Università degli Studi di Milano-Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Data Science

BERT for Forecasting: Stock Price Prediction with Daily News

Relatore: Gabriella Pasi

Co-relatore: Alessandro Raganato

Tesi di Laurea Magistrale di:

Davide Miori

Matricola: 813692

Anno Accademico 2021/2022

Abstract

The recent growth of inflation pushed many people to invest their money rather than save. Among many, one of the preferred investment areas is the stock market since it is becoming easier and easier to create a personal account and trade by yourself. In contrast, nobody can exactly predict stock price behavior since it is influenced by many factors. Several technical indicators are usually used to understand price past movements and to have a hint of future actions. These indicators are statistical and financial indexes devoted to describing market conditions. An alternative to this approach is using the events that occur in the world since socioeconomic and financial episodes could impact related stock prices. However, even by combining these two strategies, predicting accurately the price future trend or worse, the price future value could be challenging.

This thesis aimed to reduce the error in the second of the above-cited task, i.e., the prediction of the stock price with a 7-day, 30-day, and 120-day time horizons. Several models from different areas of *Data Science* were trained to forecast future price values. To enhance the prediction, we integrated daily topical news as an additional feature. We proposed a new solution integrating text through a *Language Model*, *BERT (Bidirectional Encoder Representations from Transformers)*. Thanks to structural modification and feature engineering, we adapted the *Language Model* to a forecasting problem and enabled it to process numerical data. The proposed model outperformed the traditional ones, resulting in a significant reduction in *Root Mean Squared Error (RMSE)*.

Acknowledgments

I want to spend some words thanking all the people that had a role in my journey.
Firstly, I would like to thank Professor Gabriella Pasi and Alessandro Raganato for helping me during this project. Discussions, where we shared opinions about this thesis, were truly helpful and enjoyable.
I would also like to thank my whole family. First, my parents supported all my choices and made this possible so I'm genuinely grateful for what they did for me. My grandparents were always interested in my studies and in the same way as my brother were always close to me.
I especially would like to thank my girlfriend who has always been by my side during moments of difficulty and joy. She always encouraged me to give my best.
I also thank my fellow students with whom I shared every adventure and with whom I grow up culturally.
Finally, I would like to dedicate this journey to my grandmother. She has been my greatest and truest inspiration along the way.

Contents

List of Figures

Introduction

Time Series Analysis is growing in importance due to the massive production of such kind of data in several areas. Sequential data are spread in every field and its collection over time is a common behavior since it offers the possibility to study the evolution of a phenomenon according to time. Sometimes the trend of a series could be predicted just by analyzing the series itself, sometimes it is necessary to take advantage of external factors to improve the prediction. These factors could be directly related to the series, or they could have a relation with the topic of the analysis.

In recent years, text analysis grew exponentially thanks to the enormous production of text on the web allowing advanced techniques in the *Natural Language Processing (NLP)* field to process and extract value from textual documents. These discoveries brought additional possibilities to many areas such as *Time Series Analysis*. Social Media are characterized by millions of posts every day about every tendency or personal event. Newspapers and news agencies publish articles on their website keeping users updated. These reports are published daily so they can be used to document an event over time. *Time Series Analysis* and *Natural Language Processing* can be used in conjunction to explore, describe, and predict an event characterized by a sequential behavior.

Historians and politicians should be able to understand and predict events before they happen to prevent them or handle them in the best way possible. Since history repeats itself, they can consult recent events linked to the subject they are analyzing. The same concept will be considered in this thesis: if the model can learn specific relations between the news and numerical series behavior, it can use this information for a future situation that recalls the current one.

So, we chose to predict the Natural Gas Stock Price on different time horizons by studying the news related to the topic in association with the historical series of the price. Based on what was said above, socioeconomic and political events should help the prediction by giving additional information about the price movement. We chose Natural Gas Stock Price because of the recent events: the Russian invasion of Ukraine occurred on February

24th, 2022, and led to serious consequences all over Europe. European Union is committed to providing humanitarian aid and various military support to Ukrainians, focusing its effort on this front. Russia on the other side didn't expect this reaction since they programmed a lightning war. The war itself, and its prolongation caused also socioeconomic issues. The main of which is the gas price increment since Russia is the biggest exporter. Natural gas prices reached some weeks after the invasion the highest prices in many years impacting the economy of many European countries.

This thesis aimed to train an advanced *Language Model* that could use daily news to reinforce stock price prediction. The State-of-the-art solutions related to this topic are distinguished by two main approaches: *Financial Sentiment Analysis* and *Forecasting*. The first uses classification models, usually pre-trained, to predict whether the price will grow or decrease. It consists of extracting a sentiment output from a document represented by an article or a social media post. On the other hand, the forecasting methodology aims to predict the exact value that the price will have at a specific time in the future. It makes use of regression models often adapted to be forecasting ones. This thesis will focus on the second approach trying to predict the natural gas stock price with a time horizon of 7, 30, and 120 days.

We defined two baseline models to have a benchmark with which to compare. The first is probably the most used model in the forecasting field: *ARIMA*. In contrast with this statistical choice, the second model, a *Random Forest Regressor*, was taken from the machine learning field. These two alternatives allow us to make a solid comparison with the proposed solution since the two baseline models approach the problem in two different ways. *ARIMA* model was born to accomplish specifically a forecasting task, so it receives data in its native sequential order. *Random Forest* requires several preprocessing steps to be used. It was necessary to shift the target variable based on the prediction temporal horizon to match the feature value. In addition, this last model could handle efficiently text features after the extraction from the original text.

After that, we will present the proposed model based on *BERT* architecture. *BERT* is a *Language Model* belonging to the *Natural Language Processing* field, so it was developed with a different scope than our purpose. The real challenge was to make it capable of

predicting time series values. Besides *BERT*, we experimented with various alternatives the creation of which started from the BERT architecture to maximize the model's effectiveness and take advantage of their properties. We defined a cross-validation technique that suits the sequential nature of the data allowing us to train properly the models. This method named *Rolling Cross-Validation* is characterized by the usual procedure followed when dealing with the fold divisions of the dataset while keeping the time order. This choice allows the direct comparison with the *ARIMA* model which can only deal with ordered data. The evaluation process was characterized by the usage of 2022 as a test set to verify if the model could catch strong price movement due to the current European issues. The thesis structure is organized as follows:

- **Chapter 1** is dedicated to the discussion of the time series sector explaining the power of forecasting and focusing on the most used model in this field. Moreover, we will present an extension of the topic to the machine learning field exploring a valid solution to fit a time series problem to those models. In the end, we will describe a method to effectively train a model with sequential data.

- **Chapter 2** focuses on *Text Representation*. After an introduction of the subject, we provided an explanation of an advanced *Natural Language Processing* model which is the heart of this thesis: *BERT*. We will break down its architecture and main features to provide a comprehensive view of the subject.

- **Chapter 3** presents past works related to stock price prediction focusing on two main approaches. We will explain them by providing literature works that reached state-of-the-art results.

- **Chapter 4** is designed to unveil the practical effort behind this thesis by listing and explaining every step followed during the study. We chose different approaches to deal with this problem. Firstly, we have trained two baseline models coming from two different sectors: *Statistics* and *Machine Learning*. Then, various alternatives of *BERT* models were tested to extract values from the news. After the model selection that happened in the validation phase, the best model was tested on the 2022 set.

We used *RMSE* and graph visualizations to compare models. We also proposed a reliable solution for future works that could improve the results obtained.

1. Time Series Forecasting

In this chapter, we introduce *Time Series Forecasting* explaining the background and the areas where it is used. Then, we will focus our attention on the main methods and the most used evaluation strategies.

Forecasting is a technique that uses historical sequential data to make predictions as accurate as possible. It is a valuable strategy to obtain decision-making information in many various fields. As an example, it could be useful to predict the weather which is a historical sequence of events. Knowing when it will rain could be crucial for the management of reservoirs or in areas such as agriculture.

Forecasting is a strong and valid technique for different tasks. However, it is not always possible to predict accurately an event. If we know the factors that contribute to an event and we have enough data, it will be easier to forecast the future. On the other hand, when we don't have deep knowledge of a problem or its becoming, the prediction phase could be tough or have the same results as a coin toss.

Forecasting is usually associated with Time Series which are sequential numerical data listed in time order where the intervals between values are equally long. Some examples of time series are the annual profit of a company, the monthly waterfall, or the daily price in the stock market. If the disposal of historical data is consistent, anyone can analyze time series in every field.

It is possible to forecast each observation in a time series using previous observations. The predictions are called *fitted values* and often they are represented with the symbol $\hat{y}_t$. By subtracting the *fitted values* from the actual ones, it is possible to obtain the *residuals* that are expressed as follows:

$$e_t = y_t - \hat{y}_t. \tag{1.1}$$

Residuals are a great indicator of the model's goodness and prediction accuracy. They monitor if the model can capture information from data. It is necessary that the *residuals* are uncorrelated, and they have zero mean. If these properties are not satisfied, then the model could be improved. The correlation implies that some information is left in the

residuals while zero-mean residuals signal a biased forecast [9].
Forecasting aims to predict the value of the sequence based on the time interval. Considering the monthly amount of rain, the goal will be to predict the amount of the next month or the next five months. The forecasting horizon is something to be considered too. Will the prediction be necessary for one unit of time, three units of time, or more? As the forecast horizon increases, the prediction accuracy decreases.

Various techniques could be applied to this type of task. It is possible to identify two main categories: statistical methods and machine learning methods. The former comprehends time series decomposition, exponential smoothing, and *ARIMA* models. The latter includes both classical machine learning models and neural networks. As we will discuss later, machine learning methods need a specified pre-processing phase to transform data in the appropriate format.

1.1 Arima Models

ARIMA models are the widely used approach for time series analysis and forecasting. The acronym stands for *Autoregressive Integrated Moving Average* model which is a generalization of the *Autoregressive Moving Average* model called *ARMA*. The former was developed in the 70s by George Box and Gwilym Jenkins to reduce as close to zero as possible the difference between the values predicted by the model and the observed ones. *ARIMA* models can describe both stationary and non-stationary behaviors thanks to the Integrated component.
Before getting into the details, it is necessary to explain what stationarity is. When predicting a time series' future behavior, it is important to assume that each observation is independent of the others. Since the series is expressed in temporal terms, what is necessary to be verified is that the statistical properties do not change over time. So, a series is stationary when doesn't present a trend, seasonality, or every behavior dependent on time [9].
Figure 1.1 shows an example of non-stationary time series of monthly airline passengers. The increasing trend over time and the seasonal fluctuations are clear symptoms of not

being stationary.

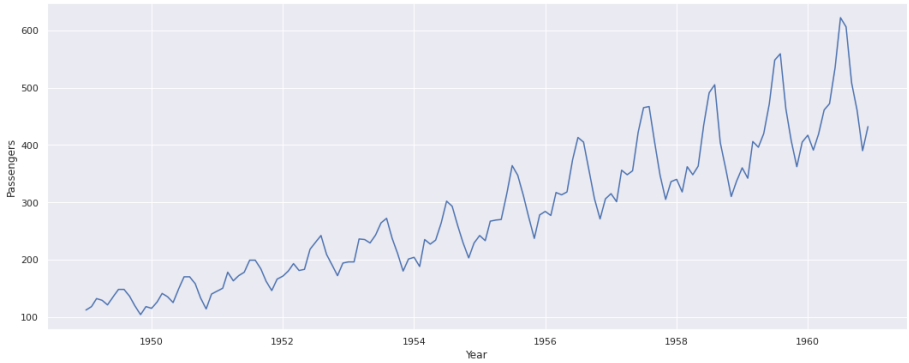

Figure 1.1: Non-stationary series

Stationarity could be related to the mean or the variance of a series. In the first case, if the mean doesn't change concerning time, then the series is stationary related to the mean. The second one applies the same concept to the variance. Figure 1.1 is an example of mean and variance non-stationarity. In fact, both increase over time.
To solve the non-stationary issue, it is possible to apply different techniques. *Differencing* is one of them related to the mean. It consists in computing the difference between consecutive values. It eliminates a trend and solves seasonality whose behavior depends on the season or time of the year. Mathematically, it is expressed as follows:

$$y'_t = y_t - y_{t-1}. \tag{1.2}$$

Instead, if the problem is connected to the variance of the series, a proper solution could be a transformation such as the logarithmic. Usually, the major reference is the *Box-Cox Transformation* which is expressed as follows:

$$\begin{cases} \frac{y^{\lambda}-1}{\lambda}, & \text{if } \lambda \neq 0 \\ logy, & \text{if } \lambda = 0 \end{cases} \tag{1.3}$$

where the value of λ determines the type of transformation to apply.

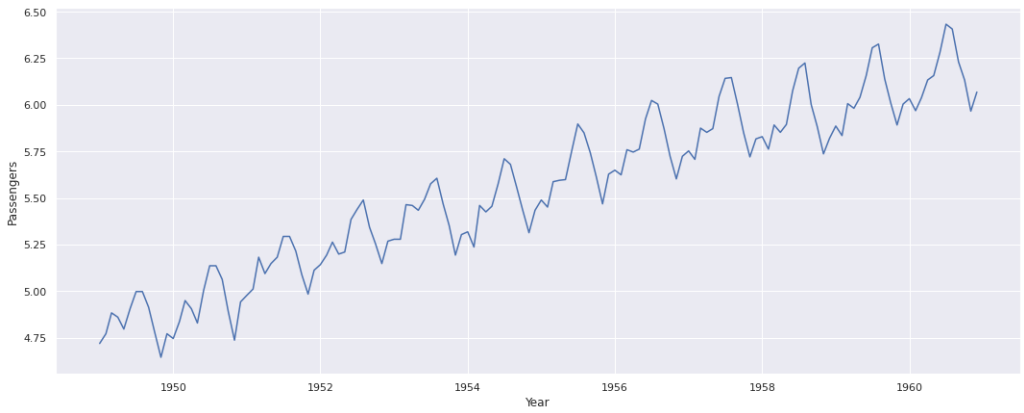

Figure 1.2: Log-transformed series

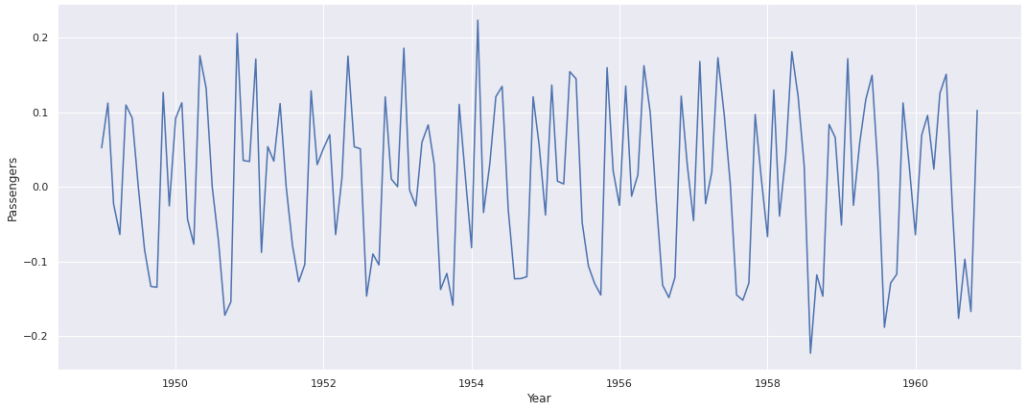

Figure 1.3: Log-trasformed and differenced series

Figure 1.2 displays the airline passenger series after the application of a logarithmic transformation. It deleted the variance over time. Figure 1.3 manifests the same series after a one-lag difference that eliminates the trend. Therefore, the goal of removing non-stationarity is to keep mean and variance constant over time.

To identify non-stationarity, a valid alternative to the simple plot of the series is the *Autocorrelation Function* or *ACF*. Autocorrelation measures the linear relationship between lagged values over time. It is usually expressed as follows:

$$r_k = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2} \tag{1.4}$$

where T is the length of the time series and k determines how many lags to consider i.e., r_2 measures the correlation between y_t and $y_t - 2$.

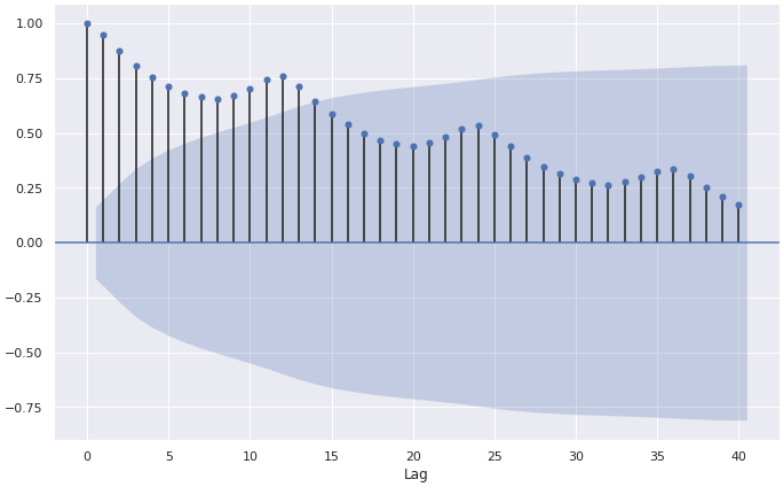

Figure 1.4: ACF plot

Figure 1.4 shows an example of an *ACF plot* where the blue area represents the 95% confidence interval which is an indicator of significance. When the sticks come out of the area, the value of correlation is statistically non-zero. So, there is a correlation between consecutive values. The first stick is always set to one because it represents the correlation between the first value and itself.

ARIMA models are formed by two components: *Autoregressive (AR)* and *Moving Average (MA)*. An *Autoregressive* model exploits past values through linear combinations [9]. The written expression of the *AR* model is

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t \tag{1.5}$$

where ε is the white noise, a time series that has independent values with a mean of zero, and ϕ are the parameters that influence the series' patterns. It is like a multiple regression with lagged values as predictors. The model written above is called *AR(p)* where p represents the order. *Moving Average* models, rather than using the past values, uses past forecast errors with the same form of regression as the formula shows:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q}. \tag{1.6}$$

In this case, θ are the parameters and ε is still the white noise. We associate that model with the expression *MA(q)*.

Combining these two models (*AR* and *MA*) we obtain the *ARMA* model which could become an *ARIMA* if differencing is computed. The letter "I" stands for *Integrated* which represents how many differences have been applied to the series. An *ARIMA* model has the following form:

$$y'_t = c + c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + ... + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q} + \varepsilon_t \quad (1.7)$$

where p is the order of the autoregressive part, q is the order of the moving average part, and d is the number of differences applied. The value of d can be determined by studying the line plot or the *ACF plot*. On the other hand, the order p and q could be identified through the combination of the *ACF plot* and the *PACF plot* which is the partial autocorrelation function. The latter graph explains the partial correlation between the series' values computing the linear relationship between observations separated by k time units that the previous lag doesn't explain. For example, the partial autocorrelation for lag 3 is the correlation that lags 1 and 2 do not explain. It is possible to deduce the value of p and q based on the form that the two graphs assume.

ARIMA models can handle non-stationary data, thanks to their integrated part. In addition, they can handle seasonal data by introducing seasonal terms that find their counterparts in p,q, and d. We can now express the model as follows:

$$\text{ARIMA } (p,d,q)(P,D,Q)_m \quad (1.8)$$

where m is the number of observations per year. If the series presents a monthly seasonality, the value of m will be 12.

1.2 Machine Learning Models

Machine Learning has been used with great success in many fields lately. *Machine Learning* is a sub-field of *Artificial Intelligence and Computer Science* that is characterized by the usage of data and algorithms. These algorithms or models "learn" from data identifying patterns which are a recurrent form of knowledge on that event. The learning process is called *Training Phase* and it is followed by the *Test Phase*. During the first phase, a

sample of the original data, called *training set*, is passed to the model so it could learn from it. Based on the type of algorithms the learning process can vary. Then, the second phase is a performance test that is applied to a different sample of data called *test set*. So, before the model is initialized, it is necessary to split the original dataset into at least two portions. We will show in the next paragraph different alternatives.

Usually, a tabular dataset is formed by two different types of variables, the target variable (or dependent variable), and the features (or independent variables). The former is the objective of the analysis, what we want to understand, comprehend, or predict. Features are instead variables that bring, hopefully, information about the target. The model will study the relationship between features and the target variable. The *training set* is composed of a sample of observations from the dataset keeping both the features and the target. The *test set*, instead, is composed only of the independent variables in such a way that it could predict the target without knowing the original value. Briefly, the model learns from the *training set*, and it is examined on the *test* set which is composed of unseen observations. The target values associated with the test observations are used to compute the evaluation metrics which usually compare the predicted values with the actual or real ones. After the training phase, the model produces the error computed on the *training set* which is not the proper way to evaluate its performance. The *training error* is biased because the model has seen those observations and the prediction on them should be good. The appropriate procedure consists in computing the error on the *test set*, the so-called *Generalization error*, which represents the ability of the model to generalize (to predict correctly new observations). When the test error is way larger than the training one the model is called an *overfitted* model. *Overfitting* occurs when the model can't generalize. An event akin to *overfitting* is *underfitting*. In this situation, the algorithm fails to predict the value of the target related to the training observations and obtains a poor performance also in the *test set* [1].

Machine learning approaches are divided into four categories: *supervised learning*, *unsupervised learning*, *semi-supervised learning*, and *reinforcement learning* [1].

- *Supervised Learning*: it is defined by a labeled dataset which means that each observation has a value for the target variables. The goal of those kinds of tasks is to define

a model which can predict the target value of an unseen observation. *Classification* and *Regression* are both two supervised methodologies. The first is characterized by a categorical target variable i.e., it is formed by classes. The second presents a numerical target variable.

- *Unsupervised Learning*: it is a sub-field of *Machine Learning* which is defined by the absence of labels. It consists in discovering hidden patterns and variable relationships to group data points based on similarities. The most famous technique is called *Clustering* which aims to create clusters (groups) of observation that present similar characteristics.

- *Semi-Supervised Learning*: unlike the previous ones, it is a developing field. It is characterized by a dataset composed of some labeled observations and others not.

- *Reinforcement Learning*: it is a machine learning technique that consists in training a model based on a system of punishment and reward. Thanks to a bonus and malus structure the model learns from its actions and corrects its behavior.

These methods can be applied to different fields such as medicine, agriculture, computer vision, and speech recognition and data could be of various formats: tabular, time series, images, texts, videos, or audio files. Forecasting problems are on the list too. However, no eligible classical methods can treat sequential data directly. Classical machine learning models require data to be in tabular form so if we are working on a different kind of task, it is necessary to transform and convert data to the proper form. On the other hand, advanced machine learning techniques such as neural networks can work with data in sequential order.

1.3 Forecast Approach and Evaluation Methods

Machine learning models can be applied to forecast problems as anticipated in the previous paragraph. It is necessary though to adapt the dataset to the task because tabular data require an association between the covariate's values and the target values. Previously, we discussed that the forecasting horizon could be of different lengths based on the request of the task we are working on. It is possible to shift the dataset to associate the current observation with the future one we would like to predict.
For example, if the time horizon is of seven days, we need to associate with the current time unit the values of the target variable seven days ahead. By doing so, the model can learn what happens seven units ahead and will be able to produce a prediction at that unit of time. The forecasting problem has been converted to a supervised task.

Once the model is trained, it is necessary to evaluate the accuracy of its predictions. As discussed in the previous paragraph, the dataset is usually divided into two subsets: *training* and *test set*. This procedure can be followed also for a forecasting problem. The main difference is the sequential order that characterizes time series. Statistical methods such as *ARIMA* models require data to be ordered by time so the splitting could be performed by keeping the sequential information. The original series is divided into two sub-series at a certain point in time. The observations before that time will be part of the *training set* while the values after that time will be the test set. Machine learning strategies don't require that the order should be kept. However, if both statistical and machine learning methods are used, the timewise split will be performed.

Several measures could be used to evaluate the performance of a model [9]. The most used strategies are the *Mean Absolute Error (MAE)* expressed as

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}| \tag{1.9}$$

and the *Root Mean Squared Error (RMSE)* which has the following form:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2}. \tag{1.10}$$

The former represents the absolute value of the difference between the true values and the predicted ones. The latter is the squared root applied to the mean of the squared

difference between true values and predicted ones. They are both popular thanks to their interpretability and short computing time. Besides, they are scale-dependent, so their values are on the same unit of measurement as the time series values.

As an alternative to those metrics, the Mean *Absolute Percentage Error (MAPE)* is a unit-free measure that offers the result in a percentage form allowing comparison between different datasets. Mathematically, it is expressed as follows:

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} |\frac{y_i - \hat{y}}{y_i}|. \tag{1.11}$$

It could be biased when the values of the series are close to zero.

R-squared is another metric that could be computed to evaluate the goodness of a model. It measures the variance in the dependent variable that can be explained by the independent one. In other words, it measures the linear relationships between the features and the target variable. It takes values between zero and one where one indicated that all the target variance is explained by the model (the model fits the data the best it could). Its formulation is:

$$R^2 = 1 - \frac{var(r)}{var(y)}. \tag{1.12}$$

1.3.1 Cross-Validation

We have discussed in this chapter the train-test split to evaluate properly the model. Complex and better versions of this split exist and are normally referred to as *Cross-Validation*. The basic split (train-test split) consists of the performance evaluation on the *test set* which could be biased because the result will depend on the observations the test is composed of. We refer to this technique as the *Out-of-Sample approach (OOS)*. Cross-validation techniques consist of the creation of a new subset of observations: the *validation set*. This set is used to compare different types of models or models with different parameters to find the best solutions. To avoid the problem defined above, *Cross-Validation* offers the chance to create a loop of training and evaluation phases where the observations in the validation set vary at each iteration.

The most famous cross-validation technique is called *K-fold Cross-Validation* or *k-fold cv*. After the creation of the *test set*, the *training set* is split into k folds or groups of random

observations. At each iteration, the model has trained on $k-1$ folds, and it is evaluated on the group left over. Usually, these training-evaluation phases are equal to the number of folds. If the number of groups is equal to the number of observations in the training set, the cross-validation is called *Leave One Out CV* or *LOOCV*.

This advanced splitting strategy can be applied to time series tasks with some adjustments. Time order is a problem when *ARIMA* models are applied. The main concept remains the same while the way the folds are created changes. To maintain the time sequential order, the time series is firstly divided into three subsets: *training*, *validation*, and *test set*. The *validation set* is split into k groups timewise so the first group will contain observations consecutive to the training set ones. At this point, the model is trained on the *training set* and evaluated on the first fold. After that, this fold joins the *training set*, and the model will be trained again on this new set. The evaluation phase will be performed on the second timewise group of the *validation set*. This iteration strategy perdures until the last fold.

As for the *Cross-Validation* discussed before, it is possible to make the number of folds equal to the number of observations in the validation set. *Out-Of-Sample* procedure could also be used. When applying it, the series is divided into two blocks: an initial fit period in which the model is trained, and a testing period held out for estimating the generalization performance [3].

2. Text Representation

Text Representation refers to the process of converting textual data into a structured format that can be easily processed and analyzed by computer programs. This is often necessary in natural language processing (NLP) tasks, where algorithms need to work with text data in order to perform tasks such as sentiment analysis, language translation, or text classification. Text representation techniques involve converting the raw text data into a mathematical representation, such as a vector or a matrix, that captures the meaning and relationships between words or phrases, sometimes called feature vectors. For simplification, we briefly describe two common text representation methodologies:

- *Bag of Words Representation*: they enclose the methodologies that represent texts as term-document matrices. The simplest one is the *Bag of Words (BOW)* representation. The method groups the words in the document by their appearance. It simply counts how many times a term appears in a text. The text "Your dog is obedient. My dog is not." could be represented as {your: 1, my: 1, dog: 2, is: 2, obedient: 1, not: 1}. It is possible to improve the efficiency of this representation method by simply dividing the number of occurrences of a word by the total number of words in the document.

  Bag of Words is the simplest representation tool for text but sometimes it could be a valid solution because it is easy to compute and interpret. A natural extension of the *BOW* model is *Term Frequency – Inverse Document Frequency (TF-IDF)* [17]. This concept is based on *Luhn's Analysis* which states that quite rare words and very common ones don't bring discriminant power [10]. Common words are defined as stop words. Usually, articles, prepositions, pronouns, and conjunctions are qualified as stop words since their presence can't bring solid information to the analysis. However, sometimes these groups of words could be useful; for example, web search engines use stop words in their query system. Rare words have the same impact on the analysis since their rarity can't bring enough information to discriminate between documents. Derived from *Luhn's Analysis*, a term weighting system

was proposed. It provided two measures: *Term Frequency* and *Inverse Document Frequency*. The first expressed as $\text{tf}_{t,d}$ indicates the number of times a term t occurs in document d. Usually, it is normalized through a division by the maximum value of $\text{tf}_{t,d}$. The expression is the following:

$$w_{t,d} = \frac{\text{tf}_{t,d}}{\max_{t_i} \text{tf}_{t_i,d}} \tag{2.1}$$

The second has the succeeding expression

$$\text{idf}_t = \log(\frac{N}{\text{df}_t}) \tag{2.2}$$

where df_t is the document frequency of term t i.e., the number of documents that contain t, and N is the number of documents in the collection. The product between these two measures produces the tf-idf weights

$$w_{t,d} = (\frac{\text{tf}_{t,d}}{\max_{t_i} \text{tf}_{t_i,d}}) \cdot \log(\frac{N}{\text{df}_t}). \tag{2.3}$$

If a term is common in a document, it will have a high $\text{tf}_{t,d}$. At the same time, if it is rare in the collection, it will have a high idf_t. This term has great discriminant power.

- *Embedding Representations*: Embeddings have been one of the dominating concepts since the early 2010s for Natural Language Processing (NLP). Text embedding is a technique in *Natural Language Processing (NLP)* that transforms text into a numerical vector representation, which can then be used as input for various machine learning algorithms. The goal of text embedding is to capture the meaning of words and sentences in a way that is computationally useful [14]. Text embeddings have become an important tool in *NLP*, by representing text as numerical vectors, machine learning algorithms can easily compare and classify text, enabling a wide range of applications. Several approaches have been proposed that compute text embeddings, such as *Word2Vec* [15], *GloVe* [18], and *BERT* [20]. The last and most recent one, i.e. *BERT*, will be discussed in detail in the next section.

These strategies allow the creation of feature vectors. Each word or group of words will have a value or a sequence of values, based on the techniques used, used as input to address natural language processing tasks.

2.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) [5] is a neural language model developed by Google's researchers whose paper was published in May 2019. It is a recent and innovative solution in the *Natural Language Processing (NLP)* field, showing state-of-the-art results for a wide variety of *NLP* tasks, such as *Question Answering* and *Natural Language Inference*.

BERT is trained on a *Language Model* task. Language modeling objective is to predict the next token (given a sequence of tokens), and *BERT* makes use of a *Transformer* architecture [24] to compute a probability distribution over words.

Before the development of *BERT*, *Language Models* could read the text from either left-to-right or combining left-to-right and right-to-left. *BERT* introduced the bidirectional functionality that allows reading in both directions making it capable of catching better the context [5].

2.1.1 Architecture

BERT is defined as a multi-layer bidirectional transformer encoder that makes use of an attention mechanism to extract features from words [5]. A *Transformer* is based on an encoder-decoder framework: the encoder is a neural network that maps an input sequence $(x_1, \ldots, x_n)$ to a continuous representation $z = (z_1, \ldots, z_n)$ while the decoder generates an output sequence $y = (y_1, \ldots, y_n)$ one element at a time based on the mapped representation. The structure is depicted in Figure 2.1 where the encoder is represented on the left and the decoder on the right. *Transformers* use stacked multi-head attention and fully connected layers for both the encoder and decoder [24].

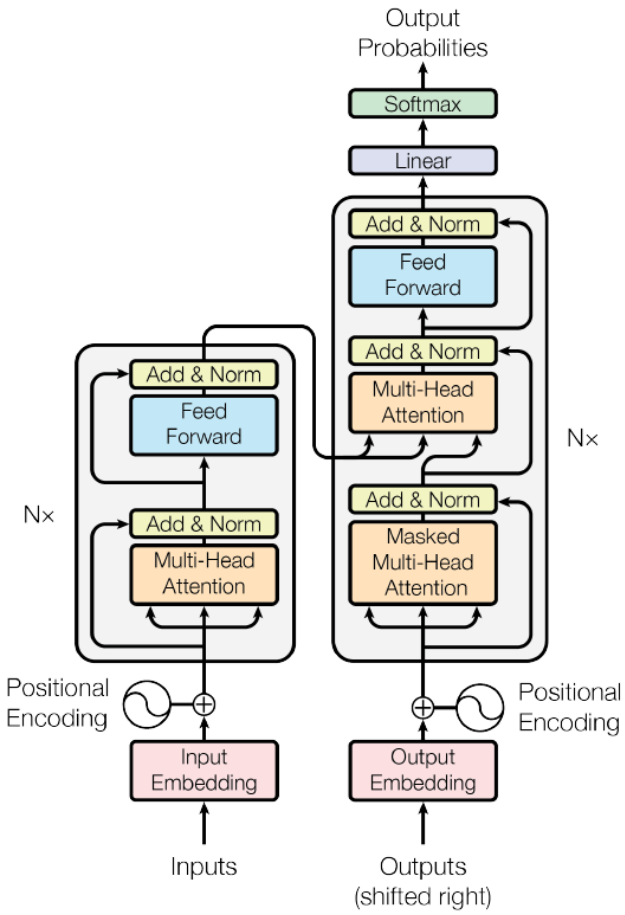

Figure 2.1: Transformer Architecture

Multi-Head Attention

To understand a multi-head attention layer, it is necessary to dive into self-attention. Self-attention is defined by Peter Bloem as a sequence-to-sequence operation in which a sequence of vectors goes in, and one comes out [1]. In the self-attention mechanism, every input vector is used in three different ways: the Query, the Key, and the Value. The output is computed as a weighted sum of the values where the weights related to each value are computed by a function of the query with the corresponding key. The name given by the researchers to this kind of attention mechanism is *Scaled Dot-Product Attention* and it is composed of queries and keys of dimension d_k and values of dimensions d_w. The weights of the values are computed as a dot product between the query and all the keys divided by

[1] Peter Bloem, "Transformers from scratch" blog post, 2019.

the square root of d_k and passed through a *softmax* function which transforms the vector number into a vector of probabilities [24]. Figure 2.2 displays the mechanism.

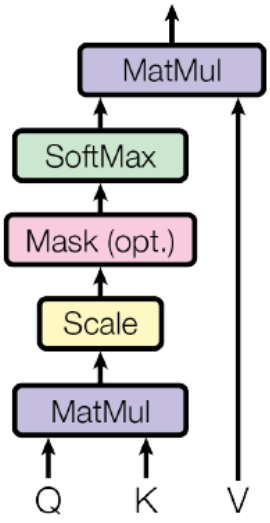

Figure 2.2: Scaled Dot Product Architecture

The final formula is:

$$Attention(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.4}$$

where the *softmax* function is expressed as

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}. \tag{2.5}$$

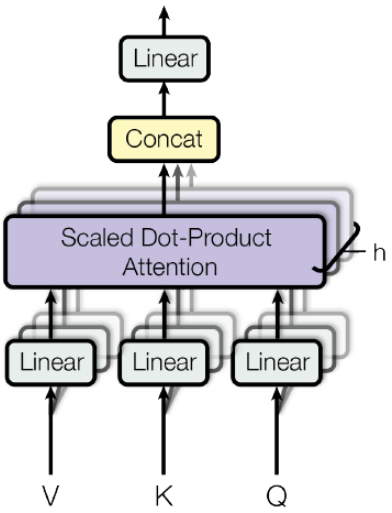

Figure 2.3: Multi-Head Attention Architecture

Although this procedure is effective, *Transformers* relies on a multi-head attention mechanism to take advantage of the ability to jointly attend multiple positions that allows the creation of richer representations. Figure 2.3 shows the multi-head attention structure.

Feed Forward Layers

In addition to multi-head attention layers, *Transformers* are composed of position-wise *Feed Forward networks* that are applied to each token in the same way. The layer consists of two linear transformations with a *ReLU* activation function in between. Activation functions are mathematical functions (usually non-linear) used in neural network models to transform the input allowing the model to learn complex patterns [19]. *ReLU* function is expressed as follows:

$$ReLU(z) = \max(0, z), \tag{2.6}$$

and it has the shape depicted in Figure 2.4.

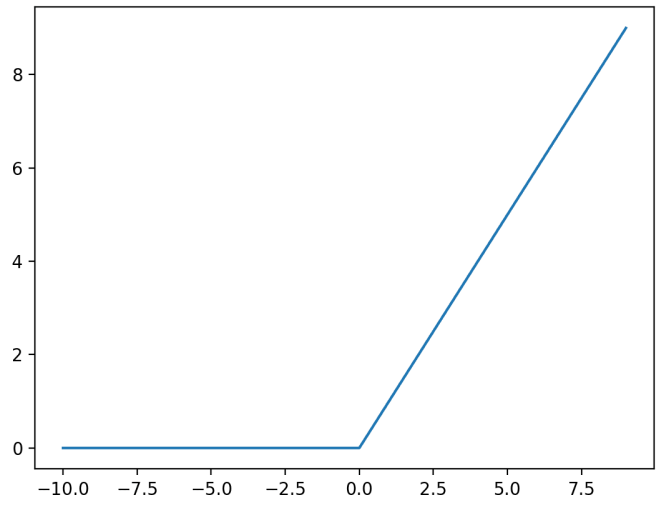

Figure 2.4: ReLU Function

Positional Encoding

Positional encoding layers are located after the input and output embeddings that are used to convert and adapt tokens to vectors of dimension d_{model}. This encoding layer allows the model to understand the order of the given sequence such as the relative and absolute position of tokens in the sentence. Both the encodings have the same dimension d_{model} as the embeddings [24]. The functions used for them are:

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right); \tag{2.7}$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right), \tag{2.8}$$

where *pos* is the position in the sentence and *i* is the dimension.

Other components

After each multi-head attention and fully connected layer, a normalization layer is located to standardize the data inside the same range. This is applied after a summing-up process. Besides, a residual connection is positioned around each sub-layer (multi-head attention

and fully connected layers). A residual connection is displayed in Figure 2.5 represented by the arrow. It could be seen as a skip connection since it allows the flow of the gradient to skip some non-linear functions. This method is used to avoid problems with gradient computations. All these additional layers are used to speed up the training process and obtain better accuracy.

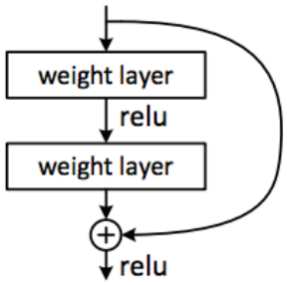

Figure 2.5: Residual Connection

Regularization is also applied in the form of a dropout layer. To be precise, dropout is used before the normalization phase and before both the embeddings and positional encodings.
Regularization techniques are used to avoid *overfitting* allowing the model to generalize better. Dropout is one of the most effective regularization methods that randomly mask some of the neurons of a neural network to strengthen the training process. At the very end of the Transformer's structure, a linear transformation is positioned to transform the stacked output into a larger vector of scores which is passed through a *softmax* function that turns the scores into probabilities.

BERT architecture

BERT was proposed in two versions: $BERT_{BASE}$ and $BERT_{LARGE}$. The former is composed of 12 Transformers, 768 Hidden layers, and 12 self-attention heads while the latter is formed of 24 Transformers, 1024 Hidden layers, and 16 self-attention heads.
To make *BERT* handle different tasks, the researchers make the input representation able to represent both a single sentence and a pair of sentences. This fact introduced the problem of distinguishing between the two sentences. To solve it, a *[CLS]* token is inserted at the

beginning of the first sentence and a *[SEP]* token at the end of each sentence. This phase is called *Token Embedding*. After that, a marker is introduced to identify sentence A tokens and sentence B tokens (*Segment Embedding*). Then, a positional embedding is added to each token to define its position in the sentence. Delvin J. et al. in the original paper of *BERT* [5] showed this process with the representation depicted in Figure 2.6.

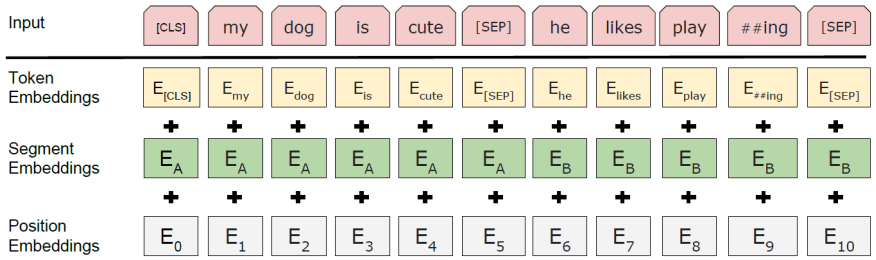

Figure 2.6: BERT Input Representation

2.1.2 Training

One of the main challenges in *NLP* is the lack of labeled training data which could seem confusing since millions of documents and texts are available online. However, if you are looking for a task-specific dataset it is quite difficult to split all the relevant sections from the enormous online collection. In addition, if you manage to do so, the result will be a dataset composed of relatively few rows. To solve this problem, experts developed various strategies to train a model on enormous unlabeled text calling it *pre-training*.
This process allows the so-called *fine-tuning* of task-oriented training to focus the model expression on the required area. Briefly, the model is pre-trained on an unannotated enormous dataset, and then it is fine-tuned on the specific task with a small dataset saving time and reducing the computational cost. *BERT* was developed as a pre-trained model to establish context by looking at the surrounding words.

Pre-Training

Unlike the traditional language models that can learn left-to-right or right-to-left, *BERT* was pre-trained using two unsupervised separated tasks: *Masked Language Model (MLM)* and *Next Sentence Prediction (NSP)*. The former randomly masks some of the input text

tokens to predict the masked word's original vocabulary id based on its context. The task consists of randomly masking out 15% of the words and replacing them with a specific token defined *Masked Token*. The sequence will be passed to *BERT* attention-based encoder which will predict the masked word. This procedure comes with a downgrade. During *fine-tuning*, the masked token won't appear creating a mismatch between the two steps of the framework. To mitigate it, the researchers adopted the following strategy. The masked token will be replaced 80% of the time with the masked token *[MASK]*, 10% of the time with the actual one, and 10% of the time with a random one.

To make an example, let's consider the sentence "the apple is red". The random masking strategy selects the word "red" which will be replaced with *[MASK]* 80% of the time obtaining the sentence "The apple is *[MASK]*". 10% of the time it will be replaced with a random word: "The apple is book". In the remaining time the word "red" would be kept.

Besides, *BERT* is also pre-trained on a *Next Sentence Prediction* task to give the model the comprehension of the logical sequential connection between sentences. The developers chose two input sentences at a time to feed the model. 50% of the time the second sentence comes after the first while in the remaining half the following sentence is replaced with a random one extracted from the corpus. In the first case, the following sentence is called *IsNext* while in the second it is called *NotNext*. *NSP* is a binary classification task in which the two sentences are fed to the model, and it must predict whether the second sentence is the subsequent one of the first sentence.

To pre-train the model, two massive datasets have been used. The first is *Wikipedia Corpus* which contains roughly 2.5 billion words; the second is *Toronto's BookCorpus* composed of approximately 800 million words. The procedure required a long time to be completed so the experts use *Tensor Processing Units (TPU)* to speed up the process. To be specific, they used 64 *TPUs* to train *BERT* for four days [5].

Fine-Tuning

The *fine-tuning* phase is inexpensive compared to the *pre-training* step. It consists of a model re-training on a specific task. This new training will have as initialized parameters the ones from the pre-training phase. Then, it will update these weights on a new dataset

oriented to the required task. This procedure is straightforward since it ensures fast training on new tasks and requires small datasets. The pre-training procedure followed by the fine-tuning one comes with the name of *Transfer Learning*.
Empirical studies demonstrated the effectiveness of unsupervised *pre-training* in an integral part of many language understanding systems allowing low-resource tasks to benefit from it. *BERT* outperformed various state-of-the-art tasks such as *Natural Language Interference*, *Sentiment Analysis*, *Question Answering*, and *Named Entity Recognition*. Adding a new layer on the top of the architecture it is possible to fine-tune *BERT* on different problems. Text Classification, for example, could be performed simply by adding a classification layer on top of the transformer output. The classification layer is composed of a *fully connected* layer and a *softmax* function.

2.1.3 Variants

We discussed in the previous paragraphs how the combination of *pre-training* and *fine-tuning* is straightforward since it brings value in terms of computational time and dataset size. This procedure also allows the training of larger models that obtain better performance as demonstrated in the literature. However, there are some concerns about so. Mainly, real-time operation on mobile devices needs to be fast and can't bear substantial computational effort or large memory occupation. Sanh V., Debut L., Chaumand J., and Wolf T. developed a much smaller language model that obtained roughly the same performance as *BERT*. To reach this goal, they adopted the knowledge distillation technique reducing both the computational time for training and inference.
Knowledge Distillation is a process of transferring knowledge from a larger teacher model to a smaller student model. The latter is trained to behave the same way as the former, aiming to obtain the same performance or increase accuracy. The parent or teacher model could be one or an ensemble of models. Using *BERT* as the teacher model the researchers obtained a new smaller version of it called *DistilBERT* [23]. This new model presents the same architecture as the main one with some adjustments. The token-type embedding was removed, and the number of layers is reduced by a factor of two. *DistilBERT* obtained 97% of *BERT* performance with 40% fewer parameters and faster inference time. Besides,

the training time was 4 times less than *BERT*'s [23].

In addition to *DistilBERT*, many variants of *BERT* exist and can be easily used as an alternative. *RoBERTa (Robustly Optimized BERT Pretraining Approach)* [12] is one of them trained for more time and on larger data substantially improving results. *ALBERT (A lite version of BERT)* [11] was proposed to enhance training and results by using sharing and factorization techniques.

A quite different variant is *FinBERT* or *Financial BERT* [2]. Unlike the others, the researchers behind *FinBERT* decided to develop a new version of *BERT* focusing on the *pre-training* phase. As discussed in this Chapter, *BERT* was pre-trained on a large corpus without a main theme. *FinBERT* was born from the idea to pre-train *BERT* on a financial-oriented corpus. The main goal of this procedure was to define a classification model for sentiment financial analysis. The concept is the same as the classic approach, but the language used changes. In the financial market, a specialized lexicon is used to describe price trends or predictions. So, financial sentiment analysis researches different patterns to describe the tone of text processed.

FinBERT follows this theory since it was pre-trained on a large financial corpus called *TRC2-financial* which contains about 46 thousand documents. Then, a sentiment analysis dataset was used adding a fully connected layer at the end of the model. Performing classification, the model was fine-tuned and evaluated through cross-validation. The developers specified that the model has been also trained to be a regression model [2].

3. State-of-the-Art Methodologies

Stock price trend has been an interesting topic in recent history since the growth of the inflation rate brought people to invest rather than save money. However, predicting the price trend is complicated because of its volatility and many influential factors.

Machine learning and deep learning models have been used to support investment in this kind of market. Two types of approaches are used to predict the future price of a stock: *Financial Sentiment Analysis* and *Forecasting*. The former consists of the prediction of the price direction based on a tone analysis of the financial news. The price will grow if the sentiment is defined as positive. On the other hand, the latter approach predicts the numeric value that the price will have in the future. The literature has various approaches related to both methodologies. Forecasting methods make use of both statistical and machine learning models to predict the correct value at a specific time. We presented in the first Chapter *ARIMA* models and the process to adapt a forecasting problem to a machine learning model. The first historical works tried to predict the stock price just using the numeric time series. Chen and Long in their papers [4] [13] explained this process. Then, new data were added to improve model performance. These features were related to the market analysis and comprehended various technical indicators.

Subsequently, Gondaliya et al [6] introduced the news (scrapped from social sources such as Twitter) in the form of newspaper articles or headlines to help the prediction. They applied both *BoW* and *TF-IDF* text representations techniques to adapt the text to the model. *Recurrent Neural Networks (RNN)* advent was a game changer since it brings improvement in many tasks including stock price forecasting. *RNNs* are artificial neural networks characterized by a memory system that allows them to take advantage of past information. They fit well with sequential data since historical behavior is a valid source of information. Unlike the classic *neural networks*, they don't consider observations as independent making use of previous computations to perform new ones. *Gated RNNs* were even more effective thanks to their properties that solve the vanishing gradient problem of classic *RNNs*. They include *Long-Short-Term Memory (LSTM)* and *Gated Recurrent Unit (GRU)*

[25]. *Financial Sentiment Analysis* models instead use classification models to predict the direction of the price. Initially, machine learning models were used to predict the sentiment of tweets and news. *Sentiment Analysis* is defined as a natural language processing task that is used to determine a sentiment value of a text or document, usually described as a positive, negative, or neutral tone. These algorithms are mainly used for social media analysis. *Sentiment Analysis* is often applied to posts on social media to discover people's sentiments on a particular subject. Many variants of this approach exist that differ from the output they provide. The application of *Sentiment Analysis* to finance-based documents is one of them. For this task, a recent work demonstrated that *Support Vector Machine (SVM)* models perform better than *Decision Trees* and *Naïve Bayes* classifiers [6].

Then, the rise of deep learning approaches allowed the extraction of complex and more useful features thanks to their multiple layers of non-linear functions. Many pieces of research confirmed the efficiency of the *RNN* and *Attention Mechanisms* in sentiment extraction of *Natural Language Processing*. Recent discoveries also brought *BERT* inside the *deep learning* group because of its capability of learning context.

If on the one hand, this system is more widespread on the other the forecasting methods meet more difficulties since it is undoubtedly simpler to predict the price direction than the exact value the price will have.

We will present two approaches: one belonged to the *financial sentiment analysis* sector and the other to the *forecasting* one.

3.1 Financial Sentiment Analysis approach

Sousa and colleagues in [21] studied stock price sentiment analysis fine-tuning *BERT* on a manually labeled dataset with three classes: positive, negative, and neutral. They used as the target dataset the Dow Jones Index before the opening time. They gathered financial news from several sources and applied tokenization to them.

Then, when the data was ready, they fine-tuned $BERT_{BASE}$ using their labeled dataset and cross-validated it with a 10-fold cv. In the end, they compared *BERT*'s results with a *Naïve Bayes* and *SVM* classifier showing how the *NLP* model outperformed the other ones

in terms of accuracy and F1-score which are metrics used in classification.

This study brings value to the new *BERT* model demonstrating the power of *Transfer Learning*. As discussed in the previous Chapter, Araci in [2] proposed a new version of *BERT* called *FinBERT* to perform *Financial Sentiment Analysis* which could have been used to improve the results of [22].

The same procedure was followed by Mishev in [16] where numerous strategies were applied. They tested both machine learning classifiers such as *Support Vector Classifier* and *Extreme Gradient Boosting* and deep neural networks such as *GRU* and *LSTM* to extract features. They also experimented with *BERT* and many of its versions to complete the entire field related to sentiment analysis.

Overall, *BERT* models outperformed the alternatives in terms of accuracy and F1-score, demonstrating these algorithms' effectiveness. Among many, *FinBERT* and *DistilBERT* were used. The first was one of the worst among the *BERT* models while the second reached great overall results considering its architecture and speed.

3.2 Forecasting approach

LNM Institute of Information Technology researchers proposed several strategies to break down stock price prediction publishing an interesting paper [21]. They proposed a new model based on *Generative Adversarial Network (GAN)* and *BERT* comparing it with *ARIMA* and *Gated RNN* models. They demonstrated how their model called *S-GAN* outperformed traditional forecasting methods in terms of RMSE on 5-day, 15-day, and 30-day predictions. They used as a dataset the historical data of Apple Inc. in a range of ten years (2010-2020) and applied standardization to it. Then, they split the dataset considering 70% as training and the remaining as a test set.

Input Features Besides the actual value of the price, they made use of various technical indicators to improve the prediction effort.

- *Moving Average (MA)*: computed considering the previous 7 and 21 days.
- *Bollinger Bands (BB)*: comprehend the moving average and its confidence interval.

- *Moving Average Convergence Divergence (MACD)*: it indicates the threshold where the market is overbought or oversold.

- *Relative Strength Index (RSI)*: it generates a signal on buying and selling momentum, so it is expressed as a dummy variable having three values, one for the bullish, one for the bearish, and one indicating neither.

- *Fourier transforms*: Fourier transformation applied to the closing stock price.

In addition, they performed sentiment analysis on several newspaper and article headlines classifying them as positive, neutral, or negative. The news was scrapped from Seeking Alpha and they match the price time interval. They also processed these texts using natural language tools before passing them to *FinBERT* which was the model candidate for this kind of task. Both the technical indicator features, and the sentiment ones were used as input features to feed the model during the training phase.

Baseline Models To give value to their study, the researchers identified three models as a baseline. Firstly, they selected an *ARIMA* model with 4 AR parameters, 0 MA parameters, and 1 difference applied. This model ran only on a univariate time series of the stock price.

Then, they trained two more sophisticated models which belong to the deep learning field: an *LSTM* model and a *GRU* model. The former was built with 128 cells with two dense layers having 64 and 1 unit respectively. Using MSE as a loss function they trained the model on 150 epochs. The latter was built with two *GRU* cell layers with 128 and 64 units along with two dense layers of 32 and 1 units. This time the number of epochs was reduced to 50.

Both *LSTM* and *GRU* layers allow the recurrent structure of the neural network to avoid vanishing and exploding gradient problems which could cause computational issues or bring the model to randomly predict the future. Thanks to three types of gates (input, output, and forget) *LSTM* layers manage the flow of information conserving useful past knowledge and shutting down useless ones. The forget gate decides which information has to be discarded or retained, the input gate learns the relations between the sequences

passed to the model and the output gate is used to propagate the information to the next cell.

GRU layers apply the same logic merging the input and output gate in a single one reducing consistently the number of parameters and increasing as a consequence the speed of the computation. Both of the variations of the *RNN* structure work exceptionally well with sequential data.

Forecasting model The proposed model is inspired by the work of Goodfellow et al [7] based on a generative model called *Generative Adversarial Network (GAN)*. The goal of such kinds of models is to analyze a collection of data (training data) to learn the probability distribution and be able to generate new examples from that.

GANs architecture is characterized by two components: a discriminator, and a generator. The whole process consists of a game, as Goodfellow defined it in his paper, between the two neural networks. The generator has the goal to learn to generate samples while the discriminator is built to distinguish between generated examples and real ones. So, the discriminator $D(X)$ should output the probability that the sample x is real. This value would be 1 if the sample is real and close to 0 if it is generated.

The generator's job is to fool the discriminator trying to create a new sample as close as possible to the real ones. Doing so, the game cited by Goodfellow could be expressed as a minimax game where the discriminator maximizes the expected conditional log-likelihood i.e., it tries to identify correctly the examples and the generator minimizes the distance between real data and generated ones.

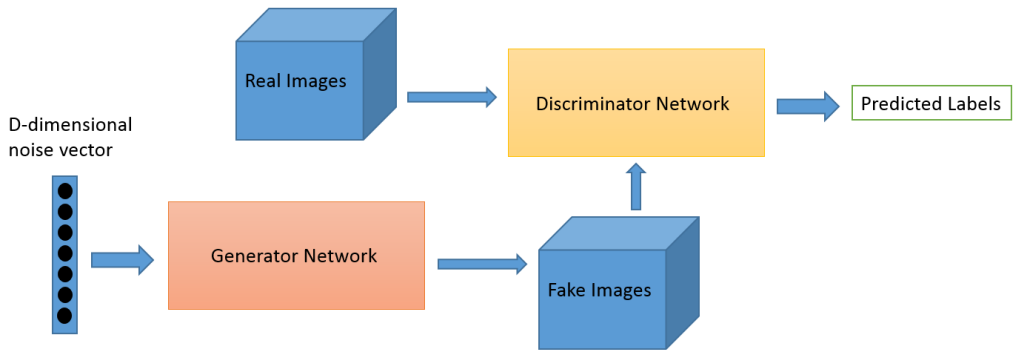

Figure 3.1: GAN Architecture

The architecture of the *GAN* is depicted in Figure 3.1 where it is easy to spot both components. The generator receives a noise vector and creates new fake samples. These new data are passed through the discriminator as the real ones, and it computes predictions. Comparing these predictions to real labels the discriminator is updated to refine the process. Backpropagation allows the generator to improve its capability of generating new samples. So, the generator is a black box to the discriminator while the latter cannot be a black box for the former.

GAN is born to deal with bi-dimensional data such as images however the researchers adapted it to generate sequences. They used *GRU* as a generator and a *Convolutional Neural Network* as a discriminator. Their final model called *S-GAN* is represented in Figure 3.2.

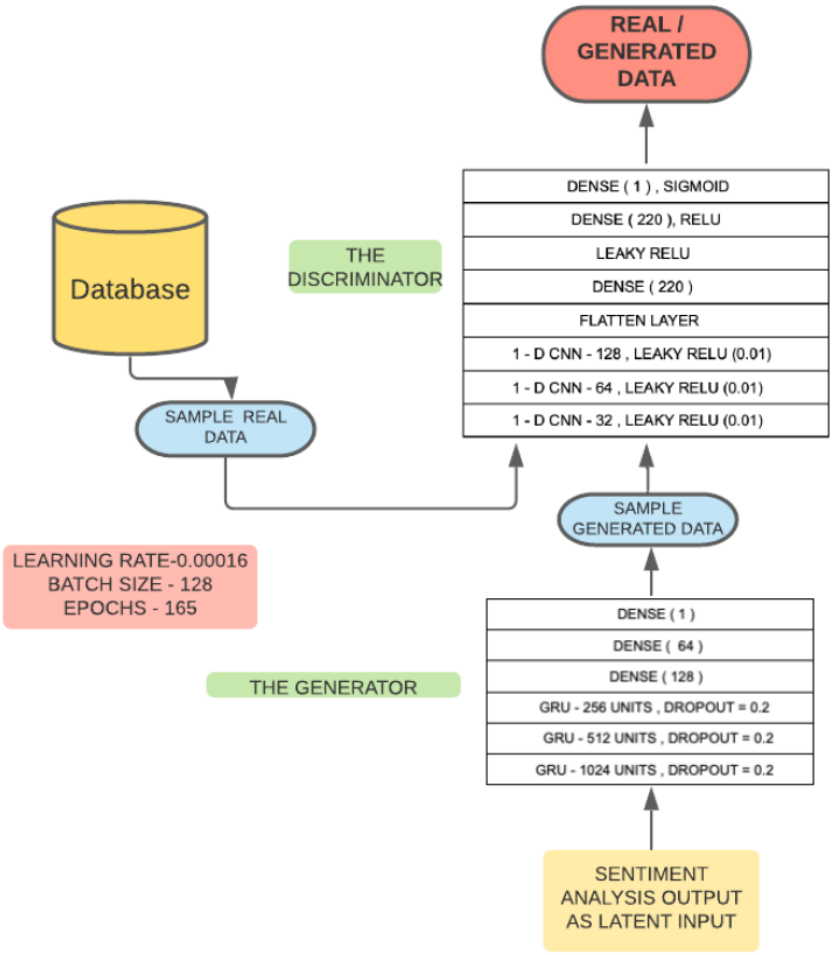

Figure 3.2: S-GAN Architecture

GRU generator is expressed by the following equation:

$$g_\theta : Z \longrightarrow X, \tag{3.1}$$

where θ are the weights of the neural network. Unlike the classic *GAN*, Z represents the output of the sentiment analysis. The generator aims to define θ in a way that the output would be close to the original distribution X. They used 1024 units in the first layer, 512 in the second, and 256 in the third alternating them with dropout layers to prevent overfitting. They added three dense layers at the end. The discriminator structure was composed of a unidimensional *CNN* with 32, 64, and 128 units for the first three layers followed by a *LeakyReLU* function which is an alternative to the *ReLU* and it is expressed as:

$$LeakyReLU(z) = \max(0.1z,\ z). \tag{3.2}$$

At the very end of the architecture, a dense layer with one unit was placed to produce the predictions.
The following equation synthesizes the entire process:

$$min_\theta max_w E[\log(D_w(X)) + \log(1 - D_w(g_\theta(Z)))]. \tag{3.3}$$

The goal of the discriminator is to distinguish between the real distribution of stock prices and the generated distribution. It tries to maximize the probabilities in the equation to allow the generator to put more effort into the generation of accurate samples.
The proposed model outperformed the previously cited ones in terms of RMSE. The most accurate baseline approach has been the *GRU* ones reaching a test RMSE equal to 2.96 while the *S-GAN* established a 1.83 RMSE reducing the error of roughly 38%. This was an outstanding result since they compared it also with a standard *GAN* model which obtained a 2.37 RMSE.

4. Natural Gas Prediction

In this Chapter, we will discuss the practical effort behind this thesis. As anticipated, the main goal is to predict the Natural Gas stock price value in a year that history will remember for the Russian invasion of Ukraine. Russia currently is the major exporter of natural gas, especially in Europe. Its controversial behavior impacts Europe's socio-economic sphere and the entire world.

So, the main aim of this work is to define and train a model capable to predict the future based on news related to the topic.

To reach this goal, this approach uses *BERT* directly as a forecasting model, unlike past research, to predict the future value of the stock price. To reduce the computational cost and due to system liability, *DistilBERT* was selected. Besides *BERT*, other models were selected to compare the proposed one with a baseline. The comparison was made with the *RMSE* which can picture the error on the same time series unit of measurement. In addition, various forecasting time horizons were tested such as 7 days ahead, 30 days ahead, and 120 days ahead in order to have a complete view of the model consistency over time.

The principal idea behind this thesis is to analyze historical news related to natural gas in parallel with the numeric time series to catch trend continuation or reversal at the right time. The procedure followed in this study is divided into several steps:

- *Data Retrieval*: numeric time series collection and news scraping.
- *Data Pre-processing*: this step is split into two phases, the first is related to the time series processing while the second is dedicated to textual data.
- *Data Splitting*: because of its nature, time series require a specific train-test split. In addition, cross-validation is performed as presented in Chapter 1 to compare models.
- *Modeling*: this section will be dedicated to enlightening the models used to reach the main goal. Two baseline models coming from different areas were selected to provide a benchmark. Then, the proposed model will be discussed. Moreover, an

additional path has been covered to better take advantage of all the possibilities the proposed models could bring.

- *Model enhancement*: this step will deal with some strategies adopted to improve the power prediction of the model.

- *Results and future developments*: at the end, the results obtained will be discussed graphically and using proper metrics. Besides, some future insights will be proposed.

4.1 Data Preparation

This main step was characterized by the collection of relevant data and their processing. *Scraping*, *Data Cleaning* and *Splitting* were performed to lay the foundation of the application.

4.1.1 Data Retrieval

Two kinds of data were collected to start the task. Firstly, the natural gas time series was downloaded from *Yahoo Finance*[1] selecting daily opening prices from January 2010 to September 2022. Then, textual data were scraped from *Reuters*[2] selecting all the headlines from articles in the same range of the time series. To do so, the python library *Beautiful Soup* was used. Headlines in *Reuters* are composed of a title and a short text synthesizing the news content. Both were collected to form a single text. Figure 4.1 shows the structure of the news from the cited website. Every news was stored in a row of a data frame during the scraping, so the dataset obtained had the shape represented in Figure 4.2: the column Head contains the title of the news while the column Text contains the brief description of the article topic.

[1] https://it.finance.yahoo.com/

[2] https://www.reuters.com/

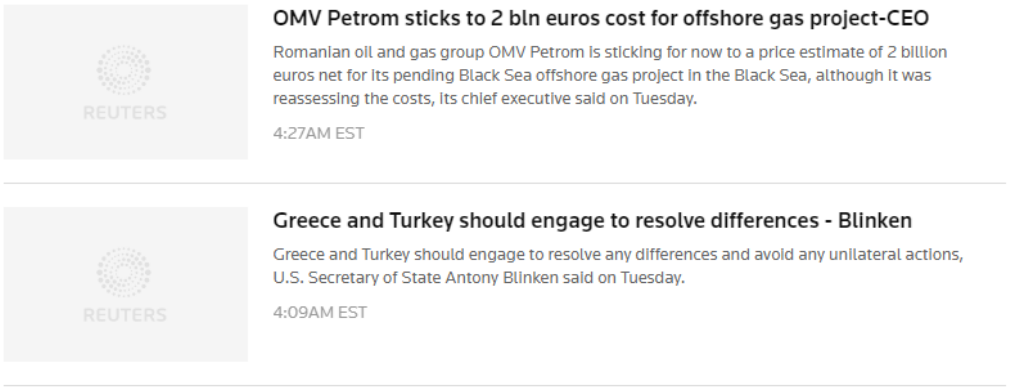

Figure 4.1: Reuters News

	Time	Head	Text
0	2022-10-02	UPDATE 1-Russian Deputy PM says restoration of...	Russian Deputy Prime Minister\nAlexander Novak...
1	2022-10-01	UPDATE 5-Eni expects halt in Russian gas flow ...	Italy's Eni said it\nwould not receive any of ...
2	2022-10-01	Italy's Eni says it will not receive Russian g...	Italy's Eni said on\nSaturday that it would no...
3	2022-10-01	Italy's Meloni vows government action to tackl...	Italy's next government will\ndirect its effor...
4	2022-10-01	EU leaders to discuss infrastructure security ...	EU leaders will discuss the\nsecurity of their...

Figure 4.2: Dataset Format

4.1.2 Data Preprocessing

The data pre-processing phase depends on the type of data, so it was necessary to handle the time series and the text data separately. The former needed simple and quick processing, which consisted of fixing the datetime column and properly ordering the price. The major transformation was applied depending on the model used, so we will discuss it later. On the other hand, the news required several steps to allow algorithms to process them. Firstly, the two textual columns of the dataset shown in Figure 4.2 were merged into one since they should bring the same information about that specific day. For the same reason and since the gas price was daily oriented, at the end of the steps listed below the news that belonged to the same day was compacted into one i.e., every day with at least one news refers to a single compact text comprehending all the news of that day.

- *Lower Case*: every single word was converted into a lowercase word.

- *Special character removal*: this step was necessary to remove URL-related symbols, and characters created during the scraping phase.

- *Stop words removal*: this step was necessary to remove all the words that couldn't bring information or discriminant power to the analysis. It was performed both before and after the punctuation removal step. Although the simplicity of this phase, some problems made their appearance. Words such as "u.s." which refer to the United State of America were removed because it was interpreted by the function as the pronoun "us". In addition, some company names were reported in the text with abbreviations that occasionally, after the removal of punctuation, could resemble articles or small speech particles. To solve this issue, a proper function was built to adjust the words related to this problem by replacing words that could be cut with new identifying words.

- *Punctuation and whitespace removal*: as anticipated, all the punctuation marks were removed as also the additional whitespaces.

The newly generated text was incorporated into the original data frame at the end of these steps and saved for later usage. This pre-processed news will be passed through the *BERT* tokenizer. In contrast, machine learning models could not handle textual data in this format, so the following operations were necessary before getting into the training phase. *Tokenization* was performed in conjunction with *Lemmatization* to bring the word to its root form. Then, a *TF-IDF* matrix was created to have a feature matrix ready to be used as model input. In the end, the numeric time series was joined to this matrix defining the final dataset.

In Chapter 1 we discussed the necessary procedure to tailor a dataset to a forecasting problem when machine learning models are selected. Since this approach makes use of these kinds of algorithms that process was followed. The target variable represented by the stock price was shifted to a value that corresponded to the forecast time horizon used. Before training the model, an ad hoc function was built to satisfy this necessity. On the other hand, *ARIMA* models didn't require this step.

4.1.3 Splitting

According to the common practice of classification/regression problems, the dataset was split into two components: training and test set. Since the current problem is characterized by the presence of sequential data, the two sets were created timewise. The training set comprehends all the observations between January 2010 and December 2021 while the test set the remaining ones. In addition, a validation set was extracted from the training set considering all the price values between January 2020 and December 2021.

Figure 4.3 represents the time series divided by colors into three groups. It is immediately clear how the 2022 year is quite different from the previous ones since the recent events shocked the economic world. The gas price, as already mentioned, was the most influenced because of Europe's dependence on it.

Figure 4.3: Time Series Splitting

As discussed in Chapter 1, it is strictly necessary to keep the time order when dealing with *ARIMA* models because of their nature. The same split was kept for all the strategies followed to have a solid comparison between them. However, this procedure comes with a drawback. Assuming a seven-day forecasting horizon, if the model is trained on the training set (2010-2019) it should produce predictions with the same parameters on the validation set whether the observation is related to early 2020 or late 2021 which is not fair. Predictions related to time intervals close to training observations should be more accurate than those far.

To mitigate this, a *Rolling Cross Validation* was suited and works as follows. Firstly, the

model was trained on the training set discussed above. Then, the validation set was divided into k folds timewise (a fold corresponds roughly to a month). After that, the model computed the predictions on the first fold i.e., the first month after the last observation of the training set, and the *RMSE* for that specific group was saved. At this point, the fold was incorporated into the training set and the parameters of the model were adjusted on those observations allowing it to produce new predictions for the next fold. This iteration process lasted until the last month of the validation set was used to make the predictions. In the end, the mean of the *RMSEs* was computed to have a single solid metric to explain the model prediction power. Figure 4.4 depicts the process from a synthetic point of view.

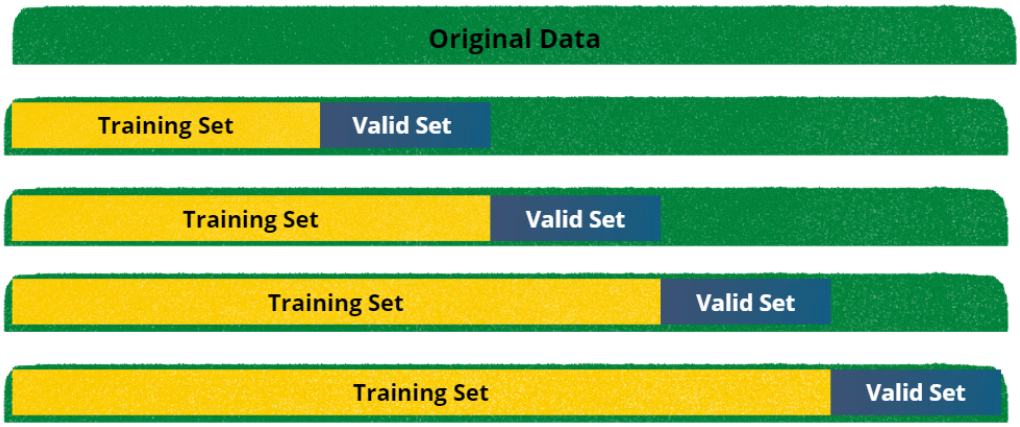

Figure 4.4: Rolling Corss-Validation

Before getting into the modeling phase, specifying and discussing *ARIMA* model behavior inside a single fold is important. Let's consider as an example a seven-day horizon. When the model computes the first prediction it started from the last observation of the training set to forecast the seventh. After that, to predict the eighth position it needed to perform the forecast action from seven days before. So, after every prediction, every first observation after the last of the training set was incorporated into the training set without updating the parameters. This procedure was applied only to *ARIMA* models.

4.2 Modeling

After the preparation phase, the modeling phase took place. Several models were defined and trained to reach the proposed goal. Firstly, some baseline models were selected then the proposed one was trained and upgraded. The comparison was performed on the validation set thanks to the *RMSE* which allows us to define the best model. This one was finally applied to the test set to see if it could predict observation belonging to the year characterized by a huge influential socio-economic factor.

4.2.1 Baseline Models

As anticipated, we selected two baseline models belonging to two different fields of time series forecasting. The first is a statistical *ARIMA* model which is notorious to be the first selection for many time series tasks. After the splitting phase, some exploratory data analysis was performed to discover internal patterns and useful information to incorporate into the basic form of the model. However, the most significant difference statistically proven was underlined before and post-war which unfortunately couldn't be inserted inside the training process since it was part of the test set. Monthly and yearly time series presented similar distribution and behavior so the introduction of dummies related to date-time features was discarded.

In Chapter 1 we explained all the necessary characteristics a time series should or must have to be passed through an *ARIMA* model. So, those were checked and solved as follows. In the first place mean and variance stationarity were analyzed. The series in the training set presented both non-stationary behaviors. To solve it, a *BoxCox* transformation was applied estimating a lambda equal to zero which is the proof of a logarithmic transformation. Then a first-period difference was performed to solve mean non-stationarity. Figure 4.5 depicts the series after the logarithmic transformation while Figure 4.6 after the first-period difference. Seasonality wasn't spot so a second-order difference wasn't necessary.

Figure 4.5: Time Series after logarithmic transformation

Figure 4.6: Time Series after first-period difference

Before the validation phase, the same pre-processing and transformation techniques were applied to the validation set. When the data was ready, the study of the *ACF* and *PACF* plot allowed the definition of candidates' values for *ARIMA* parameters. All of those models were trained, and the best ones were selected based on *AIC*. It is important to specify that the model was trained on just the univariate series of the stock price. Before getting into the validation phase, residuals were checked since some unusual behavior could have discarded some models. However, all the candidates passed the test and moved to the next step showing incorrelation. The final step concerned the application of the rolling cross-validation discussed in the previous paragraph. The model that reached the highest performance in the form of the lowest *RMSE* was an *ARIMA(0,1,1)* with just one *MA* parameter. In addition to standard *ARIMA*, we tried to add external features such as

the *Fourier series* in the form of sine and cosine without success.

The second kind of model selected as a baseline was a *Random Forest Regressor*, one of the best solutions among the classical machine learning models due to its incorporated feature selector. This characteristic allows it to select the most informative attributes as a discriminant in the classification or regression problem. In the first place, the model was trained just on the moving average with a value of seven to have a fair comparison with the *ARIMA* results. Then, the *TF-IDF* created in the pre-processing phase was added as an input feature matrix. Although the matrix presented many columns, no feature selection or transformation was performed for the reason explained above. The model evaluation was performed as discussed in the previous paragraph using the *Rolling Cross-Validation*.

4.2.2 BERT

After the discussion of the baseline models, we finally introduce the proposed solution for this problem. During the procedure, three *BERT* architectures were used: the classic *BERT* was applied to perform text tokenization, *DistilBERT* was selected as the regressor and *FinBERT* was tested firstly as a regressor and then for its creation: *Financial Sentiment Analysis*. *DistilBERT* was chosen because of its computational speed and light architecture that reduced the training and inferential time. At the end of its architecture, a dense layer with one unit was added to make it capable of handling regression problems. The loss function corresponds to the *Mean-Squared Loss*. The same procedure was followed for *FinBERT* when used as a regressor. The only difference was the computational time registered. *FinBERT* was almost 1.5 times slower. The training phase was set for 10 epochs with an early stopping based on the validation *RMSE* to prevent a useless time of training and preserve the best parameters.

Various alternatives were tested on the validation to create the best solution based on the news. Firstly, the *BERT* tokenizer was applied to the raw text of the news (right after the scraping phase) since it is structured to deal with this kind of text. Then, the pre-processed text discussed in the previous paragraph was passed to the tokenizer. The latter approach showed improved results on both *DistilBERT* and *FinBERT*, so the following analyses were performed on that corpus. Besides so, we also reported a better validation *RSME*

for *DistilBERT* in comparison to the *Financial BERT* which made us keep it apart for the moment.

4.2.3 Feature Extraction

We discussed in Chapter 2 the power of *Transfer Learning* describing both the pre-training and fine-tuning phases. This technique was presented as a combination of long and generic training followed by a more task-oriented and fast one. Another similar approach to engaging neural networks is the *Feature Extraction* technique. Briefly, it consists in extracting useful and task-specific features with a selected architecture and then using these attributes as input to a new model.

The deep learning world has many pre-trained algorithms that could be used to extract features. Usually, these models required some adjustments to perform the task. However, *BERT* was born to produce a formal representation of the input so it was able to perform this kind of task without any additional operations. We tested it in this thesis by resizing the *BERT* output to be a matrix to match the *Random Forest* input dimensions. To do so, the average of the values corresponding to the words between the *[CLS]* and *[SEP]* token and the *[CLS]* tokens was performed. As anticipated, the feature matrix was passed to a *Random Forest Regressor* which was evaluated while keeping the same conditions of the past attempt to make them comparable. Unfortunately, the results did not satisfy our expectations failing to correctly predict the price value.

4.2.4 Model Enhancement

Until now, the models based on *BERT* were trained with just the scraped news, unlike *Random Forest* which dealt with the moving average. To enhance the model power of prediction we decided to add two main features to its input. Since *BERT* can receive only textual data we added the moving average as a string to the news corpus at the specific time. By doing so, the model would be able to process the numeric information as a textual one. This addition highly improved the model's performance.

Likewise, a new feature was added taking advantage of the *FinBERT* variant. The news

used as input for the forecasting *DistilBERT* was passed to this model to produce the sentiment based on financial terms usage. The result was a string that could assume a value of positive, negative, or neutral facilitating the addition to the Forecasting *DistilBERT* input. Since *FinBERT* was developed mainly to solve this kind of task the result should be more accurate and give a hint on the trend direction. The model evaluation showed a slight performance improvement.

4.2.5 Test Phase

Concluded the validation phase, the best model was tested on the data related to the current year (2022) which is characterized by values that the gas natural stock price never reached. The same procedure followed for the validation set was adopted for the test set so after the model produced the predictions for the first group, this fold will be used to update the model parameters allowing it to compute new predictions for the next group. In the end, the mean of the *RMSEs* was computed obtaining the final test *RMSE*. This strategy allows us to understand how the model adapted itself over the months.

4.3 Results Discussion

This section is dedicated to reporting the results obtained in the validation phase which allowed the model comparison and selection and, in the test phase discussing the strengths and weaknesses of this methodology. In addition, we will propose a possible future solution that could bring improvement to our model.

4.3.1 Results

Table 4.1 reports the validation *RMSE* for different forecast horizons and all the models tested. *DistilBERT* trained with just the textual features couldn't improve the baseline model performance showing how the single news couldn't describe and predict the price movement. However, after the introduction of the moving average and the sentiment analysis output, the model accuracy experienced a great improvement. Considering the final

solution which reached an *RMSE* equal to 0.193 for 7-day prediction, it decreased the error by roughly 34% in comparison to *ARIMA* and 16% in comparison to *Random Forest*. In the same way, *FinBERT* improved the performance relating to baseline models even if it shows poor results in comparison to *DistilBERT*. Before testing it, the expectations on this model were high since it was pre-trained on financial documents. The reason behind the results obtained could be identified in the purpose for its creation. *FinBERT* was developed for Financial Sentiment analysis, so a forecasting problem couldn't be up in its potential. Besides, as anticipated, *FinBERT* was quite slower than the colleague in both training and inferential time. Having said that, it reached acceptable results overall.

Models	7-Day	30-Day	120-Day
ARIMA	0.294	0.622	X
Random Forest	0.23	0.438	1.13
DistilBERT	0.51	X	X
FinBERT Enhanced	0.218	0.392	0.872
DistilBERT Enhanced	**0.193**	**0.334**	**0.733**

Table 4.1: RMSE Results

Figures 4.7, 4.8, and 4.9 show the 7-day predictions on the validation set (represented in orange) for *ARIMA*, *Random Forest*, and *DistilBERT* respectively compared with the real values (in blue). It is clear how *ARIMA* couldn't anticipate price movement while the other two provide accurate solutions. Besides, the *DsitilBERT* model along with the price action shows better predictions, especially when dealing with price peaks. This difference is more evident in the final period of the validation set. This general behavior is respected for every time horizon demonstrating the consistency of the models tested over time.

Figure 13 depicts the prediction on different time horizons for the *DistilBERT* model where we can deduce how the model struggles to predict bigger values when the forecasting time horizon increases.

Figure 4.7: ARIMA Predictions

Figure 4.8: Random Forest Predictions

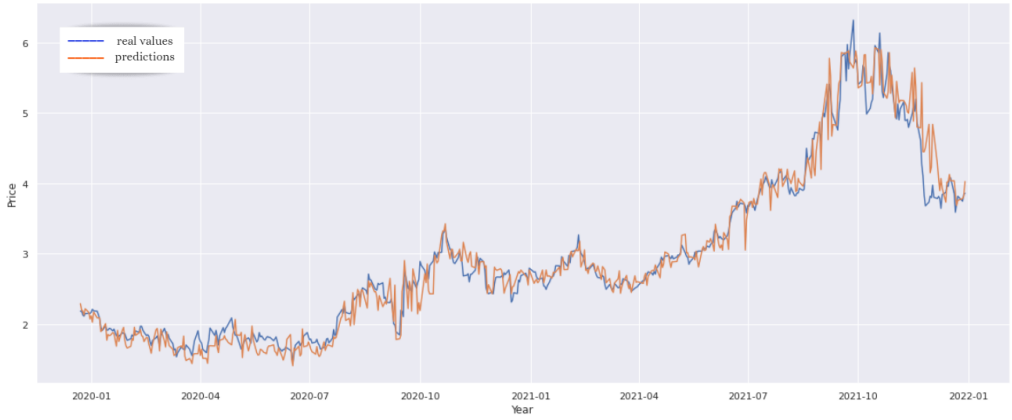

Figure 4.9: DistilBERT Predictions

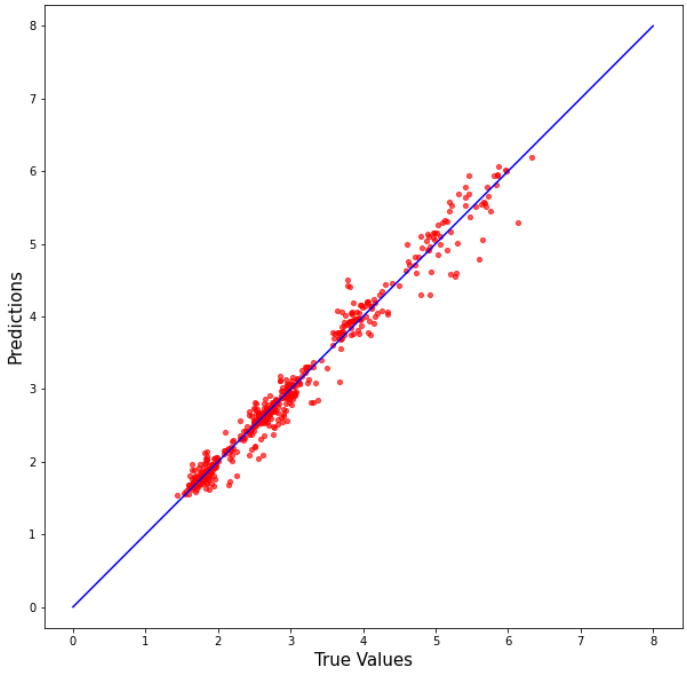

Figure 4.10: DistilBERT 7-Day Predictions

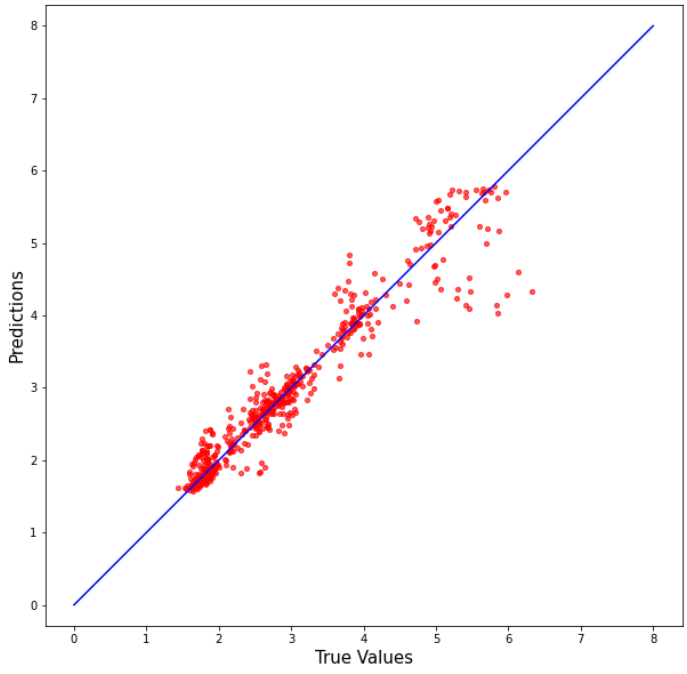

Figure 4.11: DistilBERT 30-Day Predictions

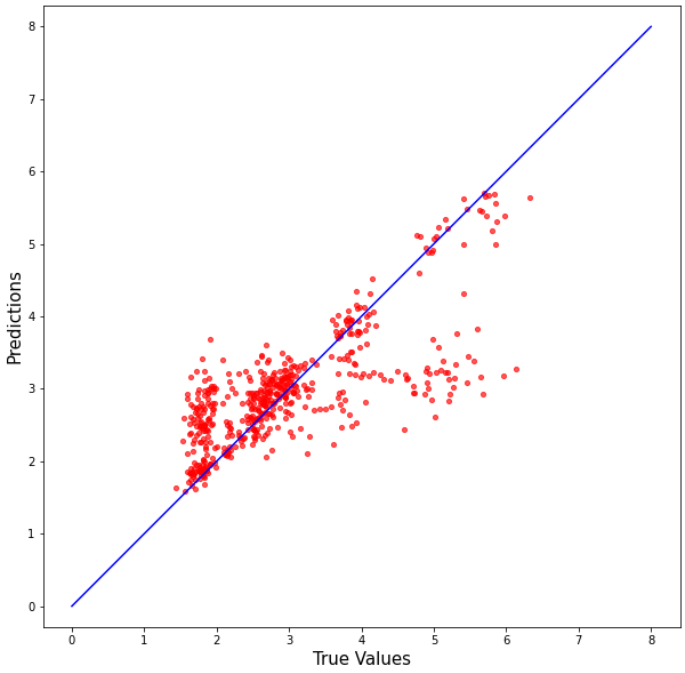

Figure 4.12: DistilBERT 120-Day Predictions

The candidate model that was evaluated also on the test set was *DistilBERT* with the feature enhancement. 2022 was a critical year so we expected a significant decrement in performance. As anticipated, the model produced the prediction for almost twenty days and then incorporated these observations to move to the next fold. Registering the *RMSE* for each group allows us to understand the model capacity of adaptation to new events such as the war.

Observing Table 4.2 where the *RMSEs* are listed we can identify the fold that comprehends the beginning of the conflict. After that, the model adjusted the parameters and the precision of the following predictions. Besides, it is possible to notice how the error suffered a drawback in the 7th group. This is represented by the price decrement during the summer months which was more consistent than the years before because of the volume of the 2022 price.

Folds	1	2	3	4	5	6	7	8	9
RMSE	0.347	0.448	0.383	1.594	1.082	0.887	1.827	1.084	1.296

Table 4.2: Folds RMSE

These two main movements never appeared in the previous years used as a training set forcing the model to predict a completely new situation. However, the model adjusted the prediction immediately after those two periods thanks to the rolling training. This is a good point since 2022 could be a valid example for future events that could have a huge impact on the natural gas price.

4.3.2 Future Developments

Since the introduction of sentiment analysis output produced better results in the terms of *RMSE*, we decided to begin a new analysis that could bring value in the future. Classical Sentiment Analysis is usually performed on social media posts to discover specific population reactions to certain events. This concept is based on the fact that knowing how people react and what they think as a collection of individuals could bring insight on a specific topic.

These two reasons, the increment of *RMSE* and this theoretical assumption, moved us to study how European reacted to the war.

The process started with data collection. All the tweets of European people were gathered thanks to a scraping library of Python named *Snscrape* allowing us to collect all the posts from January 2022 to September 2022 which is the same period as our test set of stock price time series. The tweets research was made delimiting the search to a circle with the center in Hamburg. Drawing a radius made possible the definition of a European zone. Besides, tweets were skimmed based on a hashtag-oriented filter that considered words related to the current war. Since Twitter imposed some restrictions on this process, we couldn't get older tweets leaving this scenario to future developments.

Once the gathering process was over, tweets were translated into the English language to facilitate both the pre-processing phase and the sentiment analysis algorithm. Tweets text was processed by applying just the necessary steps such as removing new line symbols or stop words. No additional modifications were made since the sentiment analysis algorithm used could identify and give value to punctuation and emoticons. It is called *VADER* Sentiment Analysis where *VADER* [8] stands for *Valence Aware Dictionary and sEntiment Reasoner*. We chose this method because of its accurate results in social media analysis

and above all, it provides more granular sentiment than other algorithms.

Before starting to search for relations and correlations between these results and the stock price, a quick exploratory analysis was performed. Figure 4.13 shows the number of positive and negative tweets over time emphasizing crucial moments of 2022 related to Ukraine and Russia. Two of them stand out over the others: the beginning of the war which presents the highest peak and the win of Ukraine at the Eurovision. The first is inherent to the topic while the second could be misleading.

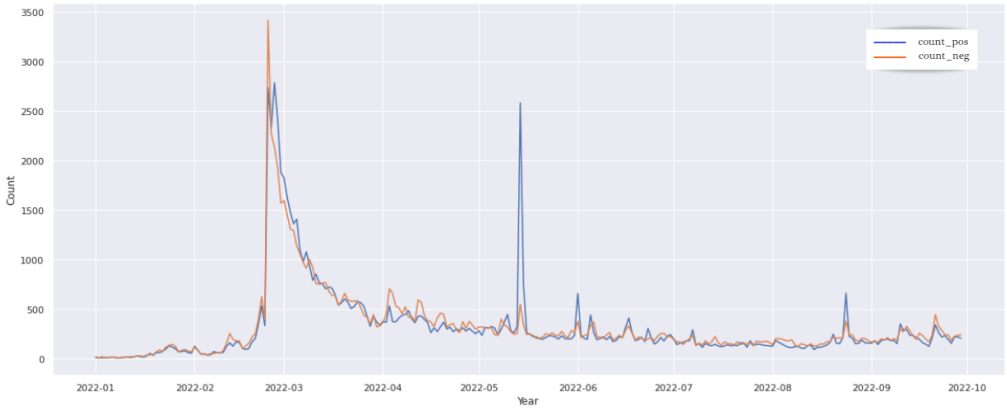

Figure 4.13: Positive and Negative Tweet Count

Different analyses were performed to find relations between this sentiment results and the stock price time series considering also as a separate sequence the sentiment of the news. Since the structure of the data was different, picturing a solid correlation graphically was quite complicated. Even after careful pre-processing the comparison couldn't state a statistically proven result.

So, we decided to test directly if those sentiment outputs could help to predict the natural gas price. Since we gathered the tweets only from 2022, we performed this methodology just considering this year.

Firstly, a dataset was created concatenating the stock price of 2022 and the tweet sentiment. A single day could be tied to multiple tweets, so we defined a unique solution for the single day based on how many positive and negative tweet was registered for that day. This allowed us to have a single value for the day. Then, a *Random Forest Regressor* was trained on a proper random subset of the 2022 and tested on the remaining observations.

The training was initially performed just considering the price and the moving average and secondly with the addition of the sentiment analysis output. This process was performed several times changing the seed with which the data was split. Doing so, we could compute several *RMSE* whose average stated the contribution of the sentiment output increasing the performance by roughly 11%. This is a good result considering the small amount of data used for the analysis. Future developments could extend the analysis by gathering more tweets related to past years and applying a filter that can mask events not related to the topic such as the Eurovision peak.

Conclusions

Time Series Analysis and *Natural Language Processing* could be paired to describe and predict an event over time. While the former has been established for some time, the latter gained huge success in recent years. Extracting information from the text could be a game-changer solution for many fields. Among many, the stock price prediction is a valid candidate since its behavior strongly depends on real facts. We tried to merge these two branches of *Data Science* to forecast the *Natural Gas Stock Price* which has been hit by the last tremendous events.

Various papers dealt with stock price prediction testing several models belonging to one of the two main methodologies: *Financial Sentiment Analysis* and *Forecasting*. We focused our efforts on the second trying to experiment with a recent *Language Model*, BERT, to predict the future value of the stock price.

Firstly, we defined and trained two baseline models: *ARIMA* and *Random Forest* whose results were acceptable in terms of *RMSE*. The introduction of pre-processed daily news in the machine learning model improved the prediction. After that, we tested a few alternatives of *BERT* adapting them to deal with a forecasting problem. We identified and used a cross-validation technique that considers the sequential order to evaluate and compare the models. Then, the best model, a *DistilBERT* trained on the news, the moving average of the price, and a sentiment analysis output generated with *FinBERT*, was tested on the 2022 set to measure its ability to interact with substantial price movements due to socioeconomic events. The final model was able to outperform the others decreasing the *RMSE* by a significant amount and also showing a strong capacity for adaptation to unprecedented facts. It immediately adjusted the prediction after the passage through the model of the fold containing the price associated with the beginning of the war.

In the end, we proposed a possible future integration demonstrating how topic-related tweets sentiment analysis could improve the prediction.

Bibliography

[1] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. “Machine learning from theory to algorithms: an overview”. In: 1142 (2018), p. 012012.

[2] Dogu Araci. “Finbert: Financial sentiment analysis with pre-trained language models”. In: *arXiv preprint arXiv:1908.10063* (2019).

[3] Vitor Cerqueira, Luis Torgo, and Igor Mozetič. “Evaluating time series forecasting models: An empirical study on performance estimation methods”. In: *Machine Learning* 109 (2020), pp. 1997–2028.

[4] Lu Chen et al. “A hybrid attention-based EMD-LSTM model for financial time series prediction”. In: (2019), pp. 113–118.

[5] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[6] Chetan Gondaliya, Ajay Patel, and Tirthank Shah. “Sentiment analysis and prediction of Indian stock market amid Covid-19 pandemic”. In: 1020.1 (2021), p. 012023.

[7] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[8] Clayton Hutto and Eric Gilbert. “Vader: A parsimonious rule-based model for sentiment analysis of social media text”. In: 8.1 (2014), pp. 216–225.

[9] R.J. Hyndman and G. Athanasopoulos. “Forecasting: principles and practice”. In: 2nd ed. Melbourne, Australia: OTexts, 2018.

[10] İlker Kocabaş, Bekir Taner Dincer, and Bahar Karaoğlan. “Investigation of Luhn’s claim on information retrieval”. In: *Turkish Journal of Electrical Engineering and Computer Science* 19.6 (2011), pp. 993–1004.

[11] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019).

[12] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[13] Wen Long, Zhichen Lu, and Lingxiao Cui. “Deep learning-based feature engineering for stock price movement prediction”. In: *Knowledge-Based Systems* 164 (2019), pp. 163–173.

[14] Pilehvar M. and Camacho-Collados J. “Embedding in Natural Language Processing”. In: 1st ed. Morgan and Claypool Publishers, 2020.

[15] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[16] Kostadin Mishev et al. “Evaluation of sentiment analysis in finance: from lexicons to transformers”. In: *IEEE access* 8 (2020), pp. 131662–131682.

[17] Aastha Nigam et al. “Text Representation for Machine Learning Applications”. In: ().

[18] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[19] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. “Activation functions in neural networks”. In: *Towards Data Sci* 6.12 (2017), pp. 310–316.

[20] Riad Sonbol, Ghaida Rebdawi, and Nada Ghneim. “The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review”. In: *IEEE Access* (2022).

[21] Priyank Sonkiya, Vikas Bajpai, and Anukriti Bansal. “Stock price prediction using BERT and GAN”. In: *arXiv preprint arXiv:2107.09055* (2021).

[22] Matheus Gomes Sousa et al. “BERT for stock market sentiment analysis”. In: (2019), pp. 1597–1601.

[23] Sanh V. et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019* (2019).

[24] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[25] Peter T Yamak, Li Yujian, and Pius K Gadosey. “A comparison between arima, lstm, and gru for time series forecasting”. In: (2019), pp. 49–55.