CORRELATING FACTORS OF U.S. PRESIDENTIAL SPEECHES WITH STOCK MARKET MOVEMENTS - A MACHINE LEARNING APPROACH.

Pablo Rees[a]

Thesis presented in fulfilment of the requirements for the degree of Master of Economics in the Faculty of Economics at Stellenbosch University.

Supervisor:
Dr Dawie van Lill

December 2022

[a]**Acknowledgements:**
I would like to thank the Stellenbosch University PSP foundation for sponsorship of my Masters Degree and this project by extension. I would also like to thank Dr Dawie van Lill for the provision his wealth of knowledge and resources in Machine and Deep Learning subject matter without which this project would have been impossible and finally the Stellenbosch Economics department as a whole for it's liberal approach to engagement with students on academic and administrative issues with special mention of Dr Gideon du Randt, Nico Katzke and Carina Smit for support along the way.

Abstract

The literature relating textual to stock market data is deep, but the relationship between speeches given by political figures and stock markets is relatively undefined. This research begins to rectify this by exploring the relationship between U.S. presidential speeches and daily price movements in the S&P 500 index. It was possible to explore this relationship by using natural language processing techniques, econometric time-series analysis, and machine learning models. It was found that models including presidential speech data can achieve prediction accuracy of about 60% over an S&P 500 index price movement proxy. This is an increase of about 0.3% (0.599 vs 0.601) over the models that did not include the presidential speech data (without losing ground in either recall or precision). Notably, this result was drawn from 71 years of data at a daily resolution. Thus, it is concluded that presidential speeches hold predictive power over stock market movements and that this relationship can be used to improve the power of predictive models.

Keywords: S&P 500, Machine Learning, Natural Language Processing, Presidential Speeches, Time Series Econometrics

JEL classification G17

Table of Contents

List of Figures

List of Tables

1. Introduction

The efficient market hypothesis states that the history of a market is an ineffective predictor of its future, nonetheless there are many iterations of attempts to predict markets based on their own histories and external factors (Fama, 1965). Moore's law states that computing power doubles every two years and although this is not strictly true, the lived experience is close (Mollick, 2006). This combined with the onset of massive amounts of publicly available data has given empirical research in the social sciences power that it has never been able to muster until recent years. Computer-driven analysis of large amounts of text data is rapidly becoming mainstream and specifically, political text data is one external factor useful in predicting market futures (Grimmer & Stewart, 2013; Hayo, Kutan & Neuenkirch, 2008; Maligkris, 2017; Soroka, Young & Balmas, 2015).

Hayo *et al.* (2008) performed a generalized auto-regressive conditional heteroscedastic (GARCH) analysis of the relationship between Federal Open Market Committee (FOMC) speeches and the U.S. financial market. The analysis was quantitative on the market side and qualitative on the speech side and demonstrates that the sentiments of communications influence the effect of those communications on markets. Thus, analyzing sentiment is an important factor in an accurate appraisal of the relationship between speeches and markets. Using regression analysis Maligkris (2017) demonstrates that there is a correlation between presidential candidates' speeches and stock movements. Further, they indicate that speeches laden with economic information tend to boost stock returns while also reducing volatility while speeches with a negative tone have the opposite effect and the long run effect of speeches is dependent on market conditions. Baker, Bloom & Davis (2016) show that language processing can be used to predict economic events, and particularly economic uncertainty and economic policy uncertainty. They developed an Economic Policy Uncertainty Index based on the frequency of articles containing a trio of terms in ten leading U.S. newspapers. The terms were selected over a period of 24 months during which more than 15 000 news articles were read by humans in an auditing process. The index proved to be quite accurate, spiking near the expected events, including wars, tight presidential elections, fiscal policy battles and terrorist activity. Sazedj & Tavares (2011) highlight the relationship between the content of a presidential speech and stock market returns by regressing the event of a speech during Obama's first 11 months as U.S. President on the daily excess returns of the Dow Jones, the S&P 500 and the NASDAQ indices. They found that the event of a speech had a generally insignificant effect on daily excesses. However, by regressing key terms contained in 43 speeches given during the first 11 months of Obama's presidency it was found that the content of speeches can significantly affect daily excess returns nearly uniformly across all three indices. Notably, the NASDAQ's correlation to the content of speeches was weaker – indicating that technology markets may be less susceptible to presidential rhetoric.

The aim of this project is to determine whether U.S presidential speeches have predictive power over U.S. stock market movements. A positive result would be powerful and could add to the ability of traders to accurately predict stock market movements. The undertaking of this project assumes two important conjectures – both of which have been confirmed by the research reported above. First, that there is a correlation between the linguistic factors[1] of speech employed by U.S. presidents in their speeches and U.S. stock market movements; and second, that the prominence of speech factors

[1] 'Linguistic factors' in this sense is intended to mean any and all patterns that can be detected in spoken language including verbiage, lexicon, tone, register, sentence length, word combination etc.

can be quantified by using machine learning algorithms. Broadly, machine learning has been selected as the method of modelling and the S&P 500 index has been chosen as representative of the 'stock market'. As far as I am aware, this research is novel.

In specific, this project shows that U.S. presidential speeches have predictive power over S&P 500 price fluctuations. In order to prove this, natural language processing (NLP), machine learning (ML) and time-series analysis were used. The usage of these three methods allowed for the comparison of models based on purely financial data with models based on a combination of financial and language data. The result being that the best models based on a combination of financial and language data outperformed the best purely financial models by about 0.3 percent in terms of predictive accuracy at a daily resolution over ~70 years.

2. Methodology review

In order to select the methods most applicable to this project a review of similar projects was undertaken. Seven papers are reviewed here in order to elucidate commonly used methods in data cleaning, sentiment analysis and stock market prediction. The results are summarized in two tables following the review. Table 2.1 represents the literature review in a condensed format which allows for easy comparison of the data and methods used and resulting accuracies. Table 2.2 represents the metadata, linked through 'Paper number' for Table 2.1.

2.1. Data cleaning methods for NLP

Katre (2019), in his analysis of Indian political speeches, uses the Natural Language Toolkit (NLTK) package and string methods to remove punctuation, HTML tags and English stopwords[2], as well as converting speeches to lowercase and tokenizing[3] them. Zubair & Cios (2015), before correlating the sentiment in Reuters articles with S&P 500 movements, clean their text data by tokenizing it with NLTK. Kinyua, Mutigwe, Cushing & Poggi (2021) clean their Twitter data (tweets from then U.S. President, Donald Trump) by deleting all tweets on days when the stock trading was closed, deleting all tweets that only contained standard stopwords and deleting all tweets that only contained URLs. Khedr, Yaseen & others (2017) tokenize, standardize by converting to lowercase, remove stopwords from, and stem[4] their textual data before processing the abbreviations (replacing abbreviations with the full phrase) and filtering out words that consist of two or less characters.

2.2. Machine learning methods

The following section looks first at the literature informing the sentiment analysis space and then at the literature around stock market prediction in order to determine methods that suit the intersection

[2]'Stopwords' are words that commonly occur across all speech and therefore only create noise in the data. Some examples are 'the', 'it', 'they' and 'and'.
[3]'Tokenizing' refers to the splitting of words into tokens that have linguistic importance, for example the words 'terrorism', 'terrorist' and 'terror' may all be tokenized to 'terror'. Thus, the core concept of the word is captured while also simplifying the dataset.
[4]'Stemming' refers to the removal of suffixes to reduce the complexity of a dataset.

between the two.

2.2.1. Sentiment analysis methods

Ren, Wu & Liu (2018) analyse news articles at the sentence level by assigning a sentiment polarity (using software designed for the Chinese language) to each word followed by a sentiment score for each sentence in a document. Each document is then categorized and a sentiment score between -1 and 1 for all news for each day is generated. Zubair & Cios (2015) use the positive and negative valence categories from the Harvard General Enquirer (HGI) to assign each word in a Reuters news article a positive or negative label. They then sum the positives and negatives into a tuple and divide that tuple by the number of words in an article in order to create a vector that represents each news article. The vectors are organized into time series, normalized by dividing all vectors by the first vector, parsed through a Kalman filter and then correlated to S&P 500 returns using Pearson correlation (for both the positive and negative scalar in the vector). Kinyua *et al.* (2021) use the Valence Aware Dictionary for Sentiment Reasoning (VADER) to create a sentiment feature for former U.S. President Donald Trump's tweets which was then used as a regression feature in linear, decision tree and random forest regressions. Khedr *et al.* (2017) use N-gram (n=2) to extract key phrases from their corpus of news text data, then term-frequency inverse-document-frequency (TF-IDF) is used to determine the importance of those phrases within the corpus, and finally a naïve-Bayes classifier is used to assign positive and negative labels to each news document. Purevdagva, Zhao, Huang & Mahoney (2020) use a variety of features present in both data and metadata to predict fake political speech. Two features relevant to this project were 'speaker job' and 'context' (press, direct or social) which were labelled using universal serial encoders. For the actual sentiment analysis they used the linguistic inquiry and word count (LIWC) tool to categorize and count words into emotional, cognitive and structural text components. Various further attempts to extract sentiment from the text did not yield increased prediction accuracies. They go on to use an extra tree classifier for feature selection and then support vector machine (SVM), multilayer perceptron, convolutional neural network (CNN), decision trees, fastText and bidirectional encoder representations from transformers (BERT) for prediction with the highest accuracy resulting from the SVM. Dilai, Onukevych & Dilai (2021) use SentiStrength – an automatic sentiment analysis tool – to compare the sentiment in speeches between former U.S. President Donald Trump and former Ukrainian President Petro Poroshenko.

2.2.2. Stock market prediction methods

Ren *et al.* (2018) use an SVM and five-fold cross validation approach to achieve a prediction accuracy of 98% when predicting fake news in political speech. They combined sentiment data and market indicators as their input data. Kinyua *et al.* (2021) use linear, decision tree and random forest regressions to predict S&P 500 and DJIA directional changes. Random forest regression performed best for both datasets. Khedr *et al.* (2017) use Open, High, Low and Close (OHLC) prices and the first lag of directional change as features for prediction of future market trends. Jiao & Jakubowicz (2017) extracted lag and window features from the S&P 500 and the Global 8 Index before running time series random forest, neural network and gradient boosted trees to predict movements of individual stocks in the S&P 500. Liu, Wang, Xiao & Liang (2016) used forward search feature selection to select features for SVM, naïve-Bayes, Gaussian discriminant analysis and logistic regression from a set of economic features including the crude oil daily return, currency exchange rates and major stock indices daily returns in order to forecast the S&P 500 movement.

Table 2.1: Condensed literature review

Paper number	ML Method	Cleaning method	Data type	Index predicted	Max accuracy
1	Support vector machine with fivefold cross validation		Daily online stock reviews	SSE 50	0.98
1	Support vector machine with rolling windows				0.90
1	Logistic regression with fivefold cross validation				0.87
2	Non-ML:	Tokenized and mined using the Harvard General Inquirer dictionary	Reuters textual data	S&P 500	Corr: -0.91
3	Random forest	Date validation, stop word and URL removal	Tweets	INDU	0.98
3	Decision tree				0.97
3	Logistic regression				0.81
3	Random forest			S&P 500	0.92
3	Decision tree				0.88
3	Logistic regression				0.77
4	N-gram, TF-IDF, Naïve Bayes, K-NN	Tokenize, stopwords, stemming, abbreviation processing	News articles and financial reports	Three tech stocks	0.9
5	TS logistic regression	Feature extraction: lags, window features	Financial indicators	Individual S&P 500 stocks	0.79
5	TS random forest				0.78
5	TS neural network				0.78
5	TS gradient boosting				0.78
6	Logistic regression	Date validation forward search feature selection	Market indices, exchange rates	S&P 500	0.61
6	Gaussian discriminant analysis				0.61
6	Naïve Bayes				0.60
6	Linear SVM				0.60
6	Radial Basis Function SVM				0.63
6	Polynomial SVM				0.60
7	SVM	Feature extraction and feature selection	Political speeches and metadata	Liar dataset	0.74
7	Multilayer perceptron				0.55

Paper number	ML Method	Cleaning method	Data type	Index predicted	Max accuracy
7	Convolutional neural network				0.61
7	fastText				0.66
7	BERT				0.66

Table 2.2: Table 2.1 metadata

Paper number	Title	Citation
1	Forecasting Stock Market Movement Direction Using Sentiment Analysis and Support Vector Machine	(Ren *et al.*, 2018)
2	Extracting News Sentiment and Establishing its Relationship with the S&P 500 Index	(Zubair & Cios, 2015)
3	An analysis of the impact of President Trump's tweets on the DJIA and S&P 500 using machine learning and sentiment analysis	(Kinyua *et al.*, 2021)
4	Predicting Stock Market behavior using Data Mining Technique and News Sentiment Analysis	(Khedr *et al.*, 2017)
5	Predicting Stock Movement Direction with Machine Learning: an Extensive Study on S&P 500 Stocks	(Jiao & Jakubowicz, 2017)
6	Forecasting S&P 500 Stock Index Using Statistical Learning Models	(Liu *et al.*, 2016)
7	A machine-learning based framework for detection of fake political speech	(Purevdagva *et al.*, 2020)

3. Data collection and cleaning process

This section describes the data collection and cleaning process as well as the feature extraction process. It begins by explaining the three data-subsets collected for this study, namely: control, meta and test data. Control refers to auto-regressive features extracted from the S&P 500, meta refers to S&P 500 adjacent financial data and test refers to vectorized presidential speeches – which are the focus of this study. The section begins by elaborating on the collection and cleaning steps for each of these datasets. The next subsection describes the feature engineering methods (sentiment analysis and word vectorization) that were used to extract variables with potentially strong signals from the text data. The final subsection describes the reasoning, methods and results used in a time series analysis of the financial time series (S&P 500) data in order to extract useable auto-regressive features from it for the control data-subset.

3.1. Data

Three types of data were gathered for this project, namely: presidential speeches/text data (all the transcripts from the Presidency Project website including formal and informal and written and verbal addresses), a history of the S&P 500 index and a history of five S&P 500 adjacent price histories – i.e. two subsets of financial data.

The S&P 500 index and the metadata was downloaded from Yahoo Finance while the presidential speeches were scraped from the American Presidency Project (Yahoo, n.d.) (Woolley & Peters, n.d.). The S&P 500 is downloaded using the 'fin_data_downloader' Python module and will always download the entire history of the S&P 500 index at a daily interval [5]. The same goes for the metadata. The presidential speeches on the other hand were scraped using the 'WebScrapeAndClean' Python module which will also always scrape the entire corpus available on the American Presidency Project website. This allows for perfectly updated data to be collected at any time.

The S&P 500 data contains seven variables, namely: 'Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', and 'Volume'. 'Open' records the opening price for each day, while 'Close' records the closing price. 'High' records the daily high and 'Low' the daily low. 'Volume' records the dollar amount of stock traded in the S&P 500 on each day and 'Adj Close' is irrelevant because it never differs from 'Close'. Notably, opening and closing prices are only differentiated between after April 20th, 1982, while volume was only recorded from 1950.

After scraping, the Presidency Project data contains five variables. These are 'Type' which records the category and sub-category of each speech, 'Name' which records the title and name of the main speaker, 'Date' which records the date the speech occurred on, 'Title' which records the title of each speech and 'Transcript' which contains the raw HTML transcript of the speech.

3.2. Cleaning

The S&P 500 index did not require any cleaning after download, besides the removal of the redundant 'Adj Close' variable. Conversely, the presidential speeches required extensive cleaning. Text that has been web-scraped contains HTML tags[6], thus, the first step was to remove the HTML tags from the text. Similarly, the test contained reactions from the crowds listening to the speeches[7], which were also removed. Next, the transcripts were converted to lower case and the question sections removed. Next a 'No Stops Transcript' variable was created by removing the stopwords[8] in the Natural Language Tool Kit (NLTK) stopword dictionary from the clean transcript. The original transcripts were also kept. The cleaning of the speech data and both financial data-subsets was done in their original collection modules, i.e. the 'fin_data_downloader' Python modules and the 'WebScrapeAndClean' Python modules respectively.

[5]All code used in this project can be found on my Github profile at github.com/PabloRees/Masters

[6]HTML tags include paragraphing and spacing indicators for computers to interpret such as '<p>' and '</p>'.

[7]These were tags such as '<em>Laughter</em>' and '<em>Applause</em>'.

[8]Stopwords are words that commonly occur in the English language and are therefore unlikely to contain any sort of signal and thus, constitute only noise.

3.3. Text feature engineering

Feature engineering is required to increase the strength of the signals coming out of the speech data and reduce the computational power required to run machine learning algorithms. Feature engineering refers to the emphasis of certain signals within the available data and creation of new variables which capture these signals. It results in the addition of extra features (variables) to the dataset. There are three broad methods of feature engineering: feature selection, feature extraction and the creation of new features (Géron, 2019: 27). In this section, the creation of new features occurred and took the form of sentiment analysis and vectorization via word embedding for the text data.

3.3.1. Sentiment analysis

Two versions of sentiment analysis were carried out. First, NLTK's Valence Aware Dictionary for Sentiment Reasoning (VADER) was used to extract a sentiment analysis tuple, in this instance containing four metrics, namely, negativity, neutrality, positivity and compound. VADER is a lexicon-based system of sentiment analysis (Sohangir, Petty & Wang, 2018). Each of the negativity, neutrality and positivity metrics describe a transcript independently of the others while the compound metric describes a transcript comprehensively (combining the other three). VADER, when compared with alternative NLP feature extraction techniques has performed better on social media transcripts and generalized better to other areas (Hutto & Gilbert, 2014 ; Elbagir & Yang, 2019). VADER has been used in sentiment extraction from text relative to financial data in Pano & Kashef (2020) for Bitcoin price predictions, Agarwal (2020) which found a strong correlation between VADER sentiment scores and stock price changes and Sohangir *et al.* (2018) which shows the superiority of lexicon-based approaches (specifically VADER) over ML approaches for sentiment classification.

Next, the TextBlob package's sentiment analysis tool was used. This yields two scores (in a tuple) describing the sentiment of a transcript, namely, a polarity score (ranging from -1 to 1) and a subjectivity score (ranging from 0 to 1). Polarity describes whether the emotions expressed are negative or positive, with lower scores indicating negativity while subjectivity indicates the extent of the usage of subjective words (Loria & others, 2018). Biswas, Sarkar, Das, Bose & Roy (2020) use TextBlob sentiment scores in their analysis of the effects of Covid-19 on stock markets. TextBlob was also tested in Sohangir *et al.* (2018) but did not perform as well as VADER although it did outperform ML methods in terms of area under the curve (AUC) scores. Given the superior performance of VADER in other projects it was expected that VADER's sentiment tuple would outperform TextBlob's sentiment tuple as a predictor of the S&P 500 data. The results explained in section 5 show that this expectation was correct.

3.3.2. Speech vectorization

Converting human-readable text into machine-readable data requires the conversion from words to numbers. This vectorization can be done in various ways but, in order to preserve the meaning of the texts, the Word2Vec and Doc2Vec Python packages provided by Gensim were used (Rehurek & Sojka, 2011). However, Word2Vec was not originally designed by Gensim, but rather by Mikolov, Chen, Corrado & Dean (2013) and Mikolov, Sutskever, Chen, Corrado & Dean (2013) while Doc2Vec was suggested by Le & Mikolov (2014).

3.3.2.1. Word2Vec. Gensim's Word2Vec Python package's skip-gram model is used for Word2Vec vectorization. Using the full vocabulary of words in a corpus of speeches and one-hot encoding for word vectors, Word2Vec trains a single-hidden-layer neural network to predict words based on the words around them.

The model contextually embeds each word in the entire corpus of speeches by running the speeches through a single-hidden-layer predictive neural network (NN). The NN is provided with the pseudo-task of predicting the word of focus from the words surrounding it each time it occurs in the corpus. Thus, a hidden layer is trained to contain the information that contextually embeds each word in the corpus. These hidden-layer vectors (rather than the predictions) are the real output of the Word2Vec model. Words that appear in similar contexts throughout the corpus will have similar representational vectors (hidden layers) and thus, can be said to have similar meanings in the corpus Mikolov, Sutskever, *et al.* (2013).

An example phrase might be 'The quick brown fox jumps'. If the word 'brown' is the focus word, the words 'quick' and 'fox' would be fed into the NN which would then be trained to map the input to the word 'brown'. Doing this for every instance of 'brown' in a corpus creates a hidden layer that contains all the contextual information required to predict the word 'brown' in the given corpus. Figure 3.1 depicts this process.

The model is extremely good at relating words that appear in similar contexts to each other. For example, when asked for the three words most similar to 'oil', the model trained on the presidential speeches corpus returns 'crude', 'gas' and 'petroleum'. The input 'gold' returns 'silver', 'bullion' and 'coin'; whilst 'virus' returns 'covid19', 'infection' and 'pandemic'.

In this study the representational vectors each contain 200 elements (because the hidden layers were set to contain 200 elements). The size of the vector was set to 200 because of the precedent set by Vargas, De Lima & Evsukoff (2017), however there is little consensus on the dimensions of vectors that should be used for word embedding and most researchers use a vector size between 50 and 300 (Patel & Bhattacharyya, 2017). However, this is an avenue that warrants further research. Thus, each word is described by a vector containing 200 elements. In order to create a similarly sized vector for each speech, the vectors describing all the words in each speech were averaged. Thus, each speech has been reduced to a 200-element vector averaging the contextual embeddings of each word contained therein. This averaging technique was also used in Vargas *et al.* (2017) and Qin (2018). However, this method fails to preserve word order in a document vector (Le & Mikolov, 2014).

Notably, the algorithm also makes room for two-word phrases such as 'asset backed' or 'short selling' – which are considered as 'assetbacked' and 'shortselling'. When two words that occur irregularly in the vocabulary occur together frequently, the algorithm interprets them as a phrase and treats them as such. There is, however, only room for two words in each phrase if the algorithm has only been executed once – which is the case here.

Word2vec is used by Shi, Zhao & Xu (2019) for the improvement of sentiment classification, which implies that the final vectors in this study may contain sentiment information. It is also used by Vargas *et al.* (2017) and Qin (2018) in their predictive modelling of S&P 500 changes based on Twitter data. Vargas *et al.* (2017) also used 200-element vectors while Qin (2018) used 300-element vectors. Both studies achieved prediction accuracy around 65%.

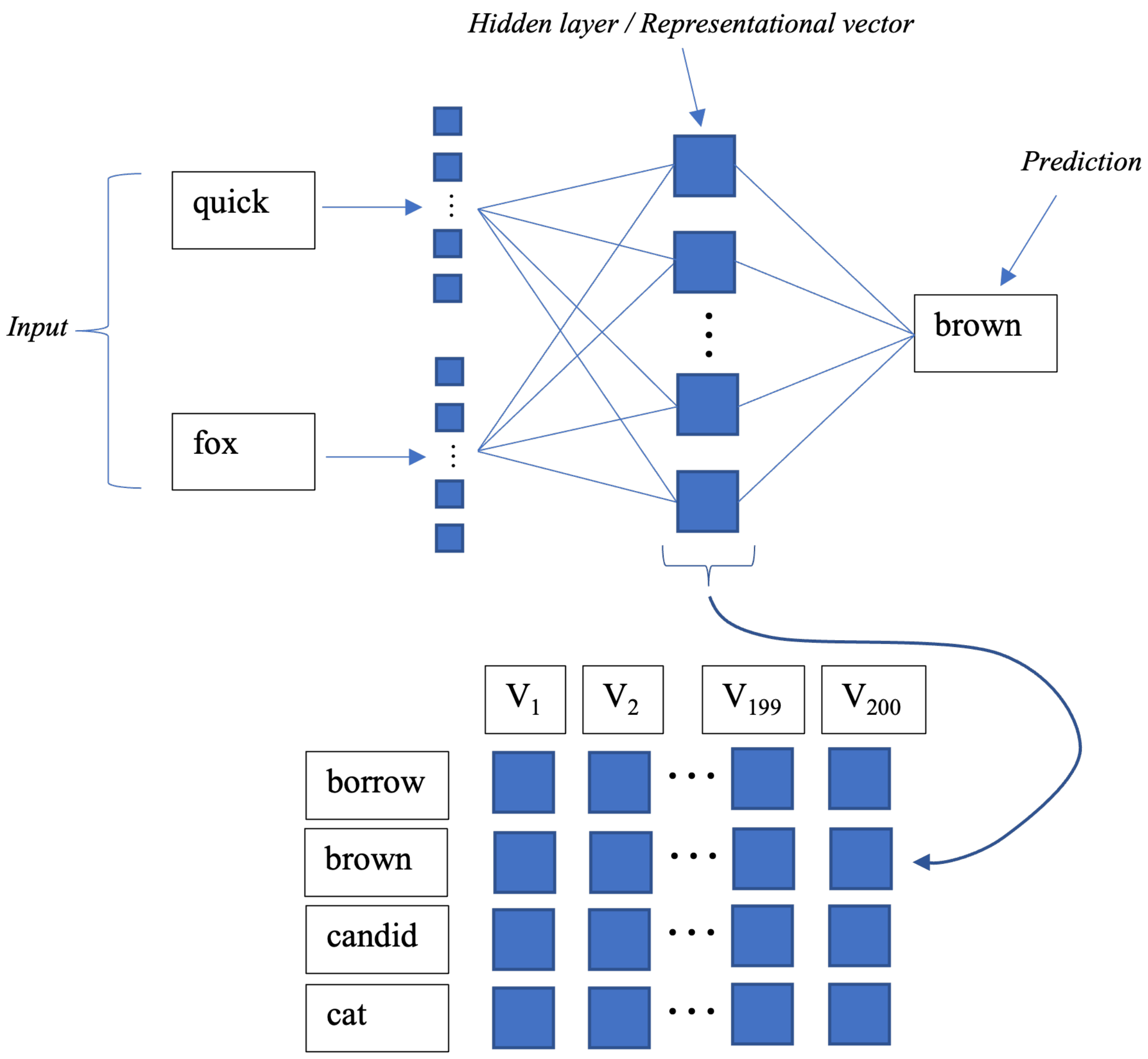

Figure 3.1: Word2Vec Model

3.3.2.2. Doc2Vec . An alternative method to creating a vector representation of a sentence or document is Doc2Vec. This method is similar to the Word2Vec method described above but includes an additional floating vector when performing its pseudo-task. This vector is maintained across every word prediction task within a document and subjected to training for each instance of every word. Thus, each document in a corpus is assigned a single comprehensive vector that embeds it in the corpus. Embedding within the broader corpus is maintained by tagging each document with a unique tagging phrase. This method outperforms other methods such as bag-of-words for text representations (Le & Mikolov, 2014).

There are two implementations of Doc2Vec, namely Distributed Bag of Words (DBOW) and Distributed Memory (DM) (Sohangir *et al.*, 2018). Both have been used in this study. In both cases each document is assigned a paragraph tag which represents the paragraph to the Doc2Vec algorithm. DM Doc2Vec does this in the same way that a word represents itself to the Word2Vec algorithm. For every word in a document, its paragraph tag is passed to the Doc2Vec algorithm along with the words relevant to the current prediction pseudo-task. Back propagation is employed in the same manner as in Word2Vec except that the document tag vector is optimized for separately from the word vectors. This document tag vector is the relevant output in this case. Because a single vector of weights is created for each document, this vector should constitute a vectorized representation of the document as a whole. Figure 3.2 depicts the architecture of DM Doc2Vec algorithms. Alternatively, DBOW Doc2Vec algorithms ignore the context of a word and use random sampling to predict words from a paragraph. Figure 3.3 depicts the architecture of a DBOW Doc2Vec algorithm.

Only one example of the use of Word2Vec for financial prediction could be found. Rashid & Leon (2019) study the possibility of predicting closing prices of the stocks of four major private technology companies and three major government contracted companies, based on tweets from then U.S. President, Donald Trump. Back-testing a trading model based on the stock price predictions of a four-layer NN using Doc2Vec data as input yielded fair portfolio gains for the government contracted companies' stocks but almost no change in portfolio values for the private technology companies' stocks.

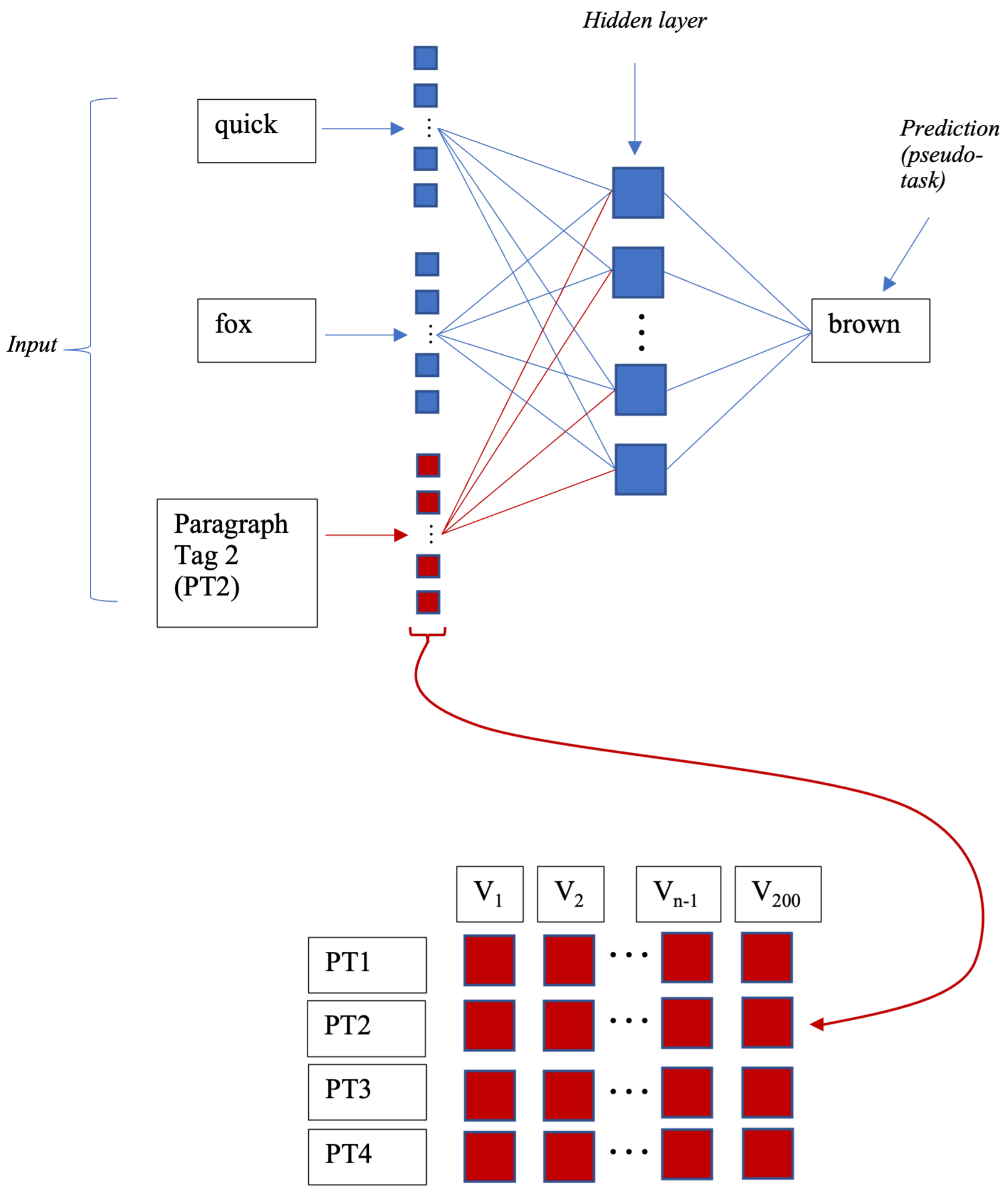

Figure 3.2: Doc2Vec model - distributed memory architecture: dm = 1

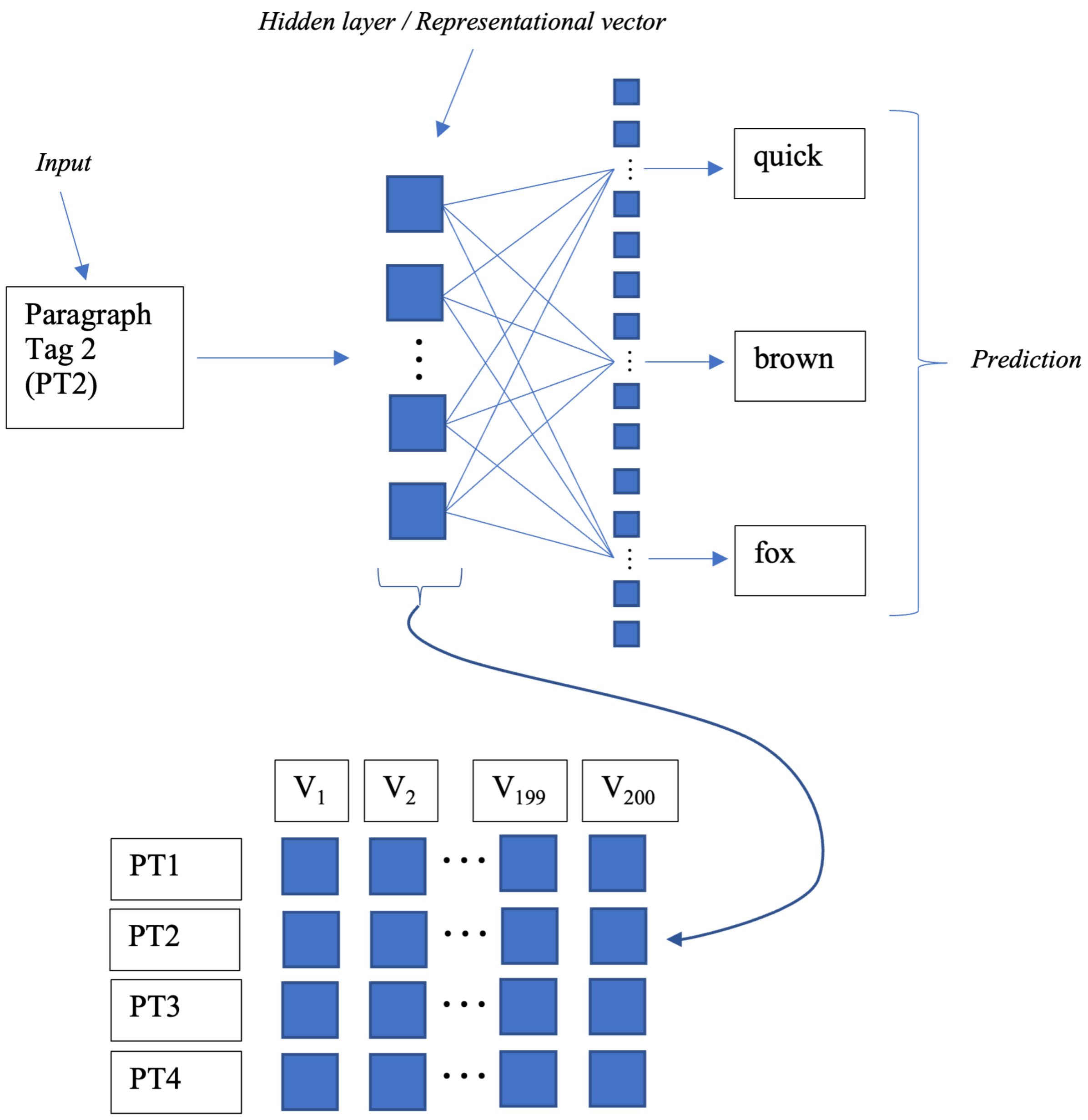

Figure 3.3: Doc2Vec model - distributed bag-of-words architecture: dm = 0

3.4. S&P 500 time series analysis

As part of feature extraction, an econometric time series analysis was run on the S&P 500 data. The aim of this analysis was to find the linear model of best fit to the S&P 500 data and then include relevant auto-regressive variables in the final dataset under the assumption that they will be relevant in the highly non-linear ML models. An initial analysis was done on the S&P 500 data after April 20th, 1982, because differentiation between Open and Close prices occurs from that date onwards. This analysis found a large array of significant variables – more than half the days of the month, the month of September, a few specific years, ten non-consecutive lags and the first lag of volume of trade. The significance of these variables, particularly the days of the month, were difficult to explain rationally. However, they did indicate persistence of volatility. This volatility persistence and the fact that volume is a strong indicator of absolute price change encouraged a second round of analysis that was done on all data following the initial recording of volume in 1950. This second round of analysis was focused on finding an auto-regressive conditional heteroskedasticity (ARCH) model that fit the data well. The analysis concluded with the selection of an ARMA (2,0) GARCH (1,1) model. Thus, the auto-regressive variables selected for inclusion in the final data set are the first and second lags of the demeaned log difference between consecutive closing prices (DlogDif), the first lag of the absolute DlogDif (absDlogDif_1) as a pseudo-variance measure and the first lag of standardized volume of trade controlled for date trend (stdVol_1DateResid) as a second pseudo-variance measure.

3.4.1. 1982 onwards

Running time series analysis on the daily percentage change in the S&P 500 after April 20th, 1982, revealed ten significant non-consecutive lags. The Daily percentage change variable was created by taking the percentage difference between the Open and Close variables for each day of the data. Before April 20th, 1982, the Open and Close variables hold identical values, hence the time series analysis only being run after that date. Daily percentage change is naturally stationary. This is supported by the results of an Augmented Dickey-Fuller unit root test which indicated that no unit root is present in the data. Running a partial autocorrelation function revealed ten significant lags (these were lagged by 1, 2, 4, 12, 15, 16, 18, 27, 32, and 34 periods, respectively). Regressing the daily percentage change variable on all ten significant lags reinforced the finding by yielding significance above the 95% confidence interval for all ten lags. Further, regressing the daily percentage change on weekday, monthday, month and year categorical variables yielded an array of significant correlations across the monthday, month and year categories (none of the weekdays were significant). Regressing on the categorical variables and the lags simultaneously yielded similar results with increased significance for 2002 (to the 99% level) and 2008 (to the 99.9% level) and the addition of 2018 (significant at the 95% level). Furthermore, an extra four monthdays were deemed significant – bringing the total to 21 (out of 31) significant monthdays; while some of the significance levels on the lags changed. Adding a normalized (distributed standard normal) volume variable lagged by one period to the regression yields a significance at the 99% level on the lagged volume variable and reduces the significance on the year variables.

While the years 2002 and 2008 are justifiable as significant because of the financial crashes that happened in each of those years (2002 dot com bubble and 2008 housing bubble) and September is also relatable to the housing bubble, it is difficult to rationally justify the monthdaY-variablys as significant regardless of their quantitative significance. The high number of lags is also difficult to justify and implies rather that there may be persistence of volatility. Thus, the following section focuses on the modelling of volatility over a time period that maximizes the inclusion of volume statistics in the data. All of the analysis reported in this section was done in R[9] (Marais, 2022).

[9]The primary packages used were 'stats', 'tidyverse', 'ggplot2' and 'fixest'.

3.4.2. 1950 onwards – ARCH model

A second round of time series analysis was performed on all the data after Jan 1st, 1950, which revealed that an ARMA(2,0) GARCH(1,1) model might be the most appropriate linear fit to the data and that the first lag on a date-controlled version of standardized volume is a good predictor of logDif. The secondary analysis was performed because the number (and non-consecutiveness) of significant lags in the first model was out of the ordinary. The choice was made to extend the dataset back to 1950 because lagged volume appeared to be significant and volume data is available from that time onwards.

The extended dataset required a different dependent variable because there was only differentiation between Open and Close prices after April 20th, 1982. Thus, the inter-day log difference in closing price constituted the dependent variable in the second round of analysis. Figure 3.4 depicts the logDif from 1950 until present – the persistence of volatility is made apparent by the intermittent spikes. Figure 3.5 illustrates the correlation between the logDif and standardized volume which justifies the extension of the dataset. Further the logDif auto-regression results depicted in Table 3.1 and the categorical regression results depicted in Table 3.2 align with the regression results presented in section 3.4.1 and further suggest persistent volatility exists in the logDif series.

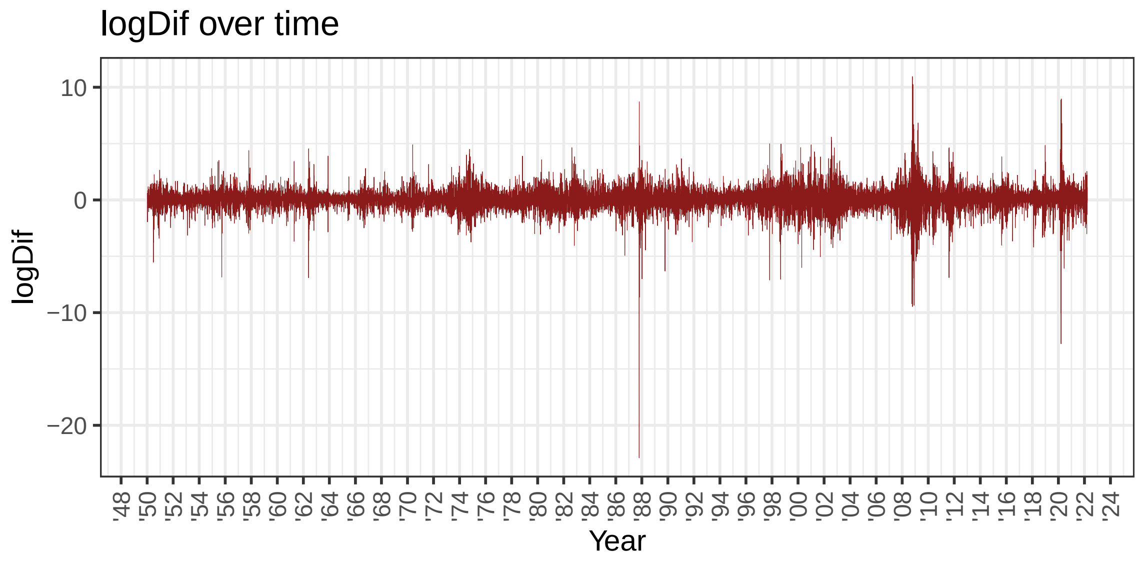

Figure 3.4: SandP 500 Closing Price Log Difference 1950 – 2022

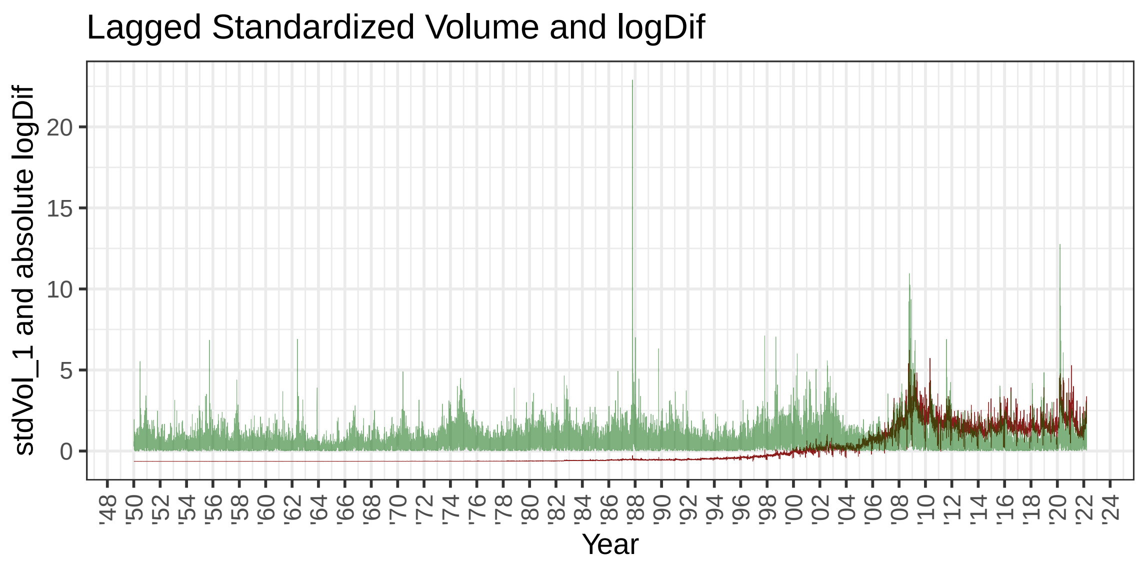

Figure 3.5: Standardized volume (red) and absolute logDiff (green)

Table 3.1: Auto-regressions of logDif on significant lags

	Reg1	Reg2
(Intercept)	0.032 ***	0.031 ***
	(0.007)	(0.007)
LD_2	-0.020 **	
	(0.007)	
LD_6	-0.022 **	
	(0.007)	
LD_8	-0.015 *	
	(0.007)	
LD_12	0.031 ***	0.031 ***
	(0.007)	(0.007)
LD_15	-0.025 ***	-0.024 **
	(0.007)	(0.007)
LD_16	0.040 ***	0.040 ***
	(0.007)	(0.007)
LD_18	-0.018 *	
	(0.007)	
LD_26	-0.026 ***	-0.025 ***
	(0.007)	(0.007)
LD_27	0.022 **	
	(0.007)	
LD_29	0.018 *	
	(0.007)	
LD_34	-0.033 ***	-0.034 ***
	(0.007)	(0.007)
N	18138	18138
R2	0.007	0.005

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

Table 3.2: Regression of logDif on significant categorical variables

	Reg3
(Intercept)	0.059 ***
	(0.008)
September::1	-0.071 **
	(0.027)
Monday::1	-0.116 ***
	(0.019)
N	18172
R2	0.002

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

3.4.2.1. Volume. Regressing the logDif on the first lag of standardized volume yielded insignificant results. However, regressing absolute logDif (pseudo-volatility) on the first lag of standardized volume (stdVol_1) yielded a coefficient of 0.148 significant above the 99.9th percentile. Figure 3.6 shows this relationship. Figure 6 also shows that standardized volume (stdVol) trends upwards over time. Further, regressing stdVol_1 on Date (regression 6) yields a positive coefficient significant above the 99.9th percentile and an adjusted R2 of 0.6. This shows that a large percentage of the variation in stdVol is explained by variation in Date. Thus, to avoid implicitly including a Date proxy[10] as a predictor of S&P 500 returns the included volume variable must be adjusted to be stationary over time. The residuals from regression 6 make up the variation in stdVol that cannot be explained by the variation in the Date variable – thus, they are a Date-neutral measure of stdVol. Figure 3.7 shows the difference between stdVol and stdVol controlled for variance due to Date (stdVolControlled). Regressing the residuals of regression 6 on Date (regression 7) shows no significant relationship between the two while regressing absolute logDif on the residuals from regression 6 (regression 8) yields a coefficient of 0.164 significant above the 99.9th percentile. Figure 3.8 shows the relationship between absolute logDif and stdVolControlled. Thus, these residuals (rather than stdVol) should be included as a variable in the final prediction data.

[10]The goal of this project is to create a model that can predict future S&P 500 returns. Thus, to include a proxy for Date (in this case in the Date trending stdVol) would cause data contamination since dates do not hold any information relevant to the S&P 500 returns besides price or volatility trends.

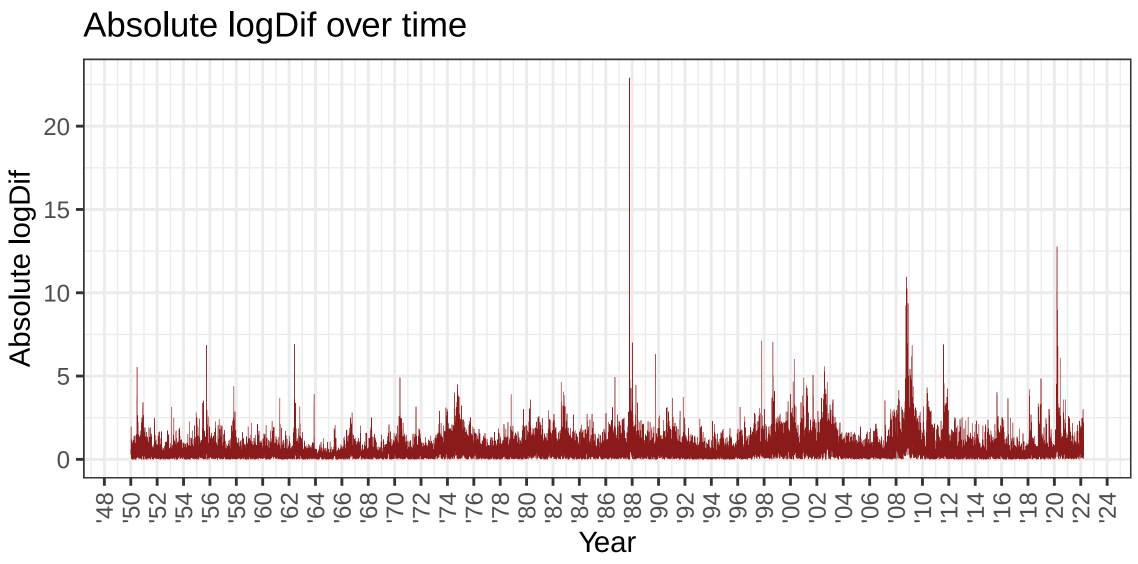

Figure 3.6: Absolute logDiff and stdVol

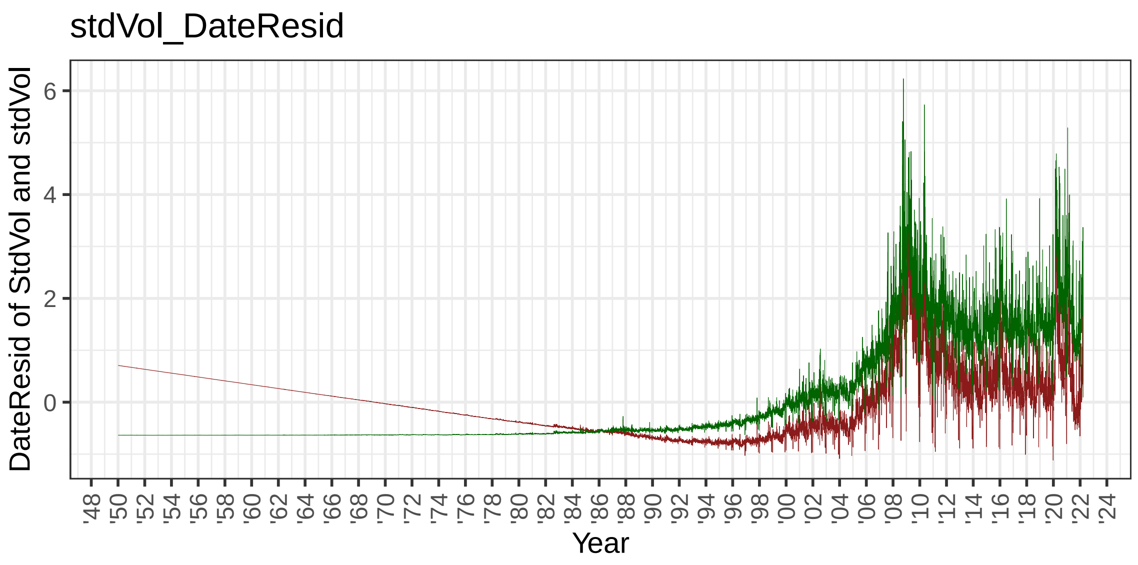

Figure 3.7: dateResid of StdVol (Red) and stdVol (Green)

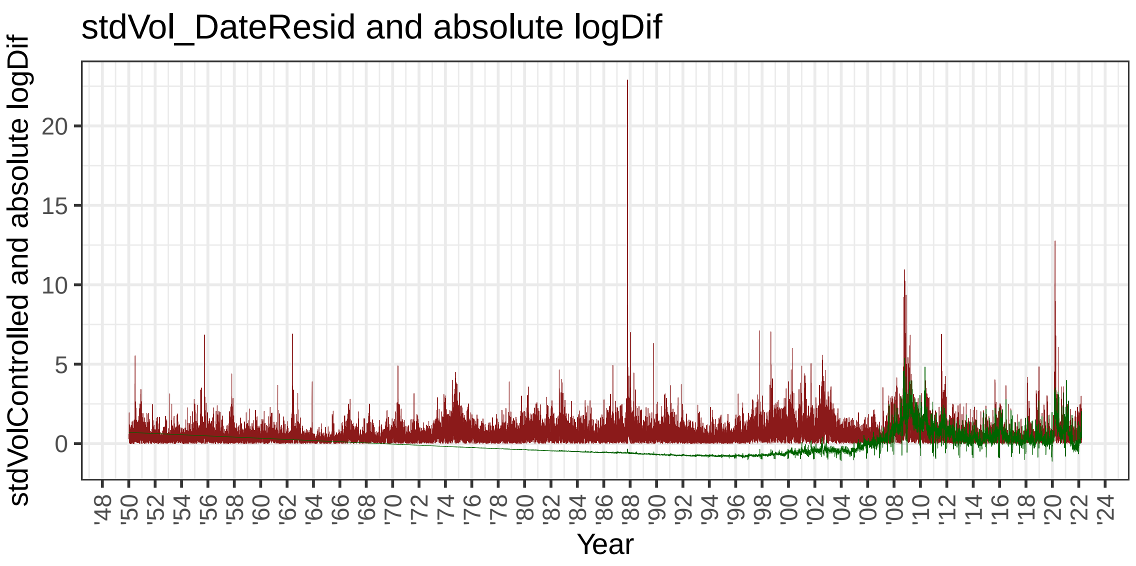

Figure 3.8: stdVolControlled and absolute LogDiff

3.4.2.2. ARCH and GARCH testing . The LJung-Box test is a standard inference test in time series analysis. It tests the null-hypothesis that a series is a white noise series (Hassani & Yeganegi, 2020). Selection of the number of lags to be included in the Ljung-Box test is discussed by Hassani & Yeganegi (2020). Critical to this decision is the ratio of lags (H) to the number of observations (T) in the data set. If the size of H is too large, the actual size of the test exceeds the nominal size of the test, and its integrity is violated. The dataset for this set of tests contains 18 143 observations. The smallest (most difficult to satisfy) ratio permitted in the discussion is 0.001 suggested by Hyndman & Athanasopoulos (2018). Thus, the maximum number of lags that satisfy the smallest H/T ratio is 18 and any number of lags less than 18 is permissible.

Running a 10-Lag LJung Box test on the logDif yields a p-value of 0.0002 which indicates that the null hypothesis should be rejected, and that the series is not a white noise series. I.e. ARCH effects are present in the series. Running autocorrelation and partial-autocorrelation tests on the absolute logDif indicates that there is a degree of persistence of volatility (Kotzé, 2021). This can be seen in Figure 3.9. A t-test indicates that the mean of the population is significantly different from zero (0.036). Thus, demeaning the equation constitutes the construction of a mean equation (essentially the demeaned logDif) represented in the series – a demeaned log difference (DlogDif) (Kotzé, 2020). A ten lag Ljung-Box test again indicates the presence of ARCH effects specifically in the 1st,2nd, 5th, 6th, 8th, and 9th lags. However, running a multiple linear regression (OLS) of DlogDif on its lags reveals only the 2nd, 6th, 8th and 9th lags as significant above the 95th percentile.

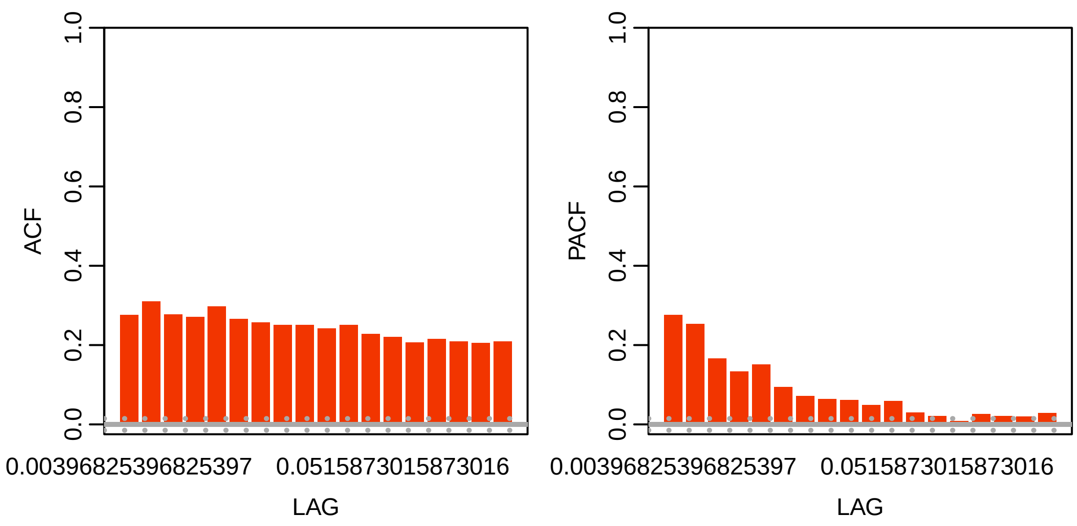

Figure 3.9: Autocorrelation and partial-autocorrelation functions of absolute LogDiff

Eight models were tested for performance considering the above results. Model 1 is an ARIMA(0,0,1), model 2 is an ARIMA(1,0,0), model 3 is an ARMA(1,0) GARCH(1,1), model 4 is a GARCH(1,1), model 5 is an EGARCH(1,1), model 6 is an ARMA(1,0) ARCH(1,0), model 7 is an ARCH(1,0) and model 8 is an ARMA(2,0) GARCH(1,1). The inclusion of an EGARCH model is motivated by its use in Nelson (1991), Blazsek & Mendoza (2016) and Tiwari, Raheem & Kang (2019) in their time series analyses of the S&P 500. Table 3.3 compares the coefficients from the 6 ARCH model variations (models 3, 4, 5, 6, 7 and 8). Figure 10, 11 and 12, 13, 14 and 15 show the results of autocorrelation and partial-autocorrelation functions of the residuals of the 6 ARCH variation models. Notably, all three of the models that do not contain an ARMA(1,0) component (models 4,5 and 7) yield a significant first residual indicating that inclusion of the ARMA(1,0) component is important. However, all the models containing an ARCH(1,0) - besides model 6 - component yield a significant second residual. Further, model 6 – the non-generalized ARCH model – has intermittent significant residuals indicating that volatility is clustered and persistent. Persistent volatility is one of the oversights of the ARCH model that is addressed in the GARCH model. As can be seen, Model 8 (Figure 3.15) yields the least significant residuals and as such has been selected as the model that best fits this time series. The model linearly predicts the dependent variable using the first and second lags of the dependent variable, the first residual of an ARMA(1,0) model and the first lag of variance in the time series (Kotzé, 2020). As such, the first and second lags of DlogDif and both the first lag of the absolute DlogDif (absDlogDif_1) and the first lag of standardized volume of trade controlled for the date trend (stdVol_1DateResid) as pseudo-variance measures.

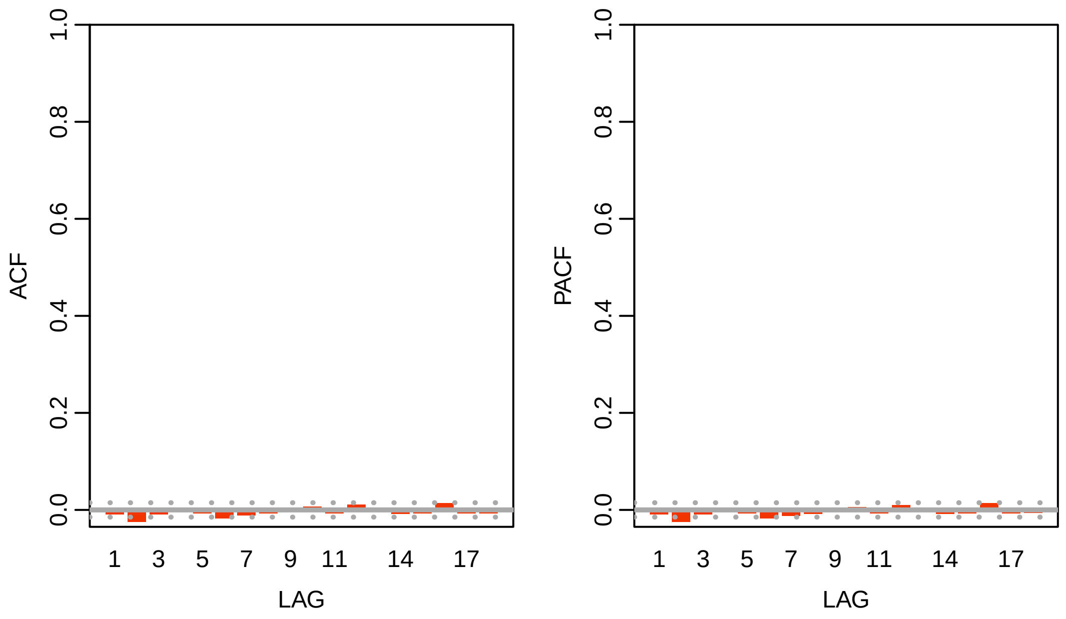

Figure 3.10: Autocorrelation and partial-autocorrelation functions of model 3 - ARMA(1,0) GARCH(1,1) - residuals

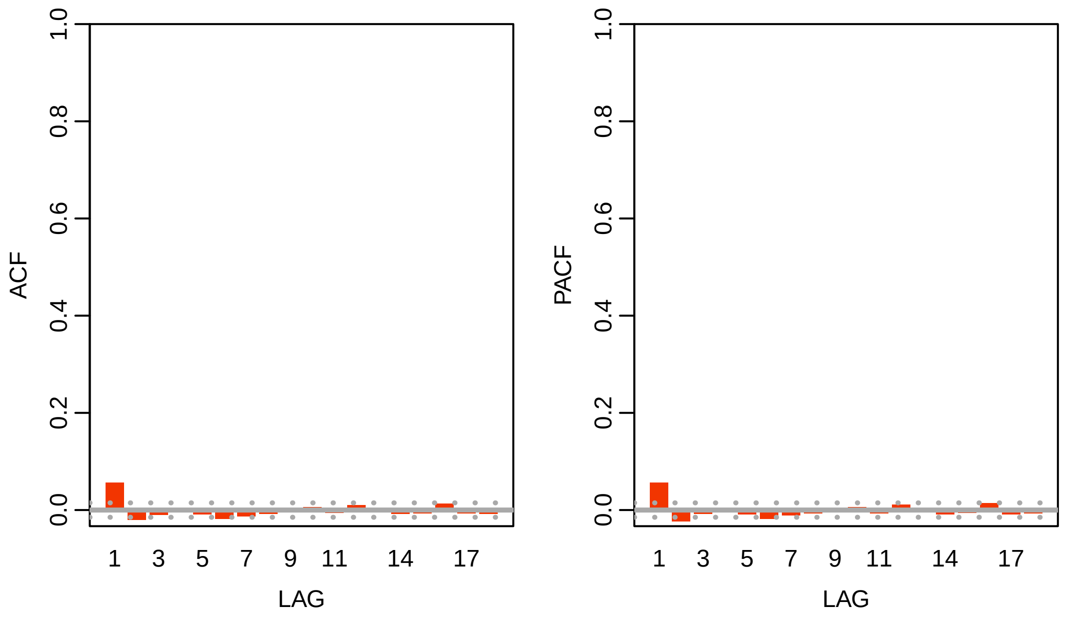

Figure 3.11: Autocorrelation and partial-autocorrelation functions of model 4 - GARCH(1,1) - residuals

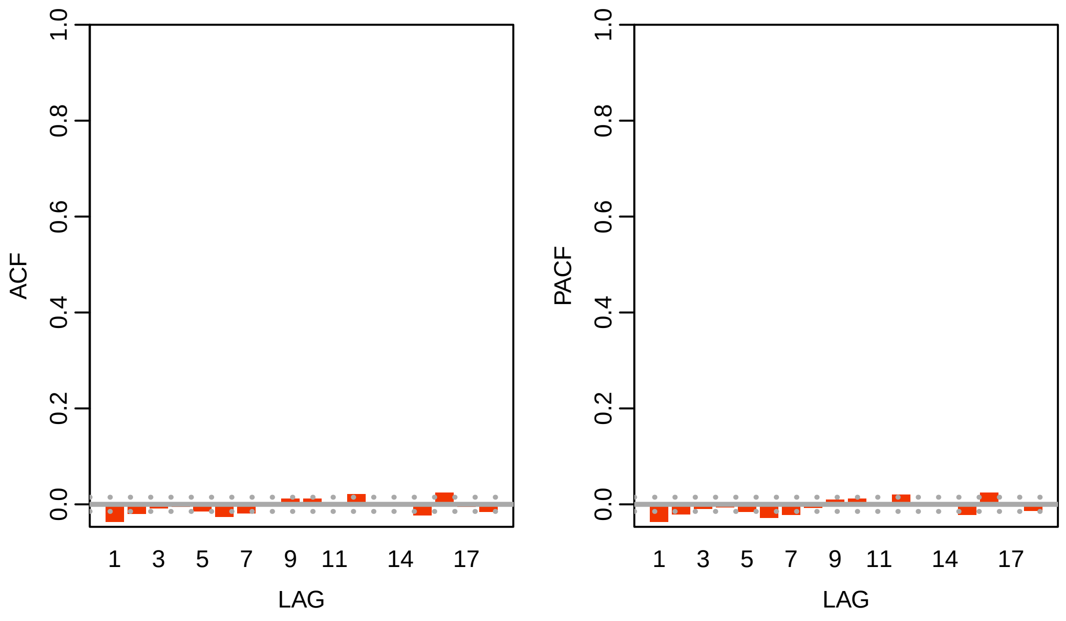

Figure 3.12: Autocorrelation and partial-autocorrelation functions of model 5 - EGARCH(1,1) - residuals

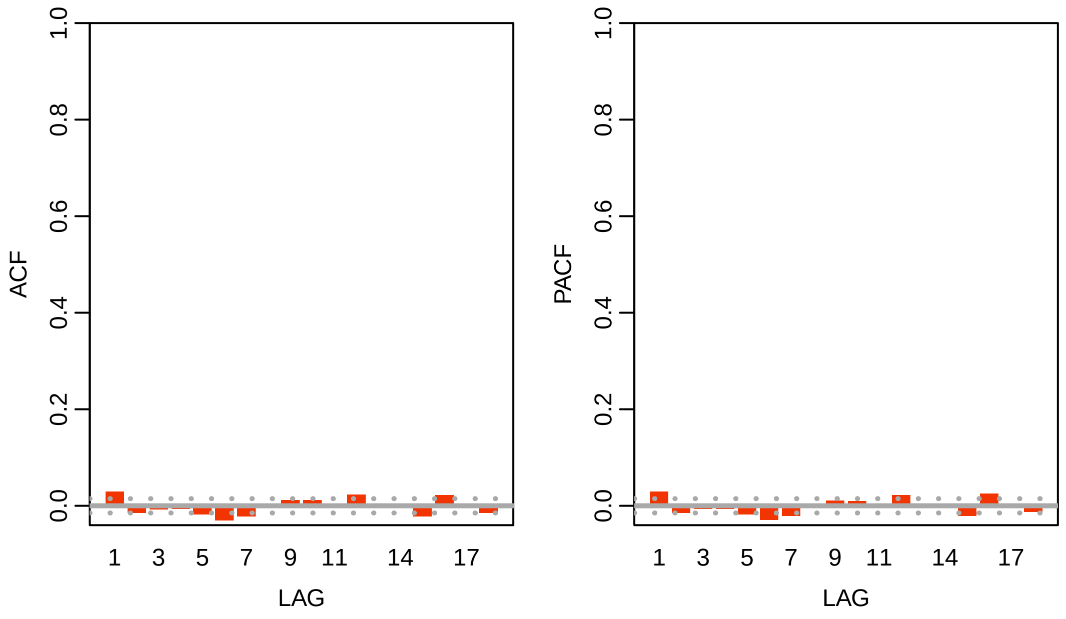

Figure 3.13: Autocorrelation and partial-autocorrelation functions of model 6 - ARMA(1,0) ARCH(1,0) - residuals

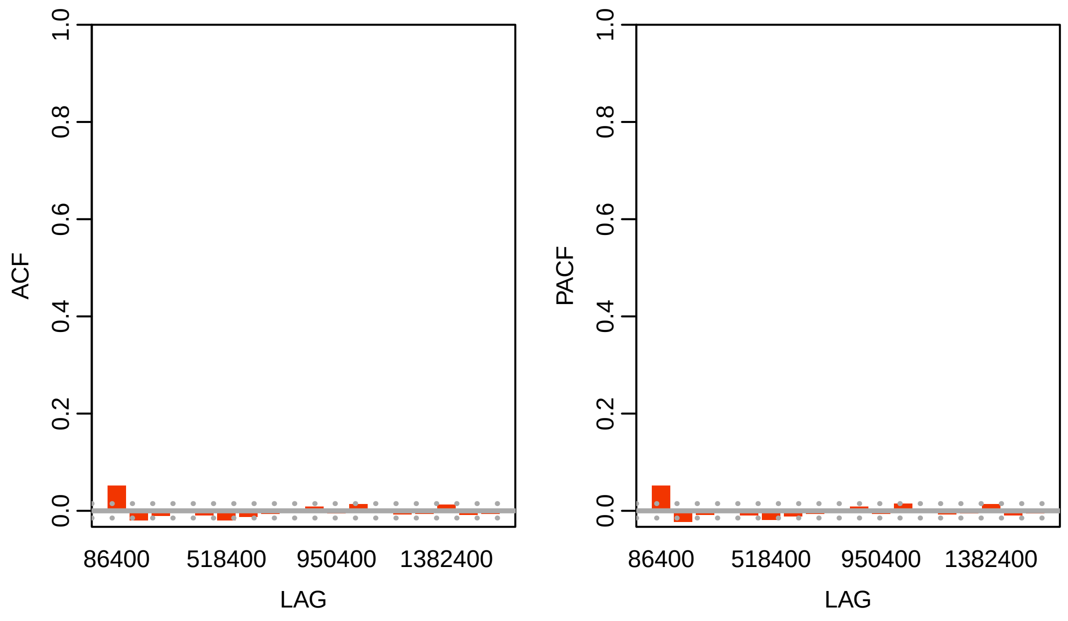

Figure 3.14: Autocorrelation and partial-autocorrelation functions of model 7 - ARCH(1,0) - residuals

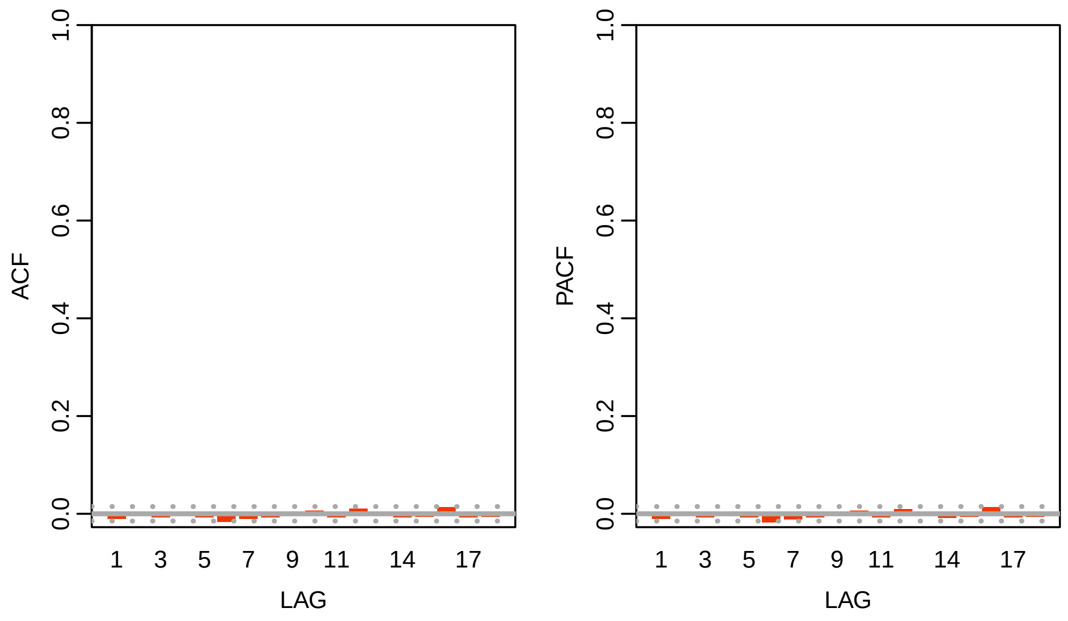

Figure 3.15: Autocorrelation and partial-autocorrelation functions of model 8 - ARMA(2,0) GARCH(1,1) - residuals

Table 3.3: Significancies of various ARCH models

Statistic	Model3	Model4	Model5	Model6	Model7	Model8
Specification	ARMA(1,0) G(1,1)	G(1,1)	EG(1,1)	ARMA(1,0) ARCH(1,0)	ARCH(1,0)	ARMA(2,0) G(1,1)
μ	1	1	1	1	1	1
AR1	-	<2e-16 ***	6.16e-08 ***	-	-	< 2e-16 ***
AR2	-	-	-	-	-	0.00355 **
ω	2.22e-16 ***	<2e-16 ***	<2e-16 ***	<2e-16 ***	<0.005 ***	<2e-16 ***
α 1	<2e-16 ***	<2e-16 ***	<2e-16 ***	<2e-16 ***	<2e-16 ***	<2e-16 ***
β 1	<2e-16 ***	<2e-16 ***	-	-	<2e-16 ***	<2e-16 ***
γ 1	-	-	-	-	<2e-16 ***	-

3.5. Additional variables

Four additional auto-regressive variables were included to bolster the information available to the algorithms under the assumption that the linear modelling completed here is insufficient to exhaustively extract useful variables. To account for persistence in positive or negative momentum, such as in an EGARCH model, a positive and negative transform variable was created. The mean of the positive entries in the DlogDif variable was juxtaposed with the mean of the negative entries to create a ratio of 1:-1.05 for positive to negative entries. Thus, where DlogDif is negative posNegTransform is -1.05 and where posNegTransform is positive posNegTransform is 1. Another three variables were added to account for persistence in extreme price movements such as occurred in the 2002 or 2008 financial crises. These are blackSwan_SD3, blackSwan_SD4 and blackSwan_SD5. They are dummY-variablys set to 1 where the DlogDif variable lies outside three, four or five standard deviations from the mean of the series, respectively. Notably, all three black swan variables and the posNegTransform are lagged by one period. Thus, another four auto-regressive variables are added to the dataset.

Further, five S&P 500 adjacent indicators were selected for this study. The five series included were the NASDAQ composite index, the oil price, the Shanghai Stock Exchange composite index (SSE), the Dollar strength index and the CBOE volatility index (VIX). These variables are dubbed 'meta-data'.

3.6. Summary of data collection and preparation

The data preparation phase of this project consisted of four sections. These were data collection, data cleaning, text data feature engineering and financial data feature engineering. Data collection and cleaning consisted of web scraping, removing HTML tags, converting to lowercase, stopword removal and lemmatization for the text data. Collection of the financial data is a done via a Python script that will always download the entire history of the S&P 500. Cleaning requires only the removal of the adjusted close variable. Text data feature engineering took the form of TextBlob sentiment analysis, VADER sentiment analysis, Word2Vec vectorization and Doc2Vec vectorization. The results of which were two sentiment descriptors from TextBlob, four sentiment descriptors from VADER, a 200-element average descriptor vector from Word2Vec, a 200- and a 20-element vector speech descriptor from Doc2Vec for the speech-centered data, and two 21-element vector speech descriptors from Doc2Vec for the date-centered data for a total of 426 variables (features) engineered from the text data for the speech-centered data and 42 variables engineered from the text data for the date-centered data. The financial data analysis took the form of time series econometrics. This analysis resulted in the fitting of an ARMA(2,0) GARCH(1,1) model to the engineered logDif_Date_Resid variable. The model linearly predicts the dependent variable using the first and second lags of the dependent variable, the first residual and the first lag of variance in the time series. As such, the first and second lags of logDif, the first lag of stdVol_1DateResid and the first lag of absDlogDif are included in the data. Thus, four variables (features) have been engineered using time series analysis of the S&P 500 data. Additionally, a further four auto-regressive variables were added to make up for the flaws inherent in purely linear modelling and six S&P 500 adjacent variables were also selected for addition to the dataset.

4. Experiment design

Two sets of experiments were run on two categories of data design. The first is a speech-centered design. For every speech in this dataset there is a row of data attached. Thus, there is no data for dates when no speeches occurred and there are duplicate dates because there are days when more than one speech occurred. This dataset is the larger of the two datasets and contains 35 251 rows of data – one for each unique speech since 1998-01-01.

Based on the assumption that every speech that occurs on a day affects the closing price of the S&P 500 on that day, the second dataset uses a date-centered design. For every date (trading day) there is one row in the dataset. Thus, on dates when more than one speech occurred, the speech data has been aggregated into a single vector (see section 3.3.2.2 for an explanation of this method). The date-centered dataset contains 25 383 rows of data – one for each trading day between 1950-01-01 and 2022-03-22. Notably, the meta-data is not included in this dataset because two types of test data are included and the decision was taken to focus on them. The speech-centered and date-centered datasets are exemplified in Tables 4.1 and 4.2, respectively.

Table 4.1: Example of the speech-centered data design

Date	Speech	LD_Date_Resid_1	V1	V2	V3
1950-01-02	Good morning...	0.02	3.6	2.6	1
1950-01-06	Ladies and...	-0.05	2.1	2	0.36
1950-01-06	Congress...	-0.05	0.12	3.4	2.8
...					

Table 4.2: Example of the date-centered data design

Date	Speech	LD_Date_Resid_1	Days since last speech	V1	V2
1950-01-02	Good morning…	0.02	0	1.2	1.8
1950-01-03	N/A	0.025	1	1.2	1.8
1950-01-04	N/A	-0.03	1	1.2	1.8
1950-01-05	(Multiple speeches)	0.05	0	2.1	-0.6
1950-01-06	Today marks…	-0.03	0	5.3	-0.86
1950-01-09	N/A	0.04	3	5.3	-0.86
…					

Next, note that for the date-centered data design every consecutive trading day is included in the data. Note also that on days when no speeches occurred the V1 and V2 (representing vectorized speech data) from the last date that a speech occurred are duplicated and the number of days since the last speech is recorded. Finally, note that on days when multiple speeches occurred there is only one row of data – i.e. the speeches are amalgamated into a single vector.

For the speech-centered dataset a total of 1152 models were tested. These consisted of 384 regression tasks and 768 classification tasks. For the date-centered data, 336 models were tested. These consisted of 112 regression models and 224 classification models. The models were selected by incrementally altering a single hyperparameter across five fields for regression and six fields for classification. These fields are detailed in the following sections.

4.1. Regression fields

The five hyperparameter fields for regression were Start_Dates, Remove_duplicates, Reg_Types, Reg_algos and Datasets. Start_Dates refers to the date from which the dataset began: 1998-01-01, 2000-01-01 and 2010-01-01. The 1998 dataset runs from 1998-01-01 until 2022-03-22, the 2000 dataset runs from 2000-01-01 until 2022-03-22 etc.

The Remove_duplicates field divides the speech-centered data into two subsets – one with all the duplicate dates removed (duplicate-exclusive) and one including the duplicate dates (duplicate-inclusive). Duplicate dates occurred because some speeches occurred on the same date. This hyperparameter is necessary because the Meta and Auto datasets only contain one value in each field. Therefore, duplicating dates duplicates Meta and Auto data which creates the risk of data contamination between the train and test sets. This field is not valid for the date-centered data-subset and is set to 'False' for all the models using it.

The Reg_Types field contains two options, TS_Regressor and CS_Regressor. These refer to the manner in which the data is split for cross validation. The TS_Regressor cross-validation method splits the data into five time-consecutive subsets using the 'TimeSeriesSplit' out of sample (OOS) method from scikit-learn. Model training is done on the first subset, then the first and second subset etc. up until the fourth subset. Training scores are calculated for each of the four training subsets and the maximum score is passed forward as the final representation of the training score. Test scores are calculated for only the fifth (and latest) split. This method is employed to force the algorithm to perform prediction instead of interpolation. It avoids giving the models access to the future of a data trend when making predictions for a data point. However, the method is not strictly necessary in this case because the dependent variable has been centered and is close to normally distributed – i.e. it does not have a trend (Bergmeir, Hyndman & Koo, 2018). The CS_Regressor option performs regular 5-fold cross validation.

The Reg_algos field refers to the four different ML algorithms available for training. These are stochastic gradient descent, a three-hidden-layer neural network, multiple linear regression and gradient boosting. The hyper-parameters for each of these fields can be found in Appendix A.1.

Datasets is a complicated field. There are three groups of prediction sub-datasets in the speech-centered dataset – the control subset: X_control ; the meta subset: X_meta ; and the test subset: X_test. X_control contains the auto-regressive variables described in section 3.4.2.2 and 3.5, X_meta contains the meta variables (also described in section 3.5) and X_test contains the variables of focus – the NLP variables. However, in total there are 458 variables across these three subsets. The X_test subset makes up the majority of this. There are four variables derived from the VADER sentiment analysis, two variables from the TextBlob sentiment analysis, 200 variables from Word2Vec, 200 variables from Doc2Vec_200 and 20 variables from Doc2Vec_20.

In order to minimize training times and redundancy the X_test dataset was reduced to contain 26 variables. These are the VADER and TextBlob sentiment scores and the Doc2Vec_20 word-embeddings. The Word2Vec word-embeddings were dropped after underperforming in a series of prediction tasks when compared with the Doc2Vec word-embeddings. The Doc2Vec_20 and Doc2Vec_200 word-embeddings performed similarly on the prediction tasks and combining them did not improve performance. Thus, to minimize the number of variables the ML-models had to contend with, only the Doc2Vec_20 word-embeddings were kept.

The date-centered dataset also contains three groups of prediction subsets. These are the same X_control dataset as in the speech-centered data, and two versions of the vectorized speech data – the distributed bag of words (DBOW) subset and the distributed memory (DM) subset – both detailed in section 3.3.2.2. A detailed description of the variables available and tested is available in Appendix A.3.

All eight X_control variables were included in both the speech-centered and the date-centered datasets. The final X_meta variables selected were Nasdaq_ld_1, Oil_ld_1, SSE_ld_1, USDX_ld_1 and VIX_ld_1. The final X_test variables selected were the VADER scores, the TextBlob scores and the set of 20 Doc2Vec variables, while the DBOW and DM datasets each contained all 20 of their variables and the number of days since the last speech.

The datasets field provided the option for any combination of each data designs' three subsets as training variables for a sum of seven combinations each. Additionally, a PossibleBest set of variables was selected for the speech-centered dataset using an elastic net regression algorithm from scikit-learn (hyperparameters available in Appendix A). It was decided to create a ten variable dataset using four NLP variables, three control variables and three meta variables. The elastic net selected DlogDif_1, DlogDif_2, pos_neg_transform, Nasdaq_ld_1, Oil_ld_1, VIX_ld_1, DV_20_6, DV_20_8, DV_20_13, DV_20_15 as the most prominent variables in each of the three subsets of variables. The elastic net regressions are depicted in Figures 4.1, 4.2 and 4.3.

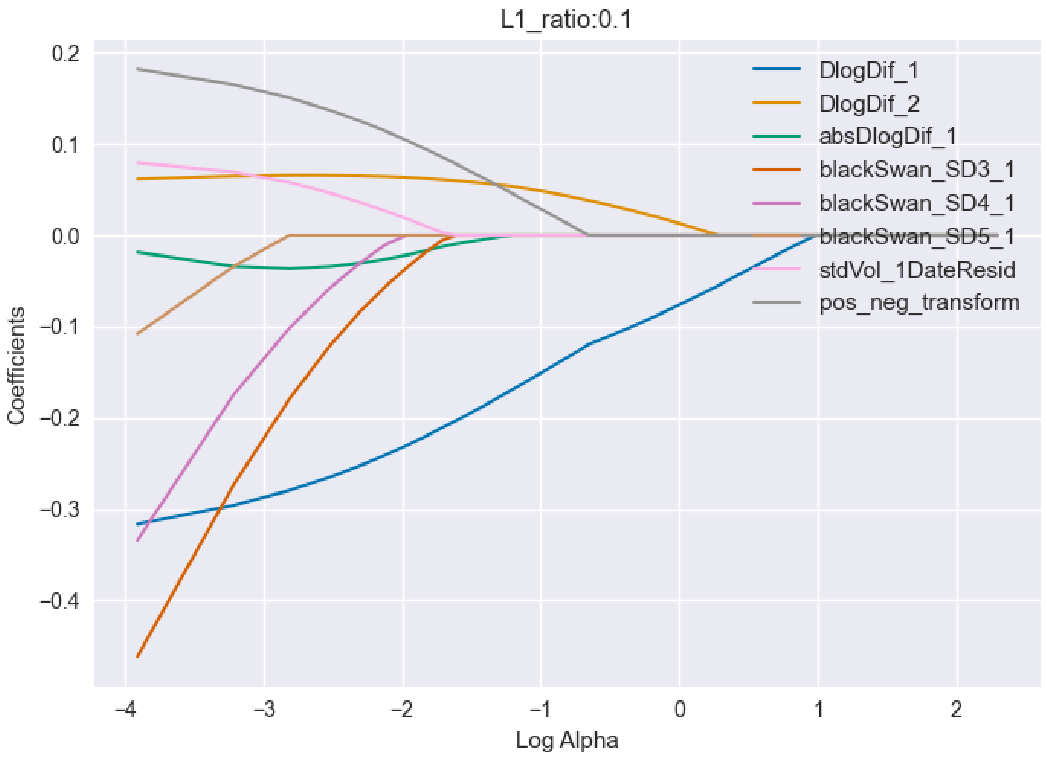

Figure 4.1: Elastic net: control set

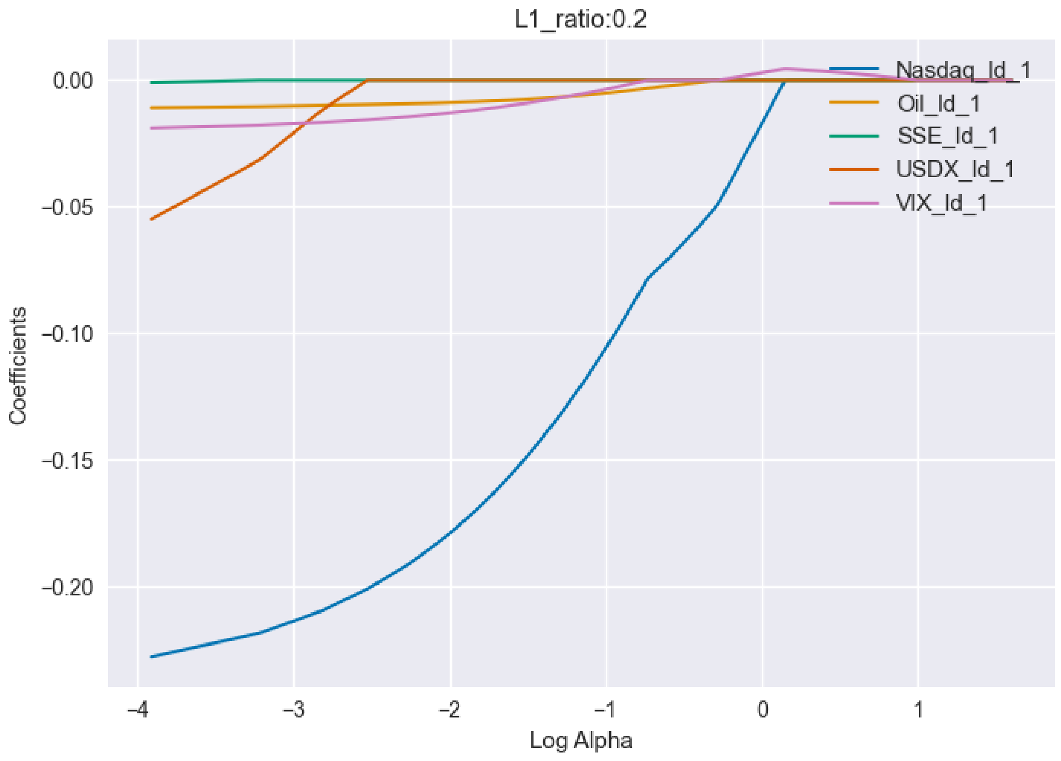

Figure 4.2: Elastic net: meta set

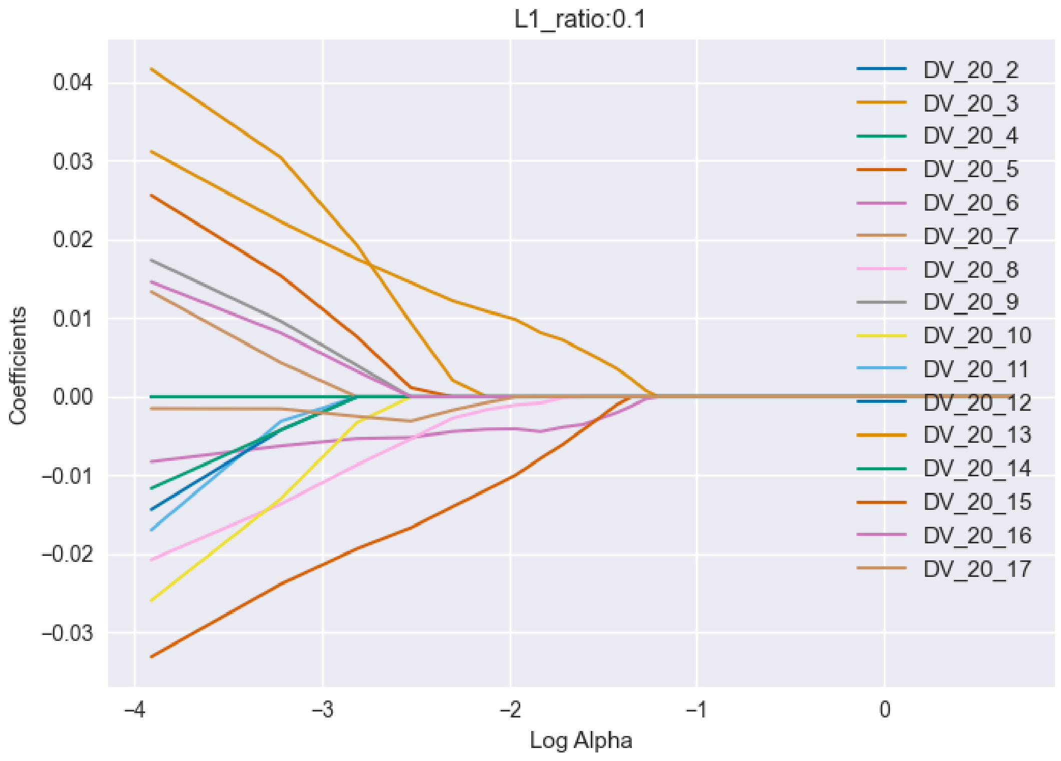

Figure 4.3: Elastic net: test set

4.2. Classification fields

The five fields of hyperparameters for classification were Start_Dates, Remove_duplicates, Binary, Clf_Types, Clf_algos and Datasets. Start_Dates, Remove_duplicates and Datasets are identical to their equivalents in the regression section above.

Binary refers to the Y-variable (classification fields) being predicted. The options for Binary are True and False. If False is selected the continuous Y-variable is split into eight categories denoted by the numbers 1–8. These categories represent a number of standard deviations from the mean of the input Y-variable. See Equation 4.1. If Binary is set to True, then the Y-variable is split into two categories denoted by 1 and 0 which indicate whether the entry is above or below the mean of the continuous Y-variably. See Equation 4.2.

$$Ycat = \begin{cases} 1 & \text{where } \mu - 2\sigma > Ycont \\ 2 & \text{where } \mu - \sigma > Ycont > \mu - 2\sigma \\ 3 & \text{where } \mu - 0.5\sigma > Ycont > \mu - \sigma \\ 4 & \text{where } \mu > Ycont > \mu - 0.5\sigma \\ 5 & \text{where } \mu < Ycont < \mu + 0.5\sigma \\ 6 & \text{where } \mu + 0.5\sigma < Ycont < \mu + \sigma \\ 7 & \text{where } \mu + \sigma < Ycont < \mu + 2\sigma \\ 8 & \text{where } \mu + 2\sigma < Ycont \end{cases} \tag{4.1}$$

$$where\ \sigma = standard\ deviation\ of\ Ycont\ and\ \mu = mean\ of\ Ycont$$

$$Ycat = \begin{cases} 1 & \text{where } Ycont > \mu \\ 0 & \text{where } Ycont < \mu \end{cases} \tag{4.2}$$

$$where\ \sigma = standard\ deviation\ of\ Ycont\ and\ \mu = mean\ of\ Ycont$$

Clf_Types refers to the Time Series cross validation and Cross-section cross validation methods as described in the regression section. The options in this category are CS_Classifier and TS_Classifier. Clf_algos is very similar to the Reg_algos field described in the previous section. The algorithms available for classification are stochastic gradient descent, a three-hidden-layer neural network, logistic regression and gradient boosting. The hyper-parameters for each of these fields can be found in Appendix A.2.

5. Results and discussion

The results of the model tests indicated that U.S. presidential speeches hold at least a small amount of predictive power over movements in the S&P 500 stock index. This conclusion was deduced from the fact that the best performing datasets in both the date-centered classification analysis and the date-centered regression analysis included vectorized speech data. It is also corroborated by the consistent appearance of speech subset inclusive data in the best performing models across both data design types, and both classification and regression tasks.

Models trained on the date-centered dataset outperformed models trained on the speech-centered dataset by about 2 percentage points (~0.58 vs ~0.6) on the test set accuracy score for binary classification analysis. Models using the date-centered dataset also achieved lower mean absolute errors (MAEs) than models trained and tested on the speech-centered dataset. Interestingly, an implication of the superior performance of the (smaller) date-centered dataset is that data with high quality has outperformed data with high quantity. The section below explains the results in more detail.

5.1. Speech-centered classification analysis

Analyzing the best performing classification models, in terms of test set accuracy, across all categories of the speech-centered data reveals a very high likelihood of data contamination in the duplicate-inclusive data. All ten of the top performing models were trained on duplicate-inclusive data and the accuracies range from 0.72 to 0.80. It is highly unlikely that these models have managed to achieve 80% accuracy in prediction of detrended S&P 500 on unseen data over a 71-year period (1950–March, 2022). Thus, results of classification models trained and tested on duplicate-inclusive data have been discarded henceforth.

Analysis of the performance of the remaining (duplicate-exclusive) data revealed that the best performing models were trained on the binary cross-sectional data from 2010 onward. Of the duplicate-exclusive results, the top ten performers contain seven models trained and tested on the 2010 dataset – including the top four. Further, eight of the top ten models performed binary classification. Finally, seven of the ten were trained on cross-sectional cross validation. Thus, further analysis took place within the binary, cross-sectional and 2010 categories.

Having reduced the data to just the cross-sectional binary data from 2010 onward, test set accuracy ranges from 0.54 to 0.58. The mean of the target variable is ∼ 0.5 and thus, every model in the top ten models outperforms chance (recall that the target variable is binary). Six of the ten best models were trained on data including NLP data, while only five of the best models were trained on auto-regression-inclusive data. Meta data was included in eight of them. Notably an NLP-only (gradient boosting) model ranks fourth with a test set accuracy of 0.56. This constitutes evidence of the predictive power of U.S. presidential speeches over S&P 500 movements and lends credibility to the hypothesis of this project.

5.2. Date-centered classification analysis

The means of both the full set of 1950 binary Ys and the test set of 1950 binary Ys are maintained at ∼ 0.5. This implies that any accuracy above 0.5 beats chance. The top ten models in the date-centered classification task achieved test set accuracies between 0.57 and 0.59 and therefore all outperformed chance. All of these ten models were trained on the full range of data: i.e. the 1950 subset. They were also all trained using time-series validation i.e. the TS subset. Seven of the best models were trained on NLP-inclusive data (five DBOW and two DM inclusive sets) and all of the ten best models were trained on auto-regressive inclusive data.

Notably, the date-centered dataset initially outperforms the (duplicate-exclusive) speech-centered dataset by 2 percentage points at the bottom of the range and 1 percentage point at the top of the range in terms of test set classification accuracy. Because of this, the models trained on the date-centered dataset were optimized for maximum performance. Log-regression is initially the best performing classifier in the TS section generally, taking the top two and the seventh positions. However, the models trained on the AutoDBOW dataset outperform the models trained on the Auto only dataset in the Binary NN and the binary gradient boosting categories. Thus, these three models were selected for optimization.

5.2.1. Optimization of classification models for TS classification of date-centered data

The training set accuracy on the 1950 binary AutoDBOW data-subset for the gradient boosting classifier was initially 0.78 indicating that it may have been slightly overfitting the training data. For the NN, the training accuracy is 0.59 which is similar to the test set accuracy and indicates a good fit for the data. Grid search was manually performed to optimize the NN and Gradient Boosting algorithms for the binary 1950 AutoDBOW and Auto datasets.

After closer optimization of hyper-parameters it becomes clear that the AutoDBOW dataset outperforms the Auto dataset on test score, overall and across both the Stochastic Gradient Descent and Log Regression trained models. The best performing algorithm for the AutoDBOW dataset was the LogReg_4 algorithm (please see Appendix A.4 for hyperparameters) which achieved a test set accuracy of 0.601 (precision: 0.6, recall: 0.614) while the best performing algorithm for the Auto dataset was the NN_7 algorithm which achieved a test set accuracy of 0.599 (precision: 0.592, recall: 0.648). This difference indicates that there is at least a slight benefit to including the DBOW dataset in predictive data and corroborates the evidence presented in section 5.1.

5.3. Speech-centered regression analysis

All of the regression models trained on the speech-centered dataset outperformed the series mean as predictors of the Y-variable (logDif_Date_Resid) on the training, test and validation datasets. The cross-sectional cross validation training method (CS data-subset) outperformed the time-series cross validation method (TS data-subset). The models trained on the duplicate-exclusive data and the duplicate-inclusive data performed equally well. The gradient boosting trained models performed best on the duplicate-inclusive data while neural-network trained models were the best performers in the duplicate-exclusive data. Models trained on NLP-inclusive data-subsets performed well across all the categories but did not outperform the exclusively auto-regressive data-subsets. Finally, models trained on the duplicate-inclusive data-subset were dropped for risk of data contamination and models trained on only the NLP data-subset were tested. These still managed to outperform the series-mean as a predictor of the Y-variable and this was taken as further evidence of the predictive power of the NLP data.

The average MAE recorded across all 384 regression models run was 0.87 for the training data, 0.95 for the test data and 0.8 for the validation data. Comparing these with a standard deviation of 1.26 for the logDif_Date_Resid variable in the 1998 subset shows that the average absolute error across all the regressions run lies within one standard deviation of the dependent variable (logDif_Date_Resid) indicating that the regressions are better predictors of the series than the mean of the series. This also holds across each of the three date subsets (1998, 2000, and 2010).

The top ten performing models in regression tasks were all trained on the cross-sectional 2010 dataset and achieved MAEs between 0.68 and 0.7. Interestingly, there was an even split between duplicate-inclusive and duplicate-exclusive datasets. The best performer was duplicate-inclusive; whilst the second, third and fourth best performers were duplicate-exclusive. Analysis of only duplicate-inclusive data trained models gave the top four places to gradient boosting models trained on the AutoMeta, All, Meta and Auto datasets (in that order). The MAEs for the top four performers ranged from 0.68 to 0.7. NLP inclusive datasets ranked in five of the top ten places. Analysis on the duplicate-exclusive top ten performers shows the NN taking eight of the top ten places – including the top seven spots. NLP inclusive dataset trained models ranked well taking the second, third, fourth, fifth and seventh spots but not beating the purely auto-regressive data trained models.

It is difficult to draw a solid conjecture about the predictive power of the NLP data from these mixed results. However, data contamination in the duplicate-inclusive dataset is likely and the results should be discarded. Given the fair performance of the NLP inclusive datasets in the duplicate-exclusive dataset, at this point, it remains likely that the NLP data holds predictive power.

A final analysis of all the regression models trained on only 1998 NLP data from the speech-centered dataset shows that the top ten performing models all achieved a MAE below 1.06. This is still below the standard deviation of 1.26 for the 1998 Y-variably (logDif_Date_Resid) and thus, indicates that regression models trained only on the NLP data outperform the mean of the series as a predictor. This again corroborates evidence of U.S. presidential speeches having predictive power over S&P 500 movements.

5.4. Date-centered regression analysis

In the date-centered regression analysis the models trained on the full dataset (the 1950 subset) using the cross-section cross validation method performed the best – achieving the lowest ten MAEs. These clustered around 0.66 while the standard deviation for the test set was 0.96. Therefore these models are better predictors than the mean of the series. Notably, the date-centered dataset has again outperformed the speech-centered dataset. The best score was achieved by a model trained on an NLP-only data-subset (DBOW) by an SGD algorithm. NLP-inclusive data was used to train seven of the top ten performing models and three of them only used NLP data. Notably, the AutoDM (NLP-inclusive) subset outperformed the Auto (NLP-non-inclusive) subset in the NN model – undeniably indicating the predictive power of the NLP data.

NN algorithms were responsible for six of the lowest ten MAEs and SGD models were responsible for a further three. The DBOW dataset appeared twice, the Auto dataset three times, the DM dataset once, AutoDBOW twice, AutoDM once and AutoBoth once. In total the Auto dataset appeared in seven of the top ten performers, the DBOW appeared in five and the DM dataset appeared three times. Given this and the predictive ability of the DBOW shown in section 5.2, DBOW and Auto data sets were compared across attempted optimizations of the NN and SGD algorithms. However, no significant improvement could be engineered and almost all models performed worse than the initial models. Given that the best performing dataset in terms of test MAE was the DBOW dataset, further evidence is presented that U.S. presidential speeches hold predictive power over the S&P 500.

6. Conclusion

The core conclusion of the research presented here is that U.S presidential speeches do hold predictive power over price changes in the S&P 500 index. This is exhibited by the generally high performance of models trained on speech-vector-inclusive data and more specifically, by the improvement of accuracy in classification models gained from the inclusion of vectorized speech data – both of which are exhibited in section 5. Further, two more useful conclusions can be drawn from this research. First, an ARMA(2,0) GARCH(1,1) model is the best linear model describing the log difference of closing prices in the S&P 500 and second, Doc2Vec outperforms Word2Vec, VADER and TextBlob as a text vectorization method when relating U.S. presidential speeches to fluctuations in the S&P 500. This information should be useful to other researchers looking to explore the relationship between text data and stock market movements as well as traders who need to make informed decisions about the S&P 500.

7. Further research avenues

Improvements in this research could be achieved by the addition of extra variables and/or the exploration of more sophisticated machine learning algorithms. Additional auto-regressive variables could have been included in the control dataset. In particular, the first lag of the residual of an ARMA(1,0) model and the prediction of an ARMA(2,0) GARCH(1,1) may have increased predictive power of the control data-subset (Aras, 2021). More powerful NLP techniques, such as Bidirectional Encoder Representations from Transformers (BERT), could also lead to more comprehensive word-embeddings and better information transfer to the final ML algorithms and better prediction results by extension. Other improvements in the NLP component could come from using more sophisticated methods of parameter selection such as that suggested by Patel & Bhattacharyya (2017) for embedding dimensions or the inclusion of more specific dictionaries such as the emotion-specific one presented in Widmann & Wich (2022).

A lot more research could be done in the political science and linguistic realms using the NLP component of the dataset. For example, it could be interesting to determine which presidents produced the most similar speech vectors and if similarities in speech correlated with similarities in policy decisions or voter behavior. Another example could be creating a timeline of speech vectors and relating modern influential speakers to time periods based on similarity in speech vectors. Further economic research could be done into the macroeconomic or macro-financial effects of certain types of speeches by simply relating the speech vectors to other macro-indicators, such as long-term bond yields or GDP figures.

References

Agarwal, A. 2020. Sentiment analysis of financial news. In IEEE *2020 12th international conference on computational intelligence and communication networks (CICN)*. 312–315.

Aras, S. 2021. Stacking hybrid GARCH models for forecasting bitcoin volatility. *Expert Systems with Applications*. 174:114747.

Baker, S.R., Bloom, N. & Davis, S.J. 2016. Measuring economic policy uncertainty. *The quarterly journal of economics*. 131(4):1593–1636.

Biswas, S., Sarkar, I., Das, P., Bose, R. & Roy, S. 2020. Examining the effects of pandemics on stock market trends through sentiment analysis. *J Xidian Univ*. 14(6):1163–76.

Dilai, M., Onukevych, Y. & Dilai, I. 2021. Sentiment analysis of the US and ukrainian presidential speeches. *Computational linguistics and intelligent systems*. 60–70.

Elbagir, S. & Yang, J. 2019. Twitter sentiment analysis using natural language toolkit and VADER sentiment. In Vol. 122 *Proceedings of the international multiconference of engineers and computer scientists*. 16.

Fama, E.F. 1965. The behavior of stock-market prices. *The Journal of Business*. 38(1):34–105. [Online], Available: http://www.jstor.org/stable/2350752 [2022, August 26].

Géron, A. 2019. *Hands-on machine learning with scikit-learn, keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".

Grimmer, J. & Stewart, B.M. 2013. Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political analysis*. 21(3):267–297.

Hassani, H. & Yeganegi, M.R. 2020. Selecting optimal lag order in ljung–box test. *Physica A: Statistical Mechanics and its Applications*. 541:123700.

Hayo, B., Kutan, A.M. & Neuenkirch, M. 2008. *Communicating with many tongues: FOMC speeches and US financial market reaction*. MAGKS Joint Discussion Paper Series in Economics.

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

Hyndman, R.J. & Athanasopoulos, G. 2018. *Forecasting: Principles and practice*. OTexts.

Jiao, Y. & Jakubowicz, J. 2017. Predicting stock movement direction with machine learning: An extensive study on s&p 500 stocks. In IEEE *2017 IEEE international conference on big data (big data)*. 4705–4713.

Katre, P.D. 2019. NLP based text analytics and visualization of political speeches. *International Journal of Recent Technology and Engineering*. 8(3):8574–8579.

Khedr, A.E., Yaseen, N., et al. 2017. Predicting stock market behavior using data mining technique and news sentiment analysis. *International Journal of Intelligent Systems and Applications*. 9(7):22.

Kinyua, J.D., Mutigwe, C., Cushing, D.J. & Poggi, M. 2021. An analysis of the impact of president trump's tweets on the DJIA and s&p 500 using machine learning and sentiment analysis. *Journal of behavioral and experimental finance*. 29:100447.

Kotzé, K. 2020. [Online], Available: https://kevin-kotze.gitlab.io/tsm/ts-8-note/.

Kotzé, K. 2021. [Online], Available: https://kevinkotze.github.io/ts-12-tut/.

Le, Q. & Mikolov, T. 2014. Distributed representations of sentences and documents. In PMLR *International conference on machine learning*. 1188–1196.

Liu, C., Wang, J., Xiao, D. & Liang, Q. 2016. Forecasting s&p 500 stock index using statistical learning models. *Open journal of statistics*. 6(6):1067–1075.

Loria, S. et al. 2018. Textblob documentation. *Release 0.15*. 2(8).

Maligkris, A. 2017. Political speeches and stock market outcomes. In *30th australasian finance and banking conference*.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. & Dean, J. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*. 26.

Mollick, E. 2006. Establishing moore's law. *IEEE Annals of the History of Computing*. 28(3):62–75.

Pano, T. & Kashef, R. 2020. A complete VADER-based sentiment analysis of bitcoin (BTC) tweets during the era of COVID-19. *Big Data and Cognitive Computing*. 4(4):33.

Patel, K. & Bhattacharyya, P. 2017. Towards lower bounds on number of dimensions for word embeddings. In Taipei, Taiwan: Asian Federation of Natural Language Processing *Proceedings of the eighth international joint conference on natural language processing (volume 2: Short papers)*. 31–36. [Online], Available: https://aclanthology.org/I17-2006.

Purevdagva, C., Zhao, R., Huang, P.-C. & Mahoney, W. 2020. A machine-learning based framework for detection of fake political speech. In IEEE *2020 IEEE 14th international conference on big data science and engineering (BigDataSE)*. 80–87.

Qin, C.J.J. 2018. Natural language processing and event-driven stock prediction. [Online], Available: http://cs230.stanford.edu/projects_spring_2018/reports/8290001.pdf.

Rashid, R. & Leon, A. de. 2019. Making trading great again: Trump-based stock predictions via doc2vec embeddings.

Rehurek, R. & Sojka, P. 2011. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic.* 3(2).

Ren, R., Wu, D.D. & Liu, T. 2018. Forecasting stock market movement direction using sentiment analysis and support vector machine. *IEEE Systems Journal.* 13(1):760–770.

Sazedj, S. & Tavares, J. 2011. Hope, change, and financial markets: Can obama's words drive the market? *Available at SSRN 1976054*.

Shi, B., Zhao, J. & Xu, K. 2019. A Word2vec model for sentiment analysis of weibo. In IEEE *2019 16th international conference on service systems and service management (ICSSSM)*. 1–6.

Sohangir, S., Petty, N. & Wang, D. 2018. Financial sentiment lexicon analysis. In IEEE *2018 IEEE 12th international conference on semantic computing (ICSC)*. 286–289.

Soroka, S., Young, L. & Balmas, M. 2015. Bad news or mad news? Sentiment scoring of negativity, fear, and anger in news content. *The ANNALS of the American Academy of Political and Social Science.* 659(1):108–121.

Vargas, M.R., De Lima, B.S. & Evsukoff, A.G. 2017. Deep learning for stock market prediction from financial news articles. In IEEE *2017 IEEE international conference on computational intelligence and virtual environments for measurement systems and applications (CIVEMSA)*. 60–65.

Widmann, T. & Wich, M. 2022. Creating and comparing dictionary, word embedding, and transformer-based models to measure discrete emotions in german political text. *Political Analysis.* 1–16.

Woolley, J. & Peters, G. n.d. [Online], Available: https://www.presidency.ucsb.edu/.

Yahoo. n.d. [Online], Available: https://finance.yahoo.com/quote/%5EGSPC/history?period1=-1325635200&period2=1641340800&interval=1d&filter=history&frequency=1d&includeAdj usted-Close=true.

Zubair, S. & Cios, K.J. 2015. Extracting news sentiment and establishing its relationship with the s&p 500 index. In IEEE *2015 48th hawaii international conference on system sciences.* 969–975.

Appendix A: Machine Learning Model Specifications and Hyper-Parameters

The following appendix contains technical details regarding machine learning model specifications and hyper-parameters as they were utilized in Python software. For the code used please see the Masters project on my Github profile at github.com/PabloRees/Masters.

A.1. Regression Algorithm Hyper-Parameters

The Stochastic Gradient Descent model uses SciKit Learn's SGDRegressor class. I specified the following hyper-parameters:
- loss =' squared_error'
- max_iter = 10000
- shuffle = True
- random_state = 42
- fit_intercept = False

The Neural Network model uses SciKit Learn's MLPRegressor class. I specified the following hyper-parameters:
- hidden_layer_sizes = (L_1, 8, 1)
- activation = 'relu'
- solver = 'sgd'
- max_iter = 10000
- random_state = 42

If the number of input variables is above 40 then L_1 is set to the number of input variables divided by 8 (rounded up to the nearest whole number). If the number of input variables is smaller than 40 then L_1 is set to 8.

The Multiple Linear Regression model uses SciKit Learn's LinearRegression class. I specified the following hyper-parameters:
- fit_intercept = False
- n_jobs = -1

The Gradient Boosting model uses SciKit Learn's GradientBoostingRegressor class. I specified the following hyper-parameters:
- random_state = 42
- loss =' squared_error'
- learning_rate = 0.1
- min_samples_split = 0.05
- min_samples_leaf = 0.02
- max_depth = 3

A.2. Classification Algorithm Hyper-Parameters

The Stochastic Gradient Descent model uses SciKit Learn's SGDClassifier class. I specified the following hyper-parameters:

- loss =' log'
- random_state = 42
- shuffle = True
- max_iter = 10000

The Neural Network model uses SciKit Learn's MLPClassifier class. I specified the following hyper-parameters:
- hidden_layer_sizes = (L_1, 8, L_3)
- activation = 'relu'
- solver = 'sgd'
- max_iter = 1000
- random_state = 42

If the number of input variables is above 40 then L_1 is set to the number of input variables divided by 8 (rounded up to the nearest whole number). If the number of input variables is smaller than 40 then L_1 is set to 8.

If Binary is True then L_3 is set to 1 while if Binary is False then L_3 is set to 7

The Logistic Regression model uses SciKit Learn's LogisticRegression class. I specified the following hyper-parameters:
- solver = 'lbfgs'
- penalty = 'l2'
- max_iter = 10000
- random_state = 42

The Gradient Boosting model uses SciKit Learn's GradientBoostingClassifier class. I specified the following hyper-parameters:
- random_state = 42
- loss =' squared_error'
- learning_rate = 0.1
- min_samples_split = 0.05
- min_samples_leaf = 0.02
- max_depth = 5

A.3. Datasets and variables

The full X_control dataset contains 13 autoregressive variables from the S&P 500 daily closing price. These are:

- DlogDif – the demeaned log difference of the S&P 500 daily closing price (this should be used as a dependent variable only)
- DlogDif_1 – the first lag of DlogDif
- DlogDif_2 – the second lag of DlogDif
- absDlogDif – the absolute of DlogDif (this should be used as a dependent variable only)
- absDlogDif_1 – the first lag of absDlogDif
- logDif – the log difference of the S&P 500 daily closing price (dependent variable only)

• logDif_Date_Resid – the residual of the regression of logDif on the Date variable
• logDif_Date_Resid_1 – the first lag of logDif_Date_Resid
• blackSwan_SD3_1 – a dummY-variably that is set to 1 if the first lag of the logDif_Date_Resid variable lies outside three standard deviations from its mean and 0 otherwise
• blackSwan_SD4_1 – the same as blackSwan_SD3_1 but for four standard deviations
• blackSwan_SD5_1 – the same as blackSwan_SD3_1 but for five standard deviations
• stdVol_1DateResid – the first lag of the residual of standardized volume regressed on the Date variable
• pos_neg_transform – the first lag of a pseudo-dummY-variably which is set to 1 if DlogDif is positive and set to the ratio of the mean of all negative DlogDifs to the mean of all positive DlogDifs

The full X_meta dataset contains 12 adjacent financial variables that may have correlation with S&P 500 movements. These are:

• BTC_ld_1 – the first lag of the log difference of the daily Bitcoin closing price.
• BTC_dr_1 – the first lag of the residual of the regression of the Bitcoin closing price on the date.
• Nasdaq_ld_1 – the first lag of the log difference of the daily Nasdaq Composite Index closing price.
• Nasdaq_dr_1 – the first lag of the residual of the regression of the Nasdaq Composite Index closing price on the date.
• Oil_ld_1 – the first lag of the log difference of the daily crude oil closing price.
• Oil_dr_1 – the first lag of the residual of the regression of the crude oil closing price on the date.
• SSE_ld_1 – the first lag of the log difference of the daily Shanghai Stock Exchange Composite Index closing price.
• SSE_dr_1 – the first lag of the residual of the regression of the Shanghai Stock Exchange Composite Index closing price on the date.
• USDX_ld_1 – the first lag of the log difference of the daily US Dollar Index closing price.
• USDX_dr_1 – the first lag of the residual of the regression of the US Dollar Index closing price on the date.
• VIX_ld_1 – the first lag of the log difference of the daily Chicago Board Options Exchange's Volatility Index closing price.
• VIX_dr_1 – the first lag of the residual of the regression of the Chicago Board Options Exchange's Volatility Index closing price on the date.

The full X_test dataset contains 426 NLP variables extracted from the US presidential speeches. Six of these are generated by the VADER and TextBlob sentiment analysis tools. They are:

• VaderNeg – The negativity score generated using the VADER tool.
• VaderNeu – The neutrality score generated using the VADER tool.
• VaderPos – The positivity score generated using the VADER tool.
• VaderComp – The compound score generated using the VADER tool. This is a combination of the negativity, neutrality and positivity scores.
• blobPol – The polarity score generated using the TextBlob tool.
• blobSubj – The subjectivity score generated using the TextBlob tool. The higher the score the more subjective a text is.

200 of these are generated using the Word2Vec text vectorization tool. These variables do not contain specified meaning but are rather the average of final parameters from hidden layers in neural net-

works performing the pseudo-task of word prediction. Their purpose is to relate words to each other depending on the context of each word within the broader corpus. These are annotated as:

- WV_0
- WV_1
- ...
- WV_198
- WV_199

The final 220 variables are generated using the Doc2Vec text vectorization tool. The Doc2Vec tool works similarly to the Word2Vec tool except that during training a second hidden layer is incorporated. This second layer is persisted across each word in a text in the training corpus and its hyper parameters adjusted so as to create a single vector describing the text as a whole. This tool was used to create two sets of variables (each one a hidden layer from a pseudo-task). These are a 200 variable set, annotated as:

- DV_200_0
- DV_200_1
- ...
- DV_200_198
- DV_200_199

and a 20 variable set, annotated as:

- DV_20_0
- DV_20_1
- ...
- DV_20_18
- DV_20_19

A.4. Hyper-parameter tuning for TS Classification models

Two datasets and four models were focussed on during the hyper-parameter tuning for in the TS Classification category. The two datasets were the 1950-01-01 AutoPVDBOW dataset and the 1950-01-01 Auto dataset. While the model focus was on neural networks (NN) and gradient boosting (GB) for the AutoPVDBOW dataset and on logistic regressions (Logreg) and stochastic gradient descent (SGD) for the Auto dataset. Only models that yielded improvements in some score or another are included here. No regression hyper-parameter tuning is reported here because the tuning failed to improve the performance of the regression models.

A.4.1. Time series classification 1950-01-01 AutoPVDBOW tuning

NN_1:

```
elif clf_type == 'clf_NN':
    inputVars = len(XVars)

    if inputVars / 5 < 8:
        L_1 = 8
    else:
        L_1 = round(inputVars / 8)

    if binary:
        L_3 = 2
    else:
        L_3 = 7

    clf = MLPClassifier(hidden_layer_sizes=(L_1, 8, L_3), activation='relu',
                        solver='sgd', max_iter=10000, random_state=random_state)
```

Dataset: Auto
Train Scores: Accuracy: 0.590 Precision: 0.598Recall: 0.794
Test Scores Accuracy: 0.573 Precision: 0.566 Recall: 0.838

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.590 Precision: 0.592 Recall: 0.708
Test Scores Accuracy: 0.582 Precision: 0.573 Recall: 0.683

NN_2:

```
elif clf_type == 'clf_NN':
    inputVars = len(XVars)
    if inputVars / 5 < 8:
        L_1 = inputVars
        L_2 = inputVars
    else:
        L_1 = inputVars
        L_2 = inputVars

    if binary:
        L_3 = 2
    else:
        L_3 = 7

    clf = MLPClassifier(hidden_layer_sizes=(L_1, L_2, L_3), activation='relu',
                        solver='sgd', max_iter=10000, random_state=random_state)
```

Dataset: Auto

Train Scores: Accuracy: 0.59 Precision: 0.598 Recall: 0.795
Test Scores Accuracy: 0.573 Precision: 0.567 Recall: 0.839

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.591 Precision: 0.598 Recall: 0.682
Test Scores Accuracy: 0.594 Precision: 0.592 Recall: 0.645

NN_3: L3 = 5

```
elif clf_type == 'clf_NN':
    inputVars = len(XVars)
    if inputVars / 5 < 8:
        L_1 = 2*inputVars
        L_2 = inputVars
    else:
        L_1 = 2*inputVars
        L_2 = inputVars

    if binary:
        L_3 = 5
    else:
        L_3 = 7

    clf = MLPClassifier(hidden_layer_sizes=(L_1, L_2, L_3), activation='tanh',
                        solver='sgd', max_iter=10000, random_state=random_state)
```

Dataset: Auto
Train Scores: Accuracy: 0.586 Precision: 0.601 Recall: 0.598
Test Scores Accuracy: 0.599 Precision: 0.592 Recall: 0.648

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.594 Precision: 0.606 Recall: 0.664
Test Scores Accuracy: 0.587 Precision: 0.573 Recall: 0.694

NN_4: solver = ‘adam’

```
elif clf_type == 'clf_NN':
    inputVars = len(XVars)
    if inputVars / 5 < 8:
        L_1 = 2*inputVars
        L_2 = inputVars
    else:
        L_1 = 2*inputVars
        L_2 = inputVars

    if binary:
        L_3 = 2
    else:
        L_3 = 7

    clf = MLPClassifier(hidden_layer_sizes=(L_1, L_2, L_3), activation='relu',
                        solver='adam', max_iter=10000,
                        random_state=random_state)
```

Dataset: Auto
Train Scores: Accuracy: 0.601 Precision: 0.619 Recall: 0.674
Test Scores Accuracy: 0.587 Precision: 0.59 Recall: 0.777

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.96 Precision: 0.937 Recall: 0.991 - overfit
Test Scores Accuracy: 0.526 Precision: 0.527 Recall: 0.621

NN_5: solver = adam, early_stopping = True

```
elif clf_type == 'clf_NN':
    inputVars = len(XVars)
    if inputVars / 5 < 8:
        L_1 = 2*inputVars
        L_2 = inputVars
    else:
        L_1 = 2*inputVars
        L_2 = inputVars

    if binary:
        L_3 = round(inputVars/2)
    else:
        L_3 = 7

    clf = MLPClassifier(hidden_layer_sizes=(L_1,L_2,L_3), activation='relu',
                        solver='adam',
                        max_iter=10000, random_state=random_state,
                        early_stopping=True)
```

Dataset: Auto
Train Scores: Accuracy: 0.595 Precision: 0.6 Recall: 0.702
Test Scores Accuracy: 0.581 Precision: 0.563 Recall: 0.736

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.605 Precision: 0.621 Recall: 0.657
Test Scores Accuracy: 0.592 Precision: 0.585 Recall: 0.793

GB_1:

```
elif clf_type == 'clf_GradientBoosting':
    clf = GradientBoostingClassifier(random_state=random_state,learning_rate=0.01,
                        min_samples_split=0.025,min_samples_leaf=0.01,max_depth=8)
```

Dataset: Auto
Train Scores: Accuracy: 0.506 Precision: 0.491 Recall: 0.412
Test Scores Accuracy: 0.513 Precision: 0.500 Recall: 0.416

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.512 Precision: 0.5 Recall: 0.441
Test Scores Accuracy: 0.511 Precision: 0.498 Recall: 0.397

GB_2:

```
else : #clf_type == 'clf_GradientBoosting'
    clf = GradientBoostingClassifier(random_state=random_state,
                        learning_rate=0.0001,
                        min_samples_split=0.01,min_samples_leaf=0.001,
                        max_depth=12)
```

Dataset: Auto
Train Scores: Accuracy: 0.529 Precision: 0.529 Recall: 1.0
Test Scores Accuracy: 0.523 Precision: 0.523 Recall: 1.0

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.529 Precision: 0.529 Recall: 1.0
Test Scores Accuracy: 0.524 Precision: 0.524 Recall: 1.0

A.4.2. Time series classification 1950-01-01 Auto tuning

Logreg_1:

```
elif clf_type == 'clf_logreg':
    clf = LogisticRegression(solver='lbfgs', penalty='l2', max_iter=10000,
                             random_state=random_state)
```

Dataset: Auto
Train Scores: Accuracy: 0.589 Precision: 0.598 Recall: 0.665
Test Scores Accuracy: 0.594 Precision: 0.594 Recall: 0.607

Logreg_2: fit_intercept = False

```
elif clf_type == 'clf_logreg':
    clf = LogisticRegression(solver='lbfgs', penalty='l2', max_iter=10000,
                             random_state=random_state,fit_intercept=False)
```

Dataset: Auto
Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.667
Test Scores Accuracy: 0.594 Precision: 0.589 Recall: 0.633

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.679
Test Scores Accuracy: 0.596 Precision: 0.601 Recall: 0.583

Logreg_3: penalty=elasticnet

```
elif clf_type == 'clf_logreg':
    clf = LogisticRegression(solver='saga', penalty='elasticnet', max_iter=10000,
                      l1_ratio=0.5,random_state=random_state,fit_intercept=False)
```

Dataset: Auto
Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.666
Test Scores Accuracy: 0.593 Precision: 0.588 Recall: 0.63

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.68
Test Scores Accuracy: 0.6 Precision: 0.599 Recall: 0.615

Logreg_4: L1_ratio = 0.4

```
elif clf_type == 'clf_logreg':
    clf = LogisticRegression(solver='saga', penalty='elasticnet', max_iter=10000,
                      l1_ratio=0.4,random_state=random_state,fit_intercept=False)
```

Dataset: Auto
Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.666
Test Scores Accuracy: 0.593 Precision: 0.588 Recall: 0.63

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.679
Test Scores Accuracy: 0.601 Precision: 0.6 Recall: 0.614

SGD_1:

```
if clf_type == 'clf_SGD':
    clf = SGDClassifier(random_state=random_state, shuffle=True, loss='log',
                       max_iter=10000)
```

Classification type: TS_Classifier: Classification algo: clf_SGD
Dataset: Auto
Train Scores: Accuracy: 0.575 Precision: 0.597 Recall: 0.965
Test Scores Accuracy: 0.591 Precision: 0.6 Recall: 0.968

SGD_2: fit_intercept = False

```
if clf_type == 'clf_SGD':
    clf = SGDClassifier(random_state=random_state, shuffle=True, loss='log',
                        max_iter=10000, fit_intercept=False)
```

Dataset: Auto
Train Scores: Accuracy: 0.59 Precision: 0.609 Recall: 0.694
Test Scores Accuracy: 0.591 Precision: 0.595 Recall: 0.754

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.577 Precision: 0.601 Recall: 0.689
Test Scores Accuracy: 0.574 Precision: 0.601 Recall: 0.929

SGD_3: learning_rate = adaptive

```
if clf_type == 'clf_SGD':
    clf = SGDClassifier(random_state=random_state, shuffle=True, loss='log',
                        max_iter=10000, fit_intercept=False,
                        learning_rate='adaptive', eta0=0.0001)
```

Dataset: Auto
Train Scores: Accuracy: 0.583 Precision: 0.597 Recall: 0.606
Test Scores Accuracy: 0.595 Precision: 0.596 Recall: 0.6

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.588 Precision: 0.595 Recall: 0.662

Test Scores Accuracy: 0.595 Precision: 0.59 Recall: 0.631

SGD_4: n_iter_no_change = 30

```
if clf_type == 'clf_SGD':
    clf = SGDClassifier(random_state=random_state, shuffle=True, loss='log',
                max_iter=10000, fit_intercept=False,
                learning_rate='adaptive', eta0=0.0001, n_iter_no_change=30)
```

Dataset: Auto
Train Scores: Accuracy: 0.587 Precision: 0.597 Recall: 0.635
Test Scores Accuracy: 0.594 Precision: 0.59 Recall: 0.625

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.59 Precision: 0.596 Recall: 0.671
Test Scores Accuracy: 0.597 Precision: 0.591 Recall: 0.644

SGD_5: n_iter_no_change = 100

```
if clf_type == 'clf_SGD':
    clf = SGDClassifier(random_state=random_state, shuffle=True, loss='log',
                    max_iter=10000, fit_intercept=False,
                    learning_rate='adaptive', eta0=0.0001, n_iter_no_change=100)
```

Dataset: Auto
Train Scores: Accuracy: 0.59 Precision: 0.599 Recall: 0.663
Test Scores Accuracy: 0.595 Precision: 0.591 Recall: 0.631

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.591 Precision: 0.6 Recall: 0.681
Test Scores Accuracy: 0.597 Precision: 0.59 Recall: 0.652

SGD_5: eta0=0.001

```
if clf_type == 'clf_SGD':
    clf = SGDClassifier(random_state=random_state, shuffle=True, loss='log',
                    max_iter=10000, fit_intercept=False,
                    learning_rate='adaptive', eta0=0.001, n_iter_no_change=100)
```

Dataset: Auto
Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.666
Test Scores Accuracy: 0.594 Precision: 0.591 Recall: 0.627

Dataset: AutoPVDBOW
Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.679
Test Scores Accuracy: 0.597 Precision: 0.593 Recall: 0.631