Lancaster University
School of Computing and Communications

Financial Natural Language Processing

Wang Pok (Thomas) Li

Supervisor: Dr Mo El-Haj

Thesis submitted in partial fulfillment of the University's requirements
for the degree of Bachelor of Science in Computer Science

Acknowledgement

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date: March 24th, 2023

Signed: *Thomas Li*

Abstract

The proposed project Financial Natural Language Processing is aimed to investigate and experiment with the effectiveness of different machine learning / deep learning models and techniques to predict stock market trends with News data. As part of its design and implementation, we would be using a dataset available on Kaggle.com combined with stock market data retrieved from yahoo finance. As part of the implementation, it would be separated into two parts and three different stages each. Data Processing, Model Training, and Model Evaluation. In part one, we would gather and process financial data according to what is provided in our News dataset. In part two, we would be experimenting with a classical machine-learning approach. In our final part, we would be going with a deep learning approach. This is due to the nature of NLP being a mix of machine learning and deep learning. The project would be implemented with Python using mainly NLTK, Scikit-Learn, and TensorFlow libraries.

Keywords: machine-learning, deep-learning, yahoo finance. nltk, scikit-learn, tensorflow, natural language processing, quantitative finance

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Aims

In this modern world, Artificial Intelligence has slowly integrated into our daily lives. From autocorrect, google search, Siri, and even as of typing this, grammar and spelling correction is also based on a product of a branch of Artificial Intelligence - Natural Language Processing. With the power and resources the financial industry offers, they are no exception to utilizing such leading technology to fulfill their monetary gains. In fact, it is one of the five industries that rely on AI the most [1].

The financial industry is known to have high working hours and require a lot of repetitive work and data analysis. This is where atomization comes in. It is also calculated that AI could potentially save North American banks $70 billion by 2025 [8]. Even more impressive, AI applications are estimated to potentially save $447 billion in aggregate cost saving for banks by 2023.

Besides saving money, AI is also used in making money. Natural Language Processing has been seen to become increasingly popular especially in the field of quantitative finance. Quantitative analysis heavily relies on statistical data to build models to make informed decisions. They also have to explain their decisions to their clients backed up with statistical data and it is something artificial intelligence models can provide whereas traditional qualitative analysis cannot. Their gain in interest in NLP is also explainable. Old-school mathematical methods are only useful for numerical data. The amount of available textual data such as news, analyst reports and etc has increased significantly alongside the growth of the internet. Secondly, the advancement in NLP techniques in recent years has made them more accurate and efficient. Utilizing these tools for numericalizing data such as news and articles where traditional methods cannot bring them an edge in such a competitive industry.

The quantitative side of the financial industry is not the only sector that utilizes AI. Most of the trades in the market right now are done by computers, we usually refer to these as Algorithmic Trading. Sweaty financial people in suits staring at a price chart at the New York Stock Exchange and buying and selling stocks might be the first thing that comes to the mind of a normal person when the financial industry is mentioned as it is portrayed in movies such as The Big Short or The Wolf of Wall Street, but this is not the reality.

According to 8topuz [13], algorithmic trading generates 50%-70% of equity market trades, 60% of future trades, and 50% of treasuries. This important role in the market works by executing orders based on market conditions, price, volume, and time. The market repeats itself and follows patterns, but the market is also constantly evolving, therefore it is inefficient to hard code an algorithm that would last and be sustainable as you would have to constantly update the algorithm

to work with the current market. AI offers such programs adaptability.

It is important to note that the goal of our project is not to build an algorithmic trading program or offer automated trading capabilities. Instead, our goal is to create a model that generates insights by quantifying popular traditional qualitative data such as News into a statistical form using Natural Language Processing Techniques. Furthermore, we do acknowledge the potential limitations in our research where the reliability of news sources might not be able to predict the stochastic nature of market dynamics. Since this is not an entirely new field and similar topics have been done before, because of the technological advancements in recent years, it is believed that we can further explore this topic with some of our state-of-the-art techniques.

1.2 Topic Overview

The structure of this report has been broken down into different chapters as below.

- **Background** chapter provides the relevant information for our project. Including the two main aspects of our project, Natural Language Processing and the Financial part of it. It also includes research we have done that inspires the methodology in our design and implementation.
- **Design** chapter guides us through the methodology we used to carry out our project. We would be talking about how we carry out the data collection process and the other two modeling approaches to our problem.
- **Implementation** chapter covers how we put our design into action, including the tools and code we have used for all three of our programs.
- **Evaluation** chapter covers the evaluation methodology for our results, the presentation and analysis of our results, and the findings we have gathered through them.
- **Conclusion** chapter covers the key findings of our project. It also covers the challenges and limitations we have faced and our proposed way of tackling them in future projects. It also includes our final thoughts throughout the course of our project.

Chapter 2

Background

This chapter covers the contextual information about our project. It is separated into two main sections, the financial part and the NLP part. We would also be covering some of the related research we have found that contributed to the design of the project.

2.1 NLP

Natural Language Processing is very broad and is its own category within Artificial Intelligence. The ultimate goal is for the computer to "understand" natural language so that it can be processed and used.

There are many uses for NLP. Such as machine translation, autocorrect, text summarization, text classification, chatbots, sentiment analysis and etc. This project will be designed based on sentiment analysis techniques.

Natural languages are surprisingly complex. Words can contain many different meanings and many things are contextually based. Sometimes words don't have to mean what they mean in a sentence, could be sarcasm or they simply just have multiple meanings. This kind of challenge has been here since the early days of NLP.

It all started back in the 1950s. Computer scientists attempted at the challenge at machine translation and they expected it to be solved within three to five years. With the technology back then they did not have much success. Since then, most of the NLP systems were based on hardcoded rules for example fixed replies from chatbots. Until the 1990s, when statistical NLP comes in and 20+ years later, in the 2010s, with the introduction of deep neural networks NLP really started seeing great success.

2.1.1 NLP Classical Machine Learning

In classical NLP, we usually separate the process into a few different stages and train the model using statistical methods. This was the most popular way to target NLP challenges during the late 1990s and early 2000s. These challenges usually take a great amount of data and "features" are then extracted from the input data. As seen in Figure 2.1, it is separated into three main parts, Preprocessing, Modeling and Output.

Before we extract the data we usually would apply some sort of preprocessing techniques on them such as stop words removal, Part-of-Speech tagging, and lowercase characterizing in order to make the extracted features more relevant. After all the preprocessing we would usually select

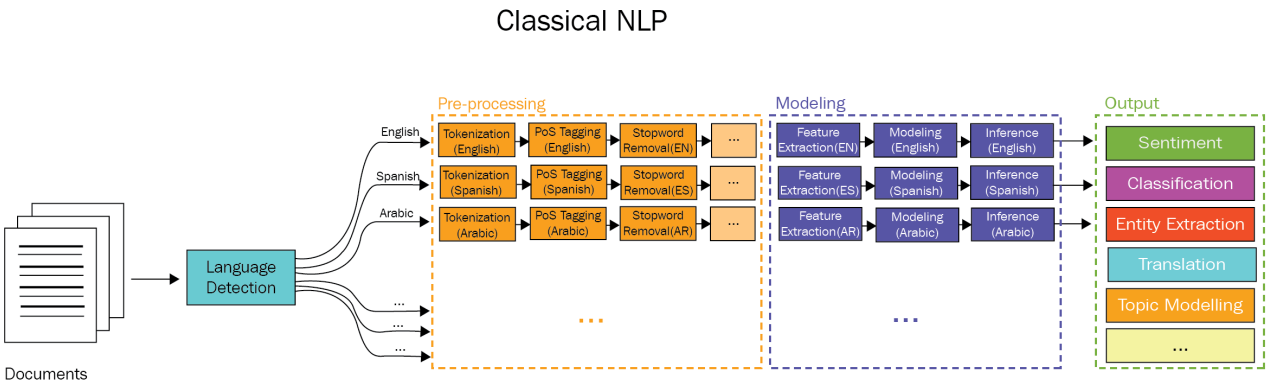


Figure 2.1: *Classical NLP approach (Image credit: https://s3.amazonaws.com/aylien-main/misc /blog/images/nlp-language-dependence-small.png)*

some sort of embedding method so that we can convert the text data into vectors so that our machines can understand. The most popular embedding techniques include Bag of Words, Term Frequency Invert Document Frequency which rely on the frequency of the words.

Once the preprocessing and embedding parts are done, we would put these extracted data into statistical machine-learning algorithms. Naïve Bayes, support vector machines, decision trees, and random forests are one of the most common ones out there. There are a few factors to consider when we are choosing the algorithm for our problems. Function complexity and amount of training data depend on each other as a high complexity function will only be able to learn with a large amount of training data and vice versa. The dimensionality of the input space is also a major factor, this is why we have to preprocess the data before using them for model training otherwise the extra dimensions can confuse the learning algorithm and reduce the accuracy of the trained model.

Last but not least there is the output. Depending on the task it could be simply a single class where the input text is classified or it could be a translation of a sentence to an entire piece of literature.

2.1.2 NLP Deep Learning

The predecessor to Classical NLP, Deep Learn Natural Language processing has gained popularity since the mid-2010s and now it is pretty much what all real-life NLP-related products are implemented on. As seen in Figure 2.2, it is simply just a neural network, unlike Classical NLP which is separated into three different parts. Although we can classify the neural network by tasks. In order to make sense of the input texts, we first have to pass the text data through a dense embedding layer which it converts texts into numbers, then we have hidden layers and an output layer.

There are three remarkable architectures I would like to mention that contributed the most to Deep Learning NLP's development, Recurrent Neural Networks, Long Short term Memory, and Transformers. The concept of RNN was first introduced in 1986, the technology back then wasn't mature enough to be implemented but in the last decade that became a possibility. It takes an input sequence and processes one element at a time while maintaining a hidden state of a summarization of data it has seen. It then goes through a loop passing in this hidden state back to the first stage until the entire sequence is processed. This recurrence allows the dependencies to be captured. LSTM is also a kind of RNN, compared to basic RNN this architecture "remembers" the past better. Inside LSTM it consists three gates, input, output and forget. The input gate

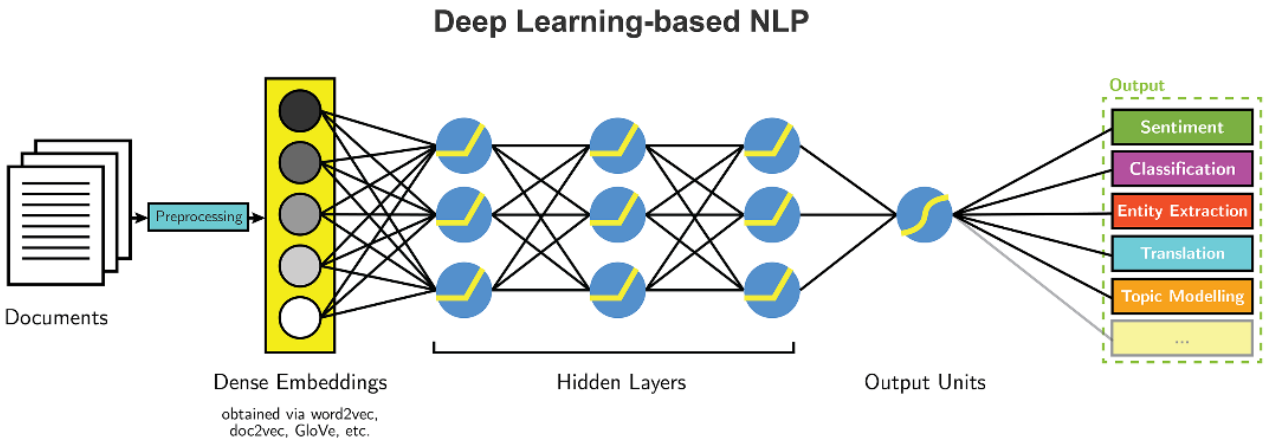


Figure 2.2: *Deep learning approach for NLP (Image credit: https://s3.amazonaws.com/aylien-main/misc/blog/images/nlp-language-dependence-small.png)*

decides what useful information to remember, the output gate decides what to output and the forget gate decides what to forget if the context has changed. This was mainly what was used until Transformers are introduced.

The encoder-decoder architecture revolutionized the world of NLP. Since the publishing of the paper "Attention Is All You Need" [20] which was made to target text translation but this made so much more impact than we expected. The attention mechanism allows each target word to have its own attention vector therefore able each vector to focus on different parts of the input data by assigning weights to them separately. Currently, this state-of-the-art architecture is what is used in the majority of NLP applications including the infamous ChatGPT which utilizes the decoder part of the architecture. In this research, we will be using BERT, the encoding part of the architecture for the encoding of our texts.

2.2 Financial Analysis

In trading, there are many factors on why the stock price is going up or why the stock price is going up or down. The main two categories are Qualitative Analysis and Quantitative Analysis. Within each, there are also subcategories such as Technical Analysis, Fundamental Analysis, and Sentiment Analysis.

The objective of this project is to quantify traditional qualitative methodologies using Natural Language Processing techniques with the goal of predicting stock trends based on people's reactions to the stock market for their change in sentiments after experiencing data including fundamental variables.

2.2.1 Qualitative

Qualitative analysis involves examining data to identify patterns, themes, and other qualitative features to help researchers understand the underlying meaning of a phenomenon. It could be done via conducting interviews, analyzing news and social media posts, conducting surveys, or simply observing human behaviors. Here are two subcategories of qualitative finance.

Fundamental Analysis

When it comes to trading, forex, stocks, ETFs, crypto, and everything that can be traded out there, are all based on some fundamental factors that could have an effect on their prices. There are two approaches, Top-down and Bottom-up. These consist of four layers, Economy, Sector, Industry, and Company. The top-down approach starts off by analyzing macroeconomic factors such as interest rates, GDP, and inflation. Whereas the Bottom-up approach is the opposite, where it focuses more on the microeconomic factors such as a company's financial metrics and earnings. In this research, we would be taking more of a top-down approach as the dataset we have focused on world news which is more related to the economy than a particular company's status.

Sentiment Analysis

When it comes to short-term trading, sentiment analysis comes more into play. "It is often seen that despite long-term fundamentals showing an uptrend, a currency remains down, due to an overall 'bad mood.' This bad mood means that the vast majority of traders are committed to a down position, due to some reason. Such sentiments often help traders assume a particular position." [14]. In this research, depending on the results, we might also be able to tell if there is a correlation between stock trends and general public sentiments, which we would assume to exist, to begin with.

2.2.2 Quantitative

Quantitative analysis involves measuring and analyzing numerical data to identify statistical patterns, trends, and relationships. It could involve financial modeling where financial data is used to create mathematical models or utilize computers to create machine learning models to learn financial patterns. Our project is sort of an overlap between both qualitative and quantitative as our goal is to quantify qualitative variables.

Here is an example of a subcategory of quantitative finance.

Technical Analysis

"Technical analysis is a tool, or method, used to predict the probable future price movement of a security – such as a stock or currency pair – based on market data." [18]. Traders believe that historical price movements and patterns generally repeat themselves and therefore they would use indicators such as moving averages, Fibonacci numbers, momentum, support, and many others to attempt to find out what a good selling or buying point is. This analysis is not related to this research as the data we are using only includes the change of prices within a timeframe and does not go into detail about the change of stock pattern or other indicators.

2.3 Related Researches

Similar topics are done many times before but since then there have been many new improvements and technology on the topic. Most of the successful projects are not open-sourced and are owned by large banks and companies for their own profits therefore resources related to these projects are very limited. There was a competition hosted on Kaggle.com by the company Two Sigma three years ago at the time of writing and after the competition ended, all data and related information

are removed [16]. Initially, I did request their news data and stock market data for research purposes but I have heard no reply thus I have decided to source my data elsewhere or even collect my own.

Data aside, I have also found another few research papers that tried to achieve a similar goal.

The first one is "Machine Learning for Stock Prediction Based on Fundamental Analysis" [7]. During the data preparation phase they used financial data extracted from companies' quarterly filings and in comparison in this research financial data would be sourced from a combination of news sources. At the learning state, they trained their models using Feed-Forward Neural Network, Random Forest, and Adaptive Neural Fuzzy Inference System. These are only three out of the many models out there and in this research instead, we would be exploring algorithms such as K Nearest Neighbor, Support Vector Classification, Gaussian Naïve Bayes, and three more others for the Classical Machine Learning implementation. For the Deep Learning implementation, we would be simply using a BERT encoder and a simple neural network.

The second related research I have found is "Deep Learning for Event-Driven Stock Prediction" [5]. This research is more similar to my second implementation. For its data extraction and embedding part, it used a neural tensor network to extract features such as a tuple (O1, P, O2), O1 being the actor, P being the action, and O2 being the object. Which in this research I would be attempting to use the Stanford CoreNLP dependency parser to extract these values for one of the techniques I am using for my Deep Learning implementation. This research extracted more than 10 million events from Reuters financial news and Bloomberg financial news. They do have more training data but they are all from the same two sources which could create a bias for the trained model creating inaccurate results. For their deep prediction model, they used a convolution neural network. Separating their data into three different time spans, monthly, weekly, and daily. In my implementation, I would train a separate model for each time frame instead as there is no way of classifying event length using the data for this research.

Chapter 3

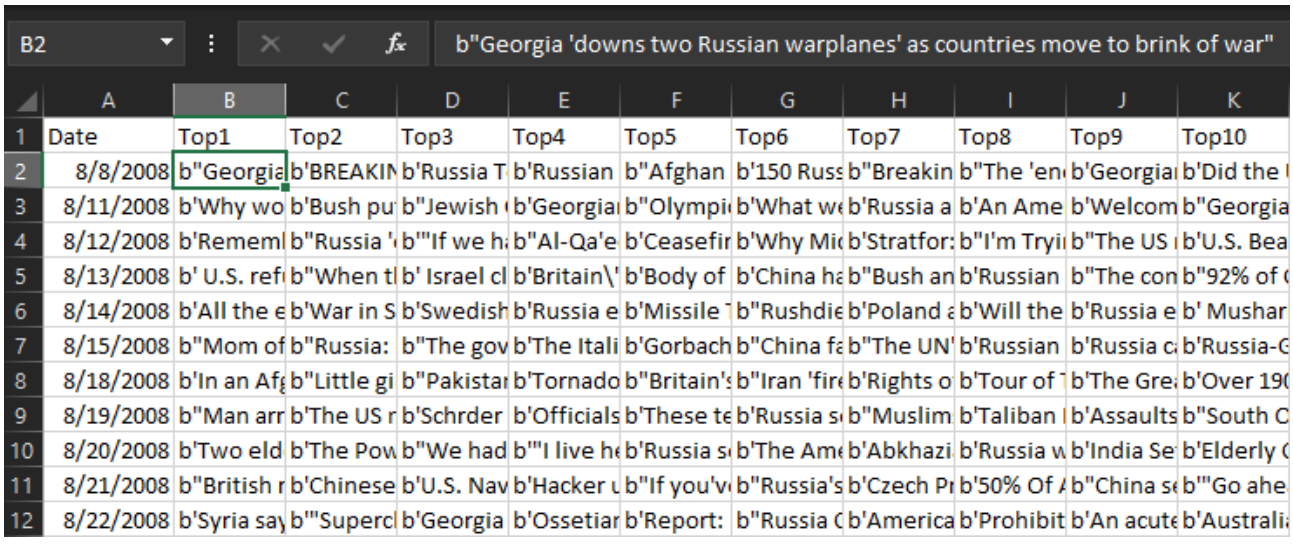
Design

3.1 Design - Data Collection

In order to train the machine to predict stock movements with news titles we would need to gather our data from multiple sources. First, we need to collect news headings and financial data that match the news. The news dataset I am using here came is resourced and collected by AARON7SUN and made available for download on Kaggle.com [17]. According to clickgiant's article "The Importance Of Headlines", they stated, "With the amount of content being pumped out onto the web, the average Internet surfer will read 80% of headlines, but only 20% of those surfers will actually read the article". Therefore news headings were chosen instead of news content as they would have a greater impact on a trader's fundamental analysis.

3.1.1 News Dataset

This dataset was originally collected from www.reddit.com/r/worldnews. The data consist of the top 25 news headings for each day from the 8th of August 2008 to the 1st July 2016. These headings are voted on by Reddit users and we can assume the more votes the more relevant to the public eyes it is. Here is a snapshot of the dataset 3.1:



B2 | b"Georgia 'downs two Russian warplanes' as countries move to brink of war"

	A	B	C	D	E	F	G	H	I	J	K
1	Date	Top1	Top2	Top3	Top4	Top5	Top6	Top7	Top8	Top9	Top10
2	8/8/2008	b"Georgia	b'BREAKIN	b'Russia T	b'Russian	b"Afghan	b'150 Russ	b"Breakin	b"The 'en	b'Georgia	b'Did the
3	8/11/2008	b'Why wo	b'Bush pu	b"Jewish	b'Georgia	b"Olympi	b'What we	b'Russia a	b'An Ame	b'Welcom	b"Georgia
4	8/12/2008	b'Rememl	b"Russia '	b'"If we h	b"Al-Qa'e	b'Ceasefir	b'Why Mi	b'Stratfor:	b"I'm Tryi	b"The US	b'U.S. Bea
5	8/13/2008	b' U.S. ref	b"When t	b' Israel cl	b'Britain\'	b'Body of	b'China h	b"Bush an	b'Russian	b"The con	b"92% of
6	8/14/2008	b'All the e	b'War in S	b'Swedish	b'Russia e	b'Missile	b"Rushdie	b'Poland a	b'Will the	b'Russia e	b' Mushar
7	8/15/2008	b"Mom of	b"Russia:	b"The gov	b'The Itali	b'Gorbach	b"China fa	b"The UN'	b'Russian	b'Russia c	b'Russia-G
8	8/18/2008	b'In an Af	b"Little gi	b"Pakistar	b'Tornado	b"Britain's	b"Iran 'fir	b'Rights o	b'Tour of	b'The Gre	b'Over 19
9	8/19/2008	b"Man arr	b'The US r	b'Schrder	b'Officials	b'These te	b'Russia s	b"Muslim	b'Taliban	b'Assaults	b"South C
10	8/20/2008	b'Two eld	b'The Pow	b"We had	b'"I live h	b'Russia s	b'The Am	b'Abkhazi	b'Russia w	b'India Se	b'Elderly
11	8/21/2008	b"British r	b'Chinese	b'U.S. Nav	b'Hacker u	b"If you'v	b"Russia's	b'Czech Pr	b'50% Of A	b"China s	b'"Go ahe
12	8/22/2008	b'Syria say	b'"Superc	b'Georgia	b'Ossetiar	b'Report:	b"Russia C	b'America	b'Prohibit	b'An acute	b'Australi

Figure 3.1: *Snapshot of news heading dataset*

3.1.2 Yahoo Finance API

The Yahoo Finance API is a range of publicly available libraries to obtain historical and real-time records of financial data as shown on Yahoo Finance (https://finance.yahoo.com/) Figure 3.2.

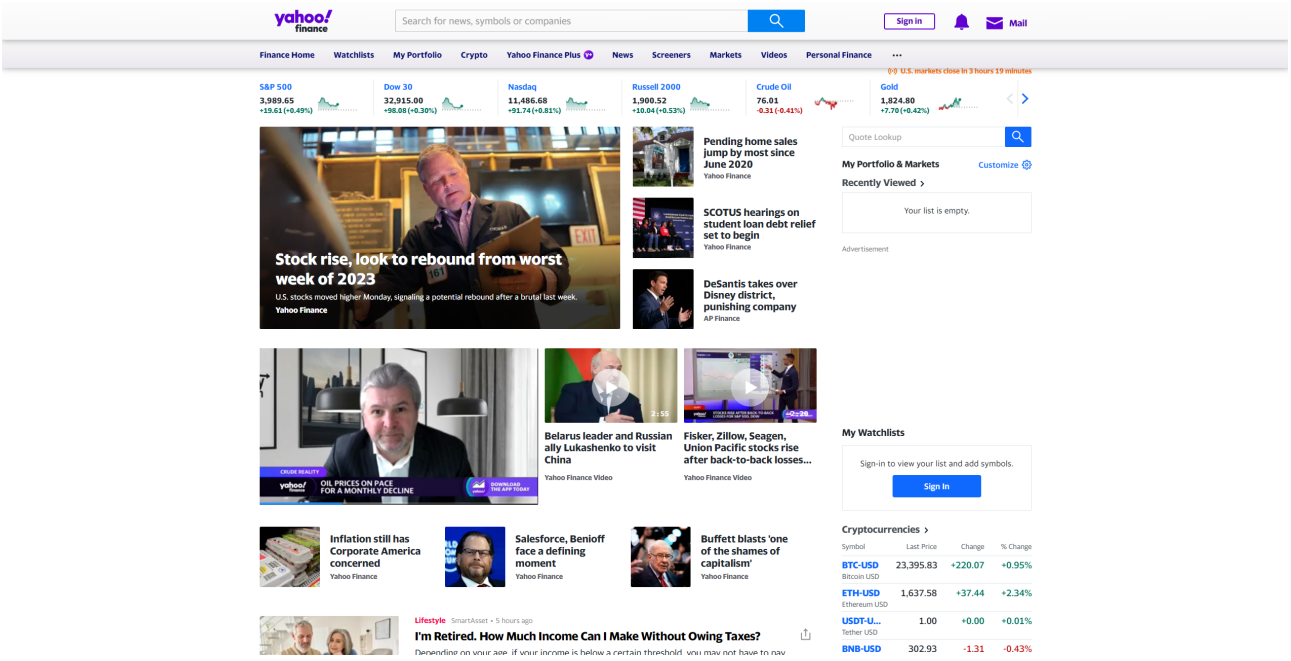


Figure 3.2: *Home page of Yahoo Finance*

I chose this service for multiple reasons. First, it has a very large range of data, beyond the standard core data it also offers options and fundamental data. Second, it is really easy to use and implement, it also offers a variety of functions to help you collect the specific data you need. It is also free to use therefore it made a great option for us.

There are also downsides. Due to the fact that it is no longer fully supported, it wouldn't be a reliable API for building products. It also has a limited rate for each IP address. With IP-based authentication, we are limited to 2000 calls per hour, but this is not a problem for us since we are not building a real-time application but only using it for data gathering.

3.1.3 Dataset Gathering and Combination

In order to test out the performance of each combination of data, we would be allowing the data collection program to take in a couple inputs, timeframe of each financial data point and the ticker (stock quote e.g. MFST, APPL, TSLA). In this project we would be experimenting with the S&P500 prices. "The Standard and Poor's 500, or simply the S&P 500, is a stock market index tracking the stock performance of 500 large companies listed on stock exchanges in the United States. It is one of the most commonly followed equity indices." - [15]

Also according to [15], "As of August 31, 2022, the nine largest companies on the list of S&P 500 companies accounted for 27.8% of the market capitalization of the index and were, in order of highest to lowest weighting: Apple, Microsoft, Alphabet (including both class A & C shares), Amazon.com, Tesla, Berkshire Hathaway, UnitedHealth Group, Johnson & Johnson and ExxonMobil." These blue chip stocks are usually very heavy fundamentally analysis based as their market cap (the total market value of publicly traded shares) are very high and the price is less

likely to be moved by technical analysis traders such as day traders as it would take a great amount of money to move the price of the stock.

Any ways once we got the data we would have to label the news heading dataset. We will be creating multiple Labels based on a different calculation to test out which one might be more relevant to the dataset we have.

Binary Labeling Method I: Change in price within each set of data with:

$$OP = \{p \in Q : p \text{ is a opening price}\}$$

$$CP = \{p \in Q : p \text{ is a closing price}\}$$

$$f(x) = \begin{cases} x^0, \text{if } x = 0 \\ x, \text{otherwise} \end{cases}$$

$$Label = \{f(\text{sign}(cp–op))\}$$

This would give us a list of binary Labeling outputs. If the result is negative then we label it "-1" which means the price has gone down and if it stayed the same or increased we label it "1" We will also include the rest of the data inside the dataset as they might come in useful and if not we can simply just discard them.

Ternary Labeling Method I: Three way split of change in price percentage

$$percentage\_changes = \{p \in Q : p = (CP_n–OP_n)/OP_n\}$$

$$lower\_split = \{x \in percentage\_changes : x > h\}$$

$$higher\_split = \{x \in percentage\_changes : x < l\}$$

Where h is the value of the higher 33.dot3% of the distribution of percentage_changes and l is the value of the lower 33.dot3% of the distribution of percentage_changes

$$f(x) = \begin{cases} 1, \text{if } x \in higher\_split \\ -1, \text{if } x \in lower\_split \\ 0, \text{if } x \notin higher\_split \cup lower\_split \end{cases}$$

$$Label = \{f(percentage\_changes)\}$$

This method first calculates the percentage changes in price each day and split it into three separate parts. If the percentage change is in the 1/3 distribution of the higher range of all data, then label it 1. Equally, if the percentage change is in the lower range of the 1/3 distribution of all data, then label it -1. The rest of the data would be treated as 0.

Ternary Labeling Method II: Change in momentum in the next N trading days

$$X = \{p \in Q : p \text{ is a closing price}\}$$

$$Label = \{\text{sign} \sum_{n=1}^{n-1} (\text{sign}(x_{n+1} - x_n))\}$$

This would give us a list of Labels that potentially indicates if the news heading has an effect on the momentum of the price in the next N days. If the number of increases in two consecutive days is larger than the number of decreases in two consecutive days then we label it 1, if it is lower then we label it -1 and if they are the same then we label it 0.

We would be experimenting with N=5 with the goal of getting a more balanced amount of labels. Because the total number of momentum changes = N - 1, which in our case makes 4 days of momentum changes for each time interval that gives 1/3 chance for each labeling within the selection. Otherwise, the larger N is, the smaller the chance that the data would be labeled 0.

Extra label: Change in volume in the next N trading days:

$$V = \{v \in Q : v \text{ is a trading volume } \}$$

$$VolChange = \{\sum_{n=1}^{n-1}(v_{n+1} - v_n)\}$$

A trading volume is a measure of the number of shares traded during a particular time period. This can possibly indicate the amount of attention the stock is getting from its traders. We take the same amount of N days as above to hopefully find the relations between the price momentum and the change in volume.

The change in volume is calculated by the sum of changes in volume in the next N days. A large positive number could possibly mean it is getting more attention thus more people trading on it, a small positive or negative number could mean not much impact and a large negative number could possibly mean it is refraining people from trading the stock.

In order to increase the association with this extra data we calculated we would be using the same N as Ternary Labeling Method II.

We would also leave in the rest of the data such as opening price, closing price, highest price, lowest price, volume, etc into the dataset and discard the unneeded information on the go while we go through the preprocessing stage of our data.

Check Figure 3.4 for the design of the data collection program.

Figure 3.3: *Combined Dataset with a price of S&P500 at a 1 Day timeframe*

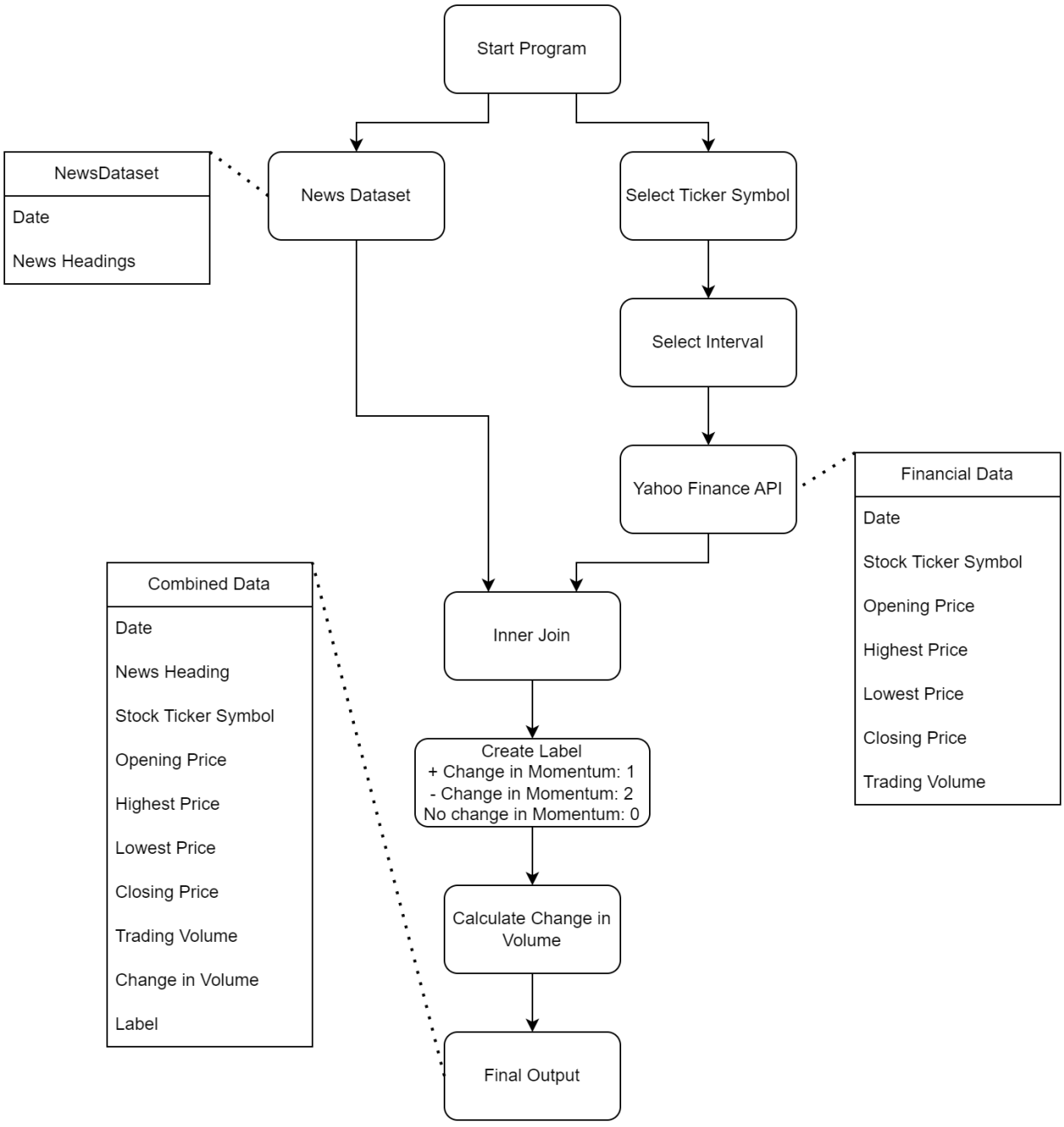


Figure 3.4: *Flowchart of data collection program*

3.2 Design – Classical Machine Learning Modeling

For this program, I would first like to show a flowchart of my design Figure 3.6.

The whole program is separated into two main stages, Preprocessing and Modeling, which include substages Tokenization, Text Preprocessing, Dataset Splitting, Encoding, Model Fitting and Evaluation.

3.2.1 Preprocessing

In the first stage, we have the dataset in the form of ["Heading", "Label"]. "Heading" is a string type object for example "2 weeks after controversial pharma CEO Martin Shkreli announced he would lower the price of Daraprim, it's the exact same price". In order to convert this news heading into something the machine learning algorithm understands, we have to first have to tokenize it. Tokenization is basically setting each word into its own token so that they are separated into their own to be processed. But the text is still full of noise that possibly would contribute nothing to our model. The usual practice would be removing punctuations, stopwords and numbers. These words don't usually have too much of an effect on the overall context of the text. Another important technique to do is to uncapitalize the words. For example the two strings "FinAnCe" and "finance" both have the same meaning but they are written differently so the program would treat them as two separate words at the encoding part.

We mentioned this encoding process but why do we need it? Till now these tokens are still stored in a string form and they really don't do any good to us. That's why now we have to convert these tokens into vectors so that our computers can understand and utilize them. There are a few classical word encoding techniques such as Bag-of-Words, TF-IDF, One-Hot Encoding, Hash Encoding, Count Encoding, and more. A simple word encoding technique such as One-Hot Encoding does the job well for one word but it can not capture the meaning of a phrase or when a phrase is made out of multiple words. That's why we can use the N-gram technique to capture longer-range dependencies and patterns in text.

In this program, we would be experimenting with CountVectorizer, TfidfVectorizer, and HashingVectorizer. CountVectorizer counts the frequency of each world in the given text. It creates a document-term matrix where the rows and columns represent a document and a word where the frequency of the corresponding word is stored in the cells. TfidfVectorizer does the same thing as CountVectorizer but it also assigns a weight to each term to reflect how important it is to the document. The weight is determined based on the rarity of the word within the document and the frequency we have calculated in the document-term matrix. HashingVectorizer on the other hand is a bit different. It hashes each vectorized element in the document to a hash map and the vector increases each time the word appears in the document. Besides these feature extraction algorithms, we can also implement our own text-parsing / part-of-speech tagging feature extraction method. There are three different types of data we can extract [Actor, Action, Object] or [Actor, Action] or [Action, Object]. These could help us identify the main features of the event happening inside the news heading.

Once we have finished encoding and preparing the data to be fed into statistical algorithms, we will have to split our data into two parts, one for training and one for testing. If we train and test our model on the same dataset, we could risk overfitting which occurs when a model is too complex and fits the training data too closely. Splitting the dataset can also help us with evaluating our model on new and unseen data. If the model performs well on the testing data then we can assume it would also work well on any new data. The usual practice would split a dataset by 80:20 but

there isn't an exact portion that you must follow, it depends on the available data. If there are not sufficient data then we can give the training dataset a larger portion and as it could help the model learn more about the underlying patterns.

3.2.2 Modeling

With the training and testing data prepared, we are now fully ready to train our machine-learning models with them. Here in this program, I have chosen six different algorithms, K Nearest Neighbor, Support Vector Classifier, Gaussian Naïve Bayes, Logistic Regression, Linear Discriminant, and Decision Tree Classifier. Figure 3.5 shows a comparison of 10 different classifiers' decision boundaries. As you can see they all show a certain kind of pattern and they all perform differently with datasets that shape differently. This is why we are testing six different classifiers here so that we can find out which one suits our data the best.

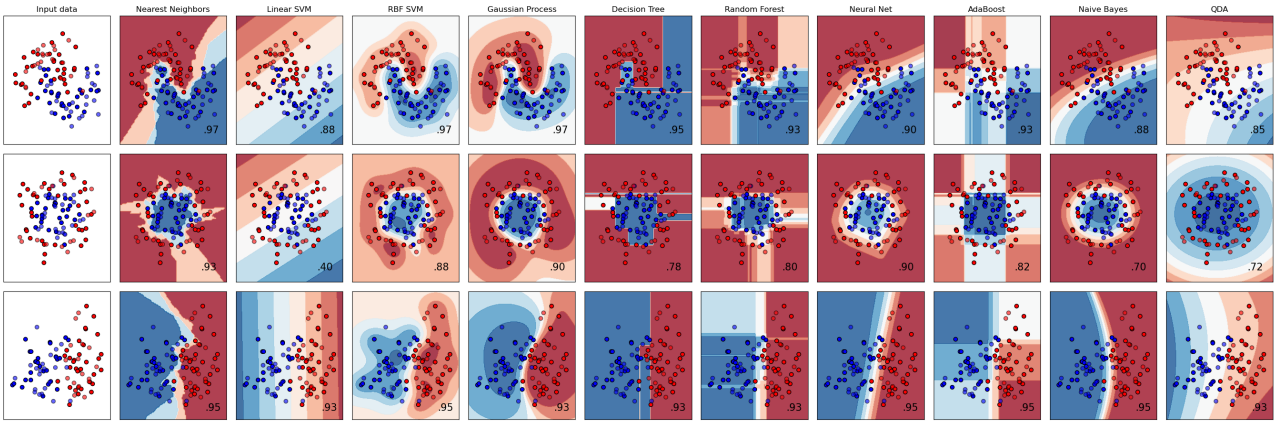


Figure 3.5: *Comparison of 10 algorithms on three two-dimensional datasets with different intrinsic properties*

[3]

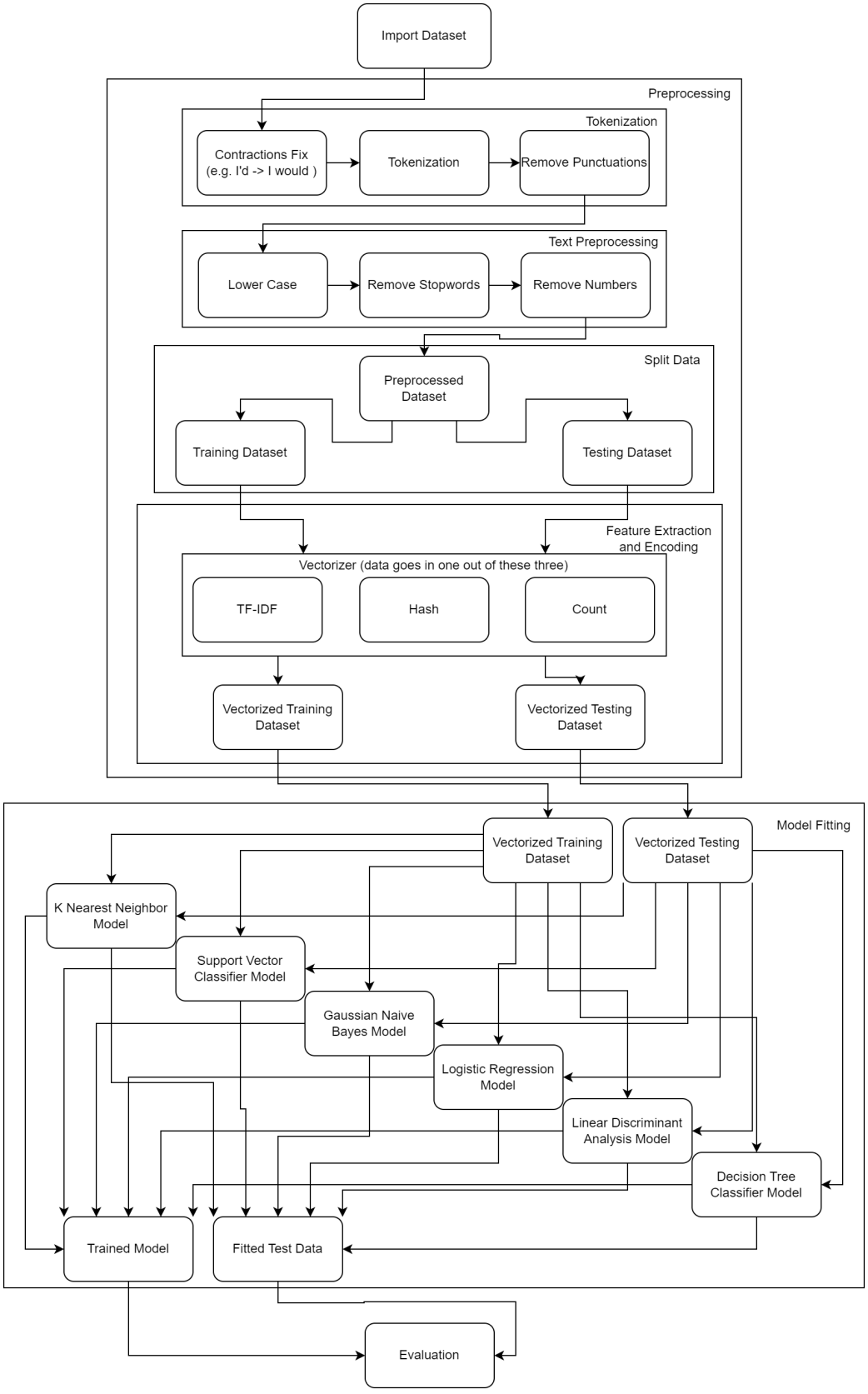


Figure 3.6: *Flowchart of the classical machine learning model design*

3.3 Design – Deep Learning Modeling

In this section I would be designing a few model with slight variations and to test them separately. It would be separated into two main categories, single input and multiple inputs. The multiple inputs are built upon the single input design. Let's begin with the overall architecture. The single input data consists of two elements, ["News Headings : String", "Label : Integer"] and the multiple inputs data consists of three elements, ["News Headings : String", "Variable : Float", "Label : Integer"]. As we see the "News Headings" is going to be our focus for the preprocessing.

The overall structure Figure 3.7 would be separated into a few main parts like in our other design using Classical Machine Learning methods. First, we go through an optional preprocessing part depending on the pre-trained BERT model we choose to use because some are trained on purely lower-cased data datasets so in order to fit our data in we would have to also lower-case our datasets. There is also another optional preprocessing part depending on our Label data. If our Labels consist of more than two different labels we would have to first pass them through a One-Hot encoding process. With categorical data we would have to convert it into a non-biased numeric form because for example with our data, the model might treat each label as a number that could have a numeric meaning and different numbers(labels) might be treated differently.

The second step we would have to split our dataset into three chunks, the training dataset, the testing dataset, and the validation dataset. The reason why we need a training dataset and testing dataset is just the same as before. The validation dataset on the other hand is used for the fitting stage of our model with the goal of finding and optimizing the best model.

The third step we would build our deep learning model. Despite of the variations, all our models would contain a few key components. First, the "News Headings" data would go through a BERT preprocessing layer where it would be preprocessed and fitted into a shape of three outputs "input_word_ids" "input_mask" "input_type_ids" so that it can be then passed into the BERT encoding layer. The BERT encoding layer would process the three inputs and give out also three different outputs, "encoder_outputs" "sequence_output" "pooled_output". Here we would only need the "pooled_out" output. Then with this output, we can modify our other hidden layers however we want. But in the end what all different variation models have in common is a dropout layer and a dense layer for its output.

With the model built, step four means we would have to compile and fit/train the model with our training data. For the compiling part we would be using the Adaptive Moment Estimation Optimizer, aka Adam, for the optimization process. The optimization function is important because it is the function that makes changes to our weights and learning rate which decides how well our model will fit. Because the BERT model is trained using the Adam optimizer and this is why we are using the same. We would also have to choose our loss and accuracy metrics. For the binary labeling label datasets, we would be using the Binary Crossentropy loss function and Binary Accuracy metrics and for the other labeling methods, we would be using the Categorical Crossentrophy loss function and Categorical Accuracy metrics. With these prepared, we can compile the model.

Having built our model and data prepared, we can finally fit and train our model, step five. Machine learning toolkits like TensorFlow made it really easy for us. But we also would have to choose some options such as epochs, batch sizes and etc. We will be running 70 epochs for each of our models as default but we are going to add an early stopping callback function to it so that we don't over-train our model and stop the epochs early when needed. We will also have to decide our batch size. Batch size is the number of samples processed before the model is updated. Our batch size affects how detailly our model can be trained and also the efficiency of the training process.

Before our final step, we would also have to make a function that graphs our training process so it can be used for finetuning our future models and using the data for evaluation. Last but not least our program would have to test our model. Once the model is finally done training we can feed our test data into our trained model. This should give us a loss and accuracy rating on our model. To further evaluate, we would be passing in our same dataset into a pre-trained news sentiment data model to get its output and pass it into our model for a second evaluation as well to see how well it does compare to other pre-trained models.

3.3.1 BERT Encoding

Bidirectional Encoder Representations from Transformers, aka BERT. Because of the architecture of this model, it allows the model to capture both the left and right context of a given token in a sequence. This also means it can take the entire context of the input text into account to produce an accurate representation of the whole meaning of the text. In our program, we will only need the "pooled_out" because this output contains the fixed-sized vectorized contextualized embedded results of the summary of the input text. In sort this is embeds the context of the texts into vectors so we can use it to fit into our deep learning model.

3.3.2 Single Input Design

This model is the design that everything else was based on Figure B.1. This design is pretty straightforward. Just like we mentioned before, the text data first goes through the text_input(InputLayer) then it gets passed onto the preprocessing (KerasLayer) then it passes the preprocessed data into the BERT_encoder (KerasLayer) and we would connect it to a dropout (DropoutLayer) so we can prevent overtraining our model and then we have a classifier (DenseLayer) that manages the output. Note that at the final layer, the output shape is not hardcoded for the output shape. Output shape would be (1,) if the labeling is binary (1 or 0) but the shape would be a number of (n,) because we are using One-Hot encoding for the labeling when there are more than two classifiers so that n would be the number of unique classifiers, in this case, is 3. Table 3.1

Additionally, we would have the option to add a SoftMax layer depending on if the model is saved for other uses. If not, then we do not need this layer because in our loss function, there is a parameter from_logits that we would set to True. When this parameter is set to True a softmax activation function would be automatically added by the loss function. So, unless we have other purposes for our model where we need the normalized output, this would be optional.

To further build on design B.1, we will be building another model B.1 that contains extra dense layers to hopefully train for better accuracy. We would be adding an extra dense layer and an extra dropout layer to increase the total parameters in the model so it can hopefully fit better. As you can see in Table B.1, the hidden layer I added contains 20 neurons. This is because "Generally speaking, the larger the internal structure of a neural network is, the more data points it needs to train. A general principle is to have about ten times as many data points as there is the total number of internal parameters." - [12]. We have a total of 1987x25, 49675 datapoints and ten percent of that equals 4967.5 parameters. Because the Bert_encoder output shape is (256,), this is divided by 4967.5 which would give us 19.4 and after rounding up we would end up with 20 neurons.

We would also be building another version addition on the base model for an experiment. Instead of adding hidden layers, we would be adding a Convolution Neural Network before to attempt to capture the relations in the BERT_encoding output better to hopefully improve our

Single Input Classification Model			
Layer (type)	Output Shape	Param	Connected to
text_input (InputLayer)	[(None,)]	0	[]
preprocessing (KerasLayer)	{'input_word_ids': (None, 128), 'input_type_ids': (None, 128), 'input_mask': (None, 128)}	0	['text_input[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 256), 'pooled_output': (None, 256), 'encoder_outputs': [(None, 128, 256), (None, 128, 256), (None, 128, 256), (None, 128, 256)], 'default': (None, 256)}	11170561	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
dropout (Dropout)	(None, 256)	0	['BERT_encoder[0][5]']
classifier (Dense)	(None, *1 for binary classification and n for non-binary classification*)	(output shape times previous output shape)	['dropout[0][0]']

Table 3.1: *Single Input Classification Model*

classification accuracy. This would add three extra layers to the model (Table B.1), a Reshaping layer, a Conv1D layer, and a Max Pooling layer. The Reshaping layer is needed because we have to shape our data so that it is compatible with the Conv1D layer. In this design the Conv1D layer would have 32 filters and a kernel size of 3, then after this, the Max Pooling layer would reduce the dimensionality of the output.

In theory this version should not be needed because the output of the "pooled_output" from the BERT_encoding layer can technically be seen as an embedding of the context but we might still be able to find patterns within the output to get a better prediction outcome.

3.3.3 Multiple Inputs Design

The reason to add an extra input to the model is for the purpose of exploring how different types of additional data would affect the accuracy and performance of our model also with the goal of finding out the relationship between the text data and the additional data. The type of data we would be experimenting with includes "Volume" and "Change of Volume" that we have calculated and gathered in section 3.1.3

The multiple inputs design is also built upon the single input design. As we mentioned in the previous section, we would maintain everything from the text input to the output of the BERT_encoder layer. This is necessary because this part of the model handles the encoding of the text which is how we are going to map our text to our labels. To make the model more flexible we would be setting the input data type to float32 so that it can accept a wider range of data including integers and non-integer rational numbers.

$$Second\_Input = \{x : x \in Q\}$$

Further built on the two single input model designs, we would be introducing two extra layers. Of course, the first one would be the input layer for our numerical input and with this data, we would have to then add a concatenate layer that goes after the ["pooled_output"] from the BERT_encoder layer (compare FigureB.1 and Figure B.2). The reason why we have to do this is that, unlike a typical neural network model where we can simply increase the input shape of the input later when we want to add an additional vector, we do not want our Second_Input to be passed through the BERT_preprocessing and BERT_encoding layers. Those two layers are designed to only take in a plain text input of shape (1,) therefore we have to get our other data to bypass these two layers.

Multiple Input Classification Model			
Layer (type)	Output Shape	Param	Connected to
text_input (InputLayer)	[(None,)]	0	[]
preprocessing (KerasLayer)	{'input_word_ids': (None, 128), 'input_type_ids': (None, 128), 'input_mask': (None, 128)}	0	['text_input[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 256), 'pooled_output': (None, 256), 'encoder_outputs': [(None, 128, 256), (None, 128, 256), (None, 128, 256), (None, 128, 256)], 'default': (None, 256)}	11170561	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
number_input (InputLayer)	[(None,1)]	0	[]
concatenate (Concatenate)	[(None,257)]	0	['BERT_encoder[0][5]', 'number_input[0][0]]
dropout_1 (Dropout)	(None, 128)	0	['hidden_layer [0][0]']
classifier (Dense)	(None, *1 for binary classification and n for non-binary classification*)	(output shape times previous output shape)	['dropout_1[0][0]']

Table 3.2: *Multiple Inputs Classification Model*

This applies to our other two versions of the model which include an extra Dense Layer Figure B.2 and the other experimental one that also goes through a Convolution Neural Network Figure B.2. The optional softmax layer we mentioned in the single input design stays true.

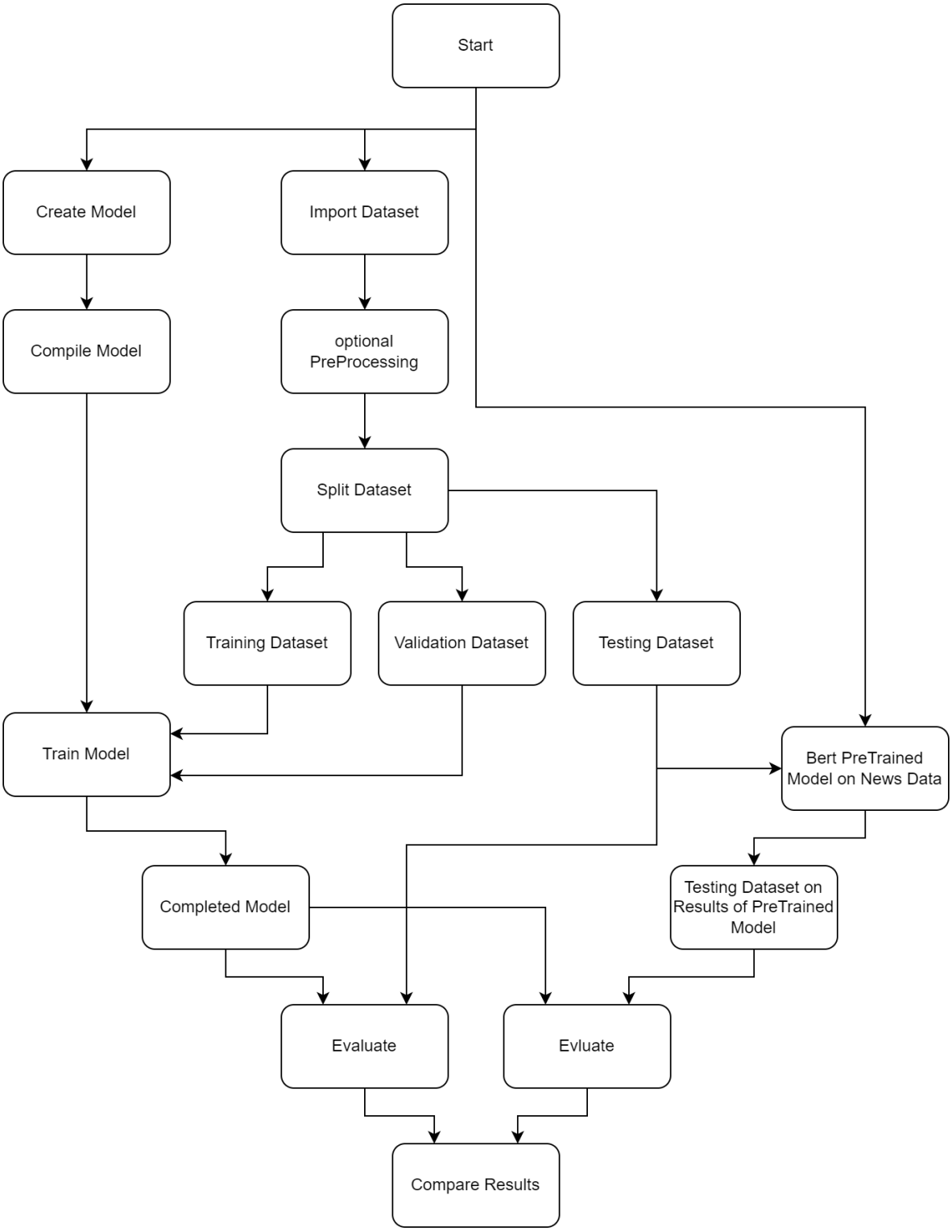


Figure 3.7: *Overall design of the program for building, testing, and evaluating the model*

Chapter 4

Implementation

This chapter outlines the core implementation elements that were coded and used for the purpose of this research. I would be discussing the details of the three programs I created and how the data is processed and used between each of them. All the implementations are done using Python.

4.1 Data Preparation Program

The main goal of this program is to combine our existing news dataset 3.1 that is stored in a csv file and combine it with the according stock data we have chosen and downloaded from yahoo finance. Then labeled with our chosen methodology. Finally with the goal of producing a labeled news dataset as a form of a csv file. Please refer to Figure 3.4 for the flow of our implementation.

4.1.1 Libraries

We would begin with importing our libraries but we are not going to include them here as it is not crucial to our implementation.

4.1.2 Importing News Datasets

This part is fairly straightforward, we would first have to read our data from the "News_Dataset.csv" file and store it in a pandas.Dataframe.

Because some of the news data gathered inside the dataset were processed as a byte() object and some are simply stored as a String() object but all stored simply as plain text, after reading the file using pd.read_csv(), they are all treated as String() objects and we could not use the bytes.decode() function to remove the byte representation characters in python, therefore we would have to manually remove them from the string. This is not a crucial part of our program therefore we would not be including the code snippet.

4.1.3 Downloading Financial data

Even after the official Yahoo Finance API was shut down by Yahoo in 2017 due to widespread abuse against their terms of service, some of the unofficial libraries still remain available on the internet and they are also easy to use.

First, we would have to select our target stock symbol, here we have chosen the one for SP500 and we would also have to choose our data's interval. We would then be retrieving our data from the start date till the end date of the data available on our news dataset.

```
# Initalize data
stock = yf.Ticker("^GSPC")
dayInterval = 1 #days. Valid intervals: 1m,2m,5m,15m,30m,60m,90m,1h,1d,5d,1wk,
                                        1mo,3mo
start_date = dataset.iloc[0][0]
end_date = dataset.iloc[dataset["Date"].size-1][0]
# Get historical market data
stockdata = stock.history(start= start_date, end= end_date, interval=(str(
                                        dayInterval)+'d'))
```

4.1.4 Merge Data

With both our datasets prepared, we would have to merge them into one. Because the Date representation is in a different format inside "stockdata", we would first have to match the format for both datasets and then preform an Inner-Join.

```
# Inner Join
datajoinned = pd.merge(dataset, stockdata, how='inner', on='Date')
```

4.1.5 Labeling And Extra Calculated Data

Over the course of my implementation, I have designed a few ways of labeling methods for experiments. Here I would first be covering the Labeling and I would cover the additional data I have calculated and used for our multiple input deep learning model.

Binary Labeling Method I

This is the simplest method out of all three methods, all we have to do is compare the opening price with the closing price for each time interval and label it 1 if the closing price is higher or equal to the opening price and label it 0 if it is lower.

```
label = []
openPrices = datajoinned["Open"]
closePrices = datajoinned["Close"]
for i in range(openPrices.size):
    if closePrices[i] >= openPrices[i]:
        label.append(1)
    else:
        label.append(0)
datajoinned["Label"] = label
```

Ternary Labeling Method I

First method out of our two ternary labeling methods. We first calculated the percentage changes in the prices and then find out which percentages would be our upper and lower limits for splitting the dataset into three parts. Then we compare the percentage change for each time interval and label them accordingly.

```
label, percentageChanges = [], []
openPrice = datajoinned["Open"]
closePrice = datajoinned["Close"]
# Calculate percentage changes
for i in range(openPrice.size):
    priceChange = closePrice[i] - openPrice[i]
    percentageChange = priceChange / openPrice[i]
    percentageChanges.append((percentageChange*100))
# Find splitting points
acceptable_max = sorted(percentageChanges, reverse=True)[math.ceil(len(
                                        percentageChanges) * (1/3))]
acceptable_min = sorted(percentageChanges, reverse=True)[math.ceil(len(
                                        percentageChanges) * (2/3))]
# Labeling
for i in range(openPrice.size):
    if percentageChanges[i] > acceptable_max:
        label.append(1)
    elif percentageChanges[i] < acceptable_min:
        label.append(-1)
    else:
        label.append(0)
datajoinned["Label_I"] = label
```

Ternary Labeling Method II

The second ternary labeling method's goal is to label the data with the assumption that the news data affects the price momentum in the next N trading days. We would first have to retrieve the same stock data but with a buffer in future prices. Here we would retrieve two extra weeks of market open trading day data because the US stock markets are closed on weekends and US public holidays or special events but 14 days should be enough.

```
later_stockdata = stockdata + stockdata_14_days_after_end_date
```

Now we would find out the days of momentum increase and days of momentum decrease within each [x_day, x_day + 1, ... ,x_day + N] and label each data accordingly.

```
days_period = 5
label_new = []
for i in range(openPrice.size):
    current_chunk = later_stockdata.iloc[i:days_period+i]
    closing_prices = current_chunk["Close"]
    days_of_increase, days_of_decrease = 0, 0
    for j in range(1, closing_prices.size):
        if closing_prices[j] > closing_prices[j-1]:
            days_of_increase += 1
        elif closing_prices[j] < closing_prices[j-1]:
            days_of_decrease += 1
        else:
            days_of_increase += 1
            days_of_decrease += 1
    if days_of_increase > days_of_decrease:
        label_new.append(1)
    elif days_of_increase < days_of_decrease:
        label_new.append(-1)
```

```
    else:
        label_new.append(0)
datajoinned["Label_II"] = label_new
```

Extra Label: Volume Change

This extra label calculates the change in volume in the next N trading days with the hope of associating with Ternary Labeling Method II. We would be using the same buffered price dataset we have retrieved earlier and the same N trading days. We would go through each data point we have and find out the total volume change in each N days period.

```
change_in_vol = []
for i in range(openPrice.size):
    current_chunk = later_stockdata.iloc[i:days_period+i]
    vol = current_chunk["Volume"]
    change = 0
    for j in range(1, vol.size):
        change += vol[j] - vol[j-1]
    change_in_vol.append(change/vol.size)
datajoinned["Vol_change"] = change_in_vol
```

4.1.6 Output

Of course last but not least, we need an output. We would be saving our combined dataset into a csv file which can then be accessed by our Machine Learning and Deep Learning models.

```
datajoinned.to_csv("Processed_Dataset.csv")
```

See Figure 3.3 for a snapshot of the final output.

4.2 Classical Machine Learning Program

The main goal of this program is to find out how compatible our dataset is with our chosen machine-learning algorithms. The program should follow the steps that are shown in Figure 3.6.

I have referenced part of [10] for the implementation of this program.

4.2.1 Libraries

Of course, we would have to import our libraries. We are not going to include the code here as it is not crucial to our implementation.

4.2.2 Load Dataset

Now we would have to load the Dataset that we got from our Data Preparation Program in section 4.1.6. And for the text processing, we would have to extract the columns that contain the news data. Code snippet not include same as before.

4.2.3 Preprocessing

This part covers the parts where we preprocess our text data so our data can be less noisy and make the fitting of the model better. As seen in Figure 3.6, this would include both Tokenization and Preprocessing modules. In the Tokenization module, theoretically, the Contraction Fix function should be done inside the Preprocessing module but due to the nature of the functions we used it would be easier to be done before. The contraction fix function takes in plain text and fixes all the contractions inside that string therefore it would be more efficient to pass in the whole string instead of tokenizing them and then combining them one by one again. Same for the Punctuation removal module, but because the tokenizer function we are using has the option to remove punctuations while tokenizing, therefore we grouped them together.

Tokenization Module

Inside this module, we would be performing the contraction reverse, tokenization and punctuation removal all together. The contraction removal function turns the contracted words such as "I'm" back into "I am" so that we can reduce tokens with duplicated meanings.

```
for index, rows in news.iterrows():
    # removes contractions eg. "I'd like to" -> "I would like to"
    rows = rows.apply(contractions.fix)
    # removes punctuations as well during tokenization
    tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
    tokenized_news.append(rows.apply(tokenizer.tokenize))
news = pd.DataFrame(tokenized_news)
```

Preprocessing Module

As we mentioned above, we have already done some of the common preprocessing practices but there is more we can do to reduce the time, resources for training the model while increasing accuracy. Here we would also be lowering the cases for our tokens, removing stopword token s("the", "is", "and" & etc) and numeric tokens. And also because of the machine learning library (sklearn) we are using, their vectorizer functions takes in an array of string that each element represents as its own training data as input instead of a 2Darray of strings(tokens) so we would be rejoinning the tokens back into a string format and stick each day's data together as one element.

```
def textCleaning(sentence):
        #transform lowercase
        sentence = [x.lower() for x in sentence]
        #remove stopword
        sentence = [word for word in sentence if word not in stop_words]
        #remove numbers
        sentence = [word for word in sentence if not word.isdigit()]
    return sentence
headlines = []
for index, rows in news.iterrows():
    #tokenized version
    news.iloc[index] = rows.apply(textCleaning)
    #un-tokenized and combined version
    headlines.append(' '.join((news.iloc[index].apply(' '.join)).tolist()))
pd_headlines = pd.DataFrame(headlines)
```

For comparison, this is how the original text looks like:

```
[day1["news1","news2", ... ,"newsN"],
 day2["news1","news2", ... ,"newsN"],
 ... ,
 dayN["news1","news2", ... ,"newsN"]]
```

This is an example of what the preprocessed tokenized text looks like:

```
[day1[["token1", ... ,"tokenN"], ... ,["token1", ... ,"tokenN"]],
 day2[["token1", ... ,"tokenN"], ... ,["token1", ... ,"tokenN"]],
 ... ,
 dayN[["token1", ... ,"tokenN"], ... ,["token1", ... ,"tokenN"]]]
```

And this is now what the preprocessed un-tokenized text looks like:

```
[day1["new1 news2 news3 ... newsN"],
 day2["new1 news2 news3 ... newsN"],
 ... ,
 dayN["new1 news2 news3 ... newsN"]]
```

Splitting Training And Test Data

With our data fully preprocessed, we would have to split our dataset into Training data and Testing data. In this step, we would end up with four lists, two for the text data and labels used in training and another two for the text data and labels used in testing. 80% for our training data and 20% for our testing data.

Feature Extraction And Encoding

In this part, we should be performing our feature extractions and encoding. But for better efficiency, most of the code that performs the processing would be done in section 4.2.4, the code in this part would be for creating our vectorizer objects. In both our TfidfVectorizer and CountVectorizer we would be setting our n-gram range to (3,3) which extracts 3-gram features.

```
tfidf_vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(analyzer='
                                        word', ngram_range=(3,3))
hash_vectorizer = sklearn.feature_extraction.text.HashingVectorizer(n_features=
                                        (2**18))
count_vectorizer = sklearn.feature_extraction.text.CountVectorizer(analyzer='
                                        word', ngram_range=(3,3))
```

With our vectorizer objects created, we can use them to perform feature extractions and encoding so that they can be passed into our classifiers. You will be seeing the code we mentioned here again later on below as these are just snapshots of code that belongs to this section but are called inside section **??** for efficiency purposes.

```
traindataset = vectorizer.fit_transform(train_data)
test_results = vectorizer.transform(test_data)
```

4.2.4 Modeling

Now with all everything prepared, we would have to train our models and see how well our models and dataset work. We would be testing each of our feature extraction and encoding methods

against five of our classifiers. We are going to call the function train_test_model() while passing in our vectorizer object to test against all five statistical models in the function in section 4.2.4.

```
train_test_model(tfidf_vectorizer)
train_test_model(count_vectorizer)
train_test_model(hash_vectorizer)
```

Model Training, Training and Evaluation

The code shown in this part is the second main part of our program. The function shown below handles all the training and testing for each feature extraction and embedding technique against all five of our statistical models. It first does what we mentioned in section 4.2.3, then it would fit our data to our k-nearest neighbor, support vector classifier, logistic regression, gaussian naive bayes, decision tree classifier models and evaluates them one by one. We can then access the evaluations via the terminal output.

```
def train_test_model(vectorizer):
    # Feature extraction and encoding
    traindataset = vectorizer.fit_transform(train_data)
    test_results = vectorizer.transform(test_data)

    # training
    classifier = sklearn.one_of_our_classifiers()
    classifier.fit(traindataset, train_labels)
    # testing
    predictions = classifier.predict(test_results)
    evaluate_results(predictions)

    # repeat the above for all our classifiers
    # ...

def evaluate_results(predictions):
    print("Score: ", sklearn.metrics.accuracy_score(test_labels, predictions))
    print("Report: ", sklearn.metrics.classification_report(test_labels,
                                                predictions))
```

4.3 Deep Learning Program

This section covers our second implementation using Deep Learning techniques. As seen in Figure 2.1 & Figure 2.2, usually deep learning-based NLP programs do not require preprocessing techniques as the neural network automatically detects the importance of each word thus we do not have to go through techniques such as lowercasing, stopword removal, number removal and such. Matter of fact these techniques might even reduce the accuracy of our model because they would remove words and changes the context of our text data. Unlike our classical machine learning implementation, we do not have to go through the Feature Extraction process. Although we would be able to keep the contraction fix function since it does not change any of our data contexts. Because of this, we would be going through a very similar preprocessing stage to the previous implementation but other than this everything would be different. Please refer to Figure 3.7 for the flow of this program.

I have referenced part of [4] and [2] for the implementation of this program.

4.3.1 Imports And Variable Initialization

In order to get our program to work we would of course have to start with importing and downloading the resources we would be using. We are also not goin to include them here as they are not crucial to be mentioned.

Because our program would be testing against different variables, we would have to initialize them before our program starts. "tfhub_handle_encoder" and "tfhub_handle_preprocess" are the of where our BERT preprocessing and encoding models are stored on the Tensorflow Hub server [19]. "file_name" would of course be our dataset's file name, "model_folder_settings" is where we would be saving our models after training. "batch_size_settings" ,"epochs_settings" and "dropout_settings" are some of the variables we would be using for our training. "first_input_settings", "second_input_settings", "label_settings" are the name of the columns we would be extracted from our dataset and used for our training. The rest of the variables are all used simply to adjust the program for each type of data being used.

```
tfhub_handle_encoder = 'https://tfhub.dev/tensorflow/small_bert/
                                    bert_en_uncased_L-4_H-256_A-4/2'
tfhub_handle_preprocess = 'https://tfhub.dev/tensorflow/
                                    bert_en_uncased_preprocess/3'
file_name = "Processed_Dataset"
batch_size_settings = 32
epochs_settings = 70
dropout_settings = 0.5
first_input_settings = 'News'
second_input_settings = 'Vol_change' # or 'Volume'
label_settings = 'Label' #or 'Label_I' or 'Label_II'
```

4.3.2 Import Dataset and Preprocessing

Of course just as before we would have to read our dataset created in section 4.1.6 and because we are only preprocessing with the text data we would have to extract the news first and process them through the reverse_contraction() function. There is also no need for tokenization because our BERT_preprocess layer takes in plain text as input instead of tokens.

```
news = reverse_contraction(news)
```

After the text preprocessing stage, we would have to combine them with our labels and second input data into one panda.Dataframe object so that it makes it easier for us to process. Instead of treating all our news data from the same day as one data entry point as we did in section 4.2.3, because all our news data are unrelated sentences, passing all of them into our BERT layers as one would trick the pre-trained model into thinking these are all the same event. Therefore here we would have to transform them from a shape of (((News * 25), Second_Input, Label) * NumOfRows) into a shape of ((News, Second_Input, Label) * 25 * NumOfRows).

```
for index, rows in news.iterrows():
    # Transform the 25xN table into a 1x25N table
    for element in rows: # each element is one news title
        combined_news.append([element, volume, label])
combined_news_df = pd.DataFrame(...)
```

4.3.3 Dataset Splitting

Similarly to section 4.2.3, we would have to split our dataset into not only two parts, the training dataset and testing dataset, but rather three parts including the validation dataset. Here we are splitting our dataset in an 80:10:10 portion. We are also going to preform one-hot encoding on our labels. The first reason behind this is to allow our model to take in any kind of label so that it is more versatile for our given dataset. Second, instead of simply using label encoding (which is already the labeling format in our dataset), could create biases where it might trick the model into thinking some of the labels are larger or smaller than others whereas one-hot encoding could eliminate this problem and because of the number of unique labels we have is small so the problems due to the increasing size of the labeling is negligible. We would also shuffle our data here in this function in order to reduce variance and keep the data general and not overfit as much.

```
def split_data_fromDataframe_multiclass(pd_dataframe):
    # shuffle dataset
    pd_dataframe = shuffleDataframe(pd_dataframe)
    # One-Hot encode our labels
    label_categorical = tf.keras.utils.to_categorical(pd_dataframe[
                                            label_settings].values, num_classes
                                            =num_classes)
    # Split data into 80\% training data, 10\%validation data, 10\% testing
                                            data
    ...
    return text_train, label_train, text_test, label_test, text_val, label_val,
                                             volume_train, volume_test,
                                            volume_val
```

4.3.4 Modeling

This is the part where we construct our deep-learning models. We will first be covering our single input models first then we can go on with our multi inputs models after. All the models are built upon Single Input Model Original in section 4.3.4.

Single Input Model Original

This is our very first model which all the other models below are based on. We first start off with creating our text input layer. Then we would add our BERT_preprocessing layer to the input layer and the BERT_encoder layer after. We then only connect the "pooled_output" to our dropout layer and last but not least our dense layer which is our final output. Of course at the end of the function we would return a tf.keras.Model object with the expected input layer and output layer as the argument for the Model() function argument.

To visualize our model here please refer to Figure B.1

```
def build_model_multi_labels():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='
                                            text_input')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='
                                            preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='
                                            BERT_encoder')
    outputs = encoder(encoder_inputs)
```

```
    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(dropout_settings)(net)
    net = tf.keras.layers.Dense(num_classes, activation=None, name='classifier'
                                                )(net)
    return tf.keras.Model(text_input, net)
```

Single Input Model Extra Hidden Layers

This model is an extra built-on to our previous model in order to achieve higher accuracy. The difference is that it has one extra dropout layers and another extra dense layer with 20 neurons.

To visualize our model here please refer to Figure B.1

```
def build_model_multi_labels_with_extra_hidden():
    # same as above
    ...
    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(dropout_settings)(net)
    net = tf.keras.layers.Dense(20, activation=None, name='hidden_layer_1')(net
                                                )
    net = tf.keras.layers.Dropout(dropout_settings)(net)
    ...
    # same as above
```

Single Input Model Experimental CNN Classifier

This model is another extra built-on to our first model in order to achieve higher accuracy as well. The difference is the CNN layer before the final dense and dropout layers. As you see we have an extra Reshape layer where it reshapes the output of the BERT_encoder layer into the shape of the Conv1D layer and the GlobalMaxPoolin1D layer reduces the dimension of the output of the CNN layers.

To visualize our model here please refer to Figure B.1

```
def build_model_multi_labels_cnn():
    # same as above
    ...
    net = outputs['pooled_output']
    net = tf.keras.layers.Reshape((-1, 1))(net)
    net = tf.keras.layers.Conv1D(32, 3, activation='relu')(net)
    net = tf.keras.layers.GlobalMaxPooling1D()(net)
    net = tf.keras.layers.Dropout(dropout_settings)(net)
    ...
    # same as above
```

Multiple Inputs Model Original

This is our second main model, designed also based on our first model. Looks similar but fairly different. We first build the model like how we did for our Single Input Model but we have an extra Input layer for our numerical value inputs. We then have a Concatenate layer that combines our "pooled_output" with our second input. The rest is the same.

To visualize our model here please refer to Figure B.2

```
def build_model_multi_labels_two_inputs():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='
                                                text_input')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='
                                                preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='
                                                BERT_encoder')
    outputs = encoder(encoder_inputs)
    text = outputs['pooled_output']
    number = tf.keras.layers.Input(shape=(1,), dtype=tf.float32, name='
                                                number_input')
    net = tf.keras.layers.concatenate([text, number], axis=1)
    net = tf.keras.layers.Dropout(dropout_settings)(net)
    net = tf.keras.layers.Dense(num_classes, activation=None, name='classifier'
                                                )(net)
    return tf.keras.Model([text_input, number], net)
model = build_model_multi_labels_two_inputs()
```

Multiple Inputs Model Extra Hidden Layers

This model is basically our Multiple Inputs Model Original in section 4.3.4 with the improvements from our Single Input Model Extra Hidden Layers model in section 4.3.4. We added a Dense layer with 128 neurons and a relu activation function to this layer for experiments to see if we can further improve our models.

To visualize our model here please refer to Figure B.2

```
    # same as above
    ...
    text = outputs['pooled_output']
    number = tf.keras.layers.Input(shape=(1,), dtype=tf.float32, name='
                                                number_input')
    net = tf.keras.layers.concatenate([text, number], axis=1)
    net = tf.keras.layers.Dense(128, activation='relu', name='hidden_layer')(
                                                net)
    net = tf.keras.layers.Dropout(dropout_settings)(net)
    ...
    # same as above
```

Multiple Inputs Model Experimental CNN Classifier

Same as above but improvements from the changes in section 4.3.4.

To visualize our model here please refer to Figure B.2

```
    # same as above
    ...
    text = outputs['pooled_output']
    number = tf.keras.layers.Input(shape=(1,), dtype=tf.float32, name='
                                                number_input')
    net = tf.keras.layers.concatenate([text, number], axis=1)
    net = tf.keras.layers.Reshape((-1, 1))(net)
    net = tf.keras.layers.Conv1D(32, 3, activation='relu')(net)
    net = tf.keras.layers.GlobalMaxPooling1D()(net)
```

```
    net = tf.keras.layers.Dropout(dropout_settings)(net)
    ...
    # same as above
```

4.3.5 Training

This part of the code covers what we do for setting up our loss and accuracy metric functions, choosing an optimizer, and what we did to optimize our training time and prevent overtraining. In order to utilize multiple GPUs to speed up our training, we would also have to create a MirroredStrategy object. All the code in the next three sub-sections would be grouped together as they all have to be done within the scope of the MirroredStrategy's scope.

Loss and Accuracy

Because of the wide range of data we have on our hand that we can use for our training, our loss and accuracy metrics has to be able to cover both our binary labeling data and non-binary labeling data. Due to this fact, we would be using CategoricalCrossentropy() for our loss metrics and CategoricalAccuracy() for our accuracy score. So why aren't we using other ones such as BinaryCrossentropy() or SparseCategoricalCrossentropy()? First, two of our labeling methods include three unique labels and BinaryCrossentropy() is only designed for two. Second, because both BinaryCrossentropy() and SparseCategoricalCrossentropy() only accepts label encoding, as explained in section 4.3.3, our label are a One-Hot encoded. We also have to set the parameter from_logits to True so that the loss function would handle our output normalization automatically.

We have also coded a function where it handles whether the data we have depends on if we are feeding the model single or multiple inputs. Code is not included here as it is not crucial.

Optimizer

We also have to set up our optimizer for our model. Because the BERT encoder is trained using the Adam optimizer, we would be using the same one here as well. There are also three other variables we have to initialize, learning rate, number of training steps, and number of warm-up steps. The learning rate is how much the optimizer adjusts our model's parameters and we are setting it to 1e-5 here. The number of training steps refers to the number of updates our optimizer is doing for our training, this is set based on our amount of epochs and batch sizes. The number of warm-up steps to help us make larger updates to its parameters during the early stages of training so that the optimizer converges more at the beginning and does not get stuck at local minima. We would be setting this to a tenth of our total training steps.

Compile Model

With our optimizer, loss, and accuracy metrics set, we would now compile our model.

Here is the code for all subsections above:

```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
    loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
    metrics = tf.metrics.CategoricalAccuracy()
    train_input, test_input, val_input = group_input_data()
    optimization.create_optimizer(init_lr=1e-5,
```

```
                                    num_train_steps=num_train_steps,
                                    num_warmup_steps=10_perent_of_total_steps,
                                    optimizer_type='adamw')
    model = build_model()
    model.compile(optimizer=optimizer,
                  loss=loss,
                  metrics=metrics)
```

Early Stopping Callback

Because we do not know how well our model will train, we usually would have to estimate our number of epochs at the very beginning and as we did earlier we have chosen a maximum of 70 epochs. Sometimes our model would also overtrain when we have left it training for too long. But don't you worry, Tensorflow offers us a way to stop our training early and it is the EarlyStopping callback function. We can set up a callback function that monitors a metric of our choice, "categorical_accuracy" in our case and we can set it so that if it doesn't improve for a certain amount over a certain period of epochs then the function would stop the training earlier to save us time and computational costs. Because Tensorflow has discontinued their updates for windows machines after version 2.10, we would not be able to set up the parameter "start_from_epoch" to ignore the unusual training curve for our warm-up steps, we have set our improvement valid epoch range to a slightly larger value so that it makes sure it would cover even the warm-up epochs. We will also be setting our EarlyStopping function to restore our best weights in the whole course of our training so that we don't have to worry about the overfitting epochs that violate our EarlyStopping settings.

```
tf.keras.callbacks.EarlyStopping(monitor = "categorical_accuracy",
                                 patience = 15,
                                 mode="auto",
                                 restore_best_weights = True,
                                 verbose=1,
                                 min_delta=0.025)
```

Fitting

Finally, we can now start our training for our model. This is a very time and resource-costly process but with the convenience that Tensorflow provides, we can simply pass in the arguments of our settings and call the Model.fit() function to start fitting our training data to our model. We would also be storing our training loss and accuracy metrics in a variable so that we can examine how did our training go which we will talk about in section **??**. We are also going to save a copy of our trained model on our local machine so we can load it up and reuse it when we want to later on.

```
history = model.fit(x=train_input,
                    y=label_train,
                    batch_size = batch_size_settings,
                    shuffle = True,
                    validation_data=(val_input, label_val),
                    epochs=epochs,
                    callbacks=[earlystop_callback])
model.save("model.h5", include_optimizer=False)
```

4.3.6 Evaluating

This is the part where we get the evaluation results of our model. We would first cover how we are going to evaluate our model against our own testing dataset and we would also cover how we can compare our model to existing pre-trained models out there.

Results Against Own Test Data

To do so, it is very easy. We can simply call the Model.evaluate() function while passing our testing input dataset and label dataset and it would return the scores for our chosen loss and accuracy metrics.

```
loss, accuracy = model.evaluate(test_input, label_test)
print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
```

We would also have to create a line chart figure using the matplotlib.pyplot library to visualize our training process just like we mentioned before in our previous section. The code is very redundant and does not affect the performance of our program so we would not be including it here but we will be including some of the chart examples in our appendix. These are mainly used during our implementation so that we can visualize how the training process went.

Results Against Pretrained Model Of Choice

Here we have chosen the "roberta-large-financial-news-sentiment-en" model by Jean-Baptiste on HuggingFace [9]. The reason for our choice is that it is one of the most popular pre-trained models that show similarity to our project. "This model was train on financial_news_sentiment_mixte _with_phrasebank_75 dataset. This is a customized version of the phrasebank dataset in which I kept only sentence validated by at least 75% annotators. In addition I added 2000 articles validated manually on Canadian financial news. Therefore the model is more specifically trained for Canadian news. Final result is f1 score of 93.25% overall and 83.6% on Canadian news.". Although this pre-trained model is not trained for predicting price trends but it is often related to news sentiment.

This model would return three classifiers that indicate "positive", "neutral", and "negative" sentiments separately. We would first retrieve the results of our testing input dataset by passing it into this model, create a one-hot encoded results that match our encoding with our original labelings, and at last find out the loss and accuracy metrics of our model against this newly created test data using the same Model.evaluate() function.

```
def pretrained(dataset):
    pretrained_model_name = "Jean-Baptiste/roberta-large-financial-news-
                                                sentiment-en"
    tokenizer = AutoTokenizer.from_pretrained(pretrained_model_name)
    model = AutoModelForSequenceClassification.from_pretrained(
                                                pretrained_model_name)
    pipe = pipeline("text-classification", model=model, tokenizer=tokenizer)
    results = pipe(dataset.tolist())
    for i in range(len(results)):
        # label map the output to our labels
        ...
    return one_hot_encode(mapped_results)
pretrained_label_test = pretrained(text_test)
```

```
loss, accuracy = model.evaluate(test_input, pretrained_label_test)
print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
```

4.4 Deprecated

This section covers some of the work that has been implemented but eventually not used.

4.4.1 Feature Extraction Using Dependency Parser Or Part Of Speech Tagging

This was originally part of our design and was mentioned slightly to be used as one of our feature extraction methods. The whole idea is that we extract a more meaningful set of context from each sentence. The methodology goes around extracting words in three categories and ignores the rest of the unrelated texts. The three categories include a set of [Actor(Noun), Action(Verb), Subject(Noun)] where an actor performs an action on an object e.g.[”Russia”-¿”Attacks”-¿”Ukraine”], [Actor(Noun), Action(Verb)] where an actor does an action e.g.[”Joe Biden” -¿ ”Resign”], or [Action(Adjective), Subject(Noun)] where it technically can also be classified as the previous category e.g. [”Starving”-¿”Hamster”] = [”Hamster”-¿”Starves”]. We would also handle consecutive nouns as one such as ”Tiger Woods(Noun)” would not be treated as ”Tiger(Noun)” and ”Woods(Noun). The original goal was to extract all the features that match these categories using a part-of-speech tagger and then encode all the data into a fixed-sized vector so that it can be passed onto our classifiers.

```
def pos_and_tokenize(news):
    for index, rows in news.iterrows():
        rows.dropna(inplace=True)
        rows = rows.apply(contractions.fix)
        #removes punctuations as well during tokenization
        tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
        tokenized_news.append(rows.apply(tokenizer.tokenize))
        tokenized_news[index] = tokenized_news[index].apply(nltk.pos_tag)
    news = pd.DataFrame(tokenized_news)
    return news
```

Example results:

```
# Input: Georgia 'downs two Russian warplanes' as countries move to brink of
                                        war
# Output: [('Georgia','NNP'),('downs','NNS'),('two','CD'),('Russian','JJ'),
#          ('warplanes','NNS'),('as','IN'),('countries','NNS'),('move','VBP
#          '),('to','TO'),('brink','VB'),('of','IN'),('war','NN')]
```

Despite the part of speech tagger works for tagging the words but there comes a challenge. It does not indicate which set of words belongs together. For example when a sentence is very long such that [”noun1”, ”verb1”, ”noun2”, ”noun3”, ”noun4, ”verb2”] and it would be difficult to determine if we should extract [”noun1”, ”verb1”, ”noun2 noun3”]&[”noun4, ”verb2] or [”noun1”, ”verb1”]&[”noun2 noun3”, ”noun4, ”verb2] or other combinations. We could be extracting the wrong features and that might affect our feature qualities. Therefore I decided to improve and go a step beyond and implement this feature extraction method with the Standford Core NLP

dependency parser. (The results separate the dependency tree with the word therefore I will be visualizing the results with the www.corenlp.run website [6] that was implemented using the same library)

```
def dependencyParsing_and_tokenize(news):
    dependency, tokenized_news = [], []
    for index, rows in news.iterrows():
        #removes contractions eg. "I'd like to" -> "I would like to"
        rows = rows.apply(contractions.fix)
        dependency.append(rows.apply(textToParse) )
        #removes punctuations as well during tokenization
        tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
        tokenized_news.append(rows.apply(tokenizer.tokenize))
    news = pd.DataFrame(tokenized_news)
    pd_dependency = pd.DataFrame(dependency)
    return news, pd_dependency
```

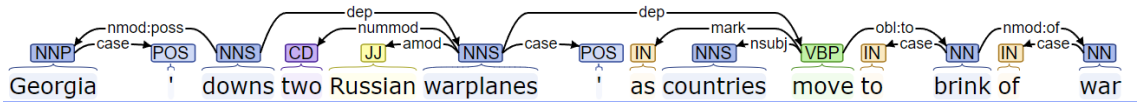


Figure 4.1: *Dependency parsing example on the sentence "Georgia 'downs two Russian warplanes' as countries move to brink of war"*

With all these implemented we face a final challenge. After analyzing the results I realized the majority of the data are labeled with the wrong part of speech tags. This is due to the fact that the part of speech tagger within both functions even takes the capitalization of the words into account. For example "United States" should be tagged as ["United": "NNP", "States": "NNP"] but when the word is lowercased it would be tagged as ["united": "JJ", "states": "NNP"] and this changes the meaning of the phrase entirely. Due to the nature of our data being news headings, in which words are always capitalized without following normal grammar rules, it became a much larger challenge to us than it should have. This is not an entirely unsolvable problem but developing an entirely new dependency parser or part of speech tagger would be out of the bounds of this research and would have to be separated as its own project. Therefore I have decided to move on with other feature extraction techniques with our project.

Chapter 5

Evaluation

In this chapter, we would be covering the evaluation of all the designs and implementations including the results and the data we have retrieved. We would be working backward from the analysis of our results and back toward our methodology and at the end a conclusion.

5.1 Presentation Of Results

In this section, I would first like to cover the results from all our implementation and we would analyze our data and pattern and pass on our conclusion to section 5.2 for further improvements

5.1.1 Classical Machine Learning Methodology Results

Result Gathering Approach

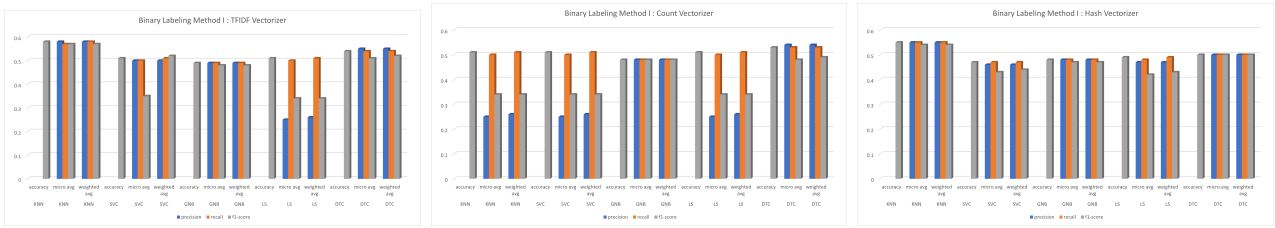
In this section, we would start off with our results gathered from the Classical Machine Learning implementation. We ran a series of models against our data and I would like to first separate it into three categories by its labeling methods.

Each labeling methods contain a few different subcategories that are divided by the Feature Extraction & Encoding method it was used, tfidf_vectorizer, count_vectorizer, and hash_vectorizer. Each of these is then ran against the same five sets of classifier algorithms, K-Nearest Neighbor (KNN), Support Vector Classifier (SVC), Gaussian Naive Bayes (GNB), Logistic Regression (LS), and Decision Tree Classifier (DTC). this gives us a total of 45 combinations of results. Each result contains an f1-score for the result's accuracy, micro average, and weighted average, a recall, and precision scores for only their micro average and weighted average. The f1-sore is a measure of accuracy. The recall score measures how well the model is doing a good job of identifying relevant items in the dataset and the precision score indicates how well the model can correctly label the items as well.

Note: some of the models do not entirely work due to Feature Extraction & Encoding and Classifier incompatibility therefore they cannot predict some categories of labels and therefore their f1-score and precision scores would not be able to be calculated due to zero division error for labels with no samples. These scores are set to 0.0 in these label categories and therefore their final scores would still be in our result datasets despite the model is incomplete. The results of these models are included in the bar charts and if you see an orange bar particularly higher than its neighbor bars then that would be the models we mentioned. Although these are excluded from

the box and whisker charts so that it does not give us unnecessary data for the calculation of their values.

Binary Labeling Method I



(a) (b) (c)

Figure 5.1: *Binary Labeling Method I : (a) Results using TFIDF Vectorizer (b) Results using Count Vectorizer (c) Results using Hash Vectorizer*

Figure 5.1 shows the results for our Binary Labeling Method which accords to the "Label" column in our dataset created in section 4.1.6. As we can see our TFIDF Vectorizer and KNN combination performed the best with an f1-accuracy score of 0.58 and the worse of 0.47 with the HASH Vectorizer and SVC combination. I wouldn't say the (tfidf, knn) combination is particularly high for a classification model but it is one of the best out of all our models. This particular combination also performed fairly well in its other metrics, with a precision score of 0.58 and 0.58 for the micro and weighted average and a recall score of 0.57 and 0.58 for those two metrics as well.

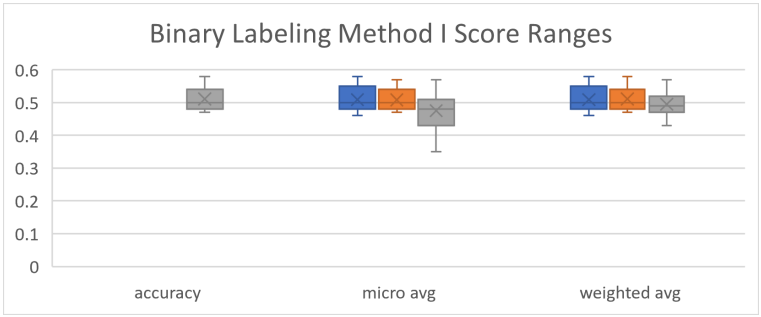
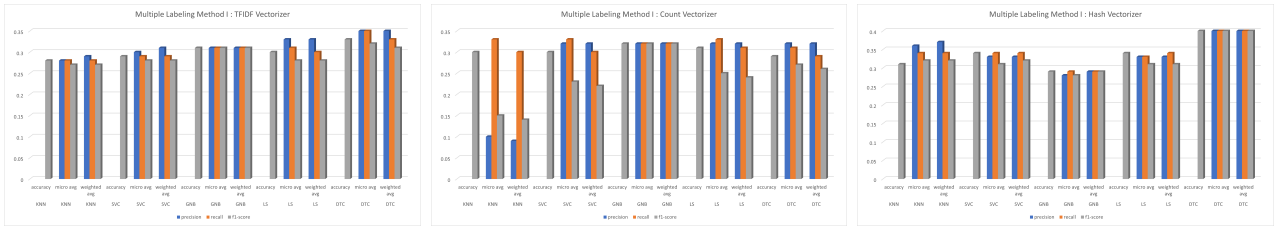


Figure 5.2: *Box and Whisker chart of the ranges of the scores for our models with Binary Labeling Method I*
Gray: F1-score , Orange: Recall score , Blue: Precision score

As shown in Figure 5.2, the range of scores is fairly compact and which shows that the label with Binary Labeling Method I works fairly well with all our combinations in this implementation method.

Ternary Labeling Method I

Figure 5.3 is the results of our Ternary Labeling Method One which accords to the "Label_I" column in our dataset created in section 4.1.6. As we can see our HASH Vectorizer and DTC combination performed the best with an f1-accuracy score of 0.4 and the worse of 0.28 with the



(a) (b) (c)

Figure 5.3: *Ternary Labeling Method I : (a) Results using TFIDF Vectorizer (b) Results using Count Vectorizer (c) Results using Hash Vectorizer*

TFIDF Vectorizer and KNN combination. The (hash, dtc) combination performance is fairly poor and this particular combination also showed disappointing results in its other metrics, with a precision score of 0.4 and 0.4 for the micro and weighted average and a recall score of 0.4 and 0.4 for those two metrics as well.

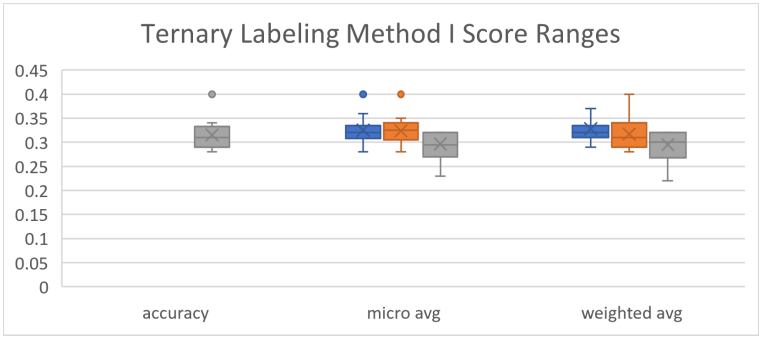
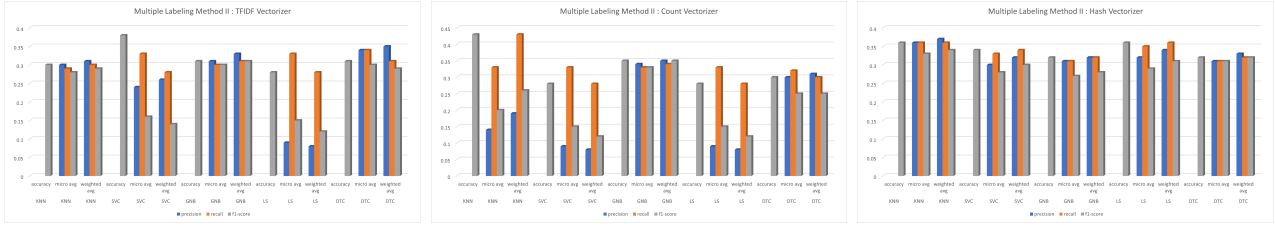


Figure 5.4: *Box and Whisker chart of the ranges of the scores for our models with Ternary Labeling Method I*
Gray: F1-score , Orange: Recall score , Blue: Precision score

As shown in Figure 5.4, the range of scores is fairly compact but it is shown that our best performance combination appears to be an outlier. Excluding this model, we can say Label_I is fairly consistent within our models but its performance is questionable.

Ternary Labeling Method II



(a) (b) (c)

Figure 5.5: *Ternary Labeling Method I : (a) Results using TFIDF Vectorizer (b) Results using Count Vectorizer (c) Results using Hash Vectorizer*

Figure 5.5 is the results of our Ternary Labeling Method Two which accords to the "Label_II" column in our dataset created in section 4.1.6. As we can see our HASH Vectorizer and KNN combination performed the best with an f1-accuracy score of 0.36 and the worse of 0.3 with the COUNT Vectorizer and DTC combination. The (hash, knn) combination performance is fairly poor and this particular combination also showed disappointing results in its other metrics, with a precision score of 0.37 and 0.36 for the micro and weighted average and a recall score of 0.36 and 0.36 for those two metrics as well.

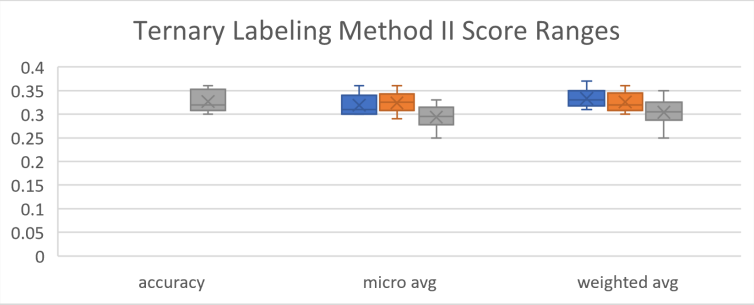


Figure 5.6: *Box and Whisker chart of the ranges of the scores for our models with Ternary Labeling Method I*
Gray: F1-score , Orange: Recall score , Blue: Precision score

As shown in Figure 5.6, the ranges of our data are fairly average but overall perform worse than our Ternary Labeling Method One. This labeling method also can be concluded to be consistent for our models but its reliability is poor.

CML Methodology Result Discussion

The results of the Binary Labeling Method have been shown to be the most effective and best in performance but there is an interesting relation that we are able to spot. When the categories of labels are at a number of two, the average f1 accuracy score is at around one-half and when the labels increase to three, both average f1 accuracy scores decrease to one-third. The f1-score should not have any direct correlation to the labeling. This could be due to a variety of factors, such as an increase in the complexity of the problem, or the model not being able to accurately distinguish between the different categories. To find out whether this was caused by the first or second factor we will need more data and in order to do that we have two options.

First, create another labeling method that has a category number that contains more than three labels, and the labels also have to be extracted from our same dataset with minimum changes and high similarity to our current methods. Since the labeling method changes and become another variable, this would not be able to provide provable answers to our suspicion of the correlation of accuracy_score = 1 / number_of_label_categories.

The second method would be to evaluate our results in our second implementation and this is going to be the approach in this chapter. If the same correlation shows again in the other implementation then we can say that the cause could be caused by the models not being able to distinguish between the different categories and we would have to further investigate the actual cause. But on the other hand, if this does not happen, then we can be sure to say it was our implementation's fault for not being able to handle the increase in complexity of the problem.

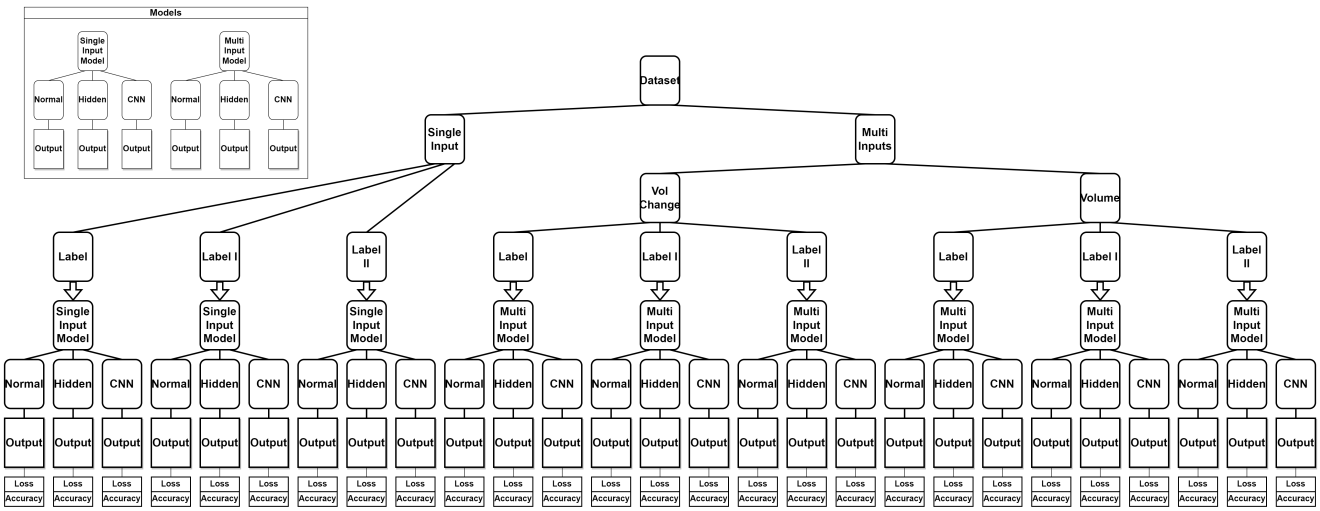


Figure 5.7: *Classification of results for deep learning approach result gathering methodology*

5.1.2 Deep Learning Methodology Results

Result Gathering Approach

In this approach, we have gathered a total of 27 sets of results from a different combination of training input data and models Figure 5.7. We can summarize this into a few different subcategories. First, there are two main designs, Single Input, and Multiple Inputs. Both model types have three different variations, the original model design and two experimental designs with extra hidden layers and parameters or extra convolution neural network layers as the classifier based on the original model. We are going to refer to these as "Normal", "Hidden", and "CNN" for short in this section. For each of the models in the Single Input category, we would be running against three different sets of different training labels "Label", "Label_I", and "Label_II". This gives us a total set of 9 combinations. And as for the Multiple Input Methodology, we would be doing exactly the same but because we have 2 sets of extra training input data this gives us 18 combinations. Both add up to 27 combinations of results. Each result contains 2 values, loss, and accuracy. Additionally, because we are comparing our models that are running against "Label_I" and "Label_II" with a pre-trained model that contains the same amount of labeling categories, this means we would have a total of 45 sets of results and a total of 90 data points.

First, we would be analyzing the accuracy scores of our results by categorizing them by their amount of input types so we can focus patterns with the modeling techniques as a cause. Then with the goal of analyzing patterns from the data labeling methods itself, we would categorize them by their labeling methodology. Accuracy measures the proportion of correct predictions made by the model. It is the ratio of the number of correct predictions to the total number of predictions.

Testing Data From Original Dataset Analysis

As we can see, the data range is at the highest of 0.55 and down to 0.24 at the lowest. We can for sure say it is quite a large range percentage-wise with our best model having a performance twice as good as our worst one. As we can see, it appears that the amount of input does not seem to have an effect on our accuracy. Our best-performing model belongs to the combination of the "Hidden" experimental version of Multiple Inputs with Volume as its extra input parameter and the

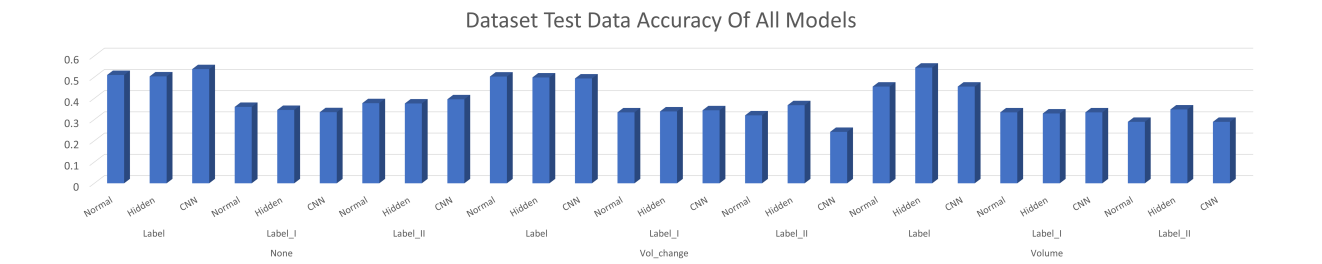
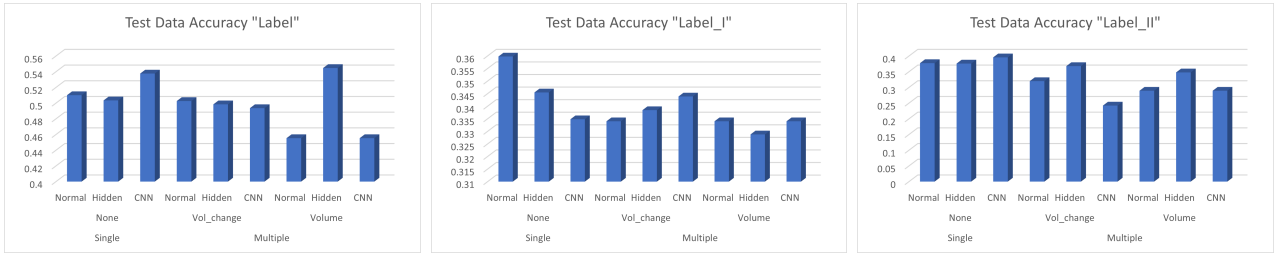


Figure 5.8: *Accuracy scores of all trained models and tested against own testing dataset categorized by input type*



(a) (b) (c)

Figure 5.9: *Accuracy scores of all trained models and tested against own testing dataset categorized by label type*

"Label" labeling method. It also appears this version of the model outperforms the "Normal" and "CNN" versions. This brings up the question of whether the phenomenon "Hidden" performance "Normal" or "CNN" performances, also applies to our other combinations. We can find this out by counting the number of times when one version outperforms the other versions grouped by Input Types and Label Types.

Total Top Performance Count		
Input Type	Label Type	Winner Model
Single	Label	"CNN"
Single	Label_I	"Normal"
Single	Label_II	"CNN"
Multiple Vol_change	Label	"Normal"
Multiple Vol_change	Label_I	"CNN"
Multiple Vol_change	Label_II	"Hidden"
Multiple Volume	Label	"Hidden"
Multiple Volume	Label_I	"Normal"
Multiple Volume	Label_II	"Hidden"
Winning Counts		
"Normal"	3 times	
"Hidden"	3 times	
"CNN"	3 times	

As we can see, all models have separately outperformed the other models 3 times each. This proves that the experiment models in fact did not improve our accuracy rates. Now let's have a look at our data by Label Types.

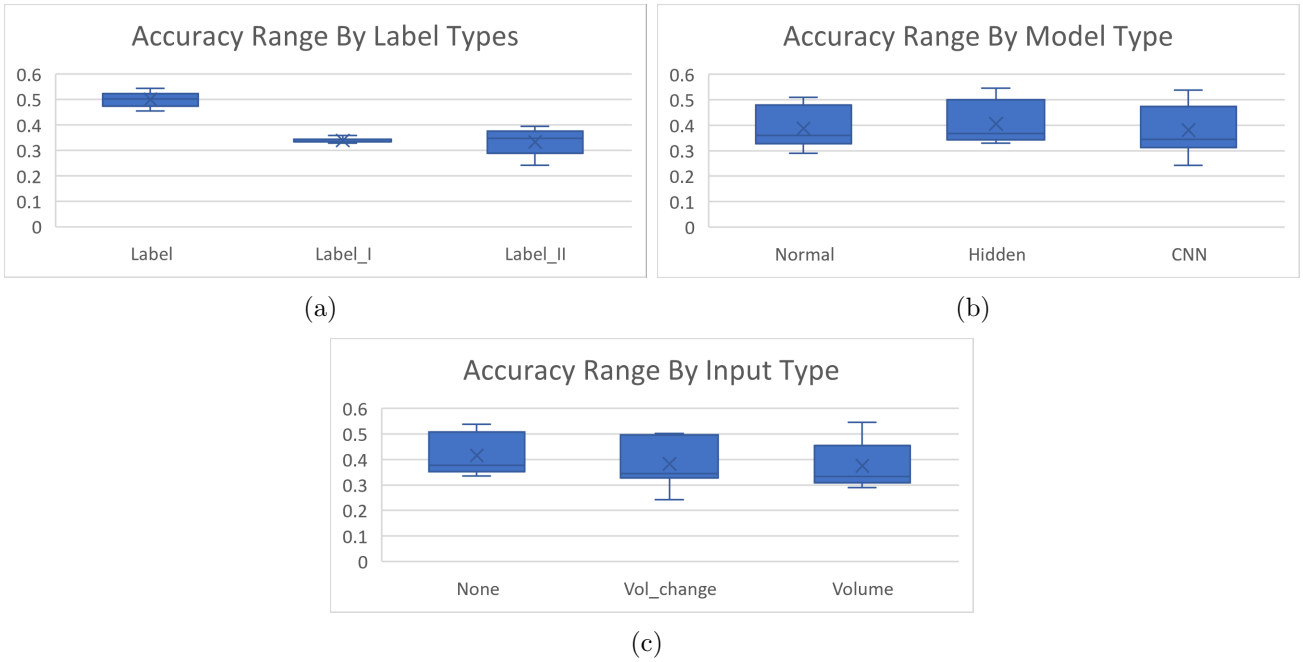


Figure 5.10: *(a) Accuracy Range By Label Types (b) Accuracy Range By Model Types (c) Accuracy Range By Input Types*

The average Accuracy for "Label", "Label_I", and "Label_II" is 0.50, 0.34, and 0.33 separately. And the highest and lowest for each are [0.54, 0.46], [0.36, 0.33] & [0.40, 0.29]. As we can see in Figure 5.10 (a), only the box and whisker chart for the range of accuracy calculated by Label types showed the most difference between all three options. This means the Label type was probably the main cause for the differences in our accuracy changes.

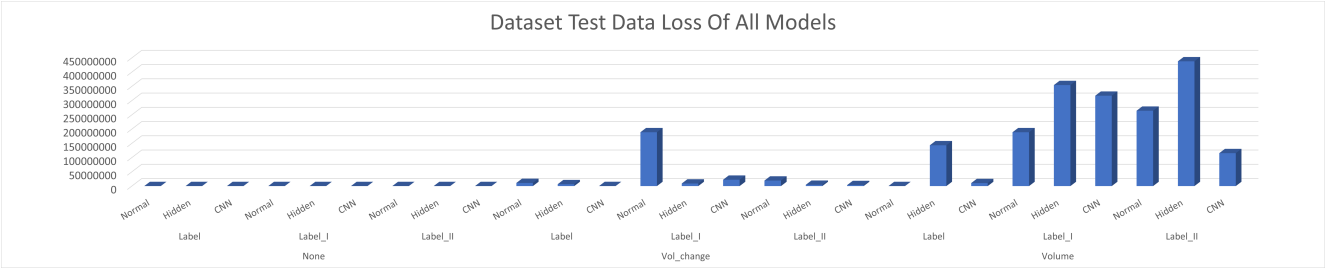
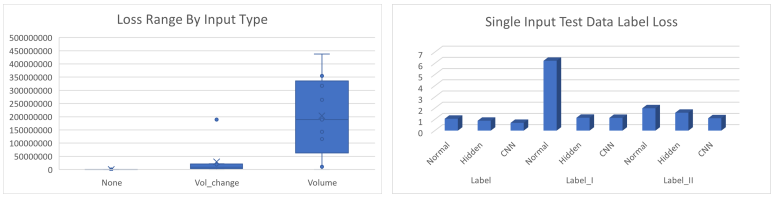


Figure 5.11: *Dataset Test Data Loss Of All Models*

Now that we have looked through the accuracy results, let's move on to our loss results. Loss measures the error between the predicted output and the true output. It is a numeric value that quantifies how well a model is able to approximate the target function. Figure 5.11 shows a very weird loss score and they are unreasonably high for our multi-input models. Figure 5.12 (a) shows the range of our loss data where the loss is so high that the scores are in the hundred million. Figure 5.12 (b) is used to complement (a) as the only models that have a normal loss score are the ones implemented with the single input data approaches. There is also one particular data point that stands out within this result set but that is probably just one outlier as the rest are acting normally.



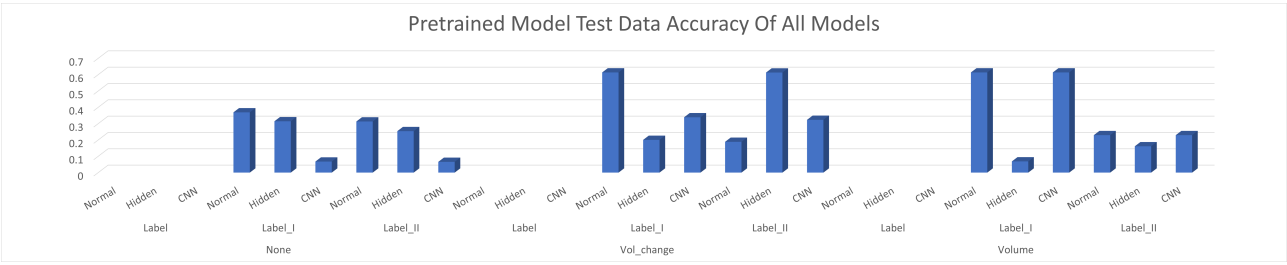
(a) (b)

Figure 5.12: *(a) Loss Range By Input Type (b) Single Input Test Data Label For Clarity Because Data Is Too Small Comparing To Other Data Side By Side*

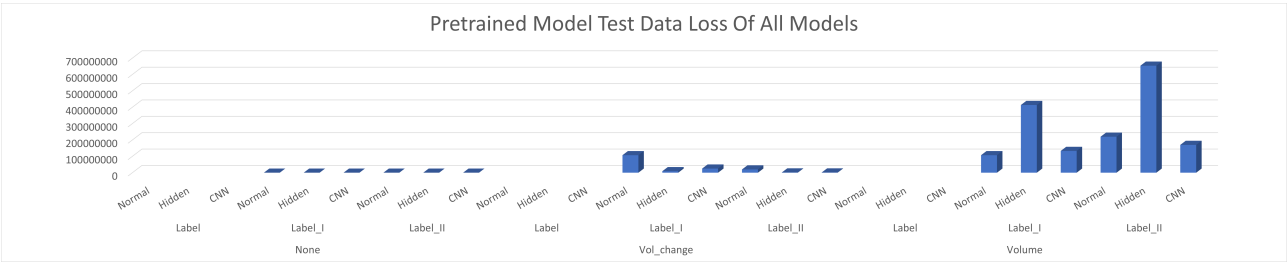
This unusual data means our models are making a lot of errors in their predictions and in other words, the model is performing terribly and is not able to fit the data well. There are several reasons why this is happening. It could either be the model is overfitting the training data or that the model is memorizing the data instead of learning its general patterns. Or it could also mean the model architecture is not suitable for the particular task and as result, could not capture the relevant patterns in the data.

Testing Data Extracted From Pretrained Model Analysis

Because we also wanted to compare our data with other similar models, we have also generated a new testing data label set to be used against our trained model. This model we have used has an f1-score of 93.25% so we can say that if our model shows a high accuracy score against the results of this then that particular labeling technique we used for each label for our dataset can be said to be highly relatable to news sentiment. Because we do not have the same amount of result outputs as we do with the "Label" label categories, the results you see in Figure 5.13 only include the one from our models trained with "Label_I" and "Label_II".



(a)



(b)

Figure 5.13: *(a) Pretrained Model Test Data Accuracy (b) Pretrained Model Test Data Loss*

The accuracy score does not show any obvious patterns. Their average score is around 0.31 which is around the same as our average score when we are testing against our own testing data. There are 4 different models that reached a score of around 0.6 and they are all the results from within our multi-input models. Other than this it shows no other significant data pattern.

The only pattern the loss score shows follows the exact pattern of their loss scores with our original test data. The loss in our single input models remains at a normal range, Vol_change and Volume models are both at a ridiculously high range. Because of this, there is no need to analyze the details of the loss scores.

DL Methodology Result Discussion

All the results we have found in this particular implementation, it has shown that the accuracy of all our models is very unreliable and none shall be used for production. "Label" in our labeling remains higher accuracy over our other labeling methods such as "Label_I" & "Label_II". If we look at our average accuracy categorized by labeling method, just like our previous implementations' results, we can also spot the pattern where the accuracy of our results also behaves in the way that the accuracy_score = 1 / number_of_label_categories.

The number or the type of inputs also showed no change to the pattern of our accuracy. The loss metrics on the other hand tell us that all multiple-input models fit our data very poorly, particularly the models trained with the Volume label as their second input.

The results from our pre-trained model evaluation also tell us two things. Our trained models' output does not relate to our chosen pre-trained model. Second, our models do manage to recognize the text pattern as it produces the same loss pattern for all our models.

5.1.3 Final Discussion Of Results

Both our NLP approach methodologies showed to have extremely poor results, in order to find out the cause we shall have a look at all the patterns and questions we have found and raised. We can then find out which part of this research is causing problems.

First Problem

Pattern Spotted: Accuracy results gathered from our CML approach showed a pattern where [accuracy_score = 1 / number_of_label_categories].
Questions Raised: Why is our accuracy related to the number of categories in our labels? Is this caused by the increase in the complexity of the problem or our CML approach models cannot distinguish between different categories?
Hypothesis: If the pattern is shown again in the results of our other implementation methods, then it means that this is a problem caused not by the CML methodology.
Evidence: The pattern is shown in the results from our DL approach using the original testing dataset and partially shown in the results using the pre-trained model dataset. This shows that all the models in our failure to understand the pattern in our data and this is not a problem caused by our CML implementation failure to handle the increase in complexity of the problem.
Possible Causes: This means it is caused by the dataset we are using to train the models. Further investigation is needed.

Second Problem

Pattern Spotted: Within our DL approaches, it is shown that the type of inputs does not affect our accuracy scores but it does with the loss scores.
Questions Raised: Why is the loss normal with our single-input models but not with our multi-inputs models? And why is the loss in the models using Volume as its second input much higher than the ones using Vol_change.
Possible Causes: Because this is something that does not happen for our single-input models, this means the cause is within our multi-input models. The second input data is also a factor. Further investigation is needed.

Third Problem

Pattern Spotted: The loss rate is a lot higher when the second input is Volume instead of Vol_change
Questions Raised: Why is this happening?
Possible Causes: The trading volume does not or barely relates to our News input data and labeling methodologies. This also proves that our methodology in calculating the change in price volume as second input does have a correlation to our data.

Findings From Results

Here is a summary of our findings from our results with the other data we have found within our results.

- There is something wrong with the dataset we used for training our models in both our NLP approaches.
- There is something wrong with our multi-input models causing them to have a high error rate between the predicted and true output.
- Our methodology in calculating the change in price volume may have a direct correlation to our data.
- The trading volume cannot be proven to have a direct correlation to our data.
- The labeling methodology is the main cause of the change in accuracy in our models.
- Our experimental improvements such as "Hidden" and "CNN" does not prove to be useful in increasing the accuracy or loss of our DL approach base models.
- Our DL models are able to recognize text pattern as it does with the pre-trained model.
- The number of inputs shows no effect in increasing the accuracy of our models.

5.2 Design And Implementation Evaluation

In this section, we would evaluate all the designs and implementations, either commenting on their own or with the conclusion of results we have gathered above in section 5.1.

5.2.1 Data Gathering Methodology

From our final findings in our results, we identified a few problems and positive points that came from the results we extracted within our data-gathering methodology. First, we have concluded that the cause of the poor accuracy results for the NLP implementations originated from the dataset created by the implementation in this methodology. Second, the trading volume cannot be proven to have a direct correlation to our data. Third, our change in trading volume methodology does show a relation to our data but has not been proven to be useful for increasing the accuracy of our model.

There are usually multiple reasons that lead back to the data on why a model is having poor results. This is a list of reasons why they should or should not be the explanation for our problem.

- Insificient data. If the dataset is too small or does not include enough variation of the that a the model may not be able to learn effectively. This is excluded from a possibility as it is shown to have a similar accuracy with our classical machine learning modeling approach which usually requires less training data than a deep learning modeling approach.

- Biased data. If the dataset contains more data for one category than the other then the model could be skewed toward that bias. This is also excluded as the "Label" and "Label_I" both have a very balanced 1:1 ratio therefore and yet they perform the same as "Label_II".

- Noisy data. If the dataset contains a lot of irrelevant or random data it can be difficult for the model to learn the underlying patterns. Despite our high loss for our multi-input models that contain extra numeric data as input such as Volume and Vol_change, because the accuracy for our single input datasets is simply just as low, this reason is also rejected. The loss score is something we should look at in section 5.2.3 as this reason takes the whole set of results into account.

- Outdated data. If the training dataset is outdated then it might not be able to correctly identify the new unseen data. This can also be excluded because our testing data is picked randomly within our dataset therefore it is within the same date range as our training data.

- Poorly labeled data. If the dataset is labeled incorrectly or inconsistently, the model might not be able to learn the patterns in the data. There is no evidence in our data that can prove this reason to be incorrect.

Within our dataset, there are three separate parts, news headings, labels, and additional data for our second input methods. The second method has been shown to have no effect on the accuracy of models therefore it can be excluded from the cause of our models' underperforming results. This leaves us only one option, news headings and labels.

This gives us the conclusion that the incorrectly labeled news headings and labels that we used and gathered for our research are probably the main cause of our problems. This could mean that either the news dataset we used has very little to virtually zero effects on the big picture of the stock market or our labeling methods simply failed to capture or represent the market trend.

5.2.2 Classical Machine Learning Methodology

Other than the problems we have identified above in our result findings, we would also like to make some improvements to our design and implementation. As we mentioned earlier, some of our

models do not exactly predict some of our labels no matter right or wrong. This is not a problem with the tools or algorithms we used but rather the theory behind it. Different feature extraction and encoding algorithms got their pros and cons. Sometimes maybe the pattern of the encoding of the features simply cannot be learned by some of the classifiers. In future implementations, this also has to be taken into account so we can not waste time and resources to train these model combinations.

As we talked about in the section 4.4.1, the Part-Of-Speech / Dependency Parsing feature extraction method did not turn out the way it should have and we failed to get it to work. Obviously, these are some old-school machine-learning techniques and they are not as widely used anymore but it is not all bad because classical machine-learning methods do not take as long to train and are very computationally cost-efficient. They do not offer as high of accuracy but for smaller models, they still work very well. Therefore, for future research, we would like to give a few suggestions to improve this method and maybe it can be implemented. Or to further improve we can improve our design on the feature extraction part of the model where we not only can extract the three categories we mentioned into account but also other extra bits of information that our current design ignores. Second, we can also implement a function or model that normalizes our news heading inputs or we might as well build a better part-of-speech tagger or dependency parser that is fine-tuned for news headings so it can identify the tags correctly.

5.2.3 Deep Learning Methodology

From our result findings, some of the problems belong to our deep learning approach. The main problem is that something went wrong with our multi-input models as they all have an unusually high loss score. In this case, we think it could be caused by two probabilities. First, it could be a result of a fundamental error in how we approached the design of this model. We concatenated our first input directly with our second input. And in other words, this added our second layer's output to the end of the tensor of our first input layer's output.

```
### original methodology ###
# text is a layer with output shape=(None,256)
text = Dense(shape=(256,))
# number is a layer with output shape=(None,1)
number = Dense(shape=(1,))
# combined_layer is a layer with output shape=(None,257)
combined_layer = concatenate([text, number], axis=1)
```

This means the "pooled_output" which was originally supposed to be the encoded and vectorized product of the context of our text, and the second input is now part of this encoded context. In order to fix this issue, I suggest we perform a tensor argumentation process. First, we would tile and expand our second_input so that it matches the shape of our pooled_output, from shape=(None,1) into shape(None,256) in other words [x] -¿ [x,x,...,x]. Then we can stack both our outputs together into one output so that it becomes one tensor of shape (None, 256, 2) where [[a,b,c,d,...],[x,x,x,x,...]]. This should be able to contain the context of both our second layer value and our encoding from our BERT layer.

```
### proposed improved methodology ###
# text is a layer with output shape=(None,256)
text = Dense(shape=(256,))
# number is a layer with output shape=(None,1)
number = Dense(shape=(1,))
# expand number to the shape of text
```

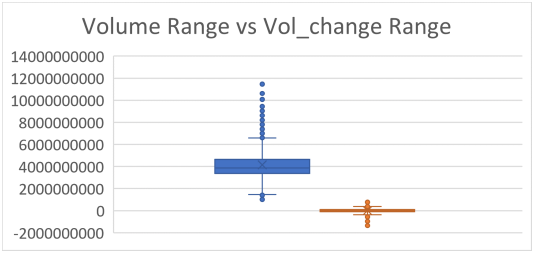


Figure 5.14: *Data range of Volume and Vol_change*

```
expand_number = tile(number, [1,shape(text)[-1]])
# stack both layers to achieve output shape=(None,256,2)
combined_layer = stack([text, expanded_number], axis=-1)
```

More to that because we haven't normalized our second_input this created a really weird scale for our concatenated tensor. As seen in Figure 5.14, the data range for Volume is a lot bigger than Vol_change and this might be another possible reason why the loss is so different from one another, or it could still be a result of the other possible cause we mentioned above. This is why in my proposed change here, we would also normalize our second_input data before we combine it with our BERT encoder output. I propose two ways of achieving the normalization so we can transform our large continuous numerical data to become a number between 0 and 1. First would be the Min-Max Normalization approach, with this we are guaranteed to reshape all our data to between 0 and 1. The downside is that it cannot handle outliers very well or else the normalization would not work very well. The second would be the Z-Score Normalization approach. It handles outliers better than min-max but the data might not spread as well as the other one.

Chapter 6

Conclusion

6.1 Hypothesis Summarization

The objective of this project is to investigate and quantify textual financial data using Natural Language Processing techniques. Natural Language Processing is increasingly being used in quantitative finance to extract valuable information that traditional methods do not offer for three main reasons. First, the increase in available textual data alongside the growth of the internet, social media, and other platforms. Secondly, NLP has been made more accurate and efficient through the advancement in machine learning and artificial intelligence. Thirdly, the increase in competition within the finance industry forces investors to gain a competitive edge by leveraging technology.

Our project aimed to achieve this goal by employing a mixed-method approach for modeling the data we have gathered through our data gathering and labeling process. First, we gather financial data for the News dataset we have acquired through online sources. Then we approach the task using old-school machine learning and another deep learning NLP approach. The results are then compared with the goal of finding out which method performs better for our task.

We have made a few assumptions for our hypothesis. It is assumed that the content within the News dataset used in our study represents the majority of cause for our market makers through their fundamental analysis's causal reasoning. Another assumption we have made is that our data annotation methodology used within the data-gathering stage is able to capture the correlation between our news data and its effect on the stock market correctly.

6.2 Summary Of Findings

With an average accuracy of 50, 33, and 33 percent for each data annotation technique we have trained all the models in both our Natural Language Processing approaches, the results show that our project failed to achieve the expectations we have hoped for. Although it did succeed to provide sufficient data to narrow down and identify the cause of failure to be traced back to our Data Gathering methodology.

6.2.1 Main Potential Causes To Failing Our Desired Objectives

- Our assumption is that News data is a market catalyst for the stock market and its trend is enough to be predicted without using other economic indicators.

- The assumption that the context within the News data we have gathered is sufficient or significant enough for our models to learn the stock market trend based on our previous assumption. Or, its effect is shown in the trend of our chosen stock market index (Standard and Poor 500).
- A follow-up assumption to our previous two is that our data labeling methods are sufficient enough to capture the trend in the stock market index of our choice.

6.2.2 Other Question Raised, Problems Identified And Their Solutions

We have also identified other problems within our project although it is believed that they did not contribute much to the fundamental cause of failure. Here's a list of what we found and the solutions that we came up with.

1. PROBLEM: Our methodology behind combining two inputs is fundamentally incorrect. When one set of outputs must remain whole, the process of combining both outputs must not change the tensor shape of that particular output.

   SOLUTION: Instead of concatenating both tensors by adding to the end of the other, scale up the second tensor to match the size of the other and then stack them up in another dimension.

2. PROBLEM: We could not identify the problem causing the huge difference in the loss for models trained using Volume and Change In Volume as secondary inputs. Could be caused by the relevance of our datasets or because of the different range of the datasets.

   SOLUTION: Normalize both datasets so that their results can be compared fairly.

3. PROBLEM: Our experiment on adding more parameters or a convolution neural network to an encoded output from a BERT encoding layer supposed to represent the context of the original text has shown to have no effect on our model accuracy, or it might be a result of our main causes mentioned above where it the models are unable to learn the underlying patterns anyways.

   SOLUTION: Train the models with a dataset that is proven to be correctly labeled and analyze the results.

6.3 Significance Of Findings And Lessons Learned

Although we failed to achieve the goal we hoped for in our project. This does not mean there is nothing we can learn and contribute to Natural Language Processing within the financial field through our findings.

We have learned that despite the recent sudden growth and advancement in machine learning and artificial intelligence, no matter how much better the algorithms are able to handle their specialized task, such as a state-of-the-art model like Transformers in our case, we can not move our focus away from the quality of our training data. Because fundamentally, it is what our algorithms are learning from.

If we are to design the data gathering part of this project again, there are a few key points we should mainly focus on. First, identify the correct dataset that is proven to be correlated enough

for our goal. Second, correctly label or increase the amount of information the labels contain. In our case, we could possibly increase the category or quantify our output so that it contains information on how steep is the change in momentum or the price pattern it contains and etc.

And the modeling design part, we would like to re-implement the multi-input models again using the solutions we have provided above where we combine two inputs correctly so that our model can actually learn the relevance of our text input data and our second input data. We would also like to normalize some of our data before feeding them into our models so that we can analyze our different options without bias.

6.4 Future Research

With the results we have gathered, we would also like to propose a few ideas for future research. Our project originally does include investigating how well the model works with purely fundamental financial data (News Headings) and how well it works when it is also combined with technical data (Volume, Volume Change), it eventually failed due to a design flaw in our multi-input models. In a future project, once this problem is also acknowledged, we may commit further investigation into a wider range of financial indicators.

Another idea we would like to propose is that we may be able to narrow down our target. Instead of training the models with global news and data from a stock market index. As there are too many stochastic moving factors, we could focus on a specific company where we can train our model with more tailored data, for example, company news, quarter earning reports and etc. We may also choose our targeted company based on their price volatility and the quality of the company to calculate our risk.

6.5 Conclusion

Despite our project not being able to achieve the desired outcomes and the challenges we have encountered that prevented us from reaching our goals. Disappointing at first, but this project is not all for nothing. The lessons that we have learned and the experiences that we have gained are much more valuable.

Before I started this project, my knowledge of artificial intelligence was minimal. When I saw that previous knowledge in AI was recommended for this project topic in the TYP booklet, it did put me off a little because I was scared. But the Finance part of the project title did lead me to have this as my choice. To be honest, my passion for computer science has been on a decline since I started university because the content we covered in the first two years did not particularly interest me. This project changed everything.

As I learn more about this field of computer science, it re-lit my excitement just like it did when I borrowed my first programming textbook for C++ from the library when I was 12. I have always had a weird passion for problem-solving and automation probably because I am lazy. And Natural Language Processing is never something I thought I would find interest in because I never did well in language classes in school. Concepts and things that I learned about AI throughout this journey opened a whole new world for me. It taught me how to look at languages in a statistical and logical way and showed me the vast possibilities of what we can achieve from sentiment analysis, machine translations to information extraction and etc.

I would also like to address some of the knowledge I gained from this research that I did not know before. I have learned about the evolution of Natural Language Processing techniques, from

classical machine learning methods to deep learning methods that are majorly used nowadays. I have learned how to make my own machine learning, build my own neural network and train it to solve a particular task. It is like raising a child, not that I know how it feels like but what I assume it is supposed to be. Skills like using tensorflow, pytorch, scikit-learn, nltk and etc now also go into my resume. I have also learned a lot more about quantitative finance than I used to and it excites me how I can combine both my specialty (computer science) and interest (finance) into one thing. Data science is also something I have never thought I would be interested in but this project lead me to choose it as one of my options for my postgraduate degree as well as of course machine learning and AI.

Last but not least, the initial failure to achieve our project's final goal is honestly tragic, but the journey has not only reignited my passion for computer science. It also showed me new paths within the subject such as Natural Language Processing and where it intersects with my other interests such as Finance. I am grateful for the challenges and learning opportunities this experience brought me. The end of this project does not mean an end to this chapter, I will continue further research and hopefully achieve better outcomes in the days to come.

Bibliography

[1] *5 Industries Which Rely Heavily on Artificial Intelligence in 2022 — UNext — UNext*. https://u-next.com/blogs/artificial-intelligence/5-industries-which-rely-heavily-on-artificial-intelligence-in-2022/. (Accessed on 03/23/2023).

[2] Nicolo Cosimo Albanese. *Multi-class Text Classification using BERT and TensorFlow — by Nicolo Cosimo Albanese — Towards Data Science*. https://towardsdatascience.com/multi-label-text-classification-using-bert-and-tensorflow-d2e88d8f488d#98ee. 01/19/2022).

[3] *Classifier comparison — scikit-learn 1.2.1 documentation*. https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html. (Accessed on 03/01/2023).

[4] *Classify text with BERT*. https://www.tensorflow.org/text/tutorials/classify_text_with_bert. (Accessed on 03/16/2023).

[5] Xiao Ding et al. *Deep Learning for Event-Driven Stock Prediction*. 2015. URL: https://www.aaai.org/ocs/index.php/IJCAI/IJCAI15/paper/viewFile/11031/10986 (visited on 03/20/2020).

[6] *https://corenlp.run*. https://corenlp.run/. (Accessed on 03/18/2023).

[7] Yuxuan Huang, Luiz Fernando Capretz, and Danny Ho. "Machine Learning for Stock Prediction Based on Fundamental Analysis". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)* (Dec. 2021). DOI: 10.1109/ssci50451.2021.9660134. (Visited on 06/08/2022).

[8] Insider Intelligence. *Artificial Intelligence in Financial Services: Applications and benefits of AI in finance*. Insider Intelligence, Jan. 2023. URL: https://www.insiderintelligence.com/insights/ai-in-finance.

[9] Jean-Baptiste. *Jean-Baptiste/roberta-large-financial-news-sentiment-en · Hugging Face*. https://huggingface.co/Jean-Baptiste/roberta-large-financial-news-sentiment-en. (Accessed on 03/19/2023).

[10] *krishnaik06/Stock-Sentiment-Analysis*. https://github.com/krishnaik06/Stock-Sentiment-Analysis. (Accessed on 03/19/2023).

[11] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[12] Ted Moore. *Neural Network Parameters*. https://learn.flucoma.org/learn/mlp-parameters/. (Accessed on 03/14/2023).

[13] Anthony Munns. *How AI Is Utilised In The Algorithmic Trading Sector*. 8topuz, Dec. 2019. URL: https://8topuz.com/how-ai-is-utilised-in-the-algorithmic-trading-sector (visited on 10/01/2022).

[14] Orbex. *Fundamental, Technical and Sentiment Analysis*. Orbex Forex Trading Blog, May 2018. URL: https://www.orbex.com/blog/en/2018/05/fundamental-technical-and-sentiment-analysis (visited on 02/04/2023).

[15] *S&P 500 - Wikipedia*. https://en.wikipedia.org/wiki/S\%26P_500. (Accessed on 02/28/2023).

[16] Two Sigma. "Using News to Predict Stock Movements". In: *Kaggle. com* (2020).

[17] J Sun. *Daily News for Stock Market Prediction*. kaggle.com, June 2016. URL: https://www.kaggle.com/aaron7sun/stocknews.

[18] CFI Team. *Technical Analysis - A Beginner's Guide*. Corporate Finance Institute, Jan. 2023. URL: https://corporatefinanceinstitute.com/resources/capital-markets/technical-analysis/ (visited on 02/04/2023).

[19] *TensorFlow Hub*. https://tfhub.dev/. (Accessed on 03/18/2023).

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

Appendices

Appendix A

Original Proposal

Financial Natural Language Processing

Abstract

The proposed project FNLP is aimed to try and predict the market trend based on news and investigate the impact news has on the market.

I have found a dataset on Kaggle.com which with news tittles and effect it had on the market that particular day and I would be using it for this project. I would also like to compare and find out how long of an effect would news have on would the trained models by comparing the change in market price for different length of time and accuracy of different models.

First stage of the project is to train a stochastic model with an existing dataset or a made-up dataset. In this stage I would scrab data online and create my own dataset and after that they can be used for the training model.

Given the dataset the program shall process the text from the news. Split the data into a percentage of seen data and unseen data. Then the machine shall create a stochastic model based on the seen data.

After creating the model, the unseen data shall be fed into the machine. We can then compare the predicted results to the result from the dataset and decide if the modelling was accurate and/or how much effects news has on the market.

At last, I can compare the accuracy from different models and sort out a solution.

Introduction

In this modern world we are surrounded more and more by AI to help with our daily basis such as autocorrect, google search, siri and even as of typing this, grammar and spelling correction is also based on a branch of Artificial Intelligence - Natural Language Processing.

Of course, the financial industry would not be left out and in fact it is one of the five industry that relies on AI the most.

The financial industry is known to have high working hours and require a lot of repetitive work and data analysing. This is where atomization comes in. It is also calculated that AI could potentially save North American banks $70 billion by 2025. Even more impressive, AI applications is estimated to potentially save $447 billion aggregate cost saving for banks by 2023.

Besides of saving money, AI is also used in making money. When we are talking about the financial industry, trading is probably one of the very first thing that comes on your mind,

trading like they do in the movies The Big Short or The Wolf of Wallstreet. Here comes the solution of atomization of human trading activities – Algorithmic Trading.

Algorithmic Trading generates 50%-70% of equity market trades, 60% of future trades and 50% of treasuries. This important role in the market works by executing orders based on the market conditions, price, volume and time. The market repeats itself and follow patterns, but the market is also constantly evolving, therefore it is inefficient to hard code an algorithm that would last and be sustainable as you would have to constantly update the algorithm so works with the current market. AI offers such programs adaptability.

The proposed project does not offer the ability to make automized trades, but it could be used to play a massive role in an algorithmic trading program as it should cover the fundamental analysis in a trade. The reason why I chose to use news other than other language-based source such as tweets, reddit posts and etc is because the reachability of it. Reachability creates impacts and can fluctuate prices.

From all the related published researches out there right now they are usually based on third party datasets and the data accuracy can really be doubted and this is what I am trying to find out with this project and see if I can find out a more reliable number for this kind of data sets.

Background

Similar topics were also done previously before. The closest one that I have found was a competition hosted on Kaggle.com by the company Two Sigma three years ago by the time of writing. Unfortunately, the company removed all its data from the website and the code were kept by the company and the competitors.

I have found alternative datasets that I can use but it is way less detailed. In theory I could collect my own data and use it for this project, but it would not be possible within the time that we were given.

There are also researches done on stock prediction based on fundamental analysis using machine learning. In the paper they investigated three machine leaning algorithms, Feed-Forward Neural Network, Random Forest and Adaptive Neural Fuzzy Interference System.

They have found out the Random Forest model out preform the other models.

But from out of all these similar projects, none of them did any analysis on the datasets and it would also simply be interesting to see if there are any other effects on other market indicators other than Dow Jones Industrial Average.

The Proposed Project

Aims and Objectives

The ai of this project is to try and predict the market trend based on news and investigate the impact news has on the market. Also investigate the optimal duration that a news have on the market.

- Create several datasets based on the same news dataset but difference would be the price difference depending on the change in duration of time.
- Create a training model that can handle these datasets
- Train the models with 80% of the data and use the rest of the 20% to test the accuracy of the model
- Find out the dataset with the highest accuracy and try to calculate the optimal length of time that news has effect on the market

Methodology

This would be separated into two stages.

First, create the datasets. The news title dataset contains the news titles and the date. This data set is scraped off reddit's top 25 news of that particular day from reddit.com. Then I would download the prices for Dow Jones Industrial Average, S&P500 or NASDAQ prices of those days, find out the opening and closing price, highest and lowest. I would then create a different other datasets for the prices but instead of one day it could be three days combined or five days combined and etc.

Second, with the datasets prepared I can then start working on the training model. The model shall processes the news titles, at the moment I am thinking using the bag of words algorithm to count the amount of each words appearance and train it with the price data. I would also try to classify if the news is a positive one or a negative one first and it should help classifying the news.

A stochastic model shall be used and in this case I am planning to use the regression model as it seems to suit the situation most because the there are too many factors in the stock market and fundamental analysis cannot be the only indicator and for this project all the other factors would seem random.

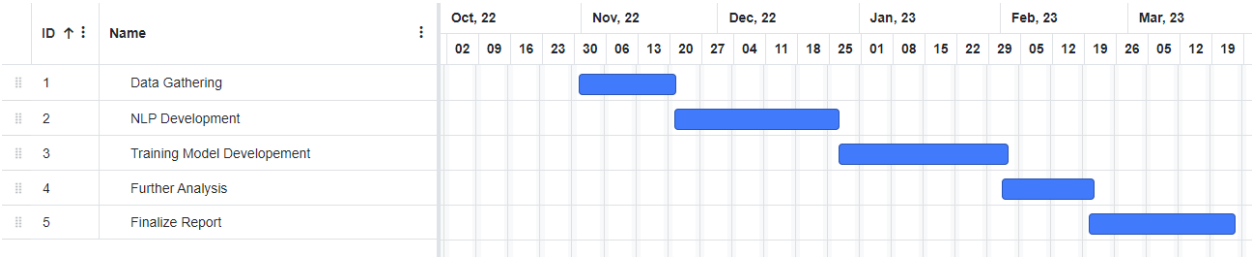
After training the model it should be able to judge if the market is going up or down based on the news. To carry this test out we are going to use the rest of the 20% of unseen data and enter them into the model. We can then compare if that calculated result matches with the actual prices. Once all the data are processed, we can calculate the probability of the accuracy of the trained models.

Programme of Work

This project will begin early November 2022, running till March 2023 and it shall be split into the stages below.

- Source historic news online.
- Match historic price with news dataset and create datasets
- Create NLP part of the training model so it can process the news dataset
- Design the training model so it can work with the datasets
- Train the models with the datasets
- Analyse the results and do further testing based on the results

The overall schedule for the project is displayed in the following Gantt chart.



Resources Required

In order to proceed with this project, a data set of news and historic market data would be required. The news datasets would be provided by an opensource dataset website Kaggle.com and the market data would be able to be found on the internet.

References

1. 6 Applications of NLP in Finance
   https://www.analyticssteps.com/blogs/6-applications-nlp-finance
   Accessed October 2022

2. Artificial Intelligence in Financial Services: Applications and benefits of AI in finance
   https://www.insiderintelligence.com/insights/ai-in-finance
   Accessed October 2022

3. 5 Industries Which Rely Heavily on Artificial Intelligence in 2022 | UNext Jigsaw
   https://www.jigsawacademy.com/blog/5-industries-which-rely-heavily-on-artificial-intelligence-in-2022

Accessed October 2022

4. How AI Is Utilised In The Algorithmic Trading Sector
   https://8topuz.com/how-ai-is-utilised-in-the-algorithmic-trading-sector
   Accessed October 2022

5. Machine Learning for Stock Prediction Based on Fundamental Analysis
   https://ieeexplore.ieee.org/document/9660134
   Accessed October 2022

6. Sun, J. (2016, August). Daily News for Stock Market Prediction, Version 1. Retrieved [27/10/2022] from
   https://www.kaggle.com/aaron7sun/stocknews

Appendix B

Complementary Tables And Graphs

B.1 Deep Learning Model Layering

B.1.1 Graphs

B.1.2 Tables

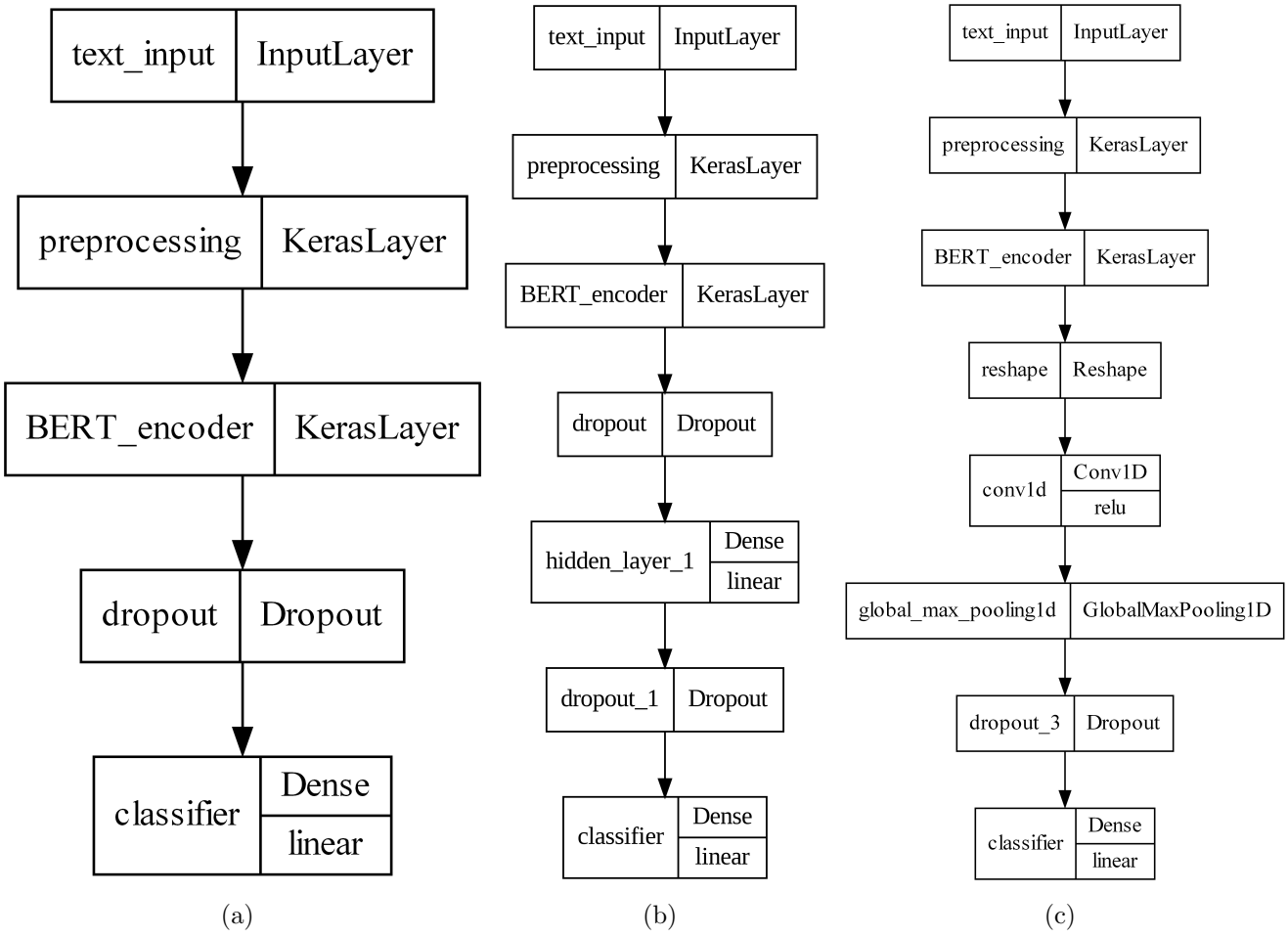


Figure B.1: *(a) Single Input Classification Model Layering*
(b) Single Input Classification Experimental Model With More Hidden Layering
(c) Single Input Classification Experimental Model With Additional Convolution Neural Network For Layering

Figure B.2: *(a) Multiple Input Classification Model Layering*
(b) Multiple Input Classification Experimental Model With More Hidden Layering
(c) Multiple Inputs Classification Experimental Model With Additional Convolution Neural Network For Layering

Single Input Classification Model With More Parameters			
Layer (type)	Output Shape	Param	Connected to
text_input (InputLayer)	[(None,)]	0	[]
preprocessing (KerasLayer)	{'input_word_ids': (None, 128), 'input_type_ids': (None, 128), 'input_mask': (None, 128)}	0	['text_input[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 256), 'pooled_output': (None, 256), 'encoder_outputs': [(None, 128, 256), (None, 128, 256), (None, 128, 256), (None, 128, 256)], 'default': (None, 256)}	11170561	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
dropout (Dropout)	(None, 256)	0	['BERT_encoder[0][5]']
hidden_layer_1 (Dense)	(None, 20)	5140	['dropout[0][0]']
dropout_1 (Dropout)	(None, 20)	0	['hidden_1 [0][0]']
classifier (Dense)	(None, *1 for binary classification and n for non-binary classification*)	(output shape times previous output shape)	['dropout_1[0][0]']

Table B.1: *Single Input Classification Experimental Model With Extra Hidden Layer For More Parameters*

Single Input Classification Model With CNN			
Layer (type)	Output Shape	Param	Connected to
text_input (InputLayer)	[(None,)]	0	[]
preprocessing (KerasLayer)	{'input_word_ids': (None, 128), 'input_type_ids': (None, 128), 'input_mask': (None, 128)}	0	['text_input[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 256), 'pooled_output': (None, 256), 'encoder_outputs': [(None, 128, 256), (None, 128, 256), (None, 128, 256), (None, 128, 256)], 'default': (None, 256)}	11170561	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
reshape_1 (Reshape)	(None, 256, 1)	0	['BERT_encoder[0][5]']
conv1d_1 (Conv1D)	(None, 256, 32)	128	['reshape_1[0][0]']
global_max_pooling1d (GlobalMaxPooling1D)	(None, 32)	0	['conv1d_1[0][0]']
dropout (Dropout)	(None, 256)	0	['global_max_pooling1d [0][0]']
classifier (Dense)	(None, *1 for binary classification and n for non-binary classification*)	(output shape times previous output shape)	['dropout[0][0]']

Table B.2: *Single Input Classification Experimental Model With Additional Convolution Neural Network*

Multiple Input Classification Model With More Parameters			
Layer (type)	Output Shape	Param	Connected to
text_input (InputLayer)	[(None,)]	0	[]
preprocessing (KerasLayer)	{'input_word_ids': (None, 128), 'input_type_ids': (None, 128), 'input_mask': (None, 128)}	0	['text_input[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 256), 'pooled_output': (None, 256), 'encoder_outputs': [(None, 128, 256), (None, 128, 256), (None, 128, 256), (None, 128, 256)], 'default': (None, 256)}	11170561	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
number_input (InputLayer)	[(None,1)]	0	[]
concatenate (Concatenate)	[(None,257)]	0	['BERT_encoder[0][5]', number_input[0][0]]
hidden_layer (Dense)	(None, 128)	33024	['concatenate[0][0]']
dropout_1 (Dropout)	(None, 128)	0	['hidden_layer [0][0]']
classifier (Dense)	(None, *1 for binary classification and n for non-binary classification*)	(output shape times previous output shape)	['dropout_1[0][0]']

Table B.3: *Multiple Inputs Classification Experimental Model With Extra Hidden Layer For More Parameters*

Multiple Inputs Classification Model With CNN			
Layer (type)	Output Shape	Param	Connected to
text_input (InputLayer)	[(None,)]	0	[]
preprocessing (KerasLayer)	{'input_word_ids': (None, 128), 'input_type_ids': (None, 128), 'input_mask': (None, 128)}	0	['text_input[0][0]']
BERT_encoder (KerasLayer)	{'sequence_output': (None, 128, 256), 'pooled_output': (None, 256), 'encoder_outputs': [(None, 128, 256), (None, 128, 256), (None, 128, 256), (None, 128, 256)], 'default': (None, 256)}	11170561	['preprocessing[0][0]', 'preprocessing[0][1]', 'preprocessing[0][2]']
number_input (InputLayer)	[(None,1)]	0	[]
concatenate (Concatenate)	[(None,257)]	0	['BERT_encoder[0][5]', number_input[0][0]]
reshape_1 (Reshape)	(None, 256, 1)	0	['concatenate[0][0]']
conv1d_1 (Conv1D)	(None, 256, 32)	128	['reshape_1[0][0]']
global_max_pooling1d (GlobalMaxPooling1D)	(None, 32)	0	['conv1d_1[0][0]']
dropout (Dropout)	(None, 256)	0	['global_max_pooling1d [0][0]']
classifier (Dense)	(None, *1 for binary classification and n for non-binary classification*)	(output shape times previous output shape)	['dropout[0][0]']

Table B.4: *Multiple Inputs Classification Experimental Model With Additional Convolution Neural Network*

B.2 Deep Learning Implementation

B.2.1 Training Graphs Examples

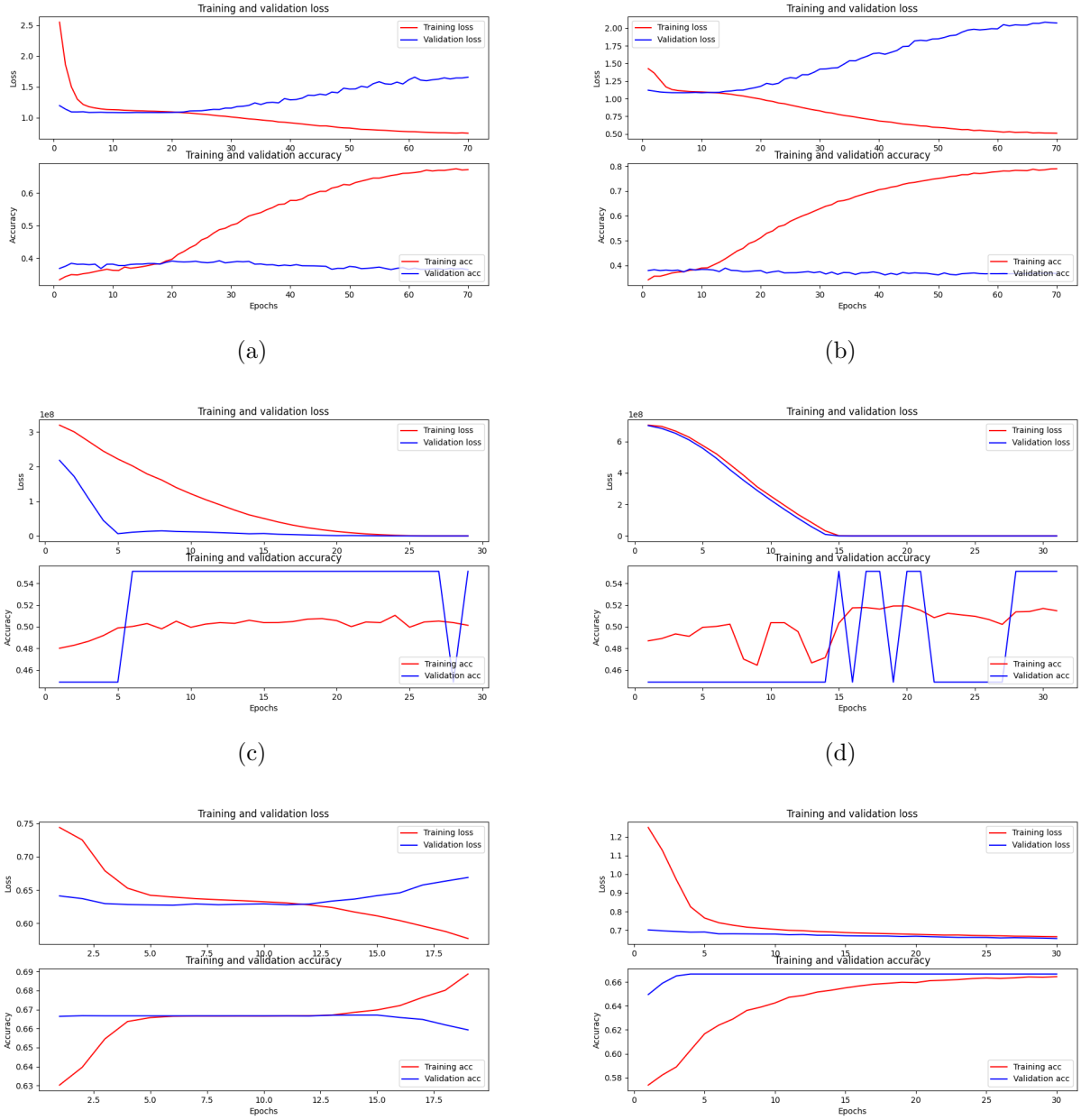


Figure B.3: *Examples of training history graph that we generated during the training of our models*