UNIVERSITY OF HULL

Understanding Social Media: An Analysis of Sentiment using Twitter Data

Final report

Submitted for the BSc in Computer Science

April 2020

by

Brian Davis

Word count: 14,987

Abstract

Sentiment Analysis is a fast track area of research in high demand for businesses, giving them the ability to understand and adapt their business strategies through understanding the way their brand is seen amongst consumers. This paper aims to analyse the Sentiment Polarity of topical tweets using modelling techniques, through the analysis of 70,000 tweets. With the combined use of Deep Learning, Natural Language Processing and Word Embeddings, training and classification of tweets is successfully implemented with good results. Variations in trainable data demonstrated that removal of stopwords, can have a negative effect on classification accuracy. The process of Indexing and Tokenization are effective techniques used to feed data into Neural Networks. Results are evaluated based on data trained alongside different type of hidden layers across Recurrent Neural Networks and Convolutional Neural Networks. The technique of visualisation is used to support the evaluation metrics, which allows for comparison where accuracies are too close to be called better. The project achieved a high accuracy of 96% using a Convolutional Neural Network on unclean data, with results showing lower accuracies on other data. This helped to show that this approach has the possibility to lead to advancements through the use of the techniques showcased.

Acknowledgements

I would like to thank my supervisor Mike Brayshaw, whose expertise in the field and background knowledge were of great use through all stages of the project. I would also like to thank my family, who are always supportive of my choices, and I wouldn't be where I am without their support.

Contents

Abstract i
Acknowledgements ii
1 Introduction 4
1.1 Background Context 4
1.2 Initial Project Background 4
1.3 Ethical Issues 5
1.4 Aims and objectives 5
1.4.1 Objective 1: Classify Textual Data through Modelling Techniques 5
1.4.2 Objective 2: Model Investigation 5
1.4.3 Objective 3: Model Evaluation 5
1.5 Research question 6
1.6 Report Structure 6
2 Literature Review 7
2.1 Technical Background 7
2.2 Natural Language Processing 8
2.2.1 Computational Linguistics 8
2.2.2 Machine Translation 9
2.2.3 Text Analytics 9
2.2.4 Sentiment Analysis 10
2.3 Previous Research 11
2.3.1 Inverse Document Frequency (IDF) 11
2.3.2 APRIORI Algorithm 11
2.3.3 Bag-of-Words and N-grams 12
2.3.4 Sentiment Classification – New Beginnings 12
2.3.5 Subjectivity Extraction 13
2.3.6 Continuous Space Use 14
2.3.7 Using Social Media for the task 14
2.3.8 Neural Networks for Machine Learning 15
2.3.9 Tensor Networks and Vector Space 16
2.3.10 Deep Learning 16
2.3.11 Word Embeddings 17
2.3.12 Big Data and Convolutional Networks 18
2.4 Conclusion 18
3 Technical Development 19

3.1 Narrative Description 19
3.2 System Requirements 19
3.2.1 Python 19
3.2.2 Jupyter Notebook 20
3.2.3 Tweepy 20
3.2.4 Keras and Tensorflow GPU 21
3.3 Data Extraction 22
3.3.1 Exploring the data 22
3.3.2 Exploring Findings 23
3.3.3 Removing Stopwords & Collection Words 24
3.4 Tokenisation 26
3.4.1 Creating the Tokenizer 26
3.4.2 Creating three sets of Data 26
3.4.3 Filtering out Neutrals 27
3.4.4 Tokenizing and Padding 28
3.4.5 Tokenizer Inverse Map 29
4 Project Implementation 30
4.1 Choosing Neural Networks 30
4.1.1 Recurrent Neural Networks 30
4.1.2 Convolutional Neural Networks 31
4.2 Incremental Modelling 32
4.2.1 Early Experimentation 32
4.2.2 Hyper-Parameter Tuning 32
4.2.3 Randomised Grid Search 33
4.2.4 Using different Datasets to test Implementation 33
4.3 Model Evaluation 34
4.3.1 Testing Against Created Data 34
4.3.2 Exploring Embedding Weights 34
4.3.3 Cosine Distance between Words 35
4.3.4 Recursive Neural Network Exploration 36
4.4 Results 36
5 Critical Evaluation 40
5.1 Project Achievements 40
5.2 Further Development 40
5.3 Future Work 41
5.4 Personal Reflection 41

5.5 Conclusion 41
Bibliography 43
Appendix A – Weekly Project Logs 46
Appendix B – Results not included in Main Report 49

1 Introduction

This project will analyse the step-by-step process of development, as well as providing a detailed specification and discussion regarding the background material, alongside a review of the implementation of the project. The report will also review the technical development of the project, discussing key stages alongside providing analysis with regards to decisions made and the testing of any algorithmic models that were made in the process.

The Project Initiation Document (PID) had a primary focus on the classification of texts toward understanding the sentiment polarity using modelling techniques. There have not been many changes regarding the PID since the project was begun, therefore the focus has remained upon the analysis and development of models to answer the research question.

1.1 Background Context

Natural Language Processing (NLP) is a processing method that looks to show how computers can understand human language, more recently through deep learning methodologies. NLP makes it possible for computers to be able to read text, hear speech and interpret that with the ability to read the Sentiment and determine the parts that are most important (SAS Institute, 2020).

Sentiment Analysis is a popular mainstay in the world of Deep Learning. Movie Reviews were always the main source of analysis before the introduction of Social Media outlets. Using TensorFlow and Keras, Natural Language Processing can be used to help businesses and organisations to understand the Sentiment of their customers, through comments posted across popular Social Media sites such as Twitter and Facebook.

Twitter has a limit on the amount of characters that anyone can write at one time of 140. This leads to ambiguity when it comes to the appearance of sentences within this blogging site due to this limitation. This influences abbreviation uses and infrequent words such as aka and lol, alongside the increase in emoji use such as 😊 or ☹.

Although many papers argue for and against the removal of Stopwords and Collection Words, the results can differ. This is something that will be explored within this paper to see if the effects make enough of a difference to support the argument for their removal. Removal of Stopwords is usually done using a pre-compiled stop list, the same goes for this project where the NLTK package was used for the removal. *(Natural Language Toolkit — NLTK 3.5 documentation, 2019)*

1.2 Initial Project Background

Twitter was the main application for data collection for this study due to the ever-growing popularity of the micro-blogging website since its establishment in 2006. Although the earliest papers into Sentiment Analysis focused on Movie Reviews as the Corpus for analysis, later studies suggested that Social Media is a better avenue for this as they are rich sources for data for opinion mining. *(Patodkar and I.R, 2016)*

The project aims to utilise machine learning techniques and expand upon them in a deep learning setting using big data. In the case of this study, 65,000 tweets were extracted from Twitter using the Tweepy library to access Twitters API.

Machine Learning is useful for the application of Sentiment Analysis as there are fundamental advantages towards its use, the main advantage being the different foundations for learning that it offers. Utilising Machine Learning allows the analysis of data and efficient accuracy results to a much higher degree. A computation framework for this is ideal as it enables opinion mining and sentiment analysis which can adapt to the domain. (Technopedia, n.d.)

The aim of this research was to analyse the effects of unigrams towards the context of polarity. Sentiment Polarity is the orientation a sentence undertakes and what emotion is expressed, this can be positive, negative or neutral. It is important to be able to understand the sentiment orientation of text in written or spoken language, which is not always the case as the use of slang and sarcasm can lead to times where even us as humans can't entirely understand the sentiment of something we are reading or hearing.

1.3 Ethical Issues

This project uses real data scraped from the Twitter API. Although the data was from a social media platform, all the data remains anonymous and while the possibility to track who writes each tweet was possible at the time of scraping, once the scraping of this data was completed, all instances of who wrote what and where it came from were not extracted alongside, therefore the data was not traceable to any individual. Data Extraction, Data Handling and Data Preparation were all ethically approved beforehand.

1.4 Aims and objectives

1.4.1 Objective 1: Classify Textual Data through Modelling Techniques

The main objective of this project was to classify data found on the Social Media platform Twitter. Data was scraped from the Twitter API using a Python Library, which was then cleaned of URLs and hashtags to make sure to keep all data anonymous with no links to the posters of the tweets. The Keras library was used to create two variations of popular Deep Learning Neural Network models with the aim to achieve high accuracy numbers towards classification of texts.

Polarity scores are obtained through a Python Library to allow the model to understand and match words to different polarities, being Positive, Negative and Neutral sentiment. Modelling Techniques allow for supervised learning to achieve end results and provide a basis for evaluation of accuracy, sensitivity and recall of results.

1.4.2 Objective 2: Model Investigation

The secondary objective of this project was the investigation and evaluation of models. Whilst many options for hyper-parameters are experimented with and adjusted, nine trained models would be kept with three different variations of the same data to allow comparison of factors which had contributed to past papers regarding the subject.

1.4.3 Objective 3: Model Evaluation

Evaluation is done through visualisation techniques commonly used such as graphs to show training progression of the model, confusion matrices to indicate error metrics and scores such as F1 scores.

It was important that all models were ran using the same hyper-parameters and layers so that comparison can be done on how well each model learns from the adjusted training data, as this allows for the ability to see how the removal, or the lack thereof effects generalisation and speed at which each model can learn from the texts it sees.

1.5 Research question

"How can modelling techniques be used to Analyse the Sentiment of Twitter Data?"

The aim of the project was to use modelling techniques through the Keras library to understand how machine learning was used to analyse noisy social media data. These techniques will be looked at, models will be trained and then evaluation will allow the ability to understand how using these techniques aids in the discovery of Sentiment within noisy data.

There is certain noise within social media data that needs to be considered when the cleaning of the data is done. For the purpose of this study, it was decided that numbers would be kept in any text, but things such as URLs or hashtags would be removed. This was done to maintain anonymity within the data being used, to comply with GDPR but also to make sure that the data cannot go back to any individual, as this is not the purpose of this study. External facing URLs also add no value to the data.

1.6 Report Structure

The remainder of this report follows the following structure. Section 2 contains the Literature Review that discusses the background of the field with exploration of the inception of AI, the first instances of Computational Linguistics, followed by Text Classification ending with Sentiment Analysis up to current day. Section 3 discusses the project development, talking through the process of obtaining the data, how the data was cleaned and tokenised to be used for the networks.

Section 4 discusses how the project was implemented, discussing the implementation style and how this was useful throughout, what results were gathered and how those results were evaluated further. Section 5 is the critical evaluation of the project with reflection on the findings and what has been produced. This section also aims to evaluate how the research could be improved and the next steps for this research area.

2 Literature Review

This section aims to discuss the technical background of the focus area, spanning from the early inception of Natural Language Processing (NLP), to Text Classification finishing with Sentiment Analysis. This section is split into three areas and makes sure to discuss what research has been done so far in the focus area to give background to this study.

2.1 Technical Background

Artificial Intelligence (AI) is an area of research that aims to demonstrate intelligence by machines that contrasts to the intelligence of humans and animals. The field of AI attempts to take what we have been trying to do for thousands of years, *understand how we think*, and show how we can use this knowledge to manipulate a world that is far larger and more complicated than we originally thought. The term AI was coined in 1956, but initial work in this field began after World War II. *(Russell and Norvig, 2016)*

One of the first forms of AI towards the approach of making it so that machines could *act humanly* was proposed by Alan Turing in October 1950 in his paper "***Computing Machinery and Intelligence"***. Turing proposed a test that he termed the *'Imitation Game'* which was later renamed to the *Turing Test*, that was a game played between a man (A), a woman (B) and an interrogator (C).

The purpose of this test was to answer the question '*Can machines think?*'. The object of the game was to determine if the interrogator could distinguish which was the man and which was the woman based on their responses to questions. They were labelled as X and Y to the interrogator. The questions were answered through type-written responses so that the interrogator couldn't guess who was answering simply based on the tones of their voice for example.

The initial tests saw a human playing the part of A and another human taking the part of B and acting as a helper to the interrogator C, but the question then became what would be the results if the human playing as A was then replaced by a computer? This created a new problem. It was important that the questions given to the computer were adjusted so that the computers responses didn't give away that it was a computer answering now rather than a human. *(Turing, 1950)*

The aim of the test was to see if a computer could be part of a conversation through the use of a teleprinter and if the computer could pass the test, then it had achieved the ability to imitate a human so efficiently that it could be perceived as being able to think for itself.

The computer passes the Turing Test if the interrogator is unable to tell if the respondent is a computer based on its responses. For a computer to have a chance to pass such a test, it must possess certain capabilities. These are *Natural Language Processing, Knowledge Representation, Automated Reasoning* and *Machine Learning*. *(Russell and Norvig, 2016)*

The Turing Test involved no physical interaction between the interrogator and the subject they were asking questions to. This was because Turing deemed physical simulation as something unnecessary for intelligence, but the test was expanded upon, known as the *Total Turing Test*, that involved objects being sent through a hatch to the participant, with the addition of a video signal. This was done to allow the interrogator to test the perceptual abilities of the participant. *(Russell and Norvig, 2016)*

The Total Turing Test would therefore require a computer to have the capabilities known as *Computer Vision* and *Robotics*. These 6 necessary capabilities are the 6 disciplines that AI is composed of. All these capabilities are known as the area of AI that allow a computer to be able to *Act Humanly* and for this, we owe Turing a lot of credit.

2.2 Natural Language Processing

2.2.1 Computational Linguistics

Natural Language Processing (NLP) is a topic within AI, and a form of Computational Linguistics with the aim to process and analyse meaningful phrases and sentences, in the form of natural language. The earliest work done in Linguistics was done by the Scientist Ferdinand de Saussure in the early 1900s, who came up with the concept known as *Language as a Science*. Although Saussure died before he published his work, two of his colleagues published this from his findings in 1916*. (Foote, 2019)*

The question brought forward by the study of Linguistics was *How does Language relate to thought?* In 1957 Noam Chomsky explained his theory in his short book titled ***"Syntactic Structures"***. Chomsky's theory was based on syntactic models that go back to the Indian Linguist Panini, but his theory was formal enough that it allowed for the ability to create programs that could test this theory.

Chomsky discussed in his book the idea of Phrase Structures, and how we use Natural Language for communication. Linguistic description at a syntactic level is formulated in terms of constituent analysis (parsing) *(Chomsky, 1957).*

The Phrase Structure tree is a physical representation of a sentence in the form of an inverted tree, with each node of the tree labelled according to the phrasal constituent it represents. *(Dictionary.com, 2020)*

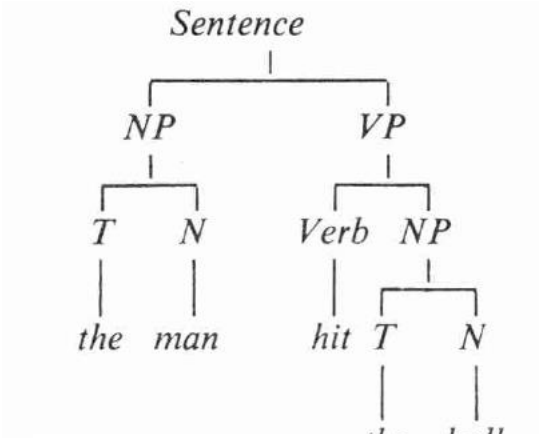


Figure 1 - Chomsky 1957 - Phrase Structure Tree

Phrase Structure became a type of grammar that enabled the ability for computers to understand languages, but this type of grammar structure had to be created to achieve this goal. The overall goal of this new grammar structure was to create a computer that had the ability to imitate the human brain in terms of how it could think and communicate.

Although there must be some consideration when it comes to using language to solve issues of generalisation within models. It is possible to observe that adjectives tend to come before nouns in English, but what about in a language such as French, where this is not the case. Phrase Structure is useful, but there are limitations to it where differences in the structure of a language can cause problems within another language, when it comes to allowing a computer to understand the relevance of the structure.

The first steps towards being able to teach machines to speak to humans or to understand humans, is to teach a machine the ordering of the language to be spoken, using these grammatical models of the phrase structure of sentences. It is useful to consider the structure of a sentence, but it is also useful to understand the syntactic categories that come from this structure.

The arrangement of words that make up a sentence is called its Syntax *(What is Syntax? Definition, Examples of English Syntax, n.d.)*. This is important because the structure of the sentence not only allows for its understandability for a human who is also using the language, but it is required if we expect a machine to be able to also compose sentences using this same structure. While the Syntax is necessary, Diction is also of use.

Simply, the Diction defines the choice of words, the combination of the two can lead to similar meaning sentences, that have a different structure to them while using the same words. E.g. "The Dog Barked Loudly" *and* "The Dog Loudly Barked". Although both phrases convey the same meaning, the Syntax is different in each phrase even if the diction is the same.

2.2.2 Machine Translation

Machine Translation is an automatic translation of text that is most used to change the language of one country into the language of another. It was first envisioned in 1949 by Warren Weaver in his paper. "*There is no need to do more than mention the obvious fact that a multiplicity of language impedes cultural interchange between the peoples of the earth, and is a serious deterrent to international understanding." (Weaver, 1949)*

The problem with translation is making sure we understand the underlying meaning of a word. It is sometimes not possible to know if a word means exactly what is written on the page. An example if we attempt to read words through an opaque mask with only a small hole, this way we can't see the word beside the one we are reading. A word such as "Fast" may mean "rapid" or "motionless", and there isn't really a way to tell which. *(Weaver, 1949)*

This is where the discussion for the need for machine translation started to become a topic of conversation in the 1960s. Around 1966, a committee of scientists known as ALPAC (Automatic Language Processing Advisory Committee), wrote a report called "***Languages and Machines***", which was meant to discuss the need for machine translation at the time. The report concluded that machine translation was more expensive than human translators who were cheaper and faster at the work.

In the mid-1960s it was clear that the need for machine translation was not cost effective, put bluntly in the report, "There is no emergency in the field of translation. The problem is not to meet some non-existent need through non-existent machine translation". *(ALPAC, 1966)*

2.2.3 Text Analytics

There was a gap in time from when the ALPAC report was released, the idea of doing research into NLP was considered dead by many, but things changed in the 1980s. With the introduction of expert systems and the new research philosophy coined as *deep learning*, the subject field of AI began to rise again. *(The History of Artificial Intelligence, 2017)*

With the introduction and popularity gained from these new machine learning techniques, using the combination of linguistics and statistics, it was possible to begin research in NLP again, but this time using a more statistical approach rather than the old machine translation approach. *(Foote, 2019)*

The biggest problem at this time was that data was unstructured, which meant that exploration was made difficult due to this. The answer to this problem would be Text Analytics *(BeyeNETWORK, 2020).* Text Classification (TC), sometimes referred to as *Text Categorisation*, is the continuation of machine translation work that began in the 1980s. "TC dates back to the early '60s, but only in the early '90s did it become a major subfield of the information systems discipline" *(Sebastiani, 2002)*.

TC is the task of taking a piece of text, and then separating it by deciding what predefined classes (or topics) this text might belong to. The process of assigning values for each text comes in Boolean

form, in the sense that we want to take a set of *predefined categories* and assign a domain of documents to each of these through true and false values.

The earliest forms of Text Classification would be in a slightly different form. Some of the first real work in text classification was email spam filtering, known as *Ham vs Spam*. This is a problem in supervised learning, which means a form of machine learning that involves the supervision of a human to develop learning.

2.2.4 Sentiment Analysis

Sentiment Analysis as a field of Research started to gain traction in 2002. The first days of this analysis was focused on movie reviews as the Corpus. Using these ratings online helped to kickstart the research topic. The first real paper based on this was written by *(Pang, Lee and Vaithyanathan, 2002)*, which is one of the first written publications to show that machine learning classification techniques can outperform human baselines.

Sentiment Analysis wasn't always about Social Media texts. On March 21st of 2006, the very first tweet was written by the founder Jack Dorsey *(Abell, 2011)*. As Social Media became more and more popular, it started to become apparent that the sentiment of these sites would become an area that would allow for the enhancement of the topic. Text mining enables the discovery of new patterns as well as existing relations and trends among various unstructured documents by methods, such as keyword mapping *(Aggarwal and Wang, 2011)*. This topic is expanded upon in section 2.3.4 onwards.

2.3 Previous Research

The aim of this section is to discuss research that has been done into text categorisation, which then led us toward the field of Sentiment Analysis.

2.3.1 Inverse Document Frequency (IDF)

Jones wrote a paper which looked at the idea of Inverse Document Frequency, although this name for her study was not penned until a bit later by others. Jones wanted to explore the specificity of index terms of texts to see if the idea of regarding these as independent had merit. She retrieved variations of term frequency and examined these to see if terms that occurred more frequently were an avenue for improvement in overall performance. She found that matching with term weighting led to substantial improvements in improvement over that of more simple term matching *(Jones, 1972).*

2.3.2 APRIORI Algorithm

Fürnkranz was one of the first to study the effect of using n-grams in the area of text categorisation. His paper aimed to explore n-grams features for text categorisation, which was done across 2 chosen domains. At the time of this paper machine learning algorithms had already been used previously, showcasing how these algorithms were useful to represent training data as feature vectors *(Fürnkranz, 1998).*

In this paper, Fürnkranz uses an APRIORI like algorithm to generate the features. The Apriori algorithm is an essential algorithm in data mining used for Market Basket Analysis, which is used for mining frequent itemsets and their occurrence *(A beginner's tutorial on the Apriori algorithm in data mining with R implementation, 2017).* The algorithm finds pairs of features, and then for each pair it finds the intersection of the position pointers of the two features, then discards the features that fall below the associated position pointers threshold.

His experimental setup here uses n-grams from lengths of one to five, with the use of a stoplist to lower the number of n-grams within the data, where a stoplist is a list of defined stopwords. An argument within text analytics is the inclusion or removal of stopwords, Fürnkranz mentions that removal of stopwords can cause some important information to be lost using this technique.

This algorithm and effective use of pruning with n-grams features resulted in very high accuracy in the two domains they were working with, which were the *21578 REUTERS newswire data* and the *20-newsgroup data*.

2.3.3 Bag-of-Words and N-grams

Mladenic also wrote a paper in the same year, involving the use of a new algorithm that would use bag-of-words and n-grams (see figure 2) with the aim to generate an algorithm that is effective at classifying text using these features. Although the use of Naïve Bayes as an algorithm for this seemed hopeful, they didn't get high results, mostly achieving 0.45 average in terms of Recall scores *(Mladenic, 1998).*

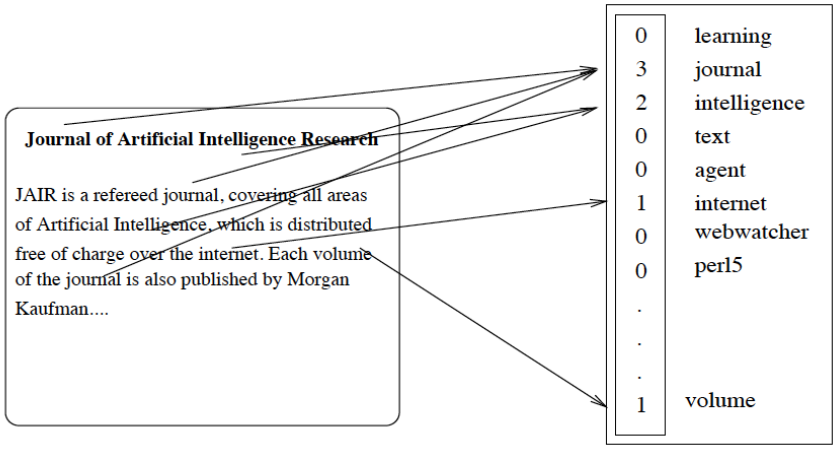


Figure 2 - Mladenic 1998 – Example of Bag-of-words representation using frequency word-vector

There was some work done involving the use of bigrams (two-words) to enhance text categorisation. This work was done by Tan, Wang and Lee. They proposed that the use of an efficient text categorisation algorithm that generates bigrams from the data would be good as features to be used within a *Naïve Bayes* classifier *(Tan, Wang and Lee, 2002).*

Fürnkranz discovered that n-grams of length 2 and 3 were the best lengths to be used for the task of text categorisation, so this was experimented with further by Tan, Wang and Lee. They created a bigram extraction algorithm that extracts bigrams with high information gain and high occurrence in the domain. They discovered that the use of bigrams helped to significantly enhance the performance of their experiment.

2.3.4 Sentiment Classification – New Beginnings

Pang, Lee and Vaithyanathan wrote a research paper title "***Thumbs Up? Sentiment Classification using Machine Learning Techniques"*** which many consider to be the baseline for Sentiment Analysis research in the field. The date of this paper is crucial, it released in 2002 but Sentiment Analysis as a subject only gained real traction in 2003, all thanks to the work within this paper *(Pang, Lee and Vaithyanathan, 2002).*

Before Pang and Lee ib id, text categorisation work was done on a few domains, but their work was one of the first to use movie reviews as the corpus for their research (corpus means a domain of data). They specified the use of this domain due to the amount of data available online in this domain that can be used for analysis. They explained the ease of a human to be able to distinguish a positive review from a negative review when reading the reviews, where they suspected the use of certain words would lean to the expression of strong sentiments. They asked two graduate students to independently choose good and bad indicator words that they would expect to find in movie reviews.

Their baseline results for 700 positive and 700 negative reviews showed accuracies of 58% and 64% respectively. Taking words from another human and combining this with stats taken from the domain led to high results of 69%. For machine learning methods, they explored 3 methods in their experimentation. These were *Naïve Bayes*, *Maximum Entropy* and *Support Vector Machines (SVM)*.

Using a combination of three-fold cross-validation and the three methods, they achieved good results. Their research used a combination of the machine learning methods with unigrams, unigrams and bigrams, unigrams with POS (Part-of-Speech tags), and adjectives and unigrams using positioning. POS tags are the fundamentals of a language, such as nouns and adjectives, this is discussed in the earlier section regarding Computational Linguistics.

The machine learning method that showed the best results in most areas of their experiments was SVM, with the best result coming from SVM using unigrams when analysing the presence of unigrams within the text. An important definition for this research is stopwords, which were not removed here, and yet they still achieved accuracies as high as 82%. Further work was discussed, relating to understanding next the *topic* of the reviews to see if that would improve accuracy metrics.

2.3.5 Subjectivity Extraction

Pang and Lee released a continuation of their work in their next research paper a couple of years later. The basis of this paper was the use of sentiment summarisation through minimum cuts to achieve higher base accuracy results *(Pang and Lee, 2004).*

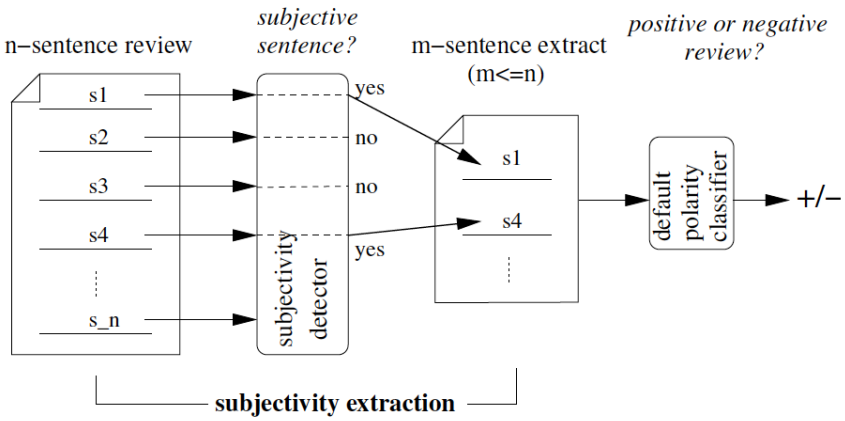
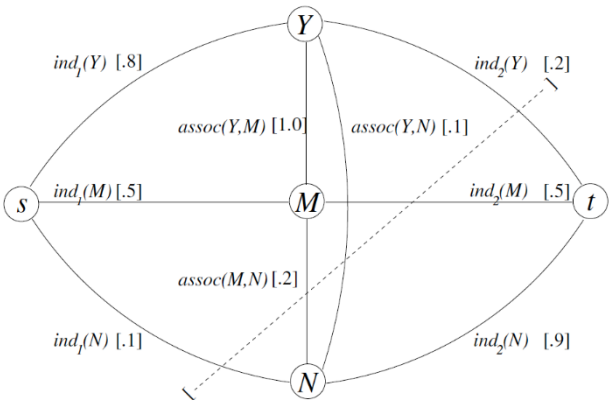


Figure 3 - Pang and Lee 2004 – Polarity Classification via Subjectivity Detection

This paper expanded on the idea of polarity to understand the sentiment of texts and began work towards looking more at the subjectivity of this and how the subjectivity influences the perceived polarity of text. The previous approaches have looked at the lexicons, such as 'good' or 'bad', but there is more to be explored to understand the polarity of texts than simply looking at the lexiconic words. Figure 3 shows the process of subjectivity extraction.

Their approach was aimed at looking at the texts and classifying them as subjective or objective, removing the objective texts and then using a machine learning classifier on the resulting extracts. They formulated an alternative strategy to this, that didn't involve feature engineering, as had been done before it. They used the idea of cut-based classification to extract information from the texts using *individual scores* and *association scores*.



C_1	Individual penalties	Association penalties	Cost
{Y,M}	$.2 + .5 + .1$	$.1 + .2$	1.1
(none)	$.8 + .5 + .1$	0	1.4
{Y,M,N}	$.2 + .5 + .9$	0	1.6
{Y}	$.2 + .5 + .1$	$1.0 + .1$	1.9
{N}	$.8 + .5 + .9$	$.1 + .2$	2.5
{M}	$.8 + .5 + .1$	$1.0 + .2$	2.6
{Y,N}	$.2 + .5 + .9$	$1.0 + .2$	2.8
{M,N}	$.8 + .5 + .9$	$1.0 + .1$	3.3

Figure 4 - Pang and Lee 2004 – Graph for classifying three items, table shows examples of how cuts were made

"Brackets enclose example values; here, the individual scores happen to be probabilities. Based on individual scores alone, we would put Y ("yes") in C1, N ("no") in C2, and be undecided about M ("maybe"). But the association scores favour cuts that put Y and M in the same class, as shown in the table. Thus, the minimum cut, indicated by the dashed line, places M together with Y in C1" (Pang and Lee, 2004, figure 4).

Pang and Lee ib id stuck with movie reviews as their corpus, using an additional subjectivity dataset that was used to train their detectors. Using their default polarity classifiers and updating them with the subjectivity dataset later, they were able to extract and train on a more subjective dataset to achieve sentiment analysis results. They used *Naïve Bayes and SVM* as their machine learning models of choice and achieved on average around a 2-5% increase on accuracy when using subjectivity over polarity as was used in the past.

2.3.6 Continuous Space Use

Schwenk looked at the idea of "***Continuous Space language models"***, where they used neural networks towards classifying spoken language. The idea of this research was to "*attack the data sparseness problem by performing the language model probability estimation in a continuous space."* The paper discussed and reviewed models using three possibilities to increase the performance of the model. These were increasing the size of the hidden layer, training several networks and then interpolating them together, and the use of large projection layers (Schwenk, 2007).

2.3.7 Using Social Media for the task

Barbosa and Feng classified the subjectivity of tweets (posts on Twitter) based on traditional features with the inclusion of some Twitter-specific clues such as retweets, hashtags, links, uppercase words, emoticons, and exclamation and question marks. This was done through a 2-step sentiment analysis classification method, that classifies messages as subjective or objective as was seen before, and then further distinguishes these as positive or negative in their polarity (Barbosa and Feng, 2010).

The idea is to detect sentiment from Twitter texts using a robust classification detection method that can detect and still be effective through biased or noisy data. The idea of noisy data is data that has a lot of information within that is not useful towards classifying the sentiment. The goal of their

methodology was to do something like what Pang and Lee did, taking tweets and classifying them into positive and negative, and now adding neutral as a classification option.

Although Pang and Lee did their classification work on larger pieces of texts that involved the use of n-grams up to a length of 5, Barbosa and Feng focused and worked on much smaller pieces of texts, mostly unigrams or bigrams (length of 2). The paper uses a POS dictionary taken from *http://wordlist.sourceforge.net/pos-readme* to improve upon the use of these types of tags that had seen some positive results in earlier papers, with the intuition being that certain POS tags are useful and good indicators for sentiment tagging.

They worked with three data sources, with the highest number of tweets data being at a total of 254081, although the chance of all of these being unique is not high due to the inclusion of retweets. Syntax features of the tweets were used here to compose the features for training. The most important classifier they used was a polarity classifier.

This work was a good step towards sentiment analysis of social media data, which is generally noisy anyway, but the results achieved were good, but not at all state of the art when compared to results achieved by more composed clean data coming from movie reviews.

2.3.8 Neural Networks for Machine Learning

Mikolov was the first modern approach to explore the variations of neural networks that could be used for machine learning within data science, and observed the differences that can be had by training with Recurrent Neural Networks, and compared this alongside a Maximum Entropy model to see how they performed against each other *(Mikolov et al., 2011).*

The premise of this is how the idea of weights can improve the performance of neural networks when working with large quantities of data. It is important to note that the premise of deep learning itself didn't become popular again until 2013. Mikolov trained some of the largest neural networks to that date, having 400million tokens, a hidden layer with 640 neurons with a vocabulary size of 84k words.

Weights within a neural network represent the strength of the connection between units. If there is more weight from node 1 to node 2, then it means that neuron 1 has higher magnitude and therefore greater influence over neuron 2, and vice versa. A weight is important as it brings down the importance of the input value.

2.3.9 Tensor Networks and Vector Space

Socher worked with Recursive Neural Tensor Networks (RNTN) looking at semantic compositionality over a sentiment treebank (see figure 5). Richard Socher used these networks to explore the sentiment treebank, which is a corpus with fully labelled parse trees which allow for complete analysis of compositional effects of sentiment in language *(Socher et al., 2013).*

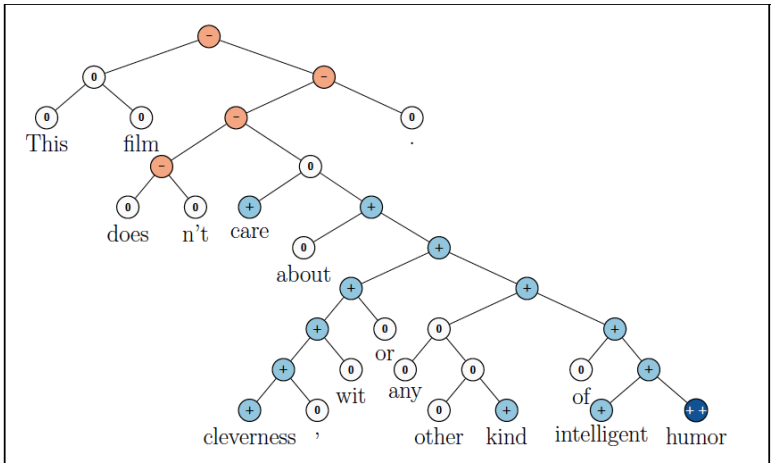


Figure 5 - Socher et al. 2013 – Example of a Recursive Neural Tensor Network

The image here shows an example of a RNTN and how this one looks at the words and classifies each word as positive, negative or neutral in sentiment. This network does this word by word, which is unigram analysis of the sentence, a word by word representation.

Semantic vector spaces are spaces that are composed of single words, which the research of Richard Socher focused the most on. Using 3 types of recursive neural networks, RNTN, RNN and the Matrix-Vector RNN (MV-RNN), the results showed that the network making use of the new tensor network achieved better results compared to the others, and actually achieved state of the art results for its time.

The problem with vector space work is that it's not possible to accurately signify the meaning of the sentiment and can thus cause lower accuracies than are wanted. This is evident as the MV-RNN shows lower results than the other RNNs used here, although it is very tight, it still comes up shorter than the rest, even if only marginally.

Saif explored the effect of Stopwords on Data Sparsity within Sentiment Analysis and found that although it decreased the feature space significantly, the effect on performance was marginal regarding the accuracy of classification. Sparse data shows many gaps throughout and this can be a problem in NLP, as there is possibility to learn things that can be deemed useless when it comes to the final achievement, although having Sparse data is not necessarily the same thing as having missing data in Big Data *(Saif et al., 2014).*

2.3.10 Deep Learning

Cho introduced the concept of Gated Recurrent Units (GRU), which is a recurrent layer that has few parameters and a forget gate within. The performance of GRU in certain tasks proved to be good and have good performance when the dataset is small. Cho used RNNs with an Encoder-Decoder style, that allowed for the learning of phrase representation (Cho et al., 2014).

Adding to the use of Neural Networks, Shirani-mehr explored Deep Learning to explore Sentiment Analysis using Movie Reviews. This study made use of a new type of neural network that gained prevalence in Image classification known as a Convolutional Neural Network (CNN). This work used the Sentiment Treebank that was used before by Richard Socher, but this time used neural networks without a Tensor layer, and then the results were compared against a Naïve Bayes model *(Shirani-mehr, 2015)*. Figure 6 shows the structure of a CNN.

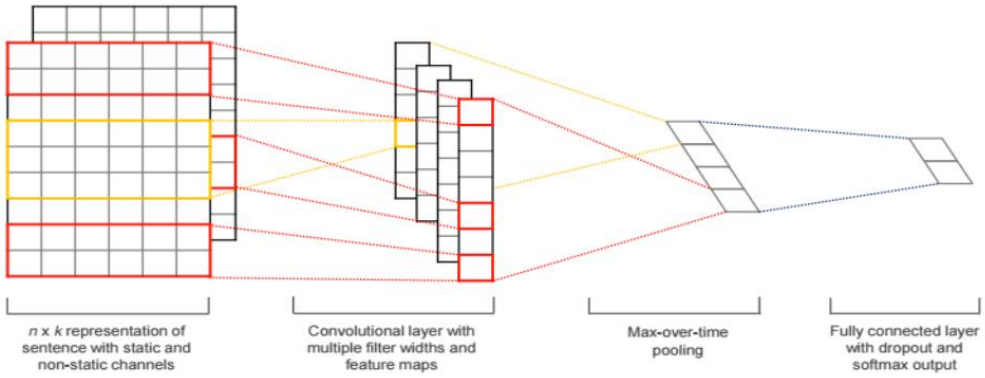


Figure 6 - Shirani-mehr. 2015 – The structure of a Convolutional Neural Network

The findings result in higher base test data accuracy against all neural networks tested when the word2vec dictionary for word embeddings is used. Word Embeddings allow you to insert words into the vectors without necessarily needing to train the words beforehand, which in the case of this study resulted in slightly higher base accuracy on the testing set, which is data that the model has not seen before.

November 9th, 2015 was a big date in deep learning, signally the release of the Tensorflow Library by Google. Abadi released a paper in 2016 (preliminary date is Nov 9th 2015) that was written more as an instruction guide into what Tensorflow enabled analysts to do with the library and how they could make use of it for deep learning *(Abadi et al., 2016).*

The paper explored and defined the programming model of the Tensorflow library and the basic concepts, along with some advanced concepts showing how the library worked, and how it could be used for big data and more advanced neural networks that used Tensor layers.

2.3.11 Word Embeddings

Yu explained and explored word embeddings in more detail, aiming to refine them in his paper. Word embeddings are great in a lot of cases when it comes to capturing semantic or syntactic information from contexts, but they have been used extensively over the time of NLP, and Yu wanted to look at a way to further refine the use of them *(Yu et al., 2017).*

Yu compared two-word embeddings against each other, which were the GloVe and Word2Vec word embeddings. Using his refinement model that required no labelled corpus to be used, he yielded higher results than both more conventionally used word embeddings and in addition, allowed for higher performance when this refined model was/is used with neural networks.

2.3.12 Big Data and Convolutional Networks

Sohangir wrote a paper that explored Big Data in a financial setting, making use of Deep Learning and Big Data Analytics, which are now thought of as two focal points of data science. The concept of Big Data has gained more interest and traction as something businesses could take advantage of, this interest was increased due to the usability of Tensorflow *(Sohangir et al., 2018).*

Sohangir found through this analysis that more features led to higher accuracy alongside a higher f1 score, signalling that keeping features high is a good way to increase the accuracy of the model when it comes to the evaluation. Sohangir made use of CNN networks with Tensorflow and found good results using this for big data sentiment analysis alongside the more traditional methods, also making use of Recurrent Neural Network layer choices of Long Short-Term Memory (LSTM) recurrent layers.

2.4 Conclusion

The early days of research in the field has shown promise and natural progression over time. It is clear from its inception that AI never stands still, it is an ever-growing science and even today, the area of Machine Learning is growing at an incredible rate. Progression from the Turing test, towards Linguistics leading us ever closer to machines that can begin to think more for themselves is a promising avenue to be involving in, as a lecturer, scientist or even a hobby enthusiast.

The tools available allow for anyone, even those without prior knowledge to start to get involved and help toward the growth of the field. Sentiment Analysis is one of the most promising areas of the overall field, and with the help of Google and their advanced Machine Learning tools, things will get better as the years go by.

3 Technical Development

This section will aim to explain the stages that were taken towards the analysis of the data, exploring extraction and tokenisation. These are the tools needed to create this project, laid out in a step-by-step approach that explains the need for each of the tools.

3.1 Narrative Description

The project itself involved many things that were unknown to the developer before undertaking the project and these had to be learnt to make significant steps towards project completion. A lot of the things involved were not studied in the first two years of the BSc Computer Science Degree at the University of Hull, which initially created some worry regarding ability to complete the project on time. Significant techniques were required to be learnt for this project, some of which were discovered within a summer school the developer partook in abroad in advance and in the Data Mining & Decision Systems module that was part of the year 3 criteria, which helped to fill in the blank spots in regard to how machine learning algorithms work. The developer was fortunate to have chosen this module in advance, as this led to a lot of skills being taught to them in the preliminary planning stage of this project. This module helped to be able to build an understanding of the different concepts required. The developer particularly found the inclusion of a practical machine learning classification problem to be the biggest help in understanding the premise of Neural Networks. The knowledge gained from the summer school experience, alongside experience gained from self-study allowed the developer to build a set of skills that helped to problem solve and come up with a completed project that far exceeded accuracy targets.

3.2 System Requirements

The development of the project makes use of a variety of libraries within the Python language. As the project was a research question, the software used involves libraries to speed up the process of certain elements alongside libraries that help to enhance workload and achieve good results due to the capabilities. Due to the nature of the project, there is no independent software solution alongside the Jupyter Notebook.

3.2.1 Python

The project programming language used was Python. This was due to the vast quantity of great libraries that can be used for Deep Learning and Machine Learning, and due to the ease of use of the language towards the task. First are the libraries available to the framework, which involve some popular ones such as Numpy, Pandas and Tensorflow. These libraries allow for clear coding facilitation and made understanding what was going on within the project's coding element easy to follow. There was also the simplicity of the language which made it a popular choice for this task.

(NumPy — NumPy, 2019), (pandas - Python Data Analysis Library, 2019), (TensorFlow, 2019)

3.2.2 Jupyter Notebook

Python was the chosen codebase, but an IDE was also recommended. Jupyter Notebook is a web-based open-source application that allows for coding within blocks that makes organising and running of code both simple and easy to read. It was useful to be able to run certain sections of code whilst also leaving other parts untouched, as this allows a model to be trained and the results to remain visible, but it was possible to save results since the code block that contains the results doesn't need to be ran again afterwards.

```
In [263]: def remove_url(txt):
              #Replace the URL with nothing but a space
              """Replace URLs found in a text string with nothing
              (i.e. it will remove the URL from the string).

              Parameters
              ----------
              txt : string
                  A text string that you want to parse and remove urls.

              Returns
              -------
              The same txt string with url's removed.
              """

              return " ".join(re.sub("([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", txt).split())

In [264]: def analyse_sentiment(tweet):
              analysis = TextBlob(tweet)

              if analysis.sentiment.polarity > 0:
                  return 1
              elif analysis.sentiment.polarity == 0:
                  return 0
              else:
                  return -1
```

Figure 7 – Visual Example of Jupyter Notebook Code-blocks

Figure 7 shows how it was possible to see code blocks within Jupyter notebook. The code number to the left helps to indicate the order of which the code block had been ran. In the figure, the blocks are in order, but it is possible to run code out of order, which can sometimes cause problems such as declaring a variable in a code block later than it is first used causing an error. There was also the ability to copy and paste code blocks, which allowed for the ability to run a different model with slight variations in parameters, or one with a different dataset without needing to rewrite all the code, therefore it was possible to have results shown for multiple models allowing for easy comparison of results achieved.

3.2.3 Tweepy

This project involved the use of noisy social media data. The data used for the project needed to be of a large quantity (that was possible with the given constraints) and unique to a chosen topic. Brexit was a big point of interest within the United Kingdom, so this was the chosen topic for this research.

Twitter is a popular site that has millions of tweets published on its site per day and is also a platform that has been involved in Sentiment Analysis in the past. For this reason, Twitter was chosen here as the source. To collect the data, a scraping library would be needed, the one used was called Tweepy. To use Tweepy, it was a requirement to have access to the Twitter Application programming interface (API) which is an interface that specifies how someone can connect to and use data from the Twitter internal system. For this, developer access would be required, which required an application form to gain access.

```
#Define your App access codes to access the API
auth = tw.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)
api = tw.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)
```

Figure 8 – Defining the Twitter API access credentials

Figure 8 shows the formatting that was required to be able to access the Twitter API and begin the process of taking tweets. The most important thing that the developer account gives to the user for the collection of the data are the authorisation details seen in figure 8, gained from registering an app on the developer section of the site. These authorisation parameters were set to find tweets with the topic of Brexit, filter out all retweets to force no duplicate tweets to be taken. The chosen volume of tweets to take was set at 100,000. In the end, 65,759 full-length tweets were obtained and then saved to a text file once URLs had been removed in the first instance.

3.2.4 Keras and Tensorflow GPU

Using Keras requires Tensorflow, as Keras is simply a library that runs on top of Tensorflow. Keras has a suite of options available to allow for the creation of neural network models. This project used a significant amount of data, which is why Tensorflow GPU was the chosen flavour of choice for the kernel to be ran within Jupyter Notebook.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_embedding (Embedding)  (None, 115, 8)            531384
_________________________________________________________________
gru_1 (GRU)                  (None, 115, 32)           3936
_________________________________________________________________
gru_2 (GRU)                  (None, 115, 16)           2352
_________________________________________________________________
gru_3 (GRU)                  (None, 115, 8)            600
_________________________________________________________________
gru_4 (GRU)                  (None, 4)                 156
_________________________________________________________________
dense_1 (Dense)              (None, 10)                50
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 33
=================================================================
Total params: 538,511
Trainable params: 538,511
Non-trainable params: 0
```

Figure 9 – A Keras Sequential Model created in Jupyter Notebook

Tensorflow GPU allows for the ability to make use of the processing power of a GPU. Figure 9 shows a Recurrent Neural Network Sequential model created within Keras. It was possible to explore the size of the network here and see the amount of expected trainable parameters that would be entered into the network from the dataset from this summary.

3.3 Data Extraction

This section discusses the process of data extraction, the exploration of the data, the tokenisation procedure and aims to discuss the useful findings from the data.

3.3.1 Exploring the data

The data was taken from the Twitter API using Tweepy. Once extracted, the data could be saved to a text file and then explored further. The process of taking the data from the text file and adding this to a data table involves cleaning of elements that are unnecessary to be included. One of the key aspects of data exploration is the cleaning of the data. Hashtags were removed as they serve a purpose within twitter to define trends of the day, or trends of the week, but have no purpose here.

Figure 10 - Visual example of trends on Twitter via Twitter.com

Figure 10 shows how hashtags define the trends on Twitter. These can be trends within a single country or from around the world. This is dependent on user location, as to what trends will be visible to them. Something that was important regarding the data that was extracted, was that there was no polarity connected to anything that was extracted. The main basis of the projects Sentiment Analysis was the training of models to gain the ability to recognise and associate words with polarity measure.

To add the Polarity, a library called TextBlob was used. Shown in figure 7, a function is created that uses TextBlob to analyse each individual tweet and then based on the numerical outcome, assigns a value for each tweet. Once the polarities have been created, a data table is made, placing the tweet into one column and the polarity values into the other, allowing for the base of the training data. The data table that was created can be seen in figure 11 below.

```
dfObj = pd.DataFrame(brexit)
df = pd.DataFrame(sentiment)

df = df.rename(columns={0: 'polarity'})

brexit_data = pd.concat([dfObj, df], axis=1)
brexit_data = brexit_data.rename(columns={0: 'tweet'})
brexit_data.head()
```

	tweet	polarity
0	If you re a Brit amp you thought Boris amp Bre...	-1
1	AndrewBowie4WAK 14m voted for your Prime Menda...	-1
2	Morte all EUroditttatura Brexit gt Italexit	0
3	Well done Boris We leave the EU 31st Jan 2020 ...	0
4	This Christmas BorisJohnson promises the gift ...	0

Figure 11 - Creating the Polarity Data table

3.3.2 Exploring Findings

3.3.2.1 Polarity Values

With 65,759 tweets in total, it was important to have an idea how the polarity of Positive, Negative and Neutral was spread. By implementing a Python value_counts() function onto the data table, it was possible to visualise this.

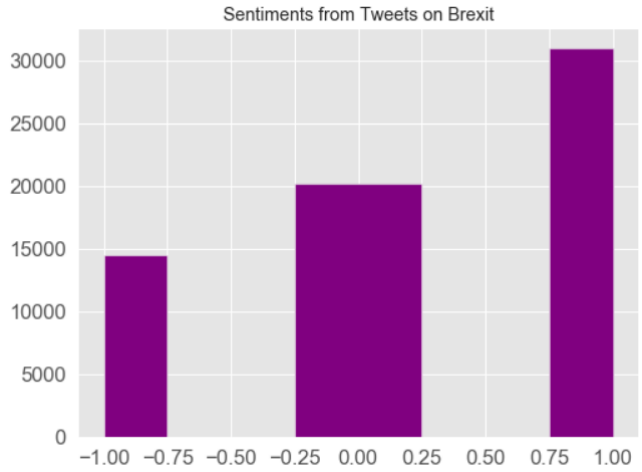


Figure 13 - Graph showing the spread of Polarity across the data

```
 1    31024
 0    20250
-1    14485
Name: polarity, dtype: int64
```

Figure 12 – Polarity values across the data

In the case of figure 12, the value of 1 corresponds to Positive, the value of 0 corresponds to Neutral, and the value of -1 corresponds to Negative. There were more Positive polarity tweets than there were Neutral and Negative, which have a total of 34,735, with the Positive polarised tweets equating to 47.17% of the data. Figure 13 shows an easier to read graphical version of the data spread in the form of a bar chart, which allows for the ability to see that there was a higher overall amount of positive polarity tweets. It was interesting to see if this near 50% spread of values towards positive would have an impact on the bias of the models when trained on this data. If that was the case, then a method to flatten the data to a more even spread would have had to be enforced.

3.3.2.2 Common Words Amongst Tweets

Some exploration of the words found within the tweets taken was necessary, as to see what words were the most common. The Polarity value that was assigned to each tweet was based on the overall tweet itself, considering all words within. If a tweet had a lot of negative words within then this frequency can cause bias which in turn can lead to issues in incorrect polarity assignment.

Figure 15 shows that the second most common word in all tweets gathered was ‘Brexit’, whilst figure 14 shows the numerical counts for the values in figure 15. What is important here is that this was the topic word, which was used to extract all the tweets as the ‘*collection word*’. With the hashtag now removed, #Brexit became Brexit. The most common word was ‘the’, not really a word that was useful in respect to discovering polarity of a sentence, but more exploration showed that a lot of the most common words were *‘stopwords’*. These are the words that are used to interconnect the language so that it is more understandable, but these words have no real meaning besides that.

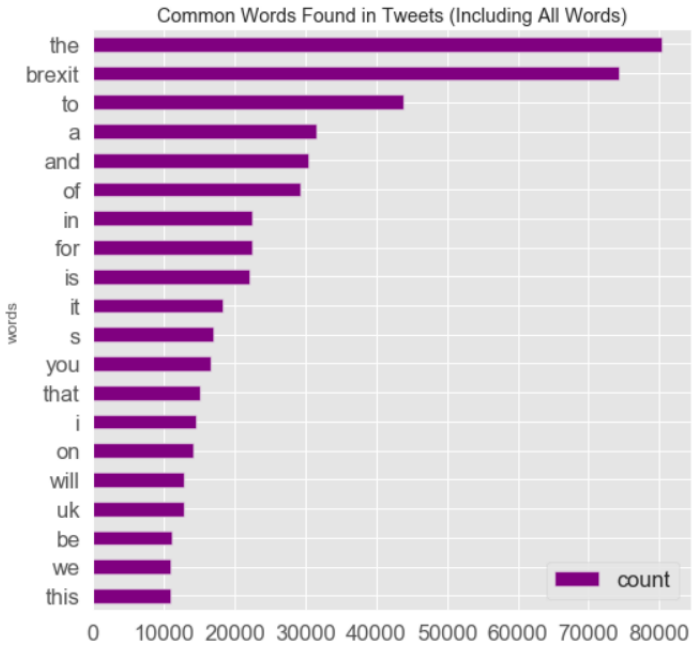


Figure 15 – Representation of common words found in tweets

```
[('the', 80423),
 ('brexit', 74216),
 ('to', 43849),
 ('a', 31455),
 ('and', 30432),
 ('of', 29320),
 ('in', 22439),
 ('for', 22437),
 ('is', 22078),
 ('it', 18366),
 ('s', 17007),
 ('you', 16627),
 ('that', 15128),
 ('i', 14593),
 ('on', 14208)]
```

Figure 14 – Numerical values for the first 15 common words found in all tweets.

To be able to collect and explore each individual word like this, the first thing that had to be done was to take all tweets, make them lowercase and then split them all by the spaces in between words into a list so that any duplicate words were put together, such as Pull and pull. If the words were not made lowercase before splitting them, then Pull and pull would have been considered different words because one is capitalised and the other is not.

3.3.3 Removing Stopwords & Collection Words

This section covers the removal of stopwords and collection words from the data and the reasoning behind taking this approach.

3.3.3.1 The NLTK Package

The Natural Language Toolkit (NLTK) is a suite of libraries and programs that help with the goal of symbolic or statistical natural language processing. NLTK was used for its stoplist library, its English stoplist. The purpose of this stoplist was to remove all the stopwords.

3.3.3.2 Using a Stoplist

To use a stoplist, it must be called from the NLTK library. To do this, the first thing that was needed was to set a local variable with the list of all words found in the English stoplist. Once this was done, then a comparison for loop with an enclosed if statement could be created which checked to find out if any words found within a tweet were present in the stoplist, and if they are, they should be passed over and not be added back to the original. Figure 16 shows this.

```
#Remove the stop words from each tweet list of words
tweets_nsw =[[word for word in tweet_words if not word in stop_words]
             for tweet_words in words_in_tweet]
```

Figure 16 – for loop to remove stopwords

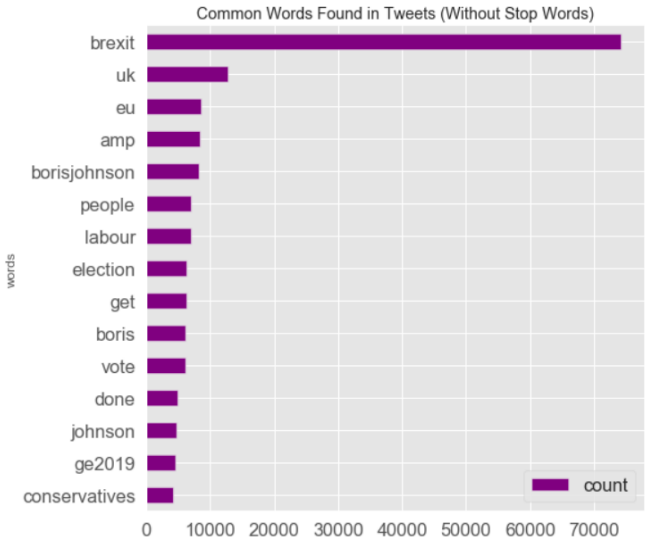


Figure 18 - Representation of common words found in tweets, stopwords now removed

```
[('brexit', 74216),
 ('uk', 12777),
 ('eu', 8505),
 ('amp', 8438),
 ('borisjohnson', 8254),
 ('people', 6994),
 ('labour', 6979),
 ('election', 6200),
 ('get', 6191),
 ('boris', 6144),
 ('vote', 6040),
 ('done', 4862),
 ('johnson', 4613),
 ('ge2019', 4566),
 ('conservatives', 4104)]
```

Figure 17 - Numerical values for the first 15 common words found in all tweets, excluding stopwords.

Figure 18 shows the results of the removal of stopwords from all the tweets, whilst figure 17 shows the numerical values of figure 18. Other than some expected words that relate to the topic such as UK and EU being amongst the most common, the word Brexit remained highly represented as the most common by a huge margin and must be removed as well to show a more even and well represented spread of common words. The other outlier in question was 'amp' which is the 4th most common word in figure 18. Amp is what Twitter uses to encode the & symbol. When extracting the tweets, the & sign was not correctly extracted, thus we are left with the unencoded version of this. As this is not really a word and thus not useful to have, this was removed alongside the word Brexit.

3.3.3.3 Collection word removal

The removal of the collection word follows the same process as that of removing stopwords except this time instead of having a predefined list to use, a list was created instead.

```
collection_words = ['brexit', 'amp']
tweets_nsw_nc = [[w for w in word if not w in collection_words]
                 for word in tweets_nsw]
```

Figure 19 – for loop used to remove collections words from all tweets

Figure 19 shows the for loop that checks and only keeps words that don't exist in the collection words list. The word 'brexit' was added to the collection word list, alongside 'amp' which was specified before to be the & symbol simply in an unencoded form. With this completed, the results can be seen in figures 20 and 21. The removal shows now that the spread of common words was more subjective towards the topic, with the values of the common words being more closely represented.

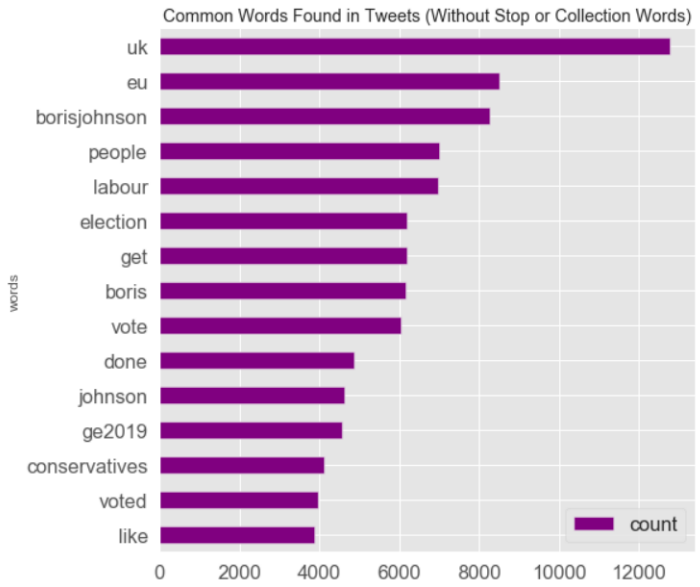


Figure 20 – Representation of common words found in tweets, cleaned

```
[('uk', 12777),
 ('eu', 8505),
 ('borisjohnson', 8254),
 ('people', 6994),
 ('labour', 6979),
 ('election', 6200),
 ('get', 6191),
 ('boris', 6144),
 ('vote', 6040),
 ('done', 4862),
 ('johnson', 4613),
 ('ge2019', 4566),
 ('conservatives', 4104),
 ('voted', 3957),
 ('like', 3877)]
```

Figure 21 - Numerical values for the first 15 common words found in all tweets, excluding stopwords and collection words.

3.4 Tokenisation

This section will discuss the process of Tokenization, and why it was an effective tool in shaping the data for model implementation. Once data cleaning was completed, it then needed to be transformed. The objective was to run the text through a Neural Network, but a problem lies within the solution, which is that neural networks can only understand numbers. To solve this problem, the data to be entered must be transformed into numbers so that it is recognisable to the network that is to be created. This process is known as Tokenisation. Keras has a pre-processing library for this task called Tokenizer.

3.4.1 Creating the Tokenizer

The first step of tokenising the words was creating a tokenizer variable that is saved into memory. The number of words to be created is set to 50,000 and the split is set to a blank space, as all words are separated only by spaces. Once this is defined, the tokenizer is fit onto the texts, which means that the words from the starting data are implemented into the tokenizer variable, which creates a library. To find out how many unique individual words there are, it's possible to check the length, which told us that there are 66,422 unique words present within the tokenizer library.

3.4.2 Creating three sets of Data

For effective training of networks and to allow for good comparisons later, three datasets were trained and tested on. The datasets involved texts with no stopwords and collection words which was the clean data, texts that remained uncleaned, and texts that lack the Polarity value of neutral. To remove the neutral polarity value effectively from the clean data, a set of three new values were created and added to a new table along with the tweets and polarity value.

The clean data with the neutral polarity was used to train the first builds of the models, later expanded to include unclean data. These three values are Positive, Negative and Neutral, and contain a 1 when the value is not empty, and a 0 when the value is empty. Figure 22 below shows

the process of filling the data table with the new entries, whilst figure 23 shows how this looked when completed.

```
pos = []
neg = []
neut = []

for l in brexit_data_stop.polarity:
    if l == 0:
        pos.append(0)
        neg.append(0)
        neut.append(1)
    elif l == 1:
        pos.append(1)
        neg.append(0)
        neut.append(0)
    elif l == -1:
        pos.append(0)
        neg.append(1)
        neut.append(0)

brexit_data_stop['positive'] = pos
brexit_data_stop['negative'] = neg
brexit_data_stop['neutral'] = neut

brexit_data_stop = brexit_data_stop[['tweet', 'polarity', 'positive', 'negative', 'neutral']]
```

Figure 22 – for loop to create new data table values for polarity

	tweet	polarity	positive	negative	neutral
0	If you re a Brit amp you thought Boris amp Bre...	-1	0	1	0
1	AndrewBowie4WAK 14m voted for your Prime Menda...	-1	0	1	0
2	Morte all EUrodittatura Brexit gt Italexit	0	0	0	1
3	Well done Boris We leave the EU 31st Jan 2020 ...	0	0	0	1
4	This Christmas BorisJohnson promises the gift ...	0	0	0	1

Figure 23 – data table with added polarity values on uncleaned data

3.4.3 Filtering out Neutrals

It is possible to do some filtering to remove the neutrals, which allows for the ability to create a dataset which excludes the neutral polarity completely. This removal was due to past research involving only Positive and Negative polarity, as the removal of Neutral allowed for the ability to test and compare to see if the inclusion of the Neutral value had any impact on the results.

```
# Make another copy to maintain neutrals
brexit_data_withneutral = brexit_data.copy()
#Dropping neutrals column
brexit_data.drop(brexit_data[brexit_data['polarity'] == 0].index, inplace=True)
```

Figure 24 - filtering out neutral values

Figure 24 shows the process of filtering. A copy of the data was created, so that when the neutrals were removed, the data with neutrals present wasn't lost. Once the copy was created, a drop function was called to drop all polarity values that had a value equal to 0, which indicates a neutral polarity. Inplace = true was used at the end to indicate to python that the result of the query should be done on the current data, if inplace = false is used here, python would return a copy of the data rather than edit the original data. The results of the filtering are shown in figure 25, which shows that there were no longer any values that equal to a polarity score of 0.

```
# Show the dataframe with neutrals removed
brexit_data.polarity.value_counts()
```

```
 1    31024
-1    14485
Name: polarity, dtype: int64
```

Figure 25 – the result of filtering out neutral

3.4.4 Tokenizing and Padding

To use the data in Neural Networks, there was a need to create holdout data. Holdout refers to data that is not seen by a Machine Learning model during its training, and is typically in the form of test data, split away from the training data. Three code blocks were created to allow for the easy ability to switch to a different dataset, to use only the data as the test and train data whenever it was needed. This strategy was used when training different models at different times and thus saved the need to rewrite code.

```
x = tokenizer.texts_to_sequences(brexit_data_withneutral['tweet'])
x = pad_sequences(x)
#y = polarity_all
y = brexit_data_withneutral[['positive', 'negative', 'neutral']]
#y = brexit_data_withneutral['polarity']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=0) #neutral class s
till within
```

Figure 26 – code block sequence to create train and test split on the clean data with neutrals

Another important aspect of a Neural Network is that all data within the layer must be the exact same length, if anything is a different length then the model will complain and will not run. It is possible to make all data the same length through a technique known as 'Padding'. Figure 26 shows the creation of the train / test data through padding.

The tokenizer texts_to_sequences pre-processing function was defined to make sure that only the words that are known by the tokenizer were used. This can be seen inside the brackets of the function, where it was specified to use the words that the tokenizer knew from the data found in the 'tweet' column. After this was done, the sequences defined inside x were padded.

```
array([    0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,    20,
         917,     9,  5487,     5,  2575, 13158,     2], dtype=int32)
```

Figure 27 – a pre-padded tokenized tweet

Figure 27 shows what a padded tokenized tweet looks like. The value of y is what is known as the target variable. This is what we want the model to guess, and how many times the model learns and tells us the correct answer, is defined as accuracy. There are two strategies when it comes to Padding, these are known as Pre-Padding (beginning) and Post-Padding (end).

Pre-Padding was the technique chosen in this instance. This was because when using RNNs or CNNs, we want to save the final output or the hidden state and pass it along to make our predictions.

If Post-Padding is used then when the model gets to the end of the data sequence, the final hidden state would get flushed out as most of the time this will be 0, but using Pre-Padding will mean that the hidden state will be correct and this will be effective for making predictions.

3.4.5 Tokenizer Inverse Map

Being able to tokenize words is great, but there also needs to be a way to convert text back if we are evaluating and want to see what sentences are wrong, for example. This was achieved by creating an Inverse Map function that when called, converted the selected tokens and returned the unpadded original text value.

```
idx = tokenizer.word_index
inverse_map = dict(zip(idx.values(), idx.keys()))

def tokens_to_strings(tokens):
    #Map from tokens back to words
    words = [inverse_map[token] for token in tokens if token != 0]

    #Concatenate all words
    text = " ".join(words)

    return text
```

Figure 28 – function to change tokens back to words

```
x_train[10]
```

```
array([    0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,   834,   450,    20,   404,   244,   172,
         235,    14,  1469,    10,   440,    14,    83,    21,   133,
          32,    11,     1,   446,  1153,    10,    11,   477,    95,
       43723,     2, 43724, 43725, 43726, 43727,  5931], dtype=int32)
```

```
tokens_to_strings(x_train[10])
```

```
'started making this stupid thing last year i finished it tonight i don t know what s the point anyway it
s called british gothic brexit leagueofgentlemen reeceshearsmith stevepemberton americangothic illustratio
n'
```

Figure 29 – A tokenized tweet before and after being reversed through an Inverse Map

Figure 28 shows the functions created to accomplish this. To reverse the text, the function takes the input and checks for all values that don't equal to 0, 0 being an empty value which has been added during padding. If the value equals 0, then it is ignored, otherwise it will use the number to find the textual value associated with the number from the words variable. This holds all the words and their token values and displays the text in the place of the number, thus reversing the tokenisation. Figure 29 shows the result of this function call.

4 Project Implementation

This section will discuss the way the models were created and adjusted towards the methodology. Hyperparameters saw adjustment over time to attempt to find the best choice. This section will also cover the process of evaluating a model after it was completed, what needed to be considered for the model to be kept, and how the use of the 3 different datasets allowed for incredible opportunity for evaluation toward finding the best model for the task. The section will conclude with a discussion regarding embedding weight exploration, the creation of unique data and the process therefore of creating metric scores and confusion matrices for final evaluation of results.

4.1 Choosing Neural Networks

4.1.1 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are networks that instead of using a feed forward approach where once the hidden layer learns a weight it passes that weight to the output layer, instead the hidden layer recurs and relearns from the weights and adjusts them for the number of times the return sequence is set to, which in this case is 4.

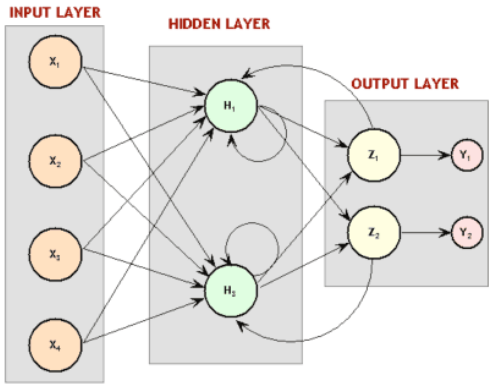


Figure 30 – An example Recurrent Neural Network

Ref: Moocarme, M., 2020

Creation of the RNNs were done through Keras. These RNNs are known as Sequential models within Keras. The first RNN created used Gated Recurrent Unit (GRU) layers for the hidden layers, with an embedding size of 50. With text classification tasks, the embedding layer is where all texts are inputted, and is initialised with random weights which are then learnt from the training data.

```
RNN_adjusted = Sequential()

embedding_size=8
# embedding size was 50
input_dim=x_train.shape[1]
RNN_adjusted.add(layers.Embedding(input_dim=vocab_size,
                    output_dim=embedding_size,
                    input_length=x.shape[1],
                    name='layer_embedding'))
```

Figure 31 – Recurrent Neural Network Embedding Layer Properties

There are three parts to an embedding layer, which are.

- Input Dimensions
- Output Dimensions
- Input Length

```
vocab_size = len(tokenizer.word_index) + 1 # This is due to the 0 being reserved for index
```

Figure 32 – Vocab_size Definition

Figure 31 shows the embedding layer creation, whilst figure 32 defines the Vocab_size variable being used. After the embedding layer, there are hidden layers. Each model was made up of 4 hidden layers with return sequences, more than 1 hidden layer defines a deep model for training. The weights of the layers were fed back to the hidden layer so that they can learn from past weights and adjust accordingly. As this happens the number of units for the hidden layer decreased by half.

Dense layers were used as the output layer, which are classic fully connected neural network layers, that connects each input node to each output node. The first dense layer used the relu activation function to allow the node to fire if it met the expected output (if it was correct) whilst the second dense layer had the size of the test length output (in this case three, as there were three outputs for polarity) and used a sigmoid activation function.

```
RNN_adjusted.add(Dense(10, activation='relu'))
RNN_adjusted.add(Dense(3, activation='sigmoid'))
optimizer = Adam(lr=1e-3)
RNN_adjusted.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

Figure 33 – Dense layers and Optimiser for a Recurrent Neural Network

One of the most popular activation functions for neural network text classification is "**softmax**", but the earliest tests of this activation function found accuracy was barely managing to achieve 70%, and the switch to "**sigmoid**" allowed for the ability for the networks to achieve accuracies closer to 90%.

Figure 33 also shows the use of the binary cross entropy loss function. This was used due to the use of numbers for the output (y value) which is the polarity scores of -1, 0 and 1. Categorical cross entropy was suggested later in the research, but the results were far worse than that of binary cross entropy, reaching 66% and never managing to get any higher regardless of number of epochs.

The use of Long Short-Term Memory (LSTM) layers was experimented with as an additional type of RNN hidden layer for test purposes. Overall, these layers turned out to be an improvement over the GRU layers and showed small gains to validation accuracy, normally around 1-3%.

```
RNN_3.add(layers.LSTM(units=32, return_sequences=True))
RNN_3.add(layers.LSTM(units=16, return_sequences=True))
RNN_3.add(layers.LSTM(units=8, return_sequences=True))
RNN_3.add(layers.Dropout(0.5))
RNN_3.add(layers.LSTM(units=4))
```

Figure 34 – Recurrent Network with LSTM layers and Dropout layer

Dropout itself, is a regularisation technique which aims to simply reduce complexity with the goal to prevent overfitting. It does this by randomly deactivating certain neurons in a layer with a certain probability (in this case, 50%). The consequence of this is that the network learns a bit differently and might make redundant representations, but training will be slightly faster. The results of the addition of this dropout layer are further discussed in the results section.

Through small evaluations with a dropout layer and 2 different hidden layer types, a decision was made to limit the number of epochs, to a much lower number of 5. This was because the accuracy grew at an efficient rate and the model was able to learn efficiently enough during the first initial 5 runs on average and conducting training for too many runs resulted in degraded performance, and things began to worsen overtime.

4.1.2 Convolutional Neural Networks

Convolutional Networks (CNN) are one of the most popular types of networks, used mainly for the classification of 2D images in Deep Learning. Figure 6 from the paper by *(Shirani-mehr, 2015)* shows the structure of a CNN. The embedding layer for a CNN is the same, but the layers following are different. The first layer that was created was the "**1D Convolutional**" Layer, which allows the model to extract features from sequences in the data and then map the internal features of this sequence.

A 1D convolutional layer requires filters, a kernel size and an activation function. For this experiment, the filter size chosen was 128, kernel size was chosen to be 5, and the activation function as relu. After this layer comes the "**Max Pooling**" 1D layer. Max Pooling is a sample-based discretization process. The objective of this layer is to down-sample the input representation to reduce its dimensionality, which allows for assumptions to be made about the features of the data.

After this were dense layers, and a dropout layer. The first initial training of a CNN showed that the issue of generalisation seen in RNNs was not as prevalent, even after 20 epochs the validation accuracy was more stable, but the line was wonky and showed gradual changes in the accuracy. The biggest difference was that the training accuracy was much lower than the validation accuracy, something not seen with RNNs. Figure 35 shows the structure of the CNN.

```
embedding_dim = 8

model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim, input_length=x.shape[1]))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(3, activation='sigmoid'))
model.compile(optimizer=optimizer,
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

Figure 35 – Convolutional Neural Network structure

4.2 Incremental Modelling

This section discusses process of evaluating a model through incremental stages. This style was chosen as it allows for the ability to evaluate a model against the requirements and then choose to return and create a new model, without the need to start over entirely, but keep the requirements updated along the way.

4.2.1 Early Experimentation

First implementations followed a very open style. Some of the earliest trained models were not evaluated graphically, and the results of the different parameters were saved in a written word format. These initial tests were explorations into setting adjustments toward the goal of performance improvement and accuracy increases. Early tests indicated that the adjustment of input dimension, the words set for the tokenizer index and settings such as batch size, epochs and clean vs unclean data lead to different results where accuracy and performance showed improvement.

4.2.2 Hyper-Parameter Tuning

When models finished training, the graphical representations seen in section 4.1 are explored and the accuracy is first evaluated, then consideration toward how to improve performance was investigated using hyper-parameter tuning. This was achieved first through the lowering of the embedding size. The first instances of embedding size began at 50, but this was deemed to be too large and was later reduced to 8, although the results of this change were not very noticeable in the early tests. Mentioned in section 4.1.1 discussing RNNs, one of the ways that tuning was done to improve model building was the addition of the dropout layer, to fight the generalisation problem that was encountered. The results of this didn't turn out as well as expected, but this strategy was kept in the later models as it didn't have a negative impact.

4.2.3 Randomised Grid Search

The technique of Random Search sets up a grid of hyperparameter values and selects random combinations to train a model and then score it. This technique was used to test for the best attributes for the CNNs. Another reason this was chosen is due to its use of cross-validation.

```
#Main Settings
epochs = 5
embedding_dim = 8
maxlen = x.shape[1]

#Parameter grid for grid search
param_grid = dict(num_filters=[32, 64, 128],
                  kernel_size=[3, 5, 7],
                  vocab_size=[vocab_size],
                  embedding_dim=[embedding_dim],
                  maxlen=[maxlen])

RGS_model = KerasClassifier(build_fn=create_model,epochs=epochs, batch_size=32)
grid = RandomizedSearchCV(estimator=RGS_model, param_distributions=param_grid,
                          cv=10, verbose=1, n_iter=8)
grid_result = grid.fit(x_train, y_train)
```

Figure 36 – Random Grid Search settings, using 5 epochs

Figure 36 shows this process, it shows the filters and kernel sizes to test, alongside the vocab size, embedding dimensions and the max length. Using 20 epochs and a cross validation of 10 over 10 iterations, the grid search managed an accuracy of 90% for training and validation accuracy.

Although this shows that adjustments into filter size and kernel size can reveal better accuracies, the accuracy was still lower than what was already being achieved by running the CNN over 20 epochs without Grid Search. Further testing involved an additional grid search, although this time using 5 epochs and an iteration size of 8. The results for accuracy were 87% for each. The results of the Grid Search were concluded to be a useful way to test settings, and led to more evenly spread results, but the results scored lower and were therefore discarded.

4.2.4 Using different Datasets to test Implementation

Many runs were undertaken using incremental modelling, but after some time, these adjustments no longer demonstrated meaningful differences. To combat this, the use of three datasets was explored. These had already been made in advance to test if the removal of stopwords and the removal of the neutral polarity would make any difference when it came to the accuracies that could be achieved.

The results of a recurrent model using unclean data, this being the noisy data taken initially with only URLs removed, achieved an improved accuracy over that of data that was cleaned. This was explored further by creating a model with the same parameters across all 3 datasets.

GRU Model on Different Data	Overall Accuracy	Precision Sensitivity	Recall	F1 Score
Unclean	**93.83%**	**91.13**	**90.29**	**90.71**
Clean	**90.72%**	**86.60**	**85.38**	**85.99**
Clean No Neutral	**87.61%**	**91.22**	**90.95**	**91.09**

Figure 37 – GRU based RNN data results on differing data

Figure 37 shows the result of running the same parameters of a GRU based RNN on different data. Here it is possible to see that cleaning the data and thus reducing the dimensionality results in gradual reduction of accuracy toward unseen data. The metrics of Precision and Recall helped to

demonstrate that the model trained against clean data with neutrals achieved a lower score in those categories against the same data without the neutral column. This can be explained by the fact that the no neutral data having no neutral column will result in higher base positive and negative correct predictions.

4.3 Model Evaluation

This section seeks to express the methods used to evaluate the completed models. The process of testing against created data, exploring the weights of the embedding layer, measuring cosine distance of words are explored and then concludes with the exploration of a different type of neural network not used until now.

4.3.1 Testing Against Created Data

As part of the evaluation phase of the project, seen in figure 38, new data was created to test against each model. This data was created to find out how well each model had been trained. This data had the added capability of being created with a specific polarity outcome in mind. One of the most common misconceptions of Sentiment unigram models is their ability to correctly predict a combination of words such as 'Not Great', which is due to the conception that "Not" Is negative while "Great" is positive.

Writing the combination of a negative word followed by "Good" or "Great", will normally lead to a positive classification due to the word good being included. One of the created data inputs included 'Not Great' to see how many of the models could correctly predict this as negative.

A new set of texts were created and then pre-padded with different max lengths, due to the unclean data models having a length of 115 and the clean data having a length of 88.

```
tokens_pad_unclean = pad_sequences(tokens, maxlen=115) #For uncleaned data models
tokens_pad = pad_sequences(tokens, maxlen=88) #For cleaned data models
```

Figure 38 – creating padded custom texts

The results of the tests indicated that only the GRU layered RNN and the CNN, both trained on unclean data, predicted correctly that the input 'Not Great' does not have a positive sentiment, with all other models predicting this input to be positive. This result observed the expected weakness of unigram models but demonstrated that all models maintained very high accuracy in predictions. The weakness of the models trained without a neutral output was that they were simply unable to decipher when a text was neutral in sentiment as they didn't have a neutral output, which leads to these models incorrectly classifying neutral sentiment texts as neutral isn't an option for their output.

4.3.2 Exploring Embedding Weights

To further explore how each model learnt words and how these words are connected, was by looking at the weights found within the embedding layer to see how similar they are. The embedding layer has a vector space size of 8 in length and holds 8 weights associated to each token. It was possible to retrieve and examine the embedding layer of any model. The CNN model trained on clean data with neutral polarity as an output was the target for this evaluation. The token representation of each word must be taken from the word index first, and then assigned into

memory. These new variables can then be compared to see how closely they match. Figure 39 shows the weights of these words and how closely the model represents them as having a similar meaning.

In the figure below, it is possible to see that the model associates good and great together most of the time, with a close representation in 6 out of the 8 weights, with 2 of the weights not being very closely represented.

```
weights_embedding[token_good]
```

```
array([-0.02167928,  0.39844897, -0.4249548 ,  0.07010969,  0.31652513,
        0.3294645 , -0.37243974,  0.35033685], dtype=float32)
```

```
weights_embedding[token_great]
```

```
array([ 0.3185309 ,  0.3755344 , -0.38657486,  0.38979816,  0.32629433,
        0.36724383, -0.359153  ,  0.370402  ], dtype=float32)
```

Figure 39 – comparing the weights of the words good and great

4.3.3 Cosine Distance between Words

Exploring the cosine distance of words is another way of seeing how closely represented the model has deemed the words it has learnt to be, which allows for the ability to see words that are closely represented and words that are far from the target word in terms of distance. A word that is far away is not related, while a word close in distance is related to the word. Figure 40 shows the cosine distance of words from the CNN model used for section 4.3.2. There are some interesting words here that the model considers close in representation to bad, such as ‘godwins’ and ‘samyoung841’. These results aim to show how the token sentences represent different words, some are as expected, but some others find that these words must be frequently expressed in a sentence with a negative sentiment polarity and thus the model now perceives those words to be negative as well. The same can be thought when observing the words closely represented to the word ‘good’.

Distance from 'bad':	Distance from 'good':
0.000 - bad	0.000 - good
0.031 - destroy	0.019 - realistic
0.061 - depressing	0.021 - checkmate
0.063 - godwins	0.022 - solange
0.064 - samyoung841	0.024 - aspiration
0.067 - socialismrejected	0.024 - unhcr
0.069 - fake	0.024 - jus
0.069 - sharks	0.024 - assembly
0.072 - blasted	0.025 - relaxing
0.072 - infrastruture	0.026 - spoonhead8
...	...
1.954 - soften	1.942 - jongen
1.954 - indigestion	1.946 - dominoeffect
1.956 - fxb	1.947 - previous
1.957 - marriage	1.947 - aprove
1.957 - drpeterjbrown	1.947 - cardinal
1.958 - thegreatawakening	1.951 - veggie
1.958 - trained	1.954 - 2020s
1.959 - skater	1.957 - owed
1.961 - ofengland	1.970 - keighley
1.965 - triwizard	1.974 - caligaegirl

Figure 40 – cosine distance of the words good and bad

The words that are nearer the bottom of figure 40 are the words that the model perceives and represents the furthest away in similarity to the word being observed.

4.3.4 Recursive Neural Network Exploration

As an extra step in the evaluation phase, another type of neural network was experimented with. This type of network had been used in previous papers but wasn't one of the chosen networks for the study. The recursive neural network is a special kind of deep learning network, which works by applying the same set of weights recursively over a structured input, which allows for the ability to produce a scalar prediction.

This network was built using a single LSTM layer and a pair of time distributed layers dense layers that allows the ability to get fully connected dense results at each timestep, rather than the results being more flattened by a traditional dense layer. A flatten layer is applied before the output reaches the traditional dense layer at the end of the network, which is using the "softmax" activation function. Through 5 epochs, this network posted results like the other networks trained on the same set of data, although accuracy was lower than both the LSTM layered RNN and CNN. There is argument that this lower result could be from the choice of the "softmax" activation function, but further tests were not done on this model.

4.4 Results

This section aims to discuss the results of the study, looking at early models through graphs, ending with discussion of the best model against each set of data. Results that are not present here can be found in the Appendix.

Figure 41 shows the first results achieved from a model trained through 50 epochs, validating against a select portion of the data unseen to the model in training. The visualisation shows a huge loss in validation accuracy, which evens-out close to the 20th epoch. Figure 41 is a visualised generalisation problem. The experimentation of dropout layers attempted to try and conquer this problem, formed of model overfitting to the training data, which meant the model was learning the training data too well.

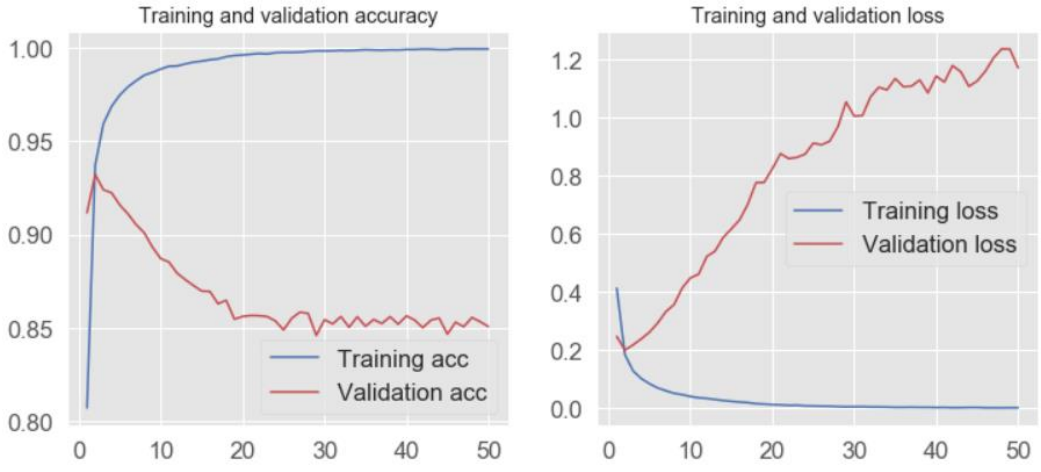


Figure 41 – Recurrent Network Training and Validation Accuracy results across 50 epochs

The result of the added dropout layer is shown in figure 42. Whilst the added dropout visibly improved the result, the drop in accuracy over time still presents itself. The chosen solution of using only 5 epochs led to results that didn't have as much drop, stopping training when the graph was at the peak accuracy gain.

Figure 42 – Recurrent Network Training and Validation Accuracy results across 50 epochs with a Dropout layer

Tests of all models against unclean data demonstrated the capabilities of the CNN. On this data, the CNN achieved an accuracy of 95.76%, higher than both RNN models, which can be seen in figure 43. Evaluating the results of the CNN through the confusion matrix demonstrated the polarity spread of correct and incorrect predictions. It was found that the high percent was well spread, with the CNN able to correctly predict most of the polarities, making more mistakes in the prediction of positive polarity, where 333 predictions toward positive were incorrectly classified.

CNN Unclean Accuracy: 96%

		Predicted		
		Positive	Negative	Neutral
True	Positive	4399	164	57
	Negative	212	1900	46
	Neutral	121	42	2923

Figure 43 – Confusion matrix of results for CNN on unclean data

The metrics of precision and recall are also important metrics, as they allow for comparison of not only accuracy, but also the ability of each model to predict true positives and how often these true positives are correctly predicted when false negatives are included (Recall). The F1 score is also investigated, as this is the difference between the Precision and Recall. Figure 44 shows these metrics for each model against the unclean data.

Model on Unclean Data	Overall Accuracy	Precision Sensitivity	Recall	F1 Score
GRU RNN	**93.83%**	**91.13**	**90.29**	**90.71**
LSTM RNN	**94.43%**	**92.27**	**90.91**	**91.58**
CNN	**95.76%**	**95.05**	**92.08**	**93.54**

Figure 44 – Results of each model against unclean data

Precision is how precise the model is when only comparing against predicted positive, and therefore how many of those were actual positive. The table above shows that alongside having the highest accuracy, the CNN also has a much higher precision score, and a higher Recall and F1 score. This table therefore shows that if there was doubt about the accuracy, the other metrics present also reinforce the argument that the CNN performed the best on this data.

Model on Clean Data	Overall Accuracy	Precision Sensitivity	Recall	F1 Score
GRU RNN	**90.72%**	**86.60**	**85.38**	**85.99**
LSTM RNN	**92.13%**	**89.16**	**86.97**	**88.05**
CNN	**91.55%**	**88.86**	**85.34**	**87.07**
Recursive RNN	**90.94%**	**86.76**	**85.92**	**86.34**

Figure 45 – Results of each model against no neutrals data

Figure 45 shows the results of each model against the no neutrals data. Comparing the models regarding their performance against clean data, it is of particular use to investigate more metrics than just accuracy, due to the distance between the accuracies on show. The recursive model created for evaluation has a slightly higher accuracy metric against the GRU RNN but falls shorts against the other models in all metrics. Looking at clean data, it is possible to see that the LSTM RNN performs the best in all areas and leads the way as a slight winner in respect of data lacking stopwords.

Model on Clean No Neutrals	Overall Accuracy	Precision Sensitivity	Recall	F1 Score
GRU RNN	**87.61%**	**91.22**	**90.95**	**91.09**
LSTM RNN	**87.70%**	**90.67**	**91.77**	**91.22**
CNN	**87.83%**	**91.90**	**90.49**	**91.19**

Figure 46 – Results of each model against no neutrals data

Figure 46 shows the results of the models against the data with neutrals removed. There is no clear winner when comparing accuracy, but the story changes when we look at the other metrics. The Precision metric shows the CNN has the highest score when comparing predictions to actual positive results, but then the LSTM RNN has a better Recall score, which is the score that looks at true positive predictions against all positive predictions, showing this model got more of these predictions correct compared to its competitors.

Choosing between the CNN and LSTM RNN in this evaluation comes up as 50/50, as there is no true model here that has a real edge against the other, this comparison is shown in figures 47 and 48.

LSTM RNN No Neutral Accuracy: 88% Predicted

		Positive	Negative
True	Positive	1625	449
	Negative	391	4362

Figure 47 – LSTM on No Neutral data

CNN No Neutral Accuracy: 88% Predicted

		Positive	Negative
True	Positive	1695	379
	Negative	452	4301

Figure 48 – CNN on No Neutral data

However, looking at the matrices for each model tells a better story. The CNN had a higher true positive rate than the RNN, a stat defined in the table. The CNN also appeared to incorrectly classify more negative results than that of the RNN, whereas also classifying fewer negative results correctly.

5 Critical Evaluation

This section aims to discuss the achievements of the projects, where the research can go from here, where this project will take the writer next, and how the use of Quantum AI can improve on this project further.

5.1 Project Achievements

This project was successful in respect to gaining on the accuracy mark that was first put forward. The highest achieved accuracy was 96%, which was 16% higher than the project aim. The project itself had 3 objectives, and it is felt that the project met these goals, and in some ways, exceeded in these goals. The main objective of the project was to classify textual data using modelling techniques, the actual techniques were not defined to allow for some degree of flexibility in finding a solution.

The use of Deep Learning was an incredible tool towards realising this objective, due to the capabilities. Machine Learning would have also allowed the ability to solve the task, but Deep Learning was far superior as it allowed for more complex and more complete network structures to be realised, such as the Convolutional Neural Network. The secondary objectives were aims that would be expected to be accomplished through natural progression of the project, and this came to light just as expected.

These were Model Investigation and Model Evaluation. Investigation was met using research done towards the literature review section. This review allowed for the choice of networks to become finalised and then realised, although not all options found within this research would be used in the project. The evaluation of models was also a clear success, leading to the positive outcome of higher than expected results, and text classification that was met with very few failings, mainly only in the way of sentence ambiguity, such as the use of the phrase ‘Not Great’ which was met with expected failure.

5.2 Further Development

To further develop on the work of this project, it is believed that the study and work toward bigram and trigram models should be done next. The models created for this project focused on the area of unigrams, and this led to expected failings in areas that could have better results if Deep Learning methodology was attempted on bigram and trigram exploration. It is also believed that the removal of stopwords didn’t have a positive outcome in this instance toward unseen data for the trained models, but it is still believed that this is also another area that could be looked at in more detail, to see where changes could be made towards generating higher results on data not using these stopwords.

5.3 Future Work

The work of Sentiment Analysis has advanced far since its inception, and the work done for this project has a long way to go. Sarcasm recognition and sentence ambiguity are still to this day areas that computers cannot read correctly, and the feeling is that more research into this with more complex neural networks and the use of Tensors can enable the ability to conquer these shortcomings. At the time of this project, Quantum Tensorflow is now available, and the aim is to advance into this area of computing power to create more powerful models and turn the corner of the problems found in text classification to create more understanding computational devices. *(TensorFlow Quantum, 2020)*

5.4 Personal Reflection

The project flowed well, and the use of a weekly log helped to ease production of the project. This weekly log helped to understand the progression the project had taken over that week and allowed for the ability to set deadlines for the following week, to maintain a realistic overview of progression throughout the early stages. One of the downsides of training Recurrent Neural Networks was that, even on a GPU, the time for the networks with 50 epochs would take close to 7 hours sometimes, and so these needed to be done overnight. This was an oversight at the beginning of the project but was something that had to be adjusted to.

The use of the incremental style of model creation was also incredibly useful because models had to be trained over some hours, and then to adjust would require another model to be built. If this was done in a different methodology, then it would have caused some issues with pacing of the project and wouldn't have allowed for the back and forth flexible approach that was available. If there was more time for this project, then the research into bigrams would have been done in more detail. Unfortunately, Quantum Tensorflow was not yet out to the public at the time of this project, or its inclusion would have been investigated as well.

The area of automatic web scraping is also another area of exploration that may be worthwhile to do more analysis into in the future. The aim of creating sentient AI in the form of chat bots to help people online rely on the context of web scraping and sentiment analysis to understand and respond to customer questions, and the element of making this process more automated would be a good step.

5.5 Conclusion

This research aimed to identify and make use of effective modelling techniques that could analyse and enhance the work done within the area of Sentiment Analysis, maintaining focus on noisy social media data. Based on the results, it can be concluded that the use of a Deep Convolutional Neural Network on noisy data can lead to high accuracy results when analysing sentiment. The results of the project indicate that data that doesn't go through the contemporary rigorous routine of data cleaning can lead to higher accuracy against unseen data, leading to more accuracy sentiment evaluation.

This study aimed to take advantage of modelling techniques to accomplish the goal, using the combination of Deep Learning, Natural Language Processing and Word Embeddings. The traditional idea of word embeddings was slightly adjusted here, where this study preferred to go with a more manually developed word embedding rather than using some of the previously created ones that

had been seen in studies before it. The feeling is that the use of a custom word embedding allowed the models to learn from words present in the index. This could lead to problems upon words that the model had never seen before, which has the possibility of leading to misclassification of words unknown.

The technique of Reinforcement Learning was not defined in the scope of this project, but was adapted in some form with the use of Recurrent Neural Networks, as they have the ability to learn from what that seen previously in their lower hidden layers, and then pass this to the next hidden layer, recursively learning the content.

The incremental approach was taken due to the nature of Data Mining, a field dominated by the vastly popular Crisp-DM methodology, which is a lot like the incremental methodology used here. The methodology itself aims to allow a researcher an alternate method to evaluate and allows for the possibility of being able to backtrack to make adjustments that help to improve upon shortcomings. This methodology is strong in this project as it helped to answer important questions regarding model validity and eased improvement in a way that didn't affect project flow unless it was crucial to do so.

Based on the findings here, other researchers in the field of Sentiment Analysis should consider more vast data when working towards sentiment classification tasks. The premise of big data is still developing, and the possibilities to this are still increasing, even for hobbyists looking to get into the field without a clear path. Sentiment Analysis of big data offers a way to do that, without focusing too much commitment away from the learning experience overall. The argument towards stopwords or no stopwords when it comes to training a Deep Learning model were answered here, but more questions will always come to light. It is encouraged that anyone interested in this discussion should take this into consideration and aim to develop their own arguments for or against this approach.

The growing availability of Quantum libraries within Tensorflow allow for the ability to create Deep models much more complex and computationally demanding where millions of data can be worked with rather than the hundreds of thousands worked on before. The speed of training will be increased dramatically allowing for a lot more time to fine tune, rather than time solely set aside for model training to be completed.

To better understand the reasoning behind the generalisation issues found with the project, future studies could aim to address this problem and find ways to achieve a more balanced and effective learning style and activation function choice that avoids this. This project aimed to fill the gap regarding deep learning against social media data, where past deep learning research focused on movie reviews and the financial sector. This research took a more topical approach focusing on a world trend at its time of initiation.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*. [online] Available at: <http://arxiv.org/abs/1603.04467> [Accessed 6 Jan. 2020].

Abell, J.C., 2011. March 21, 2006: Twitter Takes Flight. *Wired*. [online] 21 Mar. Available at: <https://www.wired.com/2011/03/0321twitter-first-tweet/> [Accessed 10 Mar. 2020].

Aggarwal, C.C. and Wang, H., 2011. Text Mining in Social Networks. In: C.C. Aggarwal, ed. *Social Network Data Analytics*. [online] Boston, MA: Springer US.pp.353–378. Available at: <https://doi.org/10.1007/978-1-4419-8462-3_13> [Accessed 10 Mar. 2020].

ALPAC, 1966. Language and machines. p.138.

Anon 2017. A beginner's tutorial on the apriori algorithm in data mining with R implementation. *HackerEarth Blog*. Available at: <https://www.hackerearth.com/blog/developers/beginners-tutorial-apriori-algorithm-data-mining-r-implementation> [Accessed 14 Mar. 2020].

Anon 2017. The History of Artificial Intelligence. *Science in the News*. Available at: <http://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/> [Accessed 18 Apr. 2020].

Anon 2019. *Natural Language Toolkit — NLTK 3.5 documentation*. [online] Available at: <https://www.nltk.org/> [Accessed 6 Dec. 2019].

Anon 2019. *NumPy — NumPy*. [online] Available at: <https://numpy.org/> [Accessed 25 Oct. 2019].

Anon 2019. *pandas - Python Data Analysis Library*. [online] Available at: <https://pandas.pydata.org/> [Accessed 25 Oct. 2019].

Anon 2019. *TensorFlow*. [online] TensorFlow. Available at: <https://www.tensorflow.org/> [Accessed 25 Oct. 2019].

Anon 2020. *A Brief History of Text Analytics by Seth Grimes - BeyeNETWORK*. [online] Available at: <http://www.b-eye-network.com/view/6311> [Accessed 10 Mar. 2020].

Anon 2020. *Definition of phrase structure tree | Dictionary.com*. [online] www.dictionary.com. Available at: <https://www.dictionary.com/browse/phrase-structure-tree> [Accessed 2 Mar. 2020].

Anon 2020. *TensorFlow Quantum*. [online] TensorFlow. Available at: <https://www.tensorflow.org/quantum> [Accessed 27 Feb. 2020].

Anon n.d. What is Syntax? Definition, Examples of English Syntax. *Writing Explained*. Available at: <https://writingexplained.org/grammar-dictionary/syntax> [Accessed 4 Mar. 2020].

Barbosa, L. and Feng, J., 2010. Robust sentiment detection on Twitter from biased and noisy data. p.9.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*. [online] Available at: <http://arxiv.org/abs/1406.1078> [Accessed 21 Mar. 2020].

Chomsky, N., 1957. *Syntactic structures*. 14. printing ed. Janua Linguarum Series minor. The Hague: Mouton.

Foote, K.D., 2019. A Brief History of Natural Language Processing (NLP). *DATAVERSITY*. Available at: <https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/> [Accessed 9 Mar. 2020].

Fürnkranz, J., 1998. *A Study Using n-gram Features for Text Categorization*.

Jones, K.S., 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28, pp.11–21.

Mikolov, T., Deoras, A., Povey, D., Burget, L. and Cernocky, J., 2011. Strategies for training large scale neural network language models. In: *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. [online] Understanding (ASRU). Waikoloa, HI, USA: IEEE.pp.196–201. Available at: <http://ieeexplore.ieee.org/document/6163930/> [Accessed 2 Mar. 2020].

Mladenic, D., 1998. Word Sequences as features in text-learning. p.4.

Moocarme, M., 2020. *Country Lyrics Created with Recurrent Neural Networks*. [online] Available at: <http://www.mattmoocar.me/blog/RNNCountryLyrics/> [Accessed 6 Apr. 2020].

Pang, B. and Lee, L., 2004. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In: *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics - ACL '04*. [online] the 42nd Annual Meeting. Barcelona, Spain: Association for Computational Linguistics.pp.271-es. Available at: <http://portal.acm.org/citation.cfm?doid=1218955.1218990> [Accessed 9 Nov. 2019].

Pang, B., Lee, L. and Vaithyanathan, S., 2002. Thumbs up?: sentiment classification using machine learning techniques. In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - EMNLP '02*. [online] the ACL-02 conference. Not Known: Association for Computational Linguistics.pp.79–86. Available at: <http://portal.acm.org/citation.cfm?doid=1118693.1118704> [Accessed 9 Nov. 2019].

Patodkar, V.N. and I.R, S., 2016. Twitter as a Corpus for Sentiment Analysis and Opinion Mining. *IJARCCE*, 5(12), pp.320–322.

Russell, S.J. and Norvig, P., 2016. *Artificial intelligence: a modern approach*. 3rd edition ed. Upper Saddle River: Pearson.

Saif, H., Fernandez, M., He, Y. and Alani, H., 2014. On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter. p.9.

Schwenk, H., 2007. Continuous space language models. *Computer Speech & Language*, 21(3), pp.492–518.

Sebastiani, F., 2002. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34(1), pp.1–47.

Shirani-mehr, H., 2015. Applications of Deep Learning to Sentiment Analysis of Movie Reviews.

Socher, R., Perelygin, A., Wu, J.Y., Chuang, J., Manning, C.D., Ng, A.Y. and Potts, C., 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. p.12.

Sohangir, S., Wang, D., Pomeranets, A. and Khoshgoftaar, T.M., 2018. Big Data: Deep Learning for financial sentiment analysis. *Journal of Big Data*, 5(1), p.3.

Tan, C.-M., Wang, Y.-F. and Lee, C.-D., 2002. The use of bigrams to enhance text categorization. *Information Processing & Management*, 38(4), pp.529–546.

Turing, A.M., 1950. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236), pp.433–460.

Weaver, W. (1949): 'Translation'. Repr. in: Locke, W.N. and Booth, A.D. (eds.) Machine translation of languages: fourteen essays (Cambridge, Mass.: Technology Press of the Massachusetts Institute of Technology, 1955), pp. 15-23.

Yu, L.-C., Wang, J., Lai, K.R. and Zhang, X., 2017. Refining Word Embeddings for Sentiment Analysis. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. [online] EMNLP 2017. Copenhagen, Denmark: Association for Computational Linguistics.pp.534–539. Available at: <https://www.aclweb.org/anthology/D17-1056> [Accessed 28 Dec. 2019].

Appendix A – Weekly Project Logs

Weekly Notes for Sentiment Analysis Project

This document aims to show weekly logs, where progression was made and what was done.

W/c 28th October 2019
First look at the project, the plan for this week is to get the data that is needed for the project through the twitter api and handle that data within the python notebook. Twitter API developer access was gained prior to completion of the PID so this is sorted now.

W/c 4th November 2019
This week was mostly spent finding and reading the first pieces of literature. It is important to discover the background of the project so the first few weeks will follow this approach. Plan to begin initial programming side of the work in December.

W/c 11th November 2019
Another week mainly focusing on literature reading, small progress was made on the coding side of the project. The URLs on all the tweets was handled, and the data was then saved into a new text file that can be pulled anytime the project notebook is resumed.
This week was also used to discover through literature the best models to create for the project. It was decided to focus on RNN and CNN networks through the Keras IDE.

W/c 18th November 2019
Data Cleaning was the focus for this week, alongside further work on the literature search and reading of the material. Many cleaning methods were looked at and left for later. I decided to do initial models on the data without cleaning things such as stop words out, but these will be removed later to compare the results with and without.

W/c 25th November 2019
This week focused on literature reading, and the first stages of model development. The first implementation of an RNN was done, words had to be tokenised so they could be fit into the model. Padding had to be done to make all tweet lengths be the exact same size so that the neural network will accept them. First initial runs of the model are planned to be done by the start of next week to start maintaining results.

W/c 2nd December 2019
First run throughs of the RNN were made. Bigrams were explored beforehand but were not used in the training at this time. I went for an initial 70-30 split for the training and testing, using the polarity scores rounded up through a built in add-on called Textblob. Initial runs without rounding up these polarity values resulted in low accuracy results of less than 1%.
After rounding up, the scores achieve around 90% give or take a few percent, but no real cleaning has been done to this data, so these results are not very useful as of yet, more cleaning needs to be done to see how these results changed.

W/c 9th December 2019
No work was done this week, other deadlines take focus and the work on this project will be resumed either next week or the week after.

W/c 16th December 2019
This week was spent doing more initial tests on the RNN model. Changes were made to the amount of layers, size of the embedding layers and other small adjustments. The first stages of model running with stop words and collection words removed showed small signs of possible overfitting to the training data. This will be explored more later. Changes of test-train split were made, adjusted to 80-20.

W/c 23rd December 2019
No work done this week, I am away from my computer in Hull and this week is spent with family for the holiday period.

W/c 30th December 2019
Considerations were made this week to consider the removal of the Neutral column. I kept it in and instead created a function that simply adds new columns for the 3 polarity values which can then be converted to an array used for the analysis. Initial accuracy results don't show much improvements.

W/c 6th January 2020
Results on the RNN runs show that there is noticeable Generalisation issues and overfitting happens quickly to the training data. Longer epoch runs make this more apparent, so the option to run much lower epochs when training an RNN model would be better for overall accuracy against data that the model has not seen yet aka the validation data or new data that will be introduced later.

W/c 13th January 2020
This week focused mostly on exam revision, so reading was done towards the background again, but not much done on the models. Although one initial run was done on a newly created CNN.

W/c 20th January 2020
Started adding dropout layers to each network built onwards to try and counteract the overfitting issue the models are having. Started drawing plots to see this overfitting after the model has been ran. More CNN models were created and gird search was used to create Cross Validation across CNN networks.
Results are positive, lower than before cleaning was done and small other changes were done, but the accuracy achieved at the moment is still around 85-87% depending on amount of epochs ran, which is good and above the target for the project of 80%.

W/c 27th January 2020
Project midway review was done this week. Results and feedback positive, small changes made to the dataset to remove some outliers that were found in the meeting, also made changes to the loss function, switching binary cross entropy to a categorical type, but this led to drops to 21% which never changed after, so this was changed back.
Neutral polarity was removed and the -1 and 1 results were dummied through one hot encoding and ran through again. No real accuracy change here though, still around 85-88%.

W/c 3rd February 2020
Made changes to the model again, nothing major, and no real adjustments to accuracy gained. Created new data and then ran the models against this to see how the model performs on unseen and newly created data. Results are good and as expected.
The results show common issues with text classification in CNN and RNN models. Consideration next is to do many comparisons and add in bigrams and possibly trigrams for training, but this is still only a consideration.

W/c 10th February 2020
So far all models were re-ran and the results were saved into a file to record all results should anything change to the notebook or the backup of it, or if new results are ran and accidentally remove results already made. Report was started, the background of the project is in progress, along with the objectives being written out.

W/c 17th February 2020
Options were explored and discussed within the weekly project meeting. It was decided that a recursive neural network would be worthwhile to take a look at, although the results are not expected to be better, rather they are an extra avenue to explore in the evaluative phase of the project. Bigrams and Trigrams options were explored, but at the time of this writing, I don't expect to attempt to implement these techniques and expect to just continue with a unigram NLP model.

W/c 24th February 2020
This week I spent time looking at the last parts of the Evaluation. I have used this time to explore the embedding layer and its weights to see what they look like, and see how far apart words are based on their weights. I used the LSTM Recurrent Neural Network trained on cleaned data with neutral class present. Most weights are as expected, with a small few with some small dissimilarities.
These words were then explored through their euclidean cosine distance, looking at which words are the closest to the word being explored through its cosine distance metric, this was done on the word good and the word bad, but more words can be explored if it deemed necessary.
Recursive Neural Network has been briefly been created, but the results are not an improvement. It is good to discuss the option of this network, but it doesn't add enough to justify it as an option for the final model.

W/c 2nd March 2020
Last finishing touches were done to the project, all results have been taken and now the important thing is to evaluate this. The literature review has been updated and the aim is to have this completed as soon as possible.

W/c 9th March 2020
Project is complete, work is being done now on updating the report structure so that it more closely fits to my own project. once this is done, it will be time to work on the writing of the introduction for the report, with the aim to have this section mostly complete by next week in a draft form. further work will be done on this later, at the same time the literature review is being updated.

W/c 16th March 2020
Literature review is considered done at this point, small adjustments to be made but otherwise looks to be complete now. small changes need to be made to make sure the references are in the right format but otherwise, everything seems complete. Will look now into starting the main section of the report which details how the project went and all the different sections of it.

W/c 23rd March 2020
Work on the report will continue on a weekly basis from now, as work gets done on this, the need for these weekly logs will now end and focus will shift to the report. At the time of writing the report has been started and literature review has been completed.

Appendix B – Results not included in Main Report

CNN Unclean Accuracy: 96%

		Predicted		
		Positive	Negative	Neutral
True	Positive	4399	164	57
	Negative	212	1900	46
	Neutral	121	42	2923

CNN Clean Accuracy: 92%

		Predicted		
		Positive	Negative	Neutral
True	Positive	4181	306	133
	Negative	370	1716	72
	Neutral	282	170	2634

LSTM RNN Unclean Accuracy: 94%

		Predicted		
		Positive	Negative	Neutral
True	Positive	4418	142	60
	Negative	420	1703	26
	Neutral	112	72	2902

LSTM RNN Clean Accuracy: 92%

		Predicted		
		Positive	Negative	Neutral
True	Positive	4205	273	142
	Negative	327	1740	91
	Neutral	256	114	2716

GRU RNN Unclean Accuracy: 94%

		Predicted		
		Positive	Negative	Neutral
True	Positive	4329	270	21
	Negative	250	1892	16
	Neutral	207	148	2731

GRU RNN Clean Accuracy: 91%

		Predicted		
		Positive	Negative	Neutral
True	Positive	4128	317	175
	Negative	381	1700	77
	Neutral	317	135	2634

GRU RNN No Neutral Accuracy: 88%

		Predicted	
		Positive	Negative
True	Positive	1658	416
	Negative	430	4323

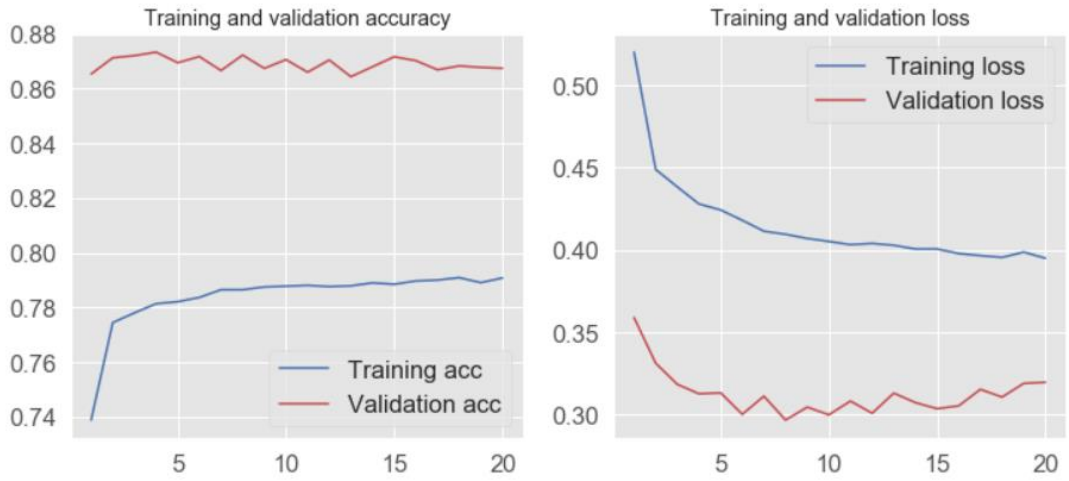


Figure – Convolutional Network Training and Validation Accuracy results across 20 epochs with a Dropout layer