulloch and Pitts, 1943 were the first people to model a biological neuron into a 'perceptron'. Hornik et al., 1989 in their paper "Multilayer feedforward networks are universal approximators" establish the fact that simple feedforward networks can approximate any measurable function. The expectation is that a neural network will be able to past stock prices and approximate 2.2 while minimizing the randomness.

§ 2.7 Training a Neuron and Neural Networks

Rumelhart et al. (1986) detailed out backpropagation as a way to train neural network like structures. This was done by propagating the error evaluated at the output layer and moving it across the hidden layers going towards the input layer. In summary, we need a differentiable error/loss function to back-propagate the errors. One of the most common error functions is the mean square error function shown in 2.10.

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2 \tag{2.10}$$

Referencing the original paper, we know that the change in weights should be modeled as shown in the function 2.11 w.r.t. error using gradient descent.

$$\triangle W = -\epsilon \frac{\partial J}{\partial W} \tag{2.11}$$

This means that our new weight after one learning iteration would be shown in 2.12.

$$W_{t+1} = Wt - \epsilon \frac{\partial J}{\partial W} \tag{2.12}$$

Performing this same operation for all layers will help the neural network learn the relationship between input and output over multiple iterations. This forms the basis for almost all modern back-propagated neural network learning.

§ 2.8 Neural Network Layers

A popular question when learning neural networks is asking a single neuron to learn a 2-input XOR -logic. The single neuron always fails at this because there is no way to linearly separate the outputs into True and False. This is where stacking neurons proves to be the best way to learn such a distribution because multiple decision boundaries help in creating complex and multiple hyperplanes that can help in solving more difficult problems. Cybenko, 1989 concluded that "finite superpositions of a fixed univariate function that is discriminatory can uniformly approximate any continuous function of n real variables with support in the unit hypercube. Continuous sigmoidal functions of the type commonly used in real-valued neural network theory are discriminatory.". This means that multiple layers of neurons can approximate any continuous function (Cybenko, 1989).

§ 2.9 Recurrent Neural Networks (RNN)

The neural network designs we have looked at until now have been completely feed-forward i.e. the output at any given instant is not affected by the outputs of the past sample. Although this may be sufficient for most tasks, time-dependent data can be predicted better with feedback loops. RNN has the capability to

dynamically incorporate past experience due to internal recurrence, making it extremely powerful (Saad et al., 1998).

§ 2.9.1 LSTM

A commonly used and popular RNN is the Long term-Short Term Neural Network (LSTM). Introduced by Hochreiter and Schmidhuber (1997), an LSTM network consists of multiple LSTM cells, each of them containing a Forget gate, Input gate, Output gate and an Hidden State. In recurrent learning, error signals may vanish or explode (Hochreiter & Schmidhuber, 1997), LSTMs circumvent this issue by having a long-term memory that is not backpropagated but rather added or removed directly. The structure of the LSTM can be seen in figure 2.3. Due to the complexities, training LSTM networks is time consuming however they are known to perform better than MLP/DNN networks for stock price forecasting (Shah et al., 2018).

The Forget gate

This part of the cell is responsible for deciding which long-term memories to forget in the cell state C_t with the help of 2.13 which computes the multiplicative factor, and 2.14, which refers to the new long-term information state.

$$f_t = \sigma(w_f[h_{t-1}, X_t] + b_f) \tag{2.13}$$

$$c_t^f = c_{t-1} * f_t \tag{2.14}$$

We can see that C_t is multiplied by f_t which changes the long term information held by the cell. The inputs of f_t are the hidden (short-term) state (h_{t-1}) and the input (X_t), therefore the cell has the capability to decide which information to retain based on recent and new data.

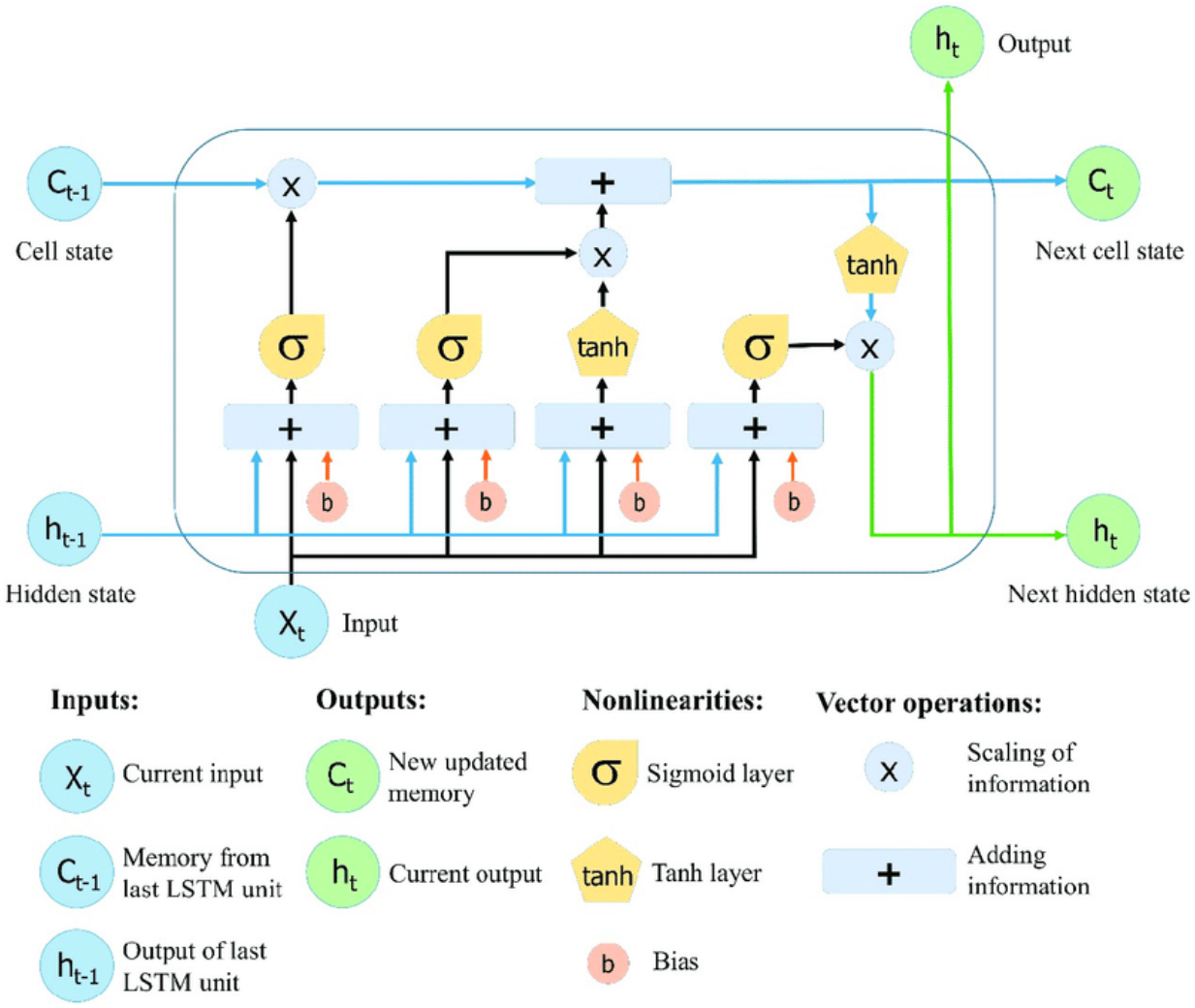

Figure 2.1: LSTM Structure diagram taken from Le et al., 2019, uploaded by the co-author Giha Lee.

The Input Gate

While the Forget gate was important in forgetting long-term information, the input gate is responsible for adding information to the long-term state of the cell. It represents the addition of the part of current information added to long-term memory. 2.15 represents the output of the neural layer responsible for combining the hidden and input states. 2.16 details the addition of data from input and short term memory to the long-term memory.

$$i_t^f = \sigma(w_i[h_{t-1}, X_t] + b_i) \tag{2.15}$$

$$c_t = f_t * c_{t-1} + i_t * tanh(w_c[h_{t-1}, X_t] + b_c) \tag{2.16}$$

The Output Gate

With our long-term information set, the only thing left to do is pass on C_t to the next cell and compute H_t for the next cell. This is shown by 2.18. The gate has two outputs and helps to further propagate the information without blowing up the error gradient (Hochreiter & Schmidhuber, 1997).

$$o_t = \sigma(w_o[h_{t-1}, X_t] + b_o) \tag{2.17}$$

$$h_t = o_t * tanh(c_t) \tag{2.18}$$

§ 2.9.2 Bi-LSTM

Schuster and Paliwal (1997) introduced the idea of a Bi-Directional Recurrent Neural Network. A bidirectional RNN allows information to be learned in both positive and negative time direction, thus giving better performance in Regression and Classification tasks (Schuster & Paliwal, 1997).

Bidirectional long-short-term memory is a type of recurrent neural network that allows information to flow from past to present and present to the past. Since the input runs in both directions, it is different from a regular LSTM which only has information flow in one direction. With the information flowing in both directions, it is able to better capture any patterns in data for regression or classification. Figure 2.2 represents a 2 layer Bi-LSTM structure (taken from Zhang et al. (2018)).

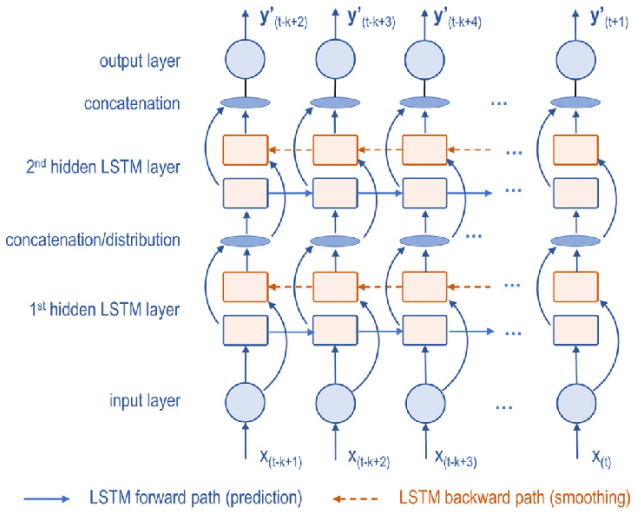

Figure 2.2: 2 Layer Bi-LSTM structure diagram taken from Zhang et al., 2018, uploaded by the co-author Robert X. Gao.

§ 2.10 Ensembling

Ensemble is a set of individually trained classifiers/regressors whose predictions are combined for a final prediction (Opitz & Maclin, 1999). There three most common methods are Bagging (Breiman, 1996), Boosting (Freund and Schapire, 1997;Drucker et al., 1994) and Stacking (Wolpert, 1992).

§ 2.10.1 Bagging

Bagging uses random samples of the training set to train a set of classifiers. The training set for every classifier is made by drawing N random samples with replacement. This ensures that any bias in the data does not impact overall performance of the ensemble (Breiman, 1996). Breiman (1996) also found out that bagging helps in improving results in 'unstable' classifiers/regressors where tiny changes in input data or outliers can radically change the results. Bagging Regressors involves taking an average of all the regressors in that ensemble. It is different than a simple average because the training data for each regressor will be different.

§ 2.10.2 Boosting

In contrast to Bagging, Boosting works by creating a sequence of classifiers/regressors whose training set constitute the errors made by the previous model . This allows data points which are usually evaluated incorrectly by a model to be trained on the next model which leads to better performance on for the whole system (Freund and Schapire, 1997; Drucker et al., 1994; Opitz and Maclin, 1999).

One of the best boosting algorithms is Ada-Boosting (Freund & Schapire, 1997) and it can be used for performing regression too. Drucker (1997) developed the 'AdaBoost.R2' algorithm which is currently used by Scikit-Learn and it utilizes regression trees for boosting.

§ 2.10.3 Stacked Ensembling

Stacked generalization / ensembling works by deducing the classifiers / regressors biases in relation to a given learning set (Wolpert, 1992). This learning is made by generating a second latent space whose inputs are the model predictions.

In simpler terms, this paper implements Stacked Ensembling by taking the outputs of the regression-performing models mentioned in this section and passing them through a deep neural network to create a final output. The purpose for performing stacked ensembling is to improve the accuracy/RMSE of the best models we presently use in stock predictions.

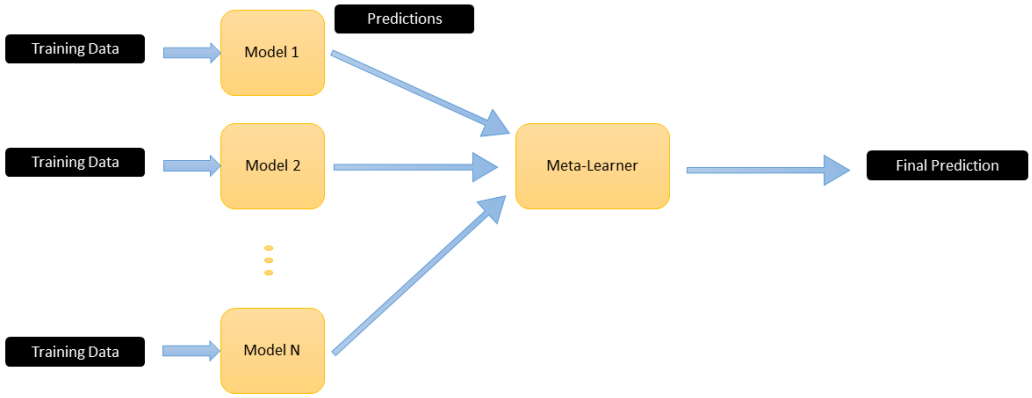

Figure 2.3: An example of Stacked Ensemble structure

CHAPTER 3

EXPERIMENT SETUP

§ 3.1 The Data

The historical stock data of IBM was collected from Yahoo Finance using the Python package *yfinance*. The dataset had the following indicators:

- Opening price: The opening price of the asset is the first price displayed at the beginning of a trading day.
- Close price: The Close Price of the asset is the last fixed price for that asset on the day of listing. It is considered the reference price for tax purposes and is one of the key pieces. of information that enables investors to value their investments on the stock exchange.
- Volume: This represents the number of securities/shares traded over a certain period of time.
- Adjusted close: The adjusted closing price reflects the closing price of the stock in relation to other stock attributes.
- High: High Price is the highest price at which the stock has traded during the given trading day.
- Low: The low price is the lowest price at which the stock has traded during the given trading day.

Figure 3.1 shows a sample of the daily closing price of the publicly traded IBM company from 2000 to 2021. There are a total of 5283 trading days. Section 4 of this thesis will be dedicated to a case study of IBM using stacked ensembling and predicting the daily returns. This will be discussed later in Section 3.4.1 where we will normalize the close price to better perform regression, as working with large values can lead to large coefficients or high bias values.

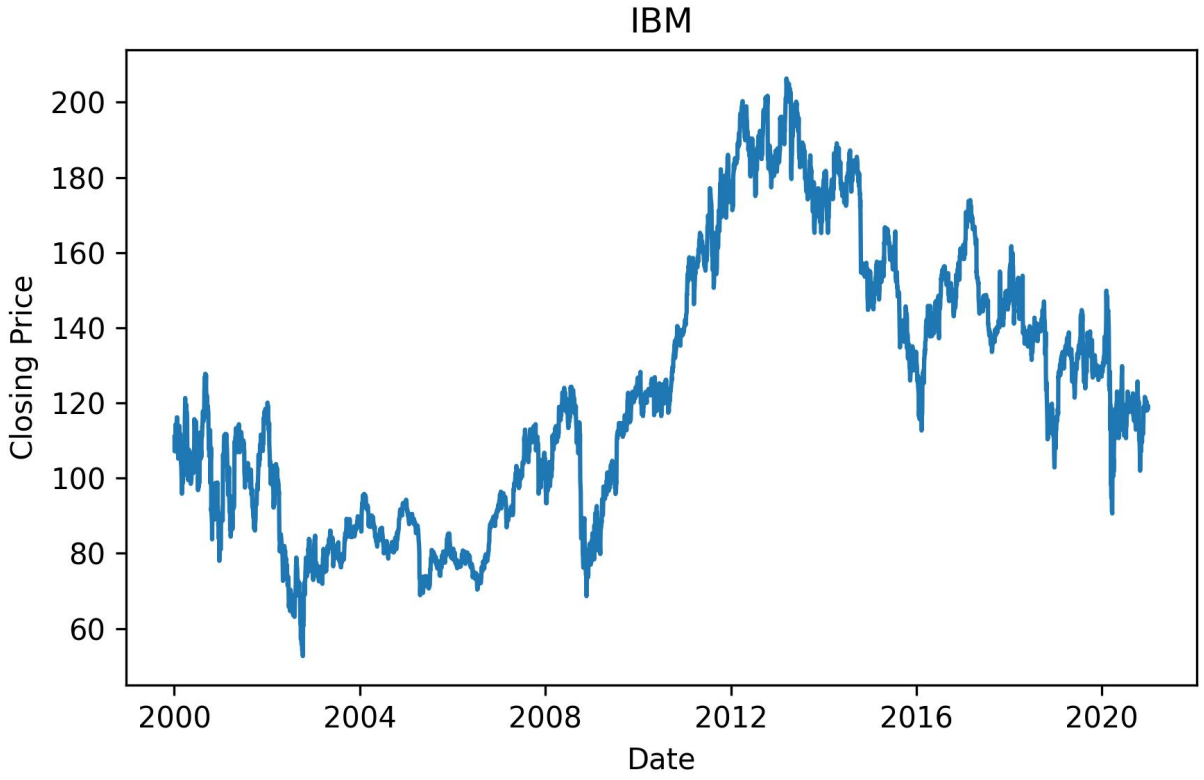

Figure 3.1: Daily close price of IBM (2000-2021)

We can see that stock prices are not guaranteed to increase or decrease in any form of consistent manner as discussed by the random walk hypothesis (Fama, 1965b). However, utilizing 2.2 mentioned in section 2.3 we know that stock market returns can be modelled as a function of previous values + randomness.

§ 3.2 The Metrics

For comparing different algorithms and predictors, we use 3 common metric functions:

1. Root Mean Squared Error (RMSE)

   RMSE is a very common way to evaluate the performance of a regression model primarily because it is easily interpretable. Since all the terms are squared, any significant error in any prediction will be highlighted by this metric. A lower RMSE is always preferred for a regressive model because it

indicates less error in predictions. 3.1 is the equation for RMSE, where $\hat{y}_t$ and y_t are the predicted value and the real value at time t, respectively. Here, T represents the number of points/time steps for which to evaluate RMSE.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{3.1}$$

While being easy to explain, it does have the drawback of being scale-dependent (Hyndman & Koehler, 2006) i.e. a reported RMSE value can be good or bad depending on the scale of data. For instance, an RMSE value of 5 with a variance of 1000 is excellent but it is terrible if variance is just 1. This is why RMSE alone is misleading because its output cannot by itself assess how good a model is. However, while comparing two different models with different RMSE, it can be easy to say that the model with a lower RMSE is better; however, correctly predicting the direction of market movement is equally important. To compensate for these drawbacks, we will use two additional metrics for evaluation.

2. Mean Absolute Percentage Error (MAPE)

   MAPE is a percentage based evaluation metric for regression models. It overcomes the problem of RMSE by being scale-invariant because it normalizes the error to a percentage. 3.2 is the equation for MAPE, where $\hat{y}_t$ and y_t is the predicted value and real value at time t, respectively. A lower MAPE is preferred for the same reason as a lower RMSE is preferred. In addition, T represents the number of points/time-steps for which to evaluate MAPE.

$$MAPE = \frac{100\%}{T} * \sum_{t=1}^{T} \frac{(|\hat{y}_t - y_t|)}{y_t} \tag{3.2}$$

   In contrast to RMSE, MAPE is scale-invariant, thus allowing us to get an accurate idea of the error percentage in our predictions. While MAPE is able to overcome the scale invariant problem, we still need another metric to evaluate the directional accuracy of our prediction.

3. Mean Directional Accuracy (MDA)

RMSE and MAPE together would suffice if we did not care about direction of change, however, when it comes to stock markets, that is an important problem. For instance, 5% MAPE is more acceptable if the error is in the direction of change of actual value. However the same is not true if direction of change is not the same for predicted and real prices. Predicting a rise in market while the market actually falls or vice versa is worse than predicting the correct direction but wrong price. Therefore, we need MDA to capture this problem.

Schnader and Stekler (1990) discussed about creating a 2x2 contingency table to evaluate whether the higher or lower erroneous forecasts were independent of each other. Later, this idea evolved to what we know today as MDA. 3.3 is used to calculate the directional movement of the predicted stocks and the true value of the stocks. $\hat{y}_t$ and y_t are the predicted value and the real value at time t, respectively. 3.4 is the unit indicator function equal to 1 if both inputs to the function are equal, otherwise it returns 0. Similar to previous equations, T represents the number of points/time steps for which to evaluate MDA.

$$\begin{aligned} \Delta\hat{Y}_t &= sgn(\hat{Y}_t - \hat{Y_{t-1}}) \\ \Delta Y_t &= sgn(Y_t - Y_{t-1}) \end{aligned} \tag{3.3}$$

$$Ind(x, y) = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases} \tag{3.4}$$

Using 3.3 and 3.4, we can define MDA as shown in 3.5. It is conceptually similar to calculating the accuracy in a classification problem.

$$MDA = \frac{100\%}{T} * \sum_{t=1}^{T} Ind(\Delta\hat{Y}_t, \Delta Y_t) \tag{3.5}$$

Having a higher MDA value is better because it indicates that the predictor is able to correctly estimate the direction of stock movement irrespective of the actual value.

§ 3.3 A Naive Approach

Let us use the moving average of the last 20 days to predict the stock price of the next day. Moving average is defined as the average of last 'N' days. 3.6 is the general equation for moving average over N days.

$$Y_t = \frac{y_{t-N} + y_{t-N+1} + \cdots + y_{t-2} + y_{t-1}}{N} \tag{3.6}$$

If we consider N=20 and consider the result as the value of tomorrow's stock price, we can see the results in figure 3.2. Although our RMSE and MAPE is not bad, we have a terrible Directional Accuracy. Metrics for evaluation are discussed in section 3.2.

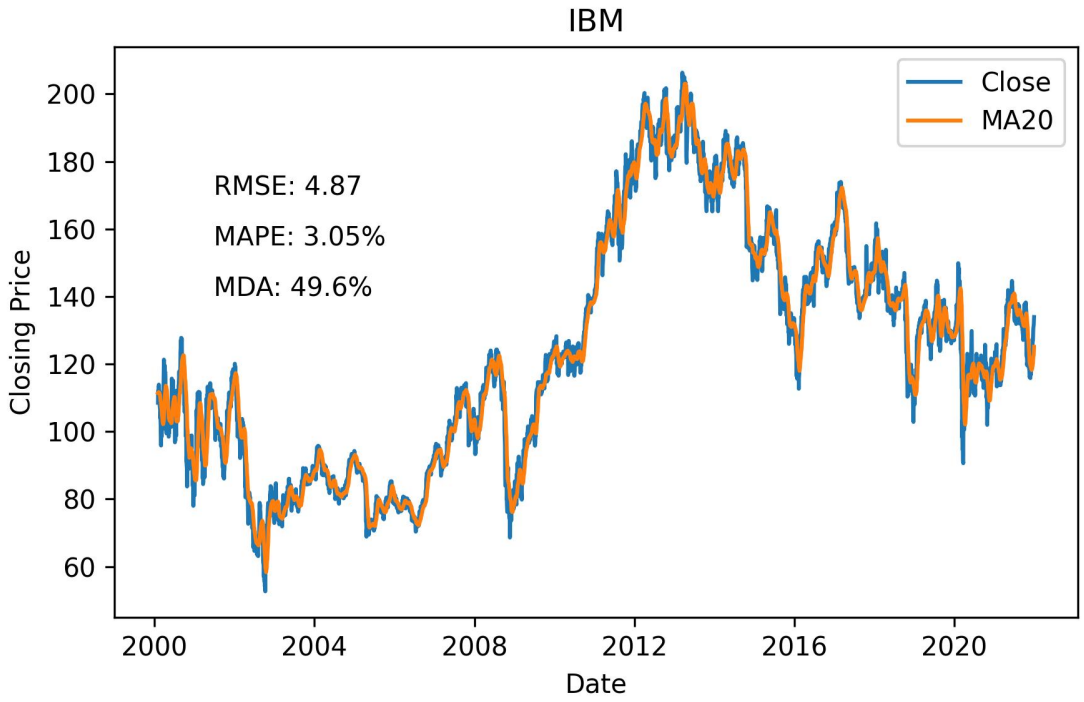

Figure 3.2: Price trend prediction on IBM using Moving Average

Clearly, this method is not useful in the real world. Therefore we need to rely on the statistical and neural network based models for predictions. The preprocessing steps and training processes for the different models is discussed ahead.

§ 3.4 Pre-Processing methods used

Before progressing into the methodology of this thesis, we need to introduce the two main preprocessing steps used to prepare and convert the stock price data for easily performing regression. Pre-processing is important for any ML model because it helps us scale or clean the data to a usable format.

§ 3.4.1 Scaling

Based on the past literature, we decided to focus only on the closing price and the log scale of the volume. We converted the close price data to log returns. As mentioned above, a simple return on the stock is shown in 3.8. As shown by 3.7, we extracted a logarithm on both sides to get our target variable Yt at any given time t. The focus was to predict the price change one day in the future without focusing on the price value to itself.

$$\frac{P_t - P_{t-1}}{P_{t-1}} = r \tag{3.7}$$

$$log_e(\frac{P_t}{P_{t-1}}) = log_e(1 + r_t) = log_e e^{R_t} = R_t = Y_t \tag{3.8}$$

- Y_0: Initial price of the stock.
- Y_t: Price of the stock at the end of the period.
- r: Simple return of the stock over time t.
- R: Continuously compounded rate over the period.

§ 3.4.2 Sliding Window

With the data normalized from price change to log returns, we need to structure our data in a manner such that we can use previous N time steps to predict the next time step. Transforming the data using this technique converts the problem from a time-series problem to a regression problem. From 2.2, we know that the current price of the stock can be modeled as a function of past values and error. Therefore, we need to modify our data set so that there would be N dimensions, where each dimension corresponding

to a value in the past N time steps. Table 3.1 presents a sample dataset converted using the sliding window technique with N=4. All rows with NA should be removed as it is impossible to perform predictions on the first $N - 1$ values with a sliding window of size N.

Table 3.1: Sliding Window Transformation for Regression (N=4)

Original Data	Sliding Window Transformation			
Close Price	X (Input)			Y (Target)
	Close Price @ t-3	Close Price @ t-2	Close Price @ t-1	Close Price @ t
Y_0	NA	NA	NA	Y_0
Y_1	NA	NA	Y_0	Y_1
Y_2	NA	Y_0	Y_1	Y_2
Y_3	Y_0	Y_1	Y_2	Y_3
Y_4	Y_1	Y_2	Y_3	Y_4
⋮	⋮	⋮	⋮	⋮

§ 3.5 Training process

Due to ensemble learning, we need a two stage process to create our final ensemble model. First, we need to train our Regression models. Then, we need to train our stacked ensemble learner to use the outputs of the regression models.

To prevent data leakage for training models, we split the dataset chronologically with respect to time into 3 parts.

1. Training and Validation set 1 - were used for hyperparameter tuning and window sizes of the regressive models. This had data from 1970 to 2012 for training and 2012-2015 for validation.

2. Training Set 2 and Validation Set 2 were used to adjust the hyperparameters of the stacked set. This had data from start to 2013 to end of 2017.

3. Final Testing Set - Closing prices from 2018 to the end of 2021.

After splitting the dataset, the data was converted to log returns for close price using the method mentioned in Section 3.4.1. Also, the logarithm of the volume was added as a separate feature. This makes our data scale invariant while also reducing the scale. Then, we use the sliding window method mentioned in section 3.4.2 to convert the data into a usable format for our regression models. After this, we can tune the hyperparameters to obtain the best models.

§ 3.6 Training and Hyperparameter tuning - An account of IBM

As previously mentioned, historical data of IBM stock from 1970-Januray 01 to 2021-December 31. To predict the stock price of IBM, we need regression models that can effectively utilize past data to extrapolate/predict the closing price of next day. For this thesis, we have implemented Linear Regression, Lasso, Multi Layer Perceptron, LSTM and Bidirectional-LSTM. We tuned every model for Window Size (N) and their own hyperparameters. Gathering the best model for each criteria was the priority; each model was evaluated for leading in either RMSE or accuracy for each type of algorithm. Exact hyperparameter search numbers are available in Appendix A.

§ 3.6.1 Linear Regression

In Python, the package *Scikit-Learn* has the ability to easily implement Linear Regression. The only hyperparameter we had to adjust was window size. Additionally, Linear Regression makes for an excellent baseline model to compare other models to; therefore, we will be using it extensively in Chapter 4 when we compare model performances. For this thesis, we chose two linear regression models, with the first one having a window size of 20 and the other with a window size of 4. The first LR model had the highest directional accuracy, while the second had the least RMSE.

§ 3.6.2 Lasso

The package *Scikit-Learn* can be used to implement Lasso. the model was tuned for Window Size (N), Regularization Strength(λ) and Tolerance(tol). Regularization Strength(λ) controls the penalty L1 term as shown in equation 2.6. Its value can range from $[0, \infty)$ and a higher number will entail higher penalty on the model for increasing weights. Tolerance (tol) is the smallest possible coefficient that is acceptable by

Lasso, below which it is reduced to zero. The tolerance coefficient is extremely important as it it eliminates noisy input dimensions and helps the model select the best features.

For the purpose of this thesis, we searched potential hyperparameters in a grid search pattern across 928 different possible combinations of hyperparameters and window size.In terms of accuracy and RMSE for Lasso, RMSE we got the same model with window size 10, Regularization Strength at 1 and tolerance at 0.0001.

§ 3.6.3 Multi-Layer Perceptron (MLP)

To create a multilayer perceptron regressor, we used *SkLearn*. *SkLearn* has multiple parameters, which allows us to set all the different hyperparameters that may be needed to tune the neural network. We trained multiple neural network configurations with varying numbers of layers, neurons in each layer, learning rate, activation functions, and window size. Tuning an MLP can become quite complex due to the large number of hyperparameters and options available. In total, we checked 9300 different types of models, and the below list contains the best two models:

- The best model in terms of accuracy had a linear/identity activation function with 4 hidden layers (128,256,256,64 neurons, respectively) and an inverse scaling learning rate, which decreases as epochs increase. All other hyperparameters were left to their default values. Window size for this model was 7. Figure 3.3 shows the loss curve for this model. Training was stopped if loss did not improve over 3 epochs.

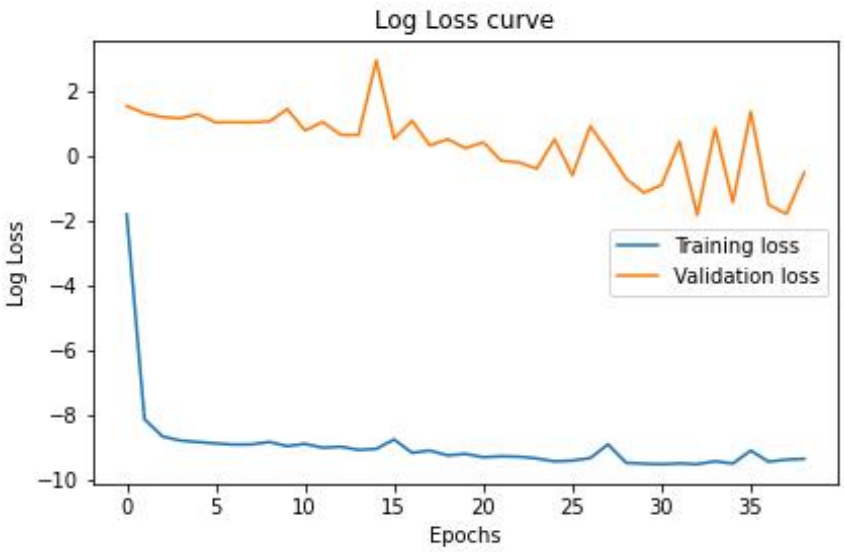

Figure 3.3: Log Loss Curve for best MLP model in terms of accuracy.

- The best model in terms of RMSE had a hyperbolic tangent activation function with only 1 hidden layer (16 neurons) and a constant learning rate of 0.001. All other hyperparameters were left to their default values. Window size for this model was 2. Figure 3.4 shows the loss curve for this model. Training was stopped if loss did not improve over 3 epochs.

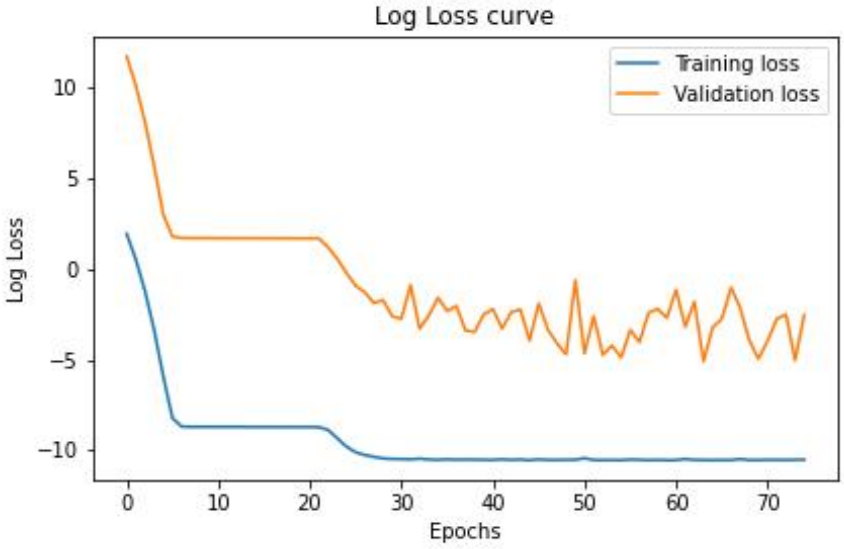

Figure 3.4: Log Loss Curve for best MLP model in terms of RMSE.

§ 3.6.4 LSTM

We used *Tensorflow 2.0* to develop and train the LSTM models. It has simple classes and functions to help create and train LSTM networks. LSTM networks are harder to train due to all the computation required at each gate, but are more powerful. Therefore, for the hyperparameter search, a simple LSTM network was searched using 1-2 hidden layers of LSTM followed by 1-2 layers of a dense neural network with dropout set to 0.1.

Tuning an LSTM network is more difficult due to the added complexities and the substantial number of higher parameters; therefore, we limited to searching through a set of 2400 possible different configurations with varying hidden layers, the number of neurons and the size of the window. The best models were as follows:

1. The best model in terms of accuracy was the LSTM model with a single hidden layer of 32 LSTM cells followed by 2 densly connected layers of sizes 4 and 16. The window size for this model was 14. Figure 3.5 shows the loss curve for this model. Training was stopped if loss did not improve over 3 epochs.

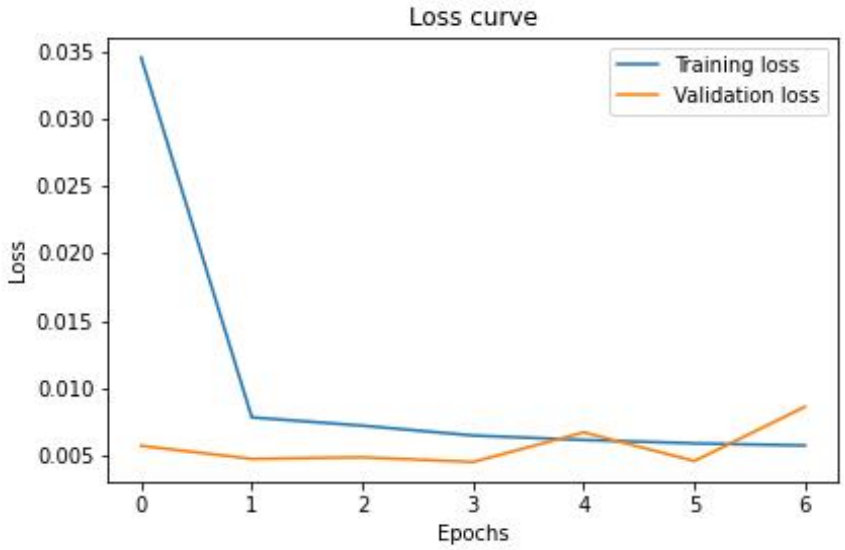

Figure 3.5: Log Loss Curve for best Bi-LSTM model in terms of Accuracy.

2. The best model in terms of RMSE was the LSTM model with 2 hidden layers that had 8 and 32 cells followed by 2 densely connected layers of sizes 16 and 1. The window size for this model was 16. Figure 3.6 shows the loss curve for this model. Training was stopped if loss did not improve over 3 epochs.

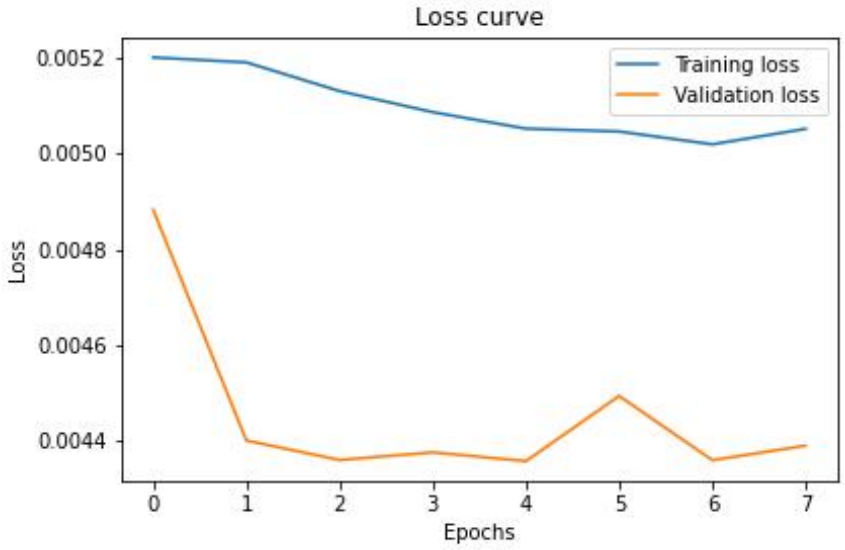

Figure 3.6: Log Loss Curve for best LSTM model in terms of RMSE.

The other hyperparameters were set to their default values, and each dense layer had a dropout set to 0.1 to avoid overfitting in the network.

§ 3.6.5 Bi-LSTM

Similar to the LSTM network, the Bi-LSTM network can also be developed and trained using *Tensorflow 2.0* as it has the *Birectional API* to convert any type of RNN to a bidirectional-RNN. A total of 2400 different configurations using 1-2 hidden layers of Bi-LSTM followed by 1-2 layers of a dense neural network with dropout set to 0.1 were validated. The best models were as follows:

1. The best model in terms of accuracy was the Bi-LSTM model with a single hidden layer of 32 Bi-LSTM cells followed by 2 densely connected layers of sizes 16 and 8. The window size for this model

was 18. Figure 3.7 shows the loss curve for this model. Training was stopped if loss did not improve over 3 epochs.

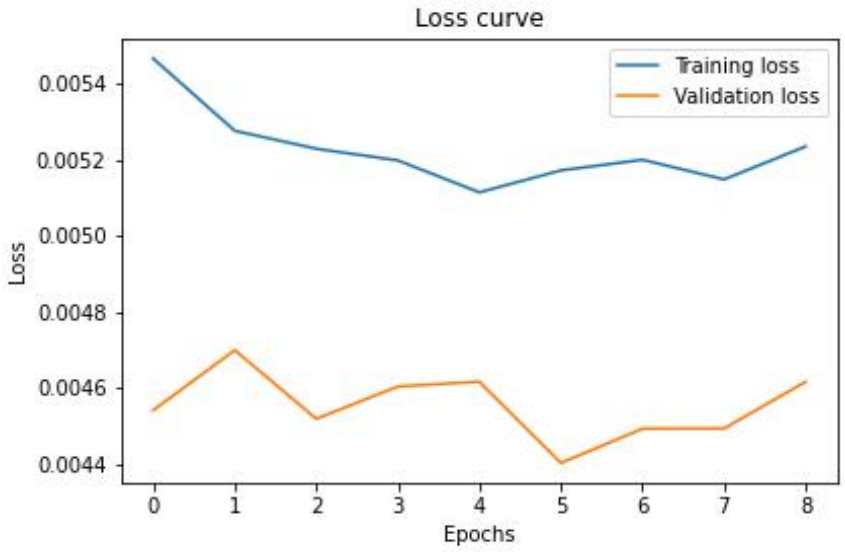

Figure 3.7: Log Loss Curve for best Bi-LSTM model in terms of Accuracy.

2. The best model in terms of RMSE was the Bi-LSTM model with a single hidden layer having 8 cells followed by 2 densely connected layer of sizes 16 and 1. The window size for this model was 12. Figure 3.8 shows the loss curve for this model. Training was stopped if loss did not improve over 3 epochs.

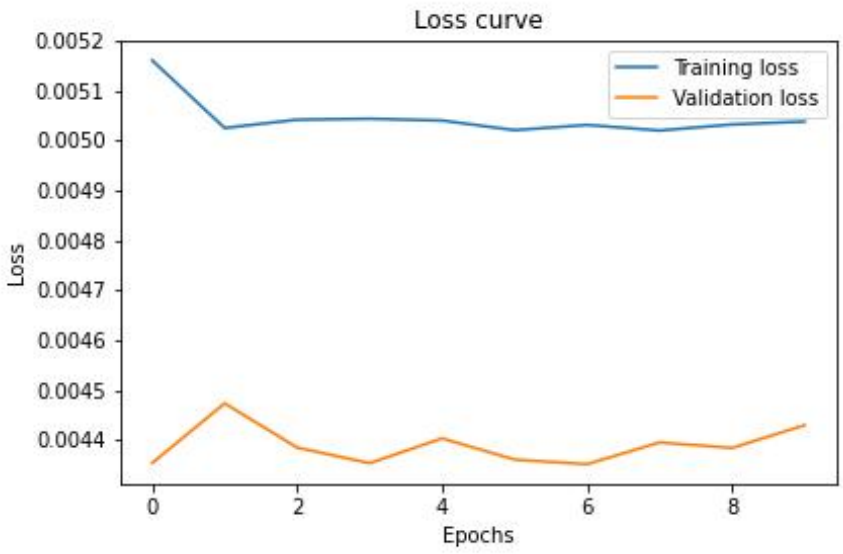

Figure 3.8: Log Loss Curve for best Bi-LSTM model in terms of RMSE.

The other hyperparameters were set to their default values, and each dense layer had a dropout set to 0.1 for the dense layers to avoid overfitting in the network.

§ 3.7 Training the ensemble

The meta learner is responsible for taking inputs from the regression models and aggregating them to obtain a better prediction. For this purpose, we chose to go with a stacked ensemble which is a Deep Neural Network. To train this network, we used all the training data until 2012 for the models and then predicted the IBM close price in a rolling fashion until the end of 2017. After that, we took the prediction values from all 9 models from 2012 to end of 2015 as training and from 2016 to the end of 2017 for validation. A sliding window is not required in this case as we just need to aggregate the results of the models.

Hyperparameters were tuned for this model by checking 3100 different models with differing numbers of layers, neurons, activation functions and learning rates. As previously mentioned, we utilized *SkLearn* to easily find the two best models in terms of accuracy and RMSE. All the other hyperparameters were left to their default values, and *Sklearn* internally took care of epochs, batch size, etc. It also allowed for the focus on creating the structure of the MLP meta-learner. They are as follows:

1. The best meta-learner in terms of accuracy had 4 hidden layers with 8,2,8,2 neurons, respectively, in the layers. It had an internally hyperbolic tangent activation function and a constant learning rate.

2. The best meta learner in terms of RMSE had 4 hidden layers too, but with 32,2,2,2 neurons respectively in the layers. It had a constant learning rate too.

Table 3.2: Metrics for both the ensemble models

Model Name	MAPE	Accuracy
Ensemble Model 1 (Best Accuracy)	0.43%	66.27%
Ensemble Model 2 (Best MAPE)	0.35%	63.49%

§ 3.8 Final testing

For the final training and creation of a final model, training data until start of 2018 to train the regression models and the ensemble. After that, the same models to predict the closing price of IBM stock from 2018 to end of 2021 and aggregated them using the meta-larner. We compared the performance of the two meta learners to simple methods like aggregating or bagging regressors. Chapter 4 covers the results obtained from all the individual models and the ensemble.

CHAPTER 4

RESULTS

This section of the thesis is split into 3 parts. The first part discusses the performance of each model on the stock price of IBM from 2018 to 2021. We have showcased the RMSE, MAPE and MDR for all the 9 different models we have used. Additional metrics are available in Appendix A. The second part compares our two stacked ensembles with averaging and bagging.

Figure 4.1 shows the daily logarithmic returns of IBM from 2018-2021. This shows us that there is no specific direction of movement. Table 4.1 summarizes the direction of stock increases shown in figure 4.1.

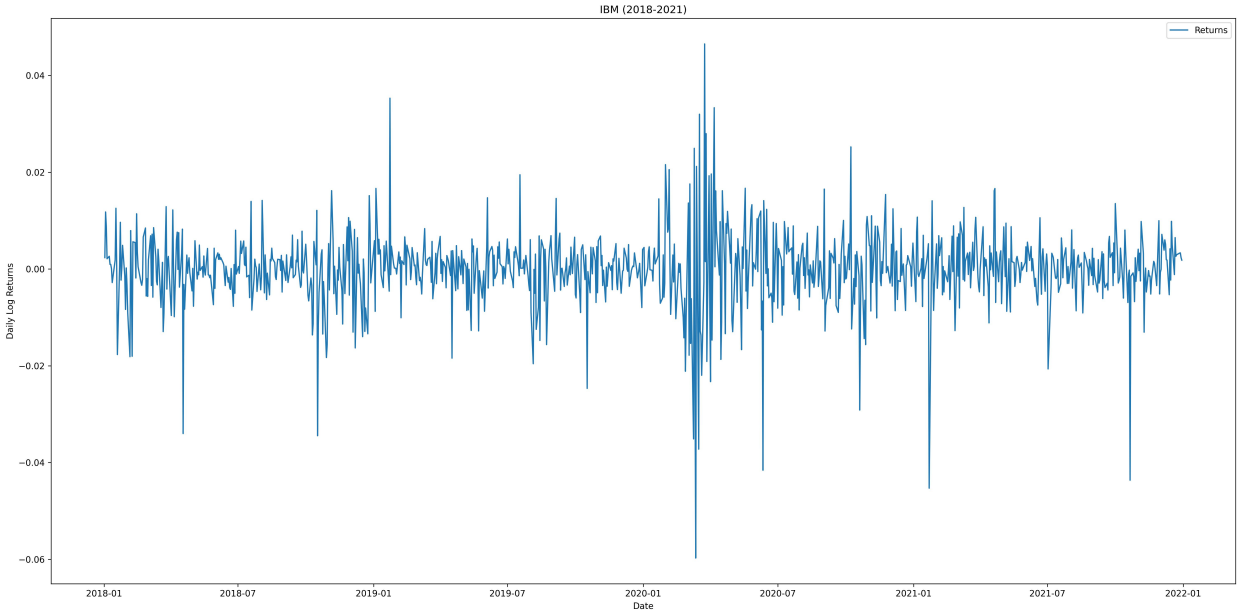

Figure 4.1: Daily Log Returns IBM (2018-2021)

Table 4.1: Frequency of Directional Movement for IBM (2018-2021)

Stock Direction	Count	Relative Count
Positive (Increasing)	528	0.525
Negative (Decreasing)	477	0.475

We can see that the daily direction of stock movement can be positive or negative with almost equal probability; therefore, this eliminates bias in training data. For a model to achieve greater that 52% accuracy, the model will have to leverage the information in past stock prices to obtain better directional accuracy.

§ 4.1 Performance of each Model

All the models were trained on the complete stock history of IBM until 2018, and then they were used to predict the prices from 2018-2021 in a rolling fashion. We can see that Linear Regression does beat the LSTM and Bi-LSTM models by 3,97%, however, apart from MLP which performed terribly ((MDA < 52%), all models had an MDA greater than 72%. Additionally, it was observed that the models which were focused more towards RMSE performed better than those focused towards Directional Accuracy. We have also discovered that Linear Regression had the best RMSE and MDA over the time period of 2018-2021 an MDA value of 76.66%, which was 3-5% better than other LSTM based networks. At the same time, having an MAPE of 0.511%, is around 0.1% better than the other LSTM based models.

Table 4.2: Performance of different models on IBM stock (2018-2021)

Model	RMSE	MAPE	MDA (Mean Directional Accuracy)
BiLSTM (optimized for Accuracy)	1.072	0.606%	72.59%
BiLSTM (optimized for RMSE)	1.087	0.604%	72.69%
Lasso	0.954	0.511%	76.25%
Linear Regression (optimized for Accuracy)	0.954	0.511%	75.86%
Linear Regression (optimized for RMSE)	**0.953**	**0.510**%	**76.66**%
LSTM (optimized for Accuracy)	1.124	0.633%	72.09%
LSTM (optimized for RMSE)	1.089	0.602%	72.49%
MLP (optimized for Accuracy)	2.923	1.945%	51.43%
MLP (optimized for RMSE)	2.396	1.512%	51.64%

§ 4.2 Comparison of different Ensembles

Next, we need to ensemble these models and we have used four different techniques to perform this task, namely:

1. Stacking: Training a Meta-Learner that can improve its predictions by taking outputs of regressors to create another layer of decision making.

2. Aggregating: Averaging or finding the median of the predictions and treating that as output.

3. Bagging: Learning multiple simple regressors to create a voting/averaging regressor in the end. For the scope of this thesis, we have compared it against bagged Random Forest Regressors and bagged Decision Tree Regressors.

4. Ada Boost: Contains multiple regressors that learn from the parent's mistakes. LR was used as the base, as that was our best model with the highest accuracy percentage.

The final metrics are shown in table 4.3 in which the best metrics of each column is highlighted. We can see that the best stacking ensemble outperforms other common / conventional ensemble methods in MDA by 1.10% with a minimal decrease in RMSE compared to the best models in Table 4.2.

Table 4.3: Metrics of Different Ensemble Methods on IBM stock(2018-2021) using the different Regressors

Ensemble Method	Number of Estimators	RMSE	MAPE	MDA
Stacked (optimized for RMSE)	1	**0.963**	**0.530%**	**76.86%**
Stacked (optimized for Accuracy)	1	1.052	0.583%	75.17%
Average Value of Regressors	1	1.067	0.601%	68.92%
Median Value of Regressors	1	1.014	0.550%	72.29%
Bagged Decision Tree Regressors	5	1.287	0.749%	70.70%
Bagged Decision Tree Regressors	10	1.305	0.752%	68.22%
Bagged Random Forest Regressors	10	1.311	0.764%	68.82%
Ada Boost(base estimater: Linear Regression)	10	1.085	0.621%	75.57%
Bagged Linear Regressors	10	0.971	0.527%	75.76%

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, we have shown 9 machine learning models and trained them on IBM stock for the prediction of daily return prices. We found out that Linear Regression had the best RMSE and MDA over the time period of 2018-2021, closely followed by Bi-LSTM and LSTM based networks. Linear Regression had an MDA value of 76.66% which was 3-5% better than other LSTM based networks. At the same time having an MAPE of 0.511% which is around 0.1% better than the other LSTM based models. Additionally, while Linear Regression may have been the best for IBM, it may not be the case for other publicly traded securities. The methodology followed in this thesis can be extended to other stocks or any time-series data.

When comparing Stacked Ensembling to other common ensembling methods, we found that a leverage of 1.1% can be used by picking a good meta-learner for stacked ensembling. This entails that while creating ensembles for stock market predictors we should always look at stacking ensembles as they may be better than the simpler ensembling methods for the stock you are working with. For the future, many possible avenues are available to improve model performance.

1. Using more Regressive Models: In this paper, we have used a limited number of models; however, we can easily expand on this to obtain more robust results.

2. Incorporating Fundamental Approach: A lot of people now focus on analyzing social media posts to obtain sentiment of the retail market which makes for a viable input for the price prediction models.

3. Hyper-Parameter Tuning: For the purpose of this thesis, we searched through a limited possible number of combinations of hyperparameters, but this can be expanded easily with more compute power and time.

4. Better Stacked Ensembles: We limited the scope of our stacked ensemble search to a multi-layer perceptron network however, there is scope for incorporating RNN, or Transformer based Neural Networks for the meta learner.

While the stock market remains as a black box whose movements appear random, we see that with the help of AI and statistical methods, that we can push the boundary of understanding patterns that seem to follow a random walk to a large extent. Over time, as our understanding of AI, ML and statistics improves, we will be able to develop better and more accurate models for stock market prediction.

BIBLIOGRAPHY

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140. https://doi.org/10.1007/BF00058655

Cybenko, G. V. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*, 303–314.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *International workshop on multiple classifier systems*, 1–15.

Ding, X., Zhang, Y., Liu, T., & Duan, J. (2015). Deep learning for event-driven stock prediction. *Twenty-fourth international joint conference on artificial intelligence*.

Drucker, H. (1997). Improving regressors using boosting techniques. *Proceedings of the 14th International Conference on Machine Learning*.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., & Vapnik, V. (1994). Boosting and other ensemble methods. *Neural Computation*, *6*(6), 1289–1301. https://doi.org/10.1162/neco.1994.6.6.1289

The encyclopedia of stock market techniques. (1963). Investors Intelligence. https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=cat06564a&AN=uga.9910037553902959&site=eds-live&custid=uga1

Fama, E. F. (1965a). The behavior of stock-market prices. *The Journal of Business*, *38*(1), 34–105. Retrieved April 12, 2022, from http://www.jstor.org/stable/2350752

Fama, E. F. (1965b). Random walks in stock market prices. *Financial Analysts Journal*, *21*(5), 55–59. Retrieved April 13, 2022, from http://www.jstor.org/stable/4469865

Felsen, J. (1976). A man-machine investment decision system. *International Journal of Man-Machine Studies*, *8*(2), 169–193. https://doi.org/https://doi.org/10.1016/S0020-7373(76)80042-9

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139. https://doi.org/https://doi.org/10.1006/jcss.1997.1504

Gorenc Novak, M., & Velušček, D. (2016). Prediction of stock price movement based on daily high prices. *Quantitative Finance*, *16*(5), 793–826.

Graham, B., Dodd, D. L. F., Cottle, S., et al. (1934). *Security analysis* (Vol. 452). McGraw-Hill New York.

Hill, T., O'Connor, M., & Remus, W. (1996). Neural network models for time series forecasts. *Management science*, *42*(7), 1082–1092.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/https://doi.org/10.1016/0893-6080(89)90020-8

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688. https://doi.org/https://doi.org/10.1016/j.ijforecast.2006.03.001

Jegadeesh, N. (1990). Evidence of predictable behavior of security returns. *The Journal of Finance*, *45*(3), 881–898. Retrieved April 15, 2022, from http://www.jstor.org/stable/2328797

Khan, W., Ghazanfar, M. A., Azam, M. A., Karami, A., Alyoubi, K. H., & Alfakeeh, A. S. (2020). Stock market prediction using machine learning classifiers and social media, news. *Journal of Ambient Intelligence and Humanized Computing*, 1–24.

Le, X. H., Ho, H., Lee, G., & Jung, S. (2019). Application of long short-term memory (lstm) neural network for flood forecasting. *Water*, *11*, 1387. https://doi.org/10.3390/w11071387

Lo, A. W., & MacKinlay, A. C. (2015). Stock Market Prices Do Not Follow Random Walks: Evidence from a Simple Specification Test. *The Review of Financial Studies*, *1*(1), 41–66. https://doi.org/10.1093/rfs/1.1.41

Lu, W., Li, J., Wang, J., & Qin, L. (2021). A cnn-bilstm-am method for stock price prediction. *Neural Computing and Applications*, *33*(10), 4741–4753.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Opitz, D., & Maclin, R. (1999). Popular ensemble methods: An empirical study. *J. Artif. Int. Res.*, *11*(1), 169–198.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Saad, E., Prokhorov, D., & Wunsch, D. (1998). Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on Neural Networks*, *9*(6), 1456–1470. https://doi.org/10.1109/72.728395

Schnader, M. H., & Stekler, H. O. (1990). Evaluating predictions of change. *The Journal of Business*, *63*(1), 99–107. Retrieved June 8, 2022, from http://www.jstor.org/stable/2353240

Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093

Shah, D., Campbell, W., & Zulkernine, F. H. (2018). A comparative study of lstm and dnn for stock market forecasting. *2018 IEEE International Conference on Big Data (Big Data)*, 4148–4155. https://doi.org/10.1109/BigData.2018.8622462

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, *19*(3), 425–442.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.

White, H. L. (1988). Economic prediction using neural networks: The case of ibm daily stock returns. *IEEE 1988 International Conference on Neural Networks*, 451–458 vol.2.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*(2), 241–259. https://doi.org/https://doi.org/10.1016/S0893-6080(05)80023-1

Zhang, J., Wang, P., Yan, R., & Gao, R. (2018). Long short-term memory for machine remaining life prediction. *Journal of Manufacturing Systems*, *48*. https://doi.org/10.1016/j.jmsy.2018.05.011

APPENDIX

APPENDIX A

We used the parameter grid available in sklearn for searching through possible options in a grid like fashion. Any parameter not mentioned below was left to its default value in Sklearn or Tensorflow.

1. Linear Regression: Only parameter to be tuned was Window Size. N=1 to N=30. Everything else was left to its default value.

2. Lasso: We worked with four parameters and the list is given below:

Table A.1: Lasso Parameters

Parameter	Possible Values
Alpha (Regularization Parameter)	1, 0.01, 0.001, 2
Max_iter (Maximum iterations)	100, 1000
tol (Tolerance)	0.0001, 0.001
selection (Feature selection)	cyclic, random

   Lasso had a total 32 possible configurations given the above options. we used all configurations along with 29 different window sizes (N-1 to 29) giving us a total of 928 different configurations.

3. Neural Network: Neural Networks have a lot of different parameters. Therefore, we limited the search space heavily.

Table A.2: MLP Parameters

Parameter	Possible Values
Number of Neurons in a layer	16,64,128,256
Number of Layers	1, 2, 4
Activation Function	identity, tanh
Learning Rate	constant, invscaling

This gives us 1104 possible combinations per window size. We tuned the model until window size 9 however upon seeing diminishing results, we capped the training at 9300 different models.

4. LSTM & Bi-LSTM: Other models were made with the help of Sklearn stack therefore making the model creation process easy however LSTM and Bi=LSTM models were made using tensorflow therefore code for the same has been added below. Similar to Neural Networks, LSTMs have a variety of parameters. The structure of the model consisted of layers of LSTM cells followed by a small MLP model to get the final output.

Table A.3: LSTM Parameters

Parameter	Possible Values
Number of Neurons in LSTM/Bi-LSTM layer	1,4,8,16
Number of Neurons in hidden MLP layer	8,16,32
Number of Layers of LSTM/Bi-LSTM	1, 2
Number of Layers of hidden MLP	1, 2
Activation Function	identity, tanh
Learning Rate	constant, invscaling
Window Size	2,4,6 ... 20

The loss was set to "mean_absolute_error" and all neurons had an activation of tanh except a seperate output layer with a linear activation. In separate testing, these activation functions performed better

than others. There are a total of 2400 possible configurations. Below is the code for generating a single LSTM model.

Tensorflow Code for creating LSTM model

```
def __createLSTM_model__(neurons,LSTMneurons,input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))
    # input_shape depends on window size
    model.add(Reshape(target_shape=(-1,int(input_shape[0]))))
    for n in LSTMneurons:
        model.add(LSTM(n, return_sequences=True))
    model.add(Flatten())
    for n in neurons[1:]:
        model.add(Dense(n,activation='tanh'))
        model.add(Dropout(0.1))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_absolute_error',
                  optimizer='adam',
                  metrics=['MeanSquaredError'])
    return model
```

To change the code to Bi-LSTM replace the LSTM layer addition line to a Bidirectional LSTM.

Tensorflow Code for Bi-LSTM model

```
def __createBiLSTM_model__(neurons,BiLSTMneurons,input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))
    model.add(Reshape(target_shape=(-1,int(input_shape[0]))))
    for n in BiLSTMneurons:
```

```
        model.add(Bidirectional(LSTM(n, return_sequences=True)))
    model.add(Flatten())
    for n in neurons[1:]:
        model.add(Dense(n,activation='tanh'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_absolute_error',
                  optimizer='adam',
                  metrics=['MeanSquaredError'])
    return model
```

APPENDIX B

The below table contains additional metrics for the models used along with the stacked ensembles. To save space, Explained Variance was abbreviated to EV.

Table B.1: Additional performance details of the final models.

Model	**RMSE**	**MAPE**	**MDA**	**Recall**	**Precision**	**EV**	**R2**
BiLSTM (Accuracy)	1.072	0.606%	72.59%	73.11%	74.52%	0.991518	0.991518
BiLSTM (RMSE)	1.087	0.604%	72.69%	73.48%	74.19%	0.991617	0.991614
Lasso	0.954	0.511%	76.25%	78.22%	76.91%	0.993418	0.993418
LR (Accuracy)	0.954	0.511%	75.86%	77.46%	76.74%	0.993421	0.993420
LR (RMSE)	0.953	0.510%	76.66%	78.41%	77.38%	0.993430	0.993429
LSTM (Accuracy)	1.124	0.633%	72.09%	73.30%	73.57%	0.991500	0.991500
LSTM (RMSE)	1.089	0.602%	72.49%	73.67%	74.10%	0.991430	0.991430
MLP (Accuracy)	2.923	1.945%	51.43%	53.41%	55.73%	0.980532	0.980531
MLP (RMSE)	2.396	1.512%	51.64%	50.19%	52.79%	0.971837	0.971787
Stacked (RMSE)	0.963	0.53%	76.86%	77.84%	77.99%	0.993298	0.993295
Stacked (Accuracy)	1.0516	0.58%	75.17%	77.08%	75.93%	0.992012	0.992004