4-19-2017 12:00 AM

Improving Long Term Stock Market Prediction with Text Analysis

Tanner A. Bohn, *The University of Western Ontario*

Supervisor: Dr. Charles Ling, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

The task of forecasting stock performance is well studied with clear monetary motivations for those wishing to invest. A large amount of research in the area of stock performance prediction has already been done, and multiple existing results have shown that data derived from textual sources related to the stock market can be successfully used towards forecasting. These existing approaches have mostly focused on short term forecasting, used relatively simple sentiment analysis techniques, or had little data available. In this thesis, we prepare over ten years worth of stock data and propose a solution which combines features from textual yearly and quarterly filings with fundamental factors for long term stock performance forecasting. Additionally, we develop a method of text feature extraction and apply feature selection aided by a novel evaluation function. We work with investment company Highstreet Inc. and create a set of models with our technique allowing us to compare the performance to their own models. Our results show that feature selection is able to greatly improve the validation and test performance when compared to baseline models. We also show that for 2015, our method produces models which perform comparably to Highstreet's hand-made models while requiring no expert knowledge beyond data preparation, making the model an attractive aid for constructing investment portfolios. Highstreet has decided to continue to work with us on this research, and our machine learning models can potentially be used in actual portfolio selection in the near future.

Keywords: financial modelling, efficient market hypothesis, machine learning

Acknowlegements

I would like to thank my supervisor, Dr. Charles Ling, for supporting me during my research. The calibrated patience, freedom, and guidance he has provided me with were critical to my positive experience at Western and the completion of this thesis.

I am also grateful to Jun Du, Robert Yan, Yemin Li, and Chang Liu, who worked with me on the project behind this thesis. Jun, Yemin, and Chang were integral in preparing the dataset used in this thesis as well as discussing, developing, and testing new ideas. Robert's expert advice related to finance also ensured that our methods did not depart from reality.

Finally, I would like to thank my parents and brothers for their constant support throughout this time. Their contact and my trips home have helped me retain some level of sanity.

Contents

Abstract **ii**

Acknowlegements **iii**

List of Figures **vii**

List of Tables **x**

1 Introduction **1**

1.1 Purpose 1

1.2 Contributions 2

1.3 Thesis Outline 3

2 Background **4**

2.1 Stock Prediction Basics 4

2.1.1 Important Terms 4

2.1.2 Types of Financial Analysis 5

2.2 Sentiment Analysis 6

2.3 Autoencoders 7

3 Related Research **10**

3.1 Performance Prediction using Classical Approaches 10

3.1.1 Technical Analysis 11

3.1.2 Fundamental Analysis 12

3.2 Performance Prediction using Machine Learning 15

3.3 Performance Prediction with Text Analysis 17

4 Challenges **21**

4.1 Market Efficiency . . . 21

4.2 Data Challenges . . . 23

4.2.1 Filing Data . . . 24

4.2.2 Fundamental Factors and Stock Prices . . . 24

5 Data Preparation and Feature Set **27**

5.1 Fundamentals . . . 27

5.2 GICS . . . 29

5.3 Company Filings . . . 31

5.3.1 Sentiment Dictionary . . . 31

5.3.2 Autoencoders . . . 33

5.4 Relative Returns . . . 37

6 Measuring Model Performance **38**

6.1 Train/Validation/Test Splitting . . . 38

6.1.1 Overfitting . . . 39

6.1.2 Expanding Windows . . . 39

6.2 Metrics . . . 41

6.2.1 Training Performance . . . 43

6.2.2 Validation Performance . . . 43

6.2.3 Combining Validation Performances . . . 44

6.2.4 Test Performance . . . 46

7 Baseline Models **47**

7.1 Overview of Models and Training . . . 47

7.2 Results . . . 48

8 Applying Feature Selection **52**

8.1 Feature Selection Method and Metric . . . 52

8.2 Feature Selection Results . . . 55

8.2.1 Accessing All Features . . . 55

8.2.2 Restricting Feature Types 55
8.3 Text Feature Extraction 60
8.3.1 Autoencoder Setup 60
8.3.2 Determining AE Size, Regularisation, and Auxiliary Loss Weight . . . 61
8.3.3 Results 65
8.4 Test Results 65
8.5 Sector-Level Models 67

9 Conclusions and Future Work **72**
9.1 Conclusions 72
9.2 Future Works 73

Bibliography **76**

Curriculum Vitae **80**

List of Figures

2.1 Basic autoencoder structure 8

3.1 Simplified depictions of four common reversal patterns, oriented to show an initial upward trend, reversal pattern, and downward trend. 13

4.1 Curves with increasing estimated Hurst exponent. 22

4.2 The Intel stock price over 5 years (top) and random walk of length 250 (bottom). Both have roughly the same number of samples shown. 23

4.3 A visualisation of the set of stocks in the S&P 500 between 2001 and 2015. The x coordinate of a pixel determines what stock it is, and the y value is the date (2001 at top to 2015 at bottom). If a pixel is light, it means no data was available, if it is dark, data was available. 26

5.1 Example of encoding a GICS by concatenating one-hot encodings for each level of the GICS. 30

5.2 Autoencoder with a main output and auxiliary output. 36

6.1 Sliding windows 41

6.2 Expanding windows 42

6.3 Example model performance curves in five windows demonstrating the effects of having a high and low mean and a high and low standard deviation across windows. The S_ϵ metric encourages models with high mean and low standard deviation. 45

7.1 Validation scores for LSLR 50

7.2 IC for each validation week for LSLR 51

8.1 Calibrating epsilon 54
8.2 Validation IC and $S_{0.15}$ score as features are added 56
8.3 Validation IC and $S_{0.15}$ score as features are added. Adding features in order of decreasing $S_{0.15}$ score when measured by themselves. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. 57
8.4 Validation IC and $S_{0.15}$ score as features are added. Adding features in order of decreasing IC when measured by themselves. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. 57
8.5 Validation IC and $S_{0.15}$ score as features are added when only accessing fundamentals, GICS, and filing features. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. 59
8.6 Validation IC and $S_{0.15}$ score as features produced by small AE are selected. Values are the average of two runs with feature sets generated by a network with the same hyperparameters. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. 62
8.7 Validation IC and $S_{0.15}$ score as features produced by larger AE are selected. Values are the average of two runs with feature sets generated by a network with the same hyperparameters. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. 63
8.8 Validation IC and $S_{0.15}$ score as features are added. Values are the average of two runs with feature sets generated by a network with the same hyperparameters. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. 64
8.9 Validation IC, $S_{0.15}$ score, and test IC as features are added. Original set of features and those produced by small AE with aux. loss are used. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. 65
8.10 test IC for each window for LSLR model trained with features shown in 8.9. The out-of-sample test window of t02-14_v15 performs slightly worse than previous years. 66

8.11 Weekly IC and its 4-week average for LSLR model trained with features shown in 8.9. Weeks in 2015 are out-of-sample. 66
8.12 Comparison of sector-level model developed in this thesis (Sector level with Linear Regression and Feature Selection – SLRFS) and the Highstreet model. . 70

List of Tables

5.1 A comparison of word sentiments when they appear in conversational English versus the Loughran and McDonald dictionary (LM). No words could be found which had a positive sentiment in the LM dictionary but a negative usual sentiment. . . . 32

5.2 Examples of negations and boosting words used by VADER. . . . 33

7.1 Baseline results . . . 49

7.2 Baseline neural network hyperparameters . . . 49

8.1 Baseline results with $S_{0.15}$ . . . 55

8.2 Model performance as features are added. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score. . . . 56

8.3 Only fundamental factor features . . . 58

8.4 Only GICS features . . . 59

8.5 Only simple sentiment features . . . 59

8.6 Final results for models, also showing 2015 test IC and standard deviation. Note that for the validation values, the IC is the average of window IC averages, and the standard deviation is the of the window ICs. For the test values, the IC is the average across the weeks in the year, and the standard deviation is of the ICs in that year. . . . 67

8.7 The sector groups to be used for training each model. The number of samples corresponding to each group in the test data is also shown. . . . 68

8.8 Number of features selected for each sector model. . . . 70

8.9 Comparison of sector-level model developed in this thesis (SLRFS) and the Highstreet model. The better model for each sector is bolded. . . . 71

Chapter 1

Introduction

1.1 Purpose

The ability to forecast changes in stock price has strong monetary motivations. Investment firms exist under the assumptions that a significant amount of money can be made by investing intelligently, and individuals can make (or often lose) thousands of dollars in a single day on a stock market. A large amount of research in the area of stock performance prediction has therefore already been done. The general techniques for prediction fall into a small number of categories [7]:

1. fundamental analysis, which is based on studying the company itself through factors designed to capture its well-being,
2. technical analysis, which is based on studying historical stock prices, and
3. data mining and artificial intelligence methods, which make use of large amount of data available from numerous possible sources.

The last technique, data mining, is attractive for a few main reasons: automation results in faster model development, personal biases that may affect judgement are avoided, and more complex or subtle patterns can be discovered with machine learning techniques than can be found by hand. Multiple existing results using data mining and machine learning have also shown that data derived from textual sources related to the stocks can be successfully

used towards short term forecasting, however these approaches have suffered some important drawbacks. Many results focus on short term forecasting on the scale of days or less, as in [13, 39, 48, 47, 34], which exhibits a higher risk than for long term investing. Previous results using textual data have implemented relatively simple sentiment analysis techniques or validated their models on a relatively short time period [13, 39, 48].

In this thesis we aim to overcome these drawbacks. We propose a long term stock prediction method that uses more advanced machine learning methods to extract data from textual sources and combine these with fundamental features. To develop and test the method, we consider the data from approximately 1500 stocks which appear in the S&P 500 between 2002 and 2016. There are many ways to predict future stock performance that can be utilized by investors, however we focus on one method; predicting the ranking of stocks from highest to lowest expected relative return.

1.2 Contributions

In this thesis, we propose a method for long term stock price prediction, which makes use of textual data. To train the predictive models we use fundamental data as well as yearly and quarterly filings for the text source. To extract features from the filings, we make use of autoencoders trained with multiple losses. For experimentation, we work with real stock data from an investment company, Highstreet Inc., and compare our forecasting results with their current methods. These models developed by Highstreet are largely built by hand with expert knowledge. Our results showed that in 2015, our models perform similarly to the Highstreet models, while requiring no expert knowledge beyond data preparation for their creation and training.

The contributions of this thesis can be summarized as follows:

1. We aggregate and clean several types of features for 1400 stocks across 14 years. This is covered in Chapter 5.

2. We extract features from filings using autoencoders. The use of a second loss for training autoencoders is also explored. This second loss attempts to get the autoencoder to

produce encodings specifically for stock price prediction. This is introduced in Section 5.3.2 and implemented in Section 8.3.

3. For training, validation and testing, we use an expanding window method which aims to improve generalizability of resultant models, as covered in Section 6.1.2.

4. Create novel criteria for choosing features which can improve generalizability of resultant models, and be tuned for different levels of acceptable risk. This is introduced in Section 6.2.3.

5. We train predictive regression models using features from fundamental data and features extracted from filing text and compare the results against baseline methods. This is covered in Chapters 7 and 8.

1.3 Thesis Outline

In Chapter 2, we will cover background knowledge in stock prediction basics, text mining and sentiment analysis, and autoencoders. Next, we review previous related works in the area of stock price forecasting and text analysis in Chapter 3. Chapter 4 discusses challenges and problems faced during the project, namely market efficiency and problems introduced by our particular data sets. How we overcome those data set issues and important stages of the data preparation are covered in Chapter 5. The set of features that will be used to construct the predictive models is also described. Chapter 6 discusses metrics used to evaluate the models and the expanding windows used to try obtain more accurate and precise scores.

In Chapters 7 and 8 are the experiments and results. Chapter 7 will contain descriptions of models used to establish a baseline performance. Chapter 8 contains experiments which use feature selection to improve performance and experiments on feature extraction from text. In Section 8.5 we will obtain results with sector-level models and compare them with the results of an existing Highstreet model.

Chapter 9 will conclude and propose directions for future work.

Chapter 2

Background

The purpose of this thesis is to use data from multiple sources to create a model that can perform long-term prediction of stock performance. Thus, useful background knowledge for the project that we will cover lies in multiple fields. In particular, we will cover the basics of stock market prediction, text mining, and autoencoders.

2.1 Stock Prediction Basics

Here we will quickly introduce important concepts related to stock prediction in general and the most common types of financial analysis techniques.

2.1.1 Important Terms

While this thesis approaches the task of stock price prediction from a computer science perspective, some knowledge of the terminology from the economic domain is important. Thus, here we define important terms used later in the thesis.

Universe: When developing a stock prediction model, the model is usually created for use on a particular set of stocks, such as those in a particular industry or index. This set of stocks used that share a common feature is called a universe. In this thesis, the universe consists of stocks from the S&P 500 stock market index. [61]

Absolute return: The absolute return value of stock (or group of stocks) is its change in

value over some duration. For example, if you invest $100 in a stock at time t=0, and at time t=1 you sell the investment for $120, then the absolute return is ($120 - $100) = $20. Relative return is often expressed as a percentage of the original investment, so in this example it would be $20/$100 = 20%. [62]

Relative return: The relative return of an investment is the difference between absolute return and performance of some benchmark. For example, imagine that at time t=0, you invest $100 in a particular stock (investment A) as well as $100 in the S&P 500 index fund (investment B). Then at time t=1 you sell A for $95, and sell B for $75. The absolute return of A is -5%, and the absolute return of B is -25%. However, the relative return of A with respect to the S&P 500 index is 20%. [62]

Backtesting: To evaluate the performance of a stock investment strategy, one generally wants to know its performance in terms of how much money they can expect to gain with it. One way the performance of an investment strategy can be tested is by using it in the real world. Alternatively, it can be tested by simulating its usage on historical stock price movement, where the profits and losses are calculated as if the strategy were applied at the time. Testing a strategy with real investments is risky because the performance of the model has not yet been estimated. Estimating the model performance on historical stock price movement before using it is thus beneficial. The underlying assumption is that if a strategy would have performed well in the past, then it is likely to perform well in the future. However, backtesting must be performed carefully, and naively performing it can lead to wildly inaccurate estimated of model/strategy performance. Backtesting will be further discussed in Chapter 5. [8]

Portfolio: A portfolio is a group of financial assets (such as stocks and bonds), constructed with the intent of earning a return. [52]

2.1.2 Types of Financial Analysis

There are two main types of analysis that have traditionally been employed to predict stock price movement, each focusing on using a different type of data. In Chapter 3 we will look at specific ways each type of analysis is used to predict future stock prices.

Fundamental analysis: Fundamental analysis makes use of factors relating to the well-

being of the company, such as revenues and expenses, market position, and annual growth rates [7].

Technical analysis: Technical analysis aims to predict future values using historical stock prices. The main underlying assumption motivating technical analysis is that there are patterns which can reliably be used to predict future values [7]. John Murphy provides a comprehensive look at technical analysis methods and application in [46].

2.2 Sentiment Analysis

To train a predictive machine learning model, structured training data is needed. Most text humans consume and produce is largely unstructured natural language text [44]. In order to extract structured data from unstructured data, information extraction or text mining can be performed.

One type of feature that can be extracted from text is what sentiment it contains. Sentiment analysis is a relatively old technique, with early applications including beliefs, metaphor interpretation, point of view, affect, and related areas. More recently, an increase in popularity of machine learning and availability of large datasets has led to an increase in academic awareness of the field [50]. As a concrete example, one type of sentiment analysis commonly used for analysing product reviews is the extraction of sentiment polarity.

Before sentiment can be extracted from text, earlier, more fundamental features from the text need to be extracted. These types of features include the presence and frequency of terms (commonly used in bag-of-words approach) [58, 68], occurrence of specific subsets of terms (as in noun phrase approaches) [58, 68], and parts-of-speech tags [50, 52, 68]. With these features, a machine learning model (such as classifier or regressor) can be trained which takes as input the features associated with a piece of text, and output the predicted sentiment. Excellent overviews of the are are provided by [50] and [68].

When using the bag-of-words approach for sentiment analysis, sentiment scores for individual terms can be specified. For example, the Loughran-McDonald dictionary contains sentiment scores specifically for language found in a specific type of financial statement, since the words do not always take on the usual sentiment [41].

2.3 Autoencoders

While text mining feature extraction methods often make use of properties specific to language and text, more general algorithms exist for extracting and generating features from arbitrary high dimensional data. Some popular types of methods include feature subset generation (really a large set of methods) [26], PCA [57], and random projections [12]. One particular feature extraction method used in this thesis makes use of autoencoders, a type of neural network [29].

An autoencoder (AE) is typically a feedforward neural network trained to reproduce the input, although recurrent versions do exist [42]. The structure of most autoencoders is quite simple, as seen in Figure 2.1. They commonly consist of an input layer, followed by an even number of fully connected hidden layers of decreasing size, a hidden layer in the middle, and a symmetric set of layers of increasing size following. If there are multiple hidden layers, it is considered a stacked autoencoder. For example, in an AE with 5 layers, if the first three layer sizes are 100, 50, 10, then the next layer sizes should be 50 and 100. The input and output layers must be the same size since the network is to reproduce the input. The first half of the AE is called the encoder, with the very middle layer the source of the hidden representation (or encoding) of the input, and the second half of the layers is called the decoder since it must take the encoding and decode it to reproduce the input.

The main purpose of an autoencoder is to learn structure present in data instead of simply learning the identity function. To ensure that this is done, constraints are applied to the network which limit its likelihood to rely on an identity mapping. For example, consider the case where an autoencoder is trained to reproduce vectors which consist of all 1s or all 0s. If the hidden layer(s) are the same size as the input, it could propagate the signal through the network unaltered (identity mapping), but we know that the information in the images can be stored in a single bit. If the autoencoder were able to reproduce the vectors with a single hidden layer of size 1, we would consider it to have learned the structure present in the data. For more complex data, autoencoders are often used with the assumption that they contain enough structure to let them be compressed using hidden layers of smaller size than the input.

There are many kinds of AEs, each with different types of constraints which force them to learn efficient encodings of data. In this thesis, we focus on the traditional and simplest kind

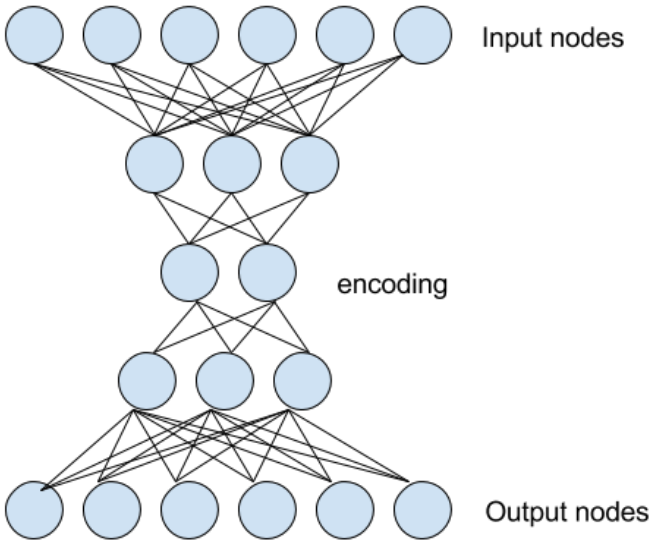

Figure 2.1: Basic autoencoder structure

of autoencoder, where the only constraint is that the encoding layer has fewer nodes than the number of input nodes (called an undercomplete AE) [71].

In Section 5.3.2 we will discuss in more detail how we will use AEs to assist in stock performance prediction and in Section 8.3 we will implement them and obtain results. This approach to text feature extraction is largely unexplored, so in the next chapter on existing work and research in the area of stock performance prediction, we will examine the results of existing methods.

Chapter 3

Related Research

A large amount of research in the area of stock performance prediction has already been done, and in this chapter we will look at some of these results using methods increasingly similar to ours. First we look at classical approaches to stock performance prediction including fundamental and technical analysis. Next, we will look at previous approaches making use of machine learning (without text analysis), and finally we will cover approaches that used machine learning with text analysis.

3.1 Performance Prediction using Classical Approaches

There are two main types of stock prediction methods: technical analysis and fundamental analysis. Technical Analysis approaches try to find and make use of important patterns in stock price movement. These approaches are usually for shorter term forecasting (can be explained by strength of market – news only has short term effect). Fundamental Analysis approaches attempt to use important factors relating to companies whose stocks they forecast. The main idea is to estimate the true value of the company and compare that to its current stock price. For example, if an investor believes that a stock is currently undervalued, they may invest in it, believing that the stock price will rise to a more appropriate value. Fundamental analysis approaches are usually longer term. In this section we will look at important ideas from technical analysis and fundamental analysis.

3.1.1 Technical Analysis

There are many established techniques in the area of technical analysis, with Murphy providing an excellent summary of the area. In it, the author suggests that the main idea of technical analysis is to identify trends and use them. There are also three main premises underlying technical analysis:

1. *Market action discounts everything.* According to Murphy, "The technician believes that anything that can possibly affect the price – fundamentally, politically, psychologically, or otherwise – is actually reflected in the price of that market." This assumption means that instead of requiring direct knowledge of fundamentals, technical analysis assumes that all fundamental information which affects price is reflected in the historical prices.

2. *Prices move in trends.* The main idea here is that if a trend is currently in motion, then the trend is more likely to continue than to reverse. This assumption is critical because without it, money could not be reliably made by short- or long-selling.

3. *History repeats itself.* Murphy suggests that technical analysis largely depends on human psychology, and assumes that it does not change over time. This allows analysts to continue using the same patterns to identify bearish and bullish markets.

There are many ways of identifying trends, with each type of method being associated with subtypes of technical analysis. According to Murphy, until the late 1990s, the terms technician and chartist (uses charts to identify trends) meant the same thing. Beyond that point, the terms became distinct, with technicians making heavier use of statistical techniques, and charting remaining largely subjective.

Common methods chartists use for identifying trends include support and resistance levels, price patterns, moving averages, and trendlines. A common idea chartists use to predict market movement is to see whether the recent history (this term is relative since there are considered to be many scales at which trends can exist) of the price movement matches a pattern which indicates that the current trend will reverse or continue. According to Murphy, the five most used reversal patterns (with four depicted in Figure 3.1) are:

1. The head and shoulders

2. Triple tops and bottoms

3. Double tops and bottoms

4. Spike tops and bottoms

5. The saucers pattern

There are several points to consider when attempting to use reversal patters, such as the required existence of a current trend and that the larger the pattern, the larger the subsequent move.

When the head and shoulders pattern is applied for example, the analyst is checking that recent price curve roughly matches the prototypical form shown in Figure 3.1. The first upward line indicates the current trend. If the recent movement curve possesses the characteristic shoulders and head, then the expectation is that the existing trend is undergoing reversal.

3.1.2 Fundamental Analysis

While technical analysts believe that all information relevant to predicting price changes is captured by historical prices, fundamental analysts prefer to study the underlying companies to determine its true value, and thus predict whether the stock price will increase or decrease in the future. In this section, we will look at the important ideas in fundamental analysis used to determine the true (future) value of a stock.

In order to more easily compare the value of different sized companies in a similar sector, ratios are often used [38]. It is important to remove the of company size when determining performance, since the profits from stock investing are a function of the percent price change instead of absolute price change. According to [38], the main financial ratios used in fundamental analysis are:

1. Profitability ratios, which measure the earning power of the firm.

2. Liquidity ratios, which measure the ability of the firm to pay its immediate liabilities.

3. Debt ratios, which measure the firms ability to pay the debt obligations over the time.

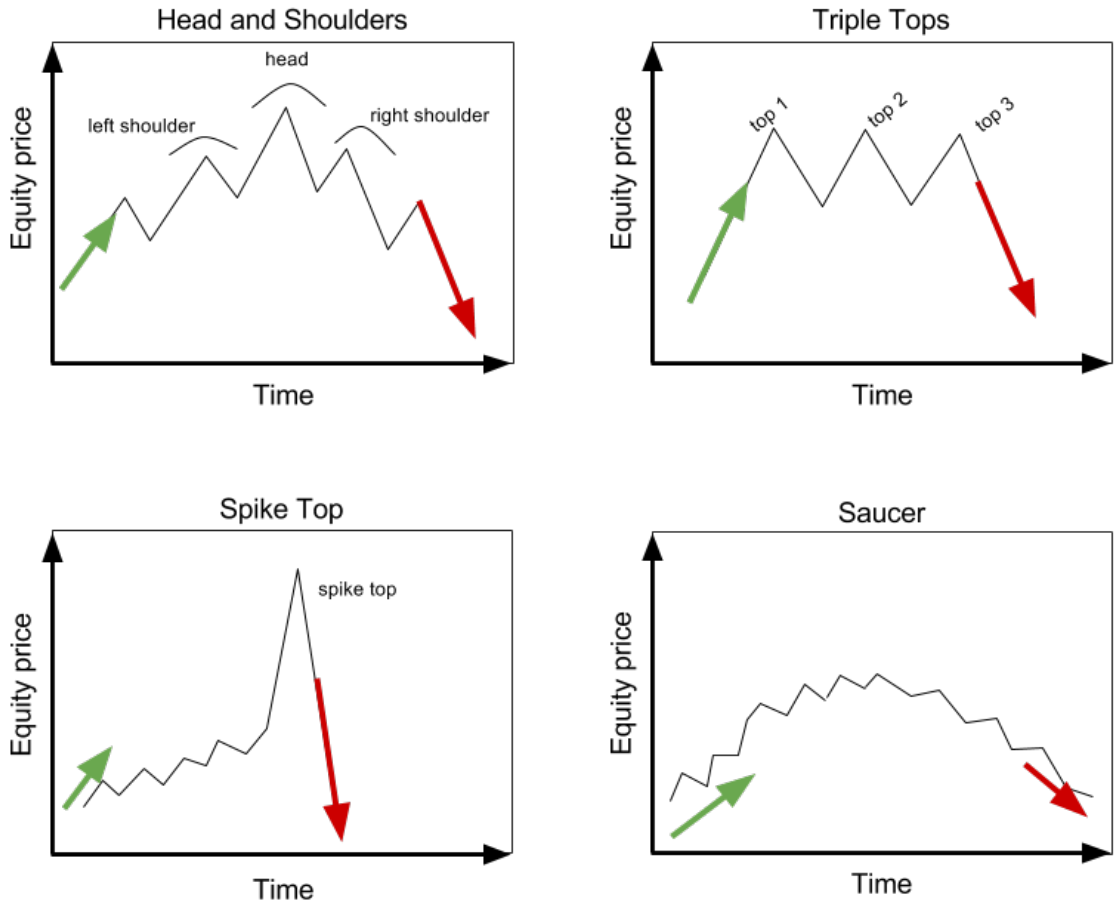

Figure 3.1: Simplified depictions of four common reversal patterns, oriented to show an initial upward trend, reversal pattern, and downward trend.

4. Asset utilization ratios, which measure the firms ability to use its assets efficiently.

5. Market value ratios are an additional group of ratios which reflect the market value of the stock and the firm.

Some profitability ratios are defined as follows [40]:

$$\text{Gross Profit Margin} = \frac{\text{Gross Profit}}{\text{Net Sales (profit)}} \tag{3.1}$$

$$\text{Operating Profit Margin} = \frac{\text{Operating Profit}}{\text{Net Sales (profit)}} \tag{3.2}$$

$$\text{Pretax Profit Margin} = \frac{\text{Pretax Profit}}{\text{Net Sales (profit)}} \tag{3.3}$$

$$\text{Net Profit Margin} = \frac{\text{Net Income}}{\text{Net Sales (profit)}} \tag{3.4}$$

The operating profit margin is arguably the most important. It is a result of subtracting operating (selling, general, and administrative) expenses from a companys gross profits and dividing by net sales. Since the management of a company has a large effect on operating costs, trends in this ratio over time can be attributed to management decisions. Investment analysts often prefer this metric for its reliability in comparing companies and composing financial projections [40].

However, it is important to note that even if the ratios for a stock are better than those of a similar company, that does not always mean the stock should be bought. One reason for this is that although a stock may perform better than similar ones, the entire market, industry, or sub-industry may underperform [38]. Additionally, when making long-term investment decisions, analysts must also take into account not only current performance, but estimate the potential for future earnings by a company in the future [38].

3.2 Performance Prediction using Machine Learning

Now, we will look at works which have used machine learning for stock performance prediction, but not including those approaches which used features derived from textual data.

In [56], similar to the task approached in this thesis, they use regression models to predict relative price change, and then rank order the stocks for usage in portfolios. One of the initial motivations for their work was to study why the Value Line ranking system [11] commonly experiences moderate prediction accuracy with short periods of very bad performance. As training and testing data, they started out with data for 1,600 equities from the Value Line universe from March 1, 1992 to Dec 1, 2001. After cleaning, they were left with 1452 equities. However, in any given quarter, similar to an issue we face, not all 1452 stocks will be ranked, since new corporations sometimes replace existing corporations in the universe. For a predictive regression model, they chose to use small recurrent neural networks with a single hidden layer.

As an input to their algorithm, [56] uses only the ten prior quarterly percentage changes in price and earnings for each stock, converted to a relative rank scaled around zero. The output of the neural network is the predicted relative price change over the next quarter. The changes for each stock are then rank ordered, resulting in a finer grained ranking than the one provided by Value Line. The Value Line ranking method uses information about historical relative earnings and a proprietary algorithm to combine the values into a forecast, and then ranks all of the stocks in its universe of approximately 1,700 from 1 (best) to 5 (worst). Since the initial weights for the neural networks are random, they average the results of multiple initializations. Unlike our work (although something similar to this is performed by Highstreet during testing), they go a step further, and with the predicted ranking every month, construct several portfolios which use increasingly more of the top equities to buy, as well as equally many bottom equities to sell short. The authors report that over 29 quarters between 1994 to 2001, the model consistently outperforms the Value Line ranking system in terms of total returns. They also analyze the performance in terms of the Sharpe ratio, which calculates a risk-adjusted return (discussed in Chapter 6). Approximately, the Sharpe ratio is the average relative return divided by the standard deviation of the relative returns over a period of time,

thus large variations are penalized, resulting in a lower score. What they find is that although their method's best portfolio achieves a value of 0.55, which is better than the VL ranking, it is much lower than the 1 to 2 which hedge funds generally claim to achieve.

In [73], new method for the stock selection problem is presented. The goal is to predict the ranking of stocks one week in advance for use in portfolio management. The dataset they use consists of 504 individual stocks with the largest market caps chosen from the NYSE and AMEX over 1963 to 2004. The reason for selecting the stocks with the largest caps is that choosing the predictive model that is best at predicting performance of all stocks is not what they are looking for, but rather models that achieve high accuracy for the stocks likely to be at the top. Features obtained for each stock on each week to be used for prediction are: return of stock from one week ago, return from two weeks ago, and volume ratio defined as $\frac{V1-V2}{V1+V2}$, where $V1$, $V2$ are respectively the volume values for the stock one week and two weeks earlier. To perform ranking prediction, they develop a new algorithm called Prototype ranking, which applies a modified competitive learning [24] technique to predict rank of stocks. To evaluate their models, trading simulations using the real stock data was performed. For every simulated week, n stocks with highest predicted and lowest predicted rankings are chosen to construct a portfolio. The top n stocks are to buy and bottom n to short sell. For a performance metric, they use the average return of the portfolios over the simulated period. To avoid data snooping [65], they use data from 1963 to 1977 for choosing optimal model parameters and use 1978 to 2004 for learning and testing. One of the main conclusions drawn is that their Prototype ranking algorithm can be used to create portfolios which earn a much higher average return as well as higher risk-adjusted return compared to Coopers method [16]. For risk-adjusted return, the Sharpe Ratios of three of their portfolios over 1978-1993 are 0.51, 0.52, and 0.52 respectively, and 0.21, 0.26, 0.27 for 1994 to 2004 (n=5, 10, 15).

In [7], the authors aim to construct a model to help investors decide the best timing for buying and selling stocks. Their dataset consisted of three major companies from the Jordan stock market included in the Amman Stock Exchange: one from the banking sector, one from the services sector, and one from the industrial sector from April 2005 to May 2007. The input to their model consists of five price related features:

- Previous day closing price

- Current day opening price
- Current day minimum price
- Current day maximum price
- Current day closing price

However, instead of representing these with continuous values, they convert them to one of three classes (positive, equal, or negative) by comparing the attribute on one day to the same attribute for the previous day. The output of their model is either to buy or sell. While this classification format is not suitable for our task of ranking, it can still be used to construct portfolios. As a classification algorithm, they used decision trees (using either ID3 [53] or C4.5 [54]). This choice was made because construction requires no machine learning domain knowledge, the fitted models can be easily interpreted, and they are simple and fast. They analyzed the performance of their models using both 10-fold cross-validation and a 66/33 train/test split. In both cases, the classification accuracy was around 45 to 55%, which they consider poor performance compared to other approaches such as [70], where they achieved about 74% using a neural network trained and tested on price data from Hong Kong and Shanghai Banking Corporation Holdings from January 2004 to December 2005.

3.3 Performance Prediction with Text Analysis

In this section, we finally look at previous work which has combined machine learning with text analysis for the purpose of stock performance prediction.

In [58], the authors explore what combination of technical analysis techniques are most valuable to stock price prediction. In particular, they combine precise textual representations and historical price information to predict discrete valued stock prices twenty minutes after a new article has been released. Their dataset consists of news articles and stock quotes updated every minute on S&P 500 stocks during a 5 week period from Oct. 26 to Nov. 28, 2005. Over this time, 9211 news articles were gathered. The research focused only on companies listed in the S&P 500 as of Oct. 3, 2005. From the news articles, different textual representations

including bag of words, noun phrases, and named entities were used to extract features. Using these stock price and textual features, the authors estimated a discrete stock price twenty minutes after a new articles was released. As a classifier, the authors used a SVM derivative specially tailored for discrete numeric prediction and models containing different stock-specific variables. The simplest model the authors tried used only regression applied to historical stock prices. When a news article for a stock was released, the model estimated what the price would be 20 minutes after release. To do this, they applied linear regression on the stock price data available 60 minutes before the news was released, and extrapolated the stock price to 20 minutes beyond the release time. Another model which used both article terms and stock price to make predictions had the best performance when measuring the closeness to the true future stock price, measuring the accuracy at predicting direction of price movement (57.1% directional accuracy), and it had the highest return when simulating trading (2.06% return). In another set of models tested, the authors also found that a Proper Noun scheme performed better than the standard BoW method. In order to evaluate model return rates with trading simulation, they developed a trading engine with rules similar to that of Mittermayer, which aims to maximize short-term trading profit [45]. Overall, the authors conclude that the Bag of Words approaches did not perform as well as the Proper Nouns, and suggest the reason may be that BoW relies on too noisy of data.

In [48], the authors aim to use posts from microblogging (such as Twitter and Tumblr) to make short term (up to 10 days) stock performance predictions. The authors chose to use microblogging for its succinctness, high volume and real-time properties to answer the following main questions:

- How well can stock micro blog sentiment predict future directional stock price movements? Specifically, how well can a bullish (or bearish) sentiment extracted from a stock micro blog predict a future upward (or downward) stock price movement?

- Is there any difference in the predictive power between bullish and bearish postings?

- Can stock micro blogs predictive power and its difference between bullish and bearish postings be explained by any existing theoretical framework?

For their research, data was collected from a period of 89 days from May 11, 2010 to August 7, 2010. A total of 72,221 micro blog postings from 3874 distinct authors for 1909 stock tickers were used. Interestingly, just 10% of the stock tickers were responsible for over 70% of all posts gathered. To prepare the training data, 10% of the posts were manually labelled with a 1 for bullish sentiment, a -1 for bearish, and a 0 for neutral sentiment. To extract features for each of the posts and to label the remaining posts, a BOW approach was used. The output of their market prediction model for a single stock is simply up or down, adjusted to account for the movement of the market (Dow Jones Industrial Average). To test their approach, for every day in the dataset and for every stock, they combine the sentiment features of the posts on that day with others such as author and market information, and then use a classifier to predict the future price movement. The authors concluded that blog sentiment does contain valuable information for investment decision making, and suggests that the success of sentiment supports the hypothesis that irrational investors have a measurable effect on market prices. Also noted is that a short three month span does not allow for strong conclusions, since predictive power of the models could be influenced by factors including economic sway, politics, and the season.

Yet another source which used news sentiment to perform stock price prediction is [39]. In this work, the authors suggest that instead of the news sentiment reflecting the future stock prices directly, investors derive sentiment from the news, which then affect the stock market. The goal of their work was then to compare the performance of various sentiment analysis methods when predicting stock returns. The dataset used in this study includes stock prices and financial news articles related to the Hong Kong Stock Exchange over the duration of five years obtained from the FINET news archive. To obtain features from the news articles, they used the Harvard IV-4 psychological dictionary and Loughran-McDonald financial sentiment dictionary [41], which provide estimations for the sentiment of words along many dimensions. To obtain labels for each stock on each day, the daily return values are converted to either a positive, neutral, or negative class as follows:

$$L(x) = \begin{cases} \text{positive} & \text{if } R(x) \geq threshold, \\ \text{neutral} & \text{if } -threshold < R(x) < threshold, \\ \text{negative} & \text{if } R(x) < -threshold. \end{cases} \tag{3.5}$$

where

$$R = \frac{close - open}{open}. \tag{3.6}$$

The performance of each sentiment analysis method tested is then determined by the classification accuracy of a support vector machine model trained using the features provided by the sentiment analysis method. This paper has a number of main interesting results. First, they show that at the levels of individual stocks, sectors, and indexes, models using sentiment analysis over bag-of-words perform better on both validation data and previously unseen data. Second, the authors results suggest that a simple sentiment polarity cannot provide predictive power. Third, the authors observed very little difference in predictive performance between the two sentiment dictionaries. This last point is interesting because while the Harvard IV-4 sentiment dictionary contains 182 sentiment dimensions grouped into 15 categories for each of 10,000 words, the Loughran and Bill McDonald sentiment dictionary only contains around 4,000 words and maps words to 6 sentiment dimensions.

Chapter 4

Challenges

In this chapter we will discuss the main challenges faced during the project. These challenges fall into two main categories. First, there are challenges inherent in the task of stock forecasting. Second, there are challenges related to the data, both its processing and quality.

4.1 Market Efficiency

Economist Eugene Fama formulated the Efficient Market Hypothesis (EMH) in 1970, which proposed that it is impossible for an investor to outperform the market, since the current stock prices reflect all available knowledge. This does not mean one cannot make money by investing in the stock market, but rather one cannot perform better than the overall trend [63]. The EMH has become a fundamental idea in stock market prediction and is one of the main challenges faced when attempting to predict stock price changes [58]. This is because market efficiency can determine how hard it is to find usable patterns to inform investments and outperform the market [19]. According to [56], it has been argued that due to market efficiency, approximately 75% of stock investment managers fail to perform better than an investor who had invested in a market-weighted basket of every stock.

There are considered to be three degrees of market efficiency: weak, semi-strong, and strong [31]. In a weakly efficient market, you cannot use only historical prices to beat market. In a semi-strong market, you cannot use any publicly available information, including historical prices, to beat the market. Finally, in a strongly efficient market, access to all information held

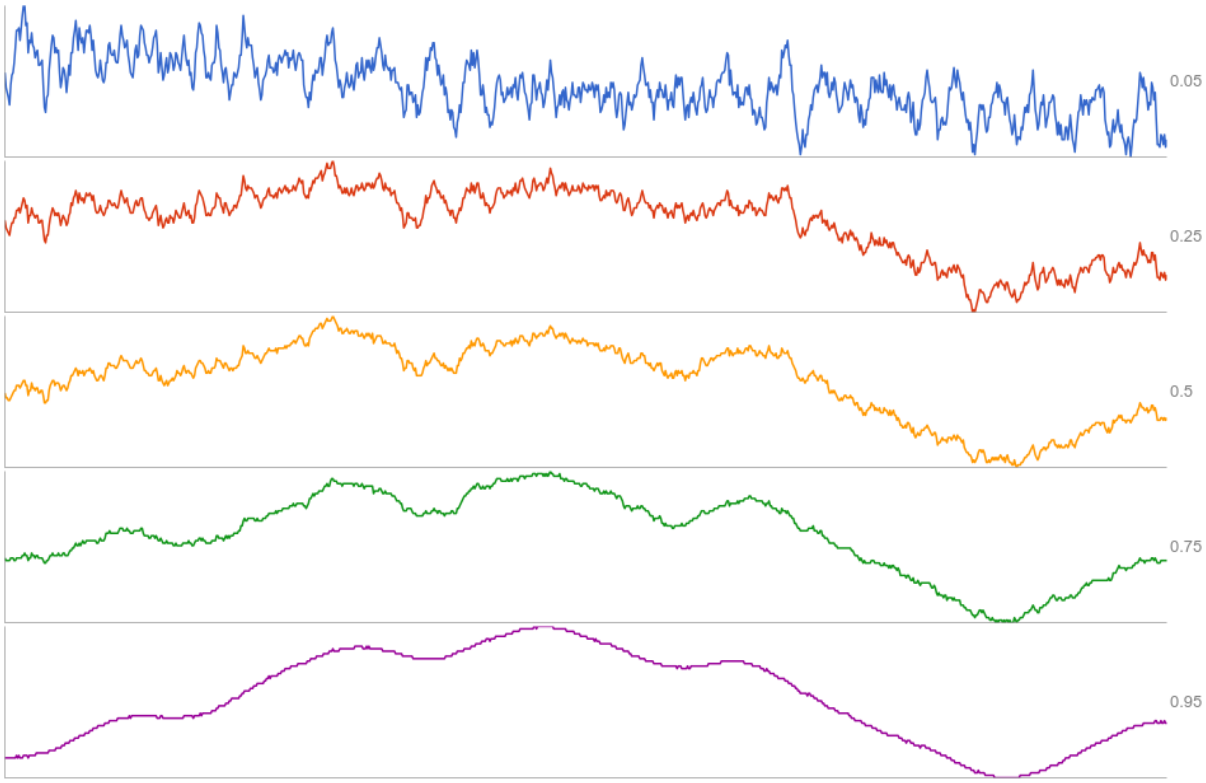

Figure 4.1: Curves with increasing estimated Hurst exponent.

by anyone is not enough to beat the market. In [20], the authors discuss how to test for this property, which allows them to develop a way to check whether insiders outperform others, potentially indicating illegal activity.

In [19], the authors examine how market efficiency affects the predictability of stock prices. They use the indices of 27 different stock markets and look at the correlation between the predictability of markets and various measures of market efficiency. Importantly, they only focus on the weak form of the EMH. The authors define prediction power as the hit rate of a model (fraction of time the predicted direction of change is equal to true direction). For efficiency measures, they use the Hurst exponent and approximate entropy to estimate efficiency, where the Hurst exponent is measure of long term memory property. In Figure 4.1, we can see examples of curves with a range of values for the Hurst exponent. The conclusion they reach is that prediction power has strong positive correlation with Hurst exponent (80%), and smaller correlation with approximate entropy (-42%).

Closely related to the EMH is Random Walk Theory [43], which states that in a strongly

Figure 4.2: The Intel stock price over 5 years (top) and random walk of length 250 (bottom). Both have roughly the same number of samples shown.

efficient market, stock price prediction is impossible since the movement is randomly determined (i.e. undergoes a random walk). In Figure 4.2, we see a comparison between the Intel stock price over 5 years (with prices shown for each week) and a random walk (with 250 steps). One could easily be led to believe both curves were generated using a random walk.

4.2 Data Challenges

The other set of problems we faced in this project were related to putting together a dataset to use for experimentation and testing. To create the dataset required gathering data from multiple sources and joining them together.

4.2.1 Filing Data

The raw data from which we acquired the 10K and 10Q filings consisted of over two terabytes of data spread across 48, 000 folders. Processing this data consistent of locating the relevant files, opening them, verifying file formatting, and analysing their text. The process of analysing this volume of data required almost a days worth of computing time, with the main bottleneck being the speed of reading files from the external hard drive upon which the files were stored. Although special precautions were taken to check code quality before analysing this data, it ultimately had to be performed several times as bugs and exceptions were discovered and requirements for the analysis added. For example, each company is expected to submit a 10Q filing once a year, and thus we expected that each folder (which contains the data for a single company for one year), would contain a single 10Q filing, however this turned out to not always hold. There are rare cases where the filings for two consecutive years end up being released in the same year. Since the file location was initially used to extract the year the filings pertained to, this caused a problem and the processing stage needed to be run again.

4.2.2 Fundamental Factors and Stock Prices

The other main types of data that needed collecting were the fundamental factors and stock prices. The raw data for these were provided by Highstreet. However, intermediate stages related to filling in missing values had to be performed. The details of this are discussed more in Section 5.1. Processing this data, unlike with the filing data, required all of it to be loaded into RAM. This processing was also done in multiple stages, so that the results from each step of this processing stage could be saved in case calculations needed to be re-done. Loading data into RAM and saving it to disk consumed most of the time when processing this data. The total time required to run these steps of processing totalled approximately 6 hours, much less than for the filing data. However, more issues presented themselves during this stage, and it was necessary to redo the processing many times. For example, while we initially assumed that we had fundamental factors available for each stock on most days, it turned out that there were particular stocks missing large durations of data, which required handling, and also particular features which were missing data for a majority of samples. As discussed in Chapter 5, we

dealt with these problems in different ways. Another issue that presented itself was in the stock price data. Although the dataset contained prices for every stock on each trading day, it was only realised halfway through the project that for many stocks, this value was only updated once a week, which led to otherwise undeservedly high performance for short-term returns.

One of the reasons for the large amount of missing data is a result of the S&P 500. At any given time, only the stocks from 500 companies are tracked, but this set changes over time. Sometimes, the set of stocks with data in a given time period can change dramatically [55]. The problem this may introduce for us is that if a model is trained to predict the performance of a given set of stocks, the patterns may not be as applicable in the future not just due to market efficiency, but because the set of stocks is changing, and each stock may possess its own characteristic behaviour.

In [58], which performed stock price prediction, stock price data was gathered for the S&P 500 stocks over the duration of only 5 weeks, and during this time, the authors note that several mergers and acquisitions took place. Fortunately, this only had an effect on less than 2% of the stocks tracked. For the experiments performed in this thesis, data is gathered over a much longer time span, thus we can expect much larger perturbations in the dataset than experienced in [58]. To visualise how the set of stocks in the S&P 500 change over the period from 2001 to 2015 (which is roughly the main window of data to be used in this thesis), Figure 4.3 was generated. Along the horizontal axis of the figure are the stocks which appear in the S&P 500 at some point. Every column in the figure shows when that stock was included in the S&P 500. To generate the figure, a simple algorithm was used:

1. a stock was chosen at random to be placed in the left-most column,
2. the unselected stock with the most similar set of days being in the S&P 500 (as measured by the taxicab distance) as the most recently added stocks is appended to the right,
3. if all stocks selected, done, otherwise repeat step 2.

One way to deal with this kind of dataset non-uniformity is by only focusing on stocks which exist in the S&P 500 over the entire time span of the dataset, however, we decided to keep the entire set of stocks. In the next chapter we will go into detail about how the data for these stocks were compiled from multiple sources into a single dataset.

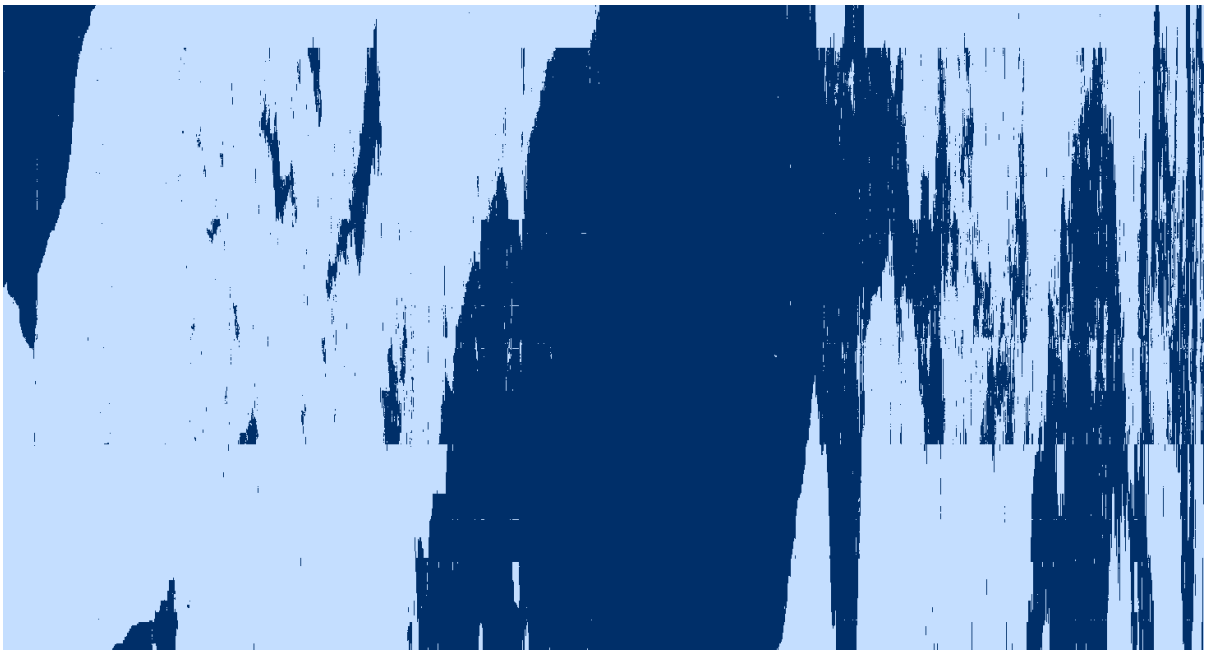

Figure 4.3: A visualisation of the set of stocks in the S&P 500 between 2001 and 2015. The x coordinate of a pixel determines what stock it is, and the y value is the date (2001 at top to 2015 at bottom). If a pixel is light, it means no data was available, if it is dark, data was available.

Chapter 5

Data Preparation and Feature Set

To develop and test predictive models, we use three main types of data: fundamental factors, filing data, and stock price data. This data is required for a universe of approximately 1500 stocks which appear in the S&P 500 between 2002 and 2016. The fundamental and stock price data has been provided for use by Highstreet. The filing data consists of 10K and 10Q reports, and were obtained from EDGAR [3], sparsely covering the time span from 1994 - 2016. In the following sections of this chapter, we will discuss how each of the types of data are compiled and prepared.

5.1 Fundamentals

As discussed in Section 3.1.2 on Fundamental Analysis, important factors reflecting the performance of a company are often given in ratios, with some important ones being profitability ratios, liquidity ratios, and debt ratios.

The raw fundamental data for stocks in the universe provided to use by Highstreet sparsely covers the years from 2001 to 2016. For many companies that appear in the data, a considerable fraction of days may be missing, as visualised in Figure 4.3. The reason for this is likely due largely to the nature of the S&P 500 company set. As the name implies, only 500 companies at a time are part of the S&P 500, and around 25 to 30 stocks in the S&P 500 are replaced every year [55]. For stocks which are included in the S&P 500 over a time span, fundamental factors may still be missing from the database. On any given day for each stock, the fraction

of fundamental factors available thus varies between 0 and 1. As pointed out in [9, 5], the existence of missing values in the dataset can make handling data harder, lead to biases in the dataset and thus biases estimates during prediction, and ultimately invalid conclusions.

As covered in [5], preferred ways of dealing with missing values include:

- **Listwise/case deletion**: every record with missing data is removed. If data is missing completely at random and the dataset is large this will only lead to a decrease in statistical power. If data is not missing at random, listwise deletion may lead to biased estimated.
- **Mean substitution**: missing values are replaced by their averaged. The assumption is that the average value of a variable is a good guess for a randomly selected observation. As with listwise deletion, if data is not missing at random, then biases may still be introduced.
- **Full data maximum likelihood estimation**.

Other methods not described in [5] include removing all rows with missing values and using expert knowledge to manually fill in missing values. Removing entire features may be a good choice if there are a large number of features, the density of missing values for a feature is high, and missing values are localised to a small number of features. Using expert knowledge to fill in missing values is only a viable method where the number of interventions required is small.

Eventually we decided to use a combination of methods including mean substitution, feature deletion, and expert knowledge. In the cases where a fundamental factor was missing for too many samples, it was removed. The entire year of 2001 was also removed because many stocks did not start having data available until 2002. For the remaining stocks and fundamental factors, the following steps were performed:

- On each day, for each fundamental factor, we first calculate the 2nd and the 98th percentile values. We regard these 2 values as our caps, and treat all the values outside this range as outliers. We then replace the low-value outliers with the 2nd percentile value, and replace the high-value outliers with the 98th percentile value.

- After all the outliers are treated, we will start to handle missing values. Each fundamental missing value will be replaced by either min, max or median of the fundamental factor of other stocks in the same sector on the same day (determined by expert knowledge). For example, if there are 10 stocks on the same day in the same sector and one factor value is missing, then it is replaced by the min, max or median of the values of the same factor from the other 9 stocks.

After these preprocessing steps, 155 fundamental factors remained.

5.2 GICS

The GICS (Global Industry Classification Standard) is a classification scheme for companies developed by MSCI and S&P Global in 1999 [1]. The GICS classification for a company consists of 8 digits, representing its location on 4 levels of a hierarchy. The first two digits reflect what sector the company is in. The second pair of digits reflect what industry group it is in. The third pair are based on the industry. Finally the fourth pair reflects the sub-industry. For example, Microsoft Corp. has a GICS of 45103020. This means that it is in the sector of Information Technology (45), the industry group is Software and Services (10), the industry is Software (30), and the sub-industry is Systems Software (20).

For each of the stocks we experiment with, a GICS is available, and we use it as a feature. To prepare the categorical data in a GICS classification for use in our models, we convert it to a vector of real numbers by concatenating one-hot encodings (so called because only one index takes a non-zero value [28]) of each level of the GICS. If the GICS used all possible values for each level (00 to 99), then each level of the GICS encoding would require 100 dimensions. However, for each level, only a small fraction of possible values are used. For example, at the top level of the GICS, only 11 of the possible 100 values are used. This sparsity allows us to encode each GICS classification in a total of 30 dimensional encoding of the GICS, spread across four one-hot encodings. This 30 dimensional encoding will be presented to our machine learning algorithms as 30 separate features. For a reduced example of how this encoding works, consider Figure 5.1. In the example, the possible values for the GICS levels are:

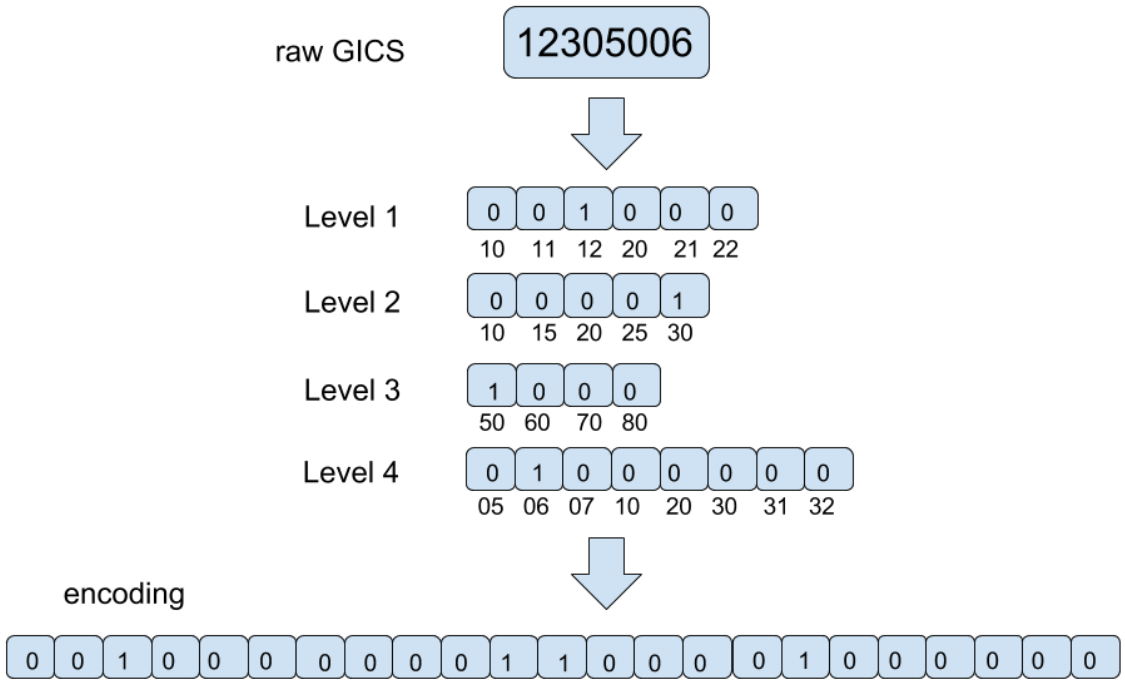

Figure 5.1: Example of encoding a GICS by concatenating one-hot encodings for each level of the GICS.

- Level 1: 10, 11, 12, 20, 21, 22
- Level 2: 10, 15, 20, 25, 30
- Level 3: 50, 60, 70, 80
- Level 4: 05, 06, 07, 10, 20, 30, 31, 32

Although more dense encoding methods exist, one-hot encodings work well with the categorical nature of the GICS levels. For example, if the encoding converted the value at each level to a percentage (the example in Figure 5.1 would be encoded as $(0.12, 0.3, 0.5, 0.06)$), then the meaning of the encoding could be very different when small amounts of error are introduced. This could make it hard for a machine learning model to utilise these features without over-fitting.

5.3 Company Filings

Along with the fundamental factors and GICS, in this thesis we attempt to extract features from textual sources to predict future stock prices. Possible sources of textual data include shareholder reports, government-mandated forms, news articles, or microblog posts concerning a companys outlook [39, 48, 58]. In our approach, we use company filings, in the form of 10-K and 10-Q filings. These forms are company generated and released with a yearly (10-K) or quarterly (10-Q) frequency. The usual format of the filings consists of a textual summary followed by numerical summaries of many aspects of financial performance. For this thesis, only the textual summary is used. The data in these reports can provide a rich source of information not explicitly shown in financial ratios indicating how the company is performing and will perform in the future [58, 35].

These filings were downloaded from the online EDGAR [3] dataset. Our portion of the dataset contains over 48,000 folders which has filings from 3,000 distinct CIKs (similar to GICS) over the years from 1994 to 2016. The 10-K and 10-Q filings were extracted from the dataset, and those without a proper body of text, before the year 2002, or with a CIK not of interest were ignored. This left a total of 65,373 filings from 1174 companies to use in the analysis.

The methods we use to extract features from the text include a simple sentiment analysis technique and autoencoders, which will now be discussed.

5.3.1 Sentiment Dictionary

One way of extracting features from textual data as discussed in Section 2.2 is with sentiment analysis. One technique for doing this makes use of a sentiment dictionary, which maps individual words to their sentiment. With the sentiment for each word in a body of text known, the sentiment for the whole can be estimated. For extracting sentiment from financial filings, we will use the Loughran and McDonald sentiment dictionary [41], which is constructed specifically for the language commonly used in 10-K and 10-Q filings. As demonstrated in Table 5.1, the sentiment that words take on in financial text does not align perfectly with the usual sentiments. For example, while the word *backdating* is usually a neutral term, when found in a

		Conversational Sentiment		
		Negative	Neutral	Positive
LM	Negative	assault	backdating	easing
	Neutral	admonish	apple	affectionate
	Positive		inventors	attractive

Table 5.1: A comparison of word sentiments when they appear in conversational English versus the Loughran and McDonald dictionary (LM). No words could be found which had a positive sentiment in the LM dictionary but a negative usual sentiment.

filing, it is predictive of bad performance for the company. The word *easing*, which is usually positive has also been determined to take on a negative sentiment in the financial context. This dictionary currently contains a list of 85,000 words and their properties.

To calculate the sentiment of a filing given a dictionary, the bag-of-words model can be used to represent the text. Given a list of all words with sentiment that could appear in the filing, the vector representing the filing can now contain the counts of each word as in Equation 5.1, where e_i is the i^{th} unit vector. If the sentiment dictionary is also represented as in Equation 5.2, the total sentiment for the filing can be calculated with their dot product: $sentiment = \text{BoW} \cdot D$. In the sentiment dictionary, the polarity is either positive or negative, so the equation can be simplified to *sentiment* = # positive words − # negative words. However, with this method of calculating the sentiment score, the magnitude scales with the length of the filing text. If we have two filings with the same density of positive and negative words but one is twice as long, then the sentiment of the longer one will be doubled, but we would likely not consider the longer one to be 'twice as happy' or 'twice as sad'. To get around this, the sentiment can be normalised, leading to Equation 5.3. Less than 1% of the filings did not have any words with non-zero positive or negative sentiments.

$$\text{BoW(filing)} = \sum_{i=1}^{\text{dictionary size}} e_i * (\text{count of } word_i \text{ in filing text}), \tag{5.1}$$

$$\text{D} = \sum_{i=1}^{\text{dictionary size}} e_i * (\text{sentiment of } word_i), \tag{5.2}$$

$$sentiment = \frac{\text{\# positive words} - \text{\# negative words}}{\text{\# positive words} + \text{\# negative words}} \tag{5.3}$$

Negations	Boosters (increment)	Boosters (decrement)
ain't	absolutely	awfully
aren't	amazingly	hardly
cannot	purely	almost
wasn't	substantially	sorta

Table 5.2: Examples of negations and boosting words used by VADER.

To improve the accuracy of this feature extraction method, techniques used in the VADER sentiment analysis tool [30] can be used. In VADER, booster words and negations, such as those in Table 5.2 are used. Whenever a negation was found immediately preceding a word with a non-zero sentiment, the value was negated. Whenever the word was preceded by a booster word, the magnitude of its sentiment was multiplied by a constant factor, either increasing or decreasing it.

5.3.2 Autoencoders

The other method we used to extract features from text makes use of a type of neural network called an autoencoder. In Section 2.3, we discussed their general structure and how they can be used to perform dimension reduction on data, and in this section we will discuss how we use AEs to extract features for our project, and later in Section 8.3 we will cover implementation specifics and experiment results of using AEs.

In this project, the autoencoders were chosen for feature extraction for multiple reasons:

1. They have been shown to work well at performing dimension reduction on high dimensionality data [29],

2. They are relatively easy to implement and train using Keras [15],

3. They are conceptually interesting and fun to use.

At a high level, the way we use autoencoders to extract features from text is by representing each document as a bag-of-words vector and then performing dimension reduction on these vectors. This dimension reduction, if the documents are not completely random, should consist of an efficient encoding of the data contained in each document. For this project, we hope that

some of those encoding dimensions have a correlation to how well the company's stocks will perform in the future. To obtain filing encodings for use in our predictive models, five high-level steps need to be performed.

Step 1. First we must obtain 10-K and 10-Q filings for autoencoder fitting. We can choose those filings we have access to which will not lie inside the training or testing data of the predictive models. The splitting of the data into these sets is discussed in detail in Section 6.1.

Step 2. Next, we need to preprocess the text in these filings and convert text to normalised bag-of-word vectors. In more detail, the way we did this is as follows:

1. For each filing, the relevant text is extracted. Doing this consists of parsing XML files to obtain a document body.

2. The extracted text is cleaned. This consists of making all of the text lowercase and removing XML terms such as & and •. All characters other than numbers, letters, and spaces are also removed.

3. The most common 3000 words among the filings are found, and for each filing the counts for each of these 3000 words are compiled into a histogram. In [32] the author performed a similar step. He used words that only occur in training data at least 3 times, and also remove stop-words. Since we are using autoencoders, the occurrence of stop-words should not have much of an effect since their occurrence rate will be roughly the same for every document, thus does not affect the encoding. However, it may also be the case that words usually classified as stop-words do have a predictive power on stock performance, in which case it is beneficial to keep the words.

4. The bag-of-word vectors are normalized:

   (a) Scale each vector to have a sum of 1. This removes the effect of text length by converting word counts to word density.

   (b) Z-score normalization is performed so that each feature is transformed to have a zero mean and unit variance. This transformation ensures that if a word has a very high or low frequency in every filing of the dataset, then that dimension in the vectors does not take on values only in a small range.

(c) Values in each vector are clipped between -10 and 10. If some filings have a word with an unusually high or low frequency, then after Z-score normalization, the corresponding value for that word will be very high or low (a large number of standard deviations from the average frequency). When this value is used in backpropagation to train the autoencoder, it may lead to very large gradients, which negatively affect learning. This problem is most notable when training recurrent neural networks, and clipping values is one strategy to overcome this [51].

Step 3. Next, we need to train the autoencoder to encode and decode the filing vectors so as to minimise a reconstruction loss. The autoencoders are fitted so as to minimize the mean squared error loss as in Equation 5.4, where $\hat{y}$ and y are the reconstructed and true normalized vectors respectively. In this thesis, the use of a second loss when fitting autoencoders was also explored. Instead of fitting the AE to only encode and decode the histograms, they were also trained so that the encodings could be used to predict the relative return associated with each filing. Since a company only releases a 10K filing once a year and a 10Q filing every quarter, the return value associated with each filing was calculated to be the average return of the stock from the time the filing was released until the next one of the same type was released. The structure of the AEs set up to perform this is shown in Figure 5.2. For these autoencoders with a main and auxiliary loss, the final loss function is shown in Equation 5.5, where $\alpha \in [0, 1]$ defines the weighting between the main and auxiliary losses. The motivation behind including this second loss was to see if the AEs could extract features from the text which are related to the performance of the stock.

$$loss_{single} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \,, \tag{5.4}$$

$$loss_{double} = \frac{1}{n} \left(\alpha \sum_{i=1}^{n} (\hat{y}_{main_i} - y_{main_i})^2 + (1 - \alpha) \sum_{i=1}^{n} (\hat{y}_{aux_i} - y_{aux_i})^2 \right) \tag{5.5}$$

Step 4. Next, we can preprocess filings which lie in the training and testing data for out predictive models. The preprocessing can be performed similarly to as in step 2. It is important not to have an overlap in the filings used for fitting the AEs and for creating encoding for the

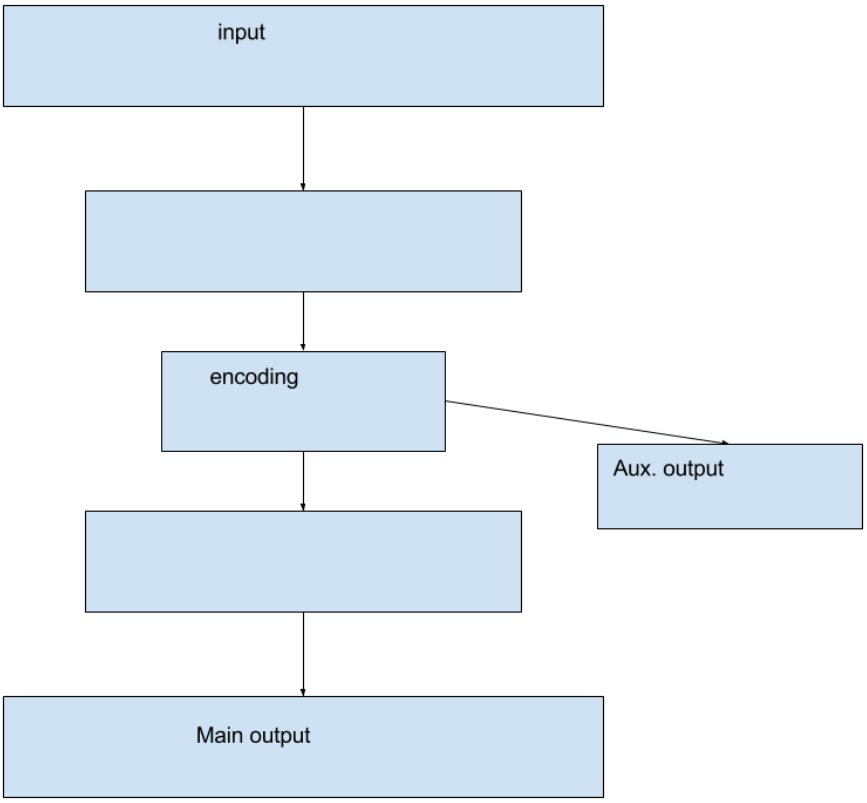

Figure 5.2: Autoencoder with a main output and auxiliary output.

predictive stock models in order to prevent over-fitting.

Step 5. Finally, these new filing vectors can be fed into a fitted AE and the encodings saved to a file in a hash table for later use.

5.4 Relative Returns

To predict the future ordering of stocks by relative return, predictive regression models are trained. The raw absolute stock price data for stocks in our universe is provided by Highstreet, and relative returns are calculated for each stock on each day. The return duration focused on in this thesis is 120 trading days (approximately 6 months). Early in the project, return durations of 5, 10, 20, and 60 days were tested, with the model performance slowly improving as the return duration increased. In the next chapter, we will discuss how to measure the performance of models train to predict these relative returns as well as how this sequential data will be partitioned to reduce over-fitting and improve model generalizability.

Chapter 6

Measuring Model Performance

When searching for and constructing financial models, it is easy to overfit. Thus, model selection needs to be carefully performed. In order to evaluate the performance of a predictive model, first it needs to be fitted to training data, next it needs to be validated on a second set of data and a validation score calculated. After using this validation score to select the model with the best hyperparameters, we can train this model using the train and validation sets (together sometimes called the design set), and get a final score on the test set. In this chapter we will discuss each of the ideas used in this process. First we will discuss how to split the data into training, validation and test samples for increased robustness. Second, we discuss the metrics used for calculating model validation and test performance. Finally we will look at how these parts all fit together.

6.1 Train/Validation/Test Splitting

When looking for a good model to use, three types of parameters often need to be optimized. First, the type of model needs to be chosen. For example, you might want to find out whether linear regression, k-nearest neighbour, or SVN performs best for your application. Next, the hyperparameters for your model need to be optimized. In the case of k-nearest neighbour, hyperparameters include the number of neighbours used, whether distance or uniform weighting is used, and what kind of distance metric might be used. Finally, the parameters of the model need to be optimized. For a linear regression model, this includes the coefficients of a linear

equation. This step is usually calling fitting or training the model.

6.1.1 Overfitting

If all of the data available is used to perform each of these steps, overfitting can occur, and the model is unlikely to generalize well to data it has not seen before. For a clear example of this, imagine you have decided to use a polynomial model for regression. If all of the data is used for training the model, selecting the degree of the polynomial, and testing it, the optimal degree of the polynomial is likely to be equal to be the number of samples. If this is the case, then the train and test error will both be zero. However, it is well known that fitting high-degree polynomials to noisy data can result in very bad interpolation and generalizability. With the data in one large set however, we cannot estimate the generalizability of the fitted model. For this reason, data is often split into a training and testing set. If the model is fitted to the training set, then we can see how well it performs on out-of-sample data by using the test set.

However, having just a train and test set does not allow us to choose the best model type or hyperparameters without overfitting. If we use the performance on the test set to choose these parameters, then we know that the model is likely to perform well on the train and test set, but again do not know how well it generalizes to unseen data. To solve this issue, the dataset can be split into three parts: training data, validation data, and testing data. The training data is used for fitting a model for when the model type and hyperparameters are being optimized. The validation data is used for optimizing the model type and hyperparameters. Finally, the test data is used for estimating the generalizability of the resultant model after it is fitted on both the train and validation data.

6.1.2 Expanding Windows

Our data set can naturally be viewed as sequential; for most weeks between 2001 and 2016, we have features for approximately 500 stocks. Since the purpose of our predictive models in the project is to take samples from before a certain date and make predictions after that date, we must split the data into training, validation, and test sets to reflect this.

A simple way of partitioning the data for training, validation, and testing it to use the first

A years for training, the next B years for validation, and the final C years for testing. However, the large amount of noise in stock data presents a unique challenge, which motivates the sliding window system, a common way of organizing sequential train, test, and validation data.

In the sliding window system, there is not a single partitioning of the data, but many. This is motivated by wanting a more accurate estimation of the validation performance, and therefore hopefully better generalization performance of the resultant selected model. The details of how this works will be discussed in the next section of the chapter on metrics.

An alternative to the sliding window system, is expanding windows, which we use in this project. Instead of each training window covering the same duration, each successive training set simply expands upon the previous one.

If the similarity of behaviour exhibited by a stock market in multiple years depended on the distance between the years, a sliding window may be a better idea, since less relevant data is removed from the training set. One reason for using an expanding window system is that it makes the most use of the data; although the large scale behaviour of the market may be different, patterns may exist on the level of single sectors or stocks which are applicable. In early experiments using 2014 as the test year, we saw that making use of an expanding window system did in fact result in better performance than sliding windows. Perhaps if even more years of data were available, the benefits of sliding windows may outweigh those of expanding windows.

For this project, we decided to use 7 windows, with each window ending one year after the previous one. To increase validation accuracy, monthly or weekly expanding windows could be used, however this would increase the computation time drastically. Since the purpose of the project is to assist in creating mid-to-long term investment portfolios, using a model which requires updating with low frequency is acceptable.

Since we are developing models to predict mid-to-long term future returns, the target value for a sample uses data which depends on the stock price some number of months into the future, relative to when the features were obtained. This means that if we are training a model for 120 day return (approximately 6 months), then the last 6 months of the training data needs to be removed. The reason for this is that if the model were to be employed in the real world and trained, it can only use data available at the time of its training. If it is currently January

1st, 2016 and we wish to train a model, the most recent data that can be used for training is from June in 2015, since its target values include data from December of 2015. This means that none of the months from July to December are directly used in training. Thus, although not entirely necessary, we decided to remove the ends of each of the datasets when used for model training to get a more accurate estimate of model performance when employed in the real world.

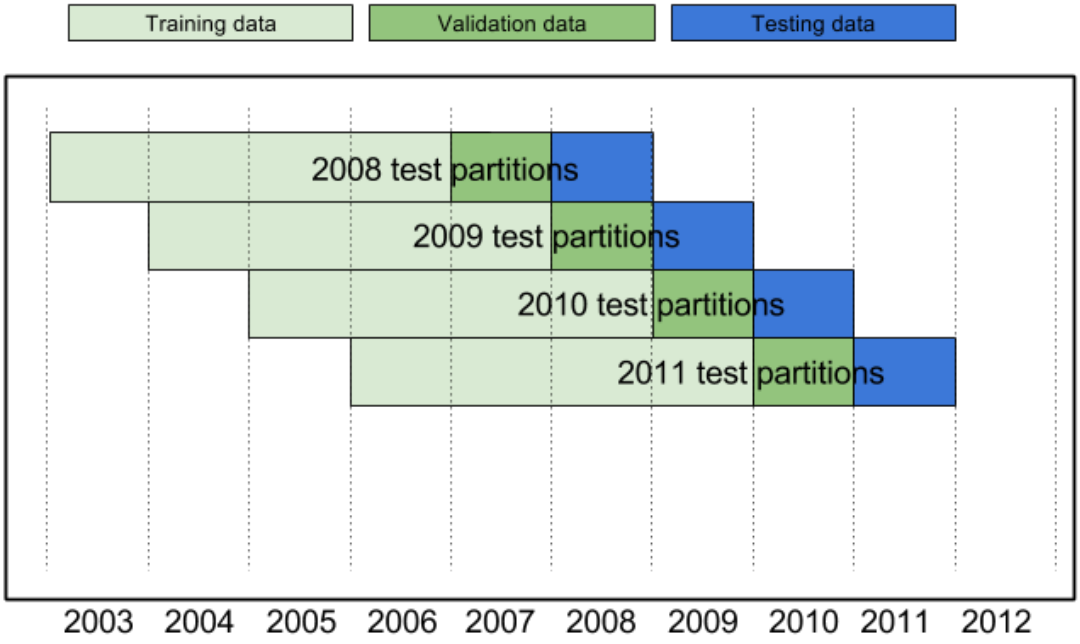

Figure 6.1: Sliding windows

6.2 Metrics

To measure the performance of a stock price prediction model, ideally, one would simulate the model being employed to construct portfolios and used for investing while retaining all details along the way such as trading costs. However, to do so requires knowledge of how exactly the predictive model is to be used. The goal of this project is to construct a model that can be used to guide the design of many types of portfolios or long term investment strategies, therefore to measure model performance, simpler, more general methods are used.

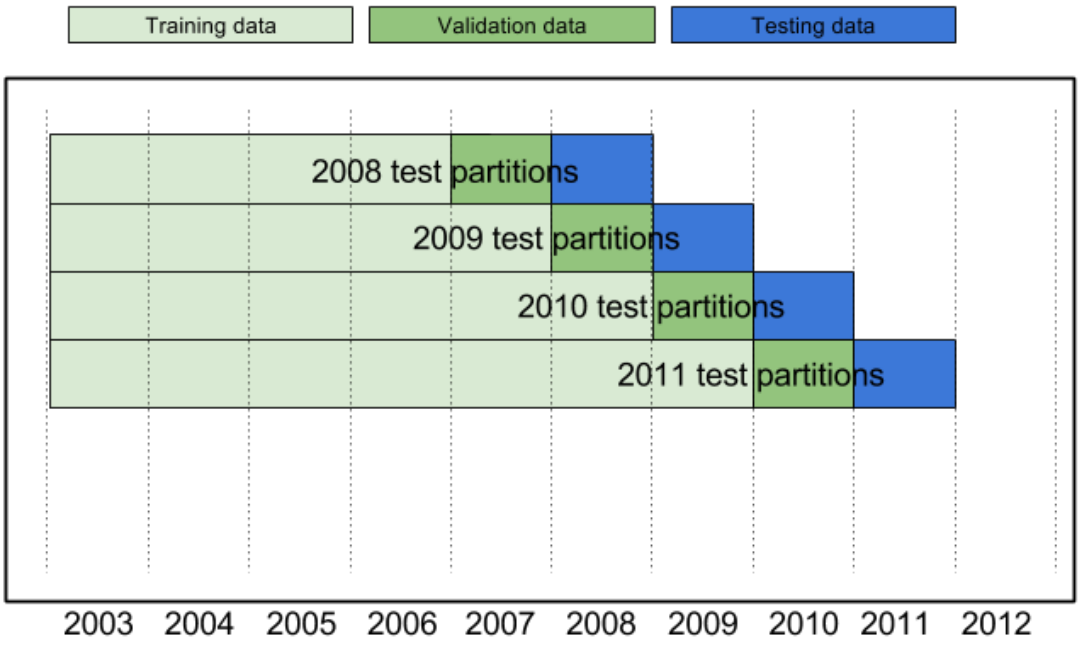

Figure 6.2: Expanding windows

There are several ways to use models to predict stock performance:

- **Regression**: With a regression model, the return (relative or absolute) or price of the stock at some time in the future is estimated.

- **Classification**: Classification models for stock performance are often used in a similar way to regression models, but with possible returns divided into a small number of classes. For example, a return less than -5% may be class 0, a return between -5% and 5% may be class 1, and a return greater than 5% may be labelled as class 2. Using this approach, a portfolio may be constructed which buys stocks labelled as class 2 since they are likely to outperform the market, and stocks labelled as class 0 may be sold if possible.

- **Ranking**: A third option for stock prediction models is to learn a ranking. One way of learning a ranking is to use a regression model to produce a value for each stock, and then use the ranking induced by sorting those values. This is different from learning a regression model to perform well at predicting return and then producing a ranking since

a regression model that leads to an accurate ranking may not be the same as one which produces accurate return estimations (except where predictions are perfect).

For this project, we tried both regression and classification techniques at first for inducing a ranking, but regression models proved to have better performance, and thus the models presented later in the thesis are based on regression algorithms. Here we will discuss how model performance is to be assessed at four stages. First, when choosing model parameters the performance must be evaluated on the training set. Second, the model performance on each of the validation sets is measures. Third, the set of validation scores are combined to produce a single validation score. Fourth, models are evaluated on the test set.

6.2.1 Training Performance

When fitting a regression model, the metric used is dependent upon the specific model. For example, when fitting a vanilla linear regression model, the Ordinary Least Squares metric is used. When using SGDRegressor provided by sklearn, many losses are available, including squared_loss, huber, epsilon_insensitive, or squared_epsilon_insensitive. Therefore, the metrics used in training will be specific to each model we test.

6.2.2 Validation Performance

Once a model has been fitted to the training data of one window, it is evaluated on the validation portion of the window. To calculate a validation score for each window, we chose the following metric:

$$\text{Validation score} = \frac{1}{\#\text{weeks}} \sum_{w=1}^{\#\text{weeks}} IC(w), \tag{6.1}$$

where $IC(w)$ is the Spearman rank correlation coefficient for the given week as calculated by:

$$IC(w) = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}, \tag{6.2}$$

where n is the number of stocks with data on the given week, and d_i is the difference between the ranking of the ith stock when using the true relative return, and the value given by the regression model. If the true ranking and predicted ranking of the stocks are the exact same, then the IC will be 1, if the correlation between the two ranking is 0, the IC will be 0 as well, and if the ranking are in reverse orders, the IC will be -1. Thus, maximizing the IC corresponds to finding models which perform well at ranking the stocks by the relative returns.

6.2.3 Combining Validation Performances

After obtaining a validation score for each window, they need to be combined into a single value. The most obvious method would be to use the average of the scores for each window, however, we decided to model our metric off of the Sharpe ratio (a risk-adjusted rate of return):

$$S = \frac{E\left[r_p - r_f\right]}{\sigma_p}, \tag{6.3}$$

where r_p is the portfolio return, r_f is the risk-free rate of return (often given by the sector or market return), and σ_p is the standard deviation in portfolio return over the considered duration. When constructing a portfolio, maximizing the Sharpe ratio corresponds to trying to get the largest return while minimizing risk. Our inter-window validation metric is also designed to both maximize the average IC across years but also minimize the variation in average weekly performance across years.

$$\text{Combined validation} = \frac{\text{average validation score for each window}}{\text{standard deviation of window validation scores}}. \tag{6.4}$$

Alternatively, one could perform the risk adjustment for each window and take the average. Maximizing this metric would produce a model which has low variability in each year, but potentially high variation in performance across years. In this thesis, we also made use of a slightly different version:

$$S_\epsilon = \frac{\text{average validation score for each window}}{\text{standard deviation of window validation scores} + \epsilon}. \tag{6.5}$$

Using this more general version of Equation 6.4 is motivated by the combination of two ideas. The first is that achieving a model with performance of a small magnitude is likely easier than creating a very good or very bad one. Second is the demonstration that investments with higher risk tend to earn more than those with less risk [56], and since the Sharpe ratio puts a high importance on minimizing risk, optimizing for it will tend to result in models with small earnings along with the lower risk. Combined, these two ideas suggest that there is a strong bias towards models with low profit (relative return or IC), and low risk (standard deviation across time). Including the extra parameter allows for changing the balance between minimizing risk and maximizing profit. Choosing a value for ϵ is not obvious however, and in Chapter 8, we will attempt to find a good value to use during the lengthy process of feature selection.

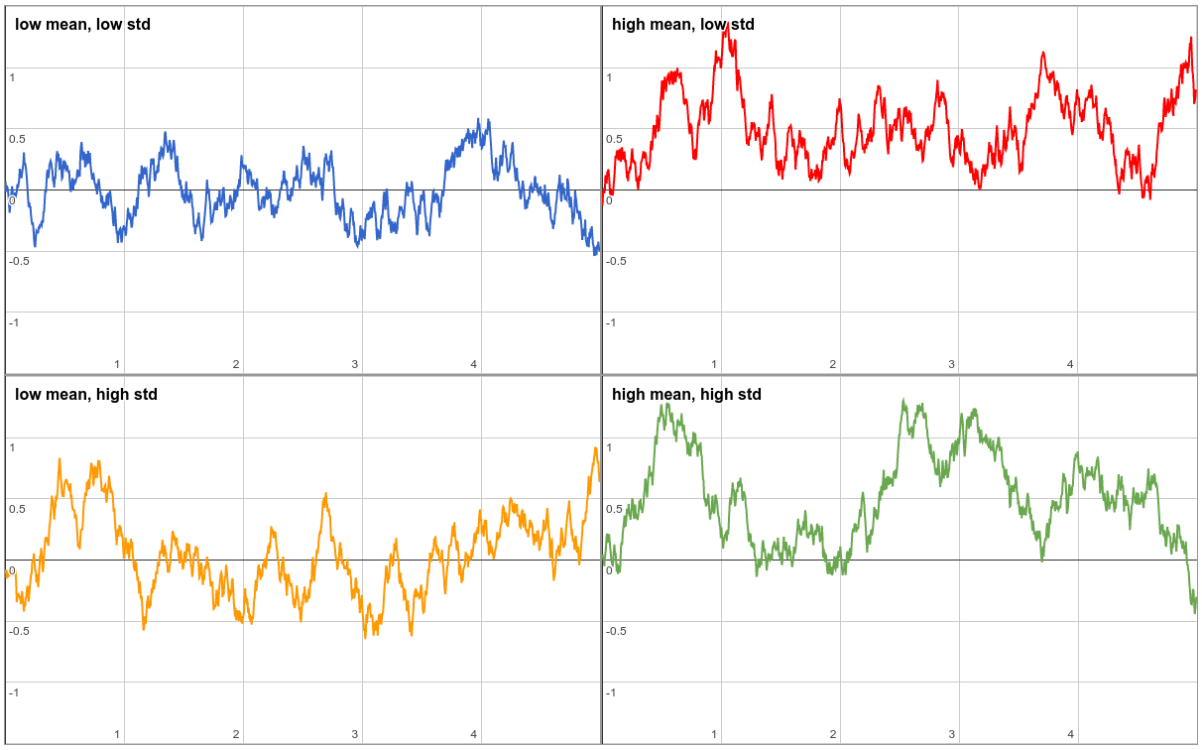

Figure 6.3: Example model performance curves in five windows demonstrating the effects of having a high and low mean and a high and low standard deviation across windows. The S_ϵ metric encourages models with high mean and low standard deviation.

6.2.4 Test Performance

After the hyperparameters for a model have been optimized, we can see how well it generalizes by calculating the performance on the test window. Since the metric used for each validation window is the IC, the metric used for the test window is also the average weekly IC.

Chapter 7

Baseline Models

To get an idea for how well a model performs, it helps to have other models to compare it to. A baseline model is often chosen to be a naive approach to the problem or to use a common algorithm whose performance is well-established. For this project, many novel factors make it hard to find models in the literature to compare our results to. We look at predicting the ranking of stocks by long term returns, while using a combination of fundamental factors, GICS, and text-derived features, a combination which is not well studied. In this chapter we will establish the performance of common algorithms on our problem so that we can later compare them to the performance of more complex approaches and models. Although a large number of algorithms were tested, only some of the more important models are included here.

7.1 Overview of Models and Training

Since there are a large number of possible algorithms to try, only some of the more popular ones tested will be shown here. The algorithms to be tested are:

- Least Squares Linear regression (LSLR)
- Stochastic Gradient Descent (SGD)
- Gradient Boosting (GB) [22] [23]
- Random Forest (RF) [14]

- Feedforward Neural Networks (NN)
- Adaboost with linear regression (ALR) [21, 18]

Before we get into the results, a few notes about some of the models:

- **SGD**: For SGD, as well as all the other models tested in this section except for LSLR, the predictions are slightly different every run, thus to get an accurate estimation of the model performance, the average score over several trials was used. For all the models, 10 trials were used. For more precise results, more trials would be beneficial, although time must be taken into account as checking the hyperparameter performance for all of the models takes on the order of a day with 10 trial averaging. An alternative is to seed the random number generators so that runs would be identical, but results from averaging over trials are less susceptible to being influenced by outliers.
- **Random Forests**: When testing small decision trees, some weeks may end up with the same prediction for every stock, leading to difficulties in calculating performance. When this occurred and the IC was undefined, we chose to simply set it to 0.
- **Neural Networks**: The space of feedforward neural network graphs is very large, and while there are many established and useful structures, for these experiments, only small networks were used which consist of a single input layer, a small number of sequential fully connected hidden layers, and an output layer consisting of a single node. In earlier experimentation, many other structures were tested, including: Ensemble, Mixture, Parallel. However, it turned out that increased complexity did not help much and a liberal application of regularisation was required to prevent overfitting.

7.2 Results

The regression algorithms will be applied to the fundamental factor features, GICS encoding features, and simple sentiment features. For each of the algorithms, a quick hyperparameter optimization is performed, and the set of hyperparameters with the best validation performance is selected to represent that method.

model	val_IC	val_stddev	val_score	val_trial_IC_stddev
LSLR	0.0231	0.0696	0.3314	**0.0000**
ALR	0.0212	**0.0300**	0.7063	0.0061
SGD	**0.0501**	0.0713	0.7026	0.0024
GB	0.0011	0.0618	0.0178	0.0190
RF	0.0179	0.0388	0.4624	0.0388
NN	0.0252	0.0334	**0.7531**	0.0227

Table 7.1: Baseline results

layer sizes	[8]
# hidden layers	1
activation	tanh
loss	mse
# training epochs	16
l2 weight	1e-6
learning rate	0.01
optimizer	rmsprop

Table 7.2: Baseline neural network hyperparameters

For each model, we will report the following values:

- val_IC: Equation 6.1, or numerator of Equation 6.4
- val_stddev: standard deviation in validation scores across windows, or denominator of Equation 6.4
- val_score: combined validation score, measured with Equation 6.4 (i.e. S_0)
- val_trial_IC_std: standard deviation in val_IC across training trials

As we can see in Table 7.1, no model tested performs the best across all fields, but the NN model has the highest validation performance according to Equation 6.4. The corresponding hyperparameters for this model are given in Table 7.2. It is clear that the model chosen is quite small, with a single hidden layer of 8 nodes. The chosen NN model also had the second-highest variation between trials.

The LSLR model will be extensively used in the next chapter for its speed and low variation across training trials, thus its performance here will be explored a little more closely. In Figure 7.1, the average weekly validation IC in each year is shown. To obtain the IC for 2014 for

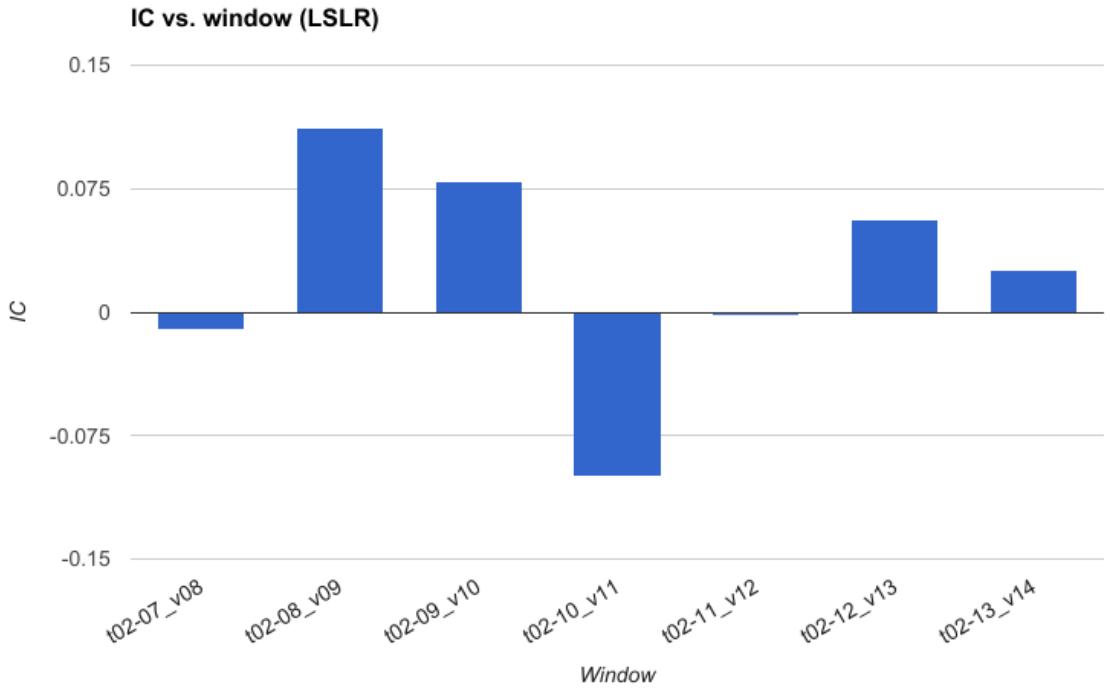

Figure 7.1: Validation scores for LSLR

example, the model is trained on data from 2002 to 2013, and validated on 2014 (hence the label t02-13_v14). As we can see, the performance for the LSLR model varies considerably from year to year.

When we look at the performance of LSLR for each validation week in Figure 7.2, we see that the performance is usually positive, with dense periods of bad performance. This type of dichotomous performance has been observed in predictive models for the stock market before [56].

In this chapter we saw that the baseline models were able to achieve a maximum IC of 0.05. In the next chapter, we will try to both increase the IC and decrease the standard deviation of ICs across years by combining the LSLR model with feature selection and more complex text-derived features.

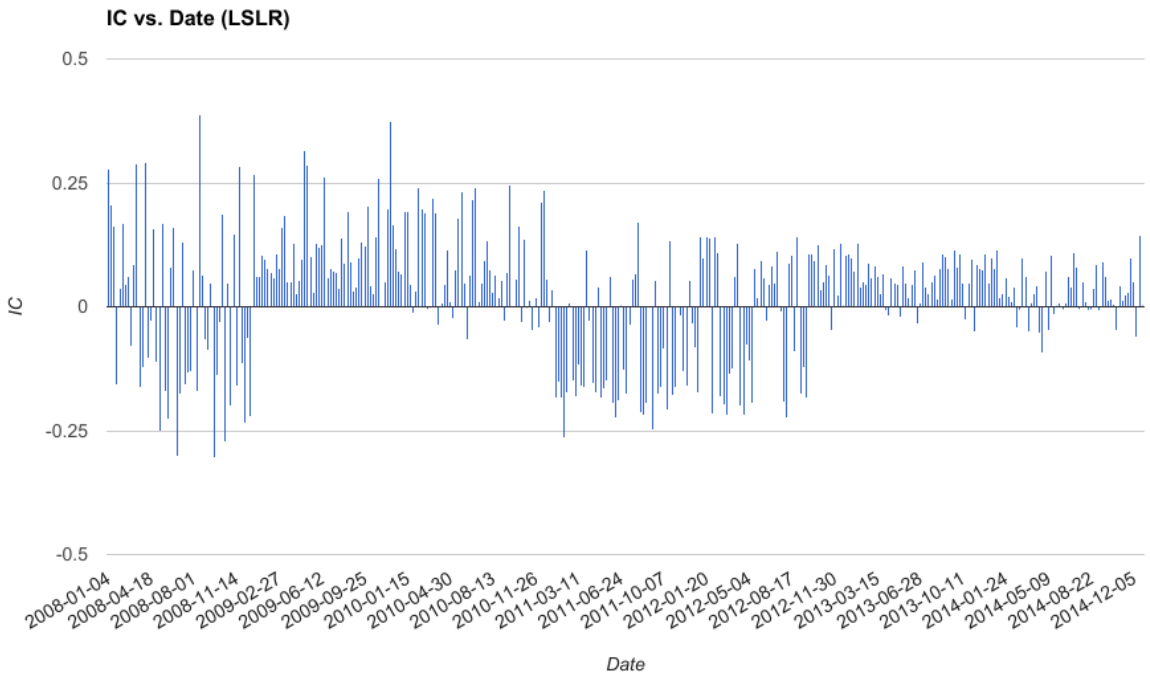

Figure 7.2: IC for each validation week for LSLR

Chapter 8

Applying Feature Selection

In this chapter we will introduce and experiment with a feature selection method. First we will look at the selection algorithm, next we will use this feature selection method to look at the effects of restricting the types of features used (i.e. fundamental factors, GICS, or sentiment) from the dataset. Finally, we will look at the performance of features extracted from text using autoencoders.

8.1 Feature Selection Method and Metric

Since we have a relatively large number of features (189), even with a linear model there is a risk of over-fitting by finding spurious patterns. Although a model may find a certain set of patterns to be useful in the training data, they may be useless or even detrimental in the following years [56]. As a simple example, an over-fitted model may learn a dictionary-like mapping from GICS and date features (which we do not include) to stock performance (this will almost work – multiple stocks can share the same GICS). The way we will try to get around this problem is with feature selection. Although feature set optimisation when aggressively applied can also result in over-fitting, we specifically try to avoid this through the use of the S_ϵ metric defined by Equation 6.5 (also known as the feature set evaluation function [6]). This metric puts importance on performance consistency across years. A motivation for this metric is the idea that if a feature consistently performs well in the past, then it is more likely to perform well in the future. Since patterns in the stock market are constantly changing (if there are any

at all – see Section 4.1), many of the features important for prediction in the test windows may be useless in the training years and validation year, so hopefully the weight on consistency will remove such features from the final set.

When performing the feature selection, a regression algorithm must be chosen to evaluate the performance of the set of features. Since this process is very time consuming, it is beneficial to use algorithms with fast training and prediction times as well as little variation in performance across trials, which is why the deterministic LSLR algorithm will be used.

The feature selection method we chose to use is a beam-search style forward sequential selection algorithm and consists of iteratively expanding the feature set to be used with the next best option [6]. To perform the feature selection, the following is performed until a certain number of features are added or until the score begins to decrease:

Algorithm 1 Iterative feature selection

```
1: function SelectFeature(B, F)  ▷ Where B - list of features already selected, F - list of all features
2:     best_feature = None
3:     best_score = None
4:     for feature in F − B do
5:         Bp = B + feature
6:         Bp_score = S_ε(Bp)
7:         if Bp_score > best_score then
8:             best_feature = feature
9:             best_score = Bp_score
       return B+best_feature, best_score, F
```

In practice, each iteration can take quite a while to run. For the results in this thesis, an optimisation was added where after every iteration, the worst scoring (bottom 20%) features in the set F were removed if the size of F was greater than 50. In earlier experimentation, multiple other techniques discussed in [6] were attempted, including backward sequential selection and randomized subset optimisation algorithms. Beam widths greater than one for the forward selection were also explored, but proved not to have a benefit over a single beam.

Using the S_ϵ with this feature selection algorithm allows for ignoring those features with low reliability and constructing a small set of features which perform consistently well across validation years. However, before we can obtain results with the feature selection, it is a good idea to get an estimate for the best value of ϵ in the metric S_ϵ to use, which influences the weight

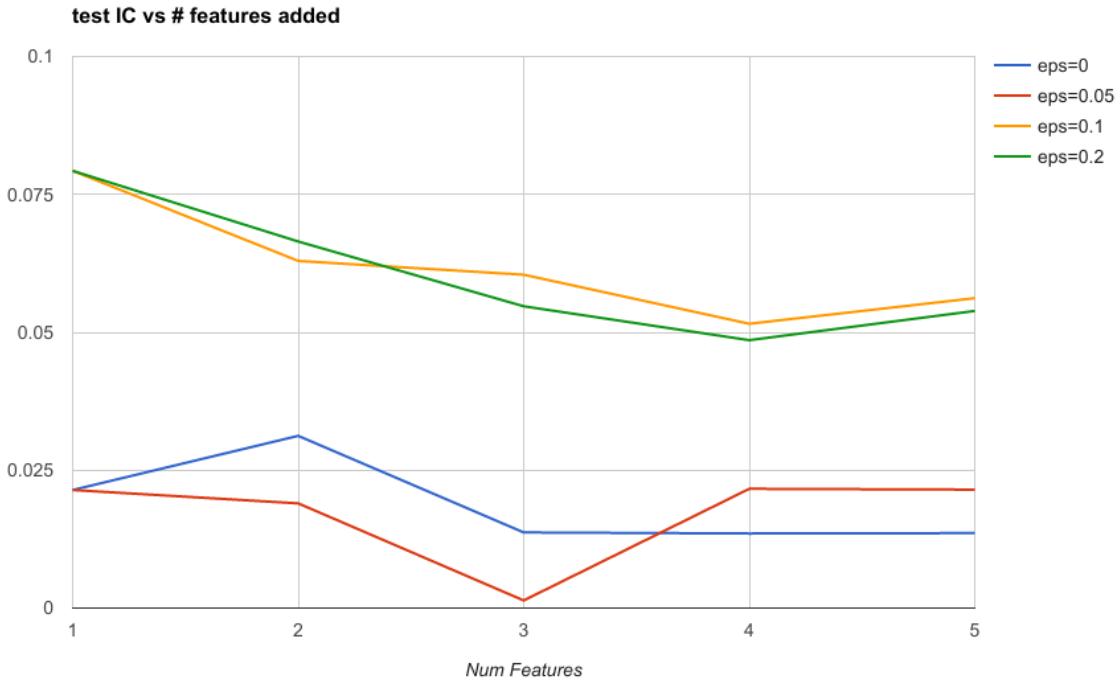

Figure 8.1: Calibrating epsilon

balance between high consistency and good performance. This selection cannot be done with a validation score, since it is what we are trying to produce, so to do this, we will use 2014 as the temporary test year, and use 2002-2013 for training and validation. For this calibration, we only went up to 5 features and ϵ values tested were 0.0, 0.05, 0.1, and 0.2. Since model performance year-to-year changes considerably, the best value of ϵ in one year may not be too close to the optimal value for another year, thus estimating it to a high degree of precision in this calibration stage is not performed. In Figure 8.1, what we find is that the best value of ϵ appears to be around 0.1 to 0.2. Thus, for further experiments, we will use 0.15. The test final performance does not appear to change very smoothly as ϵ is increased, which suggests that more carefully optimising ϵ in further experiments may be beneficial.

Now that we have a value for ϵ, we can see the updated baseline scores in Table 8.1. Interestingly, with this new validation metric, the SGD model performs best. To get these new values, the same quick hyperparameter search was done to maximise the new metric.

baseline model	val_score
LSLR	0.1050507
ALR	0.1441878
SGD	**0.2327753**
GB	0.0052034
RF	0.0949984
NN	0.1578199

Table 8.1: Baseline results with $S_{0.15}$

8.2 Feature Selection Results

8.2.1 Accessing All Features

Here we see the results of feature selection when using all of the original features (fundamental factors, GICS encodings, and simple sentiment features) in the selection process. In Table 8.2 and Figure 8.2, we see that a maximum validation score of 0.528 is reached after 16 features are selected. On our dataset, the feature selection method provides a clear improvement over the baseline scores in Table 8.1 (0.528 vs. 0.226). Using feature selection, we were able to achieve 0.226 after only one iteration with Factor_137. It is also interesting to note how the validation IC peaks at 14 features, and then decreases slightly while the $S_{0.15}$ score increases due to the inter-year performance variation decreasing.

An interesting question to ask about this feature selection method is "How much of a benefit is the iterative property?". That is, what if the features were sorted by their validation $S_{0.15}$ scores when used individually, and the top from that list were chosen? In Figure 8.3 we can see exactly that. Although the performance starts off well with Factor_137, it stays relatively constant as more top features are added. In Figure 8.8, we can see what happens if, instead of sorting by $S_{0.15}$, we sort by the validation IC. The performance is only slightly lower and the set of features at the top have a large intersection. Clearly, the iterative aspect of the feature selection is important, as it ensures that the features chosen work well together.

8.2.2 Restricting Feature Types

In this section, we will see how the feature selection method performed when only one group of features is used at a time.

Added feature	val_IC	val_stddev	val_score
Factor_137	0.05300124254	0.0347991508	0.2868045784
Factor_109	0.064387089	0.02586942825	0.3661073425
Factor_21	0.07069910576	0.02258910364	0.4096382927
Factor_99	0.07432728367	0.02408175369	0.4269676867
GICS_4_70	0.07687572059	0.02475900619	0.4398956155
GICS_4_25	0.07890748818	0.0240066607	0.4534739525
Factor_89	0.08108455964	0.02204046373	0.4713109805
Factor_100	0.08326501888	0.01720894412	0.4979698863
Factor_4	0.08411040933	0.01565303091	0.5077505004
GICS_3_20	0.08505637833	0.01631039674	0.511431516
Factor_13	0.08701098891	0.01885927511	0.5152869977
Factor_25	0.08793348136	0.01972159868	0.5181042486
Factor_65	0.08820532234	0.01940277967	0.520684032
GICS_4_80	0.08845472234	0.01897161449	0.5234886499
SCORE_10K	0.08740185402	0.01621277461	0.5258431804
Factor_15	**0.08714494319**	**0.0149387765**	**0.5283472149**
GICS_4_35	0.08714494319	0.0149387765	0.5283472149
GICS_4_45	0.08714494319	0.0149387765	0.5283472149
Factor_124	0.08682311881	0.01448618171	0.5278444542
Factor_2	0.08539276856	0.01263666027	0.5250523985

Table 8.2: Model performance as features are added. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

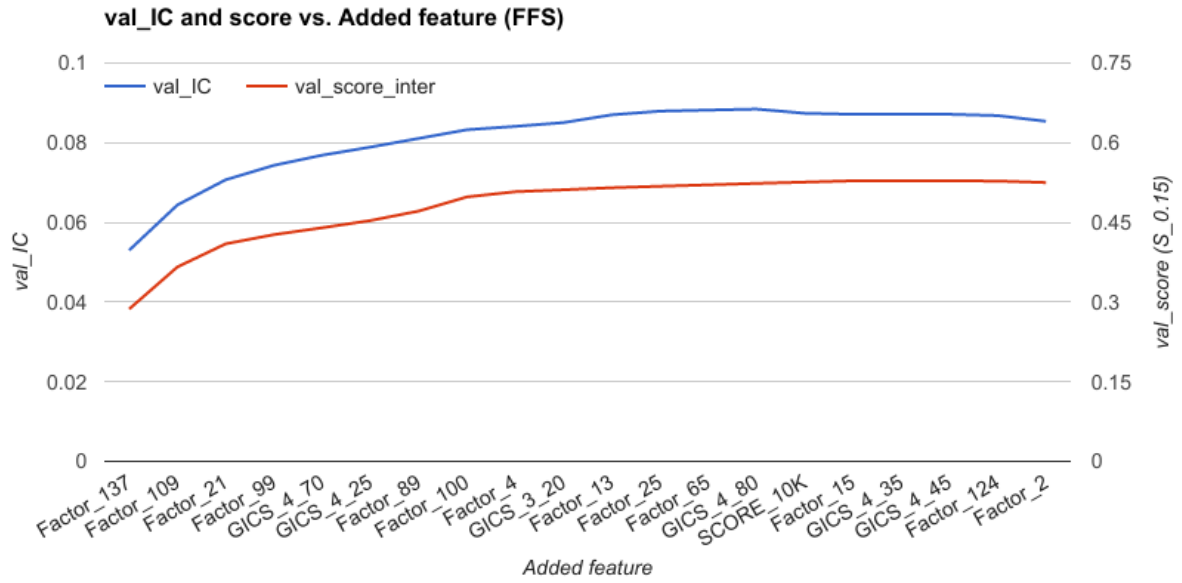

Figure 8.2: Validation IC and $S_{0.15}$ score as features are added

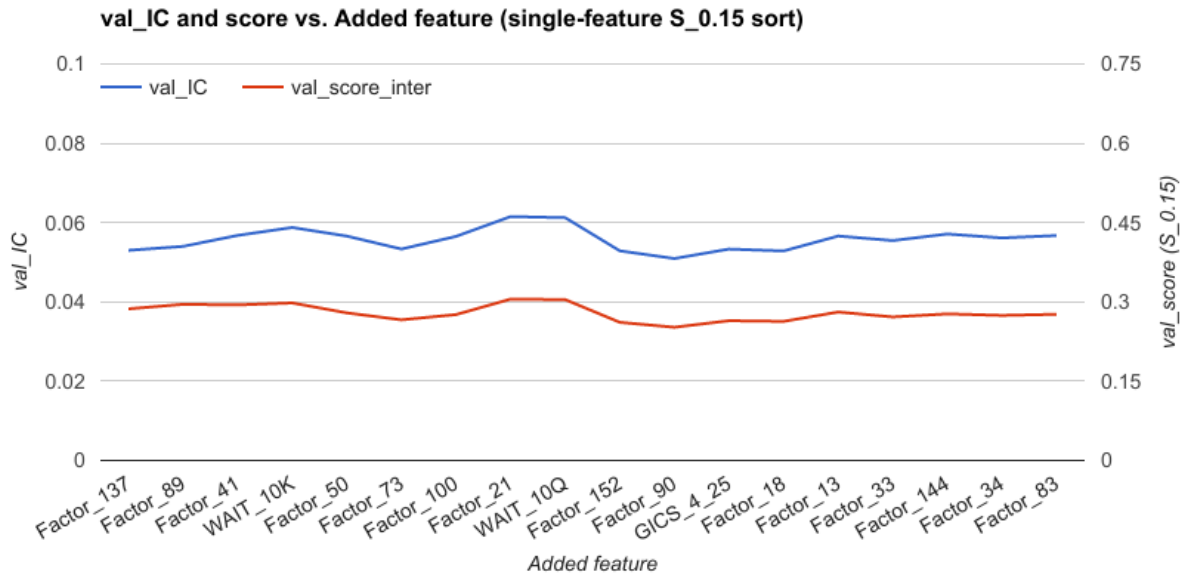

Figure 8.3: Validation IC and $S_{0.15}$ score as features are added. Adding features in order of decreasing $S_{0.15}$ score when measured by themselves. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

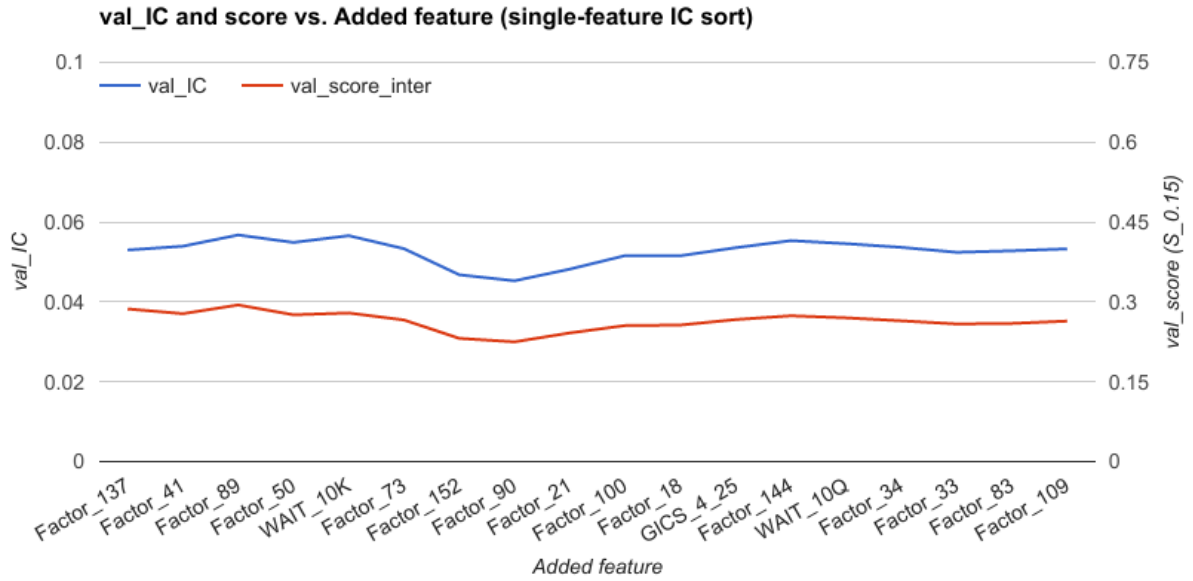

Figure 8.4: Validation IC and $S_{0.15}$ score as features are added. Adding features in order of decreasing IC when measured by themselves. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

Added feature	**val_IC**	**val_stddev**	**val_score**
Factor_137	0.048316831	0.034835805	0.261404068
Factor_161	0.075468010	0.055501466	0.367238304
Factor_109	0.080725316	0.049335831	0.404971426
Factor_21	0.074611566	0.028181393	0.418739377
Factor_99	0.077542541	0.029186859	0.432746805
Factor_89	0.078136779	0.027437070	0.440363332
Factor_51	0.079846925	0.029646559	0.444466765
Factor_73	0.080660631	0.028318941	0.452339111
Factor_25	0.081813837	0.030359566	0.453615183
Factor_144	0.079710216	0.024561580	0.456630927
Factor_5	0.078456798	0.021240133	0.458168282
Factor_13	0.079713798	0.021308227	0.465323818
Factor_2	**0.078922804**	**0.019170421**	**0.466528388**
Factor_83	0.078294249	0.018510245	0.464626048
Factor_35	0.077835879	0.018917692	0.460791754
Factor_65	0.077479519	0.019745168	0.456446091
Factor_39	0.077777398	0.021178170	0.454365168
Factor_100	0.074691324	0.014406007	0.454310186
Factor_70	0.073186209	0.011747715	0.452471364
Factor_52	0.072466643	0.010804385	0.450650912

Table 8.3: Only fundamental factor features

In Table 8.3 we use only fundamental factors. In Table 8.4, only GICS features. Finally, in Table 8.5 only simple sentiment features are used. One clear thing we see is that none of the feature sets perform as well on the validation set by themselves as they do when used together. For the GICS features, it is interesting to note that the validation score peaked after only three features were added. These three features are from the third and fourth levels of the GICS, which is interesting because they do not refer to specific industries or sub-industries because label sub-parts could exist at multiple points at the same depth of the GICS classification tree. Disappointingly, the simple sentiment features reached the validation score peak after only one feature was added, and instead of it being about the filing polarity, it is the length of time since the last 10K filing was released for a given stock. The sentiment feature SCORE_10K (which measures the polarity of 10K filings) was included in the feature set when using all of the available features.

Added feature	val_IC	val_stddev	val_score
GICS_4_25	0.0300052	0.0267206	0.1697886
GICS_3_20	0.0328544	0.0147723	0.1993925
GICS_4_80	**0.0333658**	**0.0142633**	**0.2031240**
GICS_4_35	0.0333658	0.0142633	0.2031240
GICS_4_45	0.0333658	0.0142633	0.2031240

Table 8.4: Only GICS features

Added feature	val_IC	val_stddev	val_score
WAIT_10K	**0.04119769**	**0.03519095**	**0.22246058**
WAIT_10Q	0.03223715	0.03741975	0.17200510
SCORE_10Q	0.02918234	0.03633766	0.15660998
SCORE_10K	0.01854263	0.04220288	0.09647424

Table 8.5: Only simple sentiment features

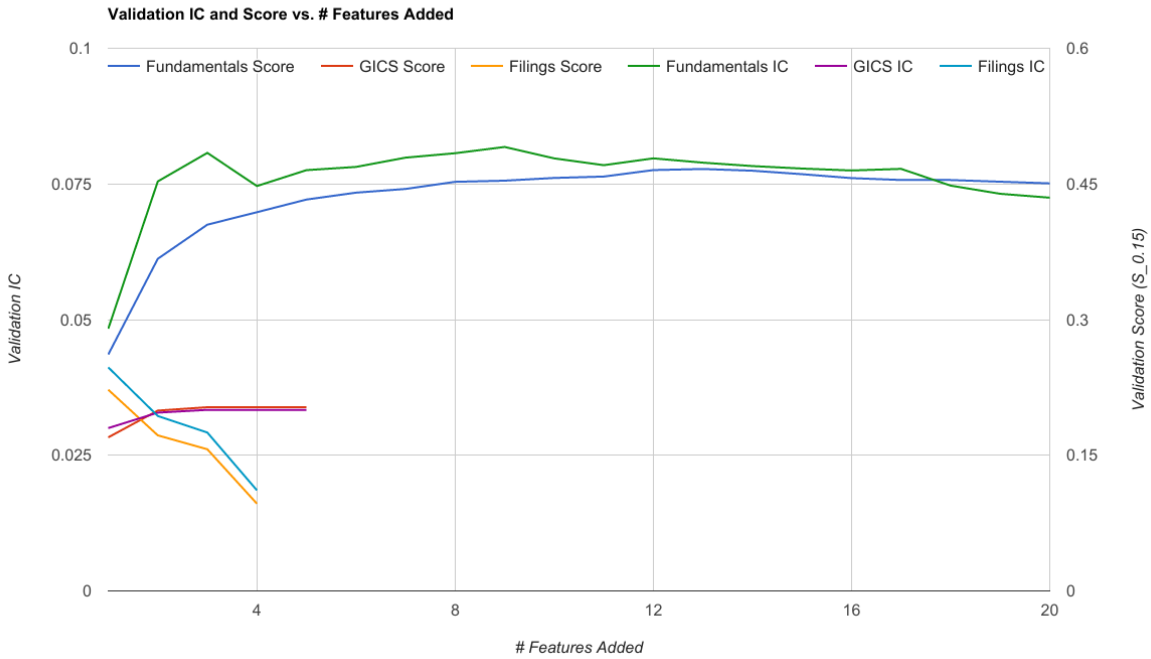

Figure 8.5: Validation IC and $S_{0.15}$ score as features are added when only accessing fundamentals, GICS, and filing features. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

8.3 Text Feature Extraction

In Section 2.3 we covered what autoencoders are and what they can be used for, and in Section 5.3.2 we looked at how features in the form of normalized word histograms are extracted from filings, and in this section we will use those features to train autoencoders and produce features. First we will cover the setup of the autoencoders used and then look at results making use of the new text-derived features.

8.3.1 Autoencoder Setup

The hyperparameter optimisation of a neural network can take a very long time as there are many factors to consider. Some important ones include the training duration, batch size, weight optimiser, graph structure and layer sizes, weight initialisation method, regularisation type and weight, and the activation functions.

For our experiments, we will restrict our autoencoders to simple fully connected feed-forward graphs. After some quick experimentation, we also found that good values for the following hyperparamaters are:

- the training duration: 30 epochs
- training batch size: 512
- optimiser: Adam (with default parameters provided by the Keras library (lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0))
- weight initialisation: glorot_uniform (chosen as default by Keras)
- graph structure: feedforward fully connected structures similar to Figure 2.1
- activation: *tanh*

To determine the precise structure, regularisation, and auxiliary loss weight, a few experiments in the next section will be performed.

8.3.2 Determining AE Size, Regularisation, and Auxiliary Loss Weight

For this section, we will select the AE layer sizes, regularisation, and auxiliary loss weight with three experiments. One with a small network with no regularisation and auxiliary loss weight of 0, one with a wider, deep network with moderate regularisation and auxiliary loss weight of 0, and the third experiment will use the best of the previous two but with a non-zero auxiliary loss weight. Since the AE training is not deterministic, each experiment will be run twice to create two sets of features, and feature selection will be run for both sets and their average results shown, for up to 5 features on each. The feature set used for selection will consist of only those generated by the AEs.

For the first experiment, one hidden layer of size 25 will be used. This will provide us with 25 features derived from 10K filings and 25 features from 10Q filings. In Figure 8.6, we see that the validation score reaches 0.24 after 5 features, slightly below the performance of the Factor_137 feature. The IC of 0.048 is about the same as for Factor_137. The performance also seems to be improving monotonically as features are selected.

Next, we will try a significantly larger/deeper autoencoder. The hidden layer sizes are 5000-500-50-25-50-500-5000, with dropouts of 0.4 for the first two and last two layers, and 0.2 for the layers just before and after the encoding layer. As we can see in Figure 8.7, the performance is at its best after the first feature, and overall is worse than the smaller AE. In earlier experiments with neural network, what we found was they they very easily over-fit to the data. This may be why the smaller AE with no regularisation performs better than the larger, deep one.

Since it appears that the smaller AE produces features at least as good as the deeper one, we will now use a small AE to determine if an auxiliary loss can be used for improvement. This will make it so that the autoencoder will simultaneously be trained to reproduce its main input (the word histograms), but also to use the encoding to predict relative returns. The weight of the auxiliary loss will be set of 0.25 and the main loss to 0.75. The auxiliary output layer is directly connected to the hidden layer with linear activations, thus explicitly encouraging the hidden representation to contain information about how well the stock will perform in the future. To avoid an information leak, only stock returns from before the year 2008 were used

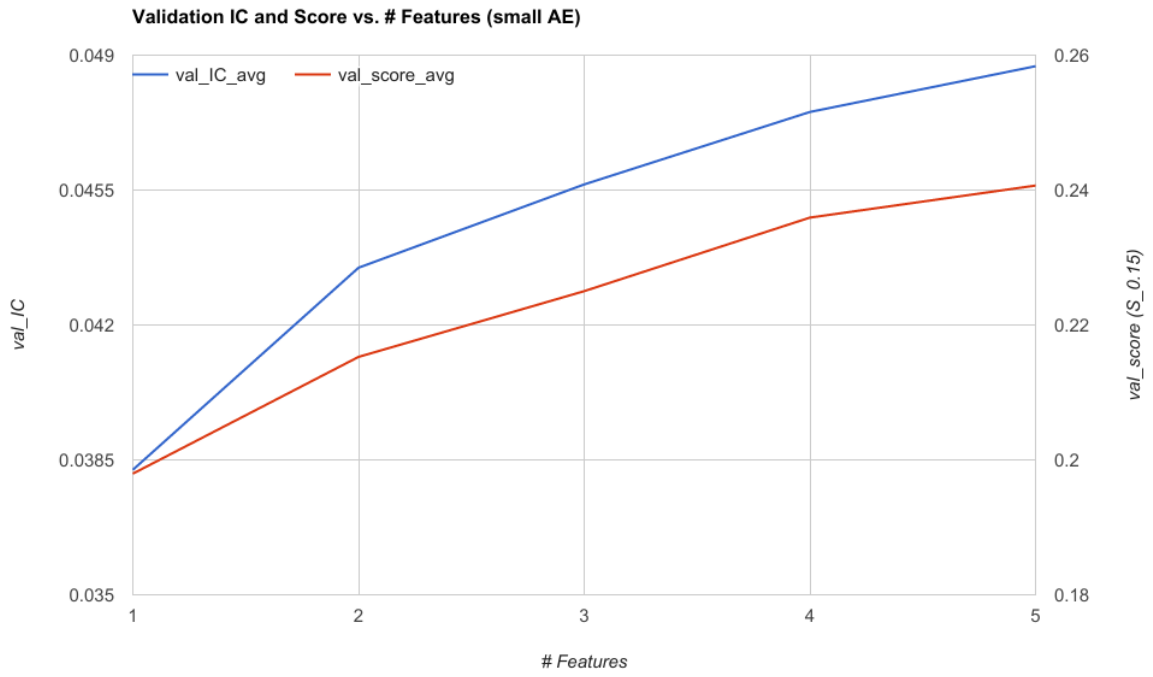

Figure 8.6: Validation IC and $S_{0.15}$ score as features produced by small AE are selected. Values are the average of two runs with feature sets generated by a network with the same hyperparameters. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

Figure 8.7: Validation IC and $S_{0.15}$ score as features produced by larger AE are selected. Values are the average of two runs with feature sets generated by a network with the same hyperparameters. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

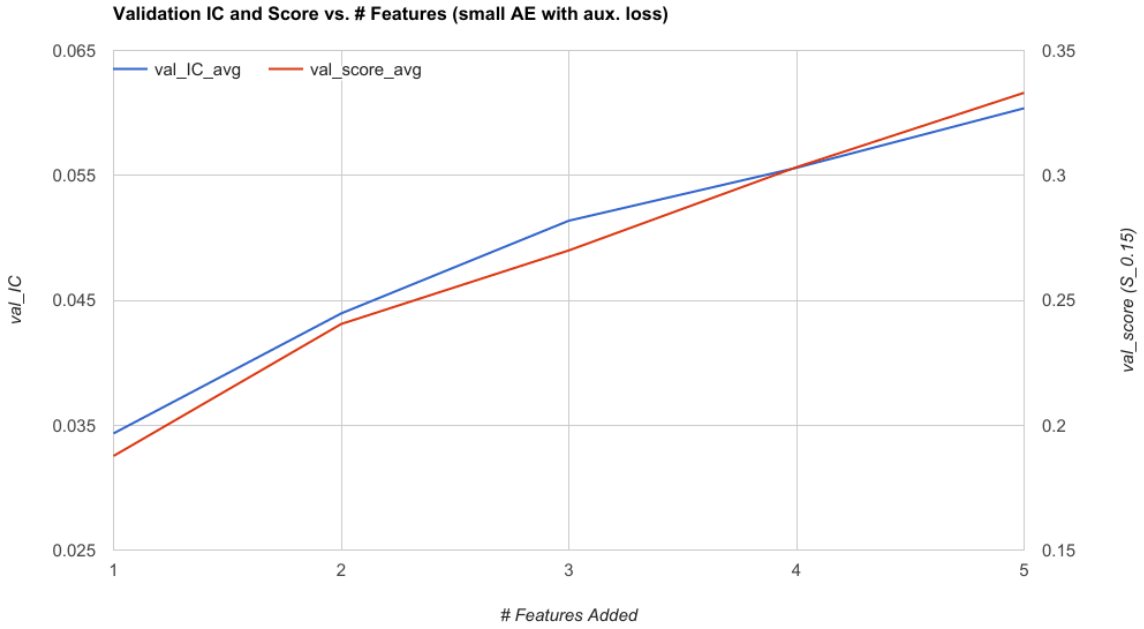

Figure 8.8: Validation IC and $S_{0.15}$ score as features are added. Values are the average of two runs with feature sets generated by a network with the same hyperparameters. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

(since this is where samples begin to be included for validation). Instead of predicting only the 120 day return, the 5, 10, 20, 60, and 120 day return values were concatenated into a target vector. The process of pairing a set of returns values with each filing was non-trivial, and what we ended up doing was finding all of the return values following the release of the filing, but before the next one, and averaging them. The intent is that this provided a summary of the effect the filing had, however, more accurate methods of filing-return pairing may exist. In Figure 8.8 we can see the performance validation performance of the model as features are added. The performance is significantly better than for the small AE with the aux. loss. After five features, it reaches a $S_{0.15}$ score and IC of approximately 0.33 and 0.06 respectively. This would place the feature set between the top two shown in Table 8.3.

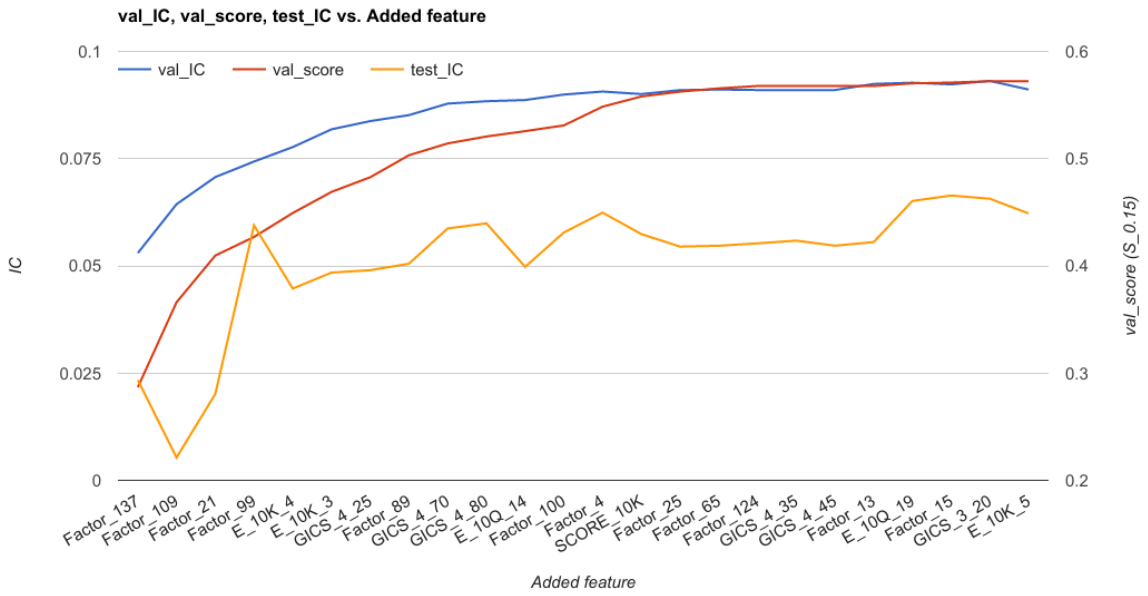

Figure 8.9: Validation IC, $S_{0.15}$ score, and test IC as features are added. Original set of features and those produced by small AE with aux. loss are used. The left vertical axis is the validation IC and the right vertical axis is the validation $S_{0.15}$ score.

8.3.3 Results

Now, we will perform feature selection with all of the original set features as well as AE produced features. For AE features, a set generated by the AE with a single hidden layer and auxiliary loss is used. Results on the 2015 test data will also be included. More test results on previously seen models will be shown in the next section of this chapter.

8.4 Test Results

Before this point in the experiments, the test results for the year 2015 were never used to guide model development. This ensures that accidental over-fitting to the test data is not performed. Now, we will finally see how well our methods were able to encourage generalizability.

Table 8.6 contains the validation and test values for many of the models encountered so far. Perhaps the most important result that can be seen is the test IC for the baseline models does go above 0.01, which is essentially random. For the models that made use of feature

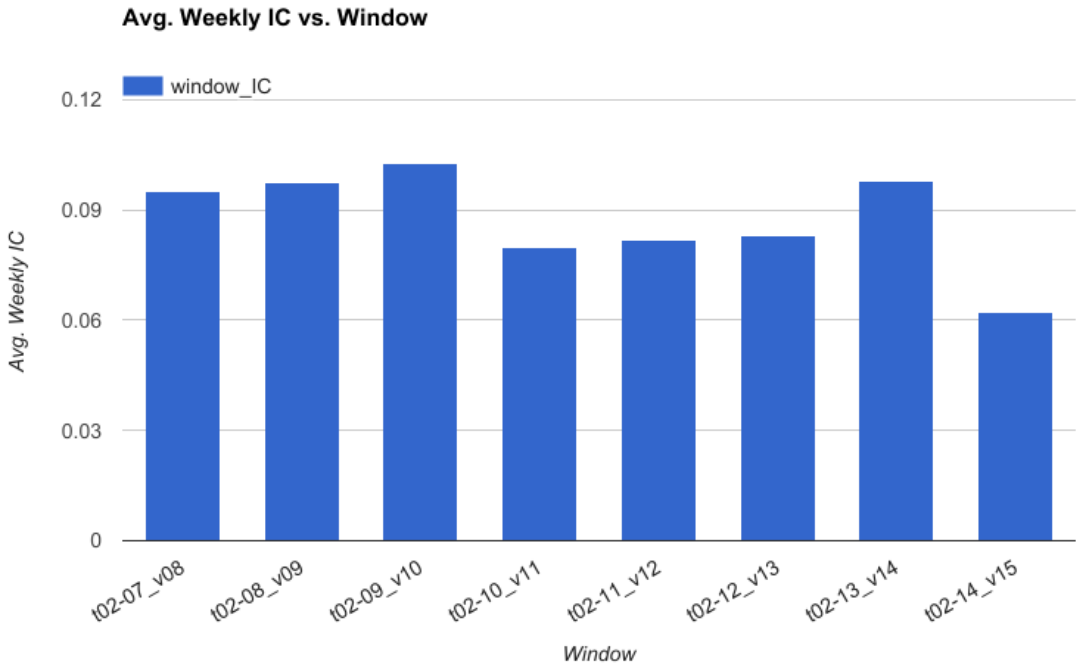

Figure 8.10: test IC for each window for LSLR model trained with features shown in 8.9. The out-of-sample test window of t02-14_v15 performs slightly worse than previous years.

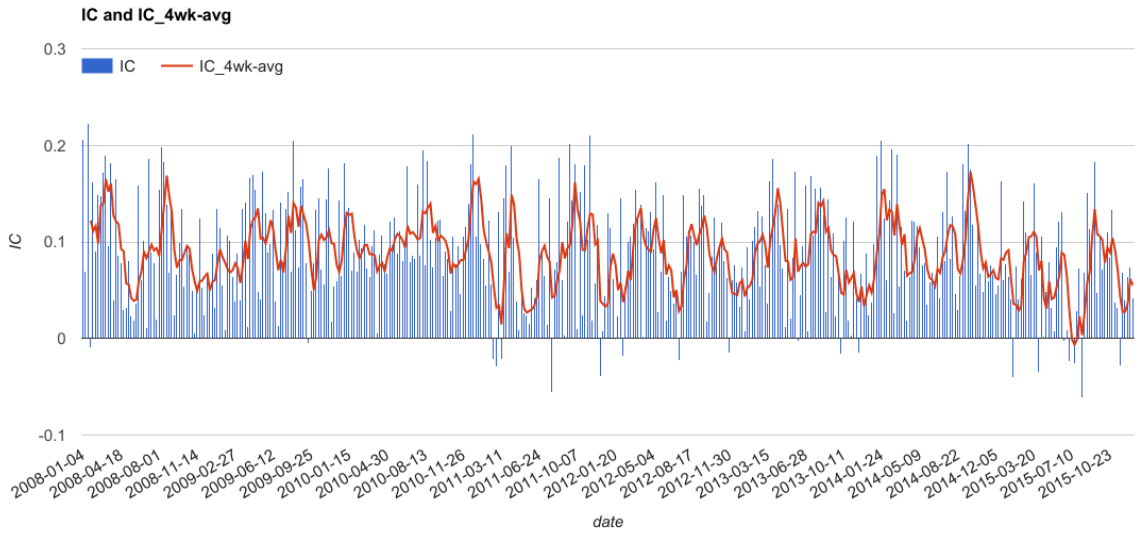

Figure 8.11: Weekly IC and its 4-week average for LSLR model trained with features shown in 8.9. Weeks in 2015 are out-of-sample.

model	val_IC	val_stddev	val_score	test_IC	test_stddev
Baseline Models					
LSLR	0.02307	0.06963	0.10505	-0.00137	0.11128
ALR	0.02787	0.04327	0.14419	0.01560	0.12737
SGD	0.05269	0.07635	0.23278	-0.02716	0.07125
GB	0.00110	0.06184	0.00520	-0.01796	0.07626
RF	0.01793	0.03878	0.09500	-0.07826	0.09173
NN	0.03196	0.05252	0.15782	-0.00190	0.07682
Using Feature Selection (FS)					
FS_Fundamentals	0.07892	0.01917	0.46653	**0.06635**	0.06400
FS_GICS	0.03337	0.01426	0.20312	0.00732	0.04173
FS_Filiings	0.04120	0.03519	0.22246	0.00181	**0.03893**
FS_all	0.08714	0.01494	0.52835	0.06469	0.04829
FS_all+AE	**0.09111**	**0.00923**	**0.57223**	0.06222	0.05427

Table 8.6: Final results for models, also showing 2015 test IC and standard deviation. Note that for the validation values, the IC is the average of window IC averages, and the standard deviation is the of the window ICs. For the test values, the IC is the average across the weeks in the year, and the standard deviation is of the ICs in that year.

selection, the best test IC was attained when only using the fundamental factors, as in Table 8.3. Adding the GICS and filing (sentiment + wait times) features actually decreased the test IC, even though when used alone, each of the feature types had a small positive test score. Adding the autoencoder features decreased the test IC by a small amount again. As we can see from Figure 8.9, the validation score is continuing to increase even after 24 features are added, and if this process were stopped after 22 features instead of 24), the test IC would instead be 0.066. This may be a result of the randomness inherent in the data as discuss in Chapter 4, however it could also indicate that over-fitting is beginning to occur after many features are added.

8.5 Sector-Level Models

All of the models so far tested have been 'universal' models, in that they are trained to make predictions for stocks in all sectors. Another approach is to create sector-level models, where each sector, or similar sectors, have their own predictive model. This may be a desirable approach for a few reasons. Having a portfolio which makes sure to have strategies for many

Sector Group	# Test Samples
10	2741
15	1748
2010+2020	3091
2030	768
2510+2520	1650
2530+2540+2550	3790
30	2032
3510	1790
3520	1578
4010	1021
4020+4030	3111
4040	1903
4510	2536
4520+4530	2032
50	407
55	1691

Table 8.7: The sector groups to be used for training each model. The number of samples corresponding to each group in the test data is also shown.

sectors ensures that if one sector ends up performing in a negative unexpected way, there are many other sectors whose performance can outweigh the loss. It is conceivable that the patterns which appear in one section and led to a loss are unrelated to patterns appearing in other sectors. This approach also has potential drawbacks. It limits shared knowledge – patterns that appear in one sector may help predict another. It also reduces the training data for each model. In this section, we will quickly look at how sector-level models can be created and how they perform compared to a model used in the industry.

Sector Grouping. The GICS, as discussed in Section 5.2 is used to specify the sector of a stock at four levels of increasing specificity. At the top level we have stock in 9 different sectors: 10, 20, 25, 30, 35, 40, 45, 50, 55. However, the number of stocks in each sector is not balanced and some sectors may contain a more diverse set of stocks. To deal with this, we decided to split the sectors intro groups as shown in Table 8.7. For each of these sector groups (which we will also refer to simply as sectors), we will train a model.

Model Training. Using the 16 predefined sectors, we will create 16 models using a method similar to that in Section 8.2.1. That is, using the training and validation data associated with each sector (2015 test year), we will use least-squares linear regression and forward feature

selection with S_ϵ as the feature set evaluation function. However, instead of $\epsilon = 0.15$, 0.10 is used (this was chosen before the tuning of ϵ took place). For each of the sector-level models, feature selection was performed until the validation score started to decrease or levelled off, up to a maximum of 15 features.

Results.

After performing the feature selection for each sector model, we can see in Table 8.8 that the number of features chosen for each sector's model varies from 1 to 15. In Figure 8.12, we can also see the 2015 test IC for our models (SLR-FS). The performance of the model provided by Highstreet is also displayed in the figure. The most salient aspect of the results is the large variation in sector performances for each of the models as well as between the models. However, the performances of the two models are not completely independent; the correlation between sector performances is approximately 0.6, indicating that some sectors may be more predictable in general than others. The size of the sectors may also influence the model performances; the correlation between the number of test samples for each sector and the corresponding sector model performance is about 0.35 for both our models and the Highstreet models.

Accurately comparing the performance of our model and the Highstreet model is not easy. Taking the average of the sector scores is unrealistic since the sector sizes vary considerably. Using a sector-size weighted average is also not very accurate since the size of sectors is not constant over time. To get an approximation, we will use a weighted average, where the weight is the number of test samples for each sector. Doing this reveals that the overall performances are quite close: the IC for the Highstreet model is 0.0625, and the IC for our model is 0.0618.

Given the disparate sector-level performances of the two models, being able to combine them would be beneficial. There are some sectors where each model has a clear benefit, which suggests that utilizing our algorithm for some sectors and not others may be beneficial, however more extensive testing is required to confirm that the sector performances are similar across years.

Sector Group	# Features Selected
10	6
15	11
2010+2020	1
2030	13
2510+2520	10
2530+2540+2550	12
30	1
3510	15
3520	12
4010	11
4020+4030	15
4040	5
4510	15
4520+4530	12
50	12
55	6

Table 8.8: Number of features selected for each sector model.

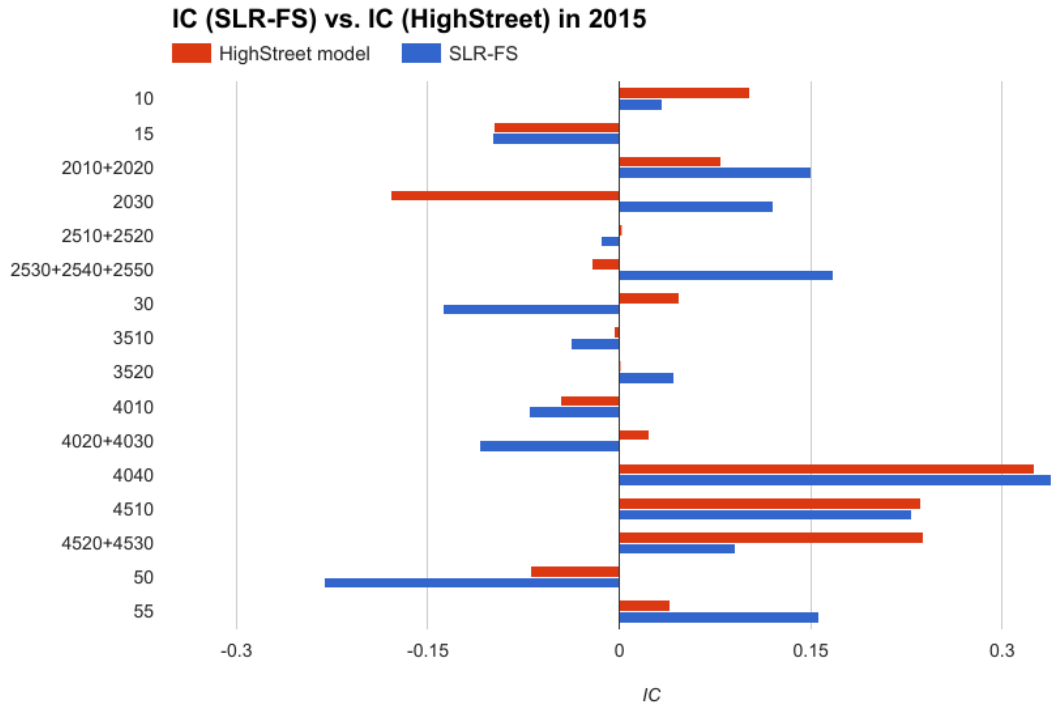

Figure 8.12: Comparison of sector-level model developed in this thesis (Sector level with Linear Regression and Feature Selection – SLRFS) and the Highstreet model.

Sector	Highstreet model	SLR-FS
10	**0.1020**	0.0341
15	**-0.0980**	-0.0998
2010+2020	0.0795	**0.1499**
2030	-0.1795	**0.1202**
2510+2520	**0.0018**	-0.0143
2530+2540+2550	-0.0216	**0.1673**
30	**0.0471**	-0.1385
3510	**-0.0040**	-0.0379
3520	0.0009	**0.0434**
4010	**-0.0461**	-0.0705
4020+4030	**0.0239**	-0.1096
4040	0.3251	**0.3382**
4510	**0.2359**	0.2291
4520+4530	**0.2379**	0.0907
50	**-0.0695**	-0.2309
55	0.0400	**0.1565**

Table 8.9: Comparison of sector-level model developed in this thesis (SLRFS) and the Highstreet model. The better model for each sector is bolded.

Chapter 9

Conclusions and Future Work

In this chapter, we will first summarize the work done in this thesis by looking at the main problems faced, how they were overcome, and draw conclusions based on our results. Next, we will discuss potential areas for future work that may improve obtained results.

9.1 Conclusions

In this thesis, we looked at the problem of forecasting stock performance. Although a substantial volume of research exists on the topic, very little is aimed at long term forecasting while making use of machine learning methods and textual data sources. We prepared over ten years worth of stock data and proposed a solution which combines features from textual yearly and quarterly filings with fundamental factors for long term stock performance forecasting. Additionally, we developed a new method of extracting features from text for the purpose of performance forecasting and applied feature selection aided by a novel evaluation function.

Problems Overcome. To produce effective models, there were two main problems we were faced with and had to overcome which were discussed in Chapter 4. The first was that of market efficiency, which places theoretical limits on how patterns can be found in the stock markets for the purpose of forecasting. This property can become a concrete problem by patterns being exhibited in the data which are useless or even detrimental for predicting future values. The way we tried to deal with this was by carefully splitting our data into training, validation, and testing data with expanding windows so as to make maximum use of it while trying to

avoid accidental overfitting. The second way we dealt with this was by using a tailored model performance metric, S_ϵ, which aimed to ensure good test performance of models by not only maximising model validation IC, but also minimising the variation across validation years of this value. The third way we dealt with market efficiency was by performing feature selection using the S_ϵ so as to remove those features which performed poorly or unreliably. The second set of problems came from putting together a dataset to use for experimentation and testing. Due to the large volume of the data, care had to be taken when cleaning and preparing it, and the inevitable mistakes along the way required reprocessing of the data. Additionally, outliers and missing values in the data were dealt with in a number of ways as discussed in Section 5.1. Using expert knowledge, we determined how to deal with the various problems in the data, and ended up using mean substitution and feature deletion.

Results. In Chapter 8 we obtain results when applying feature selection and compare the results to the baseline models obtained in Chapter 7. Our results show that using only a least-squares linear regression model, forward feature selection, and the S_ϵ metric combined with fundamentals, GICS features, and simple filing features, we significantly improved upon the baseline performances. While the maximum baseline IC was 0.05269, we achieved 0.08714. The highest test IC obtained by a baseline mode was 0.01560, whereas we achieved 0.06469. Our experiments also showed that while our novel technique for extracting features from text with autoencoders as described in Sections 5.3.2 and 8.3 achieved a validation IC of 0.09111, it did not improve upon the test IC, achieving 0.06222. In Section 8.5, we explored the idea of sector-level models and compared our set of models to Highstreet's own models. Although it is not easy to compare the performances, both models achieve an approximate test IC of 0.062. Since our method produces models which perform comparably to hand-made models while requiring no expert knowledge beyond data preparation, this makes the model an attractive aid for constructing investment portfolios.

9.2 Future Works

There are many steps in our model creation process which are ready for further exploration and improvement. In this section, we will briefly discuss several of these areas.

Model Updating Frequency. In all of the models created in Chapter 7 and 8, they are trained once and then used for predicting stock performances over the span of a year. Since we use a return duration of 120 trading days, there is a necessary wait of half a year before data can be used to train models, which means that models end up making predictions using data which is over a year old. One way to make use of data as soon as it become available is to completely retrain the model every week (or less). A faster way to improve model performance may be through updating using incremental machine learning algorithms, which can update model parameters without re-training on all data [67].

More Complex Metrics. In our experiments, we measured model performance mainly with the IC, which determines how well the model is able to order the stocks by relative return and S_ϵ, which applies a weight to both increasing the IC and minimising the variation in IC across years. When portfolios are constructed in practice, there are many other factors that may be considered. One of these is the turnover, which measures how often stocks must be bought and sold according to the portfolio. Since the action of buying and selling requires money, a model which helps create a portfolio with low turnover is preferred.

Model Evaluation Framework. The framework we use to evaluate our models uses a set of expanding windows start start in 2002 and end in 2015. This setup was chosen because we decided it could provide accurate estimates of model performance. However, as we have seen in the experiments, the validation S_ϵ score does not perfectly correlate with the test IC, indicating that there may be room for a more accurate model evaluation framework. One potential way to do this is by expanding the windows by months or weeks instead of years, allowing for a higher resolution in the validation performance estimates.

Explore More Algorithms. Although many different models were considered in this thesis, including various linear regression methods, gradient boosting, random forests, and neural networks, there is always more room to explore.

Utilize Time Series Information. Similar to the idea of updating model frequency, another area for exploration includes utilizing the time series aspect of the data. Our current models are not aware that the samples occur in any temporal order, and thus are not able to spot patterns in stock performance that depend on knowing the order of samples. One type of model that is often used to find and make use of these type of patterns are recurrent neural networks [33].

Improve Feature Extraction: In this thesis, a few methods for extracting features from filings with textual data were explored. The problem of extracting features from text and determining text sentiment in particular are well studied, and other natural language processing methods may perform better. Our approach of using autoencoders to extract features may also benefit from further exploration. In particular, when using the auxiliary loss, a more accurate method for estimating the financial effect corresponding to a given filing would be useful. A more thorough hyper-parameter search for the neural network implementation may also be beneficial.

Bibliography

[1] Gics. Available at https://www.msci.com/gics.

[2] Form 10-k, Jun 2009. Available at https://www.sec.gov/answers/form10k.htm.

[3] Filings forms, Jan 2014. Available at https://www.sec.gov/edgar.shtml.

[4] Cristina Abad, Sten A Thore, and Joaquina Laffarga. Fundamental analysis of stocks by two-stage dea. *Managerial and Decision Economics*, 25(5):231–241, 2004.

[5] Alan C Acock. Working with missing values. *Journal of Marriage and family*, 67(4):1012–1028, 2005.

[6] David W Aha and Richard L Bankert. A comparative evaluation of sequential feature selection algorithms. In *Learning from Data*. Springer, 1996.

[7] Qasem A Al-Radaideh, Aa Assaf, and Eman Alnagi. Predicting stock prices using data mining techniques. In *The International Arab Conference on Information Technology (ACIT2013)*, 2013.

[8] David H Bailey, Jonathan M Borwein, Marcos López de Prado, and Qiji Jim Zhu. Pseudomathematics and financial charlatanism: The effects of backtest over fitting on out-of-sample performance. *Notices of the AMS*, 61(5):458–471, 2014.

[9] John Barnard and Xiao-Li Meng. Applications of multiple imputation in medical studies: from aids to nhanes. *Statistical methods in medical research*, 8(1):17–36, 1999.

[10] Messod D Beneish, Charles MC Lee, and Robin L Tarpley. Contextual fundamental analysis through the prediction of extreme returns. *Review of Accounting Studies*, 6(2-3):165–189, 2001.

[11] Arnold Bernhard. *How to use the value line investment survey: A subscriber's guide*. Value Line, 1982.

[12] Avrim Blum. Random projection, margins, kernels, and feature-selection. In *Subspace, Latent Structure and Feature Selection*. Springer, 2006.

[13] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.

[14] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[15] Franois Chollet. Keras. https://github.com/fchollet/keras, 2015.

[16] Michael Cooper. Filter rules based on price and volume in individual security overreaction. *Review of Financial Studies*, 12(4):901–935, 1999.

[17] KJ Martijn Cremers. Stock return predictability: A bayesian model selection perspective. *Review of Financial Studies*, 15(4):1223–1249, 2002.

[18] Harris Drucker. Improving regressors using boosting techniques. In *ICML*, volume 97, pages 107–115, 1997.

[19] Cheoljun Eom, Gabjin Oh, and Woo-Sung Jung. Relationship between efficiency and predictability in stock price change. *Physica A: Statistical Mechanics and its Applications*, 387(22):5511–5517, 2008.

[20] Joseph E Finnerty. Insiders and market efficiency. *The Journal of Finance*, 31(4):1141–1148, 1976.

[21] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[22] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[23] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[24] Bernd Fritzke. Some competitive learning methods. *Artificial Intelligence Institute, Dresden University of Technology*, 1997.

[25] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, and Wai Lam. News sensitive stock trend prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 481–493. Springer, 2002.

[26] Isabelle Guyon and André Elisseeff. An introduction to feature extraction. In *Feature extraction*, pages 1–25. Springer, 2006.

[27] Michael Hagenau, Michael Liebmann, and Dirk Neumann. Automated news reading: Stock price prediction based on financial news using context-capturing features. *Decision Support Systems*, 55(3):685–697, 2013.

[28] David Harris and Sarah Harris. *Digital design and computer architecture*. Elsevier, 2012.

[29] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[30] Clayton J Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[31] Michael C Jensen. Some anomalous evidence regarding market efficiency. *Journal of financial economics*, 6(2-3):95–101, 1978.

[32] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[33] Ken-ichi Kamijo and Tetsuji Tanigawa. Stock price pattern recognition-a recurrent neural network approach. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 215–221. IEEE, 1990.

[34] Mahesh Khadka, Benjamin Popp, Kayikkalthop M George, and Nohpill Park. A new approach for time series forecasting based on genetic algorithm. In *CAINE*, pages 226–231, 2010.

[35] Antonina Kloptchenko, Tomas Eklund, Jonas Karlsson, Barbro Back, Hannu Vanharanta, and Ari Visa. Combining data and text mining techniques for analysing financial reports. *Intelligent systems in accounting, finance and management*, 12(1):29–41, 2004.

[36] Efthymios Kouloumpis, Theresa Wilson, and Johanna D Moore. Twitter sentiment analysis: The good the bad and the omg! *ICWSM*, 11(538-541):164, 2011.

[37] Victor Lavrenko, Matt Schmill, Dawn Lawrie, Paul Ogilvie, David Jensen, and James Allan. Language models for financial news recommendation. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 389–396. ACM, 2000.

[38] Kristina Levišauskait. Investment analysis and portfolio management. *Leonardo da Vinci programme project*, 2010. Retrieved April 2017, from http://www.bcci.bg/projects/latvia/pdf/8_IAPM_final.pdf.

[39] Xiaodong Li, Haoran Xie, Li Chen, Jianping Wang, and Xiaotie Deng. News impact on stock price return via sentiment analysis. *Knowledge-Based Systems*, 69:14–23, 2014.

[40] Richard Loth. Profitability indicator ratios: Profit margin analysis, May 2007. Available at http://www.investopedia.com/university/ratios/profitability-indicator/ratio1.asp.

[41] Tim Loughran and Bill McDonald. When is a liability not a liability? textual analysis, dictionaries, and 10-ks. *The Journal of Finance*, 66(1):35–65, 2011.

[42] Andrew L Maas, Quoc V Le, Tyler M O'Neil, Oriol Vinyals, Patrick Nguyen, and Andrew Y Ng. Recurrent neural networks for noise reduction in robust asr. In *Interspeech*, pages 22–25, 2012.

[43] Burton Gordon Malkiel. *A random walk down Wall Street: including a life-cycle guide to personal investing*. WW Norton & Company, 1999.

[44] Andrew McCallum. Information extraction: Distilling structured data from unstructured text. *Queue*, 3(9):48–57, 2005.

[45] M-A Mittermayer. Forecasting intraday stock price trends with text mining techniques. In *system sciences, 2004. proceedings of the 37th annual hawaii international conference on*, pages 10–pp. IEEE, 2004.

[46] John J Murphy. *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.

[47] Karl Nygren. Stock prediction–a neural network approach. *Master's Thesis, Royal Institute of Technology, KTH, Stockholm*, 2004.

[48] Chong Oh and Olivia Sheng. Investigating predictive power of stock micro blog sentiment in forecasting future stock price directional movement. 2011. In: ICIS 2011 Proceedings (2011).

[49] Ping-Feng Pai and Chih-Sheng Lin. A hybrid arima and support vector machines model in stock price forecasting. *Omega*, 33(6):497–505, 2005.

[50] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[51] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[52] Elvis Picardo. Portfolio investment, Jan 2016. Available at http://www.investopedia.com/terms/p/portfolio-investment.asp.

[53] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[54] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[55] Sam Ro. The makeup of the sp 500 is constantly changing, Jun 2015. Available at http://www.businessinsider.com/sp-500-index-constituent-turnover-2015-6.

[56] JB Satinover and D Sornette. Anomalous returns in a neural network equity-ranking predictor. *arXiv preprint arXiv:0806.2606*, 2008.

[57] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[58] Robert P Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Transactions on Information Systems (TOIS)*, 27(2):12, 2009.

[59] Daniel Scinto and Jo Hardin. Stock ranking and portfolio selection: Revising and developing z-scores, 2009. Volatility Trader@ BFAM Partners.

[60] Young-Woo Seo, Joseph Giampapa, and Katia Sycara. Text classification for intelligent portfolio management. Technical report, DTIC Document, 2002.

[61] Investopedia Staff. Universe of securities, Mar 2005. Available at http://www.investopedia.com/terms/u/universeofsecurities.asp.

[62] Investopedia Staff. Absolute return, Jul 2015. Available at http://www.investopedia.com/terms/a/absolutereturn.asp.

[63] Investopedia Staff. Market efficiency, Mar 2016. Available at http://www.investopedia.com/terms/m/marketefficiency.asp.

[64] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

[65] Ryan Sullivan, Allan Timmermann, and Halbert White. Data-snooping, technical trading rule performance, and the bootstrap. *The journal of Finance*, 54(5):1647–1691, 1999.

[66] Ryan Sullivan, Allan Timmermann, and Halbert White. Data-snooping, technical trading rule performance, and the bootstrap. *The journal of Finance*, 54(5):1647–1691, 1999.

[67] N Syed, H Liu, and K Sung. Incremental learning with support vector machines. In *International Joint Conference on Artificial Intelligence*, 1999.

[68] Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede. Lexicon-based methods for sentiment analysis. *Computational linguistics*, 37(2):267–307, 2011.

[69] James D Thomas and Katia Sycara. Integrating genetic algorithms and text learning for financial prediction. *Data Mining with Evolutionary Algorithms*, pages 72–75, 2000.

[70] Philip M Tsang, Paul Kwok, Steven O Choy, Reggie Kwan, Sin Chun Ng, Jacky Mak, Jonathan Tsang, Kai Koong, and Tak-Lam Wong. Design and implementation of nn5 for hong kong stock price forecasting. *Engineering Applications of Artificial Intelligence*, 20(4):453–461, 2007.

[71] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[72] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005.

[73] Robert J Yan and Charles X Ling. Machine learning for stock selection. In *Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1038–1042. ACM, 2007.

[74] Youngohc Yoon and George Swales. Predicting stock price performance: A neural network approach. In *System Sciences, 1991. Proceedings of the Twenty-Fourth Annual Hawaii International Conference on*, volume 4, pages 156–162. IEEE, 1991.

Curriculum Vitae

Name:	Tanner Bohn
Post-Secondary Education and Degrees:	University of Saskatchewan Saskatoon, SK 2011-2015 B.Sc. University of Western Ontario London, ON 2015 - 2017 M.Sc.
Related Work Experience:	Teaching Assistant and Student Researcher The University of Western Ontario 2015 - 2017