
UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

UNIVERSITY OF CAPE TOWN

MASTER'S DISSERTATION

---

# A Hybrid Multi-Modal Recommender System using Neural Collaborative Filtering and Content Based Filtering

---

Author:
Pavan SINGH

Supervisor:
Assoc Professor Ian DURBACH
Dr. Allan E CLARK

A dissertation presented for the degree of
Master of Science Advanced Analytics

from the

Department of Statistical Sciences


April 2, 2023

# *Acknowledgements*

I would like to thank many people who made this journey happy, inspiring and rewarding. First and foremost, I would like to thank my advisors Ian Durbach and Allan Clark. I've learned a lot from their deep insights in statistics and their passion about developing scientific and practically useful methodologies. They were both very generous with their time and are always there to help. I am deeply grateful for all the inspiring discussions we had and the constructive feedbacks I received. This project wouldn't be possible without their valuable guidance and input.

I'm also very grateful for the great faculty and staff at Department of Statistics. Finally, I'm thankful to my sibling and parents for their unconditional love and support. Because of them, no matter where I am and what I am doing, I always feel encouraged and loved.

I would additionally like to acknowledge the assistance and support from my friends and fellow statistics masters students, who have provided persistent support and advice during this year.

# *Declaration of Authorship*

I, Pavan SINGH, declare that this thesis titled, "A Hybrid Multi-Modal Recommender System using Neural Collaborative Filtering and Content Based Filtering" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- The contents of this thesis has not been previously submitted for a degree or any other qualification at this University or any other institution.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

UNIVERSITY OF CAPE TOWN

# *Abstract*

Department of Statistical Sciences

Master of Science Advanced Analytics

**A Hybrid Multi-Modal Recommender System using Neural Collaborative Filtering and Content Based Filtering**

by Pavan SINGH

Online shopping has become a ubiquitous aspect of modern life, and recommender systems have become a crucial tool for e-commerce platforms such as Amazon. Recommender systems aim to provide users with personalized product recommendations based on their preferences and behaviors. They analyze user data, such as their browsing history, purchase history, and ratings, to understand their preferences and make recommendations that align with those preferences. They have become fundamental applications in electronic commerce and information access, providing suggestions that effectively prune large information spaces so that users are directed toward those items that best meet their needs and preferences. Due to recommender systems being a crucial tool for e-commerce platforms, studies in this domain has been growing rapidly and has become a very active area of research.

This paper aims to develop a hybrid recommender system model which incorporates data from multi-modalities, textual data and explicit ratings data. The hybrid model is composed of a neural collaborative filtering component and content based filtering component. The goal is to examine the impact of incorporating textual features and sentiment in potentially enhancing recommendation accuracy. We further explore the development of our proposed hybrid model by comparing its performance against the individual filtering models and several other popular filtering algorithms for recommendation systems. Our model shall be trained and deployed on the Amazon Reviews dataset, which contains millions of user reviews and feedback on thousands of different products. The data set also provides a large corpus of metadata making it adequate for exploring both dimensions of filtering approaches. Our methodology is based on a literature analysis and aims to clearly extrapolate on our singular models to develop a hybridized recommender system.

Empirical results indicate that the hybrid NCF and CBF model performed with a root mean square error (RMSE) of xyz with random data splits and sentiment features included. The model evidently outperformed all the benchmark models tested and shows that there exists scope for a hybridised multi-modal model incorporating NCF and CBF to benefit e-commerce industry to better modelling consumer preferences. Moreover, there is great support for including textual features and sentiment to further enhance models. Additionally, we found that the deep learning recommendation models consistently outperformed their classical counterparts. The algorithm, models and techniques developed and used in this paper are not problem-specifics and can be applied to different recognition and prediction problems.

# Contents

# Chapter 1

# Introduction

## 1.1 Introduction To Problem

Thanks to a plethora of high-profile applications in fields like autonomous transport, intelligent robotics, and image recognition, artificial intelligence (AI) has gained popularity in recent years (Norvig, 2002). Machine learning is a subfield of AI and refers to the general techniques used to extrapolate patterns from large data sets, or as the ability to make predictions on new unseen data based on what is learned (Ahmed *et al.*, 2010). The success of AI is largely due to the use of machine learning methods that can learn and improve over time. This is in contrast to traditional programming, where step-by-step coding instructions are followed (Norvig, 2002). Machine learning methods encompass a variety of models, including support vector machines, Random Forest, and Neural Networks, among others (Ahmed *et al.*, 2010).

Given the substantial growth and popularity, machine learning has found applications in the field of time series forecasting. Time series data - data points indexed in time - are found in many problems. One such problem which is widely researched is financial data (Zhang, 2003). Our interest in this domain for this project is stock market data. Forecasting the returns of stocks is a challenging task due to the volatility and non-linearity of the data (Oancea *et al.*, 2014). Classical statistical and econometric methods like auto-regressive integrated moving average (ARIMA) and moving average (MA) have been used for financial time series prediction, but have been found to fail in efficiently handling the uncertain nature of stock market data (Oancea *et al.*, 2014). This is due to the fact that these statistical methods assume the time series is generated from a linear process (Liwicki and Everingham, 2009). Therefore, they achieve poor performance when trying to predict the non-linear movements of stock prices. In recent times, neural networks have become increasingly popular in solving a variety of supervised learning [1] problems, since they are capable of approximating any non-linear function without any a priori information about the properties of the data (Norvig, 2002).

Recently, neural networks have become popular in solving a variety of scientific and financial problems, because they can approximate any non-linear function without any a priori information about the properties of the data (Norvig, 2002).

### 1.1.1 Aim of the Paper

This paper aims to compare the performance of various machine learning models, including two ensemble decision tree methods and two neural networks, in forecasting the movement of log returns of three stocks. More specifically, we compare a Random Forest model, Adaptive Boosting model (AdaBoost), Feed-Forward neural network and a Long Short-Term Memory (LSTM) neural network. We assess these models performance by their ability to predict whether the daily log return of a stock moves up or down, using three measures - accuracy, F1 score and AUC (area under the curve) score. Three stocks are used, namely Google, Motorola and Ford Motors. In addition, we benchmark the performance of our machine learning models to that of a classical statistical approach in forecasting time series data; an auto regressive integrated moving average model (ARIMA).

[1] Supervised learning is the task of learning a function that maps an input to an output using labelled training data

### 1.1.2 Contributions of Paper

With the continuous development and growing popularity in neural networks and machine learning, there exists a great deal of literature covering machine learning models in time series forecasting. Several types of neural networks have been used in financial forecasting problems and have been extensively discussed in the literature. However, there have only been a few accounts of comparing neural networks methods and ensemble decision trees methods on financial data. In addition, our paper benchmarks and validates our models by comparing our results with that of an ARIMA model.

## 1.2 Theoretical Concepts

### 1.2.1 Time Series

From stock market prices to measuring the spread of an epidemic, it is common place for data to be recorded with a time component (Naeini *et al.*, 2010). When the measurements are gathered together, they form a time series. Essentially, a time series is a sequence of observations in chronological order, like the daily log returns on a stock (Brillinger, 2001). The data do not have to be necessarily equally spaced, however this is a common assumption made (Liwicki and Everingham, 2009). For example, daily log returns on a stock may only be available for weekdays, with additional gaps on holidays. For simplicity, in this paper we regard the consecutive observations as equally spaced.

Time-series analysis serves as a fundamental statistical method for analysing the behaviour of time-dependent data and for forecasting. In the financial domain, time series modelling is considered the traditional technique to model financial data and is still widely used in finance today (Brillinger, 2001). The time series model chosen to benchmark the performance of our non-linear machine learning methods, is the ARIMA model. The ARIMA is a univariate time series model, which is widely used for forecasting time series using the series' past values (Brillinger, 2001).

The choice of using stock market data is motivated by the inherit non-linearity [2] of the time series data. Because of the market's unpredictability, which stems from constantly changing economic and political situations, as well as the stock market's instability, it is difficult to accurately anticipate stock market movement only by looking at price history (Shively, 2003). The complex dynamics of the data serves as an interesting foundation in which we can assess and compare the accuracy of each machine learning model.

### 1.2.2 Forecasting

The process of forecasting can be described as predicting the future using past and current data (Brillinger, 2001). Forecasting spans many areas including - but not limited to - business, health science, environmental science and finance (Lago *et al.*, 2018). It is an indispensable tool that helps, for example, financial institutions manage the uncertainty in the future for investments. Forecasting is used by businesses to make operational choices in areas such as buying, marketing, and advertising. Hence, a great deal of research has been conducted in identifying and improving forecasting models and techniques.

Forecasting models can be linear [3] as well as non-linear. The machine learning methods used in this paper are examples of non-linear forecasting models.

[2] Non-linear time series are generated by nonlinear dynamic equations. They display features that cannot be modelled by linear processes: time-changing variance, asymmetric cycles, higher-moment structures, thresholds and breaks.

[3] A linear time series is one where, for each data point $X_t$, that data point can be viewed as a linear combination of past or future values or differences.

Although linear models have shown to be effective tools for forecasting, they are intrinsically constrained when dealing with non-linear data. As a consequence, forecasts as well as other conclusions drawn from them could be misleading when applying these linear models on non-linear data. This had led to great interest in non-linear time series models in recent decades. Models such as the threshold autoregressive models (Tong, 1990) and the exponential autoregressive model (Haggan and Ozaki, 1981), are examples of non-linear models that have been developed to handle the non-linearities in the data of concern. However, one key issue that arises when using these non-linear models instead of linear models is that there is no unified theory that can be applied to all non-linear models. This is due to the fact that they need the imposition of assumptions about the specific form of non-linearity. Moreover, the pre-specified non-linear model may not be broad enough to capture all essential characteristics. One alternative way to deal with non-linearities in data is to use non-linear machine modelling approaches (Kuo and Huang, 2020). In contrast to the above model-based methods, non-linear methods like machine learning models are data driven and are thus capable of capturing complex patterns inherent in the data without needing any a priori information about the data (Liwicki and Everingham, 2009).

### 1.2.3 Neural Networks

Neural networks are a class of machine learning algorithms. They are made up of a collection of neurons that connect through various layers. Neural networks attempt to learn the mapping of the input data to the output data, on being provided with a training set. These models are non-linear and operate without prior beliefs about the functional forms of the data.

The objective of machine learning methods is the same as that of traditional linear statistical methods. Both machine learning and traditional linear techniques in statistics aim to improve forecasting ability, by minimising some loss function (Kuo and Huang, 2020). Their difference lies in how such minimisation is done. Machine learning methods utilise non-linear algorithms to do so while statistical methods employ linear processes (Kuo and Huang, 2020).

The application of neural networks in prediction problems is both pertinent and promising due to their special characteristic - being general function approximators (Liwicki and Everingham, 2009). Neural networks learn from examples and can capture the subtle functional relationship existing between the input and output data (Kuo and Huang, 2020). A collection of trainable parameters called weights are dispersed over multiple layers to achieve the mapping. The weights are learned by the backpropagation algorithm whose aim is to minimise a loss function. Given their generalisation ability, after training a neural network they are able to recognise new patterns from unseen data, even if the unseen data contain noisy information, at least in principle (Liwicki and Everingham, 2009). This follows from the universal approximation theorem (Agrawal *et al.*, 2013). The theorem states that any neural network can approximate any complex continuous function. We can learn about any relationship between the input and the output of the system, thus enabling us to learn any complicated relationship between the input and the output of the system (Kuo and Huang, 2020). We look at implementing two types of neural networks in this paper - a feed-forward and long short-term memory neural network.

### 1.2.4 Ensemble Decision Trees

Decision trees are a form of supervised machine learning algorithms which employ an if-then-else decision methodology to fit data and forecast results (Ampomah *et al.*, 2020). Ensemble decision tree methods are those which combines several decision trees to produce better predictive performance than by using a single decision tree (Ampomah *et al.*, 2020). The main principle behind the ensemble model

is that a group of weak learners come together to form a strong learner [4]. Random Forests and Adaptive Boosting models are built on this same principle (Nti *et al.*, 2020). A Random Forest algorithm entails building a multitude of decision trees and then merging them together to obtain a more accurate prediction (Ampomah *et al.*, 2020). Adaptive boosting, or AdaBoost, is a type of boosting technique, which involves using very short decision trees as weak learners that are added sequentially to the ensemble (Ampomah *et al.*, 2020). The subsequent models attempt to correct the predictions made by the model before it in the sequence. The Random Forest and AdaBoost algorithms represent some of the most popular ensemble methods (Nti *et al.*, 2020).

### 1.2.5 Structure of Paper

This paper consist of 7 chapters and the remaining chapters are organised as follows: chapter 2 presents a review of the literature published on predicting stock market data, neural networks and exploration into the performance of machine learning methods in comparison to statistical ones, with respect to applications in financial data prediction. The next chapter, then briefly explores the data sets for the stocks and try to understand the data at hand through exploratory data analysis. Following this, chapter 4 describes the methodology of the paper. Here we have a look at the data and theory of the various neural network models used for our analysis, the experiments performed, and the training algorithms used. In chapter 5, the model specification and optimisation processes for our experiment are provided. Chapter 6 presents the results of our analysis. Discussion and concludion then follow in chapter 7.

[4] A learner refers to a model. A weak learner is a model that has a low skill, meaning that its performance is slightly above a random classifier for binary classification or predicting the mean value for regression. In contrast a strong learner is one that performs well on a predictive modelling problem

# Chapter 2

# Literature Review

Due to the face that temporal data is created in a variety of circumstances, time series forecasting is a very common data modelling challenge. The financial domain has extensive research into modelling and analysis of time series data. Some of the early approaches to financial forecasting were based upon autoregressive models such as ARIMA (Binner *et al.*, 2005). Such models produce a forecast estimate which is dependent on a linear combination of past values and errors. If the assumption of stationarity is true and the series is generated by a linear process, autoregressive models have been found to perform well (Binner *et al.*, 2005). The data we are concerned with are stock returns which are generated from a non-linear process (Shivley, 2003). Machine learning models overcomes the shortcoming of linear modelling processes as they makes no assumptions about the prior distribution of the data (Binner *et al.*, 2005).

In this section we cover a variety of research on financial time series including a comparison on the literature between linear time series and non-linear machine learning methods in forecasting. We begin this review of the relevant literature, by looking at the random walk and efficient market hypothesis.

## 2.1 Random Walk and Efficient Market Hypothesis

It was widely believed, for many years, that stock prices could not be predicted. The random walk theory and the efficient market hypothesis (EMH) gave rise to this notion (Malkiel, 1973). The Random-walk theory (see Appendix A) states that the price of a stock swings irrespective of its historical performance. That is, stock prices will move in a random walk pattern, and that any prediction of stock movement will be around 50% accurate. According to EMH, an asset's current price always reflects all previously available information in real time. Hence, it cannot be predicted to consistently earn economic gains greater than the overall market average. As such, the use of prediction algorithms to determine future trends in stock market prices contradicts these two notions.

After exposure to the literature, there appears to exist contention surrounding financial markets and in particular the random walk hypothesis and EMH. Many researchers accept the random walk hypothesis, including Biondo *et al* (2013) who conducted a paper which supported the theory. In studying the performance of technical trading methods, they found that a random strategy outperformed some of the more traditional technical trading methods, such as Moving Average Convergence Divergence (MACD) and Relative Strength Index (RSI), in a series of tests. Other academics, contend that stock values can be forecasted, at least to some extent (MacKinlay, 1999). That is, some researchers have claimed that movements in market prices are not random. Rather, they behave in a highly non-linear, dynamic manner (MacKinlay, 1999). Papers published by Smith (2003), Nofsinger (2005), and Bollen *et al.* (2011), provide. evidence contrary to the notions suggested by EMG and the random. walk. hypothesis. These studies all conclude that that the stock market can be predicted to some degree and therefore. This is in contrast to and undermines the EMH's underlying assumptions. In this paper, we are not concerned with proving or disproving the predictability of markets. We are simply comparing models using stock data, however the research mentioned provided some context to the data domain.

## 2.2 Stock Market Prediction Strategy

The field of financial forecasting is characterised by noise, non-stationarity, lack of structure and high degree of uncertainty. There are many factors inherent in finance that have an effect on its dynamics and movements, such as political events and traders' expectations. Aside from the discord surrounding predicting stock markets, there has been a great deal of research on the predictability of stock returns. Current strategies regarding financial investing are broadly classified into two methods; fundamental and technical analysis.

Fundamental analysis refers to the type of trading analysis which involves the in-depth analysis of a company's performance and the profitability. This is done so as to measure a company's intrinsic value. Fundamental analysis studies the company in terms of its product sales, man power quality, infrastructure, profitability on investment (Agrawal *et al*, 2013). As such, the stock value of a company is estimated by analysing its margins, earnings, return on equity and profits amongst others as well as other economic factors. It based its principles that the market value of a stock tends to move towards its "intrinsic value" (Agrawal *et al*, 2013).

In contrast, technical analysis is an method for forecasting the direction of prices. This is done, not by trying to measure a ciompany's intrinsic value, but by through the study of past market data, primarily price and volume, mostly to trigger the buy or sell rules in the technical analysis (Kirkpatrick II and Dahlquist, 2010). It looks for peaks, bottoms, trends, patterns, and other factors affecting a stock's price movement.

Technical analysis is based on the following assumptions:

- Prices are solely determined by the supply demand relationship;
- Prices fluctuate in response to trends;
- Changes in supply and demand lead trends to reverse;
- Changes in supply and demand can be graphed;
- Patterns on charts seem to repeat themselves (Kirkpatrick II and Dahlquist, 2010)

Given the above, it is clear that technical analysis methodology ignores external aspects such as political, social, and macroeconomic factors (Cambell, 2012). A paper published in 2004, by Wong *et al* (2010), looked at the role of technical analysis in signalling the timing of stock market entry and exit. Using Singapore data, their results indicated that the indicators can be used to generate positive return from the Singapore Stock Exchange (SES). Moreover, Gao (2018) looked at improving improving stock closing price prediction using recurrent neural network and technical indicators. Gao (2018) employed an LSTM model coupled with technical indicators, to predict the closing price of Apple stock. They compared their model to various other forecasting models, including an ARMA, feed-froward neural network and support vector machine. Their proposed LSTM model with technical indicators outperformed all the other models.

## 2.3 Neural Networks in Financial Forecasting

Neural networks possess certain characteristics, which deliver very promising potential in prediction problems. Neural networks are particularly suited to settings in which identifying links between data is challenging but there is a large enough training data available. Due to their innate ability to generalise and approximate any complex continuous function, they are able to learn any relationship between a system's input and output (Binner *et al.*, 2005). This makes them extremely adept in prediction problems, and hence there exists vast research available on using this machine learning architecture on the financial data (Binner *et al.*, 2005).

The earliest application of neural networks in forecasting was in 1964 by Hu (Chen *et al.*, 2015). Although it did not receive significant attention due to the absence of any learning method for neural networks - thus, making it impossible to apply these networks to complex precision problems. However, in response to the work on error backpropagation learning, which was published by Werbose (1982) and by Rumelhart *et al* (1986), the interest in neural networks have grown rapidly since.

In 1988, White (1988) made the first attempt to model a financial time series using a neural network model. Looking at asset data for IBM, White (1988) tried to model a neural network for decoding the non-linear regularities in IBM's asset price movement. His work, albeit. limited in scope, helped in establishing evidences against EMH (Chen *et al.*, 2015). Moreover, in 1990, Kimoto et al. (1990), used neural networks to predict the index of the Tokyo stock market, and achieved precision of 63% precision (Naeini, 2010).

Recurrent neural networks (RNNs) are a class of artificial neural network. In these networks, connections between neurons form a directed cycle. RNNs can be compared to normal feed-forward networks, but with loops. As such, they naturally exhibit dynamic temporal behaviour for a time sequence. This has led to them being studied in the context of time series forecasting extensively. Additionally, RNNs have produced better results with respect to time series problems than other machine learning methods (Chen *et al.*, 2015). More specifically, in this paper, we look at using a long short term memory (LSTM) network, which is a type of RNN model LSTMs were introduced in 1997 by Hochreiter and Schmidhuber, to address certain limitations that standard RNNs had (Chen *et al.*, 2015).

When compared to other machine learning modelling methods, these networks have often produced some of the best results in the context of sequential data (Graves, 2012). Their impressive performance is particularly renowned in the field of Natural Language Processing (NLP), and in handwriting recognition (Liwicki, 2009). The promising results of LSTMs models in their applications have triggered their use in the financial forecasting.

From the past literature, LSTM have delivered promising results in forecasting financial assets. A paper published by Chen *et a.l* (2015), used price data to forecast price movement. More, specifically he used historic price data and technical indicators in addiction to stock indexes to predict if a stock's price would increase or decrease on a given day. Their research looked at different stocks from the Brazilian Stock Exchange. Chen *et a.l* (2015) were able to achieve promising results from their analysis. They obtained an average of 55.9% of accuracy when predicting if the price of a particular stock is going to go up or not - using their LSTM model.

## 2.4 Ensemble Decision Trees in Financial Forecasting

Due to the success of machine learning algorithms in a variety of fields since their inception, ensemble methods have also risen in popularity. Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. Fast algorithms such as decision trees are commonly employed in ensemble methods, however ensemble methods do not only pertain to decision trees. For the purposes of this study we shall be implementing ensemble decision trees, more specifically a Random Forest and an Adaptive Boosting model.

Recently there have been several publications regarding the evaluation of tree based ensemble machine learning models in financial forecasting. Ampomah *et al.* (2020) published a paper comparing the effectiveness of several tree based ensemble models. Their paper included a random forest, XGboost model, and even an Adaptive Boosting model. The authors assessed the models on their ability in forecasting the direction of stock price movement, using several performance metrics such as, accuracy, precision and area under receiver operating characteristics curve (AUC ROC). Ampomah *et al.* (2020) found that when using their training set, the AdaBoost model performed the best, while on the test. set,

the Extra Trees classifier outperformed the other models. In the study the Adaptive Boosting model was the second best model, followed closely by the Random Forest model.

Moreover, a study by Nti *et al.* (20), compared ensemble learning methods for classification and regression machine learning tasks in stock market prediction. The goal of the paper was to clarify which ensemble strategies are most suited for stock market prediction problems. In the study, a host of methods were used; namely boosting, bagging, blending and stacking. The models used included decision trees, support vector machines and neural networks. In their comparative analysis, they found that stacking and blending ensemble techniques offered higher prediction accuracies compared with bagging and boosting.

## 2.5 Comparison of Linear and Non-Linear Forecasting Methods

The literature argues that neural networks offer significant theoretical advantages over traditional statistical methods. Neural networks have been mathematically shown to be universal approximators of functions (Hornik *et al.*, 1989). Hence, neural networks can - in theory - approximate whatever functional form best characterises a time series. Neural networks are also inherently non-linear and so can estimate nonlinear functions considerably better than linear methods (Rumelhart and McClelland 1986). However, empirically their application and results against traditional methods have been long contested. Several studies have compared neural networks and traditional time series approaches. Most of these studies have used the data from the M-competition (Makridakis *et al.*, 1982), which includes 1001 real time series (Ahmed. *et al.*, 2010).

Looking at the original M-competition study, various groups of forecasters were given all but the most recent data points in each series. Each group consisted of forecasters who were regarded as experts in a particular technique. The groups were free to apply any approaches in. their. domain of competence to forecast the time series. After each group prepared its forecasts, Makridakis compared the forecasts to the actual values. The specific results of this competition are reported in Makridakis et al. (1982). One of the main conclusions from the competition, were that statistically sophisticated or complex methods do not necessarily produce more accurate forecasts than simpler ones. This inspired our decision to include a simple benchmark model to compare the performance our machine learning models.

In the years that followed the competition ,the data which contained all 1001 series have been made available. This sparked interest in the community and led to many studies using the data and empirically analysing the conclusions from the M-competition. The results of these studies have contrasted the findings of the competition, by demonstrating that neural network models in some cases appear to outperform simpler methods. Sharda and Patil (1990) made use of. a subset of the competition data. Their findings suggested that neural network models performed as well as the automatic Box-Jenkins (Autobox) procedure. Similarly, Tang *et al.* (1990), found that neural network models and Box-Jenkins models produce comparable results, when used on time series with a long history. In 1991 however, Foster *et al.* (1991) found that neural networks to be inferior to Holt's, Brown's, and the least squares statistical models for time series of yearly data. When comparing the models on quarterly data, they found the models to be comparable, using the competition data. Kang (1991), compared neural networks and Auto-box on the 50 M-competition series and found that it was the most appropriate technique for forecasting. An important finding from Kang's research was that the neural networks often performed better when predicting on a forecasting horizon a few periods ahead. This effect has also been noted when using neural networks to forecast macroeconomic data (White, 1994) and currency exchange rates (Zimmerman, 1994). Finally, Tim Hill *et al* (1996), conducted research on time series forecasts produced by neural networks and compared them with forecasts from six statistical time series methods generated in the M-competition (Makridakis *et al.*, 1982). Across monthly and quarterly time series, the neural networks did

significantly better than traditional methods. However, the neural network model and traditional models were comparable on the annual data.

More recently, a paper published by Binner *et al.* (2005), compared neural networks to traditional time series models; univariate autoregressive integrated moving average (ARIMA) and multivariate vector autoregressive (VAR) models. They did not use the competition data, however they found that the best models for their neural network outperform the traditionally used linear ARIMA and VAR models in macroeconomic forecasting.This is in line with findings from White (1994) and Zimmerman (1994). The authors attributed the gain in forecasting accuracy for their neural network models, to their capability to capture nonlinear relationships between macroeconomic variables (Binner *et al.*, 2005).

The results from the literature above are encouraging but equivocal; we were thus inspired to include a benchmark model to our comparative analysis of machine learning models - namely, an ARIMA model which, in fact, outperformed neural network models in the M-competition (Makridakis *et al.*, 1982).

## 2.6 Comparison of Machine Learning Forecasting Methods

Although there has been extensive research into the applications of individual machine learning methods for forecasting financial data, there does not exist as much research into comparisons of these existing methods. One of the few, was published by Ahmed NK *et al.* (2010). Their paper compared empirically the performance of several families of machine learning models for time series forecasting. More specifically, they compared in their study the performance of eight machine learning models with respect to their accuracy applied to a subset (1045 series) of the M3 competition data. The M3 competition is a sequel of M forecasting competitions, organised by Makridakis and Hibon (2000). The data builds on its predecessors and consists of 3003 business-type time series, covering various fields. In the study, they consider only the monthly time series.

Before computing the forecasts, they pre-processed the series in order to achieve stationarity in their mean and variance. Ahmed NK *et al.* (2010) calculated one-step-ahead forecasts for each one of the time series. To summarise the findings of the paper, the family of models which ranked the best turned out to be a multilayer perceptron, followed closely by a Gaussian processes and Bayesian neural network. The worst performing family of machine learning models was the radial basis functions. Our study extends this analysis, by looking at four different machine learning models, assessed on financial stock data.

# Chapter 3

# Exploratory Data Analysis

This section looks at the implementation of various tools to help decompose and analyse the structure of the datasets. An important part of statistics, involves the analysis, organisation, presentation, and summary of data.

## 3.1 Time Series Analysis

A time series is a sequence of observations in chronological order and forecasting is predicting the future using past data. In our project we implement several models for time series forecasting. The data we are concerned with are the log returns of three stocks. It is not often that the original data will be appropriate to use immediately for processing. Figure 3.1 looks at the evolution of the closing price for the three stocks - Ford Motors, Google and Motorola.

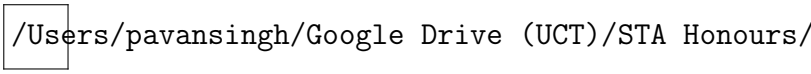



**Figure 3.1:** Closing price for Google, Ford and Motorola stocks between 2005 and 2015.

Figure 3.1 illustrates the tremendous growth Google has had during the period 2005 to 2015. In contrast, Motorola and Ford stock value has not experienced similar growth. In fact Ford stock value appears to have faced extended periods of negative growth between 2006 and 2010. The decrease of stock value between 2008 and 2010 is attributed to the Global financial crisis. Ford Motors stock value is considerably lower than Google, and hence their evolution of price is not clearly illustrated in the plot. A clearer representation for the evolution of each stock value is found in appendix B.

Given time series data, it is necessary to pre-analyse, as well as preprocess the time series. It is necessary to remove any known characteristics of the data which could hamper the performance of the models' forecast accuracy (Metcalfe, 2009). For example, figure 3.1 illustrates time series data with trend [1]. Trend and seasonality effects, mean that the behaviour of a time series is subject to repetition and change at the same time. While similar patters may repeat over time - such as periodic patterns due to a periodic influencing factor (e.g. day of the week or time of year for trading), the frequency and intensity of those are usually not constant. Another reason for eliminating trends and seasonalities (or, for that matter, any other clearly visible or well-known pattern) is that many traditional techniques such as ARIMA, require stationarity of the time series.

In our project we are concerned with predicting the movement of daily log returns, as such we computed the daily log returns for the three stocks. Figure 3.2 (pictured below) illustrates the daily log returns for Ford motors over the period 2005 and 2015. The daily log returns for Google and Motorola can be found in appendix B.

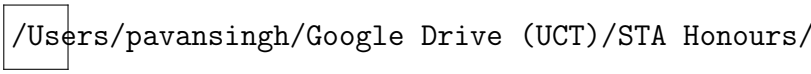



**Figure 3.2:** Log Daily Returns for Ford Motors stock from 2005 to 2015

[1] Trend refers to the phenomenon that the average value of sequence elements is constantly rising or falling.

The data shown in Figure 3.2 are typical of daily log return data for stocks. The mean of the series appears to be stable with an average return of approximately zero. As such, despite the individual large random fluctuations in daily log returns for Ford Motors, the series appears stationary, meaning that the nature of its random variation is constant over time. We also see volatility clustering, because there are periods of higher, and of lower, variation within each series that tend to be clustered together. The volatility clustering is not indicative of a lack of stationarity but rather can be viewed as a type of dependence in the conditional variance of each series (Metcalfe, 2009).

## 3.2 Stationarity

Stationarity is a property of a time series. A stationary series is one where the value of the series are not a function of time. That is, the statistical properties of the series like mean, variance and autocorrelation are constant over time (Metcalfe, 2009). It is important for the data to be stationary in order to avoid spurious regression - a regression that produces misleading statistical evidence of relationships between independent non-stationary variables. In order to receive consistent and reliable results, non-stationary data needs to be transformed into stationary data. Non-stationary data may be generated by an underlying process that is affected by a trend, a seasonal effect, presence of a unit root, or a combination of all three.

By computing the log returns, from visual inspection (shown in Figure 3.2) the time series appears to exhibit stationarity. However visual inspection is not satisfactory. To quantitatively determine if a given series is stationary or not, we used the Augmented Dickey Fuller test (ADF Test). Here, the null hypothesis is the time series possesses a unit root and is non-stationary (Metcalfe, 2009). The results of the test are shown below in Table 3.1.

| | Ford Motors | Google | Motorola |
|---|---|---|---|
| Test-Statistic | -8.564 | -17.097 | -8.920 |
| p-value | 8.556e-14 | 7.522e-30 | 1.044e-14 |

**Table 3.1:** Results from augmented Dickey Fuller test.

Across all stocks, we observe low p-values, hence we reject the null hypothesis and can proceed assuming our series are stationary. We look to achieve stationarity since it is necessary so that averaging lagged products over time will be a sensible. Moreover, the sample autocorrelation function becomes consistent - so we able to estimate autocorrelations with precision. And additionally, it is required for our ARIMA modelling. Our log transformation of the data, proved suitable to create a stationary series.

## 3.3 Data and Variables

The variables in the dataset used in this project is made up of OHLCV (open, high, low, close, volume) as well as several technical indicators. The kernel density estimation is a non-parametric way to estimate the probability density function of a random variable. To illustrate the distribution of log daily returns for Ford Motors, we use both a histogram and kernel density estimator.

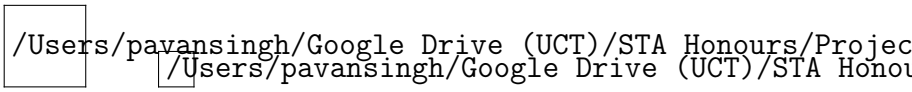



**Figure 3.3:** Histogram and kernel density estimator.

Visually the outliers may be difficult to see due to the large sample size. We see that the returns closely resemble a symmetric distribution. However, we can see that the distribution has rather narrow peak

to be normally distributed. This may suggest later exploring if the distribution is in fact normal. We can look at a QQ-plot to show the distribution of the daily log returns data against the expected normal distribution.

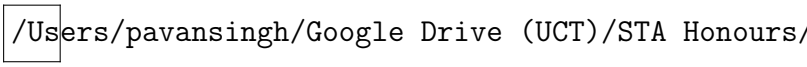



**Figure 3.4:** QQ-Plot of daily log returns for Ford Motors.

If the set of quantiles from the Ford data come from a normal distribution, then we should see the points forming a line that's roughly straight (Metcalfe, 2009). Figure 3.4 shows that this is not the case. We notice the points fall along a line in the middle of the graph, but curve off in the extremities. That is, the Q-Q plot highlights the fat tails of the distribution with extreme values more frequent than the normal distribution would suggest Normal Q-Q plots that exhibit this behaviour usually indicate that the data have more extreme values than would be expected if they truly came from a Normal distribution. Similarly, the daily log returns for Motorola and Google also display departures from normality's in the tails - see appendix B. We further explore this in the next section 3.4.

We look at some of the technical indicators included in the study on Ford Motors. The indicators are overlayed on the closing price for Ford closing price, 200 days prior to our testing set - the unseen data on which we assess our models.

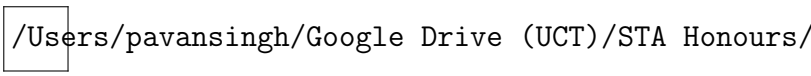



**Figure 3.5:** Several technical indicators on closing price of Ford Motors stock taken from the last 200 Days.

The Bollinger Bands are envelopes plotted at a standard deviation level above and below a simple moving average of the price. These help determine whether prices are high or low on a relative basis. We see there are a couple cases where the closing price deviates out of the envelope in the past 200 days. When stock prices continually touch the upper Bollinger Bands the prices are thought to be overbought; conversely, when they continually touch the lower band, prices are thought to be oversold, triggering a buy signal. A clear case of this signal working well, is during the last few days, where the closing price touched the lower band. Days later the stock closing price increased. These bands are used in pairs, both upper and lower bands and in conjunction with a moving average. The Moving Average essentially takes a specified number of past days, takes the average of those days, and plots it on the graph. For a 7-day moving average (MA7 on figure 3.5), it takes the last 7 days, and similarly, the 21-day moving average using the past 21 days prices. These short term moving averages are best suited for short-term trends and trading (Nelson *et al.*, 2017). The moving averages help with identifying trends to aid in identifying buy and sell signals. When the closing price breaks past an upwardly sloping moving average, this could indicate it is a good time to purchase the stock. The moving average can also act as a support line, that suggests that a closing price which approaches a support line may rally up again and thus signal a buy.

We also notice that the Ford Motors stock value has experienced substantial change in value over this 200 day period, reaching a high value of 12.77 and a low of 9.81.

## 3.4 Summary Statistics

By computing the summary statistics for our data, we can quickly understand the characteristics of our data sets. Table 4.2 shows the summary statistics for all three stocks.

In table 3.2, the mean daily log return for the three stocks are very small and close to 0. The range of returns appears to be quite high across all three stocks. The sample departures from the normal

| | Ford Motors | Google | Motorola |
|---|---|---|---|
| count | 2515 | 2515 | 2515 |
| mean | 0.000021 | 0.000658 | -0.000008 |
| SD | 0.029800 | 0.019708 | 0.023202 |
| min | -0.287682 | -0.123402 | -0.207639 |
| 25% | -0.012454 | -0.008286 | -0.009973 |
| 50% | 0.000000 | 0.000384 | 0.000150 |
| 75% | 0.012354 | 0.010165 | 0.009720 |
| max | 0.258650 | 0.182251 | 0.174097 |
| kurtosis | 13.1942306 | 9.106027 | 9.678477 |
| skewness | 0.0253992 | 0.360065 | -0.500873 |

**Table 3.2:** Summary statistics for Ford, Google and Motorola stocks from 2005 to 2015. Where SD refers to standard deviation and 25%, 50%, 75% refer to the lower, median and upper quartiles respectively.

distribution are summarised by the coefficients of skewness and Kurtosis. Formally, kurtosis is a measure of the combined weight of a distribution's tails relative to the centre of the distribution, while skewness is the measure of how much the probability distribution of a random variable deviates from the normal distribution (the skewness for a normal distribution is zero). If the kurtosis is greater than 3, then the dataset has heavier tails than a normal distribution (more in the tails). We see that Ford motors has the greatest kurtosis value. Google and Motorola have similar values for Kurtosis. In the previous section we noted that the distribution of the daily log returns for Ford Motors greatly departs from normality in the tails. We can further explore the tails of the distribution for daily log returns of all three stocks. This is illustrated by the box-plot in figure 3.6.

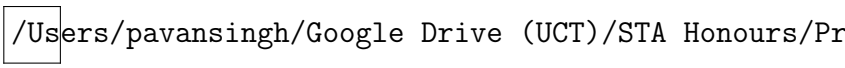



**Figure 3.6:** Box-Plot of Daily Log Returns for Ford, Google and Motorola stock from 2005 to 2015

We again notice that the mean is close to 0 amongst all the stocks and spread is generally small for the central part of the distribution - this indicates that the data are very peaked but long-tailed. Ford Motors daily log returns appears to have the greatest variation amongst the stocks. We notice a large proportion of extreme values for all the stocks. Although Ford Motors, has a noticeably greater range of values.

Finally, we can assess the association between the three chosen stocks. We explore a pairs plot, which is simply a matrix of scatterplots.

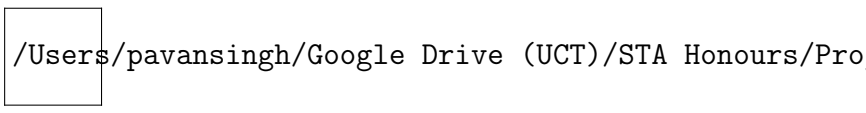



**Figure 3.7:** Pairs Plot of Log Daily Returns for Ford, Google and Motorola

Figure 3.7, lets us see both distribution of single variables and relationships between two variables. We note that Ford and Google exhibit a stronger relationship than Motorola, where higher returns for Google stock are associated with higher log returns of Ford Stock in general. These findings are confirmed by computing the correlation between these variables. The higher correlation value between Google and Ford indicates that the returns of the two time series data tend to move together.

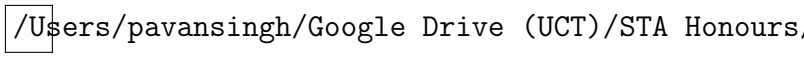



**Figure 3.8:** Correlation Heat Map of Log Daily Returns for Ford, Google and Motorola stocks for 2005 to 2015

## 3.5 Autocorrelation

After our time series has become stationary by transformation, the next step for fitting our ARIMA model is to determine whether AR or MA terms are needed to model any autocorrelation that remains in the transformed series. By looking at the autocorrelation function (ACF) and partial autocorrelation (PACF) plots of the series, we can tentatively identify the values of $p$ (AR) and $q$ (MA) terms that are needed for our ARIMA model. Autocorrelation is the correlation of a series with its own lags. The autocorrelation function )ACF) plot in figure 3.9 is shown below with the confidence band. The Partial Autocorrelation Function (PACF) conveys similar information but it conveys the pure correlation of a series and its lag, excluding the correlation contributions from the intermediate lags. We use the plots of these two measures to infer the parameterisation of our ARIMA model.

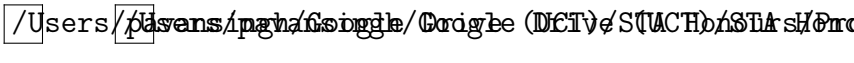



**Figure 3.9:** Autocorrelations and Partial Autocorrelation Functions for Log Returns of Ford Motors

The blue shaded region in figure 3.9 is the confidence envelope with default value of $\alpha = 0.05$. Anything within this range represents a value that has no significant correlation. The ACF and the PACF show similar patterns with autocorrelation at several lags appearing significant. The results of figure 3.9, indicate that returns may be autocorrelated till lag 2. We also note that lags 10, 13, 19 and 20 appear significant. The lags that appear to be significant, can be used to parameterise our ARIMA model. The order $q$ of the MA process of our ARIMA model is obtained from the ACF plot. We note that the PACF in figure 3.9 also has very few significant spikes. We use this figure to infer suitable parametrisation for the $p$ in our ARIMA model. We manually configured ARIMA models for different combination of values for $p$ and $q$ to achieve the best model. We chose to try values of $p = 0,1,2$ and $q = 0,1,2$. The length of the lag $p$ is to be chosen so that the residuals are not serially correlated. For examining the information criteria for choosing lags, we went about looking to minimise the Akaike information criterion (AIC). The PACF and ACF plots of the other stocks can be seen in appendix B.

# Chapter 4

# Methodology

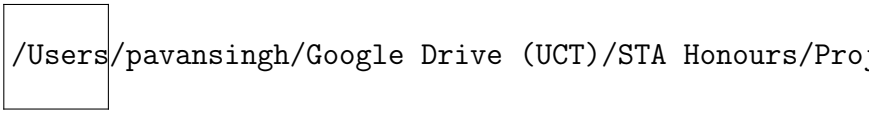



**Figure 4.1:** Process of Comparative Analysis of Non-Linear Machine Learning Models

Our approach to this project begins by collecting the stock data. The collected data is then cleaned by checking for errors by examining day to day changes, null values or abnormalities (like negative prices) and missing values. We then proceeded onto variable selection for our models. Each model will be fed the same variables to ensure consistency and allow for a comparison of performance. Data preprocessing refers to analysing and transforming the input and output variables. This greatly assists the neural networks in learning the relevant patterns (James, 2013). Since neural networks are pattern matchers, the representation of the data is critical in designing a successful network (Heaton *et al.*, 2017). We then proceeded to generate training, testing and validation sets using our data set. Following this involved, selection of appropriate parameters and implementing the various models.

## 4.1 Neural Networks

A simple artificial neural network (ANN) is typically organised into three sections: an input layer, a set of hidden layers, and an output layer - as shown in figure 4.2. The structure of ANN classifies into many types of architectures such as a Single layer, Feed-forward, and Recurrent networks (Agrwal *et al.*, 2013).

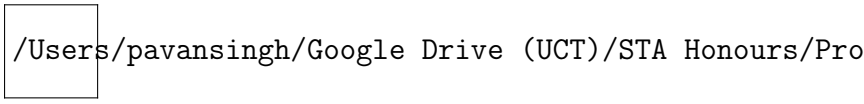



**Figure 4.2:** Simple Artificial Neural Network, with an input layer, a hidden layer and an output layer

Each layer of a simple ANN can contain one or more neurons. The first layer (input layer) is the layer connected to the input data. After that there could be one or more middle layers called hidden layers. The last layer is the output layer which shows the results (Agrwal *et al.*, 2013).

The hidden and output layers are made up of a few interconnected nodes. These interconnected nodes contain an activation function which help the network learn complex patterns in the data (James, 2013). Hidden layers allow neural networks to perform non-linear mapping between the input and the output (Agrwal *et al.*, 2013). Models can grow more deep by increasing the number of hidden layers and neurons in a given layer. This will also increase the mapping complexity (Heaton *et al.*, 2017). The connections between neurons carry weights and some biases connected to each neuron. Simply, an ANN has weights associated with each input neuron and bias which also carries weight. An activation function is applied over the net input to calculate the output (Agrwal *et al.*, 2013). The output is then compared to the target and weights are adjusted.

A neural network is defined in terms of a recursive updating equation. To move forward through a network , called a forward pass, the network uses a formula 4.1 to calculate each neuron in the next layer.

$$a^{(l)} = \sigma\left(Wa^{l-1} + b\right) \tag{4.1}$$

We note, neurons for activations $a$, weights $w$ and biases $b$ which are summed in vectors. This equation is responsible for taking us forward, until we get an output (Heaton *et al.*, 2017). With that output $\hat{y}$, we can measure how good it is by using our cost function $C$ (4.2). We compare the $\hat{y}$ from our network with the result we wanted in the output layer $y$ (Heaton *et al.*, 2017). This is done for every example in the data. The cost function which is commonly used for binary classification problems, and which is opted for in this paper, is called binary cross-entropy function (MSE):

$$\text{Binary Cross Entropy } = \frac{1}{N}\sum_{i=1}^{N} -\left(y_i{}^{\star}\log\left(p_i\right) + \left(1-y_i\right)^{\star}\log\left(1-p_i\right)\right) \tag{4.2}$$

Where, where $y$ is the label (1 or 0) and $p(y)$ is the predicted probability for label for all N points. It represents the average logarithmic error.

Now, we use our first result to go back and adjust the weights and biases. This is done so as to optimise the cost function - this is called a backwards pass (Heaton *et al.*, 2017). Essentially, in the backward phase, we try to adjust the whole neural network, so that the output value is optimised (Heaton *et al.*, 2017). We keep trying to optimise the cost function by running through new observations from our dataset. To update the network, we calculate gradients, which can be thought of as small updates to individual weights in each layer. The gradients are simply partial derivatives of the cost function, shown below:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial w^{(L)}}. \tag{4.3}$$

## 4.2 Feed-Forward Neural Network

The architecture and process outlined in section 4.1 are representative of a multilayer perceptron neural network. It is the simplest form of an ANN, and is also the most popular type of neural network used (Gurney, 2018). In feedback networks, in contrast with recurrent networks (discussed next in section 4.3) all the connections are toward the output layer. The network structure for a feedforward neural network can be summarised by the updating equation 4.4.

$$a_j^l = \sigma_l\left(\sum_{k=1}^{d_{l-1}} a_k^{l-1} w_{kj}^l + b_j^l\right), \quad l = 1,2,\ldots,L; j = 1,2,\ldots,d_l \tag{4.4}$$

Where,

- $\sigma_l(.)$ denotes an activation function on layer $l$ (e.g., sigmoid or hyperbolic tangent),
- $d_{l-1}$ denotes the number of nodes in layer $l-1$ (the number of nodes in layer $l$ is $d_l$ ),
- $w_{kj}^l$ denotes the $kj$-th weight parameter linking the $k$-th node in layer $l-1$ and $j$-th node in layer $l$ (although we transfer information from layer $l-1$ to layer $l$, we use the convention of indexing the weights using the superscript $l$-corresponding to the forward layer),
- $b_j^l$ denotes the $j$-th bias in layer $l$,
- and the equation is evaluated subject to the initial conditions $a_j^{(0)} = x_{ij}$ for all $j$ at the $i$-th training example (Pienaar, 2021)

There are many factors which should be discussed under the architecture of a neural network, apart from the weights, whose values cannot be estimated from the data - these are known as hyper-parameters. Before the learning process can begin, the value of a hyper-parameter must be determined.These include the number of input variables, the number of hidden layers and hidden nodes, the types of activation functions in the hidden and output layers and the value of the learning rate, amongst others (Agrwal *et al.*, 2013). We outline the specific model specifications and architectures in chapter 5.

With respect to the learning of the model, we are concerned with finding optimal hyper-parameters to fine to the model's complexity so as to control on the out-of-sample performance of the model. The hidden layer(s) provide the network with its ability to generalise. In theory, any continuous function may be approximated by a neural network with one hidden layer and a sufficient number of hidden neurons (Heaton *et al.*, 2017). When we consider increasing the number of hidden layers, we need to also identify. the increased computation time and increased risk of overfitting [1]. This can ultimately lead to poor out-of-sample forecasting performance. Few hidden nodes are usually recommended because there is less chance of overfitting (Agrwal *et al.*, 2013). However, a neural network with too few nodes may not be able to learn the data's richness - leads to a model that under-fits the data.

We also experiment with different optimisers and activation functions, so as to determine the appropriate model architecture and fit. Experimenting over the whole parameter space of all the parameters is beyond the scope of the paper. In this study, therefore, we focus on experimenting with the different values of key hyper-parameters, making use of a grid search. Grid search builds a model for every combination of hyper-parameters specified and evaluates each model (Gorr et al., 1994).

## 4.3 Long Short-Term Memory Neural Network

### 4.3.1 Overview of Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a class of Artificial Neural Network where connections between neurons form a directed cycle (Heaton *et al.*, 2017). In essence, these types of ANNs are designed in such a way to handle sequence dependence, as such, it has achieved great success in the field of sequence data (Che et al, 2015). Hence, RNNs in theory are perfectly suited for predicting movement of daily log returns of financial assets. According to Sak *et al.* (2014), LSTM neural networks have outperformed the Deep Neural Networks (DNNs) and the simple RNN models for predicting the movements in stock data.

A Simple RNN has a similar structure to the FFNN, but contains feedback connections (Che *et al.*, 2018). That is, instead of having only neurons with connections from the input to the output, it also has neurons with connections from the output, again to the input. This is shown in equation 4.5. This additional connection can be used to store information over time providing the network with dynamic memory (Heaton *et al.*, 2017). So, the current state of the network is a function its previous steps. Therefore, unlike feed-forward networks, they can use the information from the past to handle and predict sequential data.

$$\begin{aligned} h_t &= \tanh\left(\omega_h h_{t-1} + \omega_x x_t\right) \\ y_t &= \omega_y h_t \end{aligned} \tag{4.5}$$

where,

1. $x_t -$ input state

[1] Overfitting refers to the scenario where a model fits too closely - drawing too many characteristics - to the training data, and becomes unable to generalise well to unseen data.

2. $y_t$ – output state
3. $h_t$ – current timestamp
4. $h_{t-1}$ – previous timestamp
5. $\omega_h$ – weight associated with the previous timestamp
6. $\omega_x$ – weight associated with the current input
7. $\omega_y$ – weight associated with the output

### 4.3.2 Long Short-Term Memory Neural Networks

LSTMs are a type of RNN, introduced by Schmidhuber in 1997, to overcome the vanishing gradient problem, which standard RNNs are susceptible to, by controlling the way information flows in the network (Schmidhuber, 1997). More precisely the model structure of an LSTM is defined so as to retain recurrence but to ensure that backpropagation does not degenerate in the practical sense.

RNNs have have the property of being extremely deep, and after every iteration in the neural net, the data it holds gets vanished going deeper. LSTM does not use any activation function within its recurrent component. Thus thus the stored value is not iteratively modified. Hence the gradient does not tend to vanish when trained with backpropagation and thus resolves the gradient vanishing problem.

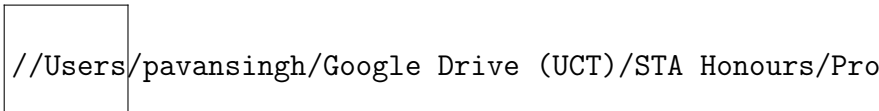



**Figure 4.3:** LSTM Layer in Model.

LSTMs comprise of gates and a memory cell to analyse and process long term sequences such as time series. The network learns from both short term and long term memory. This is ensured, as the LASTM cell enforces a constant error flow through the designed cell state. The hidden layer is replaced by a complex block (see Figure 4.3). The complex block composed of computing units consisting of gates. These gates trap the error in the block. This structure makes the LSTM capable of learning long-term dependencies.

The gates in the LSTM block include:

- One forget gate which decides what information is going to be remembered/thrown away for the new state.
- An input gate which determines which portion of a specific point in time is to be added.
- An output gate that controls what is going to be the next output.
- Sigmoid layer generates numbers between zero and one, describing how much of each component should be let through.
- Tanh layer generates a new vector, which will be added to the state (Nelson *et al.*, 2017).

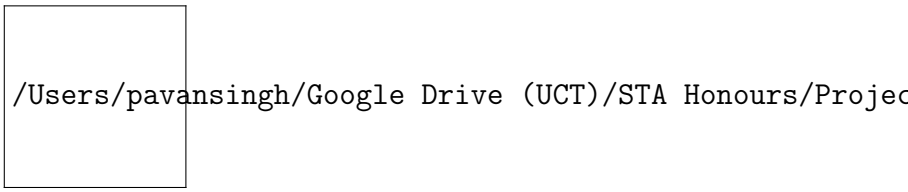



**Figure 4.4:** LSTM forget gate.

The forget gate (see figure 4.4) takes the previous hidden state $h_{t-1}$ and current input $x_t$. Then by applying sigmoid function $\sigma$ returns output $f_t$ between 0 (forget) and 1 (keep). The output from the forget gate is multiplied with previous cell state $C_{t-1}$ to drop irrelevant information (Nelson *et al.*, 2017).

$$f_t = \sigma\left(\omega_f\left[h_{t-1}, x_t\right] + b_f\right) \tag{4.6}$$

The input gate (see figure 4.5) updates the cell state $C_t$ (Nelson *et al.*, 2017). The sigmoid function $\sigma$ determines which values of previous hidden state $h_{t-1}$ and current input $x_t$ will be updated by returning

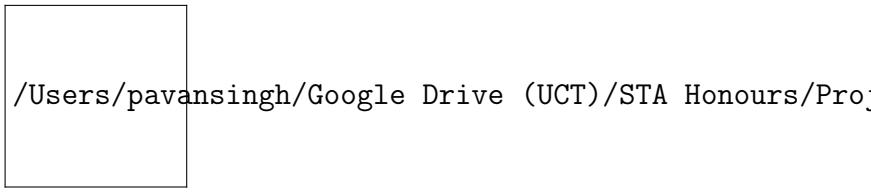



**Figure 4.5:** LSTM input gate.

the value $i_t$ between 0 (do not update) and 1 (update). Then the tanh gives weightage to the passed values and returns the level of importance $\tilde{C}_t$ ranging from $-1$ to 1, which also helps to regulate the network. Then $\tilde{C}_t$ is multiplied by $i_t$ which determines which outputs form tanh should be used to update the cell state $C_t$ (Nelson *et al.*, 2017).

$$\begin{aligned} i_t &= \sigma\left(\omega_i\left[h_{t-1}, x_t\right] + b_i\right) \\ \tilde{C}_t &= \tanh\left(\omega_c\left[h_{t-1}, x_t\right] + b_C\right) \end{aligned} \tag{4.7}$$

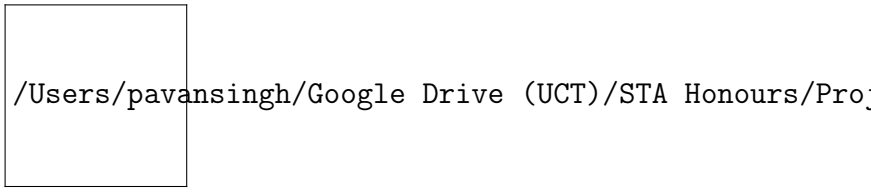



**Figure 4.6:** LSTM output gate.

Finally, the output gate (see figure 4.6) decides what the next hidden state should be (Nelson *et al.*, 2017). It applies sigmoid function $\sigma$ to resolve which states to let through $o_t$ and tanh regularises the current cell state $C_t$. Multiplication of both outputs determines which information the hidden state should carry (Nelson *et al.*, 2017).

The advantage of a LSTM network is that it can convey information gathered at an early stage and readily store it in memory for a long time, allowing for the generation of future long-distance dependencies as emphasised by Chung. J (2014). In contrast to a FFNN, in an LSTM network, each node is used as a memory cell that can store other information, whilst in a FFNN each node is a single activation function.

Like any other neural network, an LSTM needs to be trained, so as to be able to learn the features of the modelled task and be able to predict. This process consists in computing the weights and biases of the LSTM by minimising an objective function, MSE, through some optimisation algorithms - we opted for SGD. Once the model is trained on the training set and validated on a validation set, it is then tested on a real out of sample testing. Finally, we note that our LSTM model is constructed as a supervised binary classification model, using 42 features, to predict the movement of three stocks.

## 4.4 Ensemble Decision Trees

Decision trees are a type of machine learning technique that uses decision rules learned from training data to predict the value of a target variable. Each node in a decision tree represents a question leading to a yes-no answer for traversing the respective left and right nodes (Ampomah *et al.*, 2020).

There are two groups of ensemble methods. These are usually identified depending on how they integrate the results for single ensemble prediction.

- Averaging Methods: train several base estimators independently and then average their predictions (Ampomah *et al.*, 2020).

- Boosting Methods: train base estimators sequentially with the specific goal to reduce the bias of the combined estimator (Ampomah *et al.*, 2020).

The Random Forest and AdaBoost algorithms represent some of the most popular ensemble machine learning methods and are just extensions of the simple decision trees algorithm.

## 4.5 Random Forest

For a variety of machine learning applications, decision trees can be used (Ampomah *et al.*, 2020). However, trees that are grown deep to learn highly irregular patterns tend to overfit the training sets (Ampomah *et al.*, 2020). Noise in the data, can lead to the tree to grow very differently, since decision trees inherently have low bias and high variance. Random Forest's overcome this problem, since it trains multiple decision trees using different combination of features. This is done at the cost of slightly increased bias. The multitude of multitude of decision trees are then merged together to obtain a more accurate prediction, as illustrated by figure 4.7. The Random Forest algorithm is summarised in the appendix B.

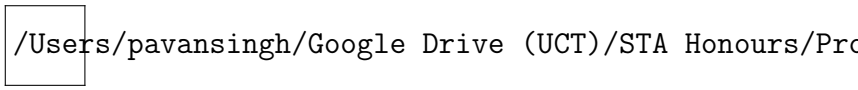



**Figure 4.7:** Illustration of how Random Forests train multiple trees and then merge these trees to obtain a single prediction.

In a Random Forest, the data is recursively split into partitions. The tree makes use of recursive binary splitting to partition the feature space in such a way that the resulting residual sum of squares (RSS) is minimised at each step - for regression trees. Here RSS is the criterion for splitting at each node. The choice for the splitting criterion is based on some impurity measures such as Shannon Entropy or Gini impurity for classification problems. The approach is to choose the best splitting decision at a node such that reduction in impurity or RSS is as much as possible.

In Random Forests, the goal of randomising the features in addition to the training observations is to further de-correlate the prediction errors of the individual trees. All features are not created equal, and a small number of highly relevant features will be selected much more frequently and earlier in the tree-construction process, making decision trees more alike across the ensemble. However, the less the generalisation errors of individual trees correlate, the more the overall variance will be reduced.

## 4.6 Adaptive Boosting

### 4.6.1 Boosting

Boosting is another class of ensemble decision tree algorithms that involve combining the predictions from many weak learners (Ampomah *et al.*, 2020). A weak learner is a model that is very simple, although has some skill on the data (Ampomah *et al.*, 2020). This is done by building a model from the training data, then the subsequent models attempts to correct the errors from the previous model.

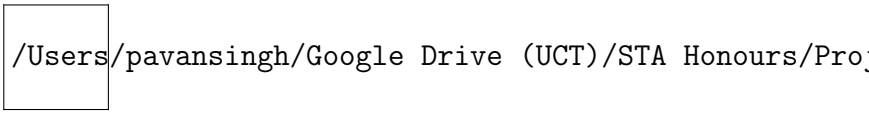



**Figure 4.8:** Boosting algorithm for illustrated. The weakness depicts the errors. The subsequent models attempt to correct the errors from previous models. The models $1, 2, 3, \ldots, N$ are individual models that can be decision trees.

### 4.6.2 AdaBoost Algorithm

Adaptive boosting, or AdaBoost, is a type of boosting technique, which involves using very short (one-level) decision trees as weak learners that are added sequentially to the ensemble (Ampomah *et al.*, 2020).

AdaBoost is a significant departure from Random Forests, which builds ensembles of deep trees to reduce bias - there is no fixed depth in a random forest. AdaBoost, in contrast, grows shallow trees as weak learners - of depth 1. The algorithm only makes a node with two leaves, known as a stump.

The algorithm starts with an equal-weighted training set and then successively alters the sample distribution (Ampomah *et al.*, 2020). After each iteration, AdaBoost increases the weights of incorrectly classified observations and reduces the weights of correctly predicted samples so that subsequent weak learners focus more on particularly difficult cases. The process continues until a pre-set number of weak learners have been created (a user parameter) or no further improvement can be made on the training dataset. Once completed, you are left with a pool of weak learners each with a stage value. Once trained, the new decision tree is incorporated into the ensemble with a weight that reflects its contribution to reducing the training error (Ampomah *et al.*, 2020).

The process of Adaboost can be summarised as follows:

1. Creating the First Base Learner
2. Calculate the Total Error
3. Calculate Performance of the Stump
4. Update weights
5. Create a new data set

## 4.7 ARIMA Model

An Auto Regressive Integrated Moving Average, or ARIMA, is a class of time series models. An ARIMA model explains a given time series based on its own past values (lags and lagged forecast errors) so that it can be used to forecast future values (Zhang, 2003).The model is characterised by 3 parameters: $p$, $d$, $q$. The parameter $p$ is the order of the auto regressive (AR) term. It refers to the number of lags of $Y$ to be used as predictors. While, $q$ is the order of the moving average (MA) term, which refers to the number of lagged forecast errors that should go into the ARIMA Model. and d is the number of differencing required to make the time series stationary. So ARIMA is essentially a time series method which combines terms of AR (autoregressive) and MA (moving average) (Zhang, 2003).

The following equations depict ARIMA modelling:

$$Y_t = \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_3 Y_{t-3} + \cdots + \beta_0 Y_0 + \varepsilon_t$$
$$Y_{t-1} = \beta_1 Y_{t-2} + \beta_2 Y_{t-3} + \cdots + \beta_0 Y_0 + \varepsilon_{t-1}$$

where,

- $Y_t =$ function of lags of $Y_t$
- $\beta_t =$ coefficient of the lag $t$ that model estimates
- $\varepsilon_t =$ the errors from the equation (Zhang, 2003).

## 4.8 Backpropagation

Backpropagation revolutionised the landscape of machine learning with its introduction to neural networks in 1986 by Rumelhart *et al.* (1986). Backpropagation is simply a way for a network to calculate the gradient of a loss function with respect to all the weights in the network. The expression for the partial derivative ($\frac{\partial C}{\partial w}$) of the cost function $C$ with respect to any weight $w$ (or bias $b$ ) in the network, serves as the fundamental expression which allows for the gradients to calculated. When we adjust the weights and biases, the expression informs us how rapidly the cost changes. The method of backpropagation helps fine-tuning the weights of a neural network based on the error rate obtained in the previous iteration (Nelson *et al.*, 2017). Essentially, backpropagation method helps calculate the gradient of a loss function with respect to all the weights in the network. This plays a key role in the learning of the model.

Backpropagation algorithm consists of 2 steps:

1. Forward pass (feed forward) the values
2. Calculate the error and propagate it back to the earlier layers.

In this algorithm, the error between the actual output and target is propagated back to the hidden unit. For minimising the error, the weights are updated. To update the weights the error is calculated at the output layer .

## 4.9 Learning Algorithms and Training

### 4.9.1 Optimisers

Optimisers is how the neural networks learn. In our neural network setup, we need an optimisation algorithm which best finds the value of the parameters (weights) that minimise the error when mapping inputs to outputs. There are several optimisers available, and the choice of optimiser can affect the accuracy of the models, as well as the speed of training (Rumelhart, 1995). Some of them include:

- Adam (adaptive moment estimation)
- RMSprop (root mean square propagation)
- SGD (stochastic gradient descent)

Stochastic gradient descent (SGD) is an optimisation algorithm for minimising the loss of a predictive model with regard to a training dataset.More specifically, SGD is a form of gradient descent that works by using an iterative process to estimate the gradient towards minimising an objective loss function. The stochastic term comes about as samples are chosen at random. When lesser iterations are used, bigger steps are taken to reach the solution, and the model is said to have a high learning rate. Likewise, with more iterations, smaller steps are taken, resulting in a model with a small learning rate.

In SGD, we take a batch of random sample and perform an update to weights and biases based on the average gradient from the batch. There exist many variations of stochastic gradient descent.

Below is the equation 4.8 for SGD, which is used to update parameters in the neural networks. Equation 4.8 is used to update the parameters of our network. This occurs in the backward pass. Backpropagation is used to calculate the gradient $\nabla$).

$$\theta = \theta - \eta \cdot \overbrace{\nabla_\theta C(\theta; x, y)}^{\text{Backpropagation}} \tag{4.8}$$

Where:

- $\theta$: a parameter. This can represent the weights, biases and activations.
- $\eta$ is the learning rate (eta)
- $C$: cost function
- $\nabla$: gradient (nabla), which is taken of $C$.
- : learning rate

Equation 4.8 indicates that we take each parameter theta $\theta$ and update it by taking the original parameter $\theta$ and subtract the learning rate $\eta$ times the ratio of change $\nabla C(\theta)$ (Agrwal *et al.*, 2013).

/Users/pavansingh/Google Drive (UCT)/STA Honours/

**Figure 4.9:** Stochastic Gradient Descent

The process of stochastic gradient descent can be summarised below:

1. Define a cost function
2. Start at a random point along the $x$-axis. Then the algorithm then looks to find a step to decrease the cost function most quickly
3. Calculate the gradient using backpropagation
4. Step in the opposite direction of the gradient.

The goal is to reach global minima (Agrwal *et al.*, 2013). However, often this is not possible. More often than not, we will find a local minima (see figure 4.9).

### 4.9.2 Training

Iteratively presenting examples of the correct known answers (targets) to a neural network, in order to learn patterns in the data is how a neural networks. trained. The goal of training is to try find the set of weights that. determine the global minima of the cost function (Heaton *et al.*, 2017). For this paper, the training to testing data ratio has been kept as 95:5 in this paper. We achieve this by dividing the time series into three distinct sets called the training, testing, and validation (out-of-sample) sets (Heaton *et al.*, 2017). The training set is used to train the neural network to learn the underlying model form the data. The validation set is used in our analysis to avoid overtraining of neural networks. And finally, our models are assessed on their predictive ability using the test set.

## 4.10 Activation Function

An activation function is part of an artificial neuron that transforms the sum of weighted inputs into another value for the next layer (Heaton *et al.*, 2017). An artificial neuron is said to be activated when it passes a non-zero value to another neuron (Agrwal *et al.*, 2013). There are several types of activation functions, some include:

- sigmoid
- hyperbolic tangent (tanh)
- rectified linear unit. (ReLU)
- softmax

## 4.11 Data

The dataset used in this paper is Ford Motors, Google and Motorola Inc stock data scrapped from the Yahoo Finance Web site. Our data includes open price, close price, high price, low price and trading volume for the day. The data ranges from 2005 to 2015. We computed the log returns for the daily stocks for each company, according to the following equation:

$$\text{Log Daily Return} = \log\left(\frac{\mathbf{P_d}}{\mathbf{P_{d-1}}}\right) \tag{4.9}$$

Where, $P_d$ : Closing Price for current day d and $P_{d-1}$ : Closing Price of previous day $d-1$.

Knowing which input variables are important for forecasting stock movements is pivotal. Although neural networks possess the ability to detect complex non-linear relationships among a number of different variables, however, for interpretability and reducing complexity, economic theory can greatly assist in choosing variables which are likely important predictors. We used the literature on trading and technical analysis, to guide us on which technical indicators and variables to use in our analysis. A popular approach, and one which we adopted, is to calculate various technical indicators which are based only on past prices (and occasionally volume and open price) of the market being forecasted.

The dataset originally had six features, namely Open, High, Low, Close, Adjusted Close, Volume. The generated technical indicators were added later. The specific list of technical indicators and description of them can be found in appendix B.

An extract of the dataset for one stock (Ford Motors) used before the technical indicators were incorporated is shown in Table 4.1.

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2005-01-03 | 14.66 | 14.75 | 14.51 | 14.71 | 9.271831 | 9852200 |
| 2005-01-04 | 14.71 | 14.75 | 14.59 | 14.66 | 9.240314 | 9035400 |
| 2005-01-05 | 14.63 | 14.66 | 14.42 | 14.43 | 9.095345 | 11376200 |
| 2005-01-06 | 14.40 | 14.52 | 14.37 | 14.45 | 9.107951 | 6672600 |
| 2005-01-07 | 14.48 | 14.65 | 14.45 | 14.65 | 9.234013 | 11452500 |

**Table 4.1:** Extract of Data Set Before Adding Technical Indicators

- Date: the date of the day on which the following trades were done
- Open: the price of a single stock at the opening of the market
- High: the highest price traded for the day
- Low: the lowest price traded for the day
- Close: the price of the single stock at the closing of the market
- Adjusted Close: the adjusted closed price reflects the closing price of a given stock accounting for any corporate actions
- Volume: refers to the amount of shares traded on that particular day

We also made sure to undertake data preprocessing, which refers to analysing and transforming the input and output variables to minimise noise, highlight important relationships, detect trends, and flatten the distribution of the variable to provide suitable input for our neural network models.

## 4.12 Performance Criteria

There are several measures of accuracy but each of them has advantages and limitations (Makridakis et al., 1983). As a result, none of them is commonly acknowledged as the best measurement. Hence, in our paper we make use of a number of performance measures to assess our models. Three measures are used to compare the forecasting accuracy of the models; accuracy, F1 score. and AUC. All four models were assessed using these three metrics and the results were reported for each of the three stocks. The results can be summarised in a confusion matrix. A confusion matrix, or error matrix, is a square matrix that helps to visualise and describe the performance of a classification model for which the true values are known. We can extract the true positive and true negative metrics which can be used to compute the performance criteria.


**Figure 4.10:** Confusion Matrix

### 4.12.1 Accuracy

Accuracy is the quintessential classification metric (James, 2018). It is well suited for binary as well as a multi-class classification problem. Accuracy is the proportion of true results among the total number of cases examined. (Heaton *et al.*, 2017). It summarises the performance of a respective model as the number of correct predictions divided by the total number of predictions. That is:

$$\text{Accuracy } = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.10}$$

Where $TP$ = True Positives, $TN$ = True Negatives, $FP$ = False Positives, and $FN$ = False Negatives.

### 4.12.2 F1 Score

Another metric we used, is the F1 Score. This measure is a number between 0 and 1 and is the harmonic mean of precision and recall (Nelson *et al.*, 2017). Precision looks at what proportion of positive identifications was actually correct, while recall looks at what proportion of actual positives was identified correctly (Nelson *et al.*, 2017). Since, the F1 score is the weighted average of Precision and Recall, this score takes both false positives and false negatives into account. An F1 score reaches its best value at 1 and worst value at 0. A low F1 score is an indication of both poor precision and poor recall. The formula of the F1 score is:

$$F_1 = 2 \cdot \frac{\text{precision } \cdot \text{ recall}}{\text{precision } + \text{ recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP}+\text{FN})} \tag{4.11}$$

### 4.12.3 AUC ROC

The final measure we used is the AUC and ROC. This indicates how well the probabilities from the positive classes are separated from the negative classes. (Nelson *et al.*, 2017).

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds (Nelson *et al.*, 2017). This curve plots two parameters:

- True Positive Rate
- False Positive Rate

More specifically the curve computes and plots all the combinations of true positive rates (TPR) and false positive rates (FPR) that result from producing predictions using any of the predicted scores as a threshold. The curve allows us to visualise, organise, and select classifiers based on their performance. A classifier that makes random predictions (taking into account class imbalance) will on average yield TPR and FPR that are equal so that the combinations will lie on the diagonal, which becomes the benchmark case.

The area under the curve (AUC) is defined as the area under the ROC plot that varies between 0.5 and the maximum of 1. It is a summary measure of how well the classifier's scores are able to rank data points with respect to their class membership. More specifically, the AUC of a classifier has the important statistical property of representing the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. In addition, the AUC has the benefit of not being sensitive to class imbalances. We opted for AUC since it Is scale-invariant and classification-threshold-invariant. That is, it measures how well predictions are ranked, rather than their absolute value.

# Chapter 5

# Model Specification and Analysis

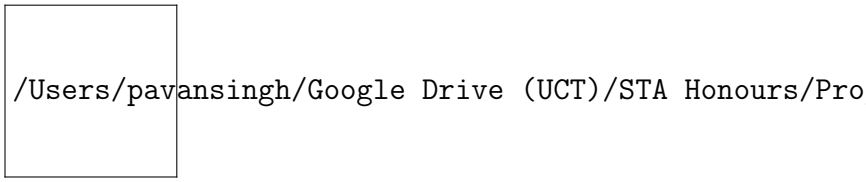



**Figure 5.1:** Diagram illustrating the models and whole experimental procedure followed.

All 42 variables, including mix of stock data (open, high, low, close and volume), as well several technical indicators. were used as inputs to our. models. Our feed-forward and long-short term memory neural networks were configured as classification models, while our ensemble trees, where configured as regression problems. As such, since we want to assess these models on their ability to predict the movements of daily log returns of stocks, we had to convert our predictions from our ensemble models to binary output. This was done, by coding returns equal to or greater than zero as one and any negative returns as zero.

## 5.1 Model Architectures

Hyper-parameters are the parameters that are set prior to the learning process (Nelson *et al.*, 2017). They are not derived via training. Hyper-parameter optimisation allows for improving and - in some cases - correcting an algorithm's performance. One of the main characteristics to consider when optimising hyper-parameters is the model's complexity (Nti *et al.*, 2020). If the model is over-complex, it might be prone to overfitting and thus will not generalise well. On the other hand, if it is too simplistic, the model might not fit the data correctly or be able to capture the relationship between input and output sufficiently under-fitting (Nelson *et al.*, 2017).

There are several approaches to find the model hyper-parameters that yield the best score on the validation set metric. In our paper, we opted for using a grid search and in some cases, manually searching for optimal hyper-parameters.

### 5.1.1 Long Short-Term Memory Neural Network

The network is based on a sequential model. We used a cross validation grid search [1] from Scikit-Learn module to try out several values for our hyper-parameters and compare the results.

Hyper-parameters tuned with grid search, with the accompanied combination of values tried, is shown in the table below.

- `batch_size`: defines the number of samples that will be propagated through the network
- `dropout`: a regularisation technique, which helps with reducing overfitting
- `LSTM units`: number of neurons in a LSTM layer

[1] Grid Search CV tries all combinations of parameters grid for a model and returns with the best set of parameters having the best performance score

| Hyper-parameter | Value |
|---|---|
| Batch Size | 4, 12, 24 |
| Dropout | 0.2, 0.3, 0.4 |
| Learning Rate | 0.1, 0.01, 0.001 |
| LSTM Units | 20, 50, 100 |
| Dense Units | 1, 5, 20 |

**Table 5.1:** Hyper-Parameters and their corresponding combination of values used in grid search for our LSTM.

- `Dense units`: number of neurons in dense layers
- `Learning Rate`: controls how much to change the model in response to the estimated error each time the model weights are updated

Our final model has two LSTM layers, each followed by a dropout layer with dropout value of 0.2. Dropout is a regularisation technique for neural network, where randomly selected neurons are ignored during training. As such, the contribution of those selected neurons to the activation of downstream neurons is temporally removed on the forward pass. Thus any weight updates are not applied to the neuron on the backward pass. Ultimately, this helps in reducing the possibility of over-fitting the network (Nelson *et al.*, 2017). In addition, we have two dense layers in our model architecture. The dense layer is the most frequently used layer which is basically a layer where each neuron receives input from all neurons in the previous layer (Nelson *et al.*, 2017).

The loss function used was the binary cross entropy function - as it was appropriate the type of problem we have - supervised binary classification. With regards to the number of neurons (units) in the hidden layers, there is no straightforward approach one should use. We chose to grid search with 20, 50 and 100 units in the LSTM layers. The final model has 100 units in the LSTM layer.

With respect to the models fit and architecture, we also consider the activation function and choice of optimiser. We used both a rectified linear unit (ReLU) and sigmoid activation function in the final model architecture. The ReLU activation function is specified for all except the output layer. The sigmoid activation function is used in the final output layer, satisfying our binary classification solution. The optimiser used for this model was stochastic gradient descent (SGD).

Our grid search results suggested we use a learning rate of 0.01. We note that setting a higher learning rate accelerates the learning but the model may not converge. Conversely, a lower rate will slow down the learning drastically as steps towards the minimum of loss function will be tiny, but will allow the model to converge smoothly. Finally, we specified 100 epochs [2] and set a stopping method, which stops training once the model performance stops improving by a pre-set threshold on the validation set. The final model architecture is summarised in the table below:

A rectified linear unit (ReLU) function is written as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{5.1}$$

The ReLU activates a node with the same input value only when the input is above zero.

### 5.1.2 Feed-Forward Neural Network

The hyper-parameters for our feed-forward neural network (FFNN), are similar to that of our LSTM model. However, we do not have dropout layer nor do we have LSTM layers in our architecture for the

[2] the number of iterations (forward and back propagation) our model needs to make.

| Hyper-parameter | Value |
|---|---|
| Epochs | 100 |
| Batch Size | 12 |
| Dropout | 0.2 |
| Dropout Layers | 2 |
| Activation Function | ReLU and Sigmoid |
| Learning Rate | 0.01 |
| Optimiser | Binary Cross Entropy |
| LSTM Units | 100 |
| LSTM Layers | 2 |
| Dense Layers | 2 |
| Dense Units | 5 and 1 |

**Table 5.2:** Chosen hyper-parameters for LSTM. We used a grid search with cross validation to select the number of units in the LSTM layer, the dropout, learning rate and batch size.

FFNN. We again used a grid search with cross validation from Scikit-Learn module to try out several values for our hyper-parameters and compare the results. The combination of values in the grid are shown in Table 5.3.

| Hyper-parameter | Value |
|---|---|
| Batch Size | 4, 12, 24 |
| Learning Rate | 0.1, 0.01, 0.005, 0.001 |
| Dense Units | 1, 10, 20, 50 |

**Table 5.3:** Hyper-Parameters and their corresponding combination of values used in grid search for our FFNN.

For our optimisation algorithm, we again opted for SGD. We tuned the learning rate, and found that 0.01 produced the best model performance on the validation set. The selection of the rest of our hyper-parameters, as well as the final model architecture is shown in table 5.4.

| Hyper-parameter | Value |
|---|---|
| Epochs | 100 |
| Batch Size | 0.01 |
| Activation Function | 1 (default) |
| Learning Rate | 1 (default) |
| Optimiser | 1 (default) |
| Dense Layers | 2 |
| Dense Units | 5, 1 |

**Table 5.4:** Chosen Hyper-Parameters for FFNN. We used a grid search with cross validation to select the number of units in the dense layers, the learning rate and batch size.

### 5.1.3 Random Forest

The hyper-parameters for our random forest were found by implementing a grid search with cross validation as well. The hyper-parameters we are concerned with for this model include:

- `n_estimators`: the number of decision trees in the forest
- `max_features`: max number of features considered for splitting a node

- `max_depth`: max number of levels in each decision tree
- `min_samples_split`: number of data points placed in a node before the node is split

By building forests with a large number of trees (high number of estimators) we can create a more robust aggregate model with less variance. This comes at the cost of a greater training time (Ampomah *et al.*, 2020). We also note that due to the randomness of Random Forests, if you have a lot of features and a small number of trees some features with high predictive power could get left out of the forest and not be used whatsoever (Ampomah *et al.*, 2020).

With respect to the maximum features considered at each node split, we have to be careful in deciding upon a value. A small value (less features considered when splitting at each node) will reduce the variance of the ensemble, at the cost of higher individual tree bias (Nti *et al.*, 2020). Increasing the maximum number of random features considered in a split tends to decrease the bias of the model, as there is a better chance that good features will be included, however this can come at the cost of increased variance. Given that we have many features we chose a combination of values for this parameter, shown in figure 5.2. For max depth of trees, if we increase this value, this would increase the possible number of feature combinations that are taken into account. The deeper the tree, the more splits it has and the more information about the data it takes into account (Ampomah *et al.*, 2020).

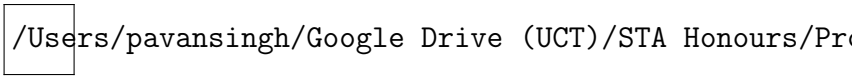



**Figure 5.2:** Grid search results using different combination of values for hyper-parameters of Random Forest. We select a max depth of 20 and minimum sample split of 2.

Figure 5.2 depicts the optimal hyper-parameters for our Random Forest model on the validation set. It can be observed, that as the maximum depth of the decision tree increases and as minimum sample leaf decreases, the algorithm tend to overfit. This results in an optimistic training-validation set performance and decreased generalisation performance on the test set. Thus, in order to obtain the best performing model, we need to determine both the optimal tree depth as well as the minimum sample leaves at the same time. After hyper-parameter tuning, we were able to obtain the parameters for our final Random Forest model. These can be seen in table 5.5.

| Hyper-parameter | Value |
|---|---|
| Max Number Estimators | 250 |
| Max Depth | 20 |
| Min Samples Split | 2 |

**Table 5.5:** Chosen model hyper-parameters for Random Forest model. We found the minimum samples split and max depth, by implementing a cross validation grid search.

### 5.1.4 AdaBoost Model

Similar to our Random Forest, we need to specify the number of estimators for our AdaBoost algorithm. That is, the maximum number of estimators (trees) at which boosting is terminated. In contrast to Random Forests, we have to specify a learning rate - how fast the tree learns.

We also have to specify the base estimator. The default value is none, which equates to a decision tree with max depth of 1 (a stump). For the hyper-parameters of our AdaBoost model we applied a grid search over a combination of values. The results of which are shown in figure 5.3.

We chose to vary the learning rate to penalise our model against overfitting. The results of using a learning rate of 0.01 and 1 are shown in the appendix C. After hyper-parameter tuning, we were able to obtain the parameters for our final model. These can be seen in the Table 5.6.

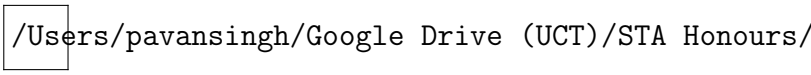



**Figure 5.3:** Grid Search Results using Different Combination of Values for Hyper-parameters of Adaboost Model. The chosen hyper-parameters, were a max depth of ten and minimum samples leaf of ten.

| Hyper-parameter | |
|---|---|
| Max Number Estimators | 250 |
| Learning Rate | 0.01 |
| Max Depth | 1 (default) |

**Table 5.6:** Chosen Hyper-Parameters for AdaBoost Model

For both ensemble decision tree models, we constructed regression trees. We predicted the daily log eturns to infer the movement of the stocks. Our binary prediction was set such that zero indicated a negative return and a one indicated a positive return.

### 5.1.5 Auto Regressive Integrated Moving Average

The modelling procedure for fitting our ARIMA model is as follows ():

- Plot the data and identify any unusual observations.
- If necessary, transform the data (using a Box-Cox transformation) to stabilise the variance.
- If the data are non-stationary, take first differences of the data until the data are stationary.
- Examine the ACF/PACF: Is an ARIMA $(p,d,0)$ or ARIMA $(0,d,q)$ model appropriate?
- Try your chosen model(s), and use the AIC to search for a better model.
- Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.
- Once the residuals look like white noise, calculate forecasts.

As such, we used AIC to determine the order of the ARIMA model (Binner et al., 2005). AIC is written as:

$$\text{AIC} = -2\log(L) + 2(p+q+k+1), \tag{5.2}$$

where $L$ is the likelihood of the data, $k = 1$ if $c \neq 0$ and $k = 0$ if $c = 0$. Good models are obtained by minimising the AIC. We looked at a combination of different p and q values. The results of our analysis are shown in the table below:

| Model Specification | AIC |
|---|---|
| (1, 0, 1) | -10549.128 |
| (1, 0, 0) | -10545.228 |
| (2, 0, 0) | -10551.196 |
| (2, 0, 2) | -10551.033 |
| (3, 0, 1) | -10548.880 |
| (2, 0, 1) | -10554.112 |

**Table 5.7:** Combination of different ARIMA model specifications used to find the optimal architecture

Of these models shown in Table 5.7, the ARIMA(2,0,1) has the smallest AIC value, hence it is our chosen model. Figure 5.3 shows the residual diagnostic plots from the ARIMA(2,0,1) model.

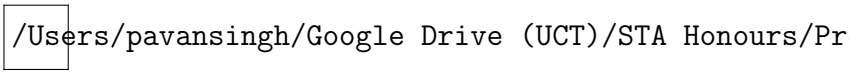



**Figure 5.4:** Model diagnostics for the ARIMA(2,0,1) model.

We see from the plot in the top right of figure 5.4, that. the residuals have a mean close to 0. Pots in figure 5.4 shows that all autocorrelations are within the threshold limits, indicating that the residuals are behaving like white noise. Any autocorrelation would imply that there is some pattern in the residual errors which are not explained in the model.

We note that the QQ-plot indicates that the distribution of the residuals deviate from normality in the tails. We apply the Box-Ljung test [3] to the residuals from the ARIMA(2,0,1) model fit to determine whether residuals are random. We received a p-value of 0.008 indicating that the residuals are random

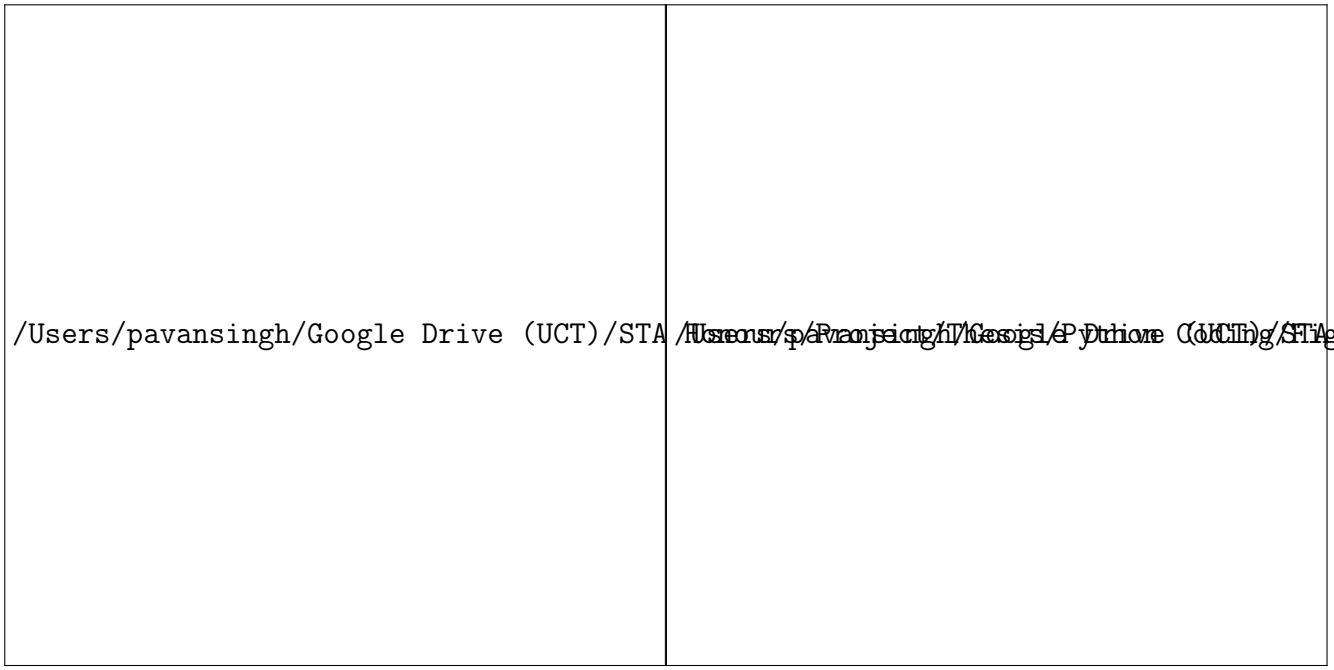
[3] Tests whether any of a group of autocorrelations of a time series are different from zero. Null hypothesis is that the data are random.

and that the model provides an adequate fit to the data. The top left plot in figure 5.4, illustrates the standardised residuals. These residual errors seem to fluctuate around a mean of zero and for the most part have a uniform variance.

## 5.2 Validation Analysis

The validation data set provides an un-biased evaluation of a model fit on the training data set while tuning the model's hyper-parameters. Using our optimised models, we illustrate the validation curves for our LSTM and FFNN models on the Ford Motors stock.

The loss (or cost) measures our model error. Specifically, the training loss indicates how well the model is fitting the training data, while the validation loss indicates how well the model fits new data.




**(a)** Multivariate LSTM **(b)** Multivariate FFNN

We note that the loss curve for our training data decreases quickly over each iteration, and flattens out after around 6 iterations. Similarly, we see that our validation curve decreases as we iterate through the data. The validation curve reaches the training loss curve at the 9th iteration. After the 12th iteration the validation curve flattens.

## 5.3 Variable Feature Importance

The feature importance captures how much the splits produced by the feature helped to optimise the model's metric used to evaluate the split quality, which in our case is the mean square error (Ampomah *et al.*, 2020).

Across both of our models we see that Moving Average Convergence Divergence (MACD) was the most significant predictive variable. Coincidentally, this happens to be one of the most popular technical indicators used in practice. The MACD is a trend following momentum indicator which shows the relationship between two different moving averages on our stocks price. It essentially serves as a signal to indicate when a price has risen/dropped by too much and will correct back to its 26- day moving average. In addition, another strongly predictive feature was the 21-day Exponential Moving Average


**Figure 5.6:** Variable importance plots from our ensemble decision tree models.

(EMA). Essentially, an EMA is just a moving average of a stock's returns where the last few days are given larger weights (Ampomah *et al.*, 2020).

# Chapter 6

# Results

## 6.1 Results

| Model | Accuracy (%) | | | F1 Score | | | AUC | | |
|---|---|---|---|---|---|---|---|---|---|
| | Ford | Google | Motorola | Ford | Google | Motorola | Ford | Google | Motorola |
| Uni LSTM | 50.04 | 49.60 | 53.60 | 0.49 | 0.50 | 0.52 | 0.52 | 0.48 | 0.50 |
| Uni FFNN | 47.20 | 44.80 | 49.60 | 0.48 | 0.46 | 0.50 | 0.47 | 0.49 | 0.47 |
| Random Forest | 54.40 | 55.20 | 0.52 | 0.52 | 0.54 | 0.48 | 0.60 | 0.59 | 0.56 |
| AdaBoost | 52.80 | 52.00 | 0.51 | 0.48 | 0.51 | 0.49 | 0.62 | 0.55 | 0.53 |
| Multi LSTM | 58.40 | 55.20 | 59.20 | 0.60 | 0.53 | 0.63 | 0.65 | 0.58 | 0.56 |
| Multi FFNN | 55.20 | 53.60 | 52.80 | 0.57 | 0.54 | 0.54 | 0.57 | 0.65 | 0.56 |
| ARIMA | 45.60 | 39.20 | 42.40 | 0.44 | 0.38 | 0.38 | | | |

**Table 6.1:** Results from comparative analysis of various machine learning models on three stocks, using ARIMA as benchmark. The accuracy (in percentage), F1 score and AUC are presented for all the models.

In this paper, we assessed how accurately we can predict the one day ahead stock movement of three companies. The proposed models were trained on 95% of the entire dataset (approx. 2115 days) and tested on the remaining 125 days. The results of our comparative analysis are shown in Table 6.1. The predicted movements for the Ford Stocks by our final models are summarised in the confusion matrices in figure 6.1. The confusion matrices for Google and Motorola can be found in appendix E. The confusion matrices from our ARIMA model are shown in figure. 6.5.

Given that stock prediction is a challenging task and a naive random forecast is 50%, the accuracy achieved by our LSTM models (all above 55%) is generally reported as a satisfying result for binary stock movement prediction (Nguyen and Shirai, 2015).

Table 6.1 indicates that the LSTM model with all the technical indicators and stock data (multivariate LSTM) performed the best out of all the models, achieving a max accuracy of 59.2% for predicting the movement of Motorola stock. The model also outperformed every other in accuracy metric for the other two stocks. It is worth noting that the Random Forest model achieved the same accuracy as the multivariate LSTM for the Google stock. The Random Forest, in fact, achieved similar accuracy results to that of the feed-forward neural network. The AdaBoost model, performed the worst amongst the multivariate neural networks and ensemble methods. However, it still managed to achieve greater than 50% accuracy across all stocks.

The univariate models for the LSTM and FFNN achieved much lower accuracies and F1 scores, than the multivariate models. The univariate FFNN was outperformed by a naive random forecast, indicating the poor predictive performance of the model. This is further supported by the poor AUC scores - below 0.5. Although the univariate LSTM model achieved an accuracy of 54.6% for the Motorola stock, the model did not perform well against the other stocks - achieving 50.04 and 49.60%. This indicates that the models were not able to provide anything useful. This also suggests that we cannot achieve any valuable predictive performance in forecasting the one day ahead movement, using previous prices alone as inputs.

**(a)** Random Forest **(b)** AdaBoost **(c)** Univariate Feed-Forward

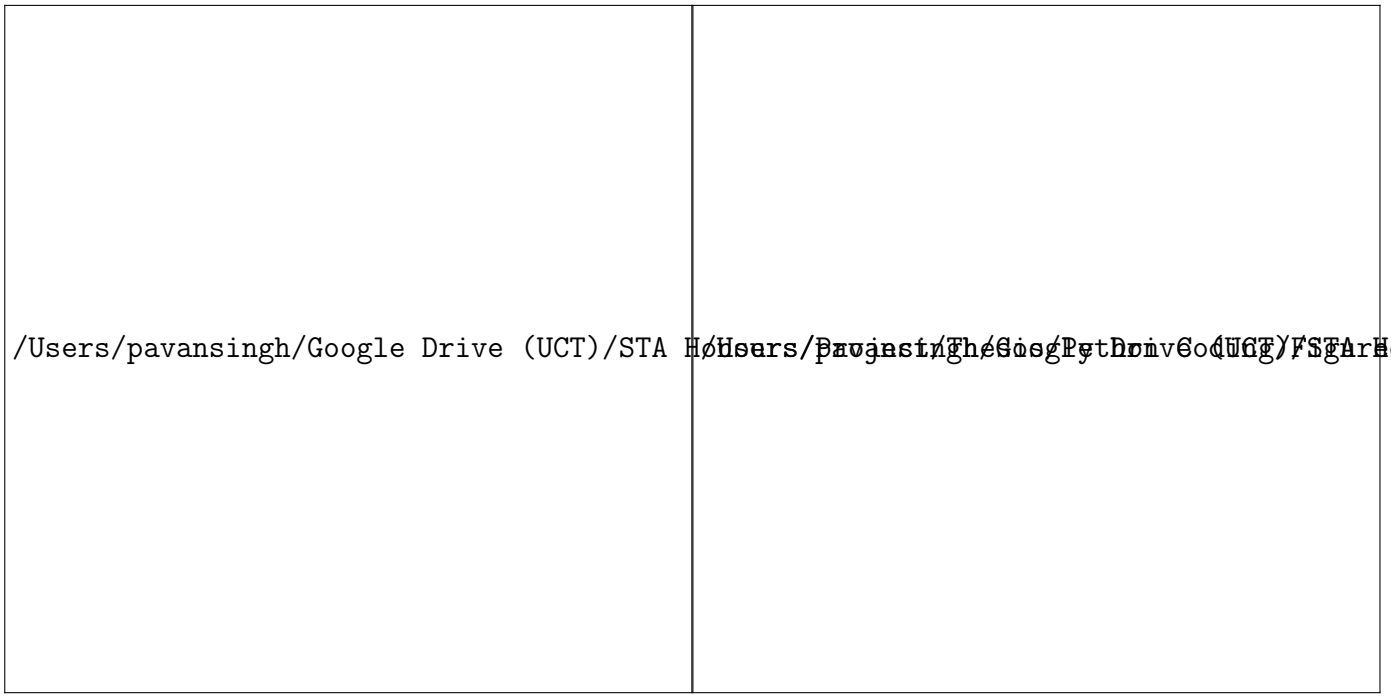
**(d)** Multivariate Feed-Forward **(e)** Univariate LSTM **(f)** Multivariate LSTM

**Figure 6.1:** Confusion matrixes for the various machine learning models for Ford Motors only. Left axes depicts the true movement of the daily log returns, while the bottom axes depicts the predicted movement

Given that our models were trying to predict the movement of stocks, a trader would be interested in the F1 score metric, given that it takes into account both false positives and false negatives. The multivariate LSTM model had an F1 score of 0.60 for Ford, 0.53 for Google and 0.63 for Motorola. The F1 score of 0.63 for Motorola and 0.60 for Ford stocks, were in fact the highest values achieved amongst the other models. Notably, the random forest and multivariate FFNN achieved the highest F1 score of 0.54 for Google stock.

Based on the computed probabilities from our predictions of the machine learning models, a ROC plot was generated and is shown in Figures 6.2 and 6.3. ROC-AUC score provides us with information about how well a model is performing its job of separating cases. AUC measures the quality of the model's predictions. Using AUC score as the metric, we now find that the multivariate FFNN and multivariate LSTM had the best classification performance - achieving an AUC score of 0.65 for Google and Ford stocks respectively. As such, our multivariate LSTM model which achieved a AUC score for Ford stock of 0.65 indicates that there is a 65% that our model can distinguish up movements from down movements of the Ford stock. We also note our Random Forest achieved relatively high AUC scores, indicating the impressive classification performance of the model.

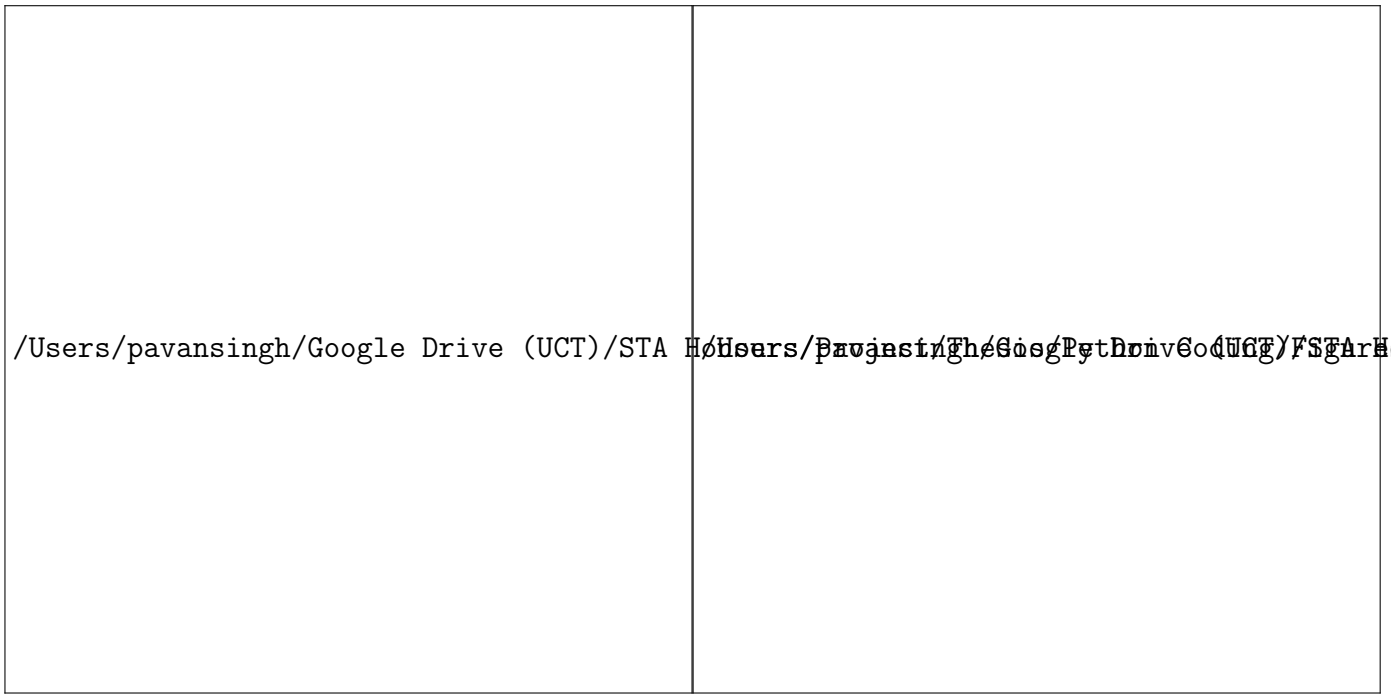
Comparing our machine learning models to that of the benchmark model - ARIMA - table 6.1 indicates that all of our machine learning models achieved accuracy greater than that of the ARIMA model for all three stocks. The benchmark ARIMA model, performed the worst, achieving a max accuracy of 45.60% for predicting the movement of the log returns for Ford stocks. The ARIMA performed substantially worse that the other multivariate models, for predicting the movement of Google stock - achieving 39.20%. This result is likely due to under-fitting of the data as the stock returns are complex and are not linear.




**(a)** Random Forest **(b)** AdaBoost

**Figure 6.2:** ROC Curves for the ensemble decision tree machine learning models for all three stocks. Left axes is the sensitivity, while the bottom axes is the specificity.

A noticeable issue shared between all our models, is the poor performance in forecasting the magnitude of the returns. Although our models may have done well to predict the movement of the stocks, we can see that in Figure 6.4 our machine learning models, on average, had very small forecasted returns which were often only a fraction of the size of the realised return.

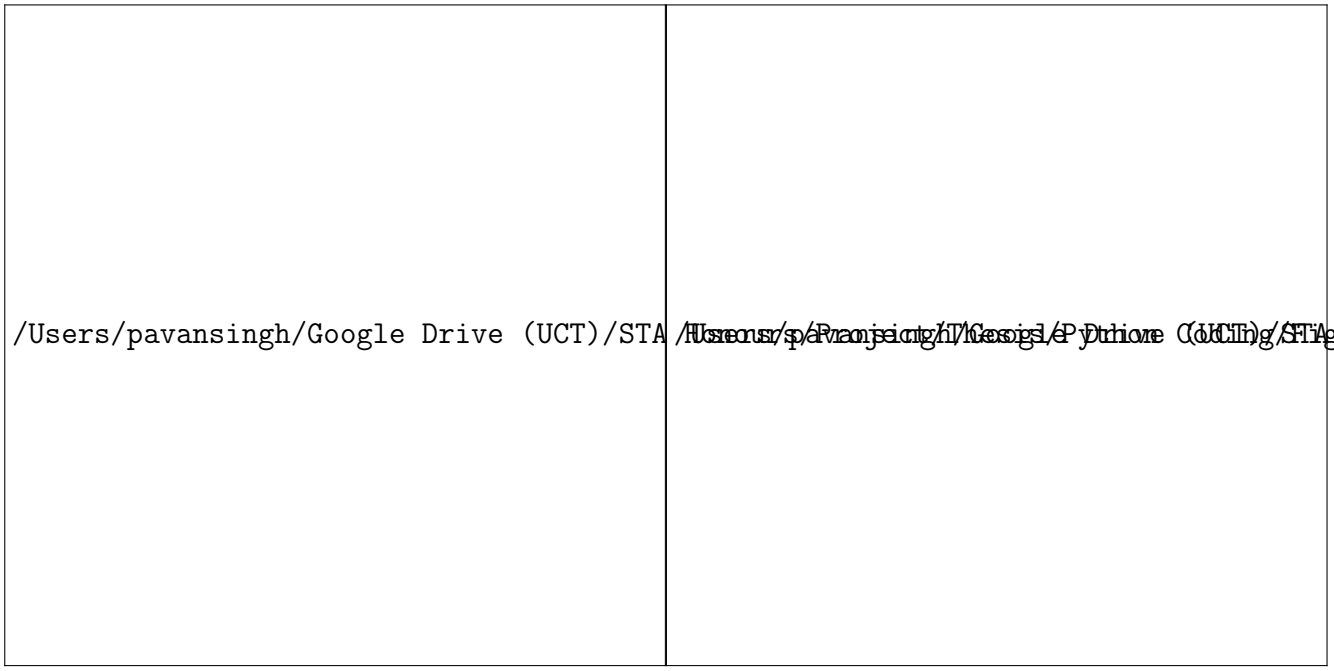


**(a)** Multivariate Feed-Forward **(b)** Multivariate LSTM

**Figure 6.3:** ROC Curves for neural network machine learning models for all three stocks. Left axes is the sensitivity, while the bottom axes is the specificity.

**(a)** Ford Motors **(b)** Google **(c)** Motorola

**Figure 6.4:** Predicted daily log returns vs the realised log returns for the various machine learning models for Ford Motors only. Left axes depicts the log return while the bottom axes depicts the day



**(a)** Multivariate LSTM **(b)** Multivariate FFNN

**(c)** Random Forest **(d)** AdaBoost

**Figure 6.5:** Predicted daily log returns vs the realised log returns for the various machine learning models for Ford Motors only. Left axes depicts the log return while the bottom axes depicts the day

# Chapter 7

# Discussion

## 7.1 Link To Literature

Given our results in chapter 6 from our comparative analysis of four machine learning models, we can reflect on how our empirical results compare with the literature. Our results concur with the findings of Graves (2012), who found that long short-term memory neural networks (LSTM) performed better than than other machine learning models in the context of sequential data. Moreover, Chen *et al.* (2015) were able to achieve an LSTM model with 55.9% accuracy in predicting whether the price of stock would go up or down. We achieved similar results with our LSTM model, achieving a max accuracy of 59.2%.

Comparing our machine learning models to the ARIMA model results, we acknowledged that the ARIMA produced unsatisfactory results - achieving an accuracy below 50% for all three stocks tested. In contrast all our multivariate machine learning models achieved accuracy over 50%. This is in agreeance with findings from Binner *et al* (2005), who also found that the neural network models they tuned achieved substantially higher accuracy than the traditional time series models; namely an ARIMA and vector autoregressive models.

## 7.2 Limitations and Further Work

Firstly, we have only examined these three stocks over a certain time-period. Thus, it is not clear if our results or conclusions can extend prior or past this period. More generally, it would be worth examining if our conclusions would differ if a different type of stock is chosen or if a stock index used.

In our model optimisation of hyper-hyper-parameters, we limited ourselves to using only grid search and manual searching methods to find the best (optimal) hyper-parameters. A grid search can be classed as relatively inefficient because they do not choose the next hyper parameters to evaluate based on previous results. That is, a grid search is uninformed by past evaluations, and as a result, can lead to spending large amounts of time evaluating results. Therefore, to find an optimal solution for complex models, a common approach used is sequential model-based optimisation (SMBO). This creates a probability model of the objective function to find the most promising hyper- parameters to test on the true objective function. Moreover, this optimisation is classed as more efficient as it takes into account the previous outcomes, so it performs an informed search (Che *et al.*, 2017).

Our machine learning models were benchmarked using an ARIMA model. Univariate time series models like the ARIMA, are limited to statistical relationships between a target variable and it's lagged values or lagged disturbances (Ampomah *et al.*, 2020). We did not include any exogenous variables to our ARIMA model. A better benchmark would be to model an ARIMAX time series model, including the same variables used for our machine learning model inputs. Moreover, common time series models in financial domain, include the Autoregressive Conditional Heteroscedasticty (ARCH) and Generalised Autoregressive Conditional Heteroscedasticty (GARCH) models. These models would be worthwhile to include in a comparative analysis. GARCH models are well suited for modelling financial data, since there have been developed to handle and forecast the volatility of financial assets.

Another improvement would be including some dimensionality reduction. As the number of factors increases, so does the complexity involved in modelling stock returns behaviour. Given that computing

resources are finite, coupled with time constraints, performing an extra computation for a new factor only increases the bottleneck on returns modelling calculations. A linear technique for dimensionality reduction is Principal Component Analysis (PCA) (Che *et al.*, 2017. As its name suggests, PCA breaks down the movement of portfolio asset prices into its principal components, or common factors, for further statistical analysis. Common factors that don't explain much of the movement of the stock receive less weighting in their factors and are usually ignored. By keeping the most useful factors, stock return analysis can be greatly simplified without compromising on computational time and space costs (Che *et al.*, 2017. Moreover, in line with feature selection, one could look at using domain knowledge of the data, and infer features to be creased to help their learning algorithms increase their predictive performance. This can be as simple as grouping or bucketing related parts of the existing data to form defining features. Even removing unwanted features is also feature engineering.

As mentioned in chapter 2, strategies in investment are broadly sorted into two categories, fundamental and technical analysis. This project implemented a technical analysis, as such our models disregarded any information pertaining to fundamental analysis of a company's stock value. For further analysis, one could improve the breadth of the dataset beyond technical indicators to also include other significant financial factors. Stock data not only depends on the trend in the historical data, it also mainly depends on the product value and the information available from news channels. Such sentiment analysis has been experimented with already with financial data. Srivastava (2012) used a twitter sentiment analysis to analyse stock market movements. In his paper he found that there was a high correlation between stock returns and twitter sentiments. Moreover the financial ratios associated With fundamental analysis, such as P/E ration, book value per share and dividend yield should be considered. So, the future implementation includes the fundamental analysis, along with sentiment of the customers reviewed on the products related to a company or its domain and add this analysis to the prediction using LSTM.

The inclusion of more models is the obvious route for further study. In this paper we used a handful of non-linear machine learning models, of which there exist dozens, including hybrid models and other ensemble neural networks which have become increasingly popular in the financial setting.

Ensembles of machine learning models have proven to improve the performance of prediction tasks in various domains. However, the additional computational costs for the performance increase are usually high since multiple models must be trained. An example application, would be to investigate eh impact of using a stacked LSTM. Stacking Stacking (STK). is an ensemble learning technique that makes use of predictions from several models $(m_1, m_2, \ldots, m_n)$ to construct a new model, where the new model is employed for making predictions on the test dataset. STK seeks to increase the predictive power of a classifier (Ampomah *et al.*, 2020). Literature suggests there have been noticeable improvements. in predictive performance, using stacking. As such it would be of interest to include other ensemble learning techniques into a comparative study of this nature.

We only assessed the models on three time series. In order to adequately assess market efficiency, it would be necessary to conduct larger analysis, including much moire time series. The M3 competition is the latest in a sequel of M forecasting competitions, organised by Makridakis and Hibon (2000). It consists of 3003 business-type time series, covering a range of domains. Moreover, the M3 data has become an important benchmark for testing and comparing forecasting models. Having that many diverse time series gives confidence into comparison results. A suitable further improvement would be to implement and remodel the chosen machine learning models in this paper on the M3 data. The results of which are more comparable to other findings in the literature.

## 7.3 Conclusion

The stock market prices can be influenced by many factors such as political situations, economic events, and natural disasters etc. As a result, forecasting stock market movements is a challenging task due to the complexity of the stock market. Our paper served as a comparative analysis of a feed-forward neural network, a long-short-term memory neural network, a Random Forest and a AdaBoost model. The models were assessed on their ability to predict the movement of the log returns of Google, Motorola and Ford Motors stock. Our results indicate that the LSTM model was the best machine learning model to forecast the one day ahead movement of stock data, using accuracy, F1 score and AUC classification metrics. The random forest and feed-forward neural networks produced similar results across all three classification metrics. The machine learning models were benchmarked against an ARIMA model. The results indicated that the ARIMA model was unable to provide any predictive performance in forecasting the movement of the stocks. All four machine learning models outperformed the ARIMA model.

At the highest level, our most complex model - the LSTM - achieved an accuracy of 59.2% for predicting the movement of Motorola stock.

# Appendix A

## A.1 Random Walk Hypothesis

The random walk hypothesis states that the single-period log returns, $r_t = \log(1+R_t)$, are independent.

Because

$$\begin{aligned} 1+R_t(k) &= (1+R_t)\cdots(1+R_{t-k+1}) \\ &= \exp(r_t)\cdots\exp(r_{t-k+1}) \\ &= \exp(r_t+\cdots+r_{t-k+1}) \end{aligned}$$

we have

$$\log\{1+R_t(k)\} = r_t+\cdots+r_{t-k+1}$$

It is sometimes assumed further that the log returns are $N(\mu,\sigma^2)$ for some constant mean and variance. Since sums of normal random variables are themselves normal, normality of single-period log returns implies normality of multiple-period log returns. Under these assumptions, $\log\{1+R_t(k)\}$ is $N(k\mu,k\sigma^2)$.

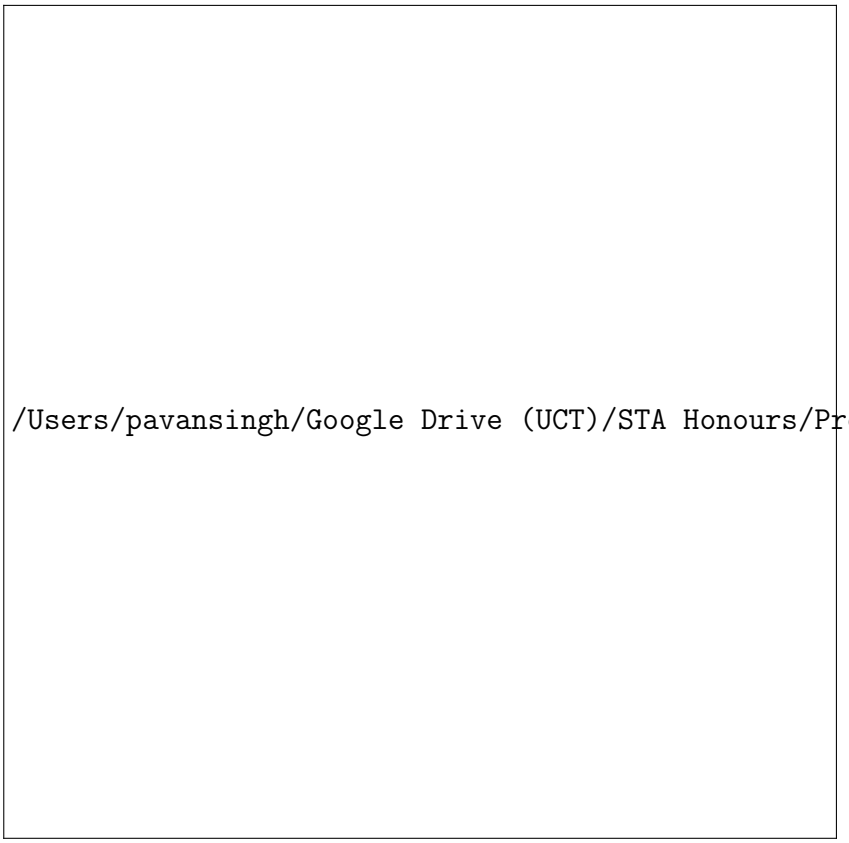
# Appendix B

## B.1 Closing Prices for Stocks

## B.2 Log Returns of Stocks

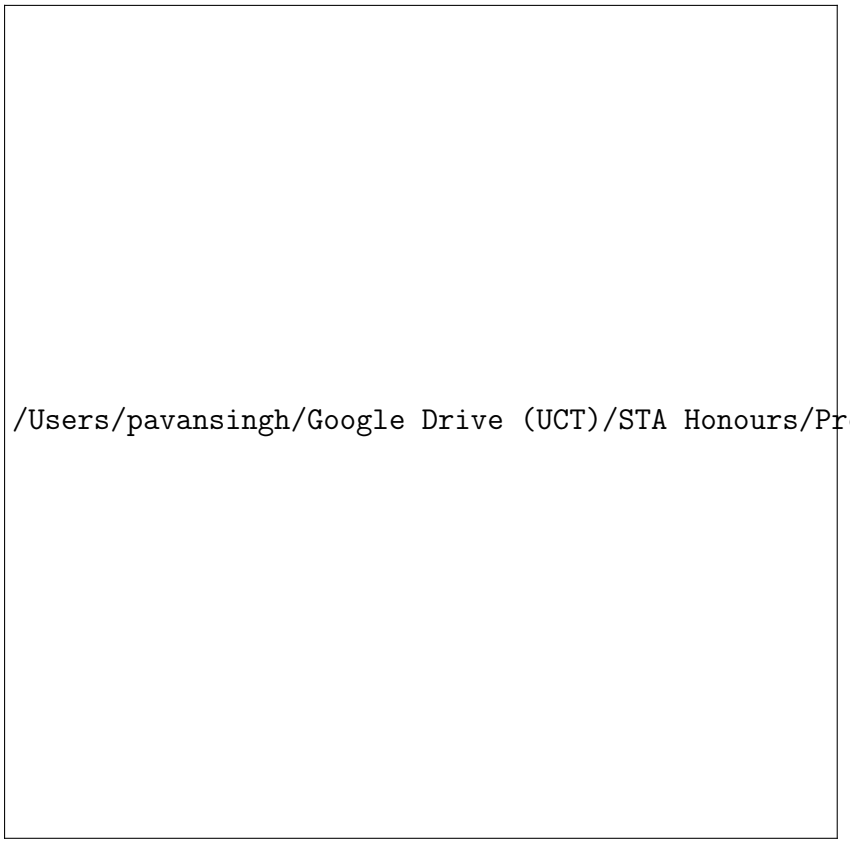
## B.3 QQ-Plots for Log Returns of Stocks

## B.4 ACF and PACF of Stocks

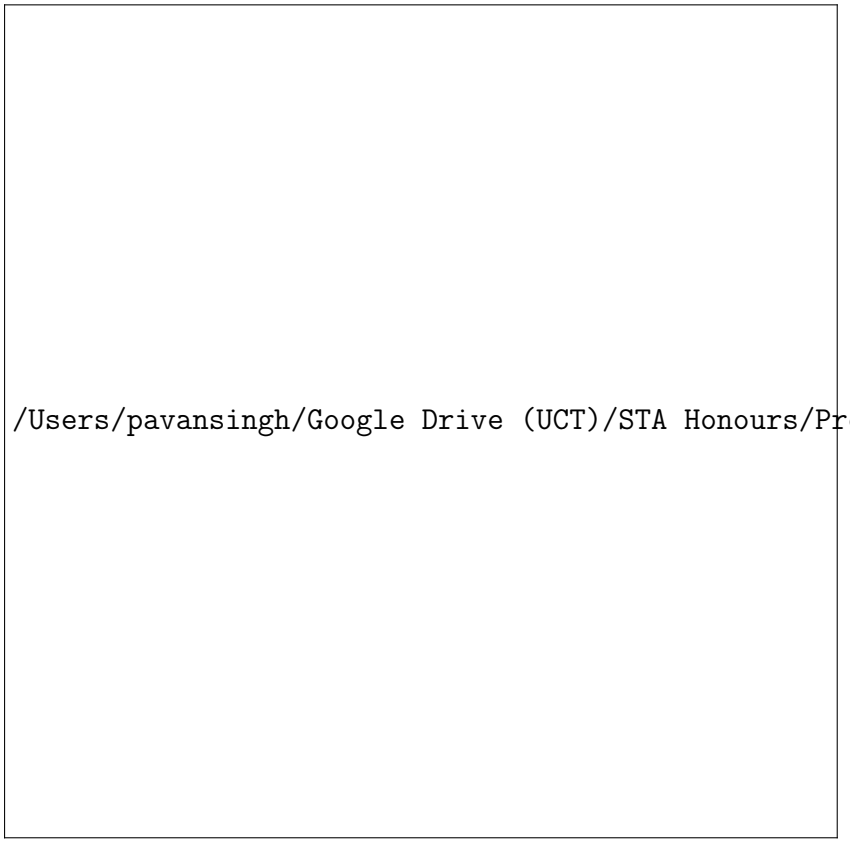



**(a)** Motorola

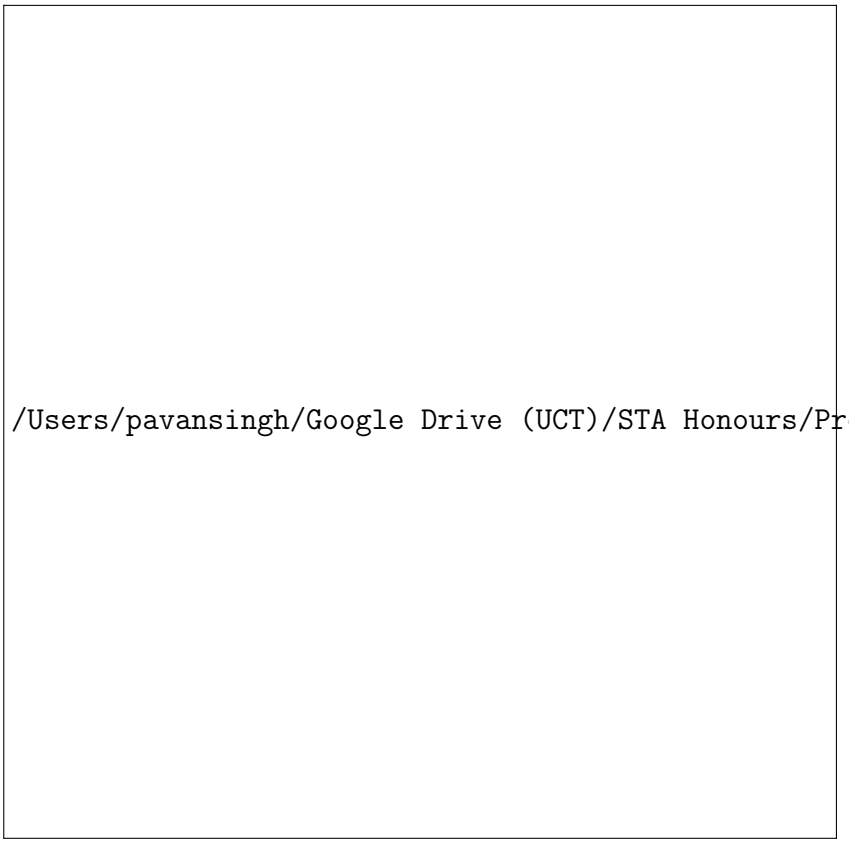

**(b)** Google




**(a)** Motorola

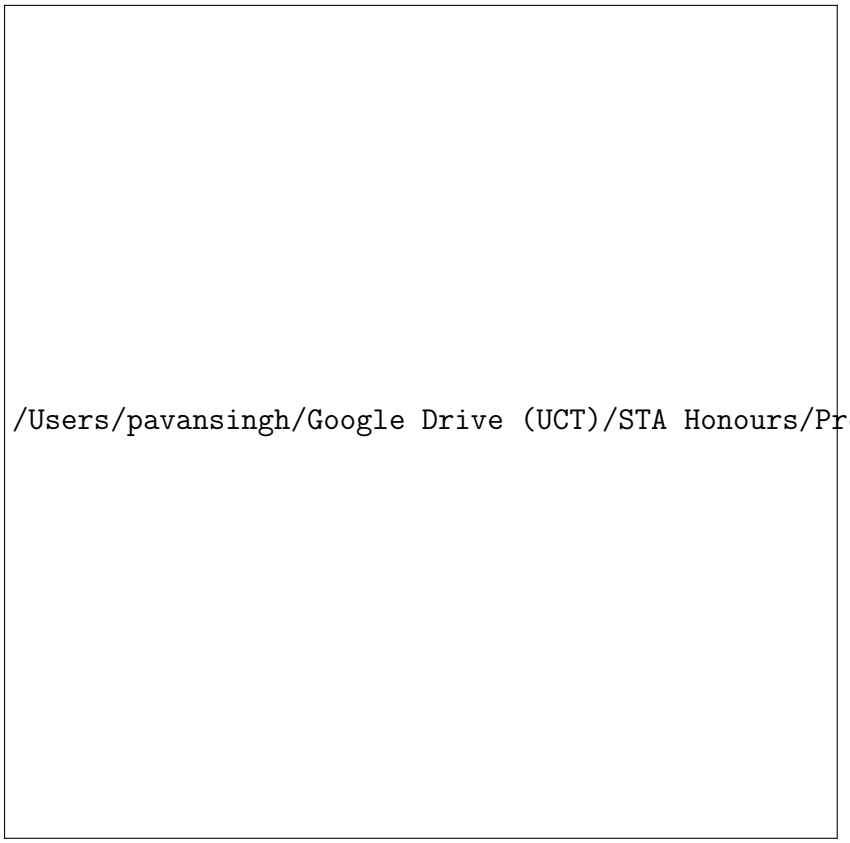



**(b)** Google

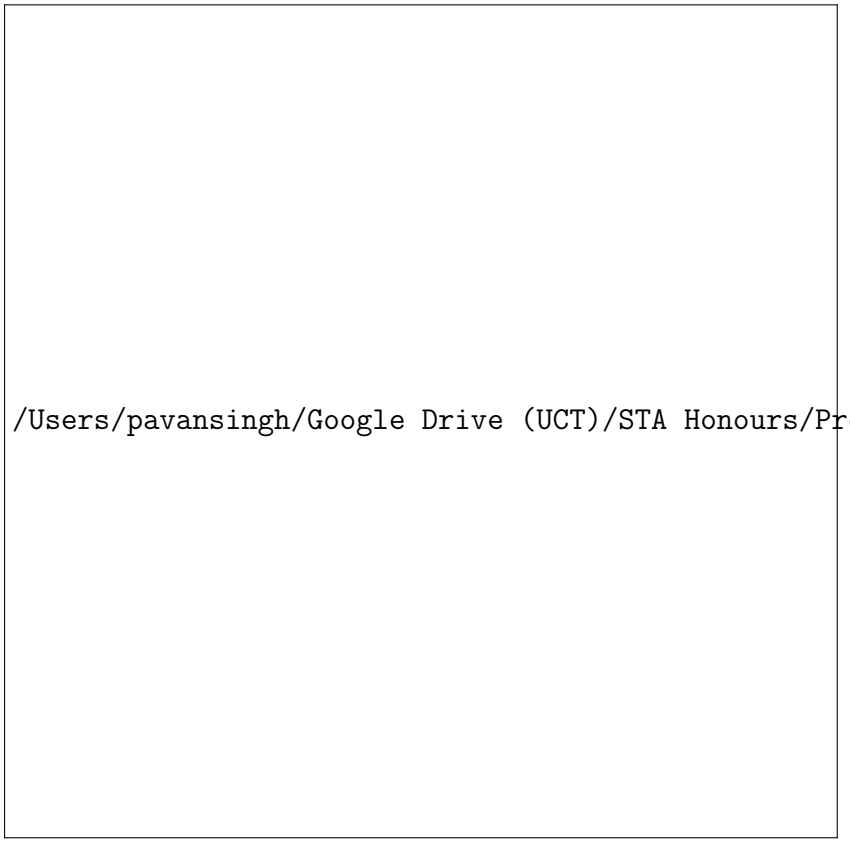



**(a)** Motorola

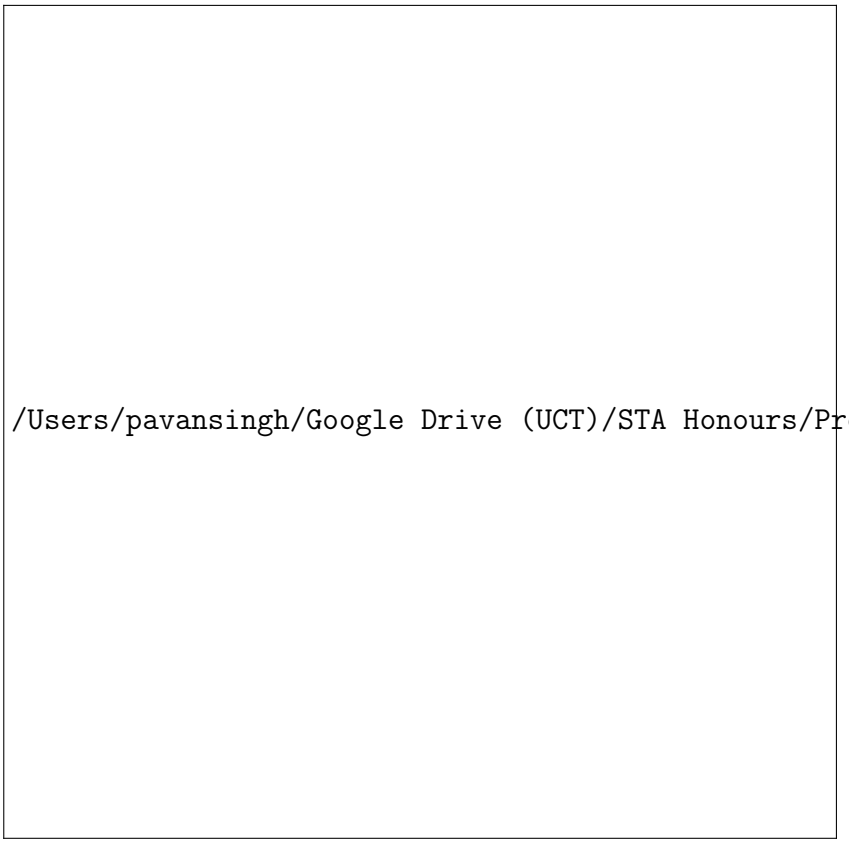

**(b)** Google




**(a)** Ford




**(b)** Google




**(a)** Ford


**(b)** Google

## B.5 Random Forest Algorithm

Random Forest for Regression or Classification (Brillinger, 2001).

1. For $b = 1$ to $B$
2. Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
3. Grow a random-forest tree $T_b$ to the bootstrapped data, by re- cursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{\min}$ is reached.
4. Select $m$ variables at random from the $p$ variables.
5. Pick the best variable/split-point among the $m$.
6. Split the node into two daughter nodes.
7. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point $x$ :

- Regression: $\hat{f}_{\mathrm{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

- Classification: Let $\hat{C}_b(x)$ be the class prediction of the $b$ th random tree. Then $\hat{C}_{\mathrm{rf}}^B(x) =$ majority vote $\left\{\hat{C}_b(x)\right\}_1^B$.

## B.6 Technical Indicators Used in Analysis

The following technical indicators were used in. our study. The definitions were scraped from Lo and Hasanhodzic (2010).

**Standard Deviation** (SD): measures market volatility and is used in statistics to describe the variability or dispersion of a set of data around the average.

**Relative Strength Index** (RSI): is a momentum indicator used in technical analysis that measures the magnitude of recent price changes to evaluate overbought or oversold conditions in the price of a stock or other asset.

**Average Directional Index** (ADX): is a technical analysis indicator used by some traders to determine the strength of a trend.

**Moving Average**(MA): is a stock indicator that is commonly used in technical analysis. The reason for calculating the moving average of a stock is to help smooth out the price data over a specified period of time by creating a constantly updated average price.

**Exponential Moving Average** (EMA): is a technical chart indicator that tracks the price of an investment (like a stock or commodity) over time.

**Momentum** (MTM): is a technical indicator which shows the trend direction and measures the pace of the price fluctuation by comparing current and past values.

**Average True Range** (ATR): is a market volatility indicator used in technical analysis. It is typically derived from the 14-day simple moving average of a series of true range indicators.

**Bollinger Bands** (BB): are envelopes plotted at a standard deviation level above and below a simple moving average of the price.

**Stochastic Oscillator** (SO): is a momentum indicator comparing a particular closing price of a security to a range of its prices over a certain period of time.

**Triple Exponential Average** (TRIX): is an oscillator used to identify oversold and overbought markets, and it can also be used as a momentum indicator.

**Moving Average Convergence Divergence** (MACD): It is designed to reveal changes in the strength, direction, momentum, and duration of a trend in a stock's price.

**Mass Index** (MI): used in technical analysis to predict trend reversals.

**Vortex Indicator** (VI): is used to spot trend reversals and confirm current trends.

**True Strength Index** (TSI): is a technical momentum oscillator used to identify trends and reversals.

**Chaikan Oscillator** (CO): is the difference between the 3-day and 10-day EMAs of the Accumulation Distribution Line.

**Force Index** (FI): measures the amount of power used to move the price of an asset.

**Ease of Movement** (EOM): is a technical study that attempts to quantify a mix of momentum and volume information into one value.

**Commodity Channel Index** (CCI): is a technical indicator that measures the difference between the current price and the historical average price.

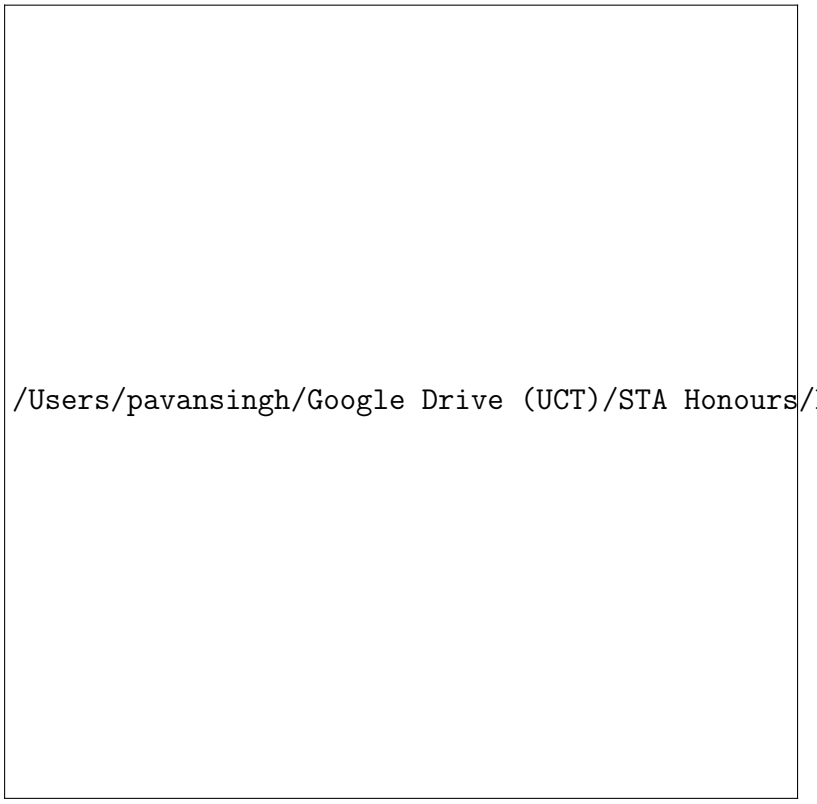
**Keltner Channel** (KC): technical indicator that day traders can use to help assess the current trend and provide trading signals.

**Ultimate Oscillator** (UO): is a range-bound indicator with a value that fluctuates between 0 and 100. Used to identify oversold or over bought assets.

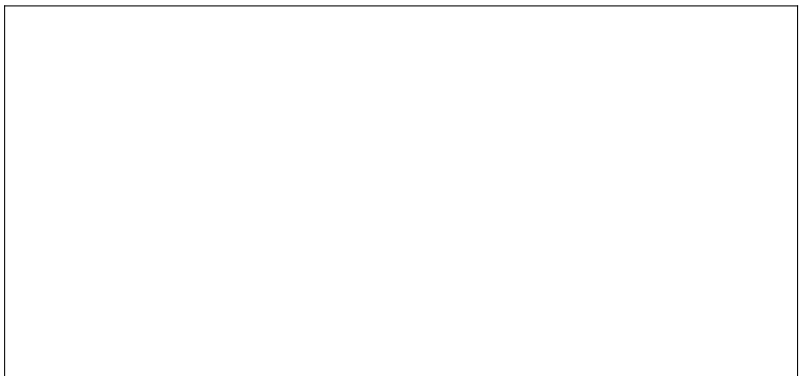
## Appendix C

# Boosting Grid Search Learning Rates


**(a)** Learning Rate: 0.01



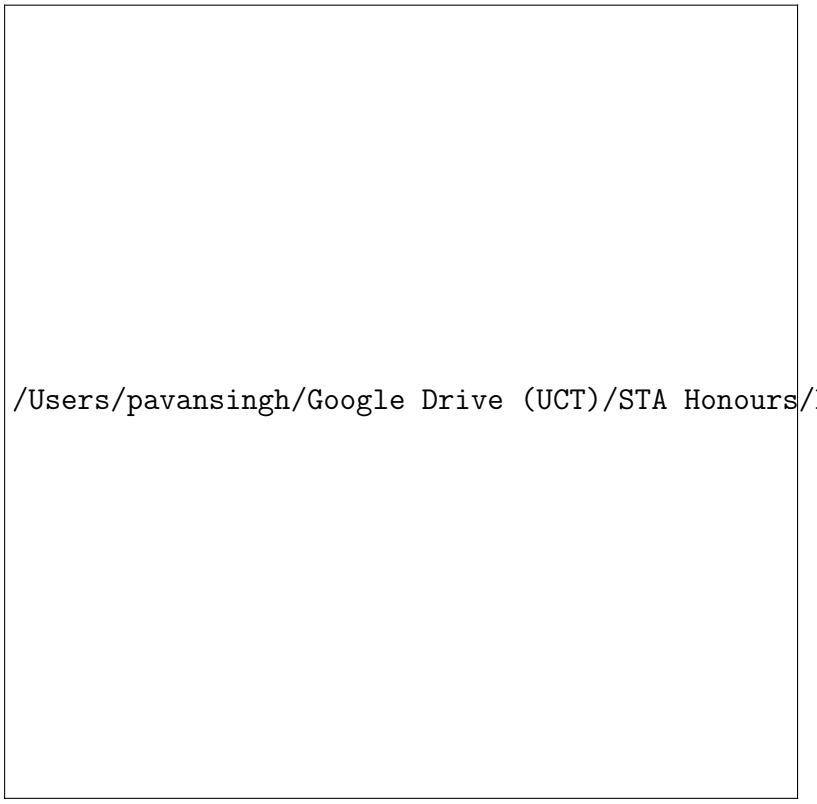

**(b)** Learning Rate: 0.1



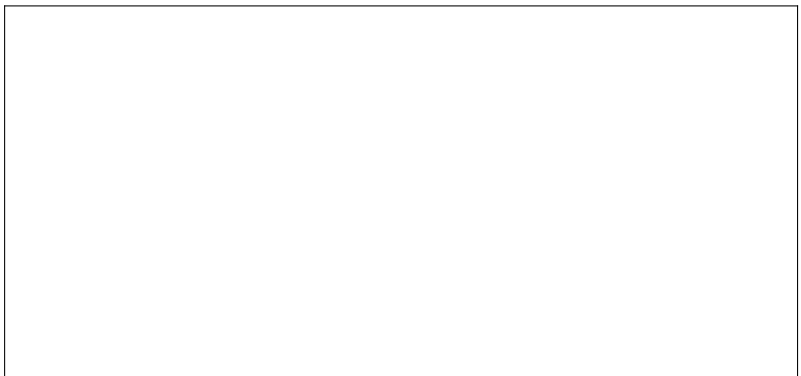
## Appendix D

# Validation and Loss Curves for LSTM


**(a)** Multivariate LSTM Ford




**(b)** Multivariate LSTM Ford



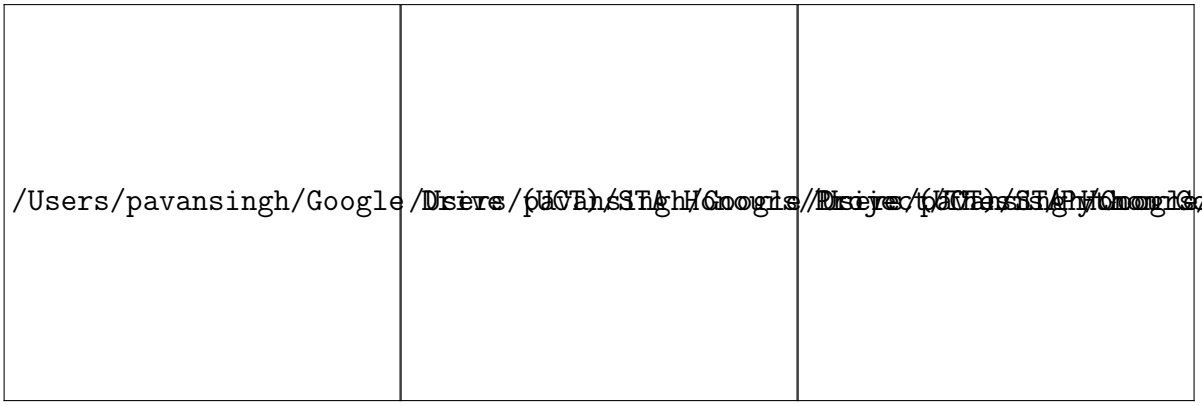
## Appendix E

# Confusion Matrices

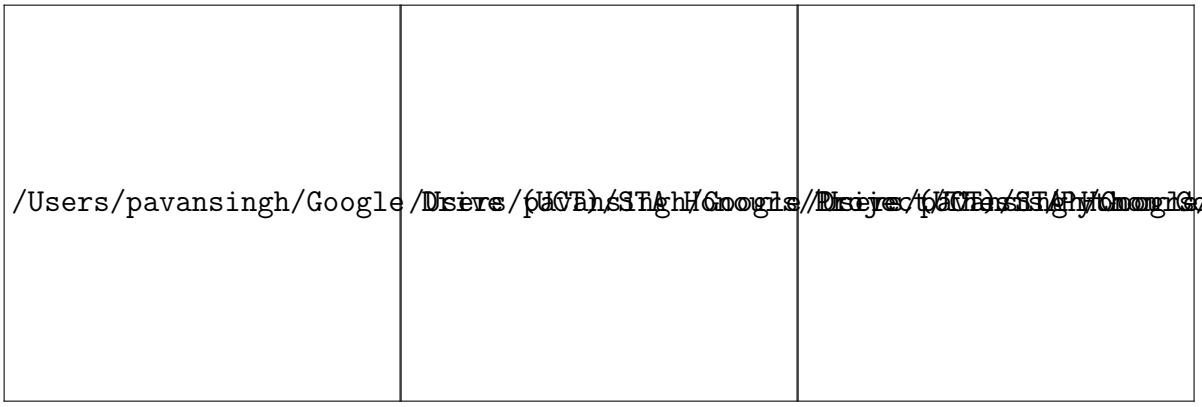
**(a)** Random Forest **(b)** AdaBoost **(c)** Univariate Feed-Forward

**(d)** Multivariate Feed-Forward **(e)** Univariate LSTM **(f)** Multivariate LSTM

**Figure E.1:** Confusion matrixes for the various machine learning models for Google only. Left axes depicts the true movement of the daily log returns, while the bottom axes depicts the predicted movement



**(a)** Random Forest **(b)** AdaBoost **(c)** Univariate Feed-Forward



**(d)** Multivariate Feed-Forward **(e)** Univariate LSTM **(f)** Multivariate LSTM

**Figure E.2:** Confusion matrixes for the various machine learning models for Motorola only. Left axes depicts the true movement of the daily log returns, while the bottom axes depicts the predicted movement