Using social media engagement for stock market prediction with machine learning

David Baxter
Supervisor: Dr. Ke Chen
Word count: 14,982

April 29, 2022

Abstract

The Efficient Market Hypothesis states that stock market prices reflect all available information, and thus cannot be predicted; however, this hypothesis long been in question. A large volume of research has attempted to predict the stock market, but with varying levels of success and practicability to investors. Following advances in sentiment analysis, time series analysis, and increased social media usage, I utilise 9M collected tweets and stock price data from 4,612 stocks in the tasks of stock price and trend prediction. A range of machine learning (ML) models are pitted against traditional models such as ARIMA and exponential smoothing. The results are largely consistent with the EMH, however impressive trend prediction accuracies of 58.1% and 73.9% are achieved by the ML models at the monthly and quarterly horizons. The findings highlight the potential power of social media data, and reaffirm the conventional wisdom that simpler, statistical models tend to outperform complex machine learning models.

Acknowledgements

I would like to thank my supervisor, Dr. Ke Chen, for his continued support and guidance throughout this project. I would also like to thank Cameron Sabuda, for kindly lending his GPU for the purpose of running the sentiment analysis of 9M tweets, which made this work feasible with the limited time and resources allotted to it.

Contents

Chapter 1

Introduction

Predicting the stock market is a problem with both practical and theoretical benefit. Practically, investors could make better decisions with lower risk, leading to financial gain. Theoretically, this has long been considered a challenging problem, and many researchers have (without widespread consensus) attempted to address the question of whether prediction is even possible. At the heart of the issue is the **Efficient Market Hypothesis** (EMH), which asserts that stock prices reflect all available information that is relevant to them, and thus react only to *new* information (Fama et al. 1969; Malkiel 1989) . An implication of this is that it is impossible to utilise present information to predict future prices, because market prices would already reflect all available information – private or otherwise.

Despite these theories, a great number of researchers and financial analysts have attempted to predict the stock market. Approaches to this effect are usually divided into two categories: technical analysis, which makes use of past stock prices, trading volume, and other market data; and fundamental analysis, which aims to determine the intrinsic value of companies in question (which may differ from the stock price) using, for instance, the company's financial statements and considerations of the economy as a whole.

Since the early 2000s, concomitantly with the growth of the World Wide Web, a third category of *sentiment analysis* has gained interest as an additional approach to predicting stock prices. Sentiment analysis is a natural language technique whereby the sentiment of text is determined, often by classifying text as positive or negative (and sometimes additionally 'neutral'). When applied to text written by traders, *market sentiment* can be ascertained. However, it remains an open question as to whether this has predictive value for stock prices. Early researchers utilised forums and message boards frequented by traders, and answered this question in the negative (Tumarkin and Whitelaw 2001; Antweiler and Frank 2004). However, the rise of social media websites such as Twitter have led to digital communication on an unprecedented scale , and more optimistic research has emerged supporting the predictive power of social media for book sales (Gruhl et al. 2005), box-office revenues (Asur and Huberman 2010), elections (Tumasjan et al. 2010), and stock market prices (Gilbert and Karahalios 2010; Bollen, Mao, and Zeng 2011).

Machine learning (ML) has seen great advancements in the 21st century, particularly in the fields of computer vision (Taigman et al. 2014), artificial intelligence (Silver et al. 2016), and, pertinently, natural language processing (Hannun et al. 2014; Devlin et al. 2018): a number of ML models now outperform humans in a range of language understanding tasks (Wang et al. 2019). Sentiment analysis has also improved dramatically in the last decade: binary classification ('positive' or 'negative') accuracy on a corpus of 215,154 labelled English phrases rose from 85.4% in 2013 (Socher et al. 2013) to 97.5% accuracy in 2019 (Jiang et al. 2019).

Machine learning has also more recently been leveraged for the tasks of time series forecasting (Fan et al. 2019; Lim, Arik, et al. 2019) and classification (Lines, S. Taylor, and Bagnall 2016; Ismail Fawaz et al. 2020). Traditionally, statistical models have dominated in these tasks (Makridakis and Hibon 2000). However, with increased computing power and data, ML approaches have gained some traction.

Building on these advances, I apply ML techniques and state-of-the-art sentiment analysis to the task of short–medium term stock market prediction using tweets and stock prices. With this, I aim to produce practicable models to guide investments, test the EMH, and investigate whether ML can now exceed the performance of dominant statistical models.

1.1 Structure

The remainder of this report shall be structured as follows. Chapter 2 introduces the necessary theory and provides an overview of the relevant literature. Chapter 3 then defines the problem to be solved mathematically. Chapter 4 describes the data collected to conduct this project, and Chapter 5 describes and explains the methods applied with the data to the tasks of stock prediction. Chapter 6 details the conducted experiments and discusses the results. Finally, Chapter 7 summarises the work, draws conclusions, and suggests ideas for future work in the area.

Chapter 2

Background

2.1 Theory

In this section, I provide a brief overview of the concepts and background knowledge required to understand this work. Note that this is limited only to the concepts which are relevant.

2.1.1 Machine learning fundamentals

Regression is the task of predicting a numerical value (output variable) given a set of input variables. **Linear regression** models this as a linear function of the input variables, $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$; the error of this model is the difference from the true value, $\epsilon = y - \hat{y}$.

Classification is the task of assigning a *label* to given input data. In **binary classification**, data is assigned to one of two classes (usually with labels of 0 and 1).

A **neural network** is a machine learning model inspired by human biology, comprising *nodes* arranged in *layers* with weighted connections between layers, and *activation functions* which determine the response of a node given its input connections. A **recurrent neural network** (RNN) involves feedback connections where outputs of the model are fed back into itself. A **convolutional neural network** (CNN) is a network in which *convolutional layers* perform discrete convolutions in 1D, 2D, or 3D space over their input. Noriega (2005) provide an introduction to neural networks, while Lim and Zohren (2021) describes recurrent and convolutional neural networks in more detail in the specific context of time series forecasting (defined below).

Supervised machine learning involves training models by providing *training examples* of inputs and outputs. In evaluating a model's performance, data can be divided into a **training set** and **test set**, where the model is given both input and output from the training set, but only input from the test set; its outputs on the test set can then be scored against the true outputs.

2.1.2 Time series

A time series is a series of data-points in time, $x_1, x_2, x_3, ..., x_n$ where $x_i \in \mathbb{R}$.

A time series is said to be **stationary** if its statistical parameters, such as mean and variance, do not change with time (Hyndman and Athanasopoulos 2018). Time series may have a **trend**, an increase or decrease in the long-term, or **seasonality**, patterns in the time series of a fixed (and known) frequency, such as annually.

Time series forecasting refers to predicting future values of a time series. The *forecasting horizon* is the length of time into the future that the forecast is for. For instance, a next-day forecast has a forecasting horizon of one day. Predicting future values for a range of forecasting horizons is known as *multi-horizon forecasting*.

Time series classification refers to assigning a label to a given time series. Many problems can be framed as a time series classification task, such as speech recognition (labels are words/phonemes being spoken); electrocardiogram diagnosis (labels are the binary presence of a heart condition); and stock prediction (labels are whether the stock price is expected to increase or decrease).

2.1.3 Stock market fundamentals

In order to allow traders to buy and sell shares of stock in a company, the stock must be listed on a **stock exchange**. The largest stock exchanges in the world[1] are the New York Stock Exchange (NYSE) and Nasdaq.

The overall performance of the stock market is often tracked by investors via **stock indices**, which are aggregations of a selected group of stocks with some form of weighting applied to each stock. Stock indices are generally treated like stocks, i.e. their price is tracked over time, though they cannot be bought in the same way[2]. Two of the most well-known stock indices in the English-speaking world are the Dow Jones Industrial Average (DJIA), which tracks 30 large companies selected by a committee; and the Standard & Poor's 500 (S&P 500), which tracks 500 large companies, also selected by a committee.

Stock market prediction is a loosely defined term, which refers to predicting *some aspect* of a stock, stocks, or the market as a whole — most commonly price or trend (though e.g. volatility is also sometimes the object of prediction). I divide this more specifically into **stock price prediction**, where the aim is to predict actual values of stock prices; and **stock trend prediction**, where the aim is to predict the direction of movement of a stock's price, such as 'up', 'down', or sometimes 'flat' (often defined as an increase or decrease within some small threshold). Throughout this work, I employ 'stock market prediction' as an umbrella term encompassing both of these tasks.

2.1.4 Sentiment analysis

As noted in the introduction, sentiment analysis is the task of determining the sentiment of text. There are a number of different approaches to the task of sentiment analysis. One of the simplest involves the use of a **sentiment lexicon**, where sentiment values

[1] By market capitalization, $\sum_{\text{all stocks on exchange}}$ stock price $\times$ shares owned by investors

[2] It is possible to invest in the stocks tracked by a stock index through the use of an *index fund*

of a number of words are explicitly recorded — e.g. 'great' may have a strong positive sentiment value. This can be used directly in a simplistic manner to compute the sentiment of an entire sentence or document (e.g. by summing or averaging the sentiments of words which are in the sentiment lexicon).

Some sentiment analysis approaches associate sentiments with 'aspects' extracted from the text — e.g. in "the prospects of Apple are good, but the economy is bad", 'Apple' and 'economy' may be aspects. This leads to richer sentiment analyses than approaches which extract only one sentiment value for the entire text.

More recently, **deep learning** approaches have been utilised in sentiment analysis. Deep learning is able to learn rich representations of data by incorporating many layers of abstraction (Y. LeCun, Bengio, and Hinton 2015). Applied to natural language tasks, deep learning can learn rich numerical representations of words. From this, representations can readily be fed into a classification layer, which can be trained using labelled corpora of sentences, texts, etc.

2.2 Related work

2.2.1 Stock market prediction with social media

Bollen, Mao, and Pepe (2011) investigated the link between public sentiment via Twitter (utilising 1.1M tweets and a sentiment lexicon of 793 'mood adjectives') and socio-economic phenomena, including DJIA prices. This was among the first to note that Twitter was an interesting source to explore public sentiment trends due to "the very flexible and ephemeral nature of its content". They concluded that socio-economic, political, and cultural events have a significant impact on mood measured on Twitter, validating the idea that online sentiment could have predictive power via this impact, if the EMH does not hold. Subsequently, the seminal work of Bollen, Mao, and Zeng (2011) attempted to predict the movement of the DJIA using Twitter and a similar lexicon-based sentiment analysis approach, and obtained an accuracy of 86.7% in predicting stock movement direction (up or down). However, this study suffered from a small test set of just 19 days, and hence the validity of these results may not generalise to performance in real-world applications.

Later works directly attempted to address some of these methodological concerns, however generally focused on a relatively small number of stocks/stock indices. Porshnev, Redkin, and Shevchenko (2013) predicted the DJIA with an accuracy of 64.1% using a mood lexicon-based sentiment analysis approach with 74 days in the test set, however found this was not a significant increase in accuracy over using price data alone (technical analysis). T. H. Nguyen, Shirai, and Velcin (2015) predicted 18 stocks over one year with an average accuracy of 54.4% and 78 days in the test set, using posts on the Yahoo! Finance message board and an aspect-based sentiment analysis considering consecutive nouns to be aspects. Derakhshan and Beigy (2019) improved on this with Latent Dirichlet Allocation (Blei, Ng, and Jordan 2003) sentiment analysis, achieving 56.2–66.0% accuracy with the same data-set.

Deep learning has recently become more popular in the task of stock prediction for both sentiment analysis, and prediction of stock direction and price. Xu and Co-

hen (2018) provided one of the first attempts to predict a large number of stocks (88) using deep neural networks to encode tweet/price information and make predictions. They achieved an accuracy of 58.2% and a Matthews correlation coefficient[3] (MCC) of 0.0808. However, they ignored over one third of samples of price movements corresponding to days where a stock's price changed by $[-0.55\%, 0.5\%]$, and thus these results may not apply to the problem of predicting stock movement for any given day. Sawhney et al. (2020) reported a higher accuracy and MCC of 60.8% and 0.195 respectively on the same data-set, using an attention-based neural network. However, this suffers from the same issues of applicability.

My work aims to provide insight into the potential for stock prediction using social media data for any stock, and hence I examine **every stock** listed on the two largest stock exchanges, and attempt prediction on a smaller but substantial subset. I maintain a focus on **applicability** throughout to ensure that my results not only provide an accurate estimate of the generalisability to real-world applications, but *can* be applied to real-world applications. I consider a period of one year to ensure a test set of sufficient size and breadth can be used, utilise **every** trading day within this period, and investigate a number of different forecasting horizons to maximise utility to investors. Moreover, I present an interface which can be utilised by investors to run the developed models and view predictions with ease.

2.2.2 Time series forecasting & classification

Time series forecasting

Charles C Holt (1957) and Brown (1957) proposed one of the simplest forecasting models: 'simple exponential smoothing'. This model assigns exponentially-decreasing weights to previous observations. However, it proved unsuitable for time series with a trend or seasonal component. Holt and his student Peter Winters extended the model to account for these (Winters 1960), creating 'Holt-Winters exponential smoothing'. This technique has remained popular for at least 50 years after its introduction, due to its simplicity, speed, and effectiveness (Goodwin et al. 2010).

Box and Jenkins (1970) developed another highly influential time series forecasting approach, which was a method of fitting a statistical autoregressive integrated moving average (ARIMA) model and using this to produce forecasts; this method is now known as the Box-Jenkins method. Along with exponential smoothing, this remains one of the most widely used approaches to forecasting (Hyndman and Athanasopoulos 2018).

Historically, these methods have been difficult to beat (Makridakis and Hibon 2000), especially by machine learning. The latest 'Makridakis competition' (a time series forecasting competiton held routinely since 1982), M4 (Makridakis, Spiliotis, and Assimakopoulos 2020), examined 61 forecasting methods applied to 100,000 time series. **All** pure machine learning methods used in the competition performed below naïve benchmarks.

Despite the results of M4, many RNN models have been utilised in time series forecasting (Salinas et al. 2020; Rangapuram et al. 2018; Wen et al. 2017; Di Persio

[3]The Matthews correlation coefficient is a metric for evaluating binary classification performance which can be used even for imbalanced labels. This will be expanded upon in Chapter 6

and Honchar 2017). Siami-Namini, Tavakoli, and Namin (2018) found that, contrary to M4's results, an long short-term memory (Hochreiter and Schmidhuber 1997) model was able to reduce error rates by 84–87% compared to ARIMA in a range of financial time series forecasting experiments.

Following the development and popularisation of attention methods in deep learning models, a number of models have been created more recently which use attention in order to better learn long-term dependencies and patterns. Fan et al. (2019) developed a temporal attention-based model for multi-horizon probabilistic time series forecasting, which they found to achieve state-of-the-art performance on two large data-sets. Building on the success of the attention-driven Transformer model in other domains, Lim, Arik, et al. (2019) created a Transformer-based multi-horizon time series forecasting model, which again achieved state-of-the-art performance on data-sets from a variety of domains.

Time series classification

State-of-the-art time series classification (TSC) models have obtained great performance improvements by transforming time series "into an alternative data space where discriminatory features are more easily detected" (Bagnall et al. 2015). Initial implementations of this idea such as HIVE-COTE (Lines, S. Taylor, and Bagnall 2016) suffered from long training times, and so more recently leaner approaches have been developed. 'InceptionTime' (Ismail Fawaz et al. 2020) implemented transformations with CNNs, achieving accuracy on-par with HIVE-COTE with a training time of one hour for 1,500 time series. Dempster, Petitjean, and Webb (2020) further improved upon this with 'ROCKET' — an approach to transforming time series data with randomly-generated convolutional kernels. This approach again achieved state-of-the-art accuracy, with a minimal training time of seconds for time series of 2,048 observations (compared to HIVE-COTE's equivalent time of several days).

Relation to the present work

To test the findings of the M4 competition, I utilise statistical models as baselines and compare a variety of machine learning models against them. Additionally, I present a model which attempts to build on the success of RNNs, attention methods, and CNNs, and apply this to the tasks of stock price prediction and stock trend prediction. ROCKET is also investigated as a fast and practicable model.

2.2.3 Sentiment analysis

In recent years, deep learning approaches have been applied extensively in the field of natural language processing (NLP), owing to increases in computing power and data (Otter, Medina, and Kalita 2020). RNN architectures have often been used due to their ability to learn long-term dependencies in data. However, these architectures suffer from long training times due to a high number of parameters and fundamental limitations in parallelizability. The Transformer architecture (Vaswani et al. 2017) was

developed to overcome these issues, and has since been utilised in many state-of-the-art NLP models such as BERT (Devlin et al. 2018). BERT makes use of *pre-training*, whereby the model's parameters are initialised by training it on unlabelled data over different tasks. This facilitates the use of large data-sets (e.g. 2300M words) and allows for comparatively smaller numbers of training examples in down-stream tasks (i.e. the actual tasks for which the pre-trained model is to be used).

A number of models have since been created which build on and improve upon BERT. RoBERTa (Liu et al. 2019) improved the pre-training process of BERT, achieving superior performance in equivalent tasks, including sentiment analysis.

Tweets are a challenging target for sentiment analysis due to their short length (280 characters[4]) and use of abbreviations, slang, emoticons, etc. Therefore, several approaches have been developed for this task. Hutto and Gilbert (2014) developed a sentiment lexicon-based model, VADER, using a 'human-centered approach'. D. Q. Nguyen, T. Vu, and A. T. Nguyen (2020) pre-trained a BERT model with the RoBERTa pre-training procedure on 850M tweets to produce 'BERTweet'. This model exceeded the performance of both BERT and RoBERTa in a number of Twitter-specific NLP tasks, including the SemEval-2017 Task 4 Twitter sentiment analysis task (Rosenthal, Farra, and Nakov 2017).

Some contemporary works have applied transformer-based models to the task of sentiment analysis of tweets for the task of stock prediction specifically. Lee, Gao, and Tsai (2020) used BERT to perform sentiment analysis on tweets relating to stocks, and obtained a promising accuracy of 87.3% in predicting investor sentiment on a labelled data-set. Bozanta et al. (2021) compared the transformer-based models BERT, XLNet, DistillBERT, and RoBERTa in classifying the sentiments of tweets relating to five prominent stocks and one stock index, and found that RoBERTa provided the best performance on this task.

My work makes use of the pre-trained BERTweet model, fine-tuned for sentiment analysis, in light of these promising results for the underlying RoBERTa training procedure, BERTweet itself, and the recent works which apply BERT-based models to sentiment analysis of tweets in stock prediction specifically. I also leverage VADER sentiment analysis to provide information where BERTweet cannot, given they are fundamentally dissimilar models.

[4]https://developer.twitter.com/en/docs/counting-characters

Chapter 3

Problem statement

3.1 Stock price prediction

Given a series of n observations of a trading day-aligned multivariate time series of input features $\mathbf{x}_t$, we wish to predict the closing price of a stock y_t for every trading day within the multi-horizon window H. That is, we wish to find a function:

$$f(\mathbf{x}_{t-n+1}, \mathbf{x}_{t-n+2}, \ldots, \mathbf{x}_t) := \begin{bmatrix} y_{t+1} & y_{t+2} & \ldots & y_{t+H} \end{bmatrix}$$

where t is the index of a trading day.

Specifically, $\mathbf{x}_t$ in this work comprises features extracted from both technical data (stock prices etc.) and social media data (sentiment values etc.). I examine $H \in \{1, 5, 20\}$, corresponding approximately to next-day, weekly, and monthly multi-horizon forecasts (where 5 trading days comprise 1 week).

3.2 Stock trend prediction

Let the trend of a stock on trading day t with respect to a horizon H be defined as follows:

$$Trend(t, H) := \begin{cases} 1 & \text{if } y_{t+H} > y_t \\ 0 & \text{otherwise} \end{cases}$$

where y_t is the closing price of the stock on trading day t.

Hence, as in Section 3.1, the aim is to find a function of the trading day-aligned multivariate time series of input features $\mathbf{x}_t$, i.e.

$$g(\mathbf{x}_{t-n+1}, \mathbf{x}_{t-n+2}, \ldots, \mathbf{x}_t) := Trend(t, H)$$

While much of the literature focuses on just the case of $H = 1$ (T. T. Vu et al. 2012; T. H. Nguyen, Shirai, and Velcin 2015; Xu and Cohen 2018), I investigate $H \in \{1, 5, 20, 65\}$ for several reasons:

- Horizons longer than one day have not been widely investigated in the literature

- Practicality: a next-day trend prediction is fairly limited in its utility, and likely to be hindered by random noise inherent in the stock market. Longer horizons can readily be converted into trading strategies[1].
- The ideal horizon is unknown, hence it makes sense to investigate a range of horizons

[1] Via the use of 'long put' and 'long call' options

Chapter 4

Data collection

I study the period of 01/01/2019–31/12/2019 for recency, ensuring that social media usage is sufficiently high[1]; and relative stability compared to later years where the COVID-19 pandemic wrought havoc on the stock market (Baker et al. 2020; Mazur, Dang, and Vega 2021). Within this period, there were 252 trading days, which provided ample data for training and testing while remaining within the limit of feasible study given the project's time constraints.

I collected two categories of data: stock data and social media data. Due to the scale of the dataset (18GiB when stored in flat-files), a PostgreSQL database was used to store and retrieve the data.

4.1 Stock information and prices

Stock symbols were obtained programmatically from the Nasdaq website[2], which provides a public list of all stock symbols on both the Nasdaq and NYSE stock exchanges. Stock price data – including open, close, high, and low prices; and volume – was then retrieved for the chosen period individually for each symbol from Yahoo! Finance.

In total, 7,789 stock symbols were collected from the Nasdaq website. Price data was retrievable for 4,935 (63%) of these, with the remainder either non-existent in the chosen period (e.g. if the company was listed after or unlisted before 2019) or missing from the Yahoo! Finance website. 323 stocks in this set did not have full price data for all 252 trading days in 2019 and therefore were excluded from study, leading to the final number of **4,612 stocks** in the data-set.

4.2 Twitter

To collect tweets from Twitter, the Python library Twint (Zacharias 2022) was employed. Twint scrapes the Twitter website in order to collect Tweets via the search

[1] Social media usage in the US – which makes up the largest portion of Twitter's user-base – steadily increased between 2006-2019 and has remained relatively stable since (*Social Media Fact Sheet* 2021).

[2] https://www.nasdaq.com/market-activity/stocks/screener

functionality, and has been employed in several publications chiefly due to its accessibility and availability as an open-source Python library (Nuzhath et al. 2020; Bonsón, Perea, and Bednárová 2019).

Finding relevant tweets for a particular stock presented a challenge: searching for the company name(s) would almost certainly result in a high number of false positive matches for many companies (e.g. Apple) due to the diversity and volume of content on Twitter. However, Twitter allows users to embed 'cashtags' in their tweets for the explicit purpose of stock (and cryptocurrency) discussion. In order to maximise relevancy of collected tweets to stocks in question, search queries of cash-tags were used to find relevant tweets for a given stock.

Specifically, for each stock in the data-set, a search query consisting of the cashtag '$*symbol*' was used. Queries were performed and tweets collected using Twint, with a few patches applied to the library to fix bugs which were present at the time of data collection. Due to the number of stocks, high volume of tweets, and deliberate rate-limiting, this process took 3 weeks to complete.

#	Symbol	Tweets
1	$LTC	279,257
2	$FB	162,879
3	$TSLA	161,873
4	$AAPL	155,116
5	$LINK	154,151
6	$AMZN	148,365
7	$NFLX	138,548
8	$BCH	116,953
9	$EOS	115,971
10	$NEO	86,944

Table 4.1: Top 10 symbols by number of tweets containing the symbol

Overall, 27,275,794 tweets were collected through this method, of which 18,226,604 were discarded as false positives with no cashtags (Twitter's search functionality does not limit results to exact matches of the search terms). From the remainder, a further 8,802 tweets contained only cashtags corresponding to symbols not in the data-set (e.g. indices such as the DJIA, cryptocurrency symbols), leaving 9,040,388 tweets in the data-set.

Figure 4.1 shows the distribution of number of tweets per symbol, based on how many tweets contain a given stock symbol. The median number of tweets per stock symbol is 1,462, which would provide a meagre 4 tweets per day to utilise for prediction. However, 5% of stock symbols have 7,893 tweets or more, corresponding to a slightly more reasonable 20 tweets per day. A small number of symbols have over 100,000 tweets (Table 4.1), however it seems more likely that some of these correspond to cryptocurrencies rather than a stock sharing the same symbol.

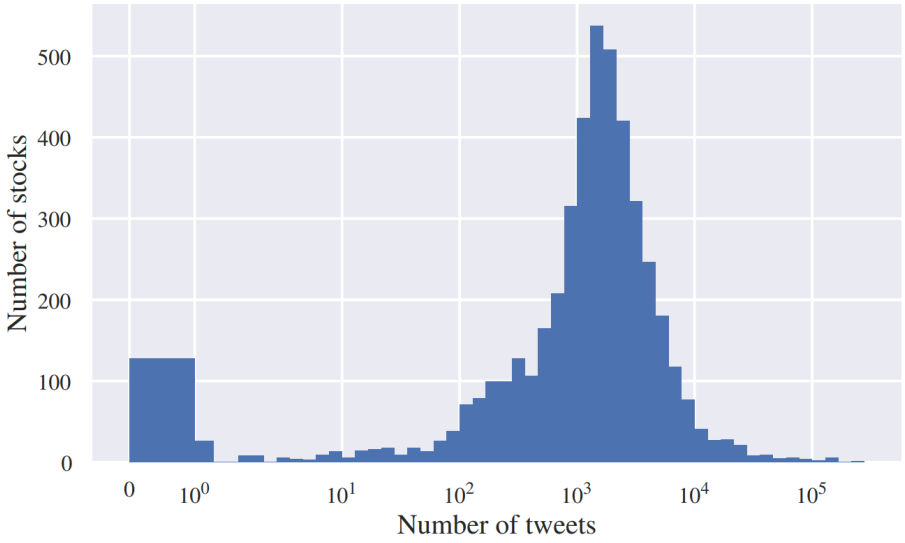

Figure 4.1: The distribution of the number of collected tweets per stock, excluding stocks for which price data was not retrievable.

4.3 Cryptocurrency symbols

In addition to stocks, cryptocurrencies are discussed extensively on Twitter using the same cash-tag format, e.g. '$BTC' (Bitcoin). I noticed that a number of these had equivalent stock symbols which corresponded to companies entirely separate from the cryptocurrency in question. For instance, '$LTC' could refer to the cryptocurrency Litecoin, or the real estate investment trust company LTC Properties Inc. (listed on the New York Stock Exchange as LTC).

To investigate this problem of this overlap, I collected a list of cryptocurrency symbols using the free CoinMarketCap API[3]. This produced 4,636 unique cryptocurrency symbols. Of these, 673 had equivalent symbols in my data-set of stock symbols. Inferring whether the tweets in my data-set were referring to a cryptocurrency or a stock would be non-trivial: for instance, '$LTC' almost certainly would refer to Litecoin, while '$FB' would almost certainly refer to Meta Platforms, Inc. rather than the Fenerbahçe Token listed on CoinMarketCap under the same symbol.

Therefore, I did not exclude any stocks/tweets from study on the basis of overlapping symbols with cryptocurrencies, and instead limited experimental study of stocks based on correlations between collected tweet sentiments and price movement (under the assumption that if tweets are referring to a cryptocurrency, their sentiments would not be correlated at all with the movement of an unrelated stock). However, I did exclude cryptocurrency symbols from the 'top N' option in the interface, which allows users to forecast for the most tweeted stock symbols.

[3]https://coinmarketcap.com/api/

Chapter 5

Methods

5.1 Feature engineering

Feature engineering is an important part of machine learning approaches, aiming to extract numerical *features* from the collected raw data, which can be fed into a machine learning model for prediction. In my project, features may be considered trading day-based functions of the form $f : \{0, 1, ..., N-1\} \to \mathbb{R}$, where N is the number of trading days in the data-set ($N = 252$). From this, it is trivial to split the data-set into subsets for training/testing, produce a vector of features for input into a model, etc.

I break this section down into the categories of features that I extracted from the data, and eventually used for prediction.

5.1.1 Technical features

Following the technical analysis approach of stock prediction, and given that sentiment data alone would be insufficient to achieve reasonable prediction accuracy, I extracted a number of features from the stock price and volume data corresponding to well-established technical indicators used by technical analysts in practice.

Prices

After every *trading day* that the stock market is open, there are four price values available to broadly represent the day: the open (the price of the stock when the market first opens), close (the price when the market closes), high (maximum price), and low (minimum price).

Some company actions can affect stock prices. For instance, a 'stock split' can be enacted by the company, whereby the number of shares in the company increase by a factor. Following a 2-for-1 split, investors will own twice as many shares as prior to the split, each with half of their value before the split. Therefore, if we were to use the open, close, high, and low prices directly, the price of the stock would appear to dramatically change after actions such as a stock split.

To account for this, I use the *adjusted* open, close, high, and low prices as features, which take these company actions into account and avoid sudden dramatic changes in stock price. This approach is often used for stock prediction in the literature (T. H. Nguyen, Shirai, and Velcin 2015; Xu and Cohen 2018).

Volume

Trading volume for a given trading day is the total number of shares traded during that day. Volume is positively associated with price change (Karpoff 1987) and therefore may provide information to aid price prediction. Being a simple integer value, volume is used directly as a feature.

Exponential moving average

An exponential moving average (EMA) is an average of data-points which provides more weight to more recent data-points. Applied to stock prices, this allows us to compute a value which utilises all prices of the stock up to a point in time, smoothing out fluctuations to provide an overall trend. Compared to a simple moving average (which is an unweighted average of all data-points in a fixed-size, moving window), an exponential moving average has the benefits of a) capturing the intuition that more recent stock prices should have more influence on the future stock price than older stock prices; and b) utilising all data-points up to the point for which the EMA is being calculated, rather than only those within a fixed-size window.

Mathematically, we can define the EMA as follows:

$$EMA(x,t) = \alpha \cdot x(t) + (1-\alpha) \cdot EMA(t-1)$$

where $x(t)$ is the series we wish to apply the EMA to, such as the time series of stock closing prices.

The factor α determines how quickly the weight of a previous stock price deteriorates, with a higher α implying faster deterioration. The most commonly used method to set α is with the formula $\alpha = 2/(N+1)$, where N is analogous to the window size of the simple moving average (Treloar 2008). The resulting EMA is often referred to as the N-day EMA.

There are several approaches for setting the first term, $EMA(0)$. For simplicity, I use the common approach of using the first observation as the first value of the EMA, i.e.

$$EMA(x,0) = x(0)$$

Since the first term of the EMA is fundamentally erroneous, the series takes some number of iterations to converge. To avoid shortening the data-set available for training and testing (as waiting for the series to converge would necessitate discarding the first few EMA values), I do not take any approach to account for this, and indiscriminately make use of the first few likely erroneous EMA values as if they were accurate.

I use the technical indicators of a 'fast' 12-day EMA and 'slow' 26-day EMA, both of adjusted closing prices, as features. These 'periods' are both commonly used in technical analysis, and seemed sufficiently short so as to avoid a long period of unreliable values before convergence.

Moving average convergence-divergence

A valuable indicator to technical analysts is the *momentum* of an asset (Kaufman 2003): if the price of a stock has risen since N days ago, then the momentum of the stock is positive and the difference in prices over this period gives the magnitude of the momentum. If this momentum is positive and large, the price of a stock is rising fast, which can be interpreted as a signal to buy stock.

Using a simple difference between two prices to calculate momentum (e.g. $p_t - p_{t-N}$) would be erratic and provide an unstable indicator of momentum highly affected by day-to-day fluctuations in price. Therefore, prices are smoothed from their original series using, for instance, an exponential moving average.

The **moving average convergence-divergence** (MACD) is an indicator of momentum commonly used by technical analysts which takes the difference between a 'fast' and 'slow' EMA — most commonly the 12-day and 26-day EMAs, respectively, of closing price. Mathematically:

$$MACD(t) = EMA_{12}(p_c, t) - EMA_{26}(p_c, t)$$

where $p_c(t)$ gives the closing price of a stock on the trading day t.

This series is again smoothed with another exponential moving average to produce a *signal line*, which may be used to show, resiliently to any fluctuations in the MACD, changes in the overall momentum of a stock. Most commonly, a 9-day EMA is used for this purpose, i.e.

$$MACD_{signal}(t) = EMA_9(MACD, t)$$

I utilise both the MACD of the adjusted closing price of a stock and the derived signal line as features for prediction.

On-balance volume

On-balance volume is another indicator of momentum popularised by Granville (1963). It is defined as follows:

$$OBV(t) = OBV(t-1) + \begin{cases} v(t) & \text{if } p_c(t) > p_c(t-1) \\ 0 & \text{if } p_c(t) = p_c(t-1) \\ -v(t) & \text{if } p_c(t) < p_c(t-1) \end{cases}$$

where $v(t)$ is the trading volume of the stock on a given trading day.

Clearly, the OBV necessitates a choice in an initial value $OBV(0)$ like in exponential moving averages. I use the simple setting of $OBV(0) = 0$, where $t = 0$ is the first trading day in the data-set.

While both MACD and OBV are momentum indicators, I utilise both as features to maximise diversity, reflecting a diversity of attitudes and approaches among technical analysts.

5.1.2 Sentiment features

To capture the sentiment of investors and traders in the collected Twitter data, I applied sentiment analysis to the data-set and extracted from this features suitable for use in prediction. I utilised complementary approaches for sentiment analysis, including one state-of-the-art approach, in an attempt to exploit the individual strengths of each approach and therefore capture a robust collection of sentiment features for use in prediction. These approaches are VADER and BERTweet sentiment analysis, which are described in this section.

VADER sentiment analysis

As discussed in Section 2.2.3, VADER (Hutto and Gilbert 2014) is a rule-based sentiment analysis technique making use of a specialised sentiment lexicon for social media text. I utilised the open source VADER Python library maintained by Hutto[1] to extract sentiments from each tweet.

VADER produces from a tweet four scores: `pos`, `neu`, `neg`, and `compound`. The former three scores are proportions of the text which fall into each category (positive, neutral, or negative sentiment), while the latter score provides a "normalised, weighted composite score" of sentiment in the range $[-1, 1]$. Following the recommendation of the authors of VADER and common practice, I utilised the compound score in VADER sentiment features.

Trading day alignment There was the question of how to represent the overall sentiment for a particular day given all tweets which occurred on that day. I devised and implemented four strategies of aggregating sentiments for a particular day. Let $Compound(t)$ be the compound sentiment scores for all tweets on a day t, e.g. $Compound(t) = \{0.14, -0.59, 0.98, -0.93, ...\}$. Then, the aggregation strategies were as follows:

$$VADER_{sum}(t) = \sum_{c \in Compound(t)} c$$

$$VADER_{avg}(t) = \frac{1}{|Compound(t)|} VADER_{sum}(t)$$

$$VADER_{bintot}(t) = \sum_{c \in Compound(t)} \begin{cases} 1 & \text{if } c > 0.05 \\ 0 & \text{if } -0.05 \leq c \leq 0.05 \\ -1 & \text{if } c < -0.05 \end{cases}$$

$$VADER_{bin}(t) = \text{sgn}(VADER_{bintot}(t))$$

The latter two aggregation strategies make use of the classification thresholds used by Hutto and Gilbert (ibid.).

Empirically, sum and average aggregation strategies worked best, with the sum obtaining the highest performance. Intuitively, this was expected as sum retains the

[1] https://github.com/cjhutto/vaderSentiment

most information: it encapsulates all sentiment strengths using the compound score directly, and the overall volume of tweets for the particular stock.

To demonstrate this, I ran an experiment which trained a random forest model to predict stock prices using these four features ($VADER_{sum}(t)$, $VADER_{avg}(t)$, etc.) and extracted the corresponding feature importances. Figure 5.1 shows the results of this experiment on the top 5% of stock symbols by number of tweets (excluding any symbols with a conflicting cryptocurrency symbol). Interestingly, sum and average aggregation had similar importance, however I utilised only sum as a final feature following my intuition.

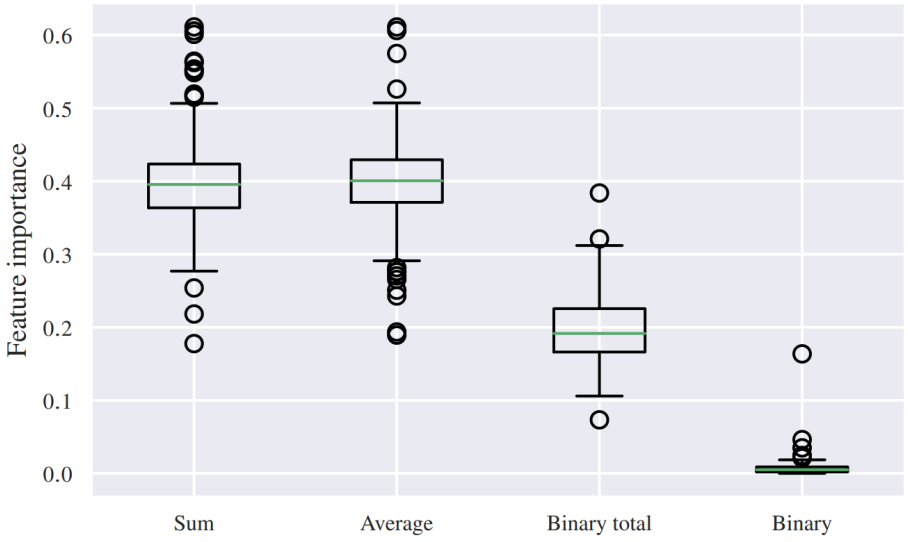

Figure 5.1: Feature importances extracted from a random forest regression model using features corresponding to each aggregation strategy of VADER sentiment (and no others), over a distribution of 200 stocks.

BERTweet sentiment analysis

I utilised the open source Pysentimiento Python library (Pérez, Giudici, and Luque 2021) in order to extract sentiments using BERTweet. This library provided a BERTweet model trained on the SemEval 2017 task 4A corpus (Rosenthal, Farra, and Nakov 2017), consisting of about 40,000 English tweets labelled as positive, neutral, or negative sentiment.

To represent the sentiment for a particular trading day given all the sentiment probabilities of each tweet in that day, I implemented the sum and average aggregation strategies as in VADER. However, as this model outputted three values, this produced

six candidate features overall:

$$BERTPos_{sum}(t) = \sum_{m \in Tweets(t)} P(m \text{ is positive}; \theta_{BERTweet})$$

$$BERTPos_{avg}(t) = \frac{1}{|Tweets(t)|} BERTPos_{sum}(t)$$

and likewise for neu/neg

I investigated a number of strategies to transform the positive, neutral, and/or negative probabilities into a single value with the aim of producing a single 'compound' feature with more predictive power than any individual probability feature. My approach to this was to apply a range of simple mathematical functions and determine experimentally which was best. The most sensible of these were as follows:

$$BERTCompound^{(1)}_{sum}(t) = BERTPos_{sum}(t) - BERTNeg_{sum}(t)$$

$$BERTCompound^{(2)}_{sum}(t) = BERTPos_{sum}(t) \cdot BERTNeg_{sum}(t)$$

$$BERTCompound^{(3)}_{sum}(t) = \log(BERTPos_{sum}(t) + 1) - \log(BERTNeg_{sum}(t) + 1)$$

and likewise for the average aggregation strategy.

Utilising all of these 12 features in prediction would likely introduce noise and hinder prediction overall. Therefore, I compared their correlations with the (adjusted) closing price of a stock, and found after manually comparing a few high-volume stocks (`$AAPL`, `$TSLA`, etc.) that Compound 3 (sum) was almost certainly the most correlated. I utilised positive, neutral, and negative probability features (both sum and average) with this Compound 3 (sum) feature in the final experiments, with the aim of avoiding noise and maximising available information for prediction.

Figure 5.2 shows the correlations more formally for demonstrative purposes (though was done after the fact): the mean correlation between each candidate feature (lagged by 1 trading day) and the adjusted close price over the top 5% of stock symbols by number of tweets (excluding any potential cryptocurrency symbols) is plotted. Compounds 1 and 3 in fact exceeded the correlations of each individual probability feature, with Compound 3 (sum) obtaining the highest mean correlation coefficient. The individual probability features have fairly low correlations, however this is likely due to lack of weighting or filtering, or tweet volume not correlating with predictive power of tweets on stock price.

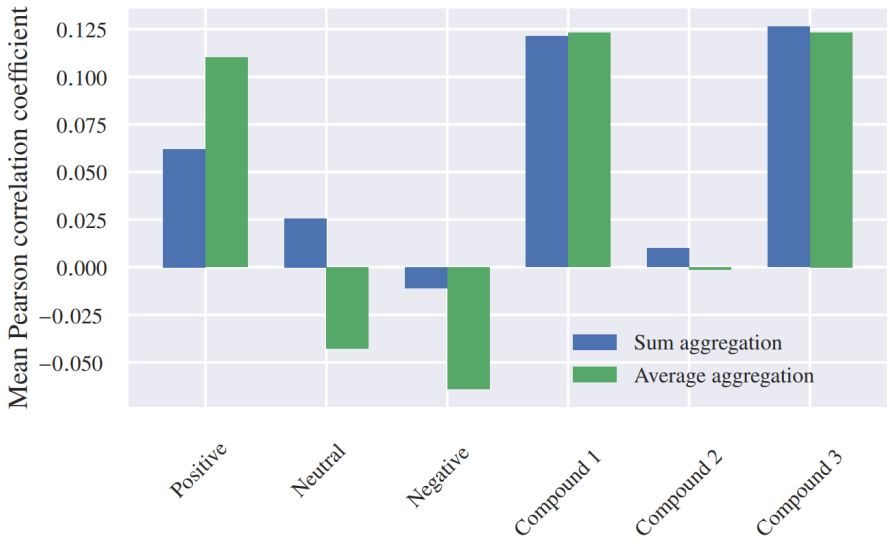

Figure 5.2: Pearson correlation coefficients (r) between candidate BERT features and stock closing prices, averaged over 200 stocks

Tweet filtering and weighting

A large number of tweets did not provide much useful information about overall investor sentiment towards a stock, and often mentioned many stocks at once (e.g. Figure 5.3). Some previous works have utilised topic and aspect-based sentiment analysis which may alleviate these issues (T. H. Nguyen, Shirai, and Velcin 2015), however the fine-tuned BERTweet model and VADER sentiments provide only an overall sentiment of text. Hence, the sentiment for a tweet mentioning multiple stocks would be the same for all stocks mentioned.

To address this issue, I filtered out all tweets which mentioned more than one symbol. As shown in Figure 5.4, this was a minority of tweets, and 6,822,957 tweets remained in the data-set after filtering.

Many tweets also had some number of retweets or likes: after filtering, 8.55% of tweets had at least one retweet, and 18.4% had at least one like. Retweets and likes are means by which users on Twitter may express agreement with a tweet and, in the former case, share it with their followers. Therefore, retweets and likes could provide a sensible way to give more weight to tweets engaged more with by users, and amplify investor sentiments with widespread agreement. This approach has been used in some prior works such as Carosia, Coelho, and Silva (2020), although broadly its use seems limited.

Figure 5.3: Example of multiple stock symbols referred to in a single tweet with no explicit information on investor sentiment.

I investigated two forms of weighting. Formally:

$$Weight_R(m) = Retweets(m)$$
$$Weight_{RL}(m) = \frac{\sum_q Retweets(q)}{\sum_q Likes(q)} \cdot Likes(m) + Retweets(m)$$

where $Weight_*(i)$ gives the weight for a tweet i, to be multiplied by the sentiment value(s) in summing and averaging over a trading day.

Empirically, testing on a small handful of stocks, I found that retweet weighting provided the highest correlation with stock prices, and therefore I utilised this weighting in the experiments.

Notably, these weights are zero for tweets with no retweets (in the first case), or no likes and no retweets (in the second case). Thus, tweets with no agreement from at least one other user are discarded, which is a majority of tweets in both cases. Given this, I included retweet-weighted BERTweet sentiments aggregated with summing and averaging for each of the positive, neutral, and negative probabilities (six features in total) *in addition* to the unweighted BERTweet sentiment features. However, all of these features made use of the filtering described above.

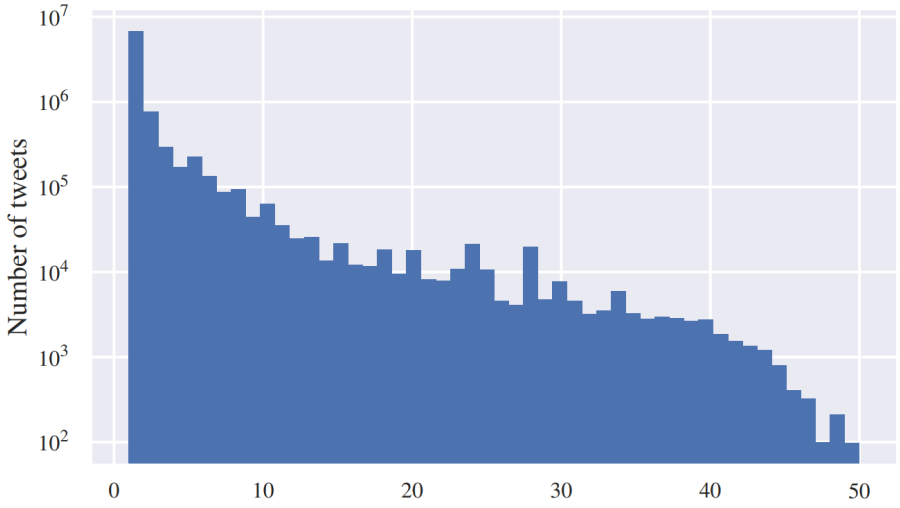

Figure 5.4: Distribution of tweets by number of symbols mentioned via cash-tags (i.e. including stock symbols not in the data-set). Around 6.8M tweets mentioned only one symbol.

Tweet counts

When news and events take place that affect companies, discussion surrounding these companies increases in volume. Additionally, news and events is often related to or reflected in changes in a company's stock price. Under these assumptions, I implemented an additional feature of 'tweet counts', which is simply the number of tweets concerning a stock on any given trading day (without filtering).

Figure 5.5 shows the change in tweet count for `$AAPL` stock; spikes in the number of tweets appear to coincide with changes in stock price, though it remains uncertain whether they precede these changes. This feature was implemented with the hypothesis in mind that at least some changes in stock price are preceded by changes in tweet volume and social media engagement.

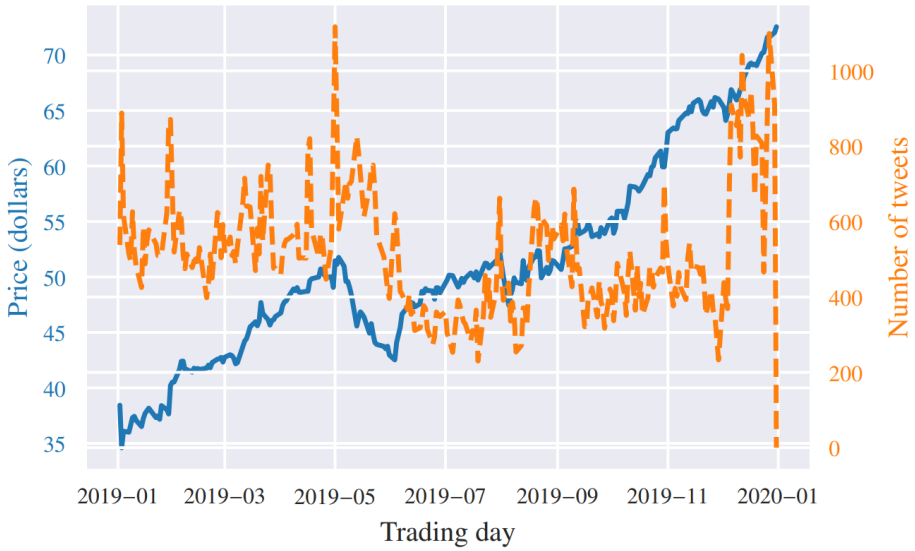

Figure 5.5: Number of tweets plotted against stock price for Apple Inc. (`$AAPL`) over all trading days in the data-set.

Trading day alignment

While the technical features discussed are all aligned with trading days (since price/volume data is only released on trading days), tweets occur on any day of the week, and hence there is the question of how to align sentiment data with trading days.

Various approaches have been proposed to deal with this problem: Xu and Cohen (2018) and Makrehchi, Shah, and Liao (2013) utilised all tweets from consecutive non-trading days preceding (and including) a trading day; Sprenger et al. (2014) aligned all tweets after 4:00PM (the closing time of US stock markets) on a trading day to the following trading day.

In this work, I utilised only tweets which fell on trading days for prediction. This should be considered to be a limitation of this work: through this method many tweets are needlessly discarded. However, even with this limitation, the volume of tweets available for prediction is still fairly substantial.

5.1.3 Summary of features

Table 5.1 summarises the features included in the prediction models and experiments.

Type	Feature
Technical	Price (close)
	Price (open)
	Price (high)

	Price (low)
	EMA of closing price (12-day)
	EMA of closing price (26-day)
	MACD of closing price (MACD line)
	MACD of closing price (signal line)
	On balance volume
Sentiment	VADER (sum)
	BERTweet (positive probability, sum, retweet-weighted)
	BERTweet (positive probability, sum, no weighting)
	BERTweet (neutral probability, sum, retweet-weighted)
	BERTweet (neutral probability, sum, no weighting)
	BERTweet (negative probability, sum, retweet-weighted)
	BERTweet (negative probability, sum, no weighting)
	BERTweet (positive probability, average, retweet-weighted)
	BERTweet (positive probability, average, no weighting)
	BERTweet (neutral probability, average, retweet-weighted)
	BERTweet (neutral probability, average, no weighting)
	BERTweet (negative probability, average, retweet-weighted)
	BERTweet (negative probability, average, no weighting)
	BERTweet (compound 3, sum, no weighting)
	Tweet counts

Table 5.1: All engineered features which were included in experiments

5.1.4 Feature lag

In predicting the change in stock price for, e.g. a week's time from the present, it would almost certainly be insufficient to utilise only the present day's price and sentiment values, as a trend could not be extracted from this. Therefore, it is necessary to utilise a window of time for prediction.

Various window sizes have been used in previous research for this purpose. Bollen, Mao, and Zeng (2011) and Porshnev, Redkin, and Shevchenko (2013) examined each day in a 7-day window separately, while T. H. Nguyen, Shirai, and Velcin (2015) used just a 2-day window of trading days prior to the target day for prediction.

Building on previous work and to avoid long training times, I utilise a window of 5 trading days (corresponding to a week, excepting holidays) prior to a 'pivot day'. I define pivot day as the first trading day for which the closing price is unavailable/hidden. Thus, next-day price predictions will target this pivot day, while e.g. 5-day predictions will target the day 4 days after the pivot day.

More formally, for a pivot day t_0 and feature set $\mathbf{F} = \{f_1, f_2, ..., f_n\}$, the data utilised for prediction is:

$$\mathbf{x}_{t_0} = \bigcup_{k \in [1,5]} \{f_i(t_0 - k) | f_i \in \mathbf{F}\}$$

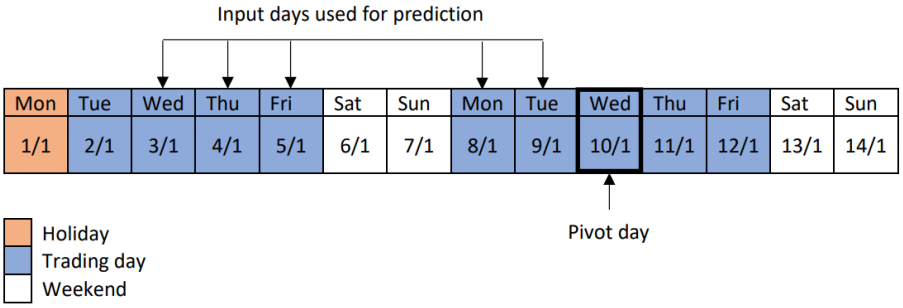

Figure 5.6: Visual explanation of pivot days and the 5-day lag window used for prediction. Dates are 'day/month', all in 2019.

5.2 Feature selection and extraction

In supervised machine learning problems, it is generally inadvisable to have highly correlated features (multicollinearity). Aside from increasing computation and/or learning time unnecessarily, such features can hinder the generalisation performance and stability of models. For instance, one independent variable underlying several correlated features could have an unduly large influence on prediction, increasing the risk of over-fitting (this is the case for e.g. an SVM with an RBF kernel).

Feature selection is an umbrella term for techniques which reduce a set of features into a smaller subset consisting of only relevant features, while feature extraction refers to *transforming* one set of features to a new set (Sammut and Webb 2011). I investigated both for the purpose of reducing redundancy and multicollinearity in the gathered features, aiming to produce a set of features highly correlated with the unlagged stock price, but not with each other. Both approaches have the additional benefit of reducing the dimensionality of inputs to the machine learning models, thereby reducing training time and simplifying the models.

Some models such as random forest and neural networks perform feature selection intrinsically (and are often employed for this express purpose). Additionally, these models are fairly resilient to multicollinearity: for instance, neural networks guard against the issues of multicollinearity due to their "redundant architecture" (Veaux and Ungar 1994); while random forest models utilise random subsets of features in training each decision tree, and these decision trees split by selecting features that best separate training examples (and therefore there is no bias towards correlated features: features are used for decisions only if they are good at separating the data). Hence, in these cases I delegate feature selection to the models themselves. However, in the case of neural networks, this comes at a cost of slightly increased training time/slower convergence.

In this section, I demonstrate the multicollinearity in the gathered features and describe in brief the feature extraction and feature selection techniques applied.

5.2.1 Multicollinearity in gathered features

To investigate the level of multicollinearity in the features gathered in Section 5.1, I employed two methods: the Pearson's correlation coefficient, and variance inflation factors. Rather than perform an exhaustive investigation, I examined a small number of stocks chosen based on their high tweet volumes manually. The results are reported for `$AAPL` stock only in this section, for brevity.

Pearson's correlation

I computed the Pearson's correlation coefficient matrix between all gathered features and plotted the result as a heat-map in order to detect high levels of correlation between features.

The result for `$AAPL` stock over the entire period under study (01/01/2019–31/12/2019) is shown in Figure 5.7. Generally, correlation coefficients of 0.80 or higher are considered diagnostic of multicollinearity (Berry, Feldman, and Stanley Feldman 1985). There are several such occurrences of collinearity among the gathered features. In particular, there is exceptionally high correlation between open/high/low/close prices and exponential moving averages, and very high correlations of 0.72–0.99 among sum-aggregated BERTweet probability features. The MACD line and MACD signal line features correlate very highly with each other, which is to be expected (since the latter is an EMA of the former); and on-balance volume correlates fairly highly with price and EMA features.

Many features have unexpectedly poor correlations with the target feature — the closing price. For instance, BERTweet with average aggregation of negative probabilities has a correlation coefficient of 0.04–0.12; while tweet count has a meagre -0.02 correlation coefficient. This is indicative that these features (in the case of a 1 day lag for `$AAPL` stock) likely do not provide any relevant information for prediction.

From these results, it seems clear that there is a very high degree of multicollinearity in the gathered features, in addition to a number of irrelevant features/noise. Thus, feature selection is highly important to address this issue.

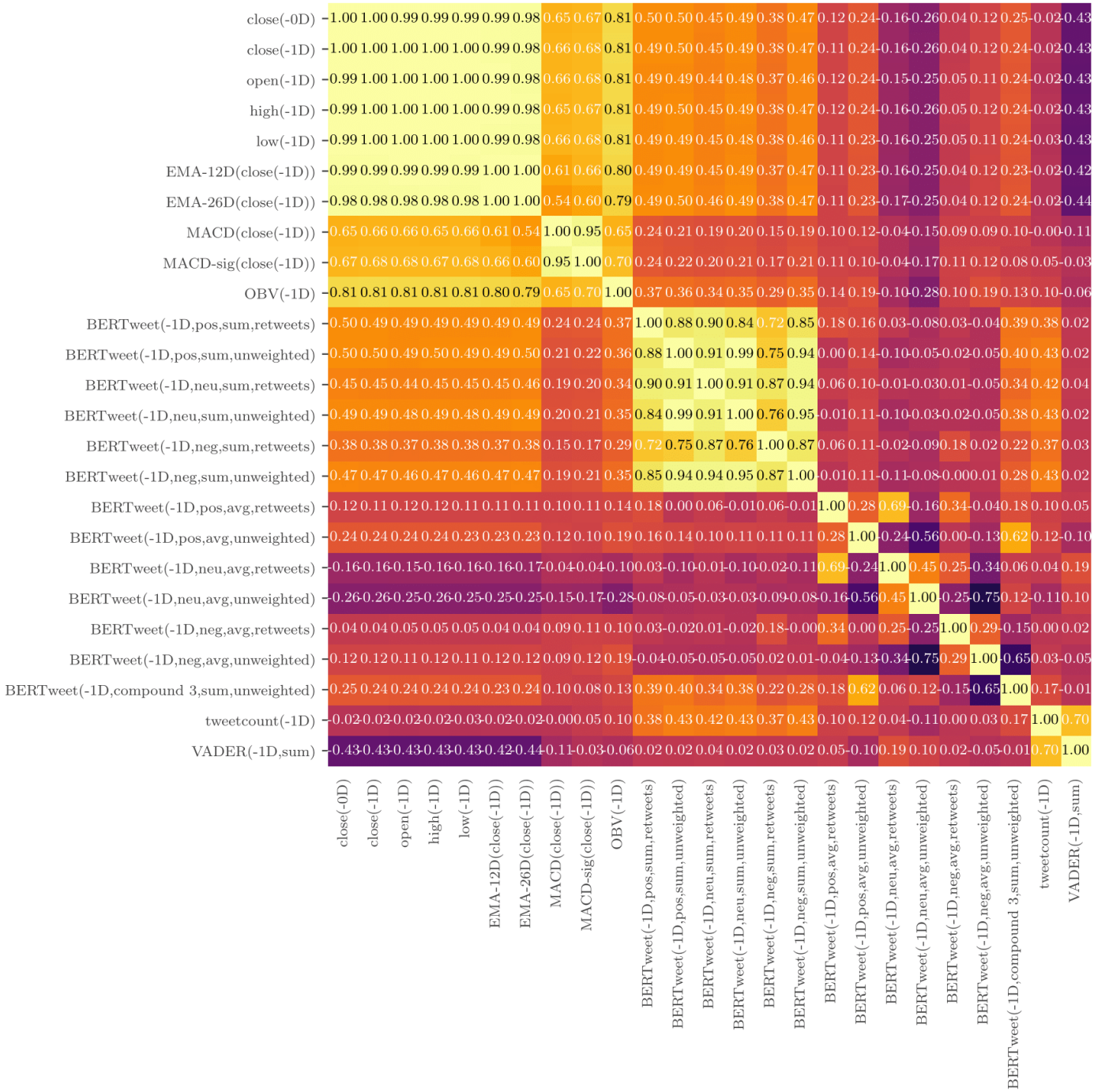

Figure 5.7: Correlations between features lagged by 1 trading day, and unlagged closing price for the pivot day for `$AAPL` stock. Light yellow indicates strong positive correlation, while dark purple indicates strong negative correlation.

Variance inflation factor

The variance inflation factor quantifies multicollinearity directly by assessing how well an ordinary least squares (OLS) regression model can be fitted to one variable (or feature) given the remaining variables through the coefficient of determination R^2. Often this can be a more informative measure than Pearson's correlation alone, since it takes into account all other variables at once, whereas Pearson's only gives the correlations between two variables at a time.

If $R^2_{X_i|X_{-i}}$ is the coefficient of determination for an OLS regression model which predicts variable X_i given all other variables X_{-i}, the variance inflation factor can be computed as follows (James et al. 2021):

$$VIF_i = \frac{1}{1 - R^2_{X_i|X_{-i}}}$$

Thus, the range of a variance inflation factor is $[1, \infty)$, where 1 indicates no collinearity/that the variable cannot be linearly predicted using only the other available variables. A cut-off of 5 or 10 is generally considered indicative of "a problematic amount of collinearity" (ibid.).

Table 5.2 shows the variance inflation factors of all of the gathered features under a one day lag. As should be expected, the 12 and 26-day EMAs and MACD line have very large variance inflation factors, since these are linearly related in that $MACD(t) = EMA_{12}(p_c, t) - EMA_{26}(p_c, t)$. However, aside from these, there is a clearly a large degree of multicollinearity in the gathered features with just a one day lag: almost certainly the extent of multicollinearity will increase when a 1–5 day lag window is used. The only features with a multicollinearity less than 5 are tweet counts and a BERTweet negative probability feature with average aggregation and retweet weighting, which (from above) have correlation coefficients with the target variable of 0.02 and 0.04, respectively. Hence, there is strong evidence of redundancy in these features, and prediction may be vastly improved if feature selection and/or dimensionality reduction techniques are applied in order to reduce the level of multicollinearity.

Feature (all lagged by one day)	VIF
EMA of closing price (12-day)	∞
MACD of closing price (MACD line)	∞
EMA of closing price (26-day)	3002399751580330.50
Price (open)	1444.63
Price (high)	1365.80
Price (close)	1156.24
Price (low)	1142.90
BERTweet (neutral probability, average, no weighting)	175.44
BERTweet (positive probability, sum, no weighting)	82.54
BERTweet (neutral probability, sum, no weighting)	79.93
MACD of closing price (signal line)	53.03
On balance volume	47.64
BERTweet (negative probability, average, no weighting)	38.09
BERTweet (negative probability, sum, no weighting)	27.89
BERTweet (positive probability, average, no weighting)	25.21
BERTweet (neutral probability, sum, retweet-weighted)	19.59
BERTweet (positive probability, sum, retweet-weighted)	10.75
VADER (sum)	8.16
BERTweet (negative probability, sum, retweet-weighted)	7.94
BERTweet (neutral probability, average, retweet-weighted)	7.11
BERTweet (compound 3, sum, no weighting)	6.40
BERTweet (positive probability, average, retweet-weighted)	6.11
Tweet counts	4.76
BERTweet (negative probability, average, retweet-weighted)	1.68

Table 5.2: Variance inflation factors among one day-lagged features for `$AAPL` stock over the full data-set period

5.2.2 Principal component analysis

Principal component analysis (PCA) is a transformation which maps data (or features, in this case) from one coordinate system to a new coordinate system with axes of *principal components* (PCs). These principal components are computed such that, successively, the mapped data has maximal variance and is completely uncorrelated with all prior principal components (Jolliffe 1990). That is, PCA eliminates multicollinearity, but this comes at a cost of interpretability: data is mapped to the new space through linear transformations over all features/variables, and hence there is no trivial mapping between the original features and the features in the new space.

PCA is often utilised as a dimensionality reduction technique by choosing to keep only the first k principal components (dimensions in the new space) and discarding the rest. Since the variance of these principal components is maximal for the first principal component and decreasing, choosing even a fairly small k can 'explain' most of the variance in the data; in this sense, PCA is said to 'summarise' the data, and thus is often used as a feature extraction technique — particularly in the context of support vector

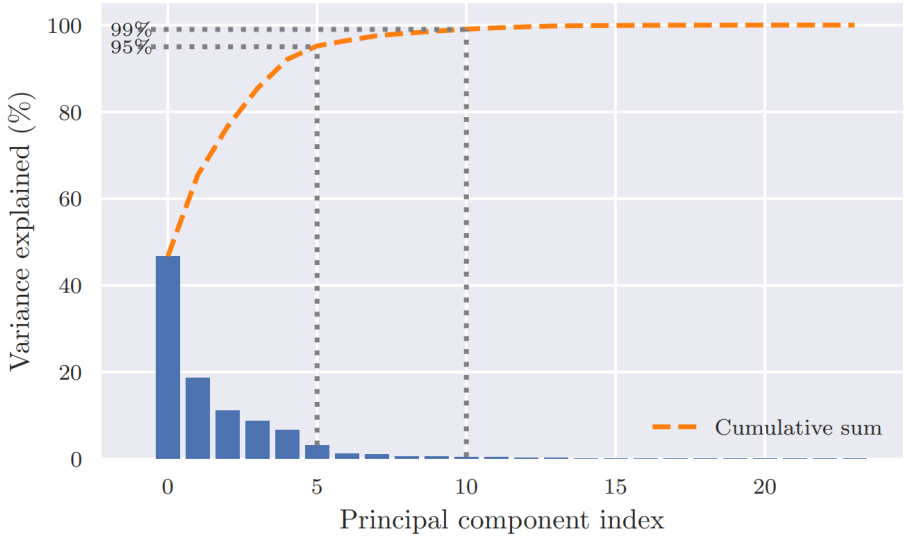

Figure 5.8: Variance explained by each principle component after PCA of one day-lagged features for `$AAPL` stock over the full data-set period. The first six (indices are zero-indexed) principle components explain 95% of variance in the original data.

machines and time series forecasting (Cao and Chong 2002; Chowdhury, Chakravarty, Hossain, et al. 2018).

I employ PCA as a feature extraction technique for models which do not perform feature selection intrinsically (namely, SVM), in order to eliminate multicollinearity and reduce redundancy and dimensionality. Rather than set the number of principal components to keep explicitly, I set a minimum proportion of the variance which must be 'explained' by the selected number of principal components. This is because a different number of principal components may be needed to explain the same amount of variance for different stocks, and hence the approach is more consistent and generalisable by choosing a proportion.

Figure 5.8 shows the variance explained by each principal component after applying PCA to the 24 gathered features lagged by one day for `$AAPL` stock. Since there was a high degree of redundancy among the gathered features, only very few principal components are necessary to explain most of the variance in the original data: 6 PCs can explain 95% of the variance, and 11 PCs explain 99% of the variance. Thus, PCA works well in this context to extract a smaller number of features while retaining most information from the original set of features.

5.2.3 Recursive feature elimination

Some machine learning models have a notion of 'feature importance' — a value associated with each feature which describes how important it is to the model in making predictions. For instance, in linear regression, (assuming that all features are of the same

scale) coefficients in the linear model can provide a crude feature importance value. **Recursive feature elimination** fits a model first with all features, then recursively, the least important features are discarded and the model is re-fitted, until a desired number of features is reached. In principle, this should mitigate multicollinearity to a large extent since two correlated features are unlikely to be equally important in prediction. Additionally, noise and dimensionality are reduced since unimportant features are removed. However, there are important choices to be made in the number of features to keep, and the model to use.

Setting the number of features to keep directly is a difficult choice and may be over-fit to a particular time period/stock. A better approach is to evaluate the performance of the model from which feature importances are being extracted after *each* discarding of features/re-fitting, and choose the number of features which yielded the best performance. To obtain a good estimate of how the model would perform on an unseen test set, cross-validation is often used (a discussion of cross-validation is given in Section 6.1.1). This approach is known as recursive feature elimination with cross-validation.

Random forest (RF) models are a good candidate for extracting feature importance from due to their ease of use as a 'black box' model with good performance, and low variance due to their use of random subsets of features in training each decision tree in the forest. There are two main RF feature importance measures in common use: Mean Decrease Impurity (MDI), which is based on the gains made in the decision trees by splitting on a particular feature; and permutation importance, which permutes occurrences of a feature in the test set and observes changes in accuracy/error (Scornet 2020). The *Scikit-Learn* Python library implements the former of these two measures.

In this project, I utilise recursive feature elimination with (nested) cross-validation[2] using a RF model, via the *Scikit-Learn* Python library, as a feature selection method. The number of trees in the RF is fixed at 200, which was chosen as a balance between accuracy and speed. I investigate this method in addition to PCA for trend prediction tasks, and **in place of** PCA for price prediction: on this latter task, it is beneficial not to *transform* features such as price. Using a feature selection approach therefore retains the original space of the features, and simply filters out unhelpful features.

5.3 Feature scaling

The collected features are frequently fundamentally different in scale; for instance, price is likely to be much lower than trading volume. This can cause problems for models which are scale-dependent: features with larger scales can dominate those with smaller scales in calculations such as the inner product between feature vectors, and cause numerical issues in the model (Hsu, Chang, Lin, et al. 2003). Models with regularization are also negatively impacted, as varying scales necessitate varying scales of coefficients/weights of each feature. Additionally, scaling features to have zero mean leads to faster convergence in the case of neural network models using back-propagation (Y. A. LeCun et al. 2012).

[2]See Section 5.5 for discussions on nested cross-validation

I used **standard scaling** (also known as 'z-score normalisation'), defined as:

$$x_i^{(n)} = \frac{x_i^{(n)} - \bar{\mathbf{x}}^{(n)}}{\sigma(\mathbf{x}^{(n)})}$$

where $x_i^{(n)}$ is the n-th component in the $\mathbf{x}_i$ feature (row) vector, and $\mathbf{x}^{(n)}$ is the column vector of the n-th component in all feature vectors in the training set.

Standard scaling is frequently used in practice, and some research has validated its use in classification tasks (D. Singh and B. Singh 2020). Further, since this method produces zero mean and unit variance, it is suitable for use in models that perform better with these characteristics, such as support vector machines or linear classifiers/regressors. Therefore, standard scaling was used for all models in this work. Scaling parameters are determined from the training set, and used to transform both the training set and the test set, so as to not utilise unseen information in training and optimistically bias the results (information leakage).

In the context of non-stationary time series analysis, the validity of standard scaling is questionable. Since it scales to the mean of features in the training set (and means of non-stationary time series shift with time), the transformed test set can have a vastly different distribution to the training set, leading to poor performance or violating expectations (such as zero mean) of the model.

Several approaches have been proposed or utilised in lieu of standard scaling in the context of financial time series analysis. Bollen, Mao, and Zeng (2011) assumed a local mean and standard deviation within a sliding window centered around a point, and performed z-score normalisation with respect to this; Jin et al. (2017) utilised a similar approach, but with a trailing sliding window. Passalis, Tefas, et al. (2019) and Passalis, Kanniainen, et al. (2021) propose approaches which adaptively normalise time series data, in contrast to the scaling approaches described above which use fixed parameters (e.g. the mean and standard deviation in standard scaling). Regrettably, these approaches were not utilised in my project due to time constraints and realising the issues of standard scaling for non-stationary time series late into development.

5.4 Models

In this section, I describe each of the models applied in the experiments, and the reasoning behind their use. A range of models are utilised in order to thoroughly investigate the question of the predictive power of social media data, irrespectively of the limitations of any individual model.

5.4.1 Baseline models

Constant

This model outputs the most frequent class label in the training set, and is only used for trend prediction. This has the dual effect of demonstrating the balance of the data-set while providing a reasonable baseline which must be beaten by the model if it is to be of any use.

Naïve

The Naïve model predicts that the future value will be equal to the last seen observation. This baseline is used in the Makridakis competitions for time series forecasting (Makridakis, Spiliotis, and Assimakopoulos 2020), and proved to be a strong baseline against machine learning methods in the M4 competition.

For multi-horizon predictions, this model assumes that the value is the same for every day on the horizon (i.e. a straight horizontal line).

5.4.2 Statistical models

I employ both ARIMA and exponential smoothing models in order to provide a strong baseline of statistical models to compare the machine learning models against. These models are used extensively in practice (Goodwin et al. 2010; Hyndman and Athanasopoulos 2018) and academia (Makridakis and Hibon 2000; Makridakis, Spiliotis, and Assimakopoulos 2020), and hence should provide a very strong baseline.

ARIMA

The autoregressive integrated moving average (ARIMA) model is characterised by three components (with parameters p, d, q):

- Autoregressive component, parameter p: assumes that the time series can be modelled by a linear regression of its own past values; p determines the number of past values (lag window) to use in this regression.
- Integrated component, parameter d: since non-stationary time series have a changing mean, it is beneficial to stabilise this mean via differencing the time series. d determines the degree of differencing, e.g. $d = 1 \Rightarrow y'_t = y_t - y_{t-1}$; $d = 2 \Rightarrow y''_t = y'_t - y'_{t-1}$, etc. Rather than using the raw values of the time series in the autoregression, these differenced values are used instead. Hence, a differenced value is forecast; to convert this back to the original time series, the forecast value is *integrated* (i.e. $y_{t+1} = y'_{t+1} + y_t$)
- Moving average component, parameter q: a moving average model assumes that a time series is a linear combination of q past and the current forecasting errors, which are assumed to be normally distributed. I.e. $y_t = c + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \ldots + \theta_q\epsilon_{t-q}$ where θ_i are fitted coefficients, ϵ_t is the forecasting error for trading day t, and c is a constant (Hyndman and Athanasopoulos 2018).

ARIMA is thus a combination of an autoregressive model and a moving average model, applied to a differenced version of the input time series.

Stationarity can be tested for with statistical tests such as the Augmented Dickey-Fuller (ADF) test (Dickey and Fuller 1979) and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test (Kwiatkowski et al. 1992). Hence, the appropriate degree of differencing d can be determined by using these tests under varying degrees of differencing. Table 5.3 shows the result of these tests on a single stock price time series for illustration.

As proposed by Box and Jenkins (1970), the remaining p and q terms can be determined by looking at plots of autocorrelation and partial autocorrelation (autocorrelation is the correlation between the time series and its own lagged values). The exact details of this are beyond the scope of this project and omitted.

These processes for determining p, q, and d underlie the 'auto ARIMA' model. In the experiments, I use this, in addition to a **manually-tuned ARIMA(6,1,0) model**, which was chosen from the results of Table 5.3 and some minor experimentation (examining error rates with different parameters).

The ARIMA models are fitted to the unlagged closing price of the stock to be predicted. For determining trend predictions, the fitted ARIMA model is used to first predict price up to the target horizon, and then the trend is computed from this (see Section 3.2).

d	ADF		KPSS	
	Value	p-value	Value	p-value
0	0.559	0.987	2.04	**0.01**
1	-16.1	**5e-29**	0.238	0.1
2	-7.01	**7e-10**	0.102	0.1

Table 5.3: Stationarity tests for `$AAPL` stock prices over the data-set period. p-values $\leq 1\%$ are emboldened. Note that the null hypotheses of the two tests are opposite: in ADF, the null hypothesis indicates that the series is non-stationary.

Exponential smoothing

While stock prices in general exhibit some form of seasonality, this seasonality is annual in frequency and often manifests as increased prices in December–January (Wachtel 1942; M. N. Gultekin and N. B. Gultekin 1983). Thus, with a data-set of only one year, I use exponential smoothing methods which do *not* incorporate seasonality components (since the seasonality cannot be inferred from such a short period), but which do incorporate trend (since stock prices do generally exhibit an upwards or downwards trend).

Holt proposed an additive trend method, which can be described mathematically as follows (Hyndman and Athanasopoulos 2018):

$$\begin{aligned}\hat{y}_{t+h} &= \ell_t + h \cdot b_t \\ \ell_t &= \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta(\ell_t - \ell_{t-1}) + (1-\beta) b_{t-1}\end{aligned}$$

Pegels (1969) proposed another method with a multiplicative trend, which is potentially useful in the case of stock prices since it accounts for trends which are non-linear. This is as follows:

$$\begin{aligned}\hat{y}_{t+h} &= \ell_t \cdot h \cdot b_t \\ \ell_t &= \alpha y_t + (1-\alpha)(\ell_{t-1} \cdot b_{t-1}) \\ b_t &= \beta\left(\frac{\ell_t}{\ell_{t-1}}\right) + (1-\beta) b_{t-1}\end{aligned}$$

Gardner Jr and McKenzie (1985) proposed an improvement on Holt's original method to add 'damping' to the trend component in an attempt to address overshooting exhibited by the additive trend at longer horizons. J. W. Taylor (2003) subsequently applied this to the multiplicative trend ES model. Both cases are summarised by the following equations:

$$\hat{y}_{t+h} = \ell_t * (h \cdot (b_t \diamond \sum_{i=1}^{h} \phi^i))$$
$$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} * (b_{t-1} \diamond \phi))$$
$$b_t = \beta(\ell_t *^{-1} \ell_{t-1}) + (1-\beta) b_{t-1} \diamond \phi$$

where $*$ is addition for additive trends and multiplication for multiplicative trends; $*^{-1}$ is the inverse of $*$ (i.e. subtraction and division); and $\diamond$ is multiplication for additive trends and exponentiation for multiplicative trends.

The exponential smoothing model used in the experiments is a combination of all of the above approaches, treating the type of trend component (additive or multiplicative) and whether to dampen the trend as hyper-parameters.

As with ARIMA, the model is fitted to the unlagged adjusted closing prices of the stock, and used to compute price forecasts; trend is then computed from these price forecasts in the case of stock trend prediction.

5.4.3 Machine learning models

For each machine learning model, the stock price and trend prediction problems are converted into supervised learning problems as follows:

1. An input feature vector $\mathbf{x}_t$ is formed for each trading day, using all 24 features described in Section 5.1 for each day in a 1–5 trading day (inclusive) lag window, forming in total a vector with 120 dimensions.

2. Input feature vectors are pre-processed with scaling (Section 5.3) and selection/extraction procedures (Section 5.2); the exact pre-processing used (and in which order) is described in Section 5.6.

3. Output vectors/target labels $\mathbf{y}_t / y_t$ are paired with each input feature vector. This is the unlagged adjusted closing price for each trading day in the multi-horizon stock price prediction, and the computed trend (1 or 0) for stock trend prediction. E.g. for a 5-day (multi-)horizon:

$$\mathbf{y}_t = \begin{bmatrix} p_c(t) & p_c(t+1) & p_c(t+2) & p_c(t+3) & p_c(t+4) \end{bmatrix} \quad \text{for price prediction}$$
$$y_t = Trend(t-1, 5) \quad \text{for trend prediction}$$

In general, the price prediction problem is modelled as a regression problem, while the trend prediction problem is modelled as a classification problem.

Of the discussed machine learning models, only CNN-RNN natively supports multi-horizon forecasting. For the remaining models, multi-horizon forecasts are achieved by training a separate instance of the model for each horizon within the multi-horizon.

Support vector machine

The support vector machine (SVM) is a highly robust classification model. SVM works by fitting a hyperplane in feature space to provided training data, such that the classes of training data-points are separated by the hyperplane. This hyperplane is chosen with a maximum *margin*, so that the two classes are maximally distant from the decision boundary.

Since generally it will not be possible to construct a hyperplane which perfectly separates the data, some misclassifications in training are allowed for in construction of the hyperplane. To control the amount of permissible misclassifications, a hyper-parameter C is introduced, where a lower value of C allows for more misclassifications, and a higher value allows for fewer. However, a higher value of C produces a narrower margin and hence may lead to over-fitting.

This approach allows only for a linear decision function in feature space. To extend this, non-linear *kernels* can be used to transform the feature space into an alternative space in which the transformed data-points may be more easily separated by a (linear) hyperplane. The so-called 'kernel trick' allows this transformation to be performed implicitly, and instead the kernel need only define distances between two data-points in feature space, i.e. a function $k(\mathbf{x}, \mathbf{x}')$ (Burges 1998).

For trend prediction, I utilise one of the following three kernels for prediction (the choice of which is treated as a hyper-parameter):

- Linear: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$ — i.e. no transformation.
- Radial basis function: $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$ — a popular non-linear kernel; γ forms a hyper-parameter.
- Sigmoid: $k(\mathbf{x}, \mathbf{x}') = \tanh(\gamma(\mathbf{x} \cdot \mathbf{x}') + r)$ — another popular kernel, inspired by sigmoid activation functions in neural networks; γ and r are hyper-parameters.

The same ideas of SVM can be applied in regression, leading to support vector regression (SVR) (Smola and Schölkopf 2004). In SVR, a hyperplane $\mathbf{w} \cdot \mathbf{x} + b = \hat{y}$ is fitted to the training data, such that the error of the training examples is at most ϵ. As with SVM, slack is allowed for training examples which exceed this ϵ, and the total allowed slack is controlled with C. ϵ forms an additional hyper-parameter.

Although I implemented an SVR model with RFE, this proved far too slow to train (taking 30 minutes to train once for a single stock, which would need to be done 75 times for 66 stocks).

Random forest

A decision tree is a tree of decision nodes, which eventually lead to a leaf node of class label or value. A single decision tree may be constructed for classification or regression, but is prone to over-fitting. Random forests (Breiman 2001) are an ensemble of decision trees, which make use of bootstrap aggregating — a technique whereby the training set is sampled with replacement to produce new training sets; each decision tree is then trained with one of these new training sets, and the outputs are aggregated by voting (classification) or averaging (regression). In addition, random forests use

random subsets of input features to grow each tree, which reduces over-dependence on a single feature and improves accuracy (Breiman 2001).

In constructing decision trees, random forests 'split' the training set at each decision node; the feature used to split is chosen based on how well the split would separate classes (classification) or reduce error (regression), and a local optima of this scoring function is used to choose the value on which to split ('cut-point').

I use random forests for a number of reasons:

- they are resilient to the issues of multicollinearity discussed in Section 5.2
- they are parallelizable and fast to train
- they offer good performance out-of-the-box with minimal hyper-parameters

Random forests do have one notable hyper-parameter in the number of trees T to construct. In general, higher T offers greater accuracy, but at increased computational expense (Probst and Boulesteix 2017). Hence, I fix T at 1,000 trees which seemed to be the limit of feasible study with the available computational resources and time.

Extra-trees

Extremely randomised ('extra') trees (Geurts, Ernst, and Wehenkel 2006) is an variation of random forest, which instead trains each tree using the whole training set and randomly chooses splits in decision tree nodes. That is, a random cut-point is generated for each feature uniformly between the feature's minimum and maximum value in the training set, and the best-scoring random split (feature and cut-point pair) is then selected.

The benefits of extra-trees are:

- Lower risk of over-fitting (or more accurately, lower variance), due to randomizing both feature subsets used in each tree *and* cut-points.
- Faster training time, due to no local optimisation of cut-points: Geurts, Ernst, and Wehenkel (ibid.) found that training times reduced on average by 64% for classification, and 19% for regression.

Hence, I used extra-trees in an attempt to see whether an increase in accuracy/decrease in error could be obtained over random forests with lower computational costs. Like random forest, I used 1,000 trees in this model.

Gradient-boosted decision trees

'Boosting' refers to a family of algorithms whereby weak learners (such as decision trees) are combined algorithmically into a 'strong' learner (Freund, Schapire, and Abe 1999). Gradient boosting (Natekin and Knoll 2013) iteratively combines weak learners, such that each iteration produces a new weak learner which aims to correct the errors of its predecessor. That is, the weak learner attempts to minimise loss in predicting the residuals from the previous iteration. In gradient-boosted decision trees (GBDT), the weak learner is the decision tree.

I use the exponential loss function $\exp(-y\,\hat{y})$ in the case of classification (this is equivalent to the 'Adaboost' model), and the mean squared error in regression. Empirically, I found these to yield the best accuracy/error rates (tested on a few stocks).

The number of iterations to perform is a hyper-parameter of the GBDT model; I set this at 1,000, for equivalent reasons as in random forest.

ROCKET-Ridge

As introduced in Section 2.2, ROCKET (Dempster, Petitjean, and Webb 2020) transforms time series with random convolutional kernels. This method achieved state-of-the-art performance in time series classification and is extremely fast to run, and hence I employ it here. While I did attempt to use this technique for stock price prediction in addition to trend prediction, the error rate proved excessively high, and hence I focused instead on using it for trend prediction/time series classification, as the authors intended.

I apply this transformation to the input feature vectors (as a multivariate time series) as a feature extraction technique. Then, the transformed features are fed into a *ridge classifier*, following the recommendations of the authors of ROCKET. Ridge classification is an extension of ridge regression, which is itself a linear regression technique for estimating regression coefficients when input features are highly correlated (Hilt and Seegrist 1977); the extension of this technique to classification simply involves mapping class labels to $\{-1, 1\}$, and taking the sign of the predicted value as the predicted class label.

Ridge classification/regression involves a single hyper-parameter, α, which controls the strength of the penalty applied to the large coefficients[3]; this penalty is a regularization technique, which intends to prevent over-fitting. I tune α using a grid search method, as will be discussed in Section 5.5.

CNN-RNN

Taking inspiration from the successful time series classification algorithms of ROCKET (Dempster, Petitjean, and Webb 2020) and HIVE-COTE (Lines, S. Taylor, and Bagnall 2016), I developed a neural network model with convolutional layers for feature extraction. To process the resulting time series of extracted features, I use a (recurrent) long short-term memory layer (LSTM) (Hochreiter and Schmidhuber 1997). LSTMs have been utilised frequently to reasonable success for time series forecasting and classification due to their ability to process sequences and handle long-term dependencies (Siami-Namini, Tavakoli, and Namin 2018; Karim et al. 2017), such as an event occurring several days before a resultant change in stock price. While gated recurrent units (Cho et al. 2014) have been proposed as a means to reduce training time compared to LSTM, I did not find a substantial enough reduction to justify their use, and hence LSTMs are used in this model instead.

After encoding the extracted features with an LSTM layer, the final hidden state is given as the initial state for a *decoder* LSTM layer, which serves to output a sequence

[3] More specifically, this is the strength of L_2 regularisation

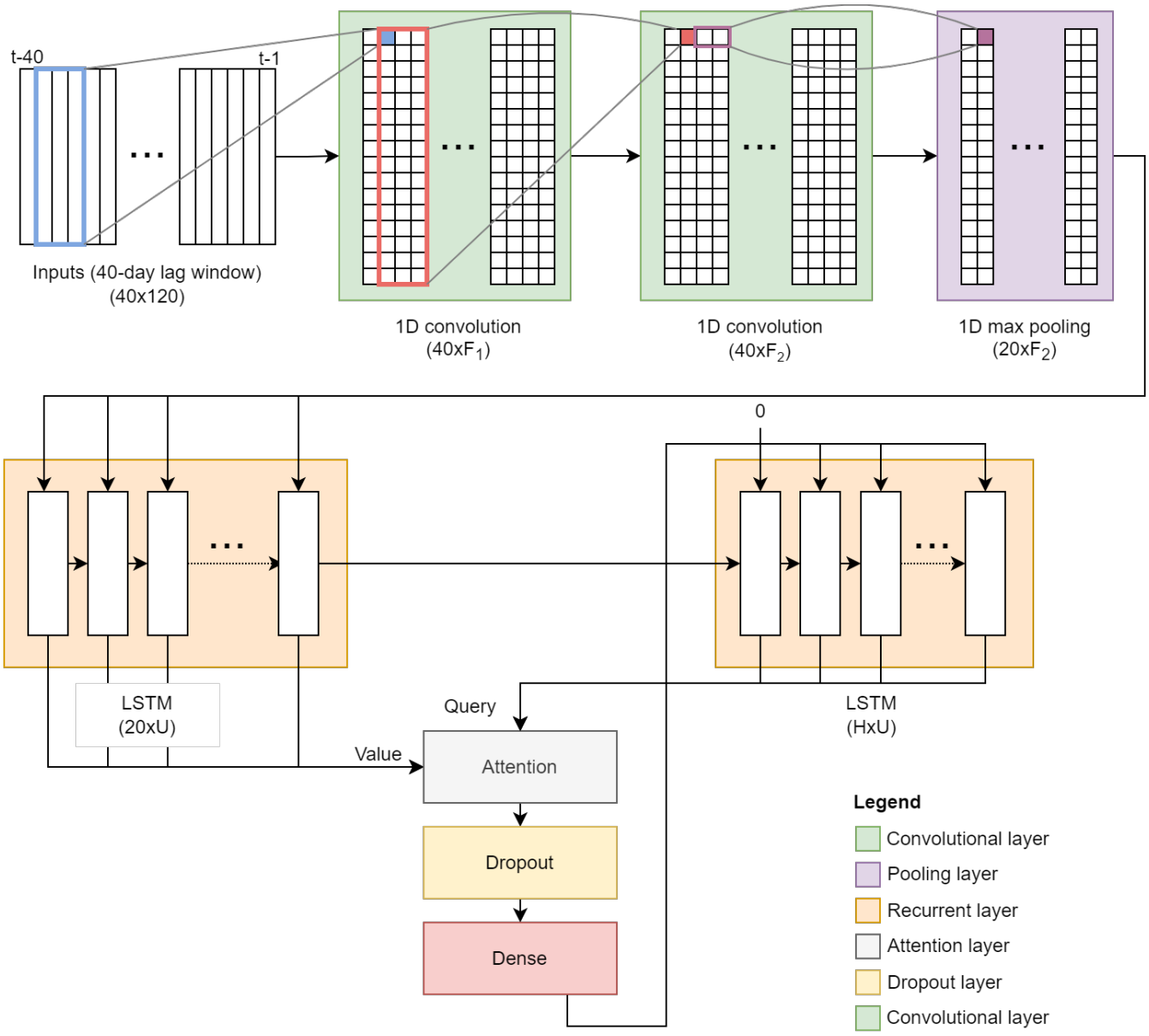

Figure 5.9: Architecture of the developed CNN-RNN model; H is the horizon of the given task, U is a hyper-parameter of the number of units in each LSTM cell, and F_1/F_2 are hyper-parameters of the number of filters to use in the convolutions.

for multi-horizon forecasting. To decode its input, the decoder layer's output is recurrently fed through prediction layers to produce a price forecast, and this is used as input to the next LSTM cell. In the case of classification and next-day horizon forecasting, only one LSTM cell is used.

To avoid the problem of *information bottleneck*, whereby the network is overly reliant on information from the entire time series being encoded in the final LSTM encoder cell, I use an attention layer (Luong, Pham, and Manning 2015) with the encoder layer outputs and the current decoder layer cell output as inputs. This has the effect of reducing the bottleneck, by allowing the network to selectively pay attention to all time-steps in the encoded time series, rather than just the final encoded time-step.

To decrease training time, I make use of *teacher forcing* (Williams and Zipser 1989): in training, the decoder layer's inputs are given from a lagged version of the intended output sequence. E.g. the first LSTM decoder cell is given $p_c(t-1)$ as an input, where the intended output is $p_c(t)$.

Dropout is added before the final dense decoder layer and within the LSTM layers for similar reasons as in multi-layer perceptron: namely, to prevent over-fitting. As the complexity of a network increases, so too does the risk of over-fitting. Hence, very aggressive dropout is used to discourage overfitting.

In contrast to the other machine learning models presented, this model uses **40 trading days** of lagged input (i.e. each input sample is a matrix of input feature vectors $\mathbf{x}_{t-39}, \mathbf{x}_{t-38}, \ldots, \mathbf{x}_t$). This is because the network is designed to handle sequential data, whereas I felt introducing such a large window to other models would impose too great a computational cost, introduce irredeemable amounts of multicollinearity, and not greatly benefit predictions (since the other ML models are not designed to handle sequential data).

The hyper-parameters of the model are summarised in Table 5.4. Although I implemented a hyper-tuning procedure for this model, I found it to be infeasible due to the long training time inherent in such a large network. Therefore, I set the hyper-parameters manually, after extensive informal experimentation. It should be noted that these hyper-parameters will likely be sub-optimal, due to the small hyper-parameter space examined.

Generally, large neural networks require many epochs to converge. Hence, I use 2000 epochs in trend prediction. In price prediction, I lower this to 250, since this uses day forward-chaining (see Section 6.1.1), and thus involves re-training the model 75 times[4] rather than 5 times, per stock.

[4] 30% * 252 trading days

Hyper-parameter	Trend prediction	Price prediction
Convolutional layer 1 filters	256	256
Convolutional layer 1 filter size	3	3
Convolutional layer 2 filters	256	256
Convolutional layer 2 filter size	5	5
Convolutional layer padding	Zero-padded	Zero-padded
Convolutional layer activation	ReLU	ReLU
Max pooling filter size	2	2
Max pooling filter stride	2	2
Max pooling padding	None	None
Encoder LSTM units	256	256
Decoder LSTM units	256	256
Dense layer units	1	1
Dense layer activation	Sigmoid	Linear
Dropout	0.5	0.5
Learning rate	3e-4	5e-3
Epochs	2000	250
Loss function	Binary cross-entropy	Mean squared error
Optimizer	Adam	Adam

Table 5.4: Hyper-parameters used in the CNN-RNN model; these were not tuned due to the computational expense/time that a suitable training procedure would require.

5.5 Hyperparameter tuning

Hyper-parameter choice is extremely important for model performance. Therefore, I utilise hyper-parameter tuning techniques where possible. These techniques involve evaluating the performance of the model under different hyper-parameter settings.

I use *nested cross validation* to obtain an estimate of the model's performance with a given set of hyper-parameter values, whereby cross validation is applied to the current fold of an outer cross validation procedure (this outer procedure is described in Section 6.1.1). Specifically, I use 5-fold time series cross validation, where the (reduced) training set[5] is divided into 5 folds and the model is trained separately for each fold; the scores (accuracy or mean squared error) of each fold are then averaged to produce an estimate of the model's performance on unseen data. Successive folds are supersets of the folds prior, reflecting the real-world usage of the model in which it would be updated as new data becomes available. In *time series* cross validation, it is important to retain the chronology of the data-set, so that future data is not used in training to predict values occurring in the past; hence, the training set is not shuffled. Five folds are chosen such that the time spent tuning hyper-parameters is feasible.

Nested cross validation avoids issues with bias in obtaining estimates of model performance (Varma and Simon 2006; Cawley and Talbot 2010), and hence improves the choice of hyper-parameter selection.

In the following sub-sections, the hyper-parameter tuning procedures applied to the

[5] i.e. the training set allocated to the current outer fold

models are described, in addition to the specific models for which they are applied and the range of values for each hyper-parameter used.

5.5.1 Grid search

Grid search is the simplest hyper-parameter tuning algorithm, and simply involves a brute-force search over some space of hyper-parameters. I use this approach in models for which it is feasible to do so. These models, and the set of hyper-parameters used (the combinations evaluated are the Cartesian product of these sets) are as follows:

- Exponential smoothing
  - Trend: $\{\text{Additive}, \text{Multiplicative}, \text{None}\}$
  - Trend dampening: $\{\text{True}, \text{False}\}$ (not used when trend is None)
- Support vector machine (trend prediction)
  - C: $\text{logspace}(-1, 2, 50)$
  - Kernel: $\{\text{Sigmoid}, \text{RBF}, \text{Linear}\}$
  - γ: $\{\frac{1}{120}, \frac{1}{120\text{Var}(\mathbf{x}_t))}\}$
- Support vector machine (price prediction)
  - C: $\text{logspace}(-2, 2, 50)$
  - Kernel: fixed at Linear[6]
  - ϵ: $\text{logspace}(-3, -1, 25)$
- ROCKET-Ridge
  - α: $\text{logspace}(-3, 3, 40)$

where the $\text{logspace}(a, b, n)$ function returns n logarithmically-spaced values in $[10^a, 10^b]$.

5.5.2 Hyperband tuning

Neural network models are in general expensive to train. This fact combined with the large hyper-parameter spaces makes grid search generally infeasible. A common approach is to use Bayesian Optimization (Shahriari et al. 2015), which models the function yielding model performance given hyper-parameter settings probabilistically. However, I opted instead to use Hyperband (Li et al. 2017), which has been shown to be around 5×–30× faster than Bayesian Optimization.

[6] I initially investigated RBF and Sigmoid kernels additionally, however this proved far too slow for the size of C that was requried to yield adequate results. Scikit-Learn provides a separate linear SVM implementation which was far faster than the implementation supporting other kernels, hence I limited the kernel to linear.

Hyperband is an extension of the 'successive halving' algorithm (Jamieson and Talwalkar 2016), which successively discards the worst-performing half of a set of hyper-parameter configurations and allocates exponentially more of a 'resource' to the evaluations of subsequent halves. In neural network hyper-parameter tuning, this resource is the number of epochs (and therefore training time). The set of hyper-parameter configurations is derived by uniformly sampling the hyper-parameter space, and (this is the contribution of Hyperband) the size of the set is varied to yield different trade-offs of number of configurations vs. average resources used per configuration (where the same finite budget of epochs is allocated to each set).

The hyper-parameter spaces used in the neural network models are as follows: (unless stated otherwise, these spaces are sampled logarithmically)

- CNN-RNN[7]
  - Number of convolutional layers: $[1, 3] \in \mathbb{N}$ (sampled linearly)
  - Convolutional layer i number of filters: $[8, 512] \in \mathbb{N}$
  - Convolutional layer i filter size: $\{2n+1 \mid n \in \mathbb{N} \wedge 0 \leq n \leq 5\}$
  - Encoder/decoder LSTM units: $[8, 512] \in \mathbb{N}$
  - Learning rate: $[10^{-4}, 10^{-1}] \in \mathbb{R}$
  - Dropout: $[0, 0.4] \in \mathbb{R}$

5.6 Summary of models

Type	Model	Pre-processing	Hyper-parameter tuning
Baseline	N/A	N/A	N/A
	Naïve	N/A	N/A
Statistical	ARIMA	-	-
	Exponential smoothing	-	Grid search
ML	SVM	Standard scaling → RFE or PCA (trend pred.), RFE (price pred.)	Grid search
	Random forest	Standard scaling	-
	Extra-trees	Standard scaling	-
	GBDT	Standard scaling	-
	ROCKET-Ridge	ROCKET → Standard scaling	Grid search
	CNN-RNN	Standard scaling	Hyperband/manual[8]

[7] This tuning procedure was not used in the experiments, since I did not have sufficient computational resources/time to perform it for all stocks/horizons under consideration; however, it is given here for completeness.

[8] Implemented Hyperband tuning for this instance, but manual was used in the experiments due to resource constraints

Chapter 6

Experiments

6.1 Experimental design

6.1.1 Evaluation

In line with standard machine learning practice, the data-set is split into training and test sets; the performance is evaluated on the test sets, and these performances are averaged in order to produce an estimate of how well the model would generalise to unseen/real-world data.

As the data-set is a time series, care is taken not to shuffle the data-set, which could produce an unfair estimate by incorporating data from the future in training. However, this means that conventional techniques such as hold-out (where the data-set is split into a single training set and test set, often at a ratio of 70:30) would most likely be unsuitable, because the time period covered in testing would be excessively short (due to the small 252 trading day data-set): e.g. consider the case where a company has a good period of profits, and stock price rises consistently for a month or two; then, a model always predicting 'up' could obtain reasonable performance estimates.

To address this issue, I use time series k-fold cross-validation for trend prediction, which has a much greater temporal coverage than a hold-out method, and is generally well-suited for smaller data-sets. Day-forward chaining with a small initial training set size may have been better suited (e.g. a 30:70 ratio instead of a 70:30 ratio), however this was beyond the realm of feasibility in the resource constraints[1].

Price prediction does not suffer from the same issues of temporal coverage, since this task is generally difficult regardless (and especially so with multi-horizon forecasting). Hence, in this case I use day forward-chaining (sometimes referred to as 'walk-forward validation') with a 70:30 split, which more accurately reflects practical usage (i.e. in practice, the model would be trained daily).

Note that all pre-processing steps described are fitted to each training set, and used to transform the test set; e.g. with standard scaling, the mean is computed from the current test set under consideration. Through this method, information from the test

[1] $70\% \cdot 252 \cdot 188$ stocks $\cdot$ 3 minutes to train one fold for all classification models per stock = 69 days

set remains unseen, and does not leak into any stage of the model training. Likewise, nested cross-validation is used in hyper-parameter tuning; 5-fold time series cross-validation is used to ensure the smallest training set used in hyper-parameter tuning is sufficiently large and to keep time spent tuning/training to a minimum.

Time series k-fold cross-validation

Time series cross-validation splits the data-set into k training and test sets such that each training set is a superset of the last, as shown in Figure 6.1. This enables a range of training set sizes to be evaluated, and hence a more robust estimate of performance is obtained.

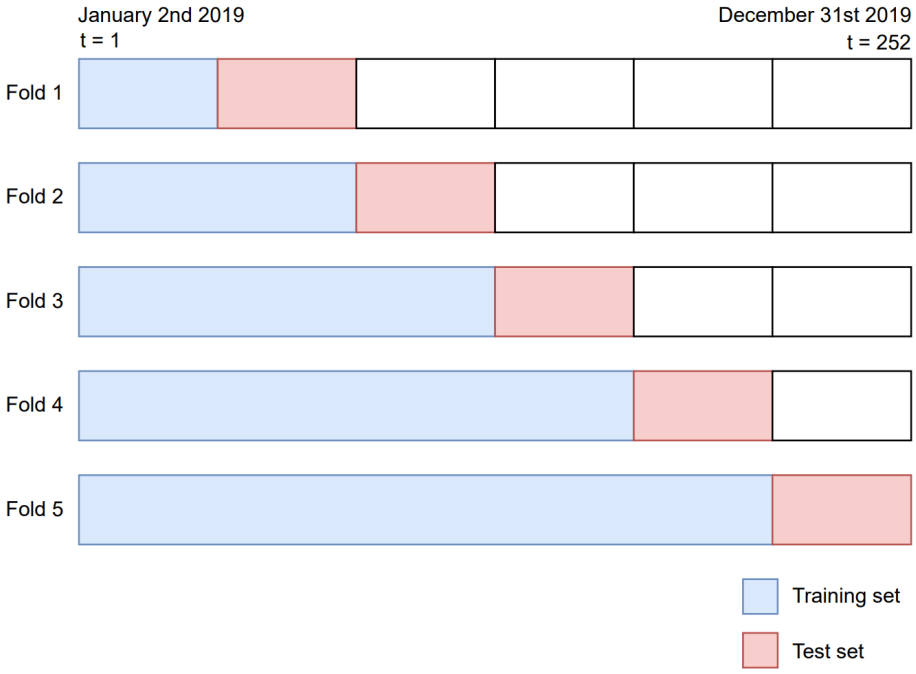

Figure 6.1: Illustration of time series cross-validation

I select $k = 5$ to balance training time and the minimum size of the training set (noting that CNN-RNN requires at least 40 days, and there are 252 days in the entire data-set).

Day forward-chaining

Day forward-chaining is another cross-validation technique (Hyndman and Athanasopoulos (2018) refer to this as 'time series cross-validation', though it is also referred to as 'day forward-chaining' (Aminanto et al. 2020) or 'walk-forward validation' or 'rolling forecast' (Siami-Namini, Tavakoli, and Namin 2018)), whereby a minimum

training set size is selected and the test set is the single day following this training set; then, the training set is expanded by one day and this process repeats until there are no days left.

6.1.2 Stock selection

Using the entire 4,612 stock data-set would inevitably yield poor trend and prediction results, as not every stock is discussed on Twitter, and nor are they discussed in equal amounts. However, limiting the stock selection to some threshold number of tweets also does not make sense, as volume of tweets does not necessarily imply predictive power (e.g. many tweets could be uninformative noise). Moreover, since a symbol in a tweet may refer to a stock or cryptocurrency (with no good way of determining which), the vast majority of tweets for a given symbol could be completely uncorrelated with the stock price.

To more fairly and usefully select stocks, I designed an experiment whereby the Pearson correlation is computed between all *sentiment* features within a $[1,5]$-day lag window, and the stock closing price. The maximum correlation is then extracted and assigned to the stock, and only stocks with a best correlation above some threshold are used in the prediction experiments. Thus, a useful estimate of performance of the models is obtained (since a user/trader is unlikely to use the models for stocks which it evidently is unlikely to work for, and it is unrealistic to expect the models to work for every stock when the data is simply inadequate).

6.1.3 Evaluation metrics

To obtain a numerical estimate of performance, I use a variety of metrics which are commonly used in regression, classification, and time series forecasting. These are detailed below.

Trend prediction

Accuracy Proportion of correct classifications; this is not well suited to unbalanced classes, however I provide the Constant model as a baseline for comparison. This metric also allows comparison to related works, which generally use accuracy.

Matthews correlation coefficient (MCC) A metric suited to unbalanced classes, which takes into account all elements of the confusion matrix (true positives, true negatives, etc.). It is defined as:

$$\mathrm{MCC} = \frac{\mathrm{TP}\cdot\mathrm{TN} - \mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})\cdot(\mathrm{TP}+\mathrm{FN})\cdot(\mathrm{TN}+\mathrm{FP})\cdot(\mathrm{TN}+\mathrm{FN})}}$$

I use this in preference to F1 score, as F1 does not take into account true negatives at all, which should be of equal importance to true positives in a stock trend prediction task. Chicco and Jurman (2020) argue in favour of using MCC over F1 and accuracy in binary classification tasks in general.

Price prediction

Symmetric mean absolute percentage error (sMAPE) While mean absolute percentage error (MAPE) is commonly used as a forecasting error metric, it is asymmetric, "favoring estimates that are below actual values" (Armstrong 1985). The symmetric MAPE addresses this issue, and was used in the M3 and M4 competitions (Makridakis and Hibon 2000; Makridakis, Spiliotis, and Assimakopoulos 2020). It is commonly defined as:

$$\text{sMAPE} = 100 \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{|y_i| + |\hat{y}_i|}$$

although note that a factor of $\frac{1}{2}$ is omitted from the denominator (contrary to Armstrong's definition), to ensure the values are between 0 to 100%.

Relative squared error (RSE) Used as an alternative to the root mean square error, the RSE is comparable across multiple stocks: root mean square error (and mean absolute error) are dependent on the scale of the data, and thus since stock prices vary wildly between stocks, it is nonsensical to compute an average of these metrics over many stocks. The relative squared error accounts for varying scales, and is defined as (Botchkarev 2018):

$$\text{RSE} = \sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{(y_i - \bar{y})^2}$$

Mean absolute scaled error (MASE) Used in M4 and propsed as the "standard measure for forecast accuracy" by Hyndman and Koehler (2006), the MASE computes the forecasting error relative to the in-sample naïve forecast. It is defined as:

$$\text{MASE} = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{\frac{1}{m-1} \sum_{j=2}^{m} |y'_j - y'_{j-1}|}$$

where y'_j are the true *in-sample* (i.e. training) price values

6.2 Results

6.2.1 Stock sentiment correlations

Figures 6.2–6.4 summarise the results of this experiment, which was run on all 4,612 stocks. In total, 66 stocks had $r \geq 0.5$, and 188 stocks had $r \geq 0.4$. As can be seen from Figure 6.2, the spread of correlation values is quite large even at the same number of tweets, thus supporting the use of using a threshold based on r rather than the number of tweets.

Figure 6.3 shows that BERTweet was the most correlated feature marginally more often than VADER or Tweet count. However, the proportion of each sentiment feature type is fairly similar, supporting the value of using complementary feature types rather than focusing on a single sentiment analysis approach.

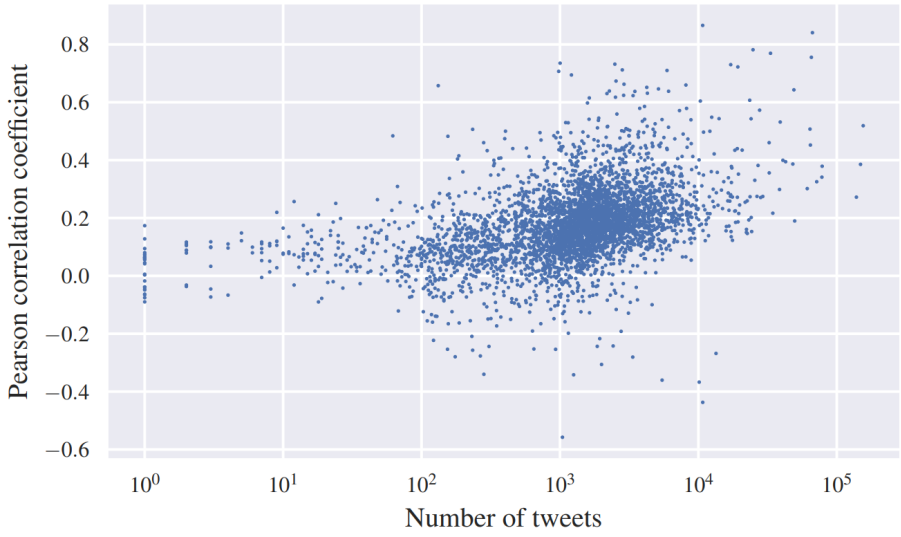

Figure 6.2: Highest Pearson correlation coefficients between any sentiment feature and stock closing price for all stocks

Figure 6.4 shows that the 1 and 5-trading day lagged sentiment features were generally the best correlated with the closing price. This supports the idea of using a lag window; however, it remains an open question as to whether 5 trading days was an sufficiently large window.

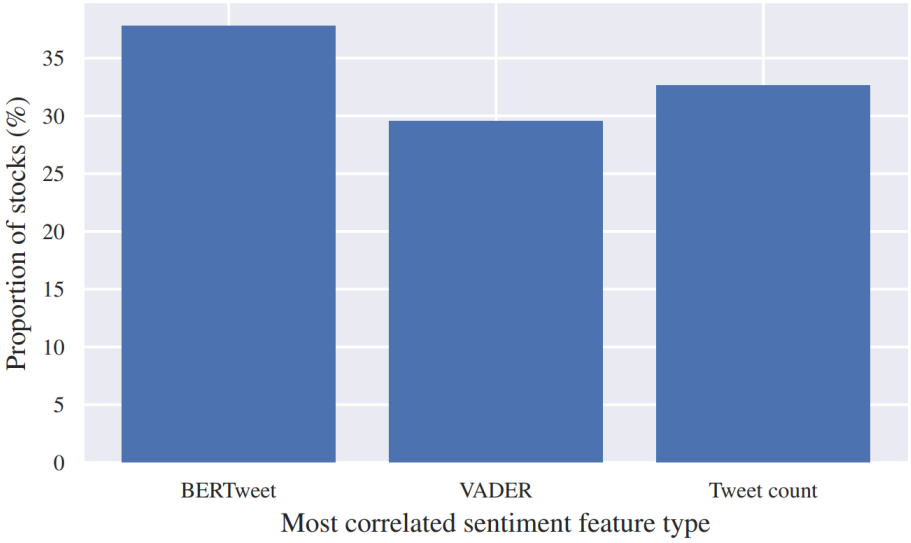

Figure 6.3: Distribution of the most correlated sentiment feature type over the stocks in the data-set

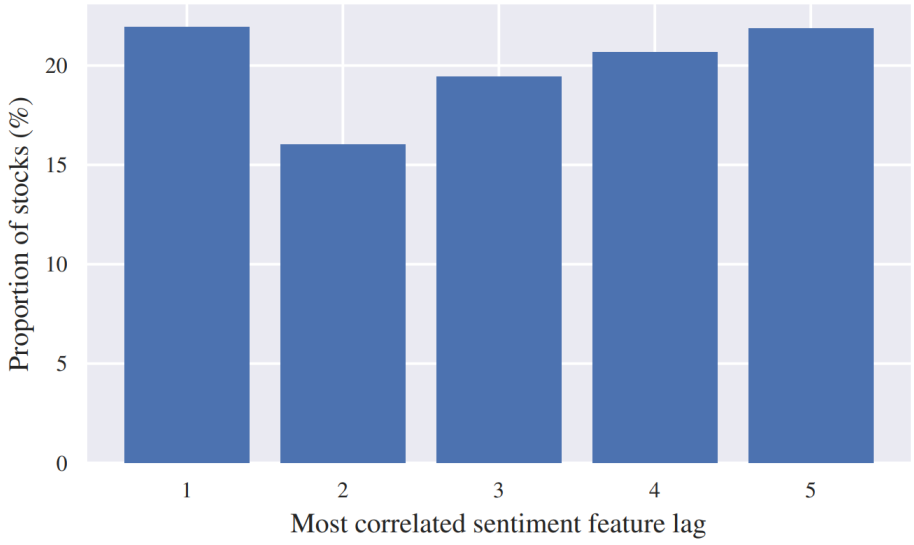

Figure 6.4: Distribution of the most correlated sentiment feature lag (in trading days) over the stocks in the data-set

6.2.2 Stock trend prediction

The threshold of $r \geq 0.4$ was chosen to determine stocks to test, yielding 188 stocks; this was chosen based on computational expense and practicability.

Table 6.1 summarises the main results of the trend prediction experiment. All models failed to out-perform the Constant baseline accuracy on the $H = 1$ and $H = 5$ horizons, however best accuracies of 58.1% and 73.9% were obtained by CNN-RNN and Extra-Trees on the $H = 20$ and $H = 65$ horizons. Although almost all machine learning models out-performed statistical models in terms of accuracy for all horizons, ARIMA models consistently had the highest MCC scores, and hence these predictions were likely of higher quality on average (Chicco and Jurman 2020).

Omissions: CNN-RNN is omitted for $H = 65$ due to exhausting the Google Cloud Platform free trial credits.

Model	$H = 1$		$H = 5$		$H = 20$		$H = 65$	
	Acc.	MCC	Acc.	MCC	Acc.	MCC	Acc.	MCC
Constant	**52.7**	0.000	**53.7**	0.000	56.3	0.000	66.3	0.000
Naïve	49.0	−0.055	50.0	−0.046	50.3	−0.069	50.3	−0.051
ARIMA(6,1,0)	49.8	**0.096**	48.6	**0.175**	47.7	**0.236**	46.8	0.135
Auto-ARIMA	50.2	0.087	49.3	0.166	48.6	0.223	46.5	**0.136**
Exp. smoothing	50.2	0.094	49.1	0.169	48.2	0.234	45.7	0.131
SVM (PCA)	51.1	0.012	51.6	0.029	54.3	0.058	65.1	0.014
SVM (RFE)	50.8	0.030	50.7	0.060	53.6	0.106	67.6	0.049
Random forest	50.7	0.058	51.4	0.107	56.9	0.170	73.2	0.087
Extra-Trees	50.7	0.045	51.3	0.107	57.5	0.173	**73.9**	0.075
GBDT	50.0	0.034	50.5	0.074	56.1	0.144	71.7	0.092
ROCKET-Ridge	50.4	0.037	51.4	0.078	55.8	0.129	68.2	0.051
CNN-RNN	50.6	0.027	51.4	0.060	**58.1**	0.093	-	-

Table 6.1: Main results of the stock trend prediction experiment, with mean scores over 188 stocks (those with $r \geq 0.4$). Bold indicates highest scoring model.

6.2.3 Stock price prediction

The threshold of $r \geq 0.5$ was chosen for this experiment, yielding 66 stocks. This higher threshold was chosen due to the greater computational expense of multi-horizon forecasting, particularly under models where one instance of the model was trained per horizon.

Table 6.2 summarises the results of the experiment. In general, prediction quality was poor; no models were able to achieve lower error rates than the Naïve baseline at the 1-trading-day horizon, and no machine learning models beat the performance of statistical models at $H = 1$ or $H = 5$. However, all but one machine learning model obtained a substantial reduction in error over the statistical models at the $H = 20$ multi-horizon, suggesting that these models may be more useful for month or longer multi-horizon forecasting.

Model	$H = 1$			$H = 5$			$H = 20$		
	sMAPE	RSE	MASE	sMAPE	RSE	MASE	sMAPE	RSE	MASE
Naïve	**2.63**	**0.114**	**1.35**	4.64	0.359	2.41	9.65	1.507	4.21
ARIMA(6, 1, 0)	2.79	0.126	1.49	**3.16**	**0.239**	**1.82**	8.21	1.320	3.94
Auto-ARIMA	2.81	0.127	1.46	3.48	0.279	1.88	9.36	1.507	4.25
Exp. smoothing	2.67	0.121	1.42	4.68	0.367	2.44	9.84	1.665	4.38
Random forest	3.72	0.245	1.92	4.99	0.497	2.39	7.02	1.295	2.95
Extra-Trees	3.44	0.189	1.74	4.21	0.281	2.01	**5.22**	**0.478**	**2.23**
GBDT	3.85	0.313	1.89	4.95	0.546	2.29	6.62	1.241	2.78
CNN-RNN	-	-	-	5.55	0.650	2.98	8.46	2.027	4.11

Table 6.2: Main results of the stock price prediction experiment, with mean scores over 66 stocks.

Omissions: Support vector regression was not attempted for all stocks, due to lack of resources. $H = 65$ was not investigated due to insufficient resources and data (this would be just 9 days in the test set). CNN-RNN was not used for $H = 1$ due to running out of Google Cloud Platform credit.

6.3 Discussion and evaluation

ML models performed better at longer horizons At the 20 and 65-trading-day horizons, the machine learning models achieved excellent accuracies of 58.1% (1.8% better than Constant; 9.5% better than any statistical model) and 73.9%; generally, 56% is reported as a satisfying result (T. H. Nguyen, Shirai, and Velcin 2015). At such long horizons, random fluctuations and noise in stock prices are dwarfed by the overall stock trend. Hence, models should achieve better accuracies at longer horizons. The higher accuracy of ML models over statistical models at longer horizons is likely due to the inherent simplicity of statistical models at long horizons: e.g. ARIMA with 1 degree of differencing tends to a straight line or constant at long horizons (Hyndman and Athanasopoulos 2018), but ML models can represent far more complex trends. However, statistical models achieved higher MCCs at every horizon, and this metric should perhaps be the point of focus, rather than accuracy (Chicco and Jurman 2020).

Ensemble methods out-perform individual ML methods As evidenced by the success of HIVE-COTE (Lines, S. Taylor, and Bagnall 2016), ensemble methods generally perform well in time series classification tasks. Ensemble models generally perform well under noise due to the aggregation of votes/predictions, and hence are well-suited to stock trend and price prediction, where noise is inherent in both the stock market and social media sentiment data. Additionally, the ensemble methods used required no hyper-parameter tuning, and therefore issues of over-fitting in this process were circumvented — contrary to ROCKET-Ridge and SVM, which were generally out-performed

by random forest.

Comparison with results of M4 M4 (Makridakis, Spiliotis, and Assimakopoulos 2020) found that 4/5 machine learning models studied were unable to exceed the performance of the naïve forecast. At a 1-trading-day horizon, this proved to be the case, however in stock price prediction, the ML models consistently out-performed Naïve at the 5 and 20 trading-day horizons (excluding CNN-RNN, which performed poorly in general). Performance did not come close at any horizon to the best-performing method in M4 (a hybrid method), which had a MASE of 1.54; this seems to reaffirm the superiority of hybrid methods to ML methods in general.

Poor performance of CNN-RNN in price prediction The CNN-RNN model was a complex model with over one million parameters. Additionally, aggressive dropout was applied in the network. Consequently, a large number of epochs would be required for the network to converge. From the results of the price prediction experiment, 250 epochs was decidedly too low in this instance. I would expect performance to greatly improve with a number such as 2,000 epochs, however given that day forward-chaining with a 70:30 split was used (each model is trained $252 \times 30\% = 75$ times), this was unfortunately outside the realm of feasibility.

Poor sentiment correlations In general, sentiment correlations were quite poor (see Figure 6.2). A large reason in this was likely the discarding of tweet data on non-trading days, which may have had very valuable information for prediction. Additionally, a more intelligent aggregation of tweet sentiments to trading day sentiments would likely yield better correlation and prediction results. Finally, although single-symbol tweets comprised a majority of the data-set (Figure 5.4), a aspect-sentiment based approach would have enabled the use of multi-symbol tweets, perhaps improving correlations and prediction results.

Interpretability A potentially important factor of stock trend/price prediction models is interpretability: investors/traders would greatly benefit from understanding *why* a prediction model is telling them to invest in or sell a particular stock. Regrettably, this property is lacking in the developed models (with the minor exception of SVM with a linear kernel), and hence this forms a limitation of this work.

Chapter 7

Conclusions

Stock market prediction is, overall, a challenging problem. In this work, I have presented the data collection, methods, and experimental evaluation in attempting to harness social media data for this task. Stock discussion on social media was investigated on a large scale, with 4,612 stocks analysed, and 9M tweets collected. State-of-the-art sentiment analysis techniques were used, and a variety of features were extracted borrowing from both technical analysis and sentiment analysis stock prediction approaches. A range of ML models were used to provide a comprehensive comparison to traditional statistical models such as ARIMA and exponential smoothing, and a carefully designed experimentation was presented to estimate the performance of these models in practice and attempt to test the EMH.

With trend prediction accuracies scarcely exceeding 50%, and price prediction errors exceeding that of the Naive model at the 1 and 5 trading-day horizons, there is some evidence to support the Efficient Market Hypothesis. However, further investigation is needed to firmly rule out the value of social media data. Additionally, satisfying results were achieved at the 20 trading-day and longer horizons with the developed machine learning models, which is not strictly consistent with the Efficient Market Hypothesis.

Machine learning models did not out-perform statistical models in trend or price prediction, save for a minor advantage in 20 trading-day price prediction. As Lim and Zohren (2021) point out in their excellent survey of time series forecasting with deep learning, the likely causes of this are a proclivity to over-fit — which is especially so with a reasonably small data-set of 252 trading days — and the potential sensitivity of ML models to feature scaling, as discussed in Section 6.3. Overall, the results obtained are in line with the findings of M4, and Hyndman and Athanasopoulos (2018)'s observation that "some forecasting methods are extremely simple and surprisingly effective".

There were several limitations in this work, which could be addressed in future work. Firstly, sentiment data on non-trading-days was discarded, which likely had a strong negative impact on prediction performance; to remedy this, better trading day alignment strategies could be employed, such as those used by Xu and Cohen (2018), Makrehchi, Shah, and Liao (2013), and Sprenger et al. (2014). Secondly, tweets with

more than one symbol mention were discarded, due to the nature of the sentiment analysis approach used. While T. H. Nguyen, Shirai, and Velcin (2015) investigated a aspect-sentiment approach, BERT could be investigated for topic or aspect-based sentiment analysis in the context of stock prediction with social media. Thirdly, standard scaling of features was used, which likely impacted the performance of the ML models due to the inherent non-stationarity in stock market prices (Section 5.3). Passalis, Tefas, et al. (2019) and Passalis, Kanniainen, et al. (2021) propose alternative scaling approaches designed for time series data, which may yield far improved stock prediction performance. Finally, although the maximum amount of data was collected given the time constraints, the 252 trading days of data was potentially insufficient, particularly for neural network models with many parameters such as CNN-RNN. To remedy this, a longer time period could be studied, or the use of higher-frequency data (such as hourly stock price data) could be investigated.

As M4 demonstrated, hybrid models are the future of time series analysis. More research is needed to investigate the use of hybrid models for stock price/trend prediction; this could incorporate both the power of traditional techniques such as exponential smoothing, and the complexity of machine learning techniques such as RNNs.

Bibliography

Aminanto, Muhamad Erza et al. (2020). "Threat alert prioritization using isolation forest and stacked auto encoder with day-forward-chaining analysis". In: *IEEE Access* 8, pp. 217977–217986.

Antweiler, Werner and Murray Z. Frank (2004). "Is All That Talk Just Noise? The Information Content of Internet Stock Message Boards". In: *The Journal of Finance* 59.3, pp. 1259–1294. ISSN: 00221082, 15406261. URL: http://www.jstor.org/stable/3694736.

Armstrong, J Scott (1985). *Long-Range Forecasting: From Crystal Ball to Computer*. 2nd ed. Wiley, p. 348.

Asur, Sitaram and Bernardo A Huberman (2010). "Predicting the future with social media". In: *2010 IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology*. Vol. 1. IEEE, pp. 492–499.

Bagnall, Anthony et al. (2015). "Time-series classification with COTE: the collective of transformation-based ensembles". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9, pp. 2522–2535.

Baker, Scott R et al. (July 2020). "The Unprecedented Stock Market Reaction to COVID-19". In: *The Review of Asset Pricing Studies* 10.4, pp. 742–758. ISSN: 2045-9920. DOI: 10.1093/rapstu/raaa008. eprint: https://academic.oup.com/raps/article-pdf/10/4/742/34416840/raaa008.pdf. URL: https://doi.org/10.1093/rapstu/raaa008.

Berry, William D, Stanley Feldman, and Dr Stanley Feldman (1985). *Multiple regression in practice*. 50. Sage.

Blei, David M, Andrew Y Ng, and Michael I Jordan (2003). "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan, pp. 993–1022.

Bollen, Johan, Huina Mao, and Alberto Pepe (2011). "Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena". In: *Proceedings of the international AAAI conference on web and social media*. Vol. 5. 1, pp. 450–453.

Bollen, Johan, Huina Mao, and Xiaojun Zeng (2011). "Twitter mood predicts the stock market". In: *Journal of computational science* 2.1, pp. 1–8.

Bonsón, Enrique, David Perea, and Michaela Bednárová (2019). "Twitter as a tool for citizen engagement: An empirical study of the Andalusian municipalities". In: *Government Information Quarterly* 36.3, pp. 480–489.

Botchkarev, Alexei (2018). "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology". In: *arXiv preprint arXiv:1809.03006*.

Box, George E. P. and Gwilym M. Jenkins (1970). *Time Series Analysis: Forecasting and Control*. Holden-Day.

Bozanta, Aysun et al. (2021). "Sentiment Analysis of StockTwits Using Transformer Models". In: *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, pp. 1253–1258.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Brown, Robert G (1957). "Exponential smoothing for predicting demand". In: *Operations Research*. Vol. 5. 1. INST OPERATIONS RESEARCH MANAGEMENT SCIENCES 901 ELKRIDGE LANDING RD, STE . . ., pp. 145–145.

Burges, Christopher JC (1998). "A tutorial on support vector machines for pattern recognition". In: *Data mining and knowledge discovery* 2.2, pp. 121–167.

Cao, LJ and WK Chong (2002). "Feature extraction in support vector machine: a comparison of PCA, XPCA and ICA". In: *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.* Vol. 2. IEEE, pp. 1001–1005.

Carosia, AEO, Guilherme Palermo Coelho, and AEA Silva (2020). "Analyzing the Brazilian financial market through Portuguese sentiment analysis in social media". In: *Applied Artificial Intelligence* 34.1, pp. 1–19.

Cawley, Gavin C and Nicola LC Talbot (2010). "On over-fitting in model selection and subsequent selection bias in performance evaluation". In: *The Journal of Machine Learning Research* 11, pp. 2079–2107.

Chicco, Davide and Giuseppe Jurman (2020). "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC genomics* 21.1, pp. 1–13.

Cho, Kyunghyun et al. (2014). "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259*.

Chowdhury, Utpala Nanda, Sanjoy Kumar Chakravarty, Md Tanvir Hossain, et al. (2018). "Short-term financial time series forecasting integrating principal component analysis and independent component analysis with support vector regression". In: *Journal of Computer and Communications* 6.03, p. 51.

Dempster, Angus, François Petitjean, and Geoffrey I Webb (2020). "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels". In: *Data Mining and Knowledge Discovery* 34.5, pp. 1454–1495.

Derakhshan, Ali and Hamid Beigy (2019). "Sentiment analysis on stock social media for stock price movement prediction". In: *Engineering Applications of Artificial Intelligence* 85, pp. 569–578.

Devlin, Jacob et al. (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805*.

Di Persio, Luca and Oleksandr Honchar (2017). "Recurrent neural networks approach to the financial forecast of Google assets". In: *International journal of Mathematics and Computers in simulation* 11, pp. 7–13.

Dickey, David A and Wayne A Fuller (1979). "Distribution of the estimators for autoregressive time series with a unit root". In: *Journal of the American statistical association* 74.366a, pp. 427–431.

Fama, Eugene F. et al. (1969). "The Adjustment of Stock Prices to New Information". In: *International Economic Review* 10.1, pp. 1–21. ISSN: 00206598, 14682354. URL: http://www.jstor.org/stable/2525569.

Fan, Chenyou et al. (2019). "Multi-horizon time series forecasting with temporal attention learning". In: *Proceedings of the 25th ACM SIGKDD International conference on knowledge discovery & data mining*, pp. 2527–2535.

Freund, Yoav, Robert Schapire, and Naoki Abe (1999). "A short introduction to boosting". In: *Journal-Japanese Society For Artificial Intelligence* 14.771-780, p. 1612.

Gardner Jr, Everette S and ED McKenzie (1985). "Forecasting trends in time series". In: *Management science* 31.10, pp. 1237–1246.

Geurts, Pierre, Damien Ernst, and Louis Wehenkel (2006). "Extremely randomized trees". In: *Machine learning* 63.1, pp. 3–42.

Gilbert, Eric and Karrie Karahalios (2010). "Widespread worry and the stock market". In: *Fourth International AAAI Conference on Weblogs and Social Media*.

Goodwin, Paul et al. (2010). "The holt-winters approach to exponential smoothing: 50 years old and going strong". In: *Foresight* 19.19, pp. 30–33.

Granville, Joseph Ensign (1963). *New key to stock market profits*. Prentice-Hall.

Gruhl, Daniel et al. (2005). "The predictive power of online chatter". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 78–87.

Gultekin, Mustafa N and N Bulent Gultekin (1983). "Stock market seasonality: International evidence". In: *Journal of financial economics* 12.4, pp. 469–481.

Hannun, Awni et al. (2014). "Deep speech: Scaling up end-to-end speech recognition". In: *arXiv preprint arXiv:1412.5567*.

Hilt, Donald E and Donald W Seegrist (1977). *Ridge, a computer program for calculating ridge regression estimates*. Vol. 236. Department of Agriculture, Forest Service, Northeastern Forest Experiment ...

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Holt, Charles C (1957). "Forecasting Trends and Seasonal by Exponentially Weighted Averages". In: *Office of Naval Research Memorandum* 52. Reprinted in (Charles C. Holt 2004).

— (2004). "Forecasting seasonals and trends by exponentially weighted moving averages". In: *International Journal of Forecasting* 20.1, pp. 5–10. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2003.09.015. URL: https://www.sciencedirect.com/science/article/pii/S0169207003001134.

Hsu, Chih-Wei, Chih-Chung Chang, Chih-Jen Lin, et al. (2003). *A practical guide to support vector classification*.

Hutto, Clayton and Eric Gilbert (2014). "Vader: A parsimonious rule-based model for sentiment analysis of social media text". In: *Proceedings of the international AAAI conference on web and social media*. Vol. 8. 1, pp. 216–225.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts. URL: OTexts.com/fpp2.

Hyndman, Rob J and Anne B Koehler (2006). "Another look at measures of forecast accuracy". In: *International journal of forecasting* 22.4, pp. 679–688.

Ismail Fawaz, Hassan et al. (2020). “Inceptiontime: Finding alexnet for time series classification”. In: *Data Mining and Knowledge Discovery* 34.6, pp. 1936–1962.
James, Gareth et al. (2021). *An introduction to statistical learning*. 2nd ed. Springer, p. 102.
Jamieson, Kevin and Ameet Talwalkar (2016). “Non-stochastic best arm identification and hyperparameter optimization”. In: *Artificial intelligence and statistics*. PMLR, pp. 240–248.
Jiang, Haoming et al. (2019). “Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization”. In: *arXiv preprint arXiv:1911.03437*.
Jin, Fang et al. (2017). “Tracking multiple social media for stock market event prediction”. In: *Industrial conference on data mining*. Springer, pp. 16–30.
Jolliffe, Ian T (1990). “Principal component analysis: a beginner’s guide—I. Introduction and application”. In: *Weather* 45.10, pp. 375–382.
Karim, Fazle et al. (2017). “LSTM fully convolutional networks for time series classification”. In: *IEEE access* 6, pp. 1662–1669.
Karpoff, Jonathan M (1987). “The relation between price changes and trading volume: A survey”. In: *Journal of Financial and quantitative Analysis* 22.1, pp. 109–126.
Kaufman, Perry J (2003). *A short course in technical trading*. John Wiley & Sons.
Kwiatkowski, Denis et al. (1992). “Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?” In: *Journal of econometrics* 54.1-3, pp. 159–178.
LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.
LeCun, Yann A et al. (2012). “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.
Lee, Chien-Cheng, Zhongjian Gao, and Chun-Li Tsai (2020). “BERT-Based Stock Market Sentiment Analysis”. In: *2020 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan)*. IEEE, pp. 1–2.
Li, Lisha et al. (2017). “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *The Journal of Machine Learning Research* 18.1, pp. 6765–6816.
Lim, Bryan, Sercan O Arik, et al. (2019). “Temporal fusion transformers for interpretable multi-horizon time series forecasting”. In: *arXiv preprint arXiv:1912.09363*.
Lim, Bryan and Stefan Zohren (2021). “Time-series forecasting with deep learning: a survey”. In: *Philosophical Transactions of the Royal Society A* 379.2194, p. 20200209.
Lines, Jason, Sarah Taylor, and Anthony Bagnall (2016). “Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification”. In: *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, pp. 1041–1046.
Liu, Yinhan et al. (2019). “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692*.
Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). “Effective approaches to attention-based neural machine translation”. In: *arXiv preprint arXiv:1508.04025*.
Makrehchi, Masoud, Sameena Shah, and Wenhui Liao (2013). “Stock prediction using event-based sentiment analysis”. In: *2013 IEEE/WIC/ACM International Joint*

Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). Vol. 1. IEEE, pp. 337–342.

Makridakis, Spyros and Michele Hibon (2000). "The M3-Competition: results, conclusions and implications". In: *International journal of forecasting* 16.4, pp. 451–476.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2020). "The M4 Competition: 100,000 time series and 61 forecasting methods". In: *International Journal of Forecasting* 36.1, pp. 54–74.

Malkiel, Burton G. (1989). "Efficient Market Hypothesis". In: *Finance*. Ed. by John Eatwell, Murray Milgate, and Peter Newman. London: Palgrave Macmillan UK, pp. 127–134. ISBN: 978-1-349-20213-3. DOI: 10.1007/978-1-349-20213-3_13. URL: https://doi.org/10.1007/978-1-349-20213-3_13.

Mazur, Mieszko, Man Dang, and Miguel Vega (2021). "COVID-19 and the march 2020 stock market crash. Evidence from S&P1500". In: *Finance Research Letters* 38, p. 101690. ISSN: 1544-6123. DOI: https://doi.org/10.1016/j.frl.2020.101690. URL: https://www.sciencedirect.com/science/article/pii/S1544612320306668.

Natekin, Alexey and Alois Knoll (2013). "Gradient boosting machines, a tutorial". In: *Frontiers in neurorobotics* 7, p. 21.

Nguyen, Dat Quoc, Thanh Vu, and Anh Tuan Nguyen (2020). "BERTweet: A pre-trained language model for English Tweets". In: *arXiv preprint arXiv:2005.10200*.

Nguyen, Thien Hai, Kiyoaki Shirai, and Julien Velcin (2015). "Sentiment analysis on social media for stock movement prediction". In: *Expert Systems with Applications* 42.24, pp. 9603–9611.

Noriega, Leonardo (2005). "Multilayer perceptron tutorial". In: *School of Computing. Staffordshire University*.

Nuzhath, Tasmiah et al. (2020). "COVID-19 vaccination hesitancy, misinformation and conspiracy theories on social media: A content analysis of Twitter data". In.

Otter, Daniel W, Julian R Medina, and Jugal K Kalita (2020). "A survey of the usages of deep learning for natural language processing". In: *IEEE transactions on neural networks and learning systems* 32.2, pp. 604–624.

Passalis, Nikolaos, Juho Kanniainen, et al. (2021). "Forecasting Financial Time Series Using Robust Deep Adaptive Input Normalization". In: *Journal of Signal Processing Systems* 93.10, pp. 1235–1251.

Passalis, Nikolaos, Anastasios Tefas, et al. (2019). "Deep adaptive input normalization for time series forecasting". In: *IEEE transactions on neural networks and learning systems* 31.9, pp. 3760–3765.

Pegels, C Carl (1969). "Exponential forecasting: Some new variations". In: *Management Science*, pp. 311–315.

Pérez, Juan Manuel, Juan Carlos Giudici, and Franco Luque (2021). "pysentimiento: A python toolkit for sentiment analysis and socialnlp tasks". In: *arXiv preprint arXiv:2106.09462*.

Porshnev, Alexander, Ilya Redkin, and Alexey Shevchenko (2013). "Machine learning in prediction of stock market indicators based on historical data and data from twitter sentiment analysis". In: *2013 IEEE 13th International Conference on Data Mining Workshops*. IEEE, pp. 440–444.

Probst, Philipp and Anne-Laure Boulesteix (2017). "To tune or not to tune the number of trees in random forest." In: *J. Mach. Learn. Res.* 18.1, pp. 6673–6690.

Rangapuram, Syama Sundar et al. (2018). "Deep state space models for time series forecasting". In: *Advances in neural information processing systems* 31.

Rosenthal, Sara, Noura Farra, and Preslav Nakov (2017). "SemEval-2017 task 4: Sentiment analysis in Twitter". In: *Proceedings of the 11th international workshop on semantic evaluation (SemEval-2017)*, pp. 502–518.

Salinas, David et al. (2020). "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: *International Journal of Forecasting* 36.3, pp. 1181–1191.

Sammut, Claude and Geoffrey I Webb (2011). *Encyclopedia of machine learning*. Springer Science & Business Media, p. 402.

Sawhney, Ramit et al. (2020). "Deep attentive learning for stock movement prediction from social media text and company correlations". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8415–8426.

Scornet, Erwan (2020). "Trees, forests, and impurity-based variable importance". In: *arXiv preprint arXiv:2001.04295*.

Shahriari, Bobak et al. (2015). "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1, pp. 148–175.

Siami-Namini, Sima, Neda Tavakoli, and Akbar Siami Namin (2018). "A comparison of ARIMA and LSTM in forecasting time series". In: *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, pp. 1394–1401.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Singh, Dalwinder and Birmohan Singh (2020). "Investigating the impact of data normalization on classification performance". In: *Applied Soft Computing* 97, p. 105524.

Smola, Alex J and Bernhard Schölkopf (2004). "A tutorial on support vector regression". In: *Statistics and computing* 14.3, pp. 199–222.

Socher, Richard et al. (2013). "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642.

Social Media Fact Sheet (Apr. 2021). Washington, D.C.: Pew Research Center. URL: `https://www.pewresearch.org/internet/fact-sheet/social-media/`.

Sprenger, Timm O et al. (2014). "Tweets and trades: The information content of stock microblogs". In: *European Financial Management* 20.5, pp. 926–957.

Taigman, Yaniv et al. (2014). "Deepface: Closing the gap to human-level performance in face verification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708.

Taylor, James W (2003). "Exponential smoothing with a damped multiplicative trend". In: *International journal of Forecasting* 19.4, pp. 715–725.

Treloar, Tim (June 2008). "Examining the EMA". In: *Stocks & Commodities* 26.6, pp. 42–46.

Tumarkin, Robert and Robert F. Whitelaw (2001). "News or Noise? Internet Postings and Stock Prices". In: *Financial Analysts Journal* 57.3, pp. 41–51. ISSN: 0015198X. URL: `http://www.jstor.org/stable/4480315`.

Tumasjan, Andranik et al. (2010). "Predicting elections with twitter: What 140 characters reveal about political sentiment". In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 4. 1, pp. 178–185.

Varma, Sudhir and Richard Simon (2006). "Bias in error estimation when using cross-validation for model selection". In: *BMC bioinformatics* 7.1, pp. 1–8.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30.

Veaux, Richard D De and Lyle H Ungar (1994). "Multicollinearity: A tale of two nonparametric regressions". In: *Selecting models from data*. Springer, pp. 393–402.

Vu, Tien Thanh et al. (2012). "An experiment in integrating sentiment features for tech stock prediction in twitter". In: *Proceedings of the workshop on information extraction and entity analytics on social media data*, pp. 23–38.

Wachtel, Sidney B (1942). "Certain observations on seasonal movements in stock prices". In: *The journal of business of the University of Chicago* 15.2, pp. 184–193.

Wang, Alex et al. (2019). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: In the Proceedings of ICLR.

Wen, Ruofeng et al. (2017). "A multi-horizon quantile recurrent forecaster". In: *arXiv preprint arXiv:1711.11053*.

Williams, Ronald J and David Zipser (1989). "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2, pp. 270–280.

Winters, Peter R (1960). "Forecasting sales by exponentially weighted moving averages". In: *Management science* 6.3, pp. 324–342.

Xu, Yumo and Shay B Cohen (2018). "Stock movement prediction from tweets and historical prices". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1970–1979.

Zacharias, Cody (2022). *TWINT - Twitter Intelligence Tool*. URL: `https://github.com/twintproject/twint` (visited on 04/13/2022).