Erasmus University Rotterdam
Erasmus School of Economics
Master Thesis Quantitative Finance

Predicting stock markets using news articles

Virgile Sadri (660856)

Supervisor:	Eoghan O'Neill
Second assessor:	Anastasija Tetereva
Date final version:	28th January 2024

Acknowledgement

I would like to express my gratitude to my thesis supervisor M.O'Neill, for his guidance and his support throughout my master's thesis. His guidance and his ideas were helpful and led to the completion of this master's thesis.

Abstract

This paper aims at investigating the impact of newspaper articles on stock prices predictions from big-cap companies. We collect the text data using web scrapping methods. We transform our news data into embeddings with numerical values. Then, we use a support vector machine, a random forest, a ridge regression, and a neural network to predict stock prices using two datasets with one of them containing news data. We use statistical tests to assess the improvements of the model when trained on news data. We find that news article data bring slight improvements to some models but are detrimental to some other models due to the number of parameters to estimate and the volatility of some stocks.

Contents

1 Introduction

Fama (1970) introduced the theory of the efficient market hypothesis, which implies that all publicly known information about a company impacts its stock price. Nowadays, we have much more means to understand these markets and more sophisticated models than Fama (1965), who modeled stock prices as random walks. Nonetheless, we are still unable to predict financial markets accurately.

Financial markets are essential to the global economy, enabling buyers and sellers to exchange various financial instruments such as stocks, bonds, commodities, and currencies. In particular, stock markets play a crucial role in promoting investments and economic growth, offering investors opportunities to earn returns on their investments. However, investing in the stock market is not riskless, and can cause significant losses. This highlights the importance of developing an optimized investment strategy. The challenge, yet, lies in the unpredictability of future returns, making the creation of a reliable strategy extremely difficult.

Thus, stock return predictions are a vital area of research. The ability to understand and predict stock market movements is of major importance, but the complex nature of stock markets makes this a daunting task, as pointed out by Gandhmal and Kumar (2019). Most existing models for predicting stock prices focus primarily on quantitative variables, such as historical prices, macroeconomic factors, or Fama-French factors, which have a direct and measurable impact on stock prices (Jan and Ayub (2019) and Nti et al. (2019)). However, relying only on these variables may not be sufficient. There is a wealth of information that these quantitative measures do not capture, leading to gaps in the accuracy of stock price predictions, as explained by Shah et al. (2019). This inaccuracy can be costly for investors since even small differences in stock prices can result in significant financial losses.

Recognizing these limitations, it becomes clear that integrating different types of information could improve the prediction of stock prices. This is where the value of textual data becomes evident. Textual data encompasses a broad spectrum of information that is often missed by quantitative analyses. For instance, market sentiment, influenced by

news events such as the outbreak of Covid, can cause significant fluctuations in stock returns that quantitative factors alone may fail to predict as highlighted by Szczygielski et al. (2023). Incorporating textual data, such as news articles and financial reports, into predictive models could provide a wider view of the factors influencing stock prices. By doing so, these models can adapt more dynamically to a range of events, potentially outperforming those that rely solely on quantitative data.

Moreover, the use of advanced data analytics techniques, such as natural language processing (NLP) and machine learning, can further improve the effectiveness of incorporating textual data into stock market analysis. NLP can process and interpret vast amounts of unstructured textual data, extracting relevant information that could impact stock prices. This information includes management discussions in annual reports, political events, economic policies, and even social media trends; all of these pieces of information can provide valuable insights into market dynamics.

Furthermore, integrating textual data with traditional quantitative analysis can lead to the development of more robust and adaptable investment strategies, as done by Sun et al. (2016) with social media text and Khedr et al. (2017) with news sentiment. By analyzing both structured and unstructured data, investors and financial analysts can gain a more comprehensive understanding of the market, enabling them to make more informed decisions. This approach also allows for the identification of emerging trends and potential risks that might not be obvious through quantitative analysis alone.

Finally, the incorporation of textual data into stock market analysis represents a significant advancement in the field of financial analytics. By combining the insights gained from textual data with traditional quantitative methods, investors and analysts can develop more accurate and effective strategies for alleviating the complexities of the stock market. This approach not only could enhance the prediction of stock returns but could also contribute to a more sophisticated and nuanced understanding of the financial markets as a whole.

The goal of this paper is to answer the following question:

To what extent are stock price predictions impacted by the information coming from news articles?

In this paper, we first collect some news articles to build our dataset, and we transform this data using a Word2Vec model so that it can be exploited by machine learning models. Then, we build regression models to predict stock returns by using solely quantitative features combined with news data, and we compare it to the benchmark model regressions which are the same models but using only quantitative features. We use the following regression models: Ridge regression, support vector regression, random forest, and a neural network. The accuracies of the models are then compared by using the root mean squared error and by using different tests such as Diebold-Mariano, introduced by Diebold and Mariano (2002). We also use a Model confidence set based on a Superior Predictive Ability test, introduced by Hansen et al. (2011), to test the significance of the results' difference. Our study differs from priors studies by not evaluating each text's sentiment to keep all the the information from the articles, like in Sun et al. (2016). It also differs by trying to predict stock prices directly instead of predicting stock price movements, so that we are in a regression framework and not in a classification case.

We find that news data can be detrimental to models due to the amount of features they bring. In some cases, news data enables to reduce high prediction errors. Nevertheless, the difference in the prediction observed by adding news data is not significant. Ultimately, models trained with news data are chosen by statistical tests to be among the best models. This result stands for stocks relatively volatile such as Microsoft or Apple.

The paper is organized as follows. Section 2 highlights the work that has already been done on the same topic or similar topics, section 3 shows the data that has been used for this study along with some quick analyses of the data, section 4 explains the methodology in this paper, which is how we handle textual data and how we predict stock returns. Then, section 5 shows the results obtained and provides some discussions and some interpretations of the results. Finally, section 6 summarizes the key findings of our study and suggests possible improvements that could be made to this paper.

2 Literature

As explained in the introduction, being able to predict stock returns and understand their variations is crucial to building an investment strategy. The literature on this subject is large. Sun et al. (2016) try to predict price movements of stocks by using text data retrieved from social media due to its relation with financial markets. They use a sparse matrix factorization to handle textual data and select their features. However, while their model using textual data outperforms their benchmark model, it has a low accuracy. Their paper differs in two ways from other papers. They use social media data instead of news data and they do not evaluate the sentiment of their textual data. They finally use their predictions to build a trading strategy that has better returns than their benchmark trading strategies. Wu et al. (2022) predict stock prices from the China Shanghai A-share market. They use financial text data, historical prices and technical indicators. They use a word2vec model combined with a convolutional neural network to get investor's sentiments. Then, they use a long short-term memory neural network to predict stock prices. They find that their model outperforms traditional models that use a single data source. Hagenau et al. (2013) emphasize on the importance of feature selection when using textual data. They try different feature extraction methods such as the dictionary approach, the n-gram and the bag-of-words. They also use two different feature selection methods that are the chi-square and the bi-normal-separation. Then, they represent their text using the tf-idf method and predict stock prices using an artificial neural network, a support vector machine and a naive Bayes classifier. They find an improvement in models when a proper feature selection is performed. Kalra and Prasad (2019) use companies-specific news articles to predict future price movements of stocks. They first perform a sentiment analysis on their textual data to categorize them into two categories: positive sentiment and negative sentiment. Nonetheless, they use the count of positive sentiment and negative sentiment text instead of text sentiment directly. After that, they use machine learning models such as k-nearest neighbor, support vector machine, neural network and naive Bayes to predict future price movements. The use of these data to train classifiers gives a significant enhancement compared to state of art methods.

Text data is also used in the economic context to predict economic variables. The first

issue met with text data is the number of features coming from it. As one word is seen as one feature, it is crucial to find ways to deal with the amount of features. Gentzkow et al. (2019) provide a complete review to the use of text as an input to economic research by explaining the steps to transform text data and the existing methods to extract the relevant data from the text. They also focus on the case of stock price prediction and explain the three ways to handle textual data when predicting stock prices. These methods consist of a dictionary-based approach to get a text sentiment, linear regressions with a penalization term and generative language models such as latent Dirichlet allocation. Mikolov et al. (2013) introduce a method to create word representation in a vector space, called word2vec, enabling maths calculations on words such as additions. Ash and Hansen (2023) give a review of the existing methods to represent words as numerical vectors with modern methods. These methods imply the simple bag-of-words to the word2vec model and the GloVe model, giving a representation of a word with respect to its context. Finally, some richer models declined from the word2vec model give the best models that exist known as GPT and PALM. These models differ from the word2vec model by adjusting the weight of the context words when computing the probability of a word given its context. They finally address four issues that are often met in most of the research dealing with textual data in economy and explain the approaches that have been taken to tackle these issues.

Kalamara et al. (2022) provide a comparison of different machine learning models to predict macroeconomic variables using economic signals from newspapers by extracting term frequencies from it and assessing if the models are improved by adding newspaper information to an autoregressive model with a single lag. They observe an improvement of the results with the Neural Network and the Ridge Regression by assessing a significant difference of the root mean squared error by using a Diebold-Mariano test and a Superior predictive ability test. Their conclusion leads to an improvement in the prediction, especially in periods of economic crises. Gardner et al. (2022) provides a study on the influence of FOMC statements on equity prices and compares it to the influence of macroeconomic news announcements on equity prices. They use FOMC statements to describe the state of the economy and compute the sentiment of these statements. They conclude that statements have a bigger impact on equity prices during bad states of the

economy (i.e. high unemployment rates and economic recessions) times. They also find that the FOMC statement is the best variable to explain the variation in stock prices due to macroeconomic news. They describe better the state of the economy than macroeconomic news announcements. However, they find that during uncertain periods, like the covid pandemic, stock prices are more driven by uncertainty than FOMC statements.

3 Data

3.1 Stock price data

The dataset contains stock prices from the New York Stock Exchange (NYSE) and Nasdaq of three different companies namely Apple, Microsoft, and Tesla. It is a time-series dataset going from 1st January 2018 to 30th December 2021 with a daily frequency. The dataset contains 1045 observations, which does not correspond to the number of days between the date ranges since there are some days when stock prices are not actualized. The dataset has been collected on the eikon datastream (Reuters, 2022).

The three companies are chosen so that we have news articles available every day. Apple and Microsoft are two of the largest companies in the world, we can find press articles talking of them nearly every day. Tesla is also a large company and has been chosen because even if it is not one of the largest companies in the world, it often appears in news articles, especially during the studied period. Tesla has also very volatile stocks, and we want to see if our models can be suited to predict volatile stocks.

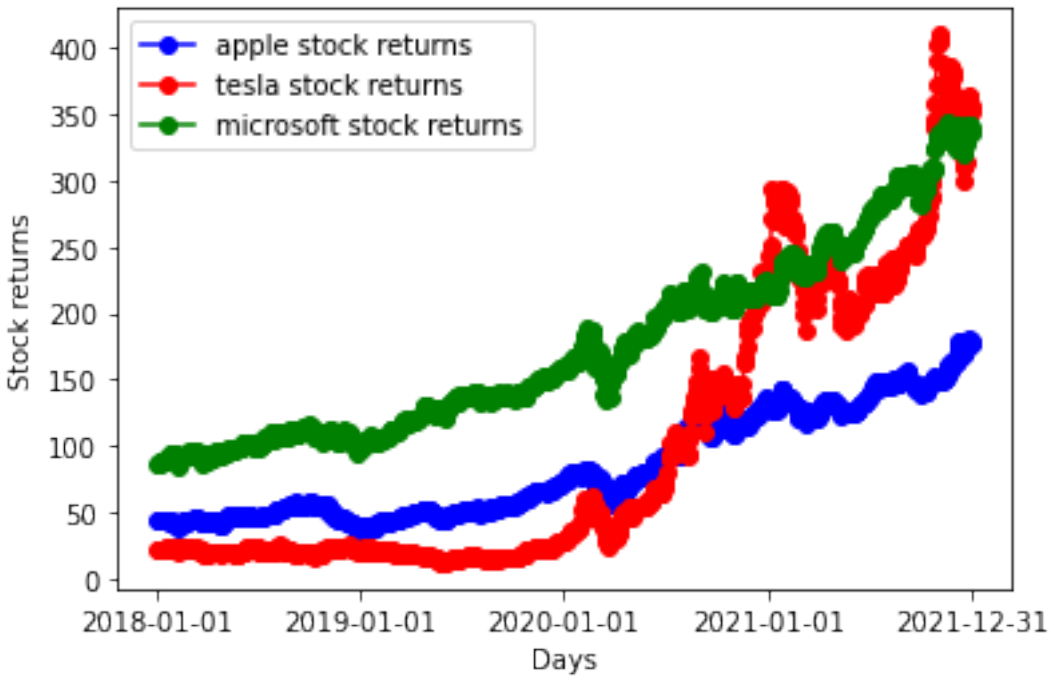

Figure 1: Daily stock returns over the studied period

According to figure 1, the stock data collected is quite volatile and has somewhat similar

trends for Apple and Microsoft. However, we see that Tesla had a huge increase in stock returns during the studied period especially after 2020 during the Covid outbreak with a lot of peaks and variations in the returns. We see that Tesla had the steepest rise, while Apple and Microsoft had a steady rise. We then can see if the news dataset can have a better prediction on stocks with high volatility like Tesla or stocks with low volatility like Apple, or if it does not have any incidence.

We have the autocorrelation plot of each company's stock return below:

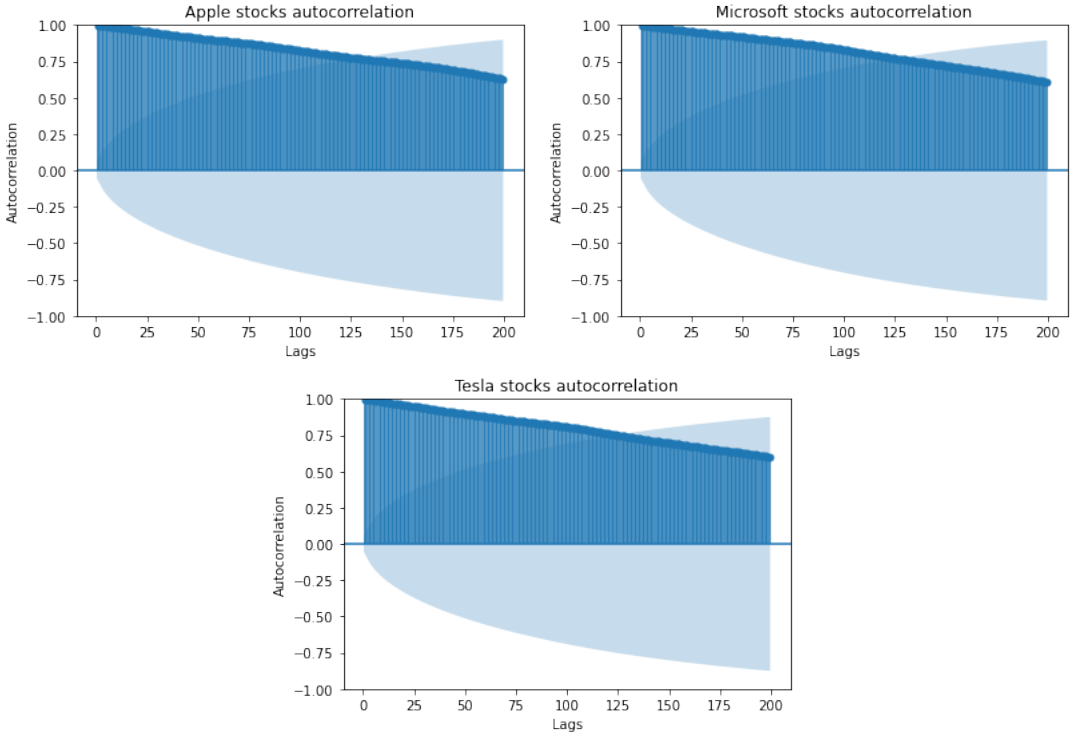

Figure 2: Autocorrelation plots of companies' stock returns

According to figure 2, the autocorrelations have a slow decay and are not quickly declining to 0. Furthermore, the confidence band shows that the correlations are statistically significant for the first hundred lags. Consequently, our stock returns are not stationary.

Yet, we can focus on the difference in prices from one day to another. We integrate our stock prices by subtracting the price of the stock the day before from the price of the actual day. By doing this, we want to transform our series into stationary ones. We have the autocorrelation plot of the difference in stock prices presented in figure 3. The figure below shows a straight decline in the autocorrelation for all the companies'

stocks. Additionally, we observe that the vast majority of autocorrelation lags are inside the confidence band, thus they are statistically not significant. This indicates we have a stationary time series and that stock prices are integrated into order one. By making these times-series stationary, we make them more predictable as they have a fixed mean and variance (Hyndman and Athanasopoulos, 2018).

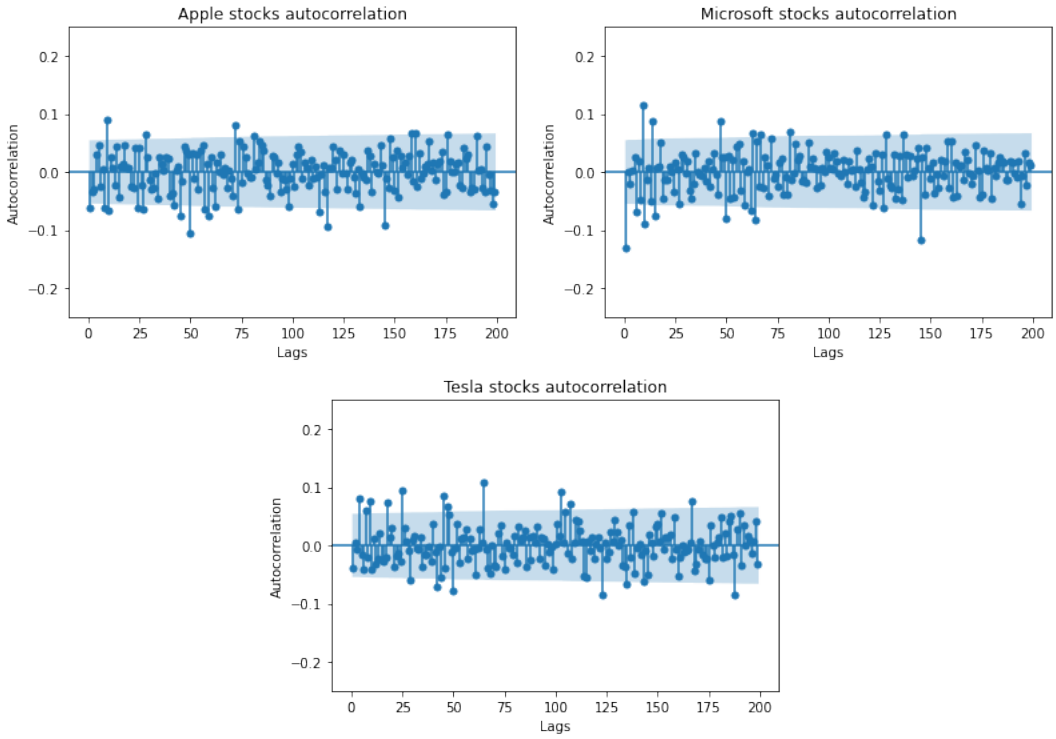

Figure 3: Autocorrelation plots of companies' stock price difference

3.2 Quantitative features dataset

The second dataset contains four different macroeconomic features, found in Andersen et al. (2007), that are namely the GDP, the unemployment rate, the consumer price index (CPI), and the retail sales in the United States since the three companies are American, and we focus on American stock markets. We chose these features because they are one of the few attributes available at a daily frequency, and they have an impact on stock markets according to Srivastava (2010) and Verma and Bansal (2021). It is a time-series dataset going from 1st January 2018 to 30th December 2021 with a daily frequency and the dataset contains 1045 observations. In addition to these macroeconomic factors, the dataset contains the Fama-French 3 research factors from Fama and French (1996) for the corresponding date range. This dataset contains 1008 observations. The final dataset,

obtained by merging the two datasets, contains 1008 observations. The macroeconomic factors have been collected from the eikon datastream Reuters (2022).

3.3 News dataset

The news dataset contains news articles every day from 1st January 2018 to 30th December 2021. The news articles come from many sources such as The New York Times or Wall Street Journal. The data also contains company-specific news on Apple, Microsoft, and Tesla but also some news about stock markets such as Nasdaq or NYSE as done in Khedr et al. (2017).

The data has been collected using the New York Times API by Den Heijer (2024) and the Googlesearch API by Hseb (2017). We use multiple queries such as 'Microsoft', and 'NYSE' for the corresponding date range. Once a result is found, the link of the article is extracted from the API and then scrapped using the requests package Chandra and Varanasi (2015), which gives the HTML code of the webpage. The article is then extracted from this code using the BautifulSoup package (Richardson, 2007). Each article is stored as one value in the dataset; an article can be found with its index being the date and the column being the query used to find it. This gives five columns since there are five different queries and 1460 observations, which correspond to the total number of days between 1^{st} January 2018 and 30^{th} December 2021.

However, the data presents some missing values since there are no news articles about every company every day and the proportion of missing data is relatively large since 1041 observations out of 1460 have at least one missing value. The number of missing articles is 1779 out of 7300 articles. Moreover, some articles contained some part of the text that did not seem to be part of the original article like warning messages such as 'Please subscribe to our newsletter for more articles from The Times' so we cleaned the data by filtering some words. However, the cleaning step cannot be perfect and there are still some articles with unrelated text. The missing data handling is treated in section 4.2.

Then, we also clean the data by transforming the text corpus as done in Ash and Hansen

(2023), we first tokenize (splitting on whitespace/punctuation) and put all the texts in lowercase, we then remove the punctuation and all the symbols from the articles. Additionally, we remove all the English stop words that are present in a predefined list provided by the nltk Python package Bird et al. (2009). We can find words like the/is/so, and we lemmatize the words to remove suffixes for example doing becomes do, in the end all of these steps enable to have only the important tokens that are relevant to understand articles. The lemmatization removes suffixes of words with respect to their context, their meaning in the sentence, and the word's part of speech, and they enable a feature selection in the text data to lower the computation expenses and also avoid the loss of meaning of the sentences. The tale 1 summarizes the different datasets explained in this section.

	Stock prices dataset	News articles dataset	Quantitative factors dataset
Number of observations	1045	1460	1008
Frequency	Daily	Daily	Daily
Number of variables	3	5	7
Type of data	Numeric	String	Numeric
Start date	1^{st} January 2018	1^{st} January 2018	2^{nd} January 2018
End date	31^{st} December 2021	31^{st} December 2021	31^{st} December 2021

Table 1: Table summarizing the data

4 Methods

4.1 Text handling

4.1.1 Word2Vec models

Numerous models enable textual data handling. We want a way to convert textual data into a numeric component, and that's why we chose the Word2Vec models.

The Word2Vec model represents each word in a vector space of 100 dimensions where each word is represented by a linear combination of these 100 vectors and thus, it enables us to do calculations on our words. For example, if we make an addition of the word cat and baby, we get an embedding that is very close to the one of the word kitten. Therefore, this model allows us to do some calculations on words and thus, we can make an average of the words to get the representation of one sentence.

However, Word2Vec models are a family of models that represent words to embeddings and the most common ones are the Skip-Gram and the Continuous Bag-of-Word (CBOW). Since we are dealing with news articles, according to Jang et al. (2019), it is better to use the CBOW method as the Skip-Gram is better for smaller texts.

4.1.2 Continuous Bag-of-Words

The CBOW model predicts a target word based on the context of the surrounding words in a sentence or text. It is trained using a feedforward neural network with three layers where the input is a set of context words in a sentence, and the output is the target word. The model learns to adjust the weights of the neurons in the hidden layer to produce an output that is most likely to be the target word. Let N be the number of words in the corpus. The input layer takes a representation of every word in the corpus by a one-hot encoded vector denoted $x_w \in \mathbf{R}^N$ where $\forall i \in [1, N], x_w[i] = 1$ if i is the index of the word w in the vocabulary and zero otherwise.

Then for each word, we define a context composed by C surrounding words, which we note $w_1, w_2, ..., w_C$ represented by their respective vectors $x_{w_1}, x_{w_2}, ..., x_{w_C}$. Furthermore,

in the hidden layer, we define a weight matrix $W \in \mathbf{R}^{N \times M}$ that we initialize with random values, M being the dimension of the embedding space. We then compute the average of the context word embeddings, this step gives the hidden layer representation:

$$h = \frac{1}{C} W^T \sum_{i=1}^{C} x_{w_i} \tag{1}$$

Finally, in the output layer, we define a second weight matrix called $V \in \mathbf{R}^{M \times N}$ that we also initialize with random values. Then, for each word w with index i in the vocabulary, we define a score by:

$$u_i = V_{:,i}^T * h \tag{2}$$

The notation $V_{:,i}$ indicates the i^{th} column of the matrix V

We then use these scores to build a probability distribution using the Softmax function:

$$p(w_i | w_1, ..., w_C) = \frac{\exp(u_i)}{\sum_{j=1}^{N} \exp(u_j)} \tag{3}$$

and the word with the highest probability is chosen to be the target word.

We then want to find estimates for the matrixes W and V, firstly, the CBOW tries to maximize the likelihood of the correct target word given the context which is:

$$\mathbf{L} = \sum_{w \in \text{Vocabulary}} \log(p(w | w_1, ..., w_C)) \tag{4}$$

Now, we compute the errors between the observed probability distribution and the predicted probability distribution, and we use backpropagation to compute the gradients of W and V noted $\nabla_W \mathbf{L}$ and $\nabla_V \mathbf{L}$. We finally update the matrixes using stochastic gradient descent, and we repeat this for the number of epochs we want. The explanation of stochastic gradient descent is given in section 4.2.4. In the end, we can find words' embeddings inside the matrix W, each line of the matrix corresponds to the embedding of

one word.

For the hyperparameter values which are the size of the window for context words, the dimension of the embedding, and the epochs, we followed the common values for these parameters explained in Major et al. (2018) and we take a window of five words with a 100 dimension in the embeddings and 100 epochs. We use the gensim package on Python introduced by Řehřek and Sojka (2010) to fit our Word2Vec model.

4.1.3 Aggregating to the whole document

Now that we have embeddings for each word in our vocabulary, we want to find embeddings for our article so that we have a way to quantify them. A simple method would be to average the words' embeddings inside an article to get an embedding for the article, but this implies that each word has the same weight no matter their frequencies. This implies that a redundant term has the same weight as any word and thus less weight than two words that are synonyms. Therefore, it is necessary to account for the frequency, it also outperforms the average approach according to Zhu et al. (2016).

Instead, we use a weighted average, where the weight of each word depends on its occurrence. To measure this occurrence, we use the Term Frequency-Index document frequency (tf-idf) score as in Ash and Hansen (2023) which gives better results than the simple average.

The tf-idf score is a score that enables the measurement of the frequency of a word in a corpus and is computed the following way, we first compute the term frequency in the document by:

$$\text{TF} = \frac{\text{number of times the word appears in the document}}{\text{number of words in the document}} \tag{5}$$

and we compute the inverse document frequency by:

$$\text{IDF} = \log\left(\frac{\text{number of documents in the corpus}}{\text{number of documents where the term appears}}\right) \tag{6}$$

In the end, we combine the two equations 5 and 6 to get the tf-idf score of a word in a document:

$$TF\text{-}IDF = \text{TF} \times \text{IDF} \tag{7}$$

Finally, for an article indexed by subscript a containing words $w_1, ..., w_n$ with their respective embeddings noted $e_{w_1}, ..., e_{w_n}$, we compute its embeddings using the formula 7 to get:

$$e_a = \frac{\sum_{i=1}^{n} TF - IDF_{w_i}^a \times e_{w_i}}{\sum_{i=1}^{n} TF - IDF_{w_i}^a} \tag{8}$$

with $TF - IDF_{w_i}^a$ being the TF-IDF score of the i^{th} word in article a.

4.2 Prediction models

After transforming our news articles into numerical vectors, we can now merge our data frame to have one data frame being the combination of news article embeddings and quantitative features and a second data frame containing only the quantitative features. Then, we scale both of these datasets using a MinMax scaler. It is necessary to scale our data for many reasons. First, it reduces the computation time and enables regression models to converge faster. Secondly, it puts the same variance on all features and this is also necessary for the models that use distance measures such as support vector machine. Having different variances for each feature could lead to giving more importance to features having a bigger magnitude. The MinMaxscaler scales each observation x_{ij} with the following formula:

$$\hat{x}_{ij} = \frac{x_{ij} - \min_{i \in [1,N]} x_{ij}}{\max_{i \in [1,N]} x_{ij} - \min_{i \in [1,N]} x_{ij}} \tag{9}$$

In addition, we handle the missingness of news articles mentioned in section 3.3 after quantifying articles. Since we have time-series data, we want to pay attention to data leakage so we cannot use future observations to impute a missing value for a present

observation. Hence, we use the previous observation to handle missing values, with the method called last observation carried forward. This method consists of taking the previous observation's value if there is a missing value. That way we ensure that we do not have future information inside the present information and we assume that if there is not any news on a company at a certain day then nothing important happened, and thus we keep the same information as yesterday.

Furthermore, since we have time-series data, a lot of issues arise if we want to use cross-validation such as the non-stationarities, the use of future observations in the training step, or the serial correlation in the data. Cross-validation is still possible according to Bergmeir and Benítez (2012) for purely autoregressive series. For stationary series, Bergmeir et al. (2018) show that standard cross-validation is possible, but not as robust as other types of cross-validations. As shown in section 3, our stock returns are stationary but they are not purely autoregressive. Therefore, we chose to implement a nested rolling window to train our model and prevent the issue of data leakage and dependent data. We divide our data frames into 30 windows with 400 observations to train the model on a one-day prediction and one observation to test it in each window. We choose these numbers so we have enough observations to train our models in each window, and we predict the one-day ahead prediction on the next day after the training set to see the performance of our model to give a one-day ahead prediction in the short-term. We focus the analysis on the next day after the training set, as the models are trained to predict this observation specifically. We also impose a gap of 20 days between each window. The gap of 20 days also enables us to use every observation of the dataset without having too many windows.
The first window starts on 11^{th} January 2018 and ends on 14^{th} August 2019. The first window does not start on 1^{st} of January because the observations are removed due to missing values. The last window starts on 4^{th} May 2020 and ends on 1^{st} December 2021. The gap between each window is 20 days.

For each of the following prediction models, each model is fitted with the dataset containing news and quantitative features. The benchmark is the same model with solely the quantitative features.

4.2.1 Ridge Regression

Our first model to predict stock prices is Ridge regression, introduced by Hoerl and Kennard (1970), which gives some of the best prediction results according to Kalamara et al. (2022) when using textual data. Here we use a ridge regression where the features are the macroeconomic variables and the news embeddings to predict the stock returns. Our benchmark is the same model but only with the quantitative dataset. The ridge regression model is a linear model written the following way:

$$y_{t+1} - y_t = \alpha + \nu^T X_t + \mu^T S_t + \epsilon_{t+1} \tag{10}$$

We compare the performance of this model to the same one without the text information present in S_t.

However, unlike ordinary least squares, the ridge regression has the following cost function:

$$J(\theta) = \frac{1}{N}(y_i - \sum_{i=1}^{N}(\alpha + \nu^T X_t + \mu^T S_t)^2) + \lambda||\theta||^2 \tag{11}$$

with $\theta = [\alpha, \nu, \mu]$

The ridge regression has an explicit solution for the estimate of the parameters, displayed in equation 12. While this formula is straightforward, it is not the one used in real-life scenarios when we have large datasets. The large number of observations and features makes the computation relatively long. Furthermore, the inversion of the matrix is computationally expensive and unstable. Hence, we use another method to estimate the parameters, that is less computationally intensive.

$$\hat{\theta} = [(J^T J + \lambda \mathbf{I})^{-1} J^T y, (X^T X + \lambda \mathbf{I})^{-1} X^T y, (S^T S + \lambda \mathbf{I})^{-1} S^T y] \tag{12}$$

with $J = [1, 1, ..., 1]^T$

We minimize this cost function to find the optimal parameter α, ν, μ by using a Stochastic Average Gradient Accelerated method (SAGA) described in Defazio et al. (2014). First,

we initialize the parameters α, ν, μ and another vector, $g \in \mathbf{R}^N$ and we compute $\bar{g} = \sum_{i=1}^{N} g_i$. Then for each iteration, we choose a random observation i and we compute the loss gradient function at this point without the penalty term:

$$\begin{aligned} \nabla J_i(\alpha, \nu, \mu) &= [\frac{\partial J_i(\theta)}{\partial \alpha}, \frac{\partial J_i(\theta)}{\partial \nu}, \frac{\partial J_i(\theta)}{\partial \mu}] \\ &= -2(y_i - \alpha - \nu^T X_t - \mu^T S_t)[1, X_t, S_t]^T \end{aligned} \tag{13}$$

Then, we adjust the gradient and include the penalty term

$$\tilde{g} = \nabla J_i(\theta) - g_i + \bar{g} + 2\lambda\theta \tag{14}$$

and we update the parameter vector $\theta = \theta - \tilde{g}$. Finally, we update the value of g_i by $\nabla J_i(\theta)$ and we also update $\bar{g}$ by $\bar{g} = \bar{g} + \frac{1}{N}(\nabla J_i(\theta) - g_i)$. We continue this process until we reach the maximum number of iterations or until the change of θ between two iterations becomes very small.

The regularization term, which is the sum of the squared parameters, enables the model to shrink the coefficients, especially the large ones. The ridge regression tends to shrink the coefficients towards 0, but none of the coefficients are set to 0. λ represents the strength of the regularization and needs to have the right value so that the model reduces the training error and also shrinks the parameters. If λ is close or equal to 0, we will just get the OLS estimate, however, if $\lambda \hookrightarrow \infty$ then the parameters will be close to 0 and the regression will not try to reduce the training error term.

To find the optimal value for λ, we use a grid search with a time-series cross-validation, as explained in Hart (1994). A time-series cross-validation solves the issue of dependent data due to its temporality and works by splitting the data into k equal folds and when the i^{th} fold is used as the validation set, the training set is all the folds from the first to the $i - 1^{th}$, with $i \in [2, k]$. Since we predict at a one-day horizon, we train our model to predict the next day so we need one observation in the validation set and we use ten folds. The figure below shows the functioning of time-series cross-validation.

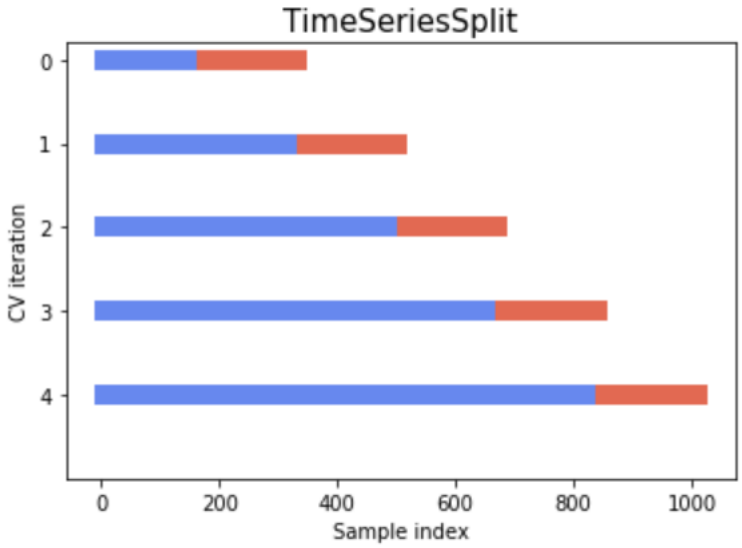

Figure 4: The different folds in time-series cross validation

Then, we measure the error inside the validation set with the following formula:

$$\text{Error} = \frac{1}{k-1} \sum_{i=2}^{k} |y_i - \hat{y}_i| \tag{15}$$

we take the λ corresponding to the smallest cross-validated error.

We do this for both datasets, the one with news information and the one without. The two datasets are trained with the same observations and test the same observations.

4.2.2 Variable selection

For the other models, our news dataset embeddings contain hundreds of features thus, it is necessary to apply some feature selection, and we do this by using Random forest (Breiman, 2001). Random forest is a supervised learning method that consists of combining multiple decision trees to build a model, which makes it a type of bagging. Random forests are used for regression and classification tasks, they can even be used for variable selection, as explained in Breiman (2017). Firstly, the dataset is divided into subsets using bootstrap sampling, we note N the number of subsets, which is also the number of trees in the random forest. Each subset is used to train one tree. The randomness of the random forest comes from the split of the decision trees. At each split, only a random subset of features is used. This creates diversity among the trees and reduces the

correlation between them.

Then, each tree is trained until it reaches its maximum depth, or the minimum number of terminal node observations, so it can catch complex relationships. Finally, the prediction is done by computing the average of each tree's prediction. By taking the average of multiple trees' prediction, the model is less likely to overfit the training data.

The random forest requires tuning its hyperparameters, which are the depth of the trees, the number of trees, the number of features we use at each split, the minimum number of observations in a leaf (i.e terminal node), and the minimum number of observations to perform a split. These hyperparameters are tuned using a random grid search through the possible values of the hyperparameters. We use a time-series cross-validation like in 4.2.1 to define the training set and the validation set, and we take the hyperparameters leading to the lowest error in the validation set.

Random forest has many advantages compared to single trees and other machine learning models. First, it gives a feature importance evaluation, giving the most prominent features used for the prediction. Secondly, it reduces the variance and the risk of overfitting compared to a single tree without increasing bias.

The feature importance is computed by computing the impurity at each node for each tree. The impurity is a measure of how accurate the node prediction is. For a regression task, we take the mean squared error (MSE) as the impurity. And we compute the decrease in MSE due to a split on feature F. We note $N_{\text{parent node}}$ the number of observations in the parent node, and we keep the same notations for the other nodes.

$$\text{MSE} = \frac{1}{N_{\text{node}}} \sum_{i=1}^{N_{\text{node}}} (y_i - \hat{y}_i)^2 \tag{16}$$

$$\begin{aligned} &\Delta\text{MSE}(F) \\ &\quad = \text{MSE}_{\text{parent node}} - \left(\frac{N_{\text{left node}}}{N_{\text{parent node}}} \times \text{MSE}_{\text{right node}} + \frac{N_{\text{right node}}}{N_{\text{parent node}}} \times \text{MSE}_{\text{right node}}\right) \end{aligned} \tag{17}$$

The importance of the feature is then computed by averaging the MSE decrease over the

trees.

$$\text{Importance}(F) = \frac{1}{N_{trees}} \sum_{i=1}^{N_{trees}} \sum_{\text{node} j \in \text{tree i}} \Delta\text{MSE}(F, i, j) \tag{18}$$

With N_{trees} being the number of trees, $\Delta\text{MSE}(F, i, j)$ is the MSE decrease of variable F in tree i at node j. If the split at node j in tree i is not performed on the feature F, then $\Delta\text{MSE}(F, i, j) = 0$. The importance is then normalized across features to have a value between 0 and 1, by dividing each variable's importance by the sum of the features' importance.

Finally, we fit a random forest using the whole dataset, and we take the most important variables to perform the feature selection. After that, we fit another random forest using only the important features, and we fit the other models with these important features.

4.2.3 Support Vector Machine

After selecting our variables, we use a support vector machine (SVM) to predict the stock returns because it can detect non-linear relations between the target and the features using different kernels. SVM is designed to find the best hyperplane to fit the target. They are initially made for classification, but a variant introduced by Drucker et al. (1996), can be used for regression. The goal of SVM is to find a function noted f that can predict a continuous variable y with a set of variables noted X (this time X denotes the entire dataset that contains the lag return, articles' embeddings, and the quantitative features) with a degree of tolerance noted ϵ. However, this function must have a prediction error inferior to ϵ for each training data point and must be the flattest. We also introduce slack variables that we add to the degree of tolerance to allow for errors and approximations.

Let's denote w a weight vector and $b \in \mathbf{R}$ such that:$f(x) = w^T x + b$. The minimization problem of SVM is the following:

$$\begin{aligned} \min_W \frac{1}{2} w^T w + C \sum_{i=1}^{N} (\xi_i + \xi_i^*) \\ \text{subject to} \\ y_i - w^T x - b \leq \epsilon + \xi_i \\ w^T x + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{19}$$

with ξ_i and ξ_i^* being slack variables measuring the violation of the degree of tolerance and C being the parameter measuring the trade-off between the goodness of fit and the flatness of f.

We compute the Lagrangian of this minimization problem by denoting the Lagrange multipliers by α and α^*:

$$\begin{aligned} L(w, b, \xi_i, \xi_i^*, \alpha_i, \alpha_i^*, \mu_i, \mu_i^*) = \frac{1}{2} W^T W + C \sum_{i=1}^{N} (\xi_i + \xi_i^*) - \sum_{i=1}^{N} \alpha_i [y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) - \epsilon - \xi_i] \\ - \sum_{i=1}^{N} \alpha_i^* [(\mathbf{w} \cdot \mathbf{x}_i + b) - y_i - \epsilon - \xi_i^*] - \sum_{i=1}^{N} (\mu_i \xi_i + \mu_i^* \xi_i^*) \end{aligned} \tag{20}$$

Then we use the following equalities $\frac{\partial L}{\partial W} = 0$, $\frac{\partial L}{\partial b} = 0$, $\frac{\partial L}{\partial \xi_i} = 0$, $\frac{\partial L}{\partial x i_i^*} = 0$ to get:

$$\begin{aligned} W = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) x_i \\ \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0 \\ C = \alpha_i + \mu_i = \alpha_i^* + \mu_i^* \end{aligned} \tag{21}$$

We can now write $f(x) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) x_i^T x + b$. Thus, the function f is determined by finding the values of α_i and α_i^*. We also introduce a kernel function $K(x_i, x_j)$ that measures the similarity of two observations in higher dimensions. This kernel function is necessary when dealing with non-linearly separable data, it represents data in a higher dimensional space where the data becomes linearly separable. The different kernels function considered is the linear kernel which is the simple dot product $K(x_i, x_j) = x_i^T x_j$, the polynomial kernel $K(x_i, x_j) = (1 + x_i^T x_j)^d$ with d being the degree of the kernel, the ra-

dial kernel $K(x_i, x_j) = \exp^{-\gamma||x_i - x_j||^2}$ with $\gamma \geq 0$. With this kernel function, the objective function becomes

$$f(x) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) K(x_i, x) + b \tag{22}$$

The coefficients α_i and α_i^* are determined during the training of the model by solving the minimization problem using sequential minimal optimization (SMO) explained in Fan et al. (2005).

Finally, we tune the hyperparameters of the model, which are the kernel, the degree of tolerance for the error ϵ, and C the regularization parameter. We find these parameters using a random grid search. We define a grid of values for each hyperparameter, and we take random combinations of these values. We try 100 different combinations of hyperparameters and we do not go over this number so that the computation time is not too long. These combinations are then used to train the model, and we use time-series cross-validation to define multiple pairs of training and validation sets. We take the combination of hyperparameters giving the lowest average error (defined in 15) over the validation sets.

4.2.4 Neural Network

Neural networks are powerful methods to find the relation between a variable and a set of features. Neural networks work like human neurons: they consist of layers of neurons with nodes that are connected. The first layer is the input layer that contains as many neurons as there are features in the dataset. Then, neural networks are composed of hidden layers, where the computations are done. Each neuron takes multiple inputs and uses a linear combination to give an output. We note $a_j^{(k)}$ the output of the j^{th} neuron in the k^{th} hidden layer. We have the following equality:

$$a_j^{(k)} = h(z_j^{(k)}) = h(\sum_{i=1}^{M^{(k)}} \beta_{ji}^{(k)} a_i^{(k-1)} + b_j^{(k)}) \tag{23}$$

with $M^{(k)}$ being the number of neurons in the k^{th} layer, h being the activation function, $b_j^{(k)}$ being the bias term of neuron j in the k^{th} layer, the term between brackets being the

layer and $\beta_{ji}^{(k)}$ being the weight of observation i in neuron j in the k^{th} layer.

For the first hidden layer, the equation is:

$$\begin{aligned} a_j^{(1)} &= h(z_j^{(1)}) \\ &= h(\sum_{t=1}^{T} \beta_{jt}^{(1)} x_t + b_j^{(1)}) \end{aligned} \tag{24}$$

Finally, we have the output layer, which is composed of only one neuron and takes the outputs of the last hidden layer as an input to give a prediction. The aggregation of the inputs to get the output is the same as in the hidden layers, and the output layer has also an activation function.

Then, once our network architecture is chosen, we initialize the parameters β and b with random values and we make predictions with the observations in the training set. Having this, we can compute the loss, which is the root mean squared error in our case, and measure the difference between the predicted value and the actual value. The goal is to find the values of β and b that minimize this loss. Thus, we compute the gradient of the loss with respect to the parameters using the chain rule with the following formula:

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial \beta} \tag{25}$$

with L being the loss function, $\hat{y}$ the predicted value, z the output of the neuron and β a parameter.

We find these optimal parameters by using the gradient descent algorithm. We update the weights with the following formula:

$$\begin{aligned} \beta_{ij} &= \beta_{ij} - \eta \frac{\partial L}{\partial \beta_{ij}} \\ b_j &= b_j + \frac{\partial L}{\partial b_j} \end{aligned} \tag{26}$$

with η being the learning rate. The learning rate scales the size of the update and determines how much the weights are updated. When the update of the weight is too small

(i.e. the gradient is getting close to 0), the algorithm stops.

We also divide the training data into batches. For each batch, the algorithm updates the weights so that the updating does not use all data points at once and uses subsets. This makes the training less time-consuming and gives better results. We repeat all these steps for the number of epochs chosen. The number of epochs is the number of times we use each data point. If the algorithm does not converge with the weight update, it stops when it reaches the number of epochs.

The number of neurons in the hidden layers and the number of hidden layers is something we decide, but the more neurons we have in a layer the more capable it will be of catching linear patterns, and we choose these parameters in accordance with Heaton (2008). The activation function is also something we need to choose. Since we are performing a regression, we choose the Relu function in the hidden layers and the identity for the output layer. We set the batch size at either 16 or 32 observations so that our updates are not noisy, but we do not increase the batch size further due to limited computational resources. The hyperparameter we need to tune is the learning rate, which plays a crucial role. If the learning rate is too low, our gradient descent algorithm might take a very long time to converge, and if it is too large the algorithm might not be able to find the optimal parameters.

4.3 Assessing the results

After fitting all our models in each window, we compute the root mean squared error (RMSE) at horizon h through all the windows with the following formula:

$$\text{RMSE}^h = \sqrt{\frac{1}{N_{\text{windows}}} \sum_{i=1}^{N_{\text{windows}}} (y_i^h - \hat{y_i^h})^2 :} \tag{27}$$

The RMSE is used to compare the accuracy of the models. The model with the biggest RMSE is the least effective one and thus, we can see if adding news data improves our predictions.

We also use a second metric called the mean absolute error (MAE), which also measures

the effectiveness of a model. It is less influenced by outliers. We compute the MAE through all the windows with the following formula:

$$MAE = \frac{1}{N_{\text{windows}}} \sum_{i=1}^{N_{\text{windows}}} |y_i^h - \hat{y_i^h}| \tag{28}$$

We also look at the absolute error of each model in each window to see if the performance differs from one window to another. The different windows have a different economic context and therefore, we can see which model is more or less dependent on the economic context.

4.3.1 Diebold-Mariano test

The Diebold-Mariano test is a statistical test. It gives a way to compare two forecasting models by assessing if their errors are significantly different.

We build this test the following way. First, we take the error in each window for each horizon and each model.

$$\begin{aligned} e_{1h}^k &= y_h^k - \hat{y_{1h}^k} \\ e_{2h}^k &= y_h^k - \hat{y_{2h}^k} \end{aligned} \tag{29}$$

where e_{1h}^k is the error for the observation in the k^{th} window at horizon h for the model trained on the quantitative dataset and e_{2h}^k is the error for the observation in the k^{th} window at horizon h for the model trained on the complete dataset.

Then, we take the squared error and compute the difference between these squared errors. We then sum these errors over the windows:

$$\bar{d_h} = \frac{1}{K} \sum_{k=1}^{K} ({e_{1h}^k}^2 - {e_{2h}^k}^2) = \frac{1}{K} \sum_{k=1}^{K} d_h^k \tag{30}$$

Having d_h, we can now set up the test. The test is built as follows. The null hypothesis is that the two models have the same forecasting error ($H_0 : E_k[d_h^k] = 0$). We build the following test statistic:

$$DM = \frac{\bar{d_h}}{\sqrt{\frac{\hat{\sigma}^2}{K}}} \tag{31}$$

with $\hat{\sigma}^2$ being the sample variance of d_h^k, so $\hat{\sigma}^2 = \frac{1}{K}\sum_{k=1}^{K}(d_h^k - d_h)^2$. Usually, the term $\hat{\sigma}^2$ is the long-term variance, like in Diebold and Mariano (2002), which includes a term for the autocorrelation, but in this case, the errors are not correlated with each other because of the nature of the sets. There are at least 20 days between each window so we can consider that there is no issue linked to the temporality of the data. therefore, we take the sample variance.

Under the hypothesis H_0, the test statistic DM, defined in 31, asymptotically follows a normal law $\mathbf{N}(0, 1)$. Thus, we compute the p-value, which is the probability of getting a result at least as extreme as the observation under H_0. If we have a low p-value, it means that our test statistic is an extreme value and that the hypothesis H_0 may not be true.

4.3.2 Superior predictive ability test with model confidence set

Another possible test is the Superior Predictive Ability (SPA) test, introduced by Hansen (2005). The SPA test can compare two models to test if one model is better than the benchmark. It uses a blocked bootstrap method. First, we create bootstrap samples and for each sample, we compute the values $\bar{d}_h$ and d_h^k defined in 31. We compute the test statistic:

$$S_h = \frac{\bar{d}_h}{d_h^k} \tag{32}$$

We take the number of bootstrap samples to be 1000 so we have 1000 values for S_h. These 1000 values define the distribution of S_h. The value of S_h on the test set is computed and compared to the bootstrap distribution. The p-value is computed by calculating the proportion of the bootstrap values greater than the observed test statistic.

$$\text{p-value} = \frac{1}{N_b}\sum_{i=1}^{N_b}\mathbb{1}(S_h \geq S_h^i) \tag{33}$$

with N_b being the number of bootstrap samples,S_h^i the test statistic in the bootstrap sample i and S_h the test statistic in the test set.

However, SPA test does not handle the issue of data snooping. Data snooping can happen when we perform a lot of tests and one of them gives a significant difference in the prediction performance. Nonetheless, it is not because one test says there is a significance that

it is true because the test has a probability of being wrong: when we perform multiple tests, the likelihood of having one false positive becomes non-negligible. We alleviate this by implementing a model confidence test based on a SPA test. The model confidence set (MCS) is introduced by Hansen et al. (2011). Another advantage of the MCS is that it gives a set of models that are considered to be the best given a level of confidence.

We start with a set of models $\mathcal{M}$, containing all the models, thus we have 8 models in this set. Now, we test each model pairwise using the SPA test, and we remove the least performing model from the set $\mathcal{M}$. We reiterate this step until we have only one model remaining in the set. Then, we want a p-value for each model to know which models stand out. We perform the following hypothesis at every step $H_0 : E(d_{ij,k}) = 0 \forall i, j \in \mathcal{M}$ with $d_{ij,k} = e^k_{i1} - e^k_{j1}$ by taking the notations from equation 29. We use the SPA test to perform the test, and we compute p-values for each model. The best models have a p-value of 1 and the least performing models have p-values close to 0. The p-values of the MCS are computed using the p-values of the SPA test at each iteration, and the computation is explained in Hansen et al. (2011). The model confidence set is then built by taking models with p-values higher than the level of significance.

5 Results and discussions

5.1 One-day horizon accuracy

We analyze the accuracy of the one-day prediction on the stock price following the training set in each window.

5.1.1 Apple performance

The results are presented in Table 2 and Table 3, where we can see the MAE and the RMSE for each model and each dataset. We see that the models have similar performances, except for the neural network and the ridge regression. In general, the quantitative dataset gives better performance than when it is combined with news. One reason is that news brings a lot of variables and therefore, the models need more observations and time to be fitted. Moreover, it could mean that the information given by the news data is not very informative or does not add any information to the dataset. Finally, there is an improvement of the MAE using news data with the SVR, but the RMSE is larger with the news dataset, meaning that on average, predictions are more accurate when using news data, but they are also more volatile.

	Quantitative dataset	News dataset
Ridge regression	1.09	1.52
Support Vector Machine	1.35	1.27
Random Forest	1.21	1.38
Neural Network	1.21	1.36

Table 2: MAE of the one-day ahead prediction for Apple stocks

	Quantitative dataset	News dataset
Ridge regression	1.50	1.80
Support Vector Machine	1.88	2.07
Random Forest	1.62	1.79
Neural Network	1.61	2.08

Table 3: RMSE of the one-day ahead prediction for Apple stocks

We also look at the performance in each window, and we have the absolute error in each window for each model presented in figure 7. SVR and ridge give very volatile results compared to the random forest and the neural network but have slightly lower RMSE and MAE. We also see a peak for every model at the 21^{th} window where news data perform

worse than the quantitative data. There is another peak at the 16^{th} window where news data bring improvements to the models. We also have higher errors from the 16^{th} to the 21^{th} window. These windows have a large part of their training set during the beginning of the Covid outbreak, when financial markets were uncertain and volatile. During this period, some models using non-text data reach their highest peak, and these are sometimes lower for the same model using text data (e.g. random forest, SVR). This indicates that news can add some information to the dataset and enable models to limit their errors when they have poor performances. Thus, the addition of news data does not bring a direct improvement to the forecasts but enables the models to be more robust during uncertain contexts.

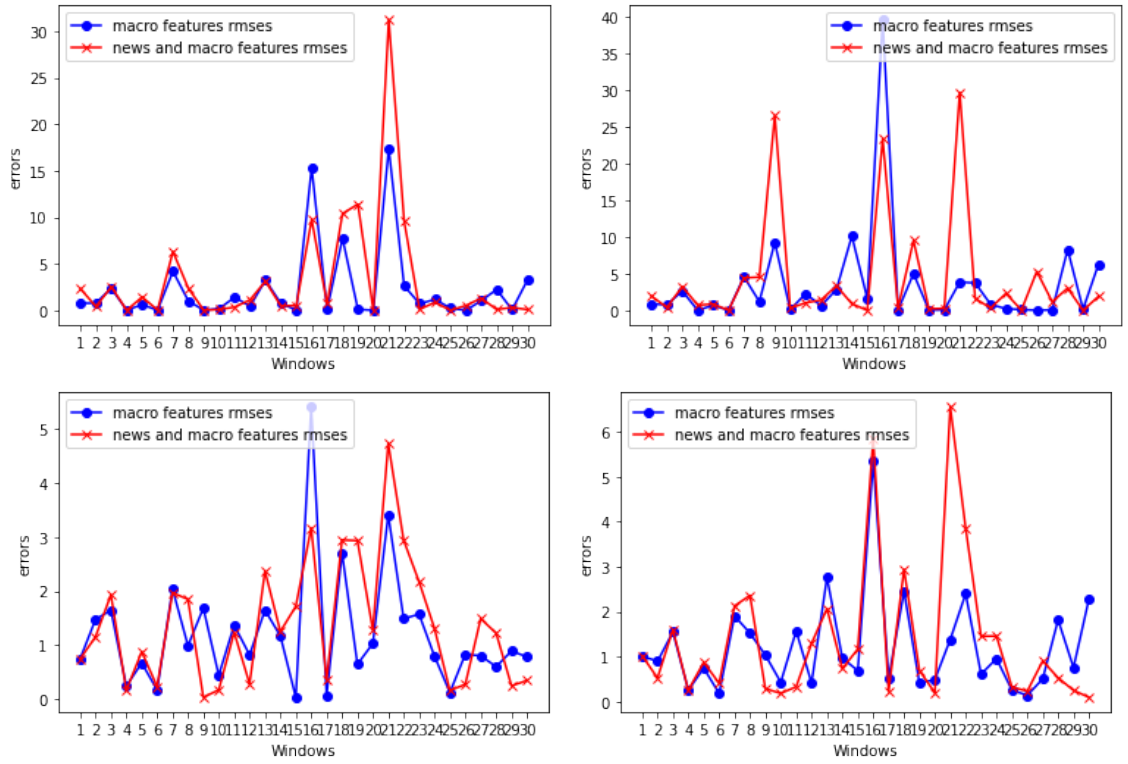

Figure 5: Absolute errors in each window for Ridge (top left), SVR (top right), random forest (bottom left), and neural network (bottom right) to predict Apple stocks

5.1.2 Microsoft performance

We also look at the performance of the models on Microsoft stocks, presented in table 4 and table 5. We have a pattern similar to the Apple stocks. We find that adding news data is detrimental to the models for the same reasons as explained in the previous section. However, we see that the errors for Microsoft stocks are slightly larger than Apple stock's predictions errors, but this is explained by the difference of scales between stock prices.

	Quantitative dataset	News dataset
Ridge regression	2.08	2.63
Support Vector Machine	2.20	2.83
Random Forest	2.15	2.56
Neural Network	2.06	2.56

Table 4: MAE of the one-day ahead prediction for Microsoft stocks

	Quantitative dataset	News dataset
Ridge regression	2.92	3.52
Support Vector Machine	2.95	3.80
Random Forest	2.82	3.37
Neural Network	2.90	3.45

Table 5: RMSE of the one-day ahead prediction for Microsoft stocks

The absolute error through the windows is presented in table 6. We see the errors of the models with the different datasets are similar except in some windows where the errors with the news datasets are larger. These errors explain the difference in the RMSE and the MAE between the two datasets in the tables above. We also observe higher peaks during the period of the Covid outbreak from the 12^{th} window to the 21^{th} window. During this period, there are higher peaks for all models and all datasets used. However, in the Microsoft case, it seems that the hypothesis made with Apple predictions does not stand for all cases, especially for the ridge regression or the SVR. However, we see the same phenomenon with the neural network and random forest. Some peaks with the non-textual data are alleviated by news data.

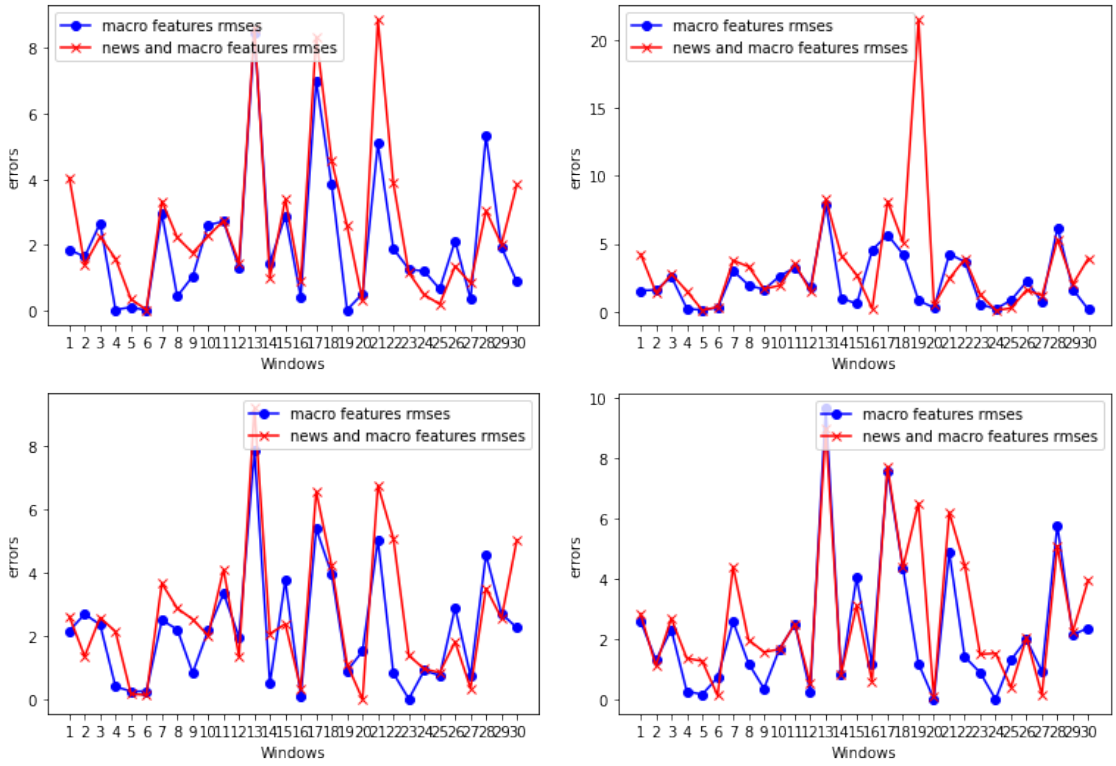

Figure 6: Absolute errors in each window for Ridge (top left), SVR (top right), random forest (bottom left) and neural network (bottom right) to predict Microsoft stocks

5.1.3 Tesla performance

We finally look at the Tesla stock's prediction. The MAE and the RMSE are presented in table 7 and table 6. The scale of the RMSE and the MAE are much larger than previous companies: this is due to the Tesla stock being much more volatile than the two other companies and therefore having a larger scale than other companies when we take the price change. There is an improvement in using news data when using random forest. The model using news data has a lower MAE using random forest, but its RMSE is larger. Therefore, the predictions are on average better, but they are also more volatile. For the other models, the use of text data seems detrimental to the predictions.

	Quantitative dataset	News dataset
Ridge regression	6	6.74
Support Vector Machine	5.69	6.84
Random Forest	6.09	6
Neural Network	6.34	6.61

Table 6: MAE of the one-day ahead prediction for Microsoft stocks

	Quantitative dataset	News dataset
Ridge regression	10.85	12.48
Support Vector Machine	10.49	12.90
Random Forest	10.76	11
Neural Network	11.79	11.98

Table 7: RMSE of the one-day ahead prediction for Microsoft stocks

We then look at the absolute error through the windows. In the Tesla case, news data do not add much information. During the Covid outbreak period, the peaks of the error are larger than the peaks of the other companies due to the volatility of Tesla stock during the Covid period. This volatility makes it very hard for models to predict stock prices. News data do not alleviate those peaks, the curve of the absolute error is very similar for both datasets. In this case, we cannot find any improvement due to news data. However, it seems we need a more complex model to predict Tesla stocks because all the models considered have poor performances due to the nature of Tesla stocks.

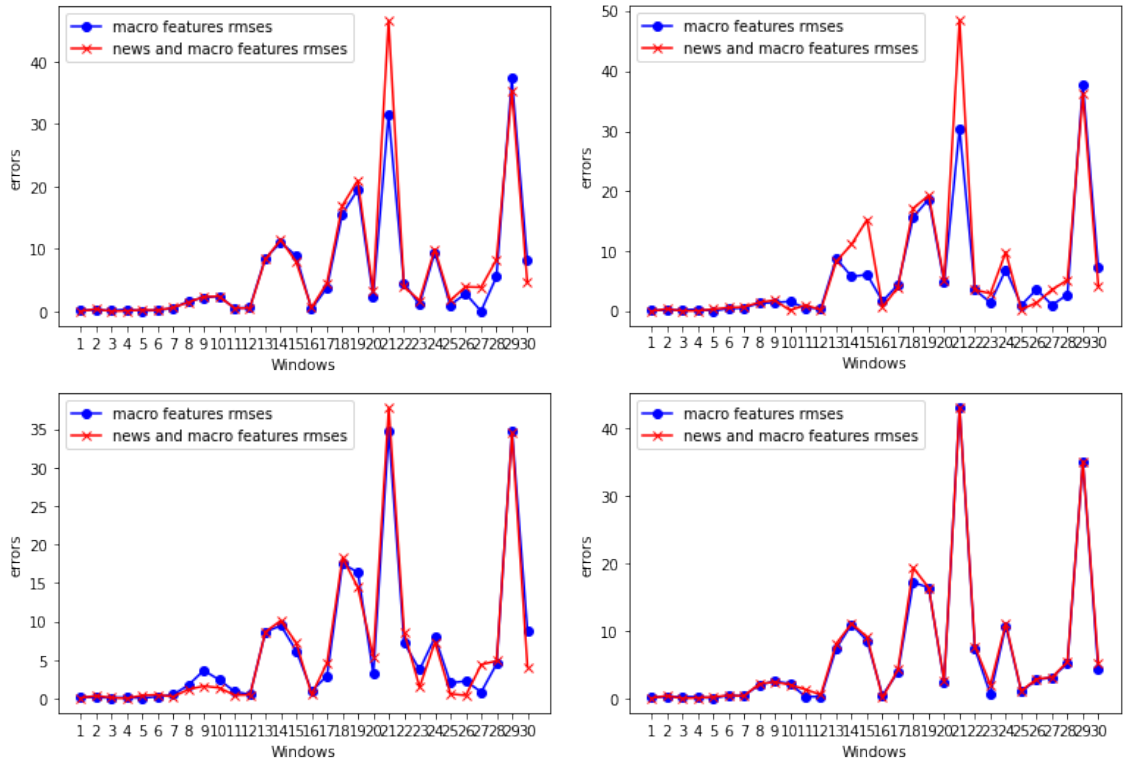

Figure 7: Absolute errors in each window for Ridge (top left), SVR (top right), random forest (bottom left) and neural network (bottom right) to predict Tesla stocks

5.2 Significance of the difference

We use the tests explained above to test whether our predictions are significantly different. We represent the p-value of the Diebold-Mariano test in table 8. The p-values show a significant result when they are lower than the level of significance. We fix the level of significance at 0.05. We find that the results are significantly different using news data for Microsoft stocks using random forest. The use of random forest to predict Microsoft stocks led to an improvement when not using news data. Additionally, this improvement is significant. For the other models, the difference observed when adding news data is not significant

	Apple	Microsoft	Tesla
Ridge	0.13	0.061	0.32
SVR	0.83	0.23	0.23
Random Forest	0.39	0.021	0.52
Neural Network	0.45	0.096	0.075

Table 8: P-values of the Diebold-Mariano test for each model and each company

We also look at the p-values of the model confidence set based on a SPA test in table 9. For each company, have the p-value of each model. A higher p-value means a better model compared to the others from the set. We find that the best models for Apple and Microsoft are random forest and ridge regression trained with text data. However, for Tesla, the best models are the ones trained without text data. The models trained with text data are among the worst for Tesla. For Apple and Tesla, random forest and ridge regression receive an upgrade when they are trained with text data. However, training neural networks and SVR with text data is detrimental to these models, as they may not be able to use the information from it. Another reason could be that these models need more computational power than we have to be trained using text data. Adding variables to the dataset with news articles constrains models to fit more parameters, and thus models such as neural networks need much more power in terms of computations to find the optimal parameters.

In the end, news data bring an improvement to models trying to predict some stock prices. Tesla stocks are very volatile and, therefore hard to predict, and news data does not help. However, for stocks with no big movements from one day to another, news data brings

an improvement.

	Apple		Microsoft		Tesla	
	Text data	Non-text data	text data	Non-text data	text data	Non-text data
SVR	0.069	0	0	0.005	0.01	0.343
RF	0.765	0.021	0	0.42	0.01	0.004
NN	0	0.206	0	0.01	0	1
Ridge	1	0	1	0	0.004	0

Table 9: p-values of the model confidence set based on SPA test

6 Conclusion

This paper focuses on three mega-cap companies and tries to predict their one-day-ahead stock prices using news data. In the beginning, we find that training on news data does not bring much improvement than training it on a dataset with quantitative features. Nevertheless, we have some slight improvements for some models: news data enables us to reduce some high errors when predicting some stock prices. This stands for big cap companies with not many fluctuations in their stock returns, but in case of very volatile returns, news data bring a very slight improvement that is negligible. Finally, we performed tests to assess the significance of this difference to evaluate if news data brings an improvement. The results of the statistical tests show an improvement in models when they are trained with news data. Models trained with news data are among the best models in our set of models for two out of the three companies. Nonetheless, some models seem unchanged or have less predictive power.

However, this paper presents some limitations. Firstly, the paper focuses only on mega-cap stocks from three of the biggest companies in the world, while it would have been relevant to see the impact of news data in predicting middle or small-cap companies' stocks. This was impossible due to the difficulty of finding news data every day for a company, and that is why we chose some massive companies. Moreover, the quality of news data was not the best possible, since it was retrieved from the web by scrapping news articles. The database had some missing values and some text unrelated to the articles. Moreover, the collection of these articles was very long. The study could give different results with a dataset retrieved from the Dow Jones database API, since the articles are already cleaned, the collection is straightforward, and we could collect more articles. The Dow Jones database API could have also give access to labeled data and consequently grant us sentiment of news articles. It could be relevant to use other more modern methods to handle text data, like the ones used in large language models such as GPT or Llama. However, these models do not provide free resources for sentiment analysis. Furthermore, these models could have been used to handle missing text data and generate new data based on the other news articles. Yet, models could be more trained and have better hyperparameter optimizations by using more powerful machines. Having more powerful machines would enable to find optimal hyperparameters for our models and

to use more observations to train our models. With better training on our models, the difference between models trained with and without news data could be clearer. Finally, it could be relevant to predict with different horizons to see if information brought by news data is pertinent for short-term or long-term predictions.

References

Andersen, T. G., Bollerslev, T., Diebold, F. X., and Vega, C. (2007). Real-time price discovery in global stock, bond and foreign exchange markets. *Journal of international Economics*, 73(2):251–277.

Ash, E. and Hansen, S. (2023). Text algorithms in economics. *Annual Review of Economics*, 15:659–688.

Bergmeir, C. and Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213.

Bergmeir, C., Hyndman, R. J., and Koo, B. (2018). A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Breiman, L. (2017). *Classification and regression trees*. Routledge.

Chandra, R. V. and Varanasi, B. S. (2015). *Python requests essentials*. Packt Publishing Ltd.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in neural information processing systems*, 27.

Den Heijer, M. (2024). pynytimes. Available at https://github.com/michadenheijer/pynytimes.

Diebold, F. X. and Mariano, R. S. (2002). Comparing predictive accuracy. *Journal of Business & economic statistics*, 20(1):134–144.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A., and Vapnik, V. (1996). Support vector regression machines. *Advances in neural information processing systems*, 9.

Fama, E. F. (1965). The behavior of stock-market prices. *The journal of Business*, 38(1):34–105.

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417.

Fama, E. F. and French, K. R. (1996). Multifactor explanations of asset pricing anomalies. *The Journal of Finance*, 51(1):55–84.

Fan, R.-E., Chen, P.-H., Lin, C.-J., and Joachims, T. (2005). Working set selection using second order information for training support vector machines. *Journal of machine learning research*, 6(12).

Gandhmal, D. P. and Kumar, K. (2019). Systematic analysis and review of stock market prediction techniques. *Computer Science Review*, 34:100190.

Gardner, B., Scotti, C., and Vega, C. (2022). Words speak as loudly as actions: Central bank communication and the response of equity prices to macroeconomic announcements. *Journal of Econometrics*, 231(2):387–409.

Gentzkow, M., Kelly, B., and Taddy, M. (2019). Text as data. *Journal of Economic Literature*, 57(3):535–574.

Hagenau, M., Liebmann, M., and Neumann, D. (2013). Automated news reading: Stock price prediction based on financial news using context-capturing features. *Decision support systems*, 55(3):685–697.

Hansen, P. R. (2005). A test for superior predictive ability. *Journal of business & economic statistics*, pages 365–380.

Hansen, P. R., Lunde, A., and Nason, J. M. (2011). The model confidence set. *Econometrica*, 79(2):453–497.

Hart, J. D. (1994). Automated kernel smoothing of dependent data by using time series cross-validation. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 56(3):529–542.

Heaton, J. (2008). *Introduction to neural networks with Java*. Heaton Research, Inc.

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

Hseb, A. (2017). google-search. Available at https://github.com/anthonyhseb/googlesearch.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Jan, M. N. and Ayub, U. (2019). Do the fama and french five-factor model forecast well using ann? *Journal of Business Economics and Management*, 20(1):168–191.

Jang, B., Kim, I., and Kim, J. W. (2019). Word2vec convolutional neural networks for classification of news articles and tweets. *PloS one*, 14(8):e0220976.

Kalamara, E., Turrell, A., Redl, C., Kapetanios, G., and Kapadia, S. (2022). Making text count: economic forecasting using newspaper text. *Journal of Applied Econometrics*, 37(5):896–919.

Kalra, S. and Prasad, J. S. (2019). Efficacy of news sentiment for stock market prediction. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 491–496. IEEE.

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.

Major, V., Surkis, A., and Aphinyanaphongs, Y. (2018). Utility of general and specific word embeddings for classifying translational stages of research. In *AMIA Annual Symposium Proceedings*, volume 2018, page 1405. American Medical Informatics Association.

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

Nti, K. O., Adekoya, A., and Weyori, B. (2019). Random forest based feature selection of macroeconomic variables for stock market prediction. *American Journal of Applied Sciences*, 16(7):200–212.

Řehřek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora.

Reuters, T. (2022). Annual economic indicators 2022. Eikon Datastream.

Richardson, L. (2007). Beautiful soup package on python. Available at https://github.com/wention/BeautifulSoup4.

Shah, D., Isah, H., and Zulkernine, F. (2019). Stock market analysis: A review and taxonomy of prediction techniques. *International Journal of Financial Studies*, 7(2):26.

Srivastava, A. (2010). Relevance of macro economic factors for the indian stock market. *Decision*, 37(3):69.

Sun, A., Lachanski, M., and Fabozzi, F. J. (2016). Trade the tweet: Social media text mining and sparse matrix factorization for stock market prediction. *International Review of Financial Analysis*, 48:272–281.

Szczygielski, J. J., Charteris, A., Bwanya, P. R., and Brzeszczyński, J. (2023). Which covid-19 information really impacts stock markets? *Journal of International Financial Markets, Institutions and Money*, 84:101592.

Verma, R. K. and Bansal, R. (2021). Impact of macroeconomic variables on the performance of stock exchange: a systematic review. *International Journal of Emerging Markets*, 16(7):1291–1329.

Wu, S., Liu, Y., Zou, Z., and Weng, T.-H. (2022). S_i_lstm: stock price prediction based on multiple data sources and sentiment analysis. *Connection Science*, 34(1):44–62.

Zhu, W., Zhang, W., Li, G.-Z., He, C., and Zhang, L. (2016). A study of damp-heat syndrome classification using word2vec and tf-idf. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1415–1420.