Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Decoding Sentimental Tweets

In Depth Analysis of Deep Learning
for Sentiment Analysis

Master's thesis in Electronic and Computer Engineering

Sarah O'Reilly

Submitted to the University of Dublin,
TRINITY COLLEGE,
May, 2015

MASTER'S THESIS 2015: DECODING SENTIMENTAL TWEETS

In Depth Analysis of Deep Learning for Sentiment Analysis

Using the Concepts of Deep Learning to Perform Sentiment Analysis on Twitter Posts

O'Reilly, Sarah

Department of Electronic and Computer Engineering
Division of Computer Engineering
TRINITY COLLEGE DUBLIN
Dublin, Ireland 2015

In Depth Analysis of Deep Learning
for Sentiment Analysis. Using the Concepts of Deep Learning
to Perform Sentiment Analysis on Twitter Posts. Sarah O'Reilly

Supervisor: Rozenn Daynot, Department of Computer Science and Statistics
Examiner: Dr.Meriel Huggard, Department of Computer Science

Master's Thesis 2015:
Department of Electronic and Computer Engineering
Division of Computer Science
Trinity College Dublin
Dublin 2, Ireland
Telephone +353 1 896 1000

Typeset in LaTeX
Printed by *Reeds*
Dublin, Ireland 2015

Declaration

I, Sarah O'Reilly, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Sarah O'Reilly	Date

Summary

Deep learning is being used more and more extensively as a method performing sentiment analysis on social media. There are studies examining sentiment analysis on social media, these range from analysing whether film reviews are positive[25], to whether or not the number of twitter posts a individual has affects their twitter popularity[24]. While many studies are currently being performed, there are very few models for sentiment analysis that can accurately classify twitter data. The difficulty with classifying twitter data lies in the fact that the language used on twitter can be colloquial and amount of text in each tweet is minimal. With the large database available for twitter the desire to perform sentiment analysis is considerable in order to determine public option on a range of topics.

The purpose of this project to investigate deep learning and neural network algorithms and how they relate to logistic regression. In particular deep learning and network algorithms are being investigated for their ability to classify data. The purpose of this investigation is to develop a method of classifying data from Twitter in order to perform sentiment analysis on it.

Currently the most state of the art method for classifying data is the Stanford implementation of Sentiment Analysis from Socher et al[25] using sentiment treebanks. This method can achieve 80.7% accuracy when classifying the sentiment of movie reviews. By using treebanks the Stanford method can effectively understand the meaning of a sentence and use this information for classification. While this method of classification is currently the state of the art in sentiment analysis, it is trained on movie reviews rather than tweets. Movie reviews are by their nature very different from tweets. The length of the average movie review used to train the Stanford implementation is much longer than the average tweet. In addition to that the language used in tweets is very different from the language generally used in Twitter.

In this project the method of classification for sentiment analysis will be trained using tweets. This means that the model will be catered to processing and classifying tweets. One of the most helpful tools in sentiment analysis is knowing what kind of analysis to perform. By knowing something about the social and cultural context of the utterance, we can make smarter assumptions about the knowledge of speaker and more accurately tailor the sentiment predictions to specific types of communication[17]. The concept behind this project is that by designing a model to classify Twitter posts the sentiment predictions will be catered for this kind of communication.

The model will be adapted in two ways to improve its ability to classify tweets. Firstly the text processing that is performed on the tweet will be designed for tweets. This means that characteristics of tweets, such as emoticons, hashtags or usernames, will be processed to improve the accuracy of the classifier. As well as this the classification for this model will be trained using tweets, allowing the model to become familiar with indicators of sentiment unique to Twitter. This model, that has been specially catered for tweets, can then be compared to the state of the art in sentiment analysis, the Stanford model[25]. For this analysis it will be clear whether the state of the art or a model specially designed for twitter is better at classifying

tweets.
For the purpose of classifying tweets, the model implemented in this project that was trained on tweets performed better than the Stanford model[25]. This shows the advantage of training the classifier on the kind of text that it will be classifying. However sentiment is a subjective property and thus can be defined differently by alternative models. If the complexity of the classification is reduced to a less subjective model the Stanford model begins to outperform the model designed for this project. This illustrates the potential of a model to classify tweets that was trained uses tweets, but also reveals the difficulties faced when classifying text with such a small amount of signal.

In Depth Analysis of Deep Learning for Sentiment Analysis on Twitter
Using the Concepts of Deep Learning to Perform Sentiment Analysis on Twitter
Sarah O'Reilly
Department of Electronic and Computer Engineering
Trinity College Dublin

Abstract

The focus of this project is to investigate the concepts of neural networks and deep learning. The aim is to implement these concepts to perform sentiment analysis on Twitter posts, tweets. The structures currently that exist to perform sentiment analysis are optimised for large bodies of text. Tweets are by their nature short and thus difficult to classify the sentiment. The objective of this project is to improve the ability to classify tweets by creating and training a system specifically for tweets.

Keywords: Deep Learning, Learning systems, Machine Intelligence, Natural Language Processing, Neural networks, Sentiment Analysis, Social Media, Twitter

Acknowledgements

I would like to thank my supervisor Rozenn Daynot for always being willing to answer my questions and for her support and encouragement throughout the project.

Sarah O'Reilly, Dublin, May 2015

Contents

List of Figures

List of Tables

1 Introduction

1.1 Sentiment Analysis

Sentiment analysis is one method of data mining, where text can be analysed to determine the sentiment of the text. More formally the polarity of the sentiment behind text, that is whether the text is positive or negative, happy or sad, etc. Sentiment analysis can used to determine how people feel about a particular topic. For example, the data can be taken from twitter, where the tweets relating to the particular topic are isolated and the sentiment of the text is then determined. Generally speaking in sentiment analysis opinion are classified into positive, negative or neutral. Opinions can also be further classified into more specific categories for example very positive or very negative.

Sentiment Analysis has been used in range of different areas from including marketing, politics and psychology. Sentiment analysis can be used to analyse trends, identify ideological bias, target advertising and gauge reactions. While the promise of what sentiment analysis can produce with the amount of open data that is now available is very appealing, sentiment analysis is not without its drawbacks. Opinions are expressed in complex ways and often depend on the individual expressing them, this makes opinions difficult to classify. Opinions can be expressed in many different forms utilising all the complexities of language. Examples of this are negation. This is when the entire meaning of a sentience can be changed with one word, or sub-sentential reversal when the meaning of the sentence changes halfway through. These methods of expression are very difficult to account for in text processing. The tone in which an opinion is expressed is often negated on in text. Rhetorical devices are also used such as sarcasm or irony that the increase the difficulty of classification.

Previously sentiment analysis has been preformed on longer sequences of text than twitter. One of the most popular datasets to train on for sentiment analysis has been the IMDB movie review dataset, which has been used by many papers such as Socher et al. [25], a paper which will be discussed in greater detail in chapter 4. The IMDB dataset contains 25,000 movie reviews for training, and 25,000 for testing. This is a dataset where individuals write reviews of movies and accompany the review with rating from 1 to 10 stars. This rating system along with the review provides and indication of the sentiment of the review written. If the rating is 10 out of 10 stars then the likelihood is that the writing review accompanying the rating is also very positive. This rating accompanying the review means that the reviews can be polarised by using the rating without the need to hand label the dataset. This also means that the dataset is labelled accurately according to the writer of the review as they have provided the rating.

Movie reviews provide a good dataset for performing sentiment analysis because generally the reviews are relatively long, which is beneficial as it provides more signal for the sentiment analysis to be performed on. As well as that all the reviews in the IMDB dataset are obviously written as reviews to movies. This means that the scope of what can be discussed in the dataset is some what limited. On the other hand tweets are more complex to classify, there are many reasons for this. Firstly tweets are much shorter than the average movie review. Tweets must be under 140 characters, whereas the length of the reviews used in the IMDB dataset are also all longer than this. This means that there is less signal for the sentiment analysis to be performed on. The movie reviews in the IMDB dataset are also more thought out than the average tweet, which is much more casual. The frequency of miss spelt words is much higher on twitter than other media. Miss spelt words are sometimes accidental and sometimes intentional, e.g love being spelt as "luv".The quantity of data available on Twitter is much greater than that available for the IMDB dataset.

1.2 Deep Learning

Deep learning is a field of artificial intelligence that is built upon years of understanding and research. The concept originates from the idea that machines can store and process information in a similar way to the brain. Information is processed by many simple computation units. Many computational units in sequence can process complex information. By connecting computation units in this model the overall system begins to mimic a brain where rather than computational units there are neurons.

The concept of deep learning first came about in the eighties by Geoffrey Hilton. According to an article by Wired, Hilton began a wildly ambitious crusade to mimic the brain using computer hardware and software, to create a purer form of artificial intelligence we now call “deep learning.” [10] What differentiates deep learning from traditional machine learning is the fact that the machine can build up understanding of something without needed labels for the provided information.

During this period in the eighties up until 2004, Hilton continued work on improving the speed and efficiency of deep learning using algorithms such as back propagation. However these algorithms went widely unnoticed during this time as the computing power required to make use of these algorithms remained out of reach of what most computers could provide at the time.

Once the computing power became available to realise the potential of deep learning, interest in this area accelerated. In 2011, deep learning was founded at Google and in 2013 Hilton joined the company. [10]Today, deep learning is still being used for some of the most difficult problems for computers to solve. Microsoft uses deep learning for voice recognition systems. Facebook is using deep learning for facial recognition.

1.3 Report Outline

This project is laid out in four distinct sections text processing, deep learning background, machine learning for sentiment analysis and the conclusion. This part of the introduction gives a general outline of each of these sections.
The text processing chapter explains how to process a document so that it can be utilised by the machine. There are many methods of text processing, which of these is used determines how a document will be used. For sentiment analysis a document is processed and organised so that the sentiment can be extracted. This section also describes some of the difficulties associated with text processing when it comes to twitter. How data is extracted from Twitter and how the dataset used for this project was collected.
The chapter on the background to deep learning explains some of the concepts in machine learning and how they are used to create deep neural networks. This chapter starts by explaining some of the basic concepts in machine learning, that are computed in each neuron of a neural network. The next section then shows how these basic concepts are organised together to create neural networks. Section 3.3 on Neural Net Optimisation describes method of improving the accuracy of neural networks. The last section in this chapter describes some of the more complex concepts in machine learning and how they are used to create a deep neural network.
Chapter 4 on Machine Learning for Sentiment Analysis explains machine learning methods that can be used for sentiment analysis. These include naïve Bayes, the Stanford implementation of Sentiment Analysis from Socher et al[25], and finally the method used for classification in this project. The results and implementation of the method used in this project and the Stanford method are also displayed in this chapter. Finally the conclusion discusses how these models performed at classifying the dataset and what may changed to the model in the future to improve its performance.

2
Text Processing

Text processing is the act of manipulating text so that information can be gathered from it, by a machine. Text needs to be altered so that it can be understood by computer programs. There are many different ways that text can be processed in order to gather different information from it. Text processing is essential to sentiment analysis as it involves determining the sentiment of text. This chapter has two sections. The first describes some methods of Natural Language Processing (NLP) and the second section describes the type of text that will be used in this project "tweets", text that has been extracted from Twitter.

2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is the process by which computer programs are used to determine the meaning of natural language. This is generally performed through artificial intelligence. The main tasks of natural language processing is to understand human language or generate human language. In NLP there are low level tasks such as part of speech tagging (POS tagging) or tokenization, both of which will be discussed in section 2.1.1.1, and high level tasks. Higher-level tasks build on low-level tasks and are usually problem-specific [18]. One example of a higher level task in NLP is sentiment analysis. Sentiment analysis is the type of NLP that will be investigated in this project. The following part of this section aims to describe the tasks that are involved in NLP and how these were used in this project in order to perform sentiment analysis. [1]

2.1.1 Text Normalisation

The purpose of text normalisation is to process text so that all words in the text are of the same form. For example in the case of the word "woodchuck" it is important that for the purposes of text processing the word "Woodchuck" is understood as having the same meaning. A method of solving this problem may be to ensure that all the words are lower case. This is an example of text normalisation. The following sections describe methods of text normalisation and the issues associated with text normalisation.

[1]Much of the content for this section was draws on infromation from the Stanford video lectures for Natural Language Processing [13]

2.1.1.1 Tokenizing Words in Running Text

In NLP words are segmented into categorises in order to determine the meaning of running text. Take the sentence "I uh main- mainly wake up at sunrise", it is important to categorise the words in the text in order to determine the relevant words that determine the meaning of the text. The word "uh" is considered a filled pause as the word contains no addition information that adds meaning to the sentence. The word "main-" is not a word at all but only part of a word, this is known as a fragmentation.

Often in running text users can use the different words that have the same meaning. For the purpose of natural language processing the similarities between these words need to be recognised so that sentences with similar meaning are categorised together. For example when using a search engine a user may enter the word "cats", the search engine may then search for sites containing the word "cat" or "cats". These two words have the same lemma. A lemma is a word which has the same stem, part of speech and rough word sense as another word. The stem of the word is its root or the core meaning of the word. Affixes are additional pieces or parts that are added to the stem. In the example above the stem of "cats" is "cat" and the affix is "s". The part of speech (POS) of the word refers to whether the word is a noun, verb, pronoun, conjunction or interjection. In this case both "cat" and "cats" are nouns. For this reason "cat" is a lemma of "cats". "Cat" and "cats" however have different word forms. If two words have the same word form they have the same exact meaning, an example of two words with the same word form is Natural Language Processing and NLP.

After method of a segmenting words in running text is to determine the number of words in a given piece of running text. There are different ways of thinking about the number of different words in text. One method is to consider each instance of each word in a sentence, this is the number of tokens in a text. Another method of determining the number words in a given sentence is to determine the number of different words in given text. This is the number of types. In the text "they lay back on the San Francisco grass and looked at the stars and their", there are 15 tokens but only 13 types. However the number of tokens and types in the text depends on several aspects of the text. Firstly the pharse "San Francisco" could be considered as one word rather than two. The word "their" is also a form of the possessive case of "they", therefore it is possible that "they" and "their" should be considered one type.

This example illustrates some of the complications with classifying types and tokens. There are many other issues that arise in text normalisation such as which characters should be eliminated from a piece of text in order to normalise the text. An example of this is apostrophes, for example "I am" as oppose to "I'm". This is an example of a word form. In practise translating all the words with apostrophes to their original form requires a large data base to compare strings.

2.1.1.2 Normalising Word Formats

Normalising the formats of words relates to ensuring that the words with the same word form but different formats are normalised. In terms of a search engine normal-

ising text formats can be very important, to ensure that regardless of the format of the input into the search engine the valid responses will be returned. Abbreviations often have many different formats, for example the abbreviation U.S (United States) is the same as the abbreviation US. These words have the same word form and should be normalised to one format or representation. A method of performing this may be to remove all the periods in a word that is capitalised. However it is also important not to remove the periods that create the end of sentences.
Case folding is the process of reducing the letters in a word to lower case in order to normalise the format. In a search engine this would mean that the words "wednesday" and "Wednesday" would be considered the same. Abbreviations can make case folding more complicated, as lower case words may have one meaning but where they are capitalised the word may represent an abbreviation. If case folding is performed on the example from above, "US" (i.e. United States) then the result is "us". This word has a completely different meaning. For this reason performing case folding can only be performed at certain parts of a text, like at the start of a sentence.
Abbreviations are words that are commonly formatted differently depending on the author, but other words are often commonly formatted differently. Take the word "window", if this word was entered into a search engine the user may be looking for "window", "windows" or "Window". Therefore both of these words must be searched for in order to return the correct result. Although this problem is not always so easily solved. If the word "windows" is entered into a search engine the user is likely referring to Windows the an operating system and may be searching for a computer product rather than windows themselves.

2.1.1.3 Word Segmentation

Another element of text normalisation is word segmentation. Word Segmentation is splitting up a large word into smaller words. Word segmentation is mainly used in languages where there is no space between words, an example of this is Chinese. In Chinese there is no space between words to indicate the end of one word and the start of the next. This makes it difficult for NLP to determine the words in running text. An algorithm called mix-match segmentation. It is possible to split up words in Chinese as the average Chinese word is a consistent length. This is means that it is possible to gauge the beginning of each new word. In English word segmentation is much more difficult as word have varying lengths, for example if the phrase "the table down there" was typed as one word in a text, "thetabledownthere", it would be difficult to determine if the first word was "the" or "theta". For this reason words segmentation is very difficult in the English language.

2.1.2 Bag-of-Words

A bag-of-words model is a representation of a sentence where the sentence is represented as a list of words rather than as a sentence with sentence structure. This means that the meaning of the sentence is lost. The difference between the sentences "cat chases mouse" and "mouse chases cat" would not be captured by a bag-of-words representation. The bag words model is a useful tool as with a large enough sample

words can be identified as indicators for certain topics. If a bag-of-words model is used for spam email detection then words such as "buy" or "discount" may become indicators that an email is spam. Rather than having to understand the meaning of the entire email, these words can be used to classify emails as spam or non-spam emails.
In a dataset the bag-of-words method is implemented by creating a dictionary of all the words that are used in the dataset. Each word is entered in dictionary once, that is the number of types in the database are recorded in the dictionary. The dictionary is ordered according to the frequency of each word that is used in the dictionary. The length of the dictionary is then specified, the main purpose of this is to exclude a word from the dictionary that only occurs once in the dataset. If words occur only once then this will not divulge any new information about the text that will be useful for classification. Once the dictionary for the dataset has been created, then each word in the dictionary is given an id based on its position in the dictionary. Each text can now be represented as features, that represent each word that is present in the text and the dictionary.
This bag-of-words representation of each text becomes the feature set for each text. Table 2.1 shows an example of the features that were created from a bag-of-words representation for spam emails and the dataset that was used for this project, the Stanford dataset of tweets for the use in academics[7]. These features were created from the dictionaries that were created from each dataset. The columns in both, for both the spam email features and the Stanford dataset features, are the same. The first column is the id of the text. For the spam email detection dataset this represents each email and for the Stanford dataset this represents each tweet. The second column shows the dictionary id for each word in the text and the third column is the frequency of that word in the dataset. Table 2.1 shows firstly that there is a large length difference between the two texts, that is the emails and the tweets. The emails are much longer than the tweets and therefore have many more features per email. This table shows only a subset of the features for the first email, the first email actually has 68 features. The average tweet on that other hand may have anything from ten features to none. This is one of the reason that the longer the text is the easier it is to classify.
Another difference between the features in the spam email features and the Stanford dataset features is the frequency of the words in the dataset. The scope of topics discussed in the Stanford dataset is much larger than the scope of topics that are discussed in the Spam email features. This means that frequency of words appearing in more than one tweet is much lower than the frequency of words appearing in more than one email. This as well as the length of the text, also increases the difficulty of classifying the Stanford dataset as apposed to the spam email dataset.
In order to accurately create the dictionary the words must be preprocessed before they enter the dictionary. This process mainly involves text normalisation as discussed in section 2.1.1. As well as this in a bag-of-words model stop words must also be taken out of the dataset. Stop words are common words in a sentence that do not themselves contain any information that add to the overall meaning of the text. Examples of stop words are "the", "it", "me", "then". [2] These words are important

[2]Figure A.3 shows the full list of the stop words that were used in this project. These stop

Spam Email Features			Stanford Dataset Features		
Text id	Dictionary id	Freq	Text id	Dictionary id	Freq
1	32	601	1	257	7
1	1	861	1	26	31
1	94	589	1	1746	1
1	92	439	1	7	25
1	91	328	2	71	3
1	127	530	2	213	4
1	377	269	3	486	1
1	27	311	3	160	1
1	262	321	3	106	3
1	175	284	3	40	8
1	12	312	3	382	2
1	85	351	3	29	36
1	99	368	3	100	4
1	156	295	3	10	10
1	231	297	3	14	6
1	904	312	4	286	1
1	44	322	6	179	2
1	1647	247	6	1385	1
1	1222	213	6	2027	1
1	143	225	6	612	1
1	551	296	7	4210	18
1	1107	225	8	505	1
1	76	198	9	103	4
1	250	234	9	1391	1
1	137	217	10	198	5
1	391	183	10	1833	6

Table 2.1: Features for a bag of words representation

in order to understand the structure of the sentence however they do not add any additional information when they are isolated. If a word is a stop word then it is removed from the sentence.

2.1.3 Treebanks

Besides a bag-of-words representation of a text there are other methods of representing text, one of these is a treebank. A treebank is hierarchical structure in which the meaning of a sentence is maintained. Treebanks are a method of attempting to truly understand the text. The text is represented in a tree structure where the weight of the word is dependent on its position in the tree. There are two different main different types of treebanks. Treebanks that are created using dependency grammar and treebanks that are created using phrase grammar. The Prague Dependency Treebank is a treebank that is create where the words are connected based on their connection to other words and eventually to the verb in the sentence. The Penn Treebank on the other hand is created based on how the text can be broken up into phrases, for example verb phrases and noun phrases. The focus of this project is on phrase structure treebanks such as the Penn Treebank.
Treebanks are created from text corpses that have been processed to contain part of speech tags. As described in section 2.1.1.1, the part of speech (POS) of the word refers to whether the word is a noun, verb, pronoun, conjunction or interjection. The tree is then created based on phrase structure rules and grammatical rules. There are pre-existing rules that determine the structure of a tree bank. The text is broken up based on the phrases within the text. Groups of words that belong together are grouped together as phrases, these phrases are then grouped together to eventually form a treebank and encapsulates the entire text.
Figure 2.1 shows the Penn Treebank from the sentence "President Obama thinks it would be great idea to have mandatory voting by citizens but is against voter i.d. What's wrong with this picture?". From this treebank the method with which text is split up to create a treebank can be seen. Each sentence is primarily split up into the noun phrase and verb phrase in the sentence. The conjunctions, such as "but" in this case, is then considered another separate phrase altogether. Punctuation is also excluded from the other phrases. The final result is the treebank shown.[3]

words were taken from *http://www.textfixer.com/resources/common-english-words.txt*

[3]A list of all POS tags used in figure 2.1 are located in the appendix figures A.4, A.5, A.6 and A.7

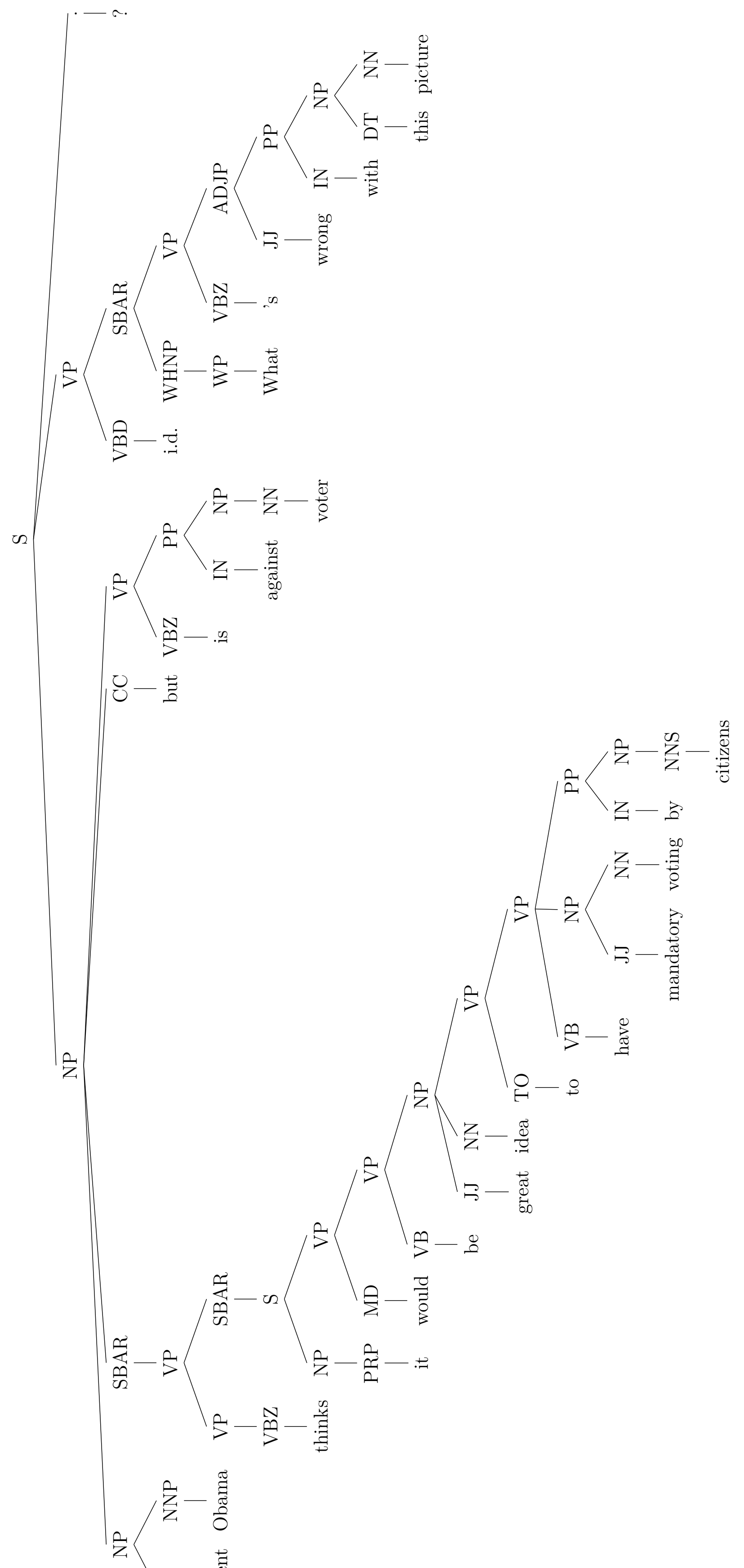


Figure 2.1: Example of a Penn Treebank for the sentence "President Obama thinks it would be great idea to have mandatory voting by citizens but is against voter i.d. What's wrong with this picture?"

2.1.4 NLP and Sentiment Analysis

Sentiment analysis is the use of NLP to annotate sentiment to text. It is in fact, as mentioned at the start of this section a type of NLP. Some NLP tasks are easier automate than others. For example the task of separating articles about basketball and football could be easily automated. A automated system could quickly classify these articles using keyword features such as the terminology specific to each sport or names of famous players. Sentiment analysis however is more complex to classify. The way that we express sentiment is a complex mix of the linguistic structures of our utterances and the assumed knowledge of the people who we are addressing[17]. This complex mix proves difficult to classify.

Increasing the difficulty of sentiment analysis are linguistic structures such as sarcasm and irony that dramatically affect the sentiment of text and are extremely difficult to detect. The use of different linguistic structures depends on the writer. Writers can express themselves differently depending their ethnicity. For example British speakers are known for their understated use of language. The sentence "I was a bit disappointed that" can be an understated may of expressing annoyance. The key for machine learning to pick up on these different methods of expression is to be exposed to as many different examples of expression as possible.

2.1.4.1 Keywords

Twittratr is a website that describes itself as a website where "you can distinguish negative from positive tweets surrounding a brand product, person or topic.This is a simple tool, which searches Twitter for a keyword and the results its get back are cross-referenced against our adjective lists, and then displayed accordingly." [4] By adjective list what Twittratr are referring to is a list of positive and negative labelled words. If a tweet contains one of these words then it is labelled accordingly. On Twittratr this list consists of 174 postive words and 185 negative words. The classifier determines the sentiment of the text by calculating if the tweet has more positive or negative words from the adjective list. There are problems with this method of classification, in that the context of the speech is not accounted for.

2.1.4.2 The Bag-of-Words Model and Sentiment Analysis

In terms of sentiment analysis a bag of words method can used to analyse a corpus to determine which words often indicate a negative sentiment and which words are generally in text with a positive sentiment. Depending on how accurate these features are at determine the sentiment of the text, they are given a certain weight. This weight is then used along with the other features of the text to determine the sentiment of the text. The problem with this method of sentiment analysis is that the sentences "That’s true, I am not a fan" and "That’s not true, I am a fan" would be understood as the same sentence by a bag of words model. This is because there is no attempt by this model to understand the sentence. For this reason text can be classified incorrectly.

2.1.4.3 Treebanks and Sentiment Analysis

Treebanks can be enhanced with semantic meaning, for example the sentiment. One example of where the sentiment is added to a tree bank is a sentiment treebank. A sentiment treebank is a treebank where the sentiment of each word is predetermined using a sentiment lexicon. A sentiment lexicon is a vocabulary of words that have a sentiment polarity. The words in this lexicon are given a value as to how positive or negative they are. This sentiment lexicon is then used to determine the sentiment of the last nodes in the sentiment tree. For each node above this the sentiment is calculated as a product of its children. The sentiment of the root node of the tree determines the sentiment of the overall text.

Treebanks take into account the position of words in a phrase. This can make them a more accurate method of determining the sentiment of a phrase. Treebanks are particularly useful for sentiment analysis when there is a negation in the sentence in question. Take the sentence "One of the most significant moviegoing pleasures of the year" and the sentence "Not one of the most significant moviegoing pleasures of the year", the difference between these sentences can only be determined by taking the structure of the sentence into account. The word "not" is a conjunction and would therefore be separated from the other phrases in the treebank. A bag of words model would be unable to correctly determine the sentiment of a negated sentence.

One study that has used sentimental treebanks to predict the sentiment of text has been the Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank [25]. Figure 2.2 shows the same sentence as in figure 2.1 however in this figure the sentiment of the sentence has been determined using the method outlined in Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank [25]. Negative nodes as illustrated as red nodes, positive nodes as illustrated as blue nodes with one plus sign and very positive nodes as illustrated as blue nodes with two plus signs. From this figure it can be seen that even though the sentence in question has very positive words the over all sentiment of the text is negative, due to the word wrong and its placement in the text.

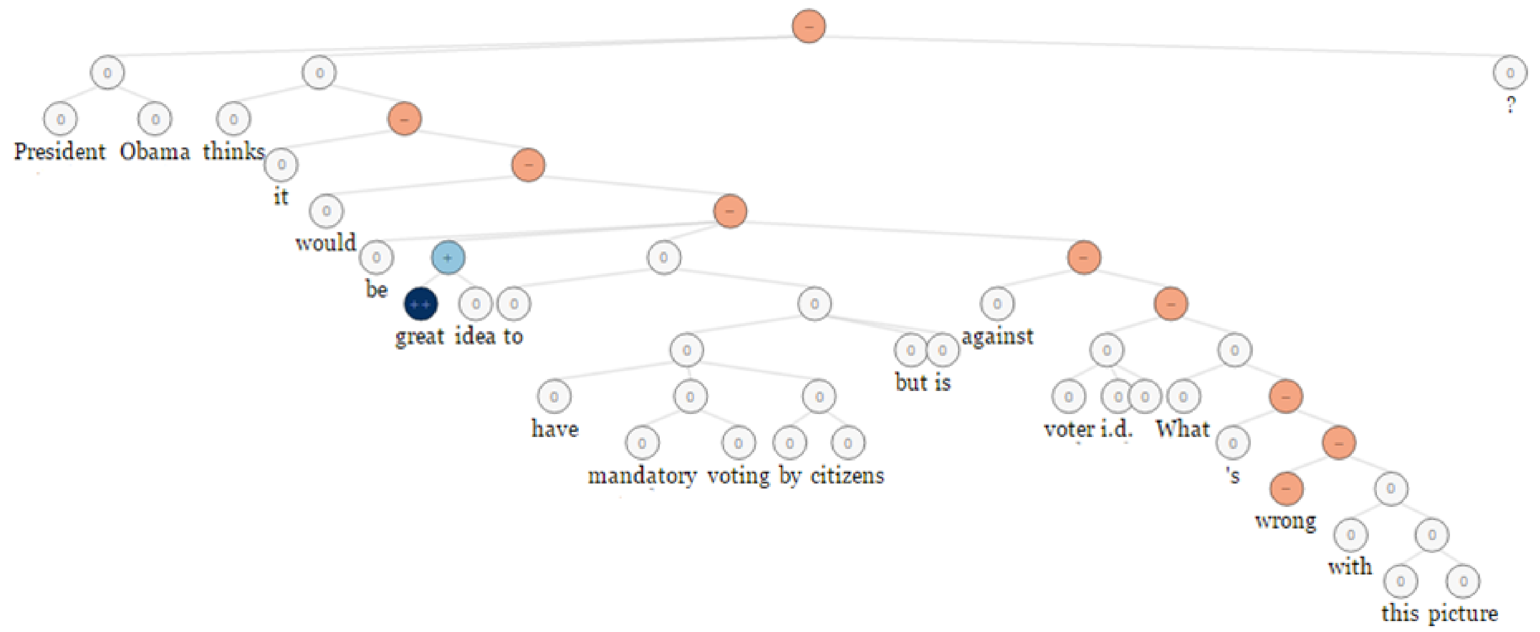


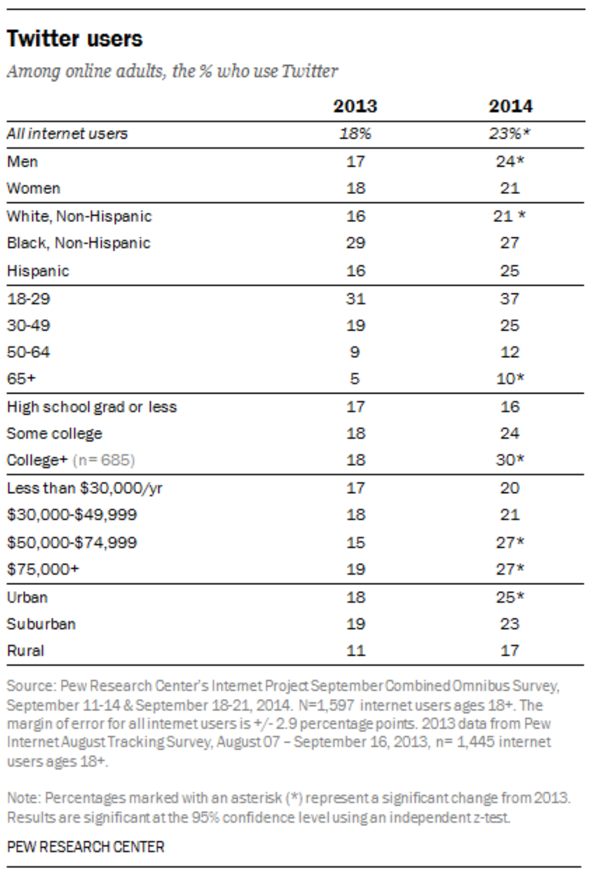
Figure 2.2: Example of a Sentiment Treebank for the sentence "President Obama thinks it would be great idea to have mandatory voting by citizens but is against voter i.d. What's wrong with this picture?" using the Stanford implementation[25]

2.2 Twitter as a Dataset

The purpose of this section is to describe Twitter and why this social media site was chosen as a dataset for this project. What are some of the complication associated with using twitter for sentiment analysis? What are some of the aspects that make it different from other documents traditionally used for text processing. The Twitter sentiment analysis is a special case of the general category of text classication[6].
Twitter is a social networking site that allows users to post short messages that must be under 140 characters. This makes twitter a microblogging website. These messages are known as "tweets". The appeal of twitter for sentiment analysis lies in the popularity of the site. According to *Internet Live Statistics*[36] since 2013 the number of tweets posted every day has reached 500 million. This number of tweets provides a huge dataset to gauge public opinion on a number of different issues. Another benefit of twitter is that by using the twitter API(Application Interface Program), live tweets can be pulled from twitter. This provides the ability to analyse the public opinion in real time.

2.2.1 Reasons for using Twitter as a Dataset

Several studies have been published on the ability of twitter to predict everything from the stock market[3] to elections[28]. Using twitter in order to gauge public opinion is an appealing one, with millions of people posting tweets everyday, the potential of such a dataset can be attractive to researchers. The most popular uses for Twitter are to assess public attitude towards a particular brand, product or political candidate. Using twitter has many advantages, Tweets can be analysed instantaneously, providing instantaneous feedback.
A dataset of this size has never been so readily available to researchers. The size of the dataset means that it is no longer required to speculate what kind of person buys a particular product or has a particular political view. It can now be statistically measured. Along with every tweet that is extracted from twitter the geographical location, gender, age and even a description of the individual is available. Figures 2.6 and 2.7 in section 2.2.4, show the information that is provided from each tweet that is pulled from Twitter using the Twitter API. From these figures the large extent of information that is available is apparent. Using this information the characteristics of customers can be easily examined.
While previously this type of data was attained using focus groups or opinion polls, it can now be accessed on a public forum where the candidates are not influenced by the focus group. Focus groups and other similar methods create artificial environments which can influence participants to respond insincerely. One outcome of these artificial environments can be that participants are unlikely to give negative feedback. Alternately on social media individuals can "hide behind the computer screen" meaning that people are more willing to express how they feel without the fear of the consequences that can otherwise occur. This negative feedback is a valuable perk of using social media . Groupthink is also a phenomenon that occurs in focus groups. Groupthink is when groups reach a consensus often orientated around a group leader, that can ignore otherwise persuasive alternatives. This means that



Twitter users

Among online adults, the % who use Twitter

	2013	2014
All internet users	*18%*	*23%**
Men	17	24*
Women	18	21
White, Non-Hispanic	16	21 *
Black, Non-Hispanic	29	27
Hispanic	16	25
18-29	31	37
30-49	19	25
50-64	9	12
65+	5	10*
High school grad or less	17	16
Some college	18	24
College+ (n= 685)	18	30*
Less than $30,000/yr	17	20
$30,000-$49,999	18	21
$50,000-$74,999	15	27*
$75,000+	19	27*
Urban	18	25*
Suburban	19	23
Rural	11	17

Source: Pew Research Center's Internet Project September Combined Omnibus Survey, September 11-14 & September 18-21, 2014. N=1,597 internet users ages 18+. The margin of error for all internet users is +/- 2.9 percentage points. 2013 data from Pew Internet August Tracking Survey, August 07 – September 16, 2013, n= 1,445 internet users ages 18+.

Note: Percentages marked with an asterisk (*) represent a significant change from 2013. Results are significant at the 95% confidence level using an independent z-test.

PEW RESEARCH CENTER

Table 2.2: Demographics of Twitter users. [5]

the opinion of a group may not accurately reflect that of the general public. Compared to traditional methods of opinion polling, extracting and manipulating data on Twitter or text mining is also relatively cheap. In addition to that the cost of this type of analysis does not increase with an increased dataset. However analysing data from Twitter presents new drawbacks in text processing that traditional methods of analysing public opinion do not face, this is discussed in section 2.1.

2.2.2 Difficulties in using Twitter as a Dataset

While there are downsides to traditional polling techniques, one of the advantages is the ability to vary the demographics of the group that is being analysed. On Twitter this is not a possibility. For example in the United States the number of Hispanic users on Twitter can be altered in order to improve the accuracy of the model for prediction. If a particular demographic are not present on social media, then their opinion will be negated when social media is used for public analysis. This is particularly true for the elderly, who are usually not as present online as other adult groups. Table 2.2 shows a table of the demographics of Twitter users in 2013 and 2014 taken from the Pew Research Centre[5]. From this table it is clear that the majority of Twitter users are between 18 and 29, affluent and urban dwellers. While this table shows the sampling bias of individuals that have Twitter profiles, it is in fact the Twitter users that post that create the dataset. When performing

sentiment analysis on Twitter on a particular topic it is the select number of people that choose to voice their opinion publicly on that topic that create the dataset. This means that when taking a dataset from Twitter the demographics of the group are further limited to the people that post, not only those that have Twitter profiles. According to the *Pew Research Centre* that carried out a study on the accuracy of the reaction on Twitter to public opinion as measured by surveys, "Much of the difference may have to do with both the narrow sliver of the public represented on Twitter as well as who among that slice chose to take part in any one conversation"[16].
The demographics of Twitter users that post needs to be considered, rather those that use Twitter. Figure 2.3 shows a geographical representation of geotagged tweets in Europe, tweets that have been tagged with a geographical location, it is taken from *The geography of Tweets*[22]. From this figure we can see that the same patterns emerge for Twitter posts as for Twitter users. In this figure the cities in Europe are a clear concentration of tweets. There are also more tweets in affluent areas. This is particularly visible when a comparison is drawn between the north and south of Italy. The Italy has a reasonably distributed population however concentration of tweets sent is much more prominent in the north of Italy than in the south. The reason for the higher concentration of tweet in the north of Italy is due to the fact that the north is much more affluent than the south. This example illustrates the influence of affluence upon the demographic of Twitter users.

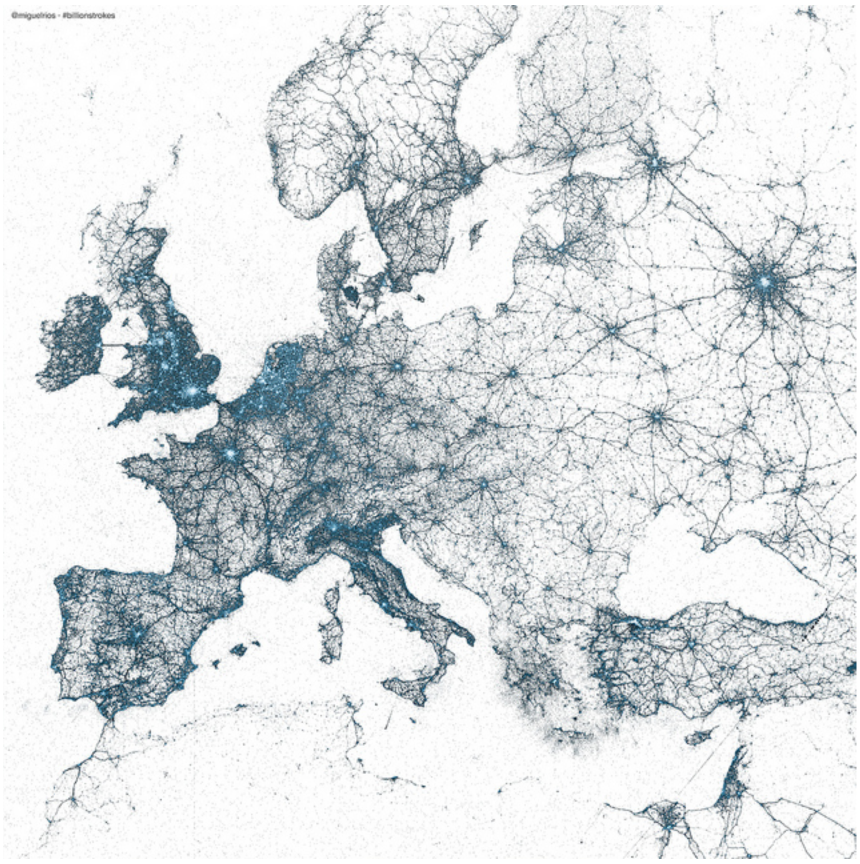


Figure 2.3: Geography of Twitter users. [22]

When analysing Twitter there are some ethical issues that need to be considered.

The main ethical issue is of analysing data of individuals without their knowledge or consent. When individuals post to Twitter, the content they post is automatically the property of twitter. This means that Twitter is entitled to allow developers access to this knowledge. However many Twitter users are unaware of this fact and post to Twitter without understanding how there tweet can be used for purposes that may not need to their benefit, such as advertising or recruitment. This project could in itself enable the use of tweets for purposes that may not benefit Twitter users.

2.2.3 Text Processing for Twitter

Tweets create a dataset that is unlike other dataset that are used for sentiment analysis. For this reason there are particular ways in which twitter has to be processes differently to other types of text. Some of the ways that twitter has to classified differently to other text are described below.

2.2.3.1 User names, hashtags and Hyperlinks

Tweets often contain user names, hashtags or hyperlinks. User names are used to direct a tweet at a particular tweeter user. Usually the @ is used to indicate that the tweet is referring to particular user. Hashtags, #, are used as topic specifier. So that if the tweet contains a hashtag it will be grouped together with all the tweets containing that hashtag if a user searches for a particular tweet. The topic specifier relates to the sentiment of the tweet, for this reason need to be maintained in the tweet. Links are sometimes included in tweets, they add no value in classifying the tweet.
Another element of hashtags that can make them difficult when it comes to text processing is that if a topic classifier consists of more than one word, for example "perfect day", then the hashtag is represented as a single word for example "perfectday". This means that word segmentation is required. As mentioned in section 2.1.1.3 word segmentation is very difficult in the English language the length of words differs greatly depending on the word. Because of this when more than one word is used in a hashtag the words are not segmented and therefore the words are not normalised correctly.

2.2.3.2 Vernacular Language and Misspellings

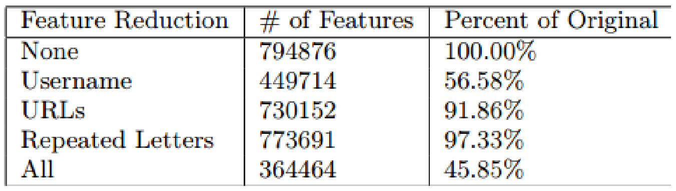
On twitter there can be many reasons for miss spelt words. As twitter users post from all kinds of devices on the go, there can often be accidental miss spelt words. For example spelling "album" as "albunm". Tweets can also contain miss spelt words that have intentionally been miss spelt. This can be done for different reasons, for example the word "you" may be miss spelt as "u" so reduce the number of characters that need to be typed by the user. Other misspelled words can be miss spelt for emphasis. Extra vowels can included in words to emphasis them, for example the word hungry (huuunry, huuuuungry, huuuuuungry). Intentionally miss spelt words can be altered in order to improve the performance of the classifier, vowels used more than twice can be removed from words that in the case of the word hungry the only the words hungry and huungry would be maintained in the dictionary. There can also be a reference list for all known misspellings used by twitter users so that the word "u" would be translated into "you". This reference list would need to contain all the known popular misspelled words on social media and their reference word. Unfortunately in the case of accidentally miss spelt words, not much can be done to process the words into useful features as accidentally miss spelt words are unpredictable.

2.2.3.3 Emoticons

Emoticons can be used to classify tweets as negative or positive by the emoticon that is included in the tweet. However emoticons are noisy labels. This means that the labels are not as accurate as hand classified labels. Emoticons can classify tweets incorrectly if the emoticon with the tweet is mismatched with the tweet. Consider the tweet "@BATMANNN :(i love chutney.......", despite the negative emoticon this tweet would probably be hand classified as positive. These types of mismatched tweets can be difficult to classify. In some cases emoticons are also stripped from the training set. This ensures that the classifier relies on unigrams or bigrams rather than the emoticon to classify the tweet. If emoticons are not used in the training data then if they occur in the testing data their effect will be ignored in the tweet and this can limit the effectiveness of the classifier.
In the case of the dataset that was used for this project, emoticons were used to classify the 1,600,000 tweets that were used for training. However these emoticons where then removed from the dataset, so that the tweet would be classified on the text that was included in the tweet rather than the emoticon. The testing data was hand classified and in these tweets emoticons where maintained. However as mentioned above as these emoticons have not been used in the testing data, they will not aid the classifier in determining the sentiment of the tweet. This limits the effectiveness of the classifier, but if the emoticons where left in the training data then every positive tweet would have a happy emoticon because the tweets where classified using the emoticons. The result of this would be that all tweets without emoticons would be classified incorrectly.

2.2.3.4 Feature Engineering

The overall effect of processing the text in this way in tweets is that the number of different features in the dataset is reduced. Table 2.3 shows a table that was used in the paper "Twitter Sentiment Classification using Distant Supervision" [8] that utilised the same dataset. Table 2.3 shows the effect of reducing the number of features in the dataset that was used for this project. For this investigation of the effect of reducing features in the dataset, the user names, URLs and repeated letters in words were removed from the dataset. From this figure it is clear that excluding the user names from the dataset in order to extract the features has the largest effect on the number of features in the dataset, however reducing the number of user names, URL's and repeated letters all has an effect on the number of features to 45.85% of its original size. Reducing the number of features in a dataset reduces the complexity of the dataset and thus makes the dataset easier to classify.



Feature Reduction	# of Features	Percent of Original
None	794876	100.00%
Username	449714	56.58%
URLs	730152	91.86%
Repeated Letters	773691	97.33%
All	364464	45.85%

Table 2.3: Effect of Feature Reduction [8]

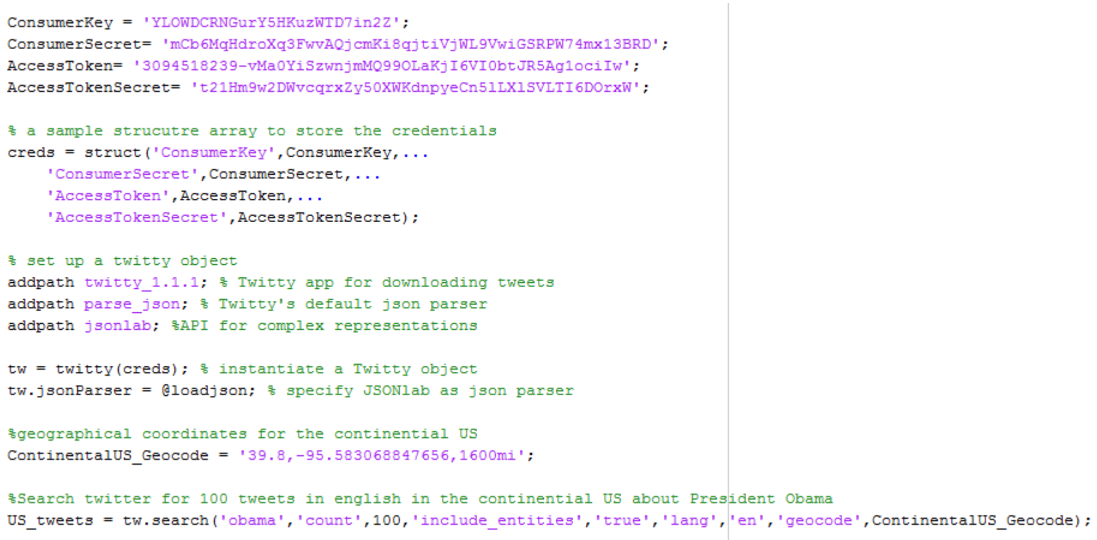
2.2.4 Extracting Twitter data using the Twitter API

Tweets can be extracted from Twitter using the Twitter API (Application Program Interface). This means that tweets that are posted by users of twitter are downloaded from the site and stored. The user of the twitter API specifies some details of the kinds of tweet they wish download, such as the subject matter. The recent tweets posted on that subject matter are then downloaded. In this way the user of the twitter API can create a corpus of tweets on any topic.

2.2.4.1 Code for Twitter API

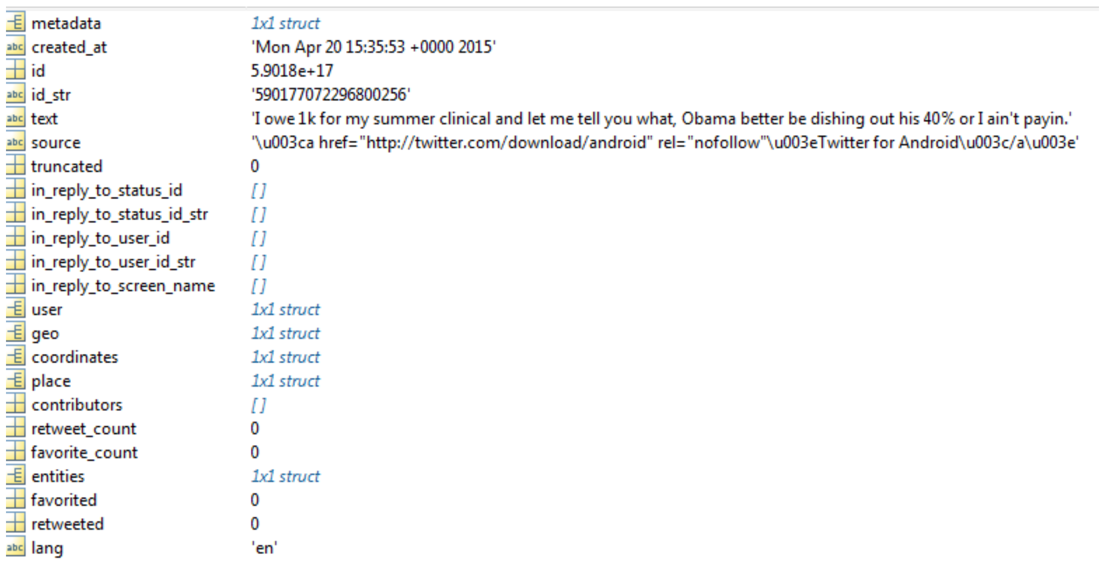
Figure 2.4 shows code taken from Matlab in order to extract tweets from Twitter that contain the tweets about Twitter.[4] The first few lines contain the credentials that are provided by Twitter when a Twitter development profile is created. These credentials are essential in order to download information for Twitter, so that the amount of data being downloaded by Twitter developer can be monitored. The next lines of code are used to add libraries to the project that are used to extract tweets from Twitter. *Twitty* is an app that is used by Twitter developers to access Twitter. The *parse json* library is a library to parse JSON format text. JSON (JavaScript Object Notation) is a format that is used to transmit data objects that is user friendly, the format the objects are transmitted in can be easily read by users. This format is generally used to transmit data from a web application to a server. In this case the data that is received from Twitter by the Matlab code is received in JSON format and parsed by this library and the final library *jsonlab* translates this into Matlab variables that Matlab can use and manipulate.
The final line of the Matlab code in figure 2.4 extracts the tweets from Twitter. The function *tw.search* searches Twitter for the most recent tweets posted on a given topic. The first parameter into the function is the topic that tweets will be searched for, for example in this case tweets on the topic President Obama will be extracted from twitter. Figure 2.5 shows a tweet that was extracted from Twitter using the Matlab code above. Each tweet than is extracted from Twitter contains additional information to the tweet. The date and time of the tweet, the id of the tweet and the source of the tweet are also available. As well as that there is also information about the location of the user and a short description of the user themselves, such as the number of friends the user has. The user information and the location of the user can be seen in figures 2.6 and 2.7.

[4]This code was adapted from code taken from Matlab Central [24]



```
ConsumerKey = 'YLOWDCRNGurY5HKuzWTD7in2Z';
ConsumerSecret= 'mCb6MqHdroXq3FwvAQjcmKi8qjtiVjWL9VwiGSRPW74mx13BRD';
AccessToken= '3094518239-vMa0YiSzwnjmMQ99OLaKjI6VI0btJR5Ag1ociIw';
AccessTokenSecret= 't21Hm9w2DWvcqrxZy50XWKdnpyeCn51LX1SVLTI6DOrxW';

% a sample strucutre array to store the credentials
creds = struct('ConsumerKey',ConsumerKey,...
    'ConsumerSecret',ConsumerSecret,...
    'AccessToken',AccessToken,...
    'AccessTokenSecret',AccessTokenSecret);

% set up a twitty object
addpath twitty_1.1.1; % Twitty app for downloading tweets
addpath parse_json; % Twitty's default json parser
addpath jsonlab; %API for complex representations

tw = twitty(creds); % instantiate a Twitty object
tw.jsonParser = @loadjson; % specify JSONlab as json parser

%geographical coordinates for the continental US
ContinentalUS_Geocode = '39.8,-95.583068847656,1600mi';

%Search twitter for 100 tweets in english in the continential US about President Obama
US_tweets = tw.search('obama','count',100,'include_entities','true','lang','en','geocode',ContinentalUS_Geocode);
```

Figure 2.4: Code for extracting data from Twitter using the API

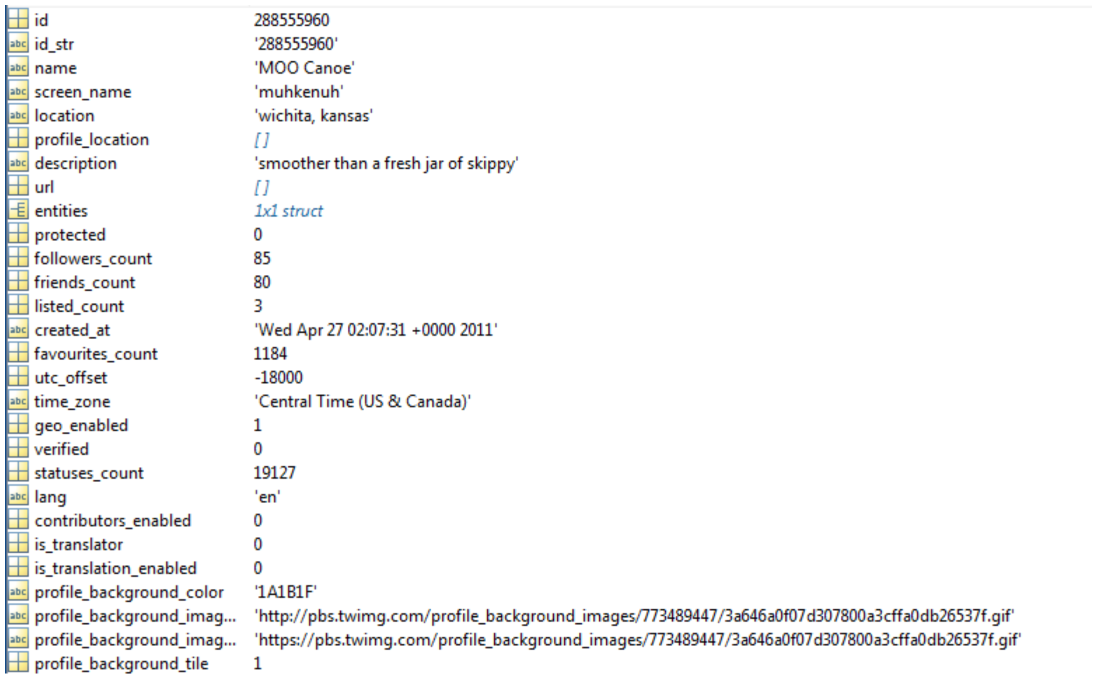


```
metadata                    1x1 struct
created_at                  'Mon Apr 20 15:35:53 +0000 2015'
id                          5.9018e+17
id_str                      '590177072296800256'
text                        'I owe 1k for my summer clinical and let me tell you what, Obama better be dishing out his 40% or I ain't payin.'
source                      '\u003ca href="http://twitter.com/download/android" rel="nofollow"\u003eTwitter for Android\u003c/a\u003e'
truncated                   0
in_reply_to_status_id       []
in_reply_to_status_id_str   []
in_reply_to_user_id         []
in_reply_to_user_id_str     []
in_reply_to_screen_name     []
user                        1x1 struct
geo                         1x1 struct
coordinates                 1x1 struct
place                       1x1 struct
contributors                []
retweet_count               0
favorite_count              0
entities                    1x1 struct
favorited                   0
retweeted                   0
lang                        'en'
```

Figure 2.5: Information from available from each tweet

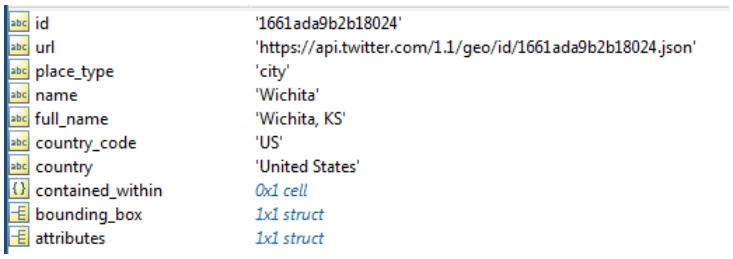
2.2.4.2 Tweet Attributes

These attributes of the tweet, such as the user location and the language of the tweet, which is denoted by the abbreviation *lang* and *en* denoting English, can be specified by the Twitter API when creating a query for tweets. In the function *tw.search* the language of the tweet, is specified by the entry *lang* and the following parameter into the function is the language tweets will be retrieved in. For example in figure 2.5 in the last line the language used, English *'en'*, is specified next to the *lang* of the tweet. The variable *geocode* specifies the location in which tweets will be taken from. The location is based on geographical coordinates. In this Matlab code in, figure 2.4, tweets in the continental US are the only tweets that are extracted from the Twitter. The coordinates of the continental US are shown as the variable



id	288555960
id_str	'288555960'
name	'MOO Canoe'
screen_name	'muhkenuh'
location	'wichita, kansas'
profile_location	[]
description	'smoother than a fresh jar of skippy'
url	[]
entities	1x1 struct
protected	0
followers_count	85
friends_count	80
listed_count	3
created_at	'Wed Apr 27 02:07:31 +0000 2011'
favourites_count	1184
utc_offset	-18000
time_zone	'Central Time (US & Canada)'
geo_enabled	1
verified	0
statuses_count	19127
lang	'en'
contributors_enabled	0
is_translator	0
is_translation_enabled	0
profile_background_color	'1A1B1F'
profile_background_imag...	'http://pbs.twimg.com/profile_background_images/773489447/3a646a0f07d307800a3cffa0db26537f.gif'
profile_background_imag...	'https://pbs.twimg.com/profile_background_images/773489447/3a646a0f07d307800a3cffa0db26537f.gif'
profile_background_tile	1

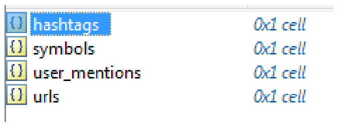
Figure 2.6: User information from available from each tweet



id	'1661ada9b2b18024'
url	'https://api.twitter.com/1.1/geo/id/1661ada9b2b18024.json'
place_type	'city'
name	'Wichita'
full_name	'Wichita, KS'
country_code	'US'
country	'United States'
contained_within	0x1 cell
bounding_box	1x1 struct
attributes	1x1 struct

Figure 2.7: User location from available from each tweet

ContinentalUs Geocode.
Other attributes of the tweet can also be included, for example in this case the parameter *include entities* specifies that entities will also be returned with the tweet. These entities can be seen in the tweet object that is returned, in figure 2.5, as the forth last field. Figure 2.8 shows the entities field from this tweet object. The entities field contains information about the tweet itself, such as the number of hashtags, symbols, user mentions, or URLs that are in each tweet. These terms and how they effect the processing of a tweet are explained in section 2.2.3.



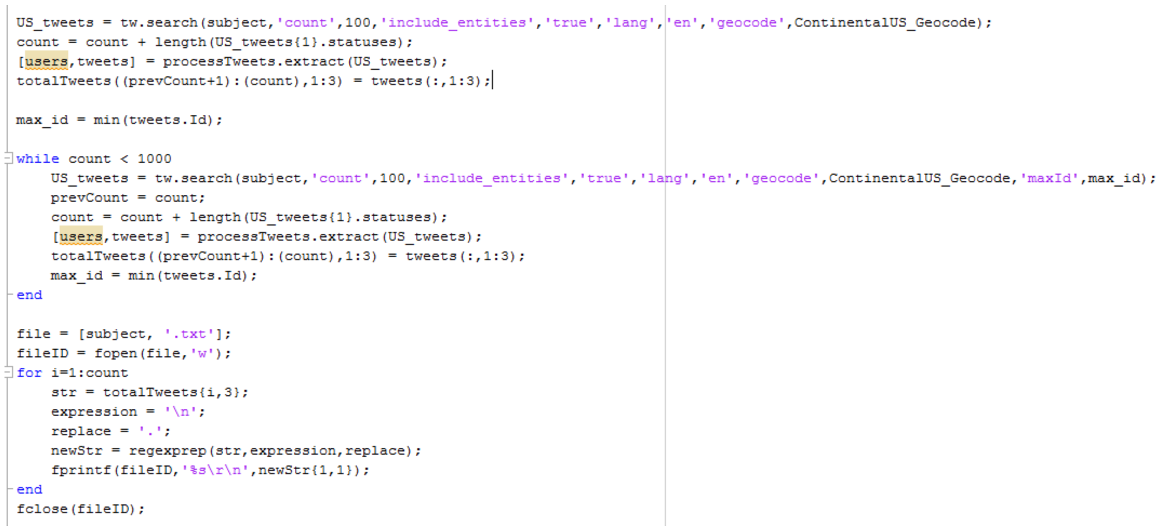
hashtags	0x1 cell
symbols	0x1 cell
user_mentions	0x1 cell
urls	0x1 cell

Figure 2.8: Entries information from available from each tweet

There are a number of other parameters that can be included in order to limit the type of tweets that are retired from Twitter. These other parameters include the ability to specify the particular dates to extract tweets from, or even locate the most popular tweets rather than the most recent tweets on a particular topic.There are also many more parameters that can be specified in order to make a detailed query to Twitter, these can be seen on the Twitter development website [35].

2.2.4.3 Twitter API Limitations

The variable *count* refers to the number of tweets that will be taken from Twitter in a request. In this case, for this Matlab code, in figure 2.4, the number of tweets that are to be extracted from Twitter are 100. This is actually the maximum number of tweets that can be returned per request. This value can however be manipulated by altering the *max id* that the query searches for. Each tweet has an id allocated with it, in figure 2.5 the id of the tweet can be seen as the third parameter of the object. The id of the tweet is increased with each new tweet that is posted. This means that if a query is made for a particular topic on the most recent tweets of that topic, then if the last id returned by that query is used as the maximum id for the next query more than 100 more tweets can be collected for each topic.



```
US_tweets = tw.search(subject,'count',100,'include_entities','true','lang','en','geocode',ContinentalUS_Geocode);
count = count + length(US_tweets{1}.statuses);
[users,tweets] = processTweets.extract(US_tweets);
totalTweets((prevCount+1):(count),1:3) = tweets(:,1:3);

max_id = min(tweets.Id);

while count < 1000
    US_tweets = tw.search(subject,'count',100,'include_entities','true','lang','en','geocode',ContinentalUS_Geocode,'maxId',max_id);
    prevCount = count;
    count = count + length(US_tweets{1}.statuses);
    [users,tweets] = processTweets.extract(US_tweets);
    totalTweets((prevCount+1):(count),1:3) = tweets(:,1:3);
    max_id = min(tweets.Id);
end

file = [subject, '.txt'];
fileID = fopen(file,'w');
for i=1:count
    str = totalTweets{i,3};
    expression = '\n';
    replace = '.';
    newStr = regexprep(str,expression,replace);
    fprintf(fileID,'%s\r\n',newStr{1,1});
end
fclose(fileID);
```

Figure 2.9: Code for extracting more than 100 tweets per topic

Figure 2.9 shows Matlab code for collecting 1000 tweets. The most recent 100 tweets are collected and then the last id of those tweets is used as the maximum id for next query. This code also writes each of the tweets to a text file, where they can be processed. However there is still a limit on the number of requests that can be made by any one developer, 100 tweets can only be collected every two seconds. The Twitter developer credentials, that were shown in figure 2.4, are used to track the number of tweets being extracted by each Twitter developer. Due to this limitation on the number of tweets that can be extracted every two seconds, when extracting large amount of tweets, twitter developers have to wait to keep extracting large amount of twitter. If the topic under investigation by twitter developers is very popular, it is possible that more than 100 tweets may be posted every second. If this is the case not all tweets on a particular topic can be collected. This means data cannot be collected from Twitter on a continuous basis and tweets can be missed during collection. This also means that collecting a large data corpus from

Label	ID	Date and Time	Topic	User	Tweet
4	3	Mon May 11 2009	kindle2	tpryan	@stellargirl I loooooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.

Table 2.4: Positive tweet from testing data

the Twitter API can be time consuming. Even more time consuming is determining the sentiment of each tweet by hand, i.e. hand labelling each tweet. For this reason a dataset was used for this project that was already collected and labelled. This dataset was collected using the twitter API in the same way as described above.

2.2.5 Dataset

For this project the dataset that was taken from Stanford for the use in academics[7]. This dataset was collected using the Twitter API which allows the users to extract tweets. The data was collected based on a variety of different topics such as including eating, china and kindles. The corpus is given with a sentiment numbered from 0 to 4 where zero represents a very negative sentiment, 2 represents a neutral sentiment and 4 represents a very positive sentiment. Along with the sentiment there is also an ID, date and time that the tweet was posted, topic that the tweet was related to, the user that posted the tweet and finally the tweet itself. When extracting tweets from the twitter API a particular topic must be specified and this is the reason for a a particular topic being associated with each tweet. [5]

2.2.5.1 How was the dataset collected?

This dataset was extracted from Twitter using the distant supervision as discussed in the paper accompanying the dataset Twitter Sentiment Classification using Distant Supervision[8]. All the tweets collected for this dataset contained emoticons, and these emoticons were then used to classify the dataset. That is a happy emoticon :) was used to indicate a positive sentiment in a tweet and a negative emoticon :(was used to indicate a negative sentiment. Tweets were classified as neutral if the tweet could be a line in wikipedia or a news paper headline.

Table 2.4 shows one of the tweets from the testing data that was contained in the Twitter corpus. The entire corpus provides 500 tweets for testing and 1,600,00 tweets for training. This represents 800,000 tweets with positive emoticons and 800,000 tweets with negative emoticons. [6]

[5]Figure A.1 shows the queries that were entered into Twitter in order to extract the tweets that made up this dataset.

[6]Figure A.2 illustrates the first 10 of the testing tweets.

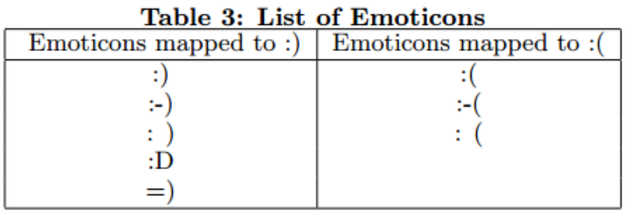


Table 3: List of Emoticons

Emoticons mapped to :)	Emoticons mapped to :(
:)	:(
:-)	:-(
:)	: (
:D	
=)	

Table 2.5: List of Emoticons [8]

2.2.5.2 Preprocessing

Before this dataset was used it was also preprocessed in several ways from the data that was extracted from the Twitter API before it was used for sentiment analysis. This preprocessed dataset was then processed again for the purposes of this project before it was used, the method in which this dataset was preprocessed is discussed in further detail in section 2.1. The first of these methods of preprocessing, was that all happy emoticons and sad emoticons where categorised together, this can be see in table 2.5. Secondly any tweets containing both positive and negative emoticons where removed. This can occur if the tweet in question has more than one topic being discussed. Emoticons were then removed from the dataset. Because emoticons were used to label the dataset, positive emoticons were present in all positive tweets and negative emoticons were present in all negative tweet. If the emoticons were retained in the dataset, then the classifier would simply learn to determine the sentiment based on the emoticon in the tweet. If a tweet did not contain an emoticon it would be classified incorrectly. The training data however does contain emoticons, as this will assess how proficient the classifier is at determining sentiment when a emoticon is present.
Retweets were also removed. Retweets are tweets that are posted again by another Twitter user. Retweets are often flagged by Twitter users with the abbreviation "RT", in order to remove the retweets all the tweets containing the abbreviation "RT" were removed. Duplicated tweets were also deleted. This meant that if a tweet was retweeted but the tweet was not flagged as a retweet with "RT", then it was still removed from the dataset.

2.2.5.3 Drawbacks

The training set consists of purely positive or purely negative tweets. This means that when training a classifier on this data it will not be trained to identify neutral tweets. The result of this is that it is very unlikely that any classifier trained on this dataset will identify any tweet as neutral. Another draw back is that because the emoticons were removed from the training dataset, as described in above, a classifier trained on this dataset will also negate the effect of emoticons when performing sentiment analysis, while emoticons can in fact be good indicators of the sentiment of a tweet.
The main reason this dataset was used as appose to others was that large datasets of classified twitter messages are difficult to find, according to the publishers of this dataset "[t]here are not any large public datasets of Twitter messages with

sentiment"[8]. However despite the drawbacks of this dataset it can still be used to assess the effectiveness of a classifier to perform sentiment analysis on tweets.

3 Deep Learning Background

In order to understand the operation of deep nets and how they preform deep learning, it is first crucial to understand neural networks. Neural networks with several hidden layers are considered deep nets. Each neural network is made up a series of neurons that are connected to create an output. The following sections will investigate the functionality of each of these neurons and how they are connected to other neurons to create neural networks. [1]

3.1 Inside Each Neuron

Classification is the act of assigning a a discrete value to data. Whether an email is SPAM or not SPAM or whether music is rock, classical or pop. This is in contrast to system that assign continuous values to data. Inside each neuron in a neural network there are computation units. In general these computational units preform classification. There are many types of classification, the primary focus of this project is logistic regression. Before discussing logistic regression, linear regression will first be discussed in order to explain some of the concepts used in logistic regression.

3.1.1 Linear Regression

Linear Regression is the aim of fitting a line to a given dataset. This is desirable because it means that the we can take a point that was not given in the dataset and estimate its likely output. If the two properties are lineally related, we can use linear regression to determine the relationship between the two. Using this relationship we can then estimate the output from a given input. Depending on the data given, the relationship between the two may be different.
Figure 3.1 shows a visual representation of linear regression. In this example the inputs are the height in meters and age in years of boys. These two properties are linearly related. The blue circles on the graph represent the dataset that given as an input in order to give the relationship between the height and age of boys. The yellow stars show test cases for the dataset. These test cases show how the height of a boy can be estimated from this dataset given the age of the boy. For this example the first yellow star represents a boy of age three and a half, with an estimated height of 0.97 meters and a boy of age 7 with an estimated height of 1.19 meters. As

[1]Much of the information discussed in section 3.1.1 through to section 3.2.1 is drawn from the Stanford Machine Learning tutorials [1].

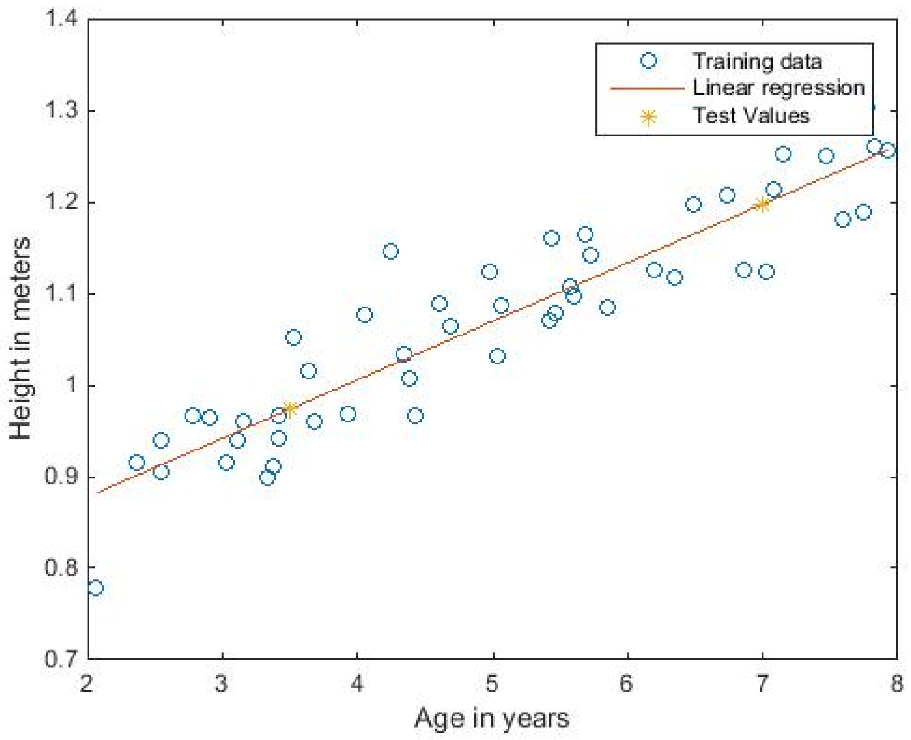


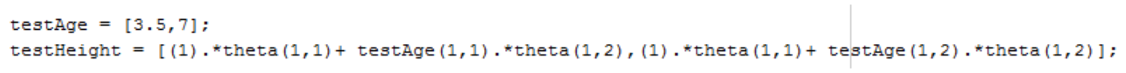
Figure 3.1: Linear Line Fit to Data

the input age is changed the height continually changes proportional to the change in the age. This continuous change is referred to as a continuous valued output, linear regression always has a continuous valued output. Classification like logistic regression of the other hand, which is discussed in section 3.1.4, has a discrete valued output.
Another aspect of figure 3.1 is that the curve does not exactly fit the dataset or training data. At an age of two, the training data has a case of a child that is much shorter than the predicted height from the relationship derived from linear regression. Children between the ages of two and three grow much more rapidly than is predicted by this relationship. This same disparity between the linear relationship and the training data can be see between the ages of four and five where children in the training set having widely varying heights. This is a result of a linear relationship being used to fix all the training data, whereas not all the parts of the training may be linear.
As described above the relationship between the two values depends on the dataset given. This means that given another dataset of different children, for example children from a different country, the relationship between the height and the age of the boys would be different. For example if the dataset was taken in a country where the population are much taller than the linear relationship between the height and the age would be steeper.
Linear regression is an example of supervised learning. It works by taking in training data using a learning algorithm to find the relationship between the input and the output. This relationship is the hypothesis of linear regression. This hypothesis

$h(x)$ is the relationship between x and y for a given dataset, where in the example above the age in year of the child is the x value and the height in meters of the child is the y value. For Linear regression the hypothesis is a linear function, such that:

$$h(x) = \phi_1 + \phi_2 x. \tag{3.1}$$

By varying ϕ_1 and ϕ_2 in 3.1 the angle of the line fitting the data changes. Choosing the best line fit for the data is in effect choosing the correct values for ϕ. Figure 3.2 shows the code that used to create the test values in figure 3.1. The variable "testAge" is the x values in this case and the variable "testHeight" is the output of both hypothesises. "testHeight" is calculated using equation 3.2.



```
testAge = [3.5,7];
testHeight = [(1).*theta(1,1)+ testAge(1,1).*theta(1,2),(1).*theta(1,1)+ testAge(1,2).*theta(1,2)];
```

Figure 3.2: Code for test cases in figure 3.1

In the example above height of boys is dependent only upon the age of boys. However the height of boys may be dependent on more than just the age of the boy, the height may also dependent on the ethnicity of the child. As some ethnicity's have taller populations than others. In linear regression the number of properties is known as the number of features. Increasing the number of features changes the hypothesis, for example two features give the following formula for the hypothesis in linear regression:

$$h(x) = \phi_1 + \phi_2 x_1 + \phi_3 x_2. \tag{3.2}$$

Where x_1 and x_2 are the features. Increasing the number of features can change how effectively the hypothesis fits the dataset. That is as long as the chosen features are relevant. Figure 3.3 shows a representation of a different problem, where the dataset is one of the cost of a house based on its square footage and the number of bathrooms. From this figure it can be see that the greater the square footage of the house and the number of bathroom the more expensive the house. Similar to the other graph the yellow star symbolises the test case where the price of the house is calculated as function of the input features, the number of bathrooms and the size of the house.
Figure 3.4 shows the Matlab code from creating the test cases for figure 3.3. This code shows how calculating the output based on two features is the same in principle to calculating the output for one feature. As shown in equation 3.2.
If the hypothesis for linear regression is determined by ϕ then there must be an effective method finding the best ϕ. ϕ is found using a cost function. The purpose of a cost function is to find the optimum values for ϕ by penalising any error. Equation 3.3 shows the equation for computing the cost function, where $J(\phi)$ is the cost function and $(h_\phi(x^i) - y^i)^2$ is the average squared error function. The goal of the cost function is to minimise ϕ for $J(\phi)$, by doing this the optimal ϕ is found. This is done by using gradient descent which will be discussed in section 3.1.2.

$$J(\phi) = \frac{1}{m}\sum_{i=1}^{m}(h_\phi(x^i) - y^i)^2 \tag{3.3}$$

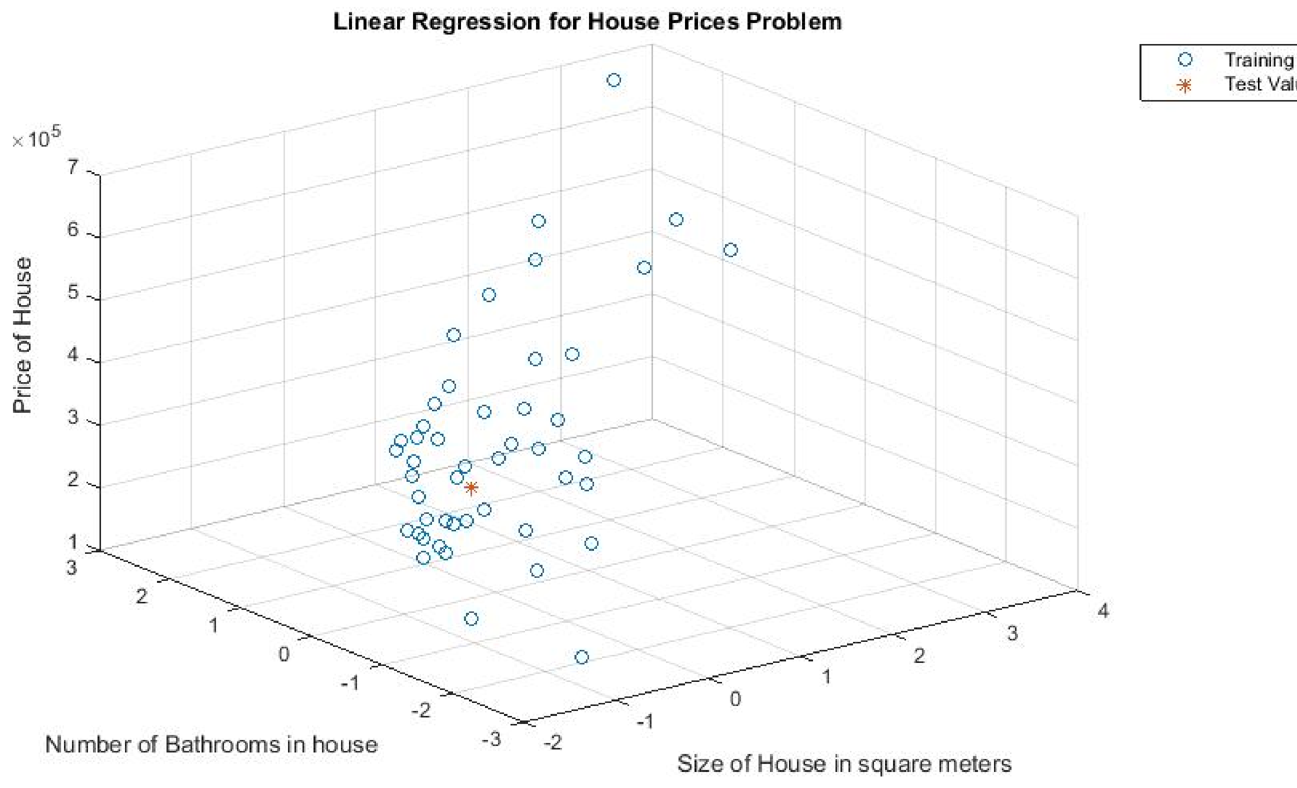
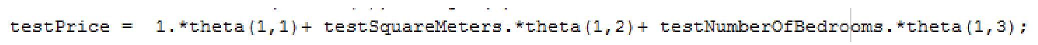


Figure 3.3: Multiple Features for the House Prices Problem



```
testPrice =  1.*theta(1,1)+ testSquareMeters.*theta(1,2)+ testNumberOfBedrooms.*theta(1,3);
```

Figure 3.4: Code for test cases in figure 3.3

3.1.2 Optimisation

Optimisation is the act of enhancing an algorithm, by minimising or maximising it, by systematically changing the input parameters.

3.1.2.1 Gradient Descent

Gradient descent is an iterative function for minimizing any given function. In this instance gradient descent is the process of changing the value of ϕ to optimise the cost function, as shown in equation 3.3. Gradient descent starts with ϕ values of zero and gradually changes them in order to find the value of ϕ that minimises the cost function. When the value of the cost function is at its minimum, the function is said to have converged. Formally gradient descent is given by the formula:

$$\phi_j := \phi_j - \alpha \frac{1}{20j} J(\phi) \forall j \tag{3.4}$$

where α is the learning rate and $J(\phi)$ is the cost function. $\frac{1}{20j}$ is a function that means that the result will always to moving towards its minim. $\forall j$ implies that this equation is carried out for all values of ϕ at the same time, that is ϕ_1, ϕ_2, etc. Therefore this function can be rewritten as

$$\phi := \phi - \alpha \frac{d}{d\phi} J(\phi) \tag{3.5}$$

what this means is that partial derivative of ϕ is found, until the cost function is at its minimum. Figure 3.5 shows gradient descent in process. Gradient descent starts with a guess value and calculates the gradient at that point. The initial guess value is then altered to step the solution in the negative gradient. Eventually the gradient is zero, at this point θ is at its optimum value to reduce the cost function, equation 3.3. This in turn results in the optimum θ for linear regression to fix the dataset.

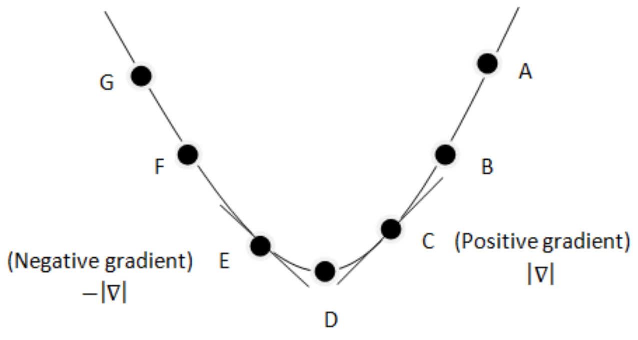


Figure 3.5: The Process of gradient descent taken from *Stack Overflow* [32]

Figure 3.6 shows the cost function against ϕ_1 and ϕ_2. The goal of gradient descent is to minimises the cost function by changing the values of ϕ_1 and ϕ_2. From this figure we can see that first values chosen for ϕ_1 and ϕ_2 have an impact upon the minimum that is found. Depending upon the starting values of ϕ_1 and ϕ_2 gradient descent may find a local minimum rather than the global minimum. This can affect the accuracy of gradient descent, however this does not effect linear regression. Linear regression is unaffected because of its cost function.The cost function for linear regression is squared, as shown in equation 3.3, which results in a parabolic function and parabolic functions only have one minimum. This can be seen in figure 3.7. Figure 3.7 shows the cost function against ϕ_1 and ϕ_2 for the linear cost function that was calculated for the example shown in figure 3.1. The cost function calculated for this dataset only has one minimum, the global minimum. If the cost function was based on a hypothesis with more input features, such as the example described in section 3.1.1 the house price problem example, then the representation of the linear cost function would form a cone.
The learning rate, α, of the function effects how large a step size is taken by each partial derivative. If the learning rate is too small then it may take many iterations of the gradient descent before the formula converges. If the value of the learning rate is too large then the formula can begin to diverge rather than converge. Each iteration overshoots the minimum. One way of checking if the learning rate is at

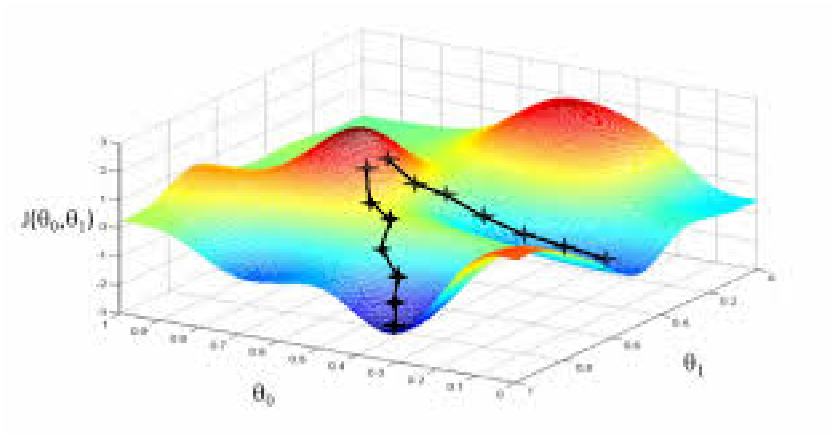


Figure 3.6: Gradient Descent effect of Starting Point on Result taken from *DatumBox*[29]

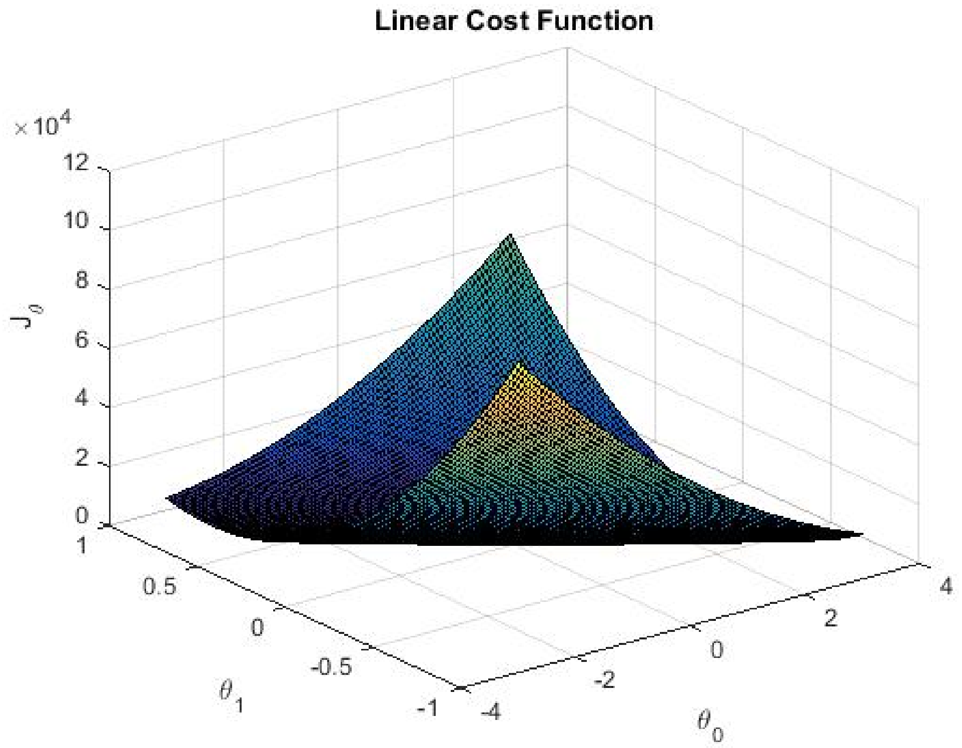


Figure 3.7: Linear cost function

an appropriate value is to ensure that the value of the cost function is falling every iteration of gradient descent. Figure 3.8 shows the cost function against the number of iterations for the gradient descent that was calculated for the example shown in figure 3.1. Figure 3.9 shows the change in ϕ_0 and ϕ_1 for the gradient descent that was calculated for the example shown in figure 3.1. For this example 300 iterations of gradient descent were required before the values converged. However the values of theta were very close to their optimum after 100 iterations. Figure 3.10 shows the code that used to calculate the gradient descent algorithm for the example in section 3.1.1 the boys height problem. The variables "thetaIterations" and "J theta" are used to create figures 3.8 and figure 3.9 that can then be used to determine if gradient descent has converged.

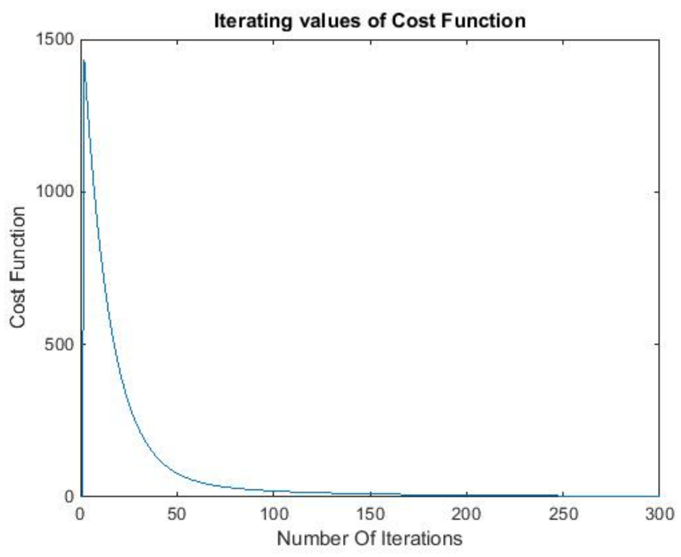


Figure 3.8: Change in Cost Function with increasing number of Iterations

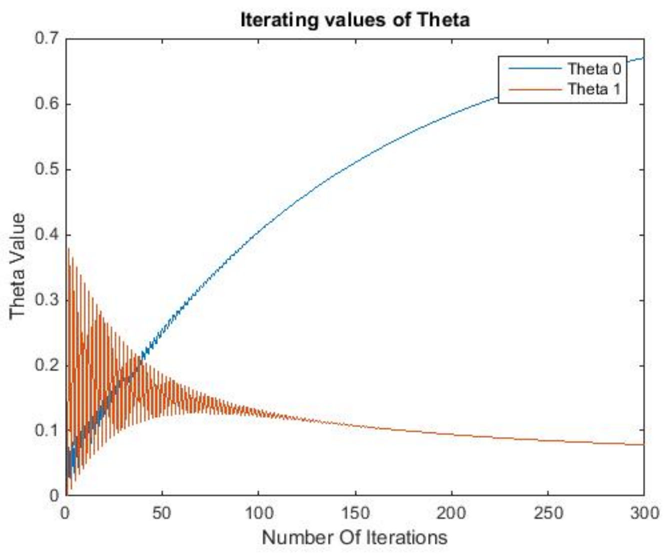
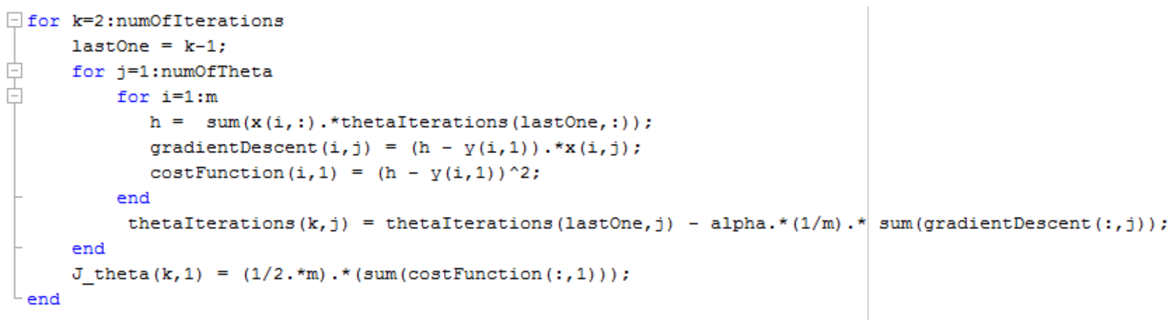


Figure 3.9: Change in Theta with increasing number of Iterations

Figure 3.11 shows a birds eye view of the graph shown in figure 3.7. If the features of the two input features have the same scale then this representation of the cost function would have perfectly circular contours. In this case the scale of the two features are not exactly the same therefore the contours are not perfectly circular. The less circular the contours, the longer convergence takes. When the scale of the two input features are widely different this rate of convergence can be dramatically effected, an example of this is described in section 3.1.1 the house price problem



```
for k=2:numOfIterations
    lastOne = k-1;
    for j=1:numOfTheta
        for i=1:m
           h =  sum(x(i,:).*thetaIterations(lastOne,:));
           gradientDescent(i,j) = (h - y(i,1)).*x(i,j);
           costFunction(i,1) = (h - y(i,1))^2;
        end
         thetaIterations(k,j) = thetaIterations(lastOne,j) - alpha.*(1/m).* sum(gradientDescent(:,j));
    end
    J_theta(k,1) = (1/2.*m).*(sum(costFunction(:,1)));
end
```

Figure 3.10: Code for calculating Gradient Descent used to create figure 3.1

example. For this example, the inputs are the square footage of the house and the number of bedrooms in the house and the output was the cost of the house. Because the scale of the square footage of a house and the scale of the number of bedrooms in the house varies greatly, the contours of the cost function for that example would be non-circular. In order to reduce the rate of converge for this problem, the input features can be rescaled so that their is not such a large difference between them.

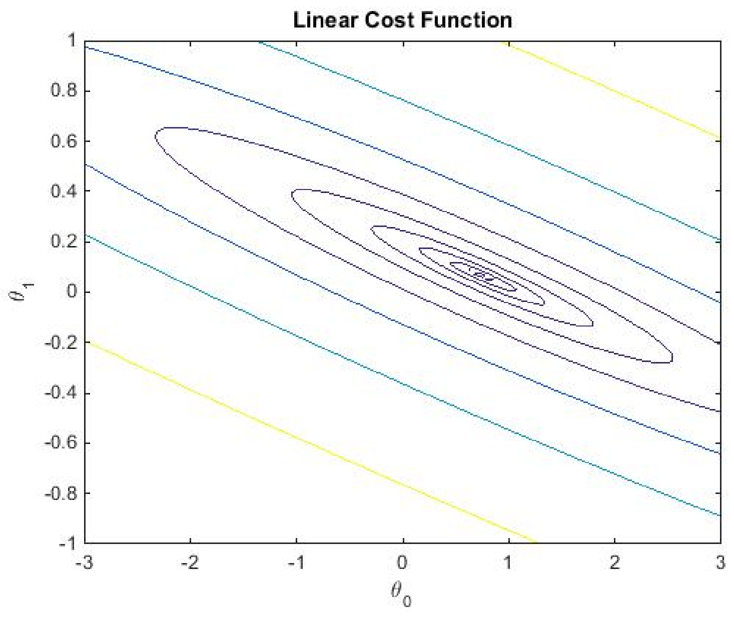


Figure 3.11: Affect of feature scale on Gradient Descent

3.1.2.2 Newton's Method

Newtons method is another method of optimisation other than gradient descent. Gradient descent often requires many iterations of the algorithm before the algorithm converges. Newtons method can be faster than gradient descent due to the fact that number of iterations required as less than that for gradient descent.Newton's method functions by finding the point in the cost function where its derivative is equal to zero. This is the same as find the minimum of the function, as shown in equation 3.6.

$$\frac{d}{d\phi}J(\phi) = 0 \tag{3.6}$$

Each iteration of newtons method finds the tangent of the cost function and where it intersects with the x-axis. This is then used to find the ϕ values that are used for the next iteration. Eventually values for ϕ are found that minimise the cost function. The formula for each iteration is given in equation 3.7, where $J^{'}(\phi^t)$ is equal to the derivative of the cost function $J(\phi)$ as shown in equation 3.8 and $J^{''}(\phi^t)$ is the derivative of $J^{'}(\phi^t)$.

$$\phi^{t+1} = \phi^t - \frac{J^{'}(\phi^t)}{J^{''}(\phi^t)} \tag{3.7}$$

$$J^{'}(\phi^t) = \frac{d}{d\phi}J(\phi) \tag{3.8}$$

Equation 3.7 shows the each iteration for newtons method where ϕ is only one vector however depending on the number of input features ϕ may be more than one vector. Equation 3.10 shows each iteration of Newtons Method for ϕ of more than one vector. H is the the hessian containing the partial derivatives of each of the vectors of ϕ, as shown in equation 3.11.[2]

$$\phi^{t+1} = \phi^t - H^{-1} \bigtriangledown_{\phi} J \tag{3.9}$$

where

$$\bigtriangledown_{\theta} J = \frac{1}{m}\sum_{i=1}^{m}\Big(h_{\phi}(x^i) - y\Big)x^i \tag{3.10}$$

$$H = \frac{1}{m}\sum_{i=1}^{m}\Big[h(x^i)\Big(1 - h(x^i)\Big)(x^i)(x^i)^T\Big] \tag{3.11}$$

3.1.2.3 Comparison of Gradient Descent and Newtons Method

Gradient descent is considered to be a slightly simpler computation than newtons method. However in order to perform gradient descent a learning rate, α, parameter must be chosen.Choosing an optimum value of is difficult and can hinder the simplicity of the function. On the other hand newtons method does not require any parameters and for that reason can be considered easier to use. In comparison to gradient descent, newtons method converges faster. Gradient descent requires more iterations to converge. However despite the fact that gradient descent takes longer to converge each iteration is cheaper to compute. Each iteration of gradient descent has a cost of $O(n)$ where n is the number of features where as newtons method has a cost of $O(n^3)$. Newtons method has a higher cost as the cost of inverting a matrix, H^{-1}, is high compared to the computation required for gradient descent. In general due to the cost of both functions gradient descent is used when the number of features is very large as the cost of computing newtons method becomes too large.

[2] $(x^i)^T$ represents the transpose of x^i

3.1.3 Features

When choosing which features to include in a dataset it is important to consider the relevance of chosen data. It can be complex to know which data is actually relevant in predicting the output of a dataset. One way to estimate this is to consider if the features are lineally dependent. To return to the example of the house price estimation used in section 3.1.1, we could take two features for linear regression that are the area of the house in meters squared and the area of the house in feet squared. As the area of the house in meters squared is a function of the area of the house in feet square, these two features are lineally dependent. These two features cannot therefore be used for linear regression. Features used for linear regression must be linearly independent, that is the features are not dependent on each other. The number of features chosen can drastically effect the hypothesis of linear regression. The features chosen can also be squared or cubed to increase the complexity of the resulting line, which can then better fit the data. Figure 3.12 shows three different types lines to fit the given data. The red line shows a linear function, the blue line is a quadratic function and the yellow line is a quadratic function. As the complexity of the line in increased the line begins to fit the data better and better.

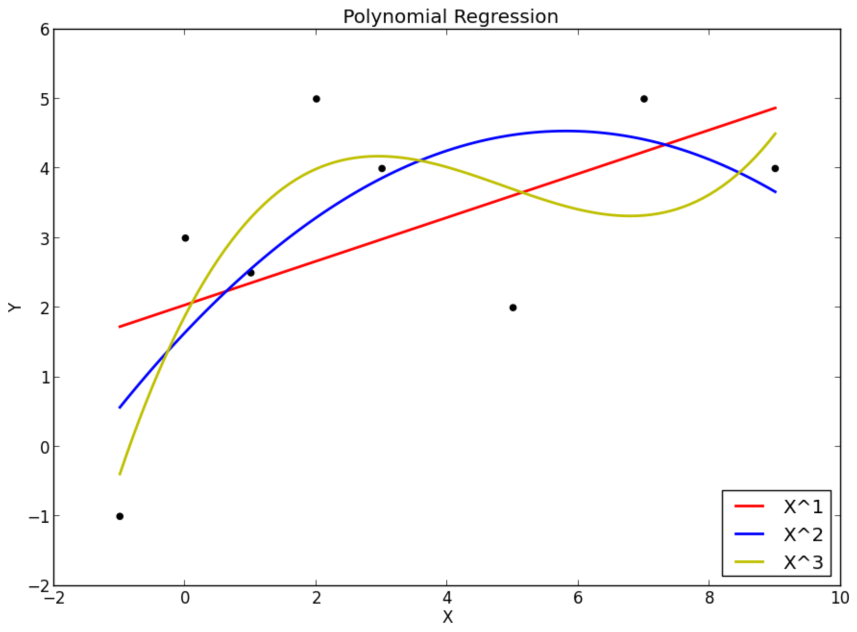


Figure 3.12: Polynomial Regression taken from *Blogger Rocapal*[31]

Increasing the complexity of the line fitting the data is not always desirable. If the data fits to closely to the dataset then the result is hypothesis that maps the relationship between the input and output for this dataset exactly rather than capturing the overall characteristics of the relationship between the two. This is known as overfitting. Figure 3.13 illustrates this problem. The first image shows the problem of under fitting where the line does not fit the data accurately enough to match the characteristic curve of the data. In the second image the line fits the data well, following the characteristic curve of the data. In the third image the line fits every point on the curve exactly, losing the characteristics of the curve.

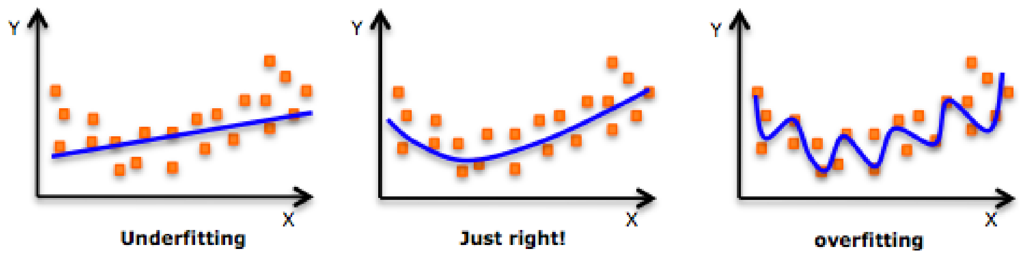


Figure 3.13: Illustration of overfitting taken from *Analytics Vidhya* [27]

The problem of overfitting can be over come by reducing the complexity of the features. As described above this affects the complexity of the line. The dataset used can also be increased or regularisation can be used. The basic principle of regularisation uses a weight decay parameter to penalise values of ϕ that are not zero, this reduces that complexity of the hypothesis.

3.1.4 Logistic Regression

In contrast to linear regression, logistic regression has a discrete valued output. This means that the output is a discrete set of values. Figure 3.14 shows an example of logistic regression. Students wishing to enter college take two exams, their acceptance into college is based on the outcome of the two exams. The data points that red circles are students that were admitted to college and the data points that are blue plus signs are students that were refused to college. Every student was either admitted or refused, this is a discrete valued output. The decision boundary indicates which students were classified as being admitted or refused by the logistic regression model.
Rather than linear regression, in order to determine the hypothesis of a logistic regression a sigmoid function is used. Equation 3.12 is used to create the sigmoid function.

$$g(z) = \frac{1}{1 - e^{-z}} \tag{3.12}$$

Figure 3.15 shows a plot of the sigmoid function. From this plot it can be seen that the output is either one or zero. The hypothesis for logistic regression is given by the following formula

$$h(x) = (y = 1|x, \phi) \tag{3.13}$$

The top of figure 3.16 illustrates a 3D representation of the exam scores against whether the student was admitted or not. The bottom of figure 3.16 shows 2D representations of each side of this graph. Superimposed on this graph is the sigmoid function which illustrates how this data is classified using the sigmoid function. The decision boundary that is need to classify this data can be created using linear regression.

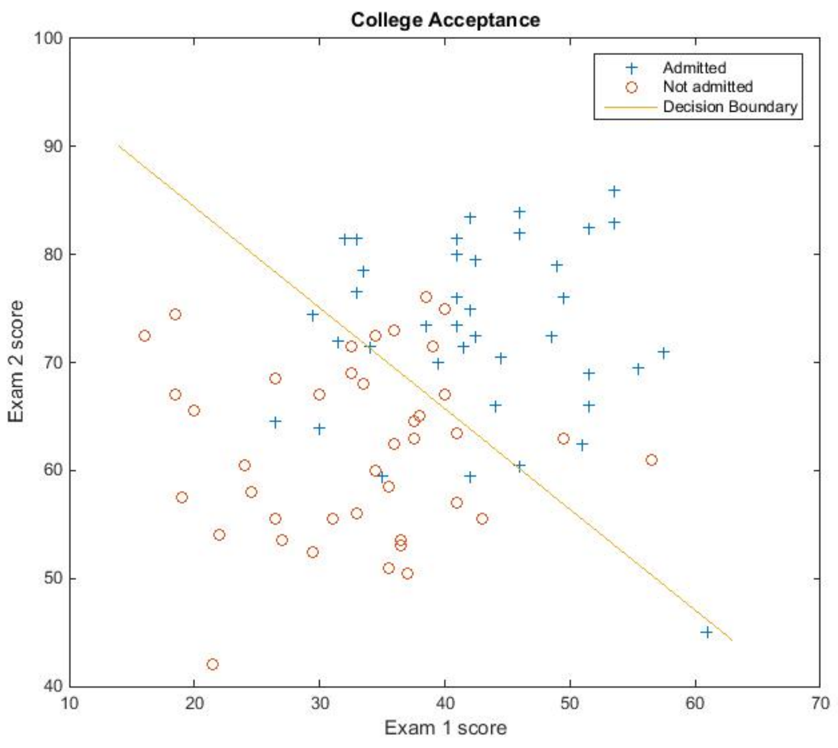


Figure 3.14: College Acceptance Example

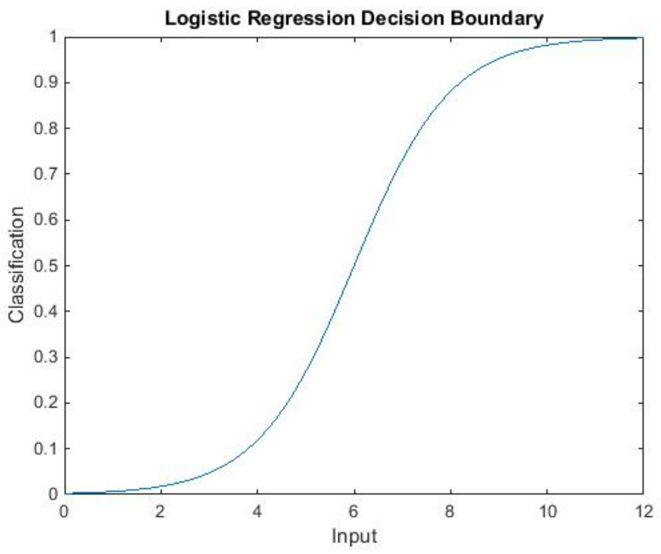
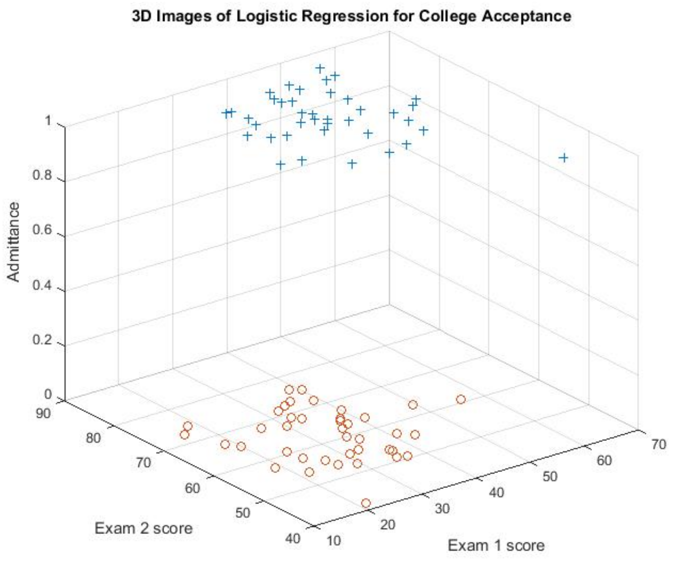


Figure 3.15: Sigmoid Function

Similar to linear regression, in logistic regression a cost function is used to find optimum values of ϕ. If in logistic regression the cost was found by squaring the input and output, like in linear regression, the result would produce a cost function with many maximum and minimum. As described above in section 3.1.2 many local maximum and minimum result in an inefficient gradient descent algorithm. In order



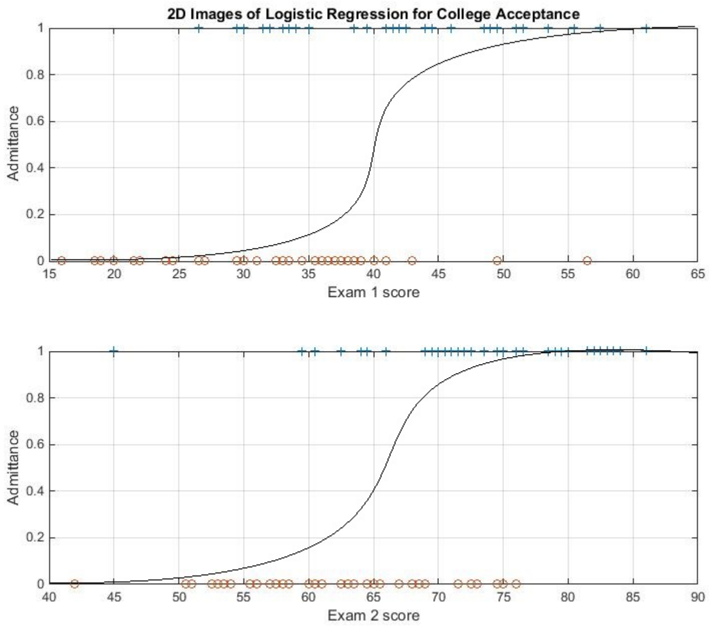
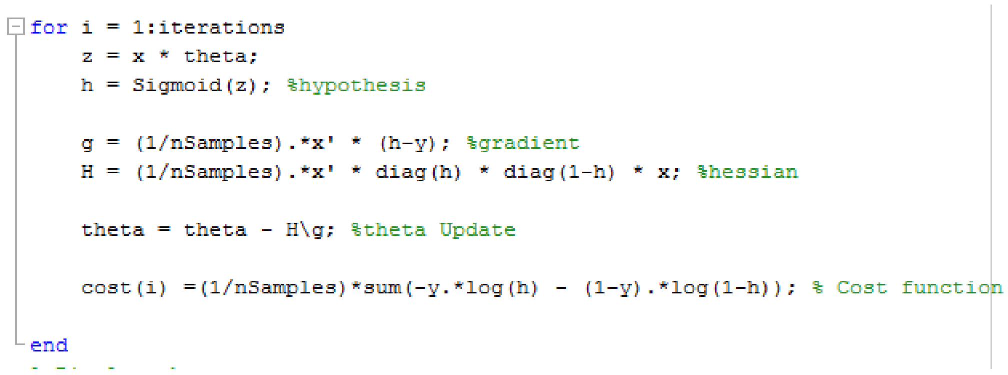


Figure 3.16: Linear Regression and Logistic Regression

to solve this problem another cost function must be found for logistic regression, this is found by finding the maximum likelihood of the sigmoid function.

$$cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases} \tag{3.14}$$

Equation 3.14 is the resulting equation for the cost function for the logistic function. There are appealing features of this function. The first is that the function has only one minimum, making gradient descent more efficient. The second is that this functions punishes the algorithm for producing the incorrect output. As mentioned in section 3.1.1 the purpose of the cost function is to penalise incorrect results. For



```
for i = 1:iterations
    z = x * theta;
    h = Sigmoid(z); %hypothesis

    g = (1/nSamples).*x' * (h-y); %gradient
    H = (1/nSamples).*x' * diag(h) * diag(1-h) * x; %hessian

    theta = theta - H\g; %theta Update

    cost(i) =(1/nSamples)*sum(-y.*log(h) - (1-y).*log(1-h)); % Cost function

end
```

Figure 3.17: Code used to Perform Logistic Regression from Figure 3.14

logistic regression as the function approaches the output zero for an output that should be one, the cost function approaches infinity. Figure 3.17 illustrates the code used to implement logistic regression for the college acceptance example.

3.2 Neural Networks

3.2.1 The Basics

Neural Nets, as the name implies, are made up of neurons. These neurons preform either logistic or linear regression and give an output. This output is then used as a input to the next neuron. This adds another layer of complexity to the computation allows for algorithms that create complex decision boundaries. Figure 3.18 shows the college acceptance problem that was introduced in section 3.1.4. The red line in the diagram shows the decision boundary that would need to be created in order to correctly classify all the data point as admitted or not admitted. This decision boundary could not created using linear or logistic regression it requires a more complex function. More complex decision bounaries can be created using neural networks.

3.2.2 Preceptrons

Preceptrons carried out a process which is carry out by each neuron in a neural net.Figure 3.19 shows a perceptron. A perceptron is takes in inputs, e.g. x_1, x_2 and x_3, and produces an output y. Perceptrons also introduce the concept of weights. In figure 3.19 weights are represented for each of the inputs, w_1, w_2 and w_3. By altering the value of the weights the output of the preceptron can be manipulated. The value of each of the weights for each input is multiplied by each input, these are then summed together along with a bias unit. A bias unit or intercept term is an additional value added to the perceptron in order to manipulate the output. In figure 3.19 the bias units is represented by the symbol b. Once sum of these terms has been found, shown in this figure by the $\sum$ symbol, an activation function in then applied to the output. Generally in artificial intelligence this activation function is a sigmoid function and therefore the perceptrons function much like logistic regression,

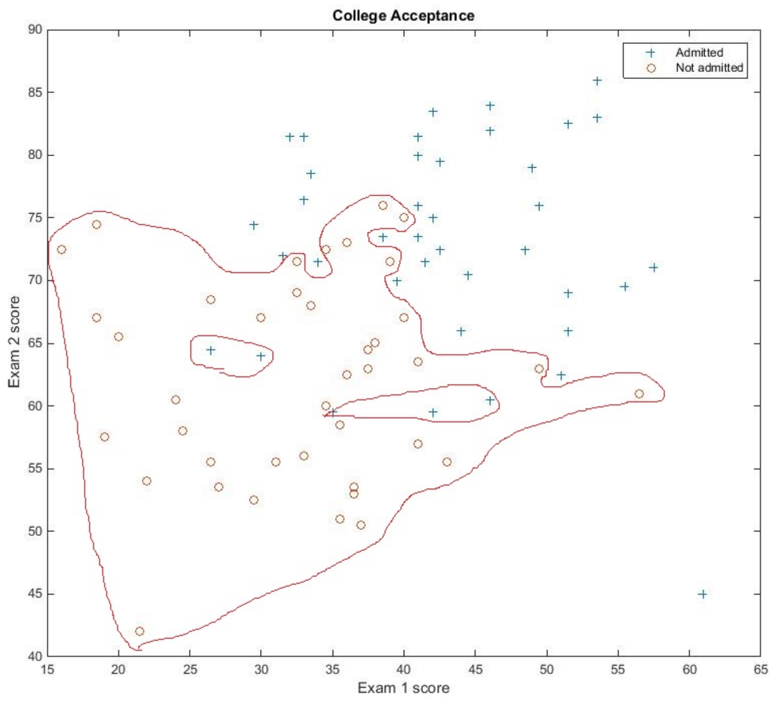


Figure 3.18: Difficult Decision Boundary for College Acceptance Problem

which was discussed in section 3.1.4. However these neurons may also preform other functions. One example of this is a hyperbolic tangent, or tanh function. A tanh function is very similar to a sigmoid function except for the fact that the slope is steeper. This process carried by the preceptron shown in figure 3.19 can summarised by equation 3.15.

$$y = f(x_1w_1 + x_2w_2 + x_3w_3) \tag{3.15}$$

The inputs to these perceptrons are the feature set, as described in previous sections. As well as having these inputs, perceptrons can also have intercept terms. Intercept terms also exist in linear and logistic regression. Equation 3.2, from section 3.1.1,

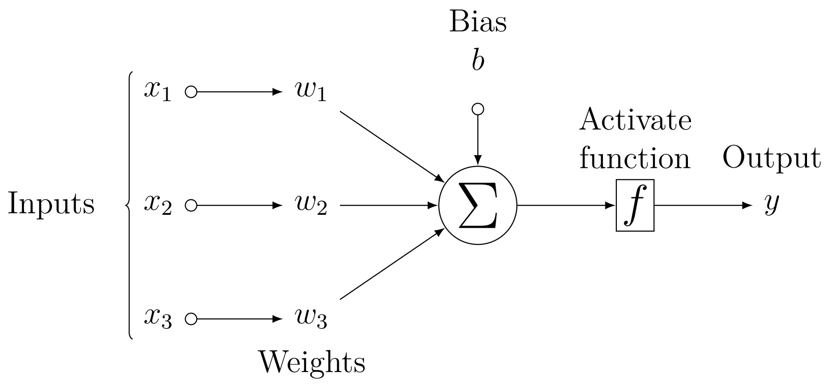


Figure 3.19: Perceptron

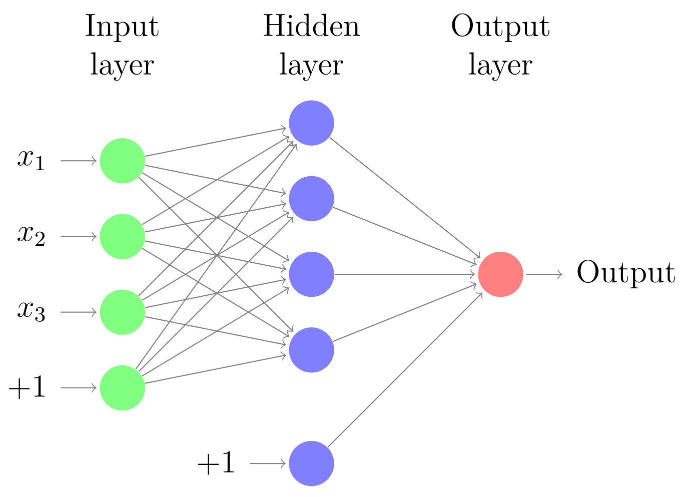


Figure 3.20: Notation for Neural Network

represents the hypothesis for a linear relationship with two features, where x_1 and x_2 are the features. x_0 can be thought of the term multiplied with θ_1, where the value of x_0 is one. x_0 is an intercept term for logistic and linear regression that is always equal to one. In the case of preceptrons and neural networks the intercept term is not always constant and is used to produce more complex outputs.
These preceptrons are what make up the building blocks of a neural network. The first layer is used to make the first decision and the second layer is then used to make decisions based on the outputs of the first later. The outputs of these gates can be even more complex then that of a signal preceptron, by doing this neural nets create more complex decision boundaries.

3.2.3 Neural Network Architecture

Figure 3.20 shows the architecture of a neural net. The green nodes shows the neurons in the input layer L_1, the blue nodes shows the neurons in the hidden layer L_2, the red node shows the neuron in the output layer L_3. The architectures of a neural net can differ depending on the requirements of the neural net. Layer 1 consists of the features input into the net. Layer 2 is the hidden layer and layer 3 is the output. Depending on the requirements of the net there may be more than one hidden layer. Neural nets with several hidden layers are deep neural nets.
In figure 3.20, each of the features, denoted with x_i, are inputs to the neural network shown on the left of the diagram as x_1 to x_4. +1 is the intercept term, in this neural network there are two intercept terms in both layers 1 and 2. The intercept term is denoted as b_i^L, where L is the layer the weight is associated with and i is the neuron in the next layer the bias effects.Each of these inputs also has a weight associated with it, w_{ji}^L where L is the layer the weight is associated with, i is the neuron the weight effects and j is the next layer neuron that it effects. For example w_{11}^2 is the weight on the input x_1 going to the first neuron in layer two. The output of each neuron is the activation and is denoted with a_i^L where again L is the layer the weight is associated with and i is the neuron the weight effects. For example a_1^2 is the output of the first neuron in the second layer. The output of this neuron

is a function of all its inputs. This function depends on the function used within the neuron, as described in section 3.2.2. The activation function used within the neuron is denoted by f. Equation 3.16 and 3.17 denote the activations in layer one and two in this neural network in figure 3.20.

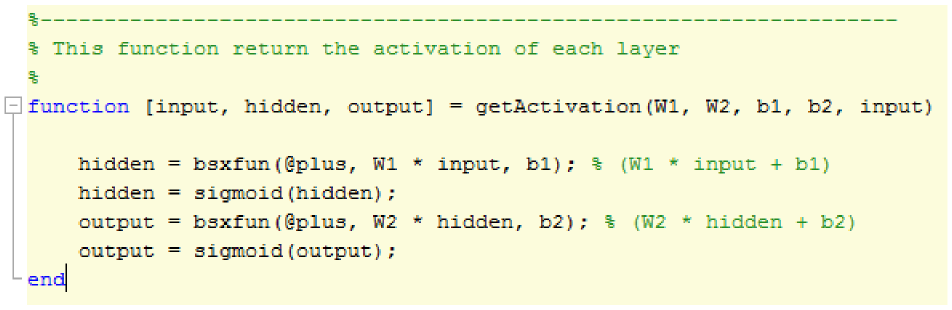
$$a_1^2 = f(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1) \tag{3.16}$$

$$a_2^2 = f(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1) \tag{3.17}$$

The output of the entire neural network is, similar to logistic and linear regression the hypothesis. The hypothesis is denoted by $h_{w,b}(x)$, it is a function of the input, (x) and of the weights and biases in the network. For this neural network the hypothesis is:

$$h_{w,b}(x) = a_1^3 = f(w_{11}^2 a_1^2 + w_{12}^2 a_2^2 + b_1^2) \tag{3.18}$$

Figure 3.21 shows the Matlab code for calculating the activation of each layer. The function *bsxfun* calculates the weights by the inputs plus the intercept terms. The following line calculates the activation function, that is thew sigmoid function, of the output. The activation of both the hidden and output layers are calculated using this method, the input however does not change it is therefore simply returned.



```
%------------------------------------------------------------------
% This function return the activation of each layer
%
function [input, hidden, output] = getActivation(W1, W2, b1, b2, input)

    hidden = bsxfun(@plus, W1 * input, b1); % (W1 * input + b1)
    hidden = sigmoid(hidden);
    output = bsxfun(@plus, W2 * hidden, b2); % (W2 * hidden + b2)
    output = sigmoid(output);
end
```

Figure 3.21: Code for calculating the Activation of each Layer in Matlab

Figure 3.20 is a feed forward network, this means that the network flows from layer 1 to layer 2 and so on. However there can be other kinds of networks that also reverse direction, these are recurrent neural networks. Recurrent neural nets have been less influential than feed forward networks, in part because the learning algorithms for recurrent nets are (at least to date) less powerful[20]. Networks can also have two outputs two outputs may represent whether two different conditions that a neural network is attempting to detect. When the output from one layer is passed to the next this is known as feed forward propagation. There is also backward propagation's that can be used for a variety or reasons, this will be discussed in the section 3.3.3.

3.2.4 Exclusive Or (XOR) Gate Example

In order to explain how neural nets can create more complex decision boundaries than logistic or linear regression, the example of a decision boundary can classifies an XOR gate can be used. By altering the weights and the intercept term the output of

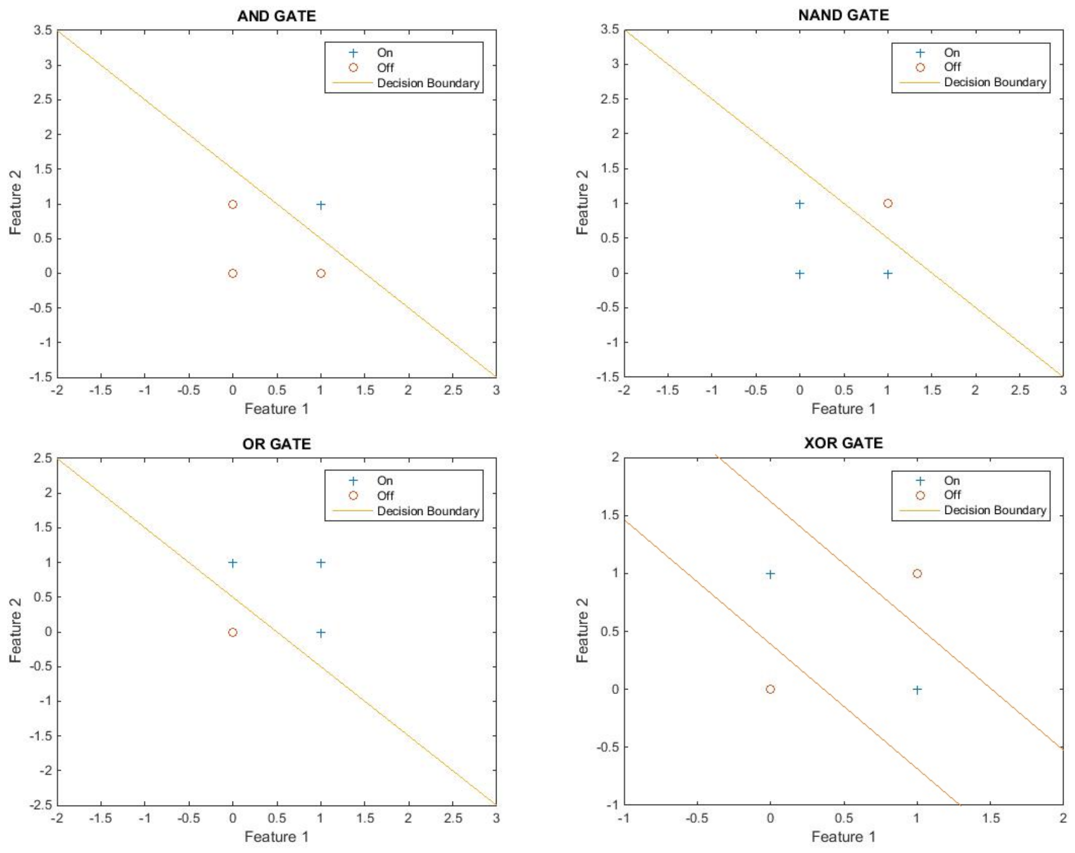


Figure 3.22: Logical Gates

a neural net can be altered. Figure 3.22 illustrates the decision boundaries required for a AND, NAND, OR and XOR gates. There are two input features which are either zero or one, the output of the gate is either on, illustrated with a blue plus sign, or off, illustrated with a red circle. The decision boundary for each gate is shown in orange. AND, NAND and OR gates are all decision boundaries that can be created using logistic regression. However XOR have more complex decision boundaries that cannot be classified by logistic regression. The XOR gate is a non-linearly separable pattern[9]. These types of patterns are of common occurrence. and require neural nets to be solved.

Weights and Biases for XOR example													
x_1	x_2	w^1_{11}	w^1_{12}	b^1_1	a^2_1	w^1_{21}	w^1_{22}	b^1_2	a^2_2	w^2_{11}	w^2_{21}	b^2_1	y
0	0	0.5	0.5	-0.75	0	-0.75	-0.75	0.5	1	-0.75	-0.75	0.5	0
0	0	0.5	0.5	-0.75	0	-0.75	-0.75	0.5	0	-0.75	-0.75	0.5	1
1	0	0.5	0.5	-0.75	0	-0.75	-0.75	0.5	0	-0.75	-0.75	0.5	1
1	1	0.5	0.5	-0.75	1	-0.75	-0.75	0.5	0	-0.75	-0.75	0.5	0

Table 3.2.4 shows the inputs, weights and biases and outputs that are required to create the decision boundary for the XOR function. This Table represents a neural net with an input layer with two inputs and a bias unit, one hidden layer with two neurons and bias unit and an output. Each of the weights and biases are labelled using the notation from section 3.2.3. a^2_1 and a^2_2 represent the activation functions of

the two neurons in the hidden layer. This table shows that activation a_1^1 represents that of an AND gate and a_2^2 represents that of a NOR gate. This illustrates how both AND and NOR gates can be created using a single neuron, therefore they can both be created using logistic regression. In fact AND, NAND and OR gates can all be created using logistic regression. This can be seen in figure 3.22, each of these gates require a simple decision boundary. On the other hand XOR gates require a more complex decision boundary, therefore a neural net most be used.
Although using neural nets to create logical gates can be helpful, this does not reflect the full advantage of neural nets. In the XOR example described above the biases and weights were constants that do not change throughout the process. Because these weights and biases are remaining constant the output produced is not changing or learning anything for its computation. Learning algorithms are be devised to automatically tune the weights and biases of the network of artificial neurons[20].

3.3 Neural Net Optimisation

3.3.1 Next Generation of Neural Nets

While preceptrons use hand coded features and weights. The second generation of neural nets overcame this problem by dynamically assigning the weights and features. Rather giving the neural net features to identify these networks are simply given the data and the most relevant features are identity from the data. The advantage of this technique is that individuals do not have to identify for the neural network the most relevant parts of the data. The neural network can in fact choose the most relevant features based on the data.
Back propagation also improves the accuracy of these nets. Back propagation works by taking the output and identifying the errors in the output compared with the expected values. The outputs which are incorrect are then back propagated through the neural net to find the weight or bias or caused the data to be classified incorrectly. This weight or bias can then be changed to increase the accuracy of the neural net. Before improvements in computational speed, the result of back propagation had many limitations as it was inefficient on less powerful machines. During this time popularity grew in other methods can could provide better accuracy then back propagation. Support Vector Machines where one example of these alternative methods. Support Vector Machines were effectively intelligent preceptrons that functions by comparing training examples and efficiently optimising the algorithms by discarding the features which lead to incorrect results and maintaining the successful features. Support Vector Machines performed better than back propagation however unlike neural networks these machines where not actually learning. The algorithm for determining the output remained the same.
The Restricted Boltzmann Machine was another type of neural net that was restricted in size, by this we mean that the number of hidden layers was restricted to just one. The advantage of this method was that this method was efficients due to its restricted size. As the computational power increases more of the concepts that had been developed could be further realised, such as the ability to learn features.

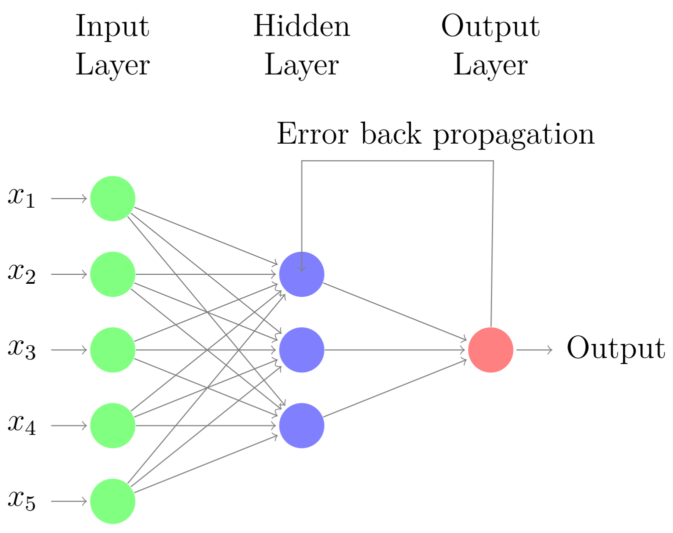


Figure 3.23: Back Propagation in a Neural Network

Features are extracted from the data using deep nets. At each layer of the net extracts features from the previous layer, creating more and more abstract features. Features are chosen if they alter the data into the behaviour we wish to see from the training set. Back Propagation can then be used to fine tune the dataset.[3]

3.3.2 Symmetry Breaking

Earlier on in this chapter in the section 3.1.2, the principle of gradient descent was discussed as a method for finding the parameters of a hypothesis. For neural networks each neuron outputs an activation function that in a feed forward network is sent on to the next layer in the neural network. Gradient Descent must be performed on each of these neurons in order to find its output. In a neural network the parameters of the output, previously referred to as ϕ_i, are in a neural network the weights associated with each input into a neuron, w^L_{ji} as described in section 3.2.3.

When performing gradient descent for linear regression each of these parameters or weights can be initialised as zero and then gradient descent finds the their optimum values. While this procedure functions well for linear and logistic regression the same technique cannot be used for a neural network. If all the weights in a neural network were initialised as zero and each had the same function, there the result of gradient descent would be the same for all the weights of the hidden nodes within the network. The hypothesis of such a network would not be very useful. In order to negate this problem the weights are initialised randomly, this is known as symmetry breaking.

3.3.3 Backpropagation Error Algorithm

Backpropagation is a method of improving the performance of a neural network. Error backpropegation is a method of fixing the intercept terms and weights to optimise the performance of the network. Error backpropegation was a break through algorithm in artificial intelligence that led to the wide spend use of neural networks. The backpropagation algorithm was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams[20]. Prior to this due to the difficulty in optimising deep neural networks restricted neural networks were mainly used, as they were easier to optimise. In recent years with increasing computing power backpropegation has become even more popular as it is now able to train much larger networks.
Similar to linear and logistic regression, neural networks need to be trained. In the same way that logistic and linear regression are trained, neural networks are trained by using minimising a cost function using gradient descent. The cost function for neural networks is defined by the equation 3.19. Rather than optimising theta, as for linear and logistic regression, the cost function for neural networks optimises the weight and biases. Overfitting, the phenomenon discussed in section 3.1.3, can also occur in neural networks were the solution to the neural network fits too closely to the training data, rather than fitting the overall characteristic of the training data. Regularisation, or weight decay as it is called for neural networks, is introduced to minimise this problem. In weight decay, weights other than with a value of zero are penalised. Weight decay is not applied to bias terms. Equation 3.20 displays the cost function with the weight decay term. Lambda in this equation is a constant and m is the number of training samples used.

$$J(W, b, x, y) = \frac{1}{2}||h_{W,b}(x) - y||^2 \tag{3.19}$$

$$J(W, b) = [\frac{1}{m}\sum_{i=1}^{m}(\frac{1}{2}||h_{W,b}(x) - y||^2)] + \frac{\lambda}{2}\sum_{l=1}^{n_t-1}\sum_{i=1}^{s_t}\sum_{j=1}^{s_t+1}(W_{ji}^l)^2 \tag{3.20}$$

The goal is now to minimise the weights and biases for the cost function, equation 3.20. Since $J(W, b)$ is a non-convex function, gradient descent is susceptible to local optima [19]. Equation 3.21 displays one iteration of gradient descent for a neural network. This is essentially the same as equation 3.5 from section 3.1.2, where α is the learning rate of gradient descent. The difficulty in solving equation 3.21 and equation 3.22 is solving the partial derivative, backpropegation gives an efficient way of solving the partial derivative.

$$W_{ij}^l = W_{ij}^l - \alpha\frac{\delta}{\delta W_{ij}^l}J(W, b) \tag{3.21}$$

$$b_i^l = b_i^l - \alpha\frac{\delta}{\delta b_i^l}J(W, b) \tag{3.22}$$

[3]This information originates from tutorials given by Hilton.[11]

The principle of error backpropegation is that first a "forward pass" is computed to calculate the activations throughout the network, including the output value of the hypothesis $h_{W,b}(x)$. After this is performed the error of hypothesis is found using training data. This error value can then be used to determine the which nodes are most responsible for causing this error. The error for the entire network is found by equation 3.23 where nl is the output layer of the network and z_i^l is the total weighted sum of inputs to unit i in layer l.

$$\delta_i^{nl} = -(y_i - h_{W,b}(x)).f'(z_i^{nl}) \tag{3.23}$$

where $f'(z_i^l)$ is given by:

$$f'(z_i^l) = a_i^l(1 - a_i^l) \tag{3.24}$$

While this function gives the error of the overall network. It does not indicate which weight and biases in particular are responsible for the error. For each node the error of the node in next layer must be must be found. That is in order to find δ_i^l, δ_i^{l+1} must first be found. δ_i^{l+1} can be found by computing the partial deviate of the cost function. Once this has been computed the error for each node can be found by evaluating equation 3.25.

$$\delta_i^l = (\sum_{j=1}^{nl} w_{ji}^l \delta_i^{l+1}) f'(z_i^l) \tag{3.25}$$

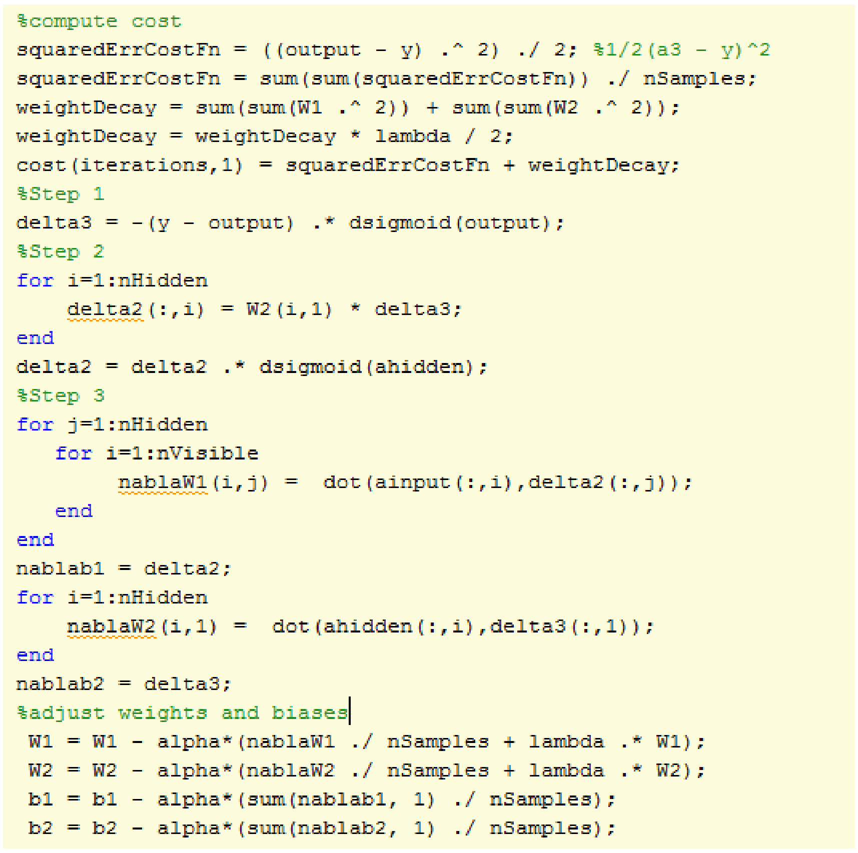
Once the error for each neuron has been determined, the partial derivatives for the weight and biases can be found. Equation 3.26 and equation 3.27 show how the partial derivatives are finally calculated. Each iteration of gradient descent can finally be calculated using these partial derivatives and the cost function can be minimised by altering the weights and biases according to how they attributed to the final error of the network.

$$\frac{\delta}{\delta W_{ij}^l} J(W, b, x, y) = a_j^l \delta_i^{l+1} \tag{3.26}$$

$$\frac{\delta}{\delta b_i^l} J(W, b, x, y) = \delta_i^{l+1} \tag{3.27}$$

Figure 3.24 shows Matlab code that calculates the backpropagation error in a neural network. The first part of the code calculates the cost function and the weight decay for network, i.e. equation 3.19 and equation 3.20. The error of the entire network is then calculated as the variable *delta3*, i.e. equation 3.23. The error for the hidden layer of the network is then calculated as *delta2*, i.e. equation 3.25. *nablaW1*, *nablaW2*, *nablab1* and *nablab2* represent the partial derivatives for the weights and biases in the network. Finally the weights and biases are adapted to reduce the cost function.

Although backpropagation improves the ability to train neural networks, as neural networks get deeper, the ability to effectively train the network becomes more difficult. According to Learning Deep Architectures for AI "gradient-based training of deep super- vised multi-layer neural networks (starting from random initialization)gets stuck[ed] in apparent local minima or plateaus”[2]. However according to this same source the performance of these deep neural nets can be improved by



```
%compute cost
squaredErrCostFn = ((output - y) .^ 2) ./ 2; %1/2(a3 - y)^2
squaredErrCostFn = sum(sum(squaredErrCostFn)) ./ nSamples;
weightDecay = sum(sum(W1 .^ 2)) + sum(sum(W2 .^ 2));
weightDecay = weightDecay * lambda / 2;
cost(iterations,1) = squaredErrCostFn + weightDecay;
%Step 1
delta3 = -(y - output) .* dsigmoid(output);
%Step 2
for i=1:nHidden
    delta2(:,i) = W2(i,1) * delta3;
end
delta2 = delta2 .* dsigmoid(ahidden);
%Step 3
for j=1:nHidden
   for i=1:nVisible
        nablaW1(i,j) =  dot(ainput(:,i),delta2(:,j));
   end
end
nablab1 = delta2;
for i=1:nHidden
    nablaW2(i,1) =  dot(ahidden(:,i),delta3(:,1));
end
nablab2 = delta3;
%adjust weights and biases
 W1 = W1 - alpha*(nablaW1 ./ nSamples + lambda .* W1);
 W2 = W2 - alpha*(nablaW2 ./ nSamples + lambda .* W2);
 b1 = b1 - alpha*(sum(nablab1, 1) ./ nSamples);
 b2 = b2 - alpha*(sum(nablab2, 1) ./ nSamples);
```

Figure 3.24: Code for Calculating the Error Backpropagation in Matlab

pre-training the network using unsupervised learning. The concept of unsupervised learning is discusses in the next section 3.4.1.

3.4 Feature Learning

3.4.1 Unsupervised Learning

There are many different types of machine learning algorithms. Supervised Learning is when the algorithm is given the desired output for a dataset and this dataset is then used to train the algorithm that is used to classify future data. Supervised learning describes all the examples that have been seen in the section 3.1. For all of these examples the desired output of the data was known and this allowed the desired output of data to be determine either as a continuous valued output as in regression (Linear Regression in the section 3.1.1) or as a discrete valued output as in classification (Logistic Regression in the section 3.1.4).
Unlike supervised learning in unsupervised learning the desired output of the dataset is not know. Often in unsupervised learning clustering can be used to determine the number of different dataset that need to classified. Clustering is used in a wide field of applications and is a very useful tool for unsupervised classification, examples of this can be gene classification, social network analysis, or market segmentation.

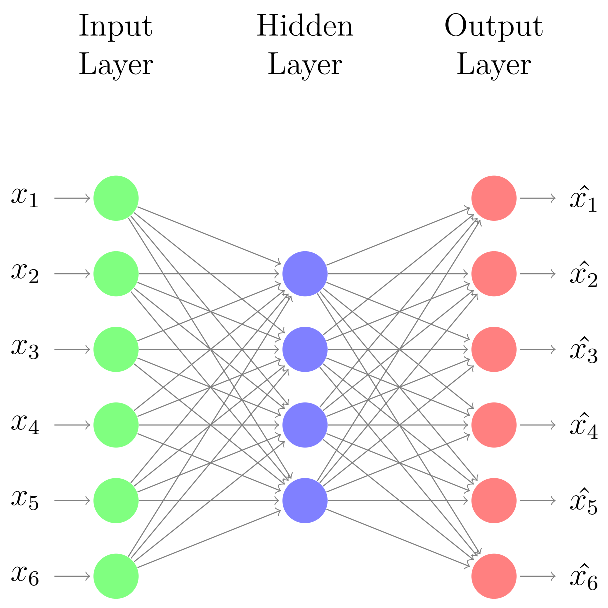


Figure 3.25: Autoencoder

3.4.2 Autoencoders

Supervised learning is a very useful AI technique that allows algorithms to learn a variety of complex data. However this technique does have drawbacks. In order for supervised neural networks to output valuable representations of data, relevant features must used as inputs. in some cases relevant features can be easily identified however this is not always the case. In applications such as computer vision choosing relevant features can be extremely complex and time consuming. In addition to that the neural networks that are trained from such features are not flexible and cannot be adapted to different problems.

An autoencoder neural network is an unsupervised learning algorithm that applies back propagation, setting the target values to be equal to the inputs. [19] What this in effect equates to is the ability to learn features from unlabelled data. Figure 3.25 shows an autoencoder. In Layer 1 inputs are taken in and layer 2 variations of these inputs are output and these can be utilised later in the neural network. By using backpropagation an autoencoder is trained to encode the input x_i to a representation $\hat{x}_i$ to that it can be reconstructed. The idea behind using a autoencoder is that through optimisation the weights and biases of the autoencoder are altered so that new representations of the input, $\hat{x}_i$, capture the main features of the variation in the data. Once the main features of the variation in the data have been captured these can then be used as the features for the network rather than the original inputs themselves.

In figure 3.25, the number of hidden units is less than the number of input units. This means that the data must be compressed into the number of hidden units. This operation can revel interesting representations in the data that is output.If all the input data is random then compressing the data into a fewer number of hidden units makes the reconstruction of the data into the output very difficult. However if the data is not random and it has features that are correlated, this neural network can discover those correlated features.

3.4.3 Sparsity

As was discussed in section 3.1.4 on logistic regression, the output of a sigmoid function is between zero and one. Inside each of the neurons a sigmoid function is used to determine the output. As discussed early in section 3.2.2 this sigmoid function is referred to as the activation function in the context of neural networks. This means that the output of each of the neurons is between zero and one. If the output of a neuron is close to one then that neuron can be thought of as being active. On the other hand if the output of the neuron is zero then the neuron can be thought of as being inactive. As described in section 3.2.3, on the architecture of neural networks the output of a neuron is referred to as the activation of that neuron, given by the equation 3.16. Therefore the activation of each neuron can be active or inactive. Whether a neuron is active or inactive depends on the input from the previous layer.

For a hidden unit there are activations for each of the training examples put through the neural network. The average activation of a hidden unit over a number of training examples is given by equation 3.28 where m is the number of training examples and a_j^2 is the j^{th} activation from the second layer of the neural network with is a function of the inputs to the neural network (x^i). This is the average activation of the j^th unit of the hidden unit. In figure 3.25, the number of j would be four, as this is the number of units in layer 2. The number of features would be six.

$$\hat{p_j} = \frac{1}{m}\sum_{i=1}^{m}[a_j^2(x^i)] \tag{3.28}$$

The average activation of each hidden unit is a parameter that can be set as a constant. This constant is the sparsity parameter. If the sparsity parameter is close to zero then the majority of the hidden units must be inactive. If the sparsity parameter is close to one then the majority of the hidden units must be active. This method of setting the sparsity parameter is another method that can be used by autoencoders to find correlations in data. As discussed in section 3.4.2, on autoencoders, interesting structures in the data can be uncovered by limiting the number of hidden units in neural network and reconstructing the data. This is however not the only to method to uncover interesting patterns in the data. By specifying a lower sparsity parameter, limitations are also imposed on the neural network that mean that the data has to restricted much as it is done when the number of hidden units is lower than the number of input and output units.

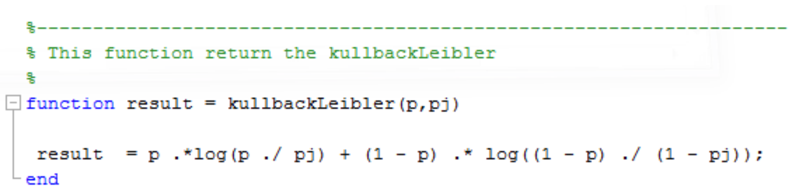
In order to impose the sparsity parameter on the hidden units another function is added to the cost function. This function acts similarly to regularisation, where a parameter is introduced to penalise the values of weights in the cost function that are not equal to zero. This purpose of regularisation is to reduce the likelihood of over fitting. With the sparsity parameter a formula is added to the cost function that penalises average activations, $\hat{p_j}$, that differ from the sparsity parameter.

Equation 3.29 shows the cost function with the sparsity parameter. β dictates the weight of the sparsity parameter on the data. The sparsity parameter is p.

$$J(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} KL(p||\hat{p_j}) \tag{3.29}$$

KL represents the Kullback Leibler function. Equation 3.30 displays the Kullback Leibler function. This function is minimised when $p = \hat{p_j}$ and reaches infinity as the disagreement between p and $\hat{p_j}$ increases. The effect of this is that if the average activation of the neural network differs greater from the sparsity parameter the cost function increases at a rate dictated by the weight parameter β. Figure 3.26 shows code for implementing the Kullback Leibler function in Matlab code. This function return the disparity between the average activation and the sparsity parameter.

$$\sum_{j=1}^{s_2} KL(p||\hat{p_j}) = plog\frac{p}{\hat{p_j}} + (1-p)log\frac{1-p}{1-\hat{p_j}} \tag{3.30}$$



```
%-------------------------------------------------------------------
% This function return the kullbackLeibler
%
function result = kullbackLeibler(p,pj)

 result  = p .*log(p ./ pj) + (1 - p) .* log((1 - p) ./ (1 - pj));
end
```

Figure 3.26: Matlab Code for Kullback Leibler Function

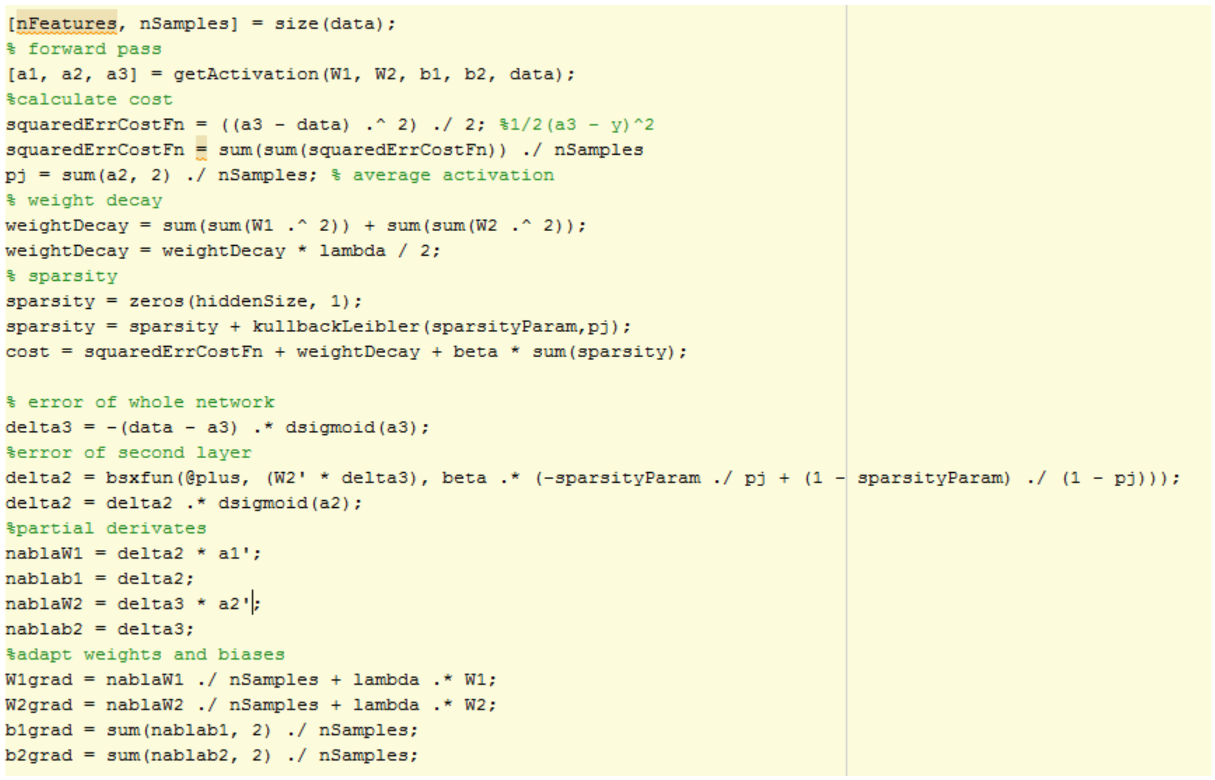
In order to incorporate the sparsity parameter into the backpropagation algorithm the sparsity is incorporated into the calculation of the error in layer two. Equation 3.31 shows the same equation as equation 3.25, in section 3.3.3, except a sparsity part of the equation has been added on. The term $\hat{p_i}$ refers to the action of the particular neuron in question. A forward pass on the neural network can be computed first that retains the activation of each of neurons. These values can then be used to calculate the error of each neuron.

$$\delta_i^l = (\sum_{j=1}^{nl} w_{ji}^l \delta_i^{l+1}) + \beta(-\frac{p}{\hat{p_i}} + \frac{1-p}{1-\hat{p_i}})f'(z_i^l) \tag{3.31}$$

An autoencoder than compresses the data by using a sparsity parameter is known as a sparse autoencoder. Figure 3.27 shows the Matlab code for creating a sparsity autoencoder. A forward pass is first run on the neural net. This determines the activations of each of neurons. The first part of the cost function is then evaluated as the *squaredErrCostFn*. One important difference in calculating the first part of the cost function for the autoencoder is that rather than $h_{W,b}(x) - y$ as is depicted by equation 3.19, in section 3.3.3, this part of the cost function is calculated as $h_{W,b}(x) - input$[4]. The reason for this is that for the autoencoder the expected output is the input, this equation is therefore adapted to find the difference between the output and the input. The next part of the code calculates the weight decay and

[4]In the code shown in figure 3.27 the input is displayed as *data*.

the sparsity. The Kullback Leibler is calculated in the equation shown in figure 3.26. Once all three part of the cost function have been calculated the entire cost function can be calculated. Similar to the first part of the cost function, the squared error cost, the error of the entire neural network is calculated using the input as the expected value. The calculation of the error in the second layer of the network is adapted to include the sparsity calculation, as in equation 3.31. The calculation of the partial derivatives and the adaption of the weights and biases does not change from the method used for backpropagation.[5]



```
[nFeatures, nSamples] = size(data);
% forward pass
[a1, a2, a3] = getActivation(W1, W2, b1, b2, data);
%calculate cost
squaredErrCostFn = ((a3 - data) .^ 2) ./ 2; %1/2(a3 - y)^2
squaredErrCostFn = sum(sum(squaredErrCostFn)) ./ nSamples
pj = sum(a2, 2) ./ nSamples; % average activation
% weight decay
weightDecay = sum(sum(W1 .^ 2)) + sum(sum(W2 .^ 2));
weightDecay = weightDecay * lambda / 2;
% sparsity
sparsity = zeros(hiddenSize, 1);
sparsity = sparsity + kullbackLeibler(sparsityParam,pj);
cost = squaredErrCostFn + weightDecay + beta * sum(sparsity);

% error of whole network
delta3 = -(data - a3) .* dsigmoid(a3);
%error of second layer
delta2 = bsxfun(@plus, (W2' * delta3), beta .* (-sparsityParam ./ pj + (1 - sparsityParam) ./ (1 - pj)));
delta2 = delta2 .* dsigmoid(a2);
%partial derivates
nablaW1 = delta2 * a1';
nablab1 = delta2;
nablaW2 = delta3 * a2';
nablab2 = delta3;
%adapt weights and biases
W1grad = nablaW1 ./ nSamples + lambda .* W1;
W2grad = nablaW2 ./ nSamples + lambda .* W2;
b1grad = sum(nablab1, 2) ./ nSamples;
b2grad = sum(nablab2, 2) ./ nSamples;
```

Figure 3.27: Matlab Code for a Sparse Autoencoder

3.4.4 Softmax Regression

The previous examples in this document of classification, such as neural networks and logistic regression, illustrate the ability to classify two possible values. This is known as binary classification. However there are also classes were there are more than two possible values or labels. This is known as multi-label classification. In the example used about for logistic regression, in section 3.1.4, student are either admitted or rejected from college based on the outcome of two exams. Say there was a third possible outcome, such as if students scored just below the level of admittance in both exams, they would have the opportunity to retake the exams. This would

[5]The difference in the code for the calculation of the partial derivatives from figure 3.24, of the code for backpropagation, and figure 3.27, of the code for the sparse autoencoder, is due to the fact that the code for backpropagation is written to function for many hidden layers whereas the code for sparse autoencoder is for only one hidden layer.

make this classification problem a multi-label classification problem, where there are three possible outcomes either the student is admitted, rejected or told to retake the exams.
For logistic regression the hypothesis is given by the sigmoid function, this is shown in section 3.1.4 in equation 3.12. If logistic regression were extended to a multi-label classification problem the hypothesis would need to be adapted to produce more than just two possible outcomes. For this problem the probably of each possible outcome needs to be calculated. Equation 3.32 shows the hypothesis for multi-label classification logistic regression. The hypothesis for multi-class logistic regression is the calculation of the probability of the output, y being $1, 2....k$, where k is the number of possible outcomes, given a particular input x^i and θ. This is depicted in the first part of the equation.

$$h_\theta(x^i) = \begin{pmatrix} p(y^i = 1|x^i;\theta) \\ p(y^i = 2|x^i;\theta) \\ \vdots \\ p(y^i = k|x^i;\theta) \end{pmatrix} = \frac{1}{\sum\limits_{j=1}^{k} e^{\theta_j^T x^i}} \begin{pmatrix} e^{\theta_1^T x^i} \\ e^{\theta_2^T x^i} \\ \vdots \\ e^{\theta_k^T x^i} \end{pmatrix} \tag{3.32}$$

The second part of equation 3.32, on the right hand side, shows the calculation of the probability of each possible outcome. The probability of each possible outcome is calculated using the sigmoid function, where the values of θ are altered for each possible output. The term θ as described in section 3.1.4, refers to the parameters of the model that are altered to optimise the classification. For multi-class logistic regression there is are different parameters, θ, for each of the different possible outputs. This is illustrated in equation 3.33. θ^T represents the transpose of θ.

$$\theta = \begin{pmatrix} \theta_1^T \\ \theta_2^T \\ \vdots \\ \theta_k^T \end{pmatrix} \tag{3.33}$$

Equation 3.34 shows the cost function for logistic regression, where $1[y^i = j]$ equals one for a true state and zero for a false statement. This cost function also applies for multi-label logistic regression, however the value of $p(y^i = j|x^i;\theta)$ differs for binary and multi-label classification. Equation 3.35 shows the value of $p(y^i = j|x^i;\theta)$ for multi-label classification for logistic regression. There are however issues with this equation the cost function of logistic regression for multi-lable classification. If an additional parameter is added to equation 3.35 it does not effect the hypothesis. This proves that the parameters in this equation are redundant, or that the equation is overparameterised. This means that there are several different values for the parameters that give the same value for the hypothesis. One method of solving this problem, of overparametrisation, is to use weight decay.

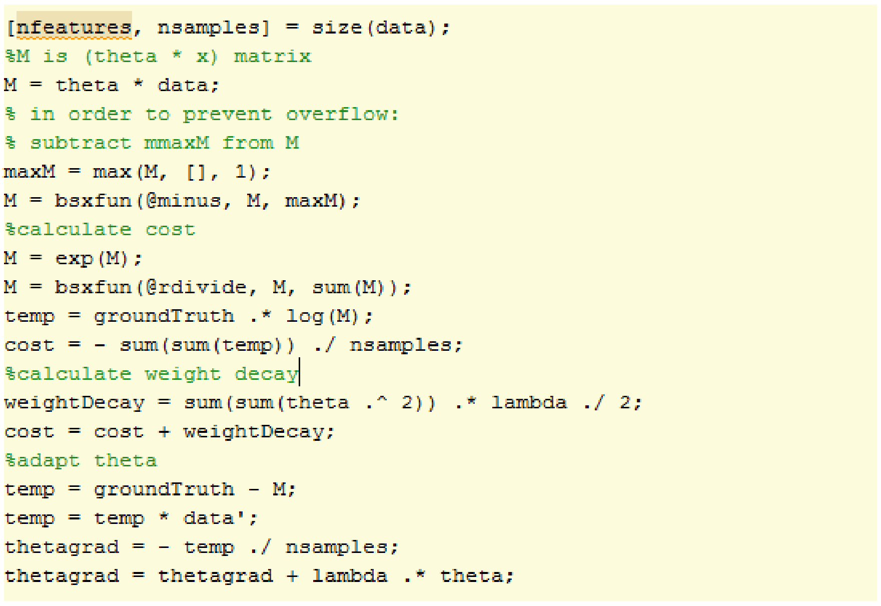
$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{j=0}^{1} 1[y^i = j] log(p(y^i = j|x^i;\theta))] \tag{3.34}$$

$$p(y^i = j|x^i;\theta) = \frac{e^{\theta_j^T x^i}}{\sum_{l=1}^{k} e^{\theta_j^T x^i}} \tag{3.35}$$

Weight decay, as described in section 3.3.3, is a method of penalising weights that differ from zero. Equation 3.36 shows the cost function for multi-label logistic regression with weight decay. Weight decay is controlled by a parameter λ. This cost function now has only one unique solution for minimsing the cost function.

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=0}^{1} 1[y^i = j] log\left(\frac{e^{\theta_j^T x^i}}{\sum_{l=1}^{k} e^{\theta_j^T x^i}}\right)\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2 \tag{3.36}$$

Figure 3.28 shows the code for implementing softmax regression. M represents the matrix of the input by the parameters shown in equation 3.32. If values in matrix, M, are large then there is strong likelihood that overflow could occur. To avoid this the maximum value in the matrix is subtracted from each of the values in the matrix, to prevent overflow. The variable *groundTruth* in the code presents the function $1[y^i = j]$ and *temp* contains the part of the equation 3.36 inside the square brackets. Once the first part of the cost has been calculated the weight decay is then calculated and the values of θ can be adapted to minimise the cost function.



```
[nfeatures, nsamples] = size(data);
%M is (theta * x) matrix
M = theta * data;
% in order to prevent overflow:
% subtract mmaxM from M
maxM = max(M, [], 1);
M = bsxfun(@minus, M, maxM);
%calculate cost
M = exp(M);
M = bsxfun(@rdivide, M, sum(M));
temp = groundTruth .* log(M);
cost = - sum(sum(temp)) ./ nsamples;
%calculate weight decay
weightDecay = sum(sum(theta .^ 2)) .* lambda ./ 2;
cost = cost + weightDecay;
%adapt theta
temp = groundTruth - M;
temp = temp * data';
thetagrad = - temp ./ nsamples;
thetagrad = thetagrad + lambda .* theta;
```

Figure 3.28: Matlab Code for a Softmax Regression

In the same way that logistic regression can be adapted for softmax regression neural networks can also be adapted to classify more than two possible labels.

4

Machine Learning For Sentiment Analysis

This chapter describes some of the methods that are used in order to preform sentiment analysis. These include methods that have been traditionally used for this purpose such as the naïve Bayes model. As well as methods that are the current state of the art in this area such as the Stanford implementation of Sentiment Analysis from Socher et al.[25]. Finally this section describes the method that was implemented in this project. This method was compared against the Stanford implementation of Sentiment Analysis from Socher et al.[25] in order to assess its effectiveness in analysing tweets compared with the current state of the art.

4.1 Naïve Bayes Classification

The naïve Bayes method of classification is one that can be used for all kinds of text classification, such as Spam detection, authorship identification or sentiment analysis.

4.1.1 Generative Learning

Similar to logistic regression and neural networks, naïvee Bayes is a classification technique. Unlike logistic regression however a naïve Bayes classifier is a generative method of classification. Logistic regression on the other hand is a discriminative method of classification. In section 3.1.4 on logistic regression the example of college acceptance was used to describe logistic regression. Figure 4.1 shows the dataset that was used for this example. Students took exams, exam one and exam two, and their acceptance into college was based on the outcome of the two exams. In logistic regression the dataset is classified calculating the probability of each output given a particular input, i.e. $p(y|x)$. In generative learning the probability of an input given a particular output is calculated, i.e. $p(x|y)$. In the case of the college acceptance example, in figure 4.1, this would mean first analysing what admitted students test scores look like. Then analysing what rejected students test scores look like. When a test example then needs to be classified the naïve Bayes classifier determines if the test case looks more like an admitted students test score or rejected student test score. What is the probability of an admitted student having these test case scores and what is the probability of a rejected student having these test case scores.

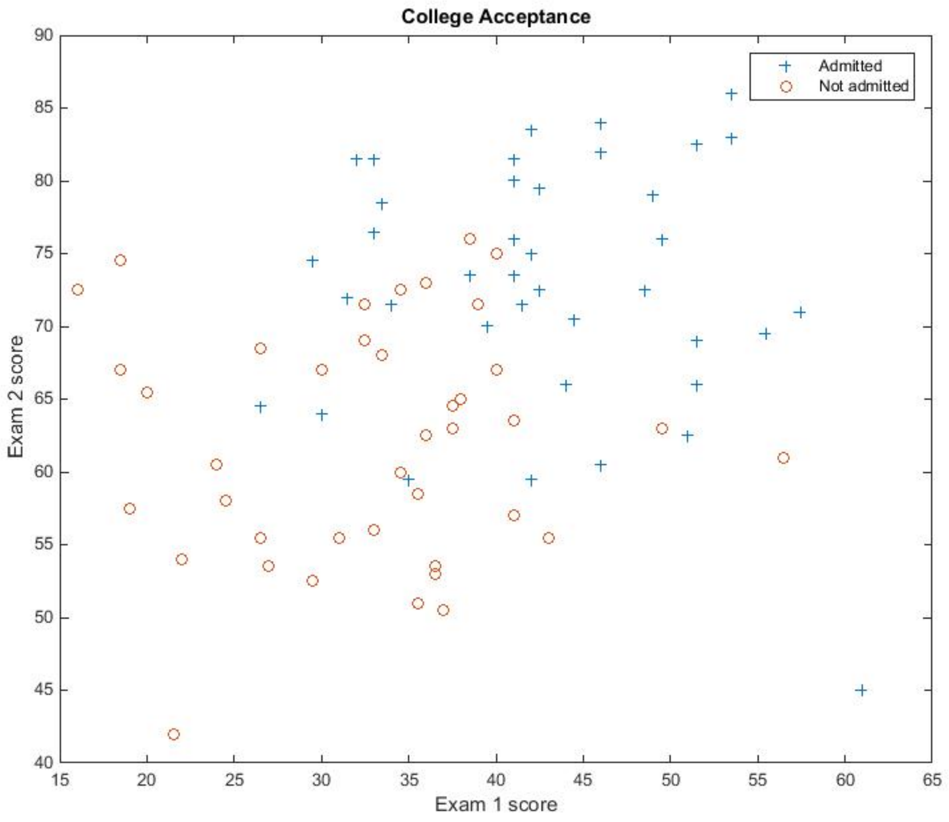


Figure 4.1: College Acceptance Example Dataset

4.1.2 Bayes Rule

Naïve Bayes classification is based on Bayes Rule. Equation 4.1 shows Bayes Rule. I implies that the probability of A given B is equal to the product of the probability of B given A by the probability of A divided by the probability of B. In the naïvee Bayes model this equation is used to calculate the probability of an output given a particular input. In this case term A in equation 4.1 is the output, y, and B is the input, x. Equation 4.2 shows Bayes rule adapted to calculate the likelihood of the output, y being one given an input, x. In order to calculate this the probability of an input when the output is one, $P(x|y=1)$, the probability of the output being one, $p(y=1)$ and the probability of that input, $p(x)$ must be calculated.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.1}$$

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x)} \tag{4.2}$$

In order to find the the probability of an output, $p(y)$, prior probability is used. Prior probability is concept of basing the probability of the classification of the next observation on past observations. For example in the college acceptance example if twice as many students were admitted than rejected then it is reasonable to assume

that the next observation is twice as likely to be classified as admitted rather an rejected. In the Bayesian analysis, this belief is known as the prior probability. Prior probabilities are based on previous experience, and often used to predict outcomes before they actually happen. [33].
Rather than finding the probability of each class given as input, as shown in equation 4.2, the most likely class can instead be found. Equation 4.3 shows an equation for finding the most likely class, maximum posteriori. *argmax* returns the argument of the maximum probability in the equation.

$$y = argmax\Big(P(x|y)P(y)\Big) \tag{4.3}$$

4.1.3 Text Classification

For text classification the inputs to the naïve Bayes method of classification are documents. These documents are processes so that they can be classified. In the naïve Bayes method the bag-of-words model, discussed in section 2.1.2 in Text Processing Chapter, is used to process the documents. The dictionary of words created by the dataset is used to determine the features in each document. Each occurrence of a dictionary word in a document is a feature of that document along with the frequency the word occurs in the dataset. Using this bag of words model features for each document are generated. Each document has different feature lengths.
These features can then be input into the naïve Bayes classifier. As with other methods of classification the features are input as x_i where i is the i^{th} features of the training example. Equation 4.4 and 4.5 show the adaption of equation 4.3 for many input features. The probability of each feature for a given output, $P(x_1, x_2, ..., x_n|y)$, is computed as the fraction of the amount of times that feature appeared for that output. With this method for calculating the probability of an input when the output given, $P(x|y)$, and prior probability for calculating the probability of a particular output, $p(y)$, the output can be calculated.

$$y = argmax\Big(P(x_1, x_2, ..., x_n|y)P(y)\Big) \tag{4.4}$$

$$y = argmax\Big(P(y)\prod_{x \in X} P(x|y)\Big) \tag{4.5}$$

4.1.4 Spam Email Example

The naïve Bayes method of classification can be used for many text classification tasks, one of these is the ability to classify emails as spam or non-spam. Emails are the inputs to the model and the features of these emails are created using a bag-of-words technique. The most frequently used words in the dataset of emails are entered into a dictionary along with the frequency they occur in the dataset. It is important to note that in order to create the dictionary the entire dataset must be used, that is all the testing and training examples. The features in each email are the words in the email that also occur in the dictionary. Figure 4.2 shows the

Matlab code for calculating the probability of spam in the dataset.[1] This is the prior probability of the next observation being spam. This is equivalent to the calculation $P(y = 1)$, where the class $y = 1$ represents spam emails.

```
% Calculate probability of spam
prob_spam = length(spam_indices) / numTrainDocs;
```

Figure 4.2: Code for Calculating the Prior Probability for the Spam Email Example

Figure 4.2 shows the Matlab code for calculating the number of features in spam and non-spam emails. This calculation is used to then calculate the probability of each feature for either spam or non-spam emails. Figure 4.3 shows the Matlab code for calculating these probabilities. The term *prob tokens spam* is the equivalent of $P(x_1, x_2, ..., x_n|y = 1)$ where $y = 1$ represents spam emails and n represents the total number of features. This is calculated from how often each word occurs in spam emails over all the words that occur in spam emails. The term *prob tokens spam* now contains a list of all words in the dictionary and the probability of that feature given the output spam. The term *prob tokens nonspam* is the opposite of *prob tokens spam*, that is equivalent of $P(x_1, x_2, ..., x_n|y = 0)$ where $y = 0$ represents non-spam emails and n represents the total number of features. This term is calculated in the same way as the *prob tokens spam* except for non spam emails.

```
% Now find the total word counts of all the spam emails and nonspam emails
spam_wc = sum(email_lengths(spam_indices));
nonspam_wc = sum(email_lengths(nonspam_indices));
```

Figure 4.3: Code for Calculating the Word Frequency for the Spam Email Example

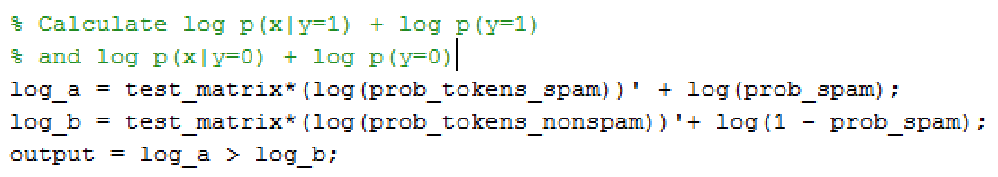
```
% Calculate the probability of the tokens in spam emails
prob_tokens_spam = (sum(train_matrix(spam_indices, :)) + 1) ./ ...
    (spam_wc + numTokens);

% Calculate the probability of the tokens in non-spam emails
prob_tokens_nonspam = (sum(train_matrix(nonspam_indices, :)) + 1)./ ...
    (nonspam_wc + numTokens);
```

Figure 4.4: Code for Calculating the Probability of Features for Spam and Non-Spam Emails for the Spam Email Example

The product of all the features in each email in the testing data can be calculated to find the classification of each email testing data. The probability of the email, in the testing set, being a spam email is then compared to the probability of the email being a non-spam email. The email is classified as the class with the higher probability. In practise the probabilities can be more easily compared using the logarithm's of the

[1]This code, and the following code in figures 4.2, 4.3, 4.4 and 4.5, is based on a tutorial given in the *Stanford Open Learning Classroom*[1]

probabilities, rather than dealing with very small numbers. Figure 4.5 shows the classification of each of the testing emails using the logarithm's of the probabilities for each classification.



```
% Calculate log p(x|y=1) + log p(y=1)
% and log p(x|y=0) + log p(y=0)
log_a = test_matrix*(log(prob_tokens_spam))' + log(prob_spam);
log_b = test_matrix*(log(prob_tokens_nonspam))'+ log(1 - prob_spam);
output = log_a > log_b;
```

Figure 4.5: Code for Calculating the Output for the Spam Email Example

4.1.5 Summary

The naïve Bayes method of classification is one that can be very useful for text classification. It was low storage requirements compared to other methods of classification and as a result can deal with larger datasets before becoming too slow. Features are also handled well by the naïve Bayes method, in that irrelevant features are cancelled out by the bag-of-words model of text processing. The naïve Bayes method can also handle features that are of equal importance. In general, the naïve Bayes method of classification is a good dependable baseline for text classificaion[12].

4.2 Stanford NLP Library for Deeply Moving: Deep Learning for Sentiment Analysis

The principle behind the Standford Deep Learning for Sentiment Analysis is that while bag-of-words text processing can be used for Sentiment Analysis it does not capture the meaning of the sentence. The Stanford implementation of Sentiment Analysis from Socher et al.[25] uses sentiment treebanks, as discussed in section 2.1.3, as the method of text processing. This means that the overall meaning of the sentence is maintained. Negation in particular is an aspect of sentence which is completely lost in a bag-of-words representation of text. The paper "Recursive Deep Model for Semantic Compositionality over a Treebank"[25] has the aim of rectifying this by utilising treebanks. This paper is built upon the knowledge from Socher's previous paper "Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions"[26] which originally used sentimental treebanks to capture the meaning of sentences.
In order to implement this type of sentiment classifier Socher et al.[25] introduces the Stanford Sentiment Treebank using the Recursive Neural Tensor Network.The Stanford Sentiment Treebank is a treebank that has been annotated with sentiment. Section 2.1.3 shows an example of a sentence that has been annotated from the Stanford Sentiment Treebank in figure 2.2. In order to create this treebank, the dataset parsed with the Stanford parser[14] in order to annotate the data with part-of-speech tags and create treebanks.

4.2.1 Dataset

The Stanford Sentiment Treebank was created using the IMDB dataset, which is a dataset of movie reviews from the IMDB website. Reviews are accompanied with ratings from one to ten for each film. Smaller segments of this dataset were extracted from each review and the larger sentence was used to label these smaller segments. Each of these sentences was classified as very negative, negative, neutral, positive, or very positive. After reviewing the dataset shorter sentences or phrases were more likely to be neutral, while longer sentences were equally spread between the five possible sentiments. This illustrates part of the difficulty with classifying shorter sentences, such as those found on twitter.

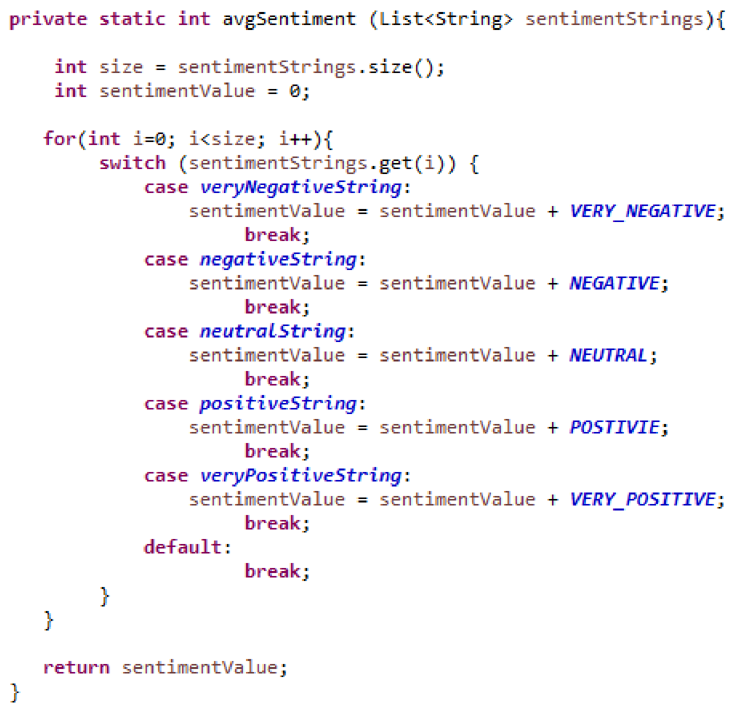
4.2.2 Text Processing

Due to the fact that the text is classifed using a sentiment treebank, the words need to tokenised into part-of-speech (POS) tags[2] before they can be used to create treebanks. The tweets that input into this model were not processed in any way. This meant that emoticons, hashtags and usernames were all used to classify the tweets. These inputs had no effect classifying the tweet, as the model was trained on movie reviews, which do not generally contain these types of text. The Stanford Sentiment Treebanks create a sentiment for each sentence rather than for the entire input. This means that sentiment of the entire sequence of sentences had to equated. Figure 4.6 shows the code used to determine the sentiment of multiple sentences. The sentiment of each sentence is added to the running sentiment of all the text. The reason that this method is used rather than for example finding the average sentiment, is that if there is one positive sentence and two neutral sentences, then the overall sentiment is still considered positive.

4.2.3 Recursive Neural Networks

Recursive Neural Networks are networks that can be used to produce parse trees by applying the same set of weights recursively. The architecture of this network is a tree structure with a neural net at each node. When a sentence is given as an input into a the recursive neural model it is first parsed in a binary tree by the Stanford parser[14]. Each word is then a leaf node on the tree.
The probability of each word given each possible output, $P(word|y)$ is calculated using softmax regression. Softmax regression, as discussed in section 3.4.4, is a method of classifing more than two possible outputs. In this case there are five possible outputs, as discussed earlier these are very negative, negative, neutral, positive, or very positive. Using this technique the probability of each word can be calculated and used to determine the sentiment for each word. Figure 4.7 shows the percentage probability for the words "with", "manditory" and "Obama" from the Stanford Sentiment Treebank for the sentence "President Obama thinks it would be great idea to have mandatory voting by citizens but is against voter i.d. What's

[2]A list of all POS tags used in figure 4.9 and 4.10 are located in the appendix figures A.4, A.5, A.6 and A.7



```java
private static int avgSentiment (List<String> sentimentStrings){

    int size = sentimentStrings.size();
    int sentimentValue = 0;

   for(int i=0; i<size; i++){
        switch (sentimentStrings.get(i)) {
            case veryNegativeString:
                sentimentValue = sentimentValue + VERY_NEGATIVE;
                     break;
            case negativeString:
                sentimentValue = sentimentValue + NEGATIVE;
                     break;
            case neutralString:
                sentimentValue = sentimentValue + NEUTRAL;
                     break;
            case positiveString:
                sentimentValue = sentimentValue + POSTIVIE;
                     break;
            case veryPositiveString:
                sentimentValue = sentimentValue + VERY_POSITIVE;
                     break;
            default:
                     break;
        }
   }

   return sentimentValue;
}
```

Figure 4.6: Code Created to Calcualte Sentiment of Multiple Sentences

wrong with this picture?" in figure 2.2. Each word is then classified based on the label with the highest probability, i.e. all the words in figure 4.7 were classified as neutral.
The probability of each output for each word then makes up a vector which is used to determine the parent vectors using compositionality functions. Once the sentiment of each word is determined, the sentiment of parent nodes, or hidden vectors are computed in a bottom-up fashion. Each parent vector is computed as a function of its children using the tanh function. The sentiment of each parent is determined using the same softmax classifier as for the children. These parent vectors are then given as inputs to determine the sentiment of the next level of the treebank. This

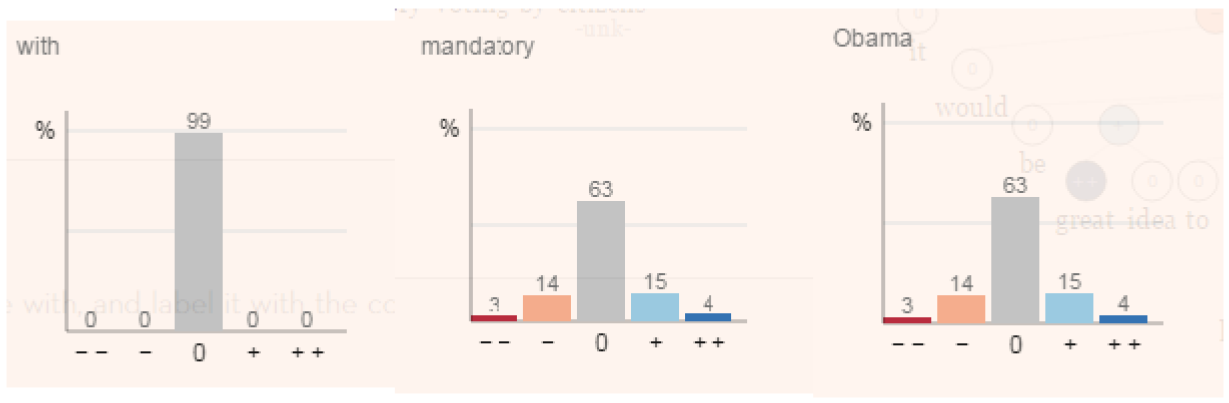
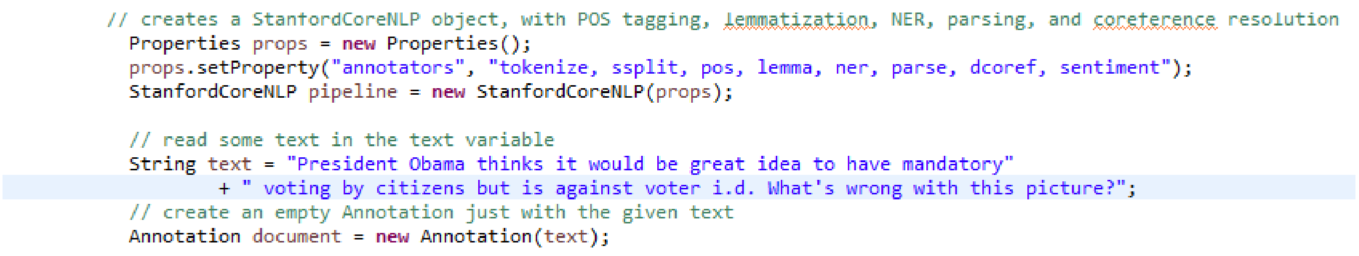


Figure 4.7: Example of Words Taken from Sentiment Treebank shown in figure 2.2 using the Stanford Implementation[25]



```
    // creates a StanfordCoreNLP object, with POS tagging, lemmatization, NER, parsing, and coreference resolution
      Properties props = new Properties();
      props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse, dcoref, sentiment");
      StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

      // read some text in the text variable
      String text = "President Obama thinks it would be great idea to have mandatory"
              + " voting by citizens but is against voter i.d. What's wrong with this picture?";
      // create an empty Annotation just with the given text
      Annotation document = new Annotation(text);
```

Figure 4.8: Code for using the Stanford NLP API for the Stanford Core NLP[15].

is continued until the sentiment of the root node is determined.

A Recursive Neural Tensor Network (RNTN) is a recursive neural network where the input vectors interact with one another more. This means that words next to one another in a treebank interect with one another more.

4.2.4 Result on IMBD Dataset

The Standford Deep Learning for Sentiment Analysis model out performed all models to date on its ability to predict the sentiment of the IMDB dataset. In the paper Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank [25] this method was compared to classification methods that ignore word order such as the naïve Bayes method and neural networks that ignore word order. This method attained 80.7% accuracy when tested.

4.2.5 Using the Stanford NLP Library API

The Stanford Core NLP Library can be downloaded and used as an API[15]. Figure 4.8 shows the code for using the Stanford API. The function *setProperty* determines which text processing annotators will be applied to the text. In this case the text will be tokenised, part-of-speech (POS) tagged and sentiment annotated. The sentence being classified is shown as the variable *text*. Figure 4.9 shows the resulting tree from this input, which is the same as the tree created in figure 2.2.

The Stanford NLP Library is not adapted for twitter, but was rather trained using the IMDB dataset. Due to this some there are some interesting aspects when the Stanford API is used for classifying tweets. Figure 4.10 shows the treebank that was created from the tweet "@BATMANNN :(i love chutney......". This tweet contains several of properties that are specific to social media such as usernames, "@BATMANN" and emoticons ":(". The tweet is also very short, making it difficult to classify. From the treebank created for this tweet it is clear that the symbols ":" and "(" are considered different features by the classifier. This means that emoticons are not considered features that determine the sentiment of the tweet. Usernames are also not used to determine the sentiment of the tweet. This tweet is however correctly classified as positive.

```
(ROOT
  (S
    (NP
      (NP (NNP President) (NNP Obama))
      (SBAR
        (S
          (VP
            (VP (VBZ thinks)
              (SBAR
                (S
                  (NP (PRP it))
                  (VP (MD would)
                    (VP (VB be)
                      (NP (JJ great) (NN idea)
                        (S
                          (VP (TO to)
                            (VP (VB have)
                              (NP (JJ mandatory) (NN voting))
                              (PP (IN by)
                                (NP (NNS citizens)))))))))))))
            (CC but)
            (VP (VBZ is)
              (PP (IN against)
                (NP (NN voter)))))))
    (VP (VBD i.d.)
      (SBAR
        (WHNP (WP What))
        (S
          (VP (VBZ 's)
            (ADJP (JJ wrong)
              (PP (IN with)
                (NP (DT this) (NN picture))))))))
    (. ?)))
```

Figure 4.9: Treebank Created by Stanford NLP API for the Stanford Core NLP[15].

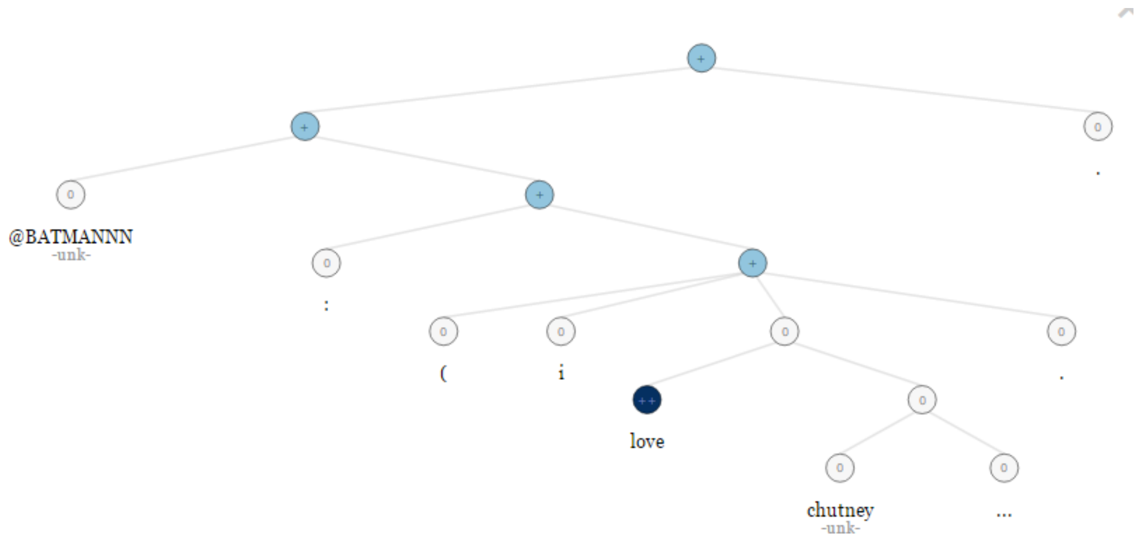


Figure 4.10: Treebank Created by Stanford NLP API for the Stanford Core NLP[15] from the Tweet "@BATMANNN :(i love chutney......".

		Actual Class					
		VN	N	NU	P	VP	
Total Population	497	177	0	139	0	181	
Predicted Class	VN	27	0	2	0	5	79.41%
	N	99	0	66	0	55	0%
	NU	35	0	63	0	49	42.86%
	P	15	0	8	0	56	0%
	VP	1	0	0	0	16	94.12%
		15.25%	0%	45.32%	0%	8.84%	21%

Table 4.1: Confusion Matrix for Stanford NLP Sentiment Analysis Implementation[25]

4.2.6 Results

Table 4.1[3] shows the results of the confusion matrix for the testing data for this method of sentiment analysis. The top of the table represents the predicted sentiment of the input tweet from the testing data and the left of the table represents the actual sentiment of the input tweet. The first column of the table shows the number of tweets of each sentiment in the testing data. There are 177 very negative tweets, 139 neutral tweets, 181 very positive tweets and zero negative or positive tweets. Of the 177 very negative tweets 27 tweet were correctly classified, 15.25% accuracy. Of the remaining tweets 99 were classified as being negative, 35 were classified as being neutral, 15 were classified as being positive, and 1 was classified as being very negative.Of the 139 neutral tweets 63 tweet were classified correctly, 45.32% accurate. Of the remaining tweets 2 were classified as very negative, 66 were classified as negative and 8 were classified as positive. Of the 181 very positive tweets, 16 were classified correctly, 8.84% accuracy. The remaining tweets 5 were classified as very negative, 55 were classified as negative, 49 were classified as neutral, and 56 were classified as positive.

Of all the tweets that were identified as being very negative 79.41% were actually very negative. Of all the tweets that were identified as being neutral 42.86% were actually neutral. Of all the tweets that were classified as very positive only 94.12% were actually very positive. Of all the tweets 21% were classified correctly. This is the overall accuracy of the sentiment analysis. It is calculated as the number of correctly classified tweets divided by the total number of tweets. Equation 4.6 shows the calculation of the accuracy for these results, as shown in table 4.1.

$$PercentageAccuracy = \frac{27 + 0 + 63 + 0 + 16}{497} * 100 = 21 \tag{4.6}$$

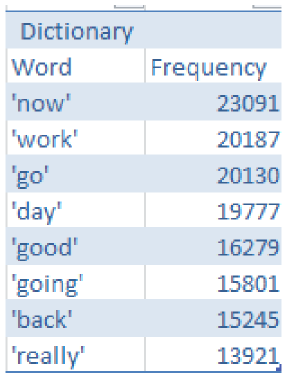
[3]The following symbols in this table have the meaning: VN - Very Negative, N - Negative, NU - Neutral, P- Positive, VP- Very Positive

4.3 Deep Learning

The techniques described in chapter 3 on Deep Learning were implemented in a neural network in order to Perform sentiment analysis. A neural network was created capable of taking tweets and determining the sentiment of the tweet, based on the language used in the tweet. The potential outputs of the neural network were very negative, negative, neutral, positive or very positive. This code used to create this neural network was written in Matlab and tutorials from the Stanford UFLDL tutorial[19] were used as a framework for this implementation.

4.3.1 Text Processing

For this implementation the text, i.e. the tweets were processed using a bag-of-words model as described in section 2.1.2. This model then created all the features that were input into the network. Using this method a dictionary of all the tweets, in the training and the testing was created along with the frequency that each word occurs in all the training and testing data. Figure 4.11 shows the first eight words in the dictionary that was created for the Stanford twitter corpus[7]. These words are words that were found to be most frequently used in the corpus, after the stop words were removed from the dataset.[4]. From this dictionary the features of each tweet are determined as the words in the tweet that are also in the dictionary.



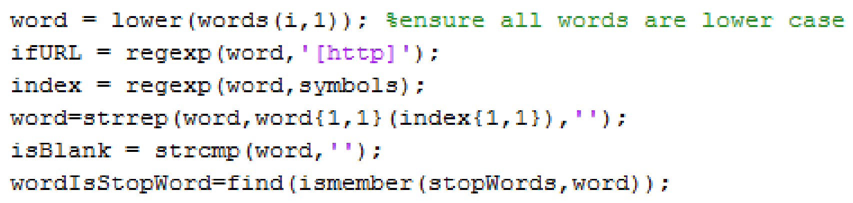
Dictionary	
Word	Frequency
'now'	23091
'work'	20187
'go'	20130
'day'	19777
'good'	16279
'going'	15801
'back'	15245
'really'	13921

Figure 4.11: Dictionary for Stanford Corpus Tweets[7]

For this project, a dictionary of 90,000 words was created. This means that each tweet could have a potential of 90,0000 features. The reason such as large dictionary was chosen was that, there are very few words in tweets. This means that there is high probability that for small dictionary a tweet may contain none of the words in the dictionary. If this is the case then this tweet has no features being input into the classify. This makes tweets very hard to classify. By choosing a large sized dictionary and creating the dictionary using the training and testing data, the model was given the best probability for having features for every tweet. Although even with this dictionary, there were cases of tweets where no word other than stop words were found.

[4]Figure A.3 in the Appendix shows the stop words that were used for this project.

In order to normalise the text in the tweets the number of words per tweet was first determined. Stop words were removed from the tweets along with URls, which were determined as words containing the prefix "HTTP:". All the words were case folded before being entered into the dictionary. Usernames symbols @ and hashtag symbols # were removed from the tweet, however the text following these symbols was maintained. The following text often contained many words concentrated into a single word such as "perfectday" as oppose to "perfect day". Due to the difficulty with word segmentation in the English language this following text was maintained as it was. Misspellings were retained in the dictionary, due to the difficulty determining the intended word. As described in section 2.2.5 the emoticons were removed from the dataset meaning that the model was not trained using emoticons, and would negate their meaning in the testing data. Figure 4.12 illustrates some of the code used for text processing. This code checks if the input word is a stop word, URL, is completely blank and removes all symbols from the text.



```
word = lower(words(i,1)); %ensure all words are lower case
ifURL = regexp(word,'[http]');
index = regexp(word,symbols);
word=strrep(word,word{1,1}(index{1,1}),'');
isBlank = strcmp(word,'');
wordIsStopWord=find(ismember(stopWords,word));
```

Figure 4.12: Code for Text Normalisation

4.3.2 Architecture

Figure 4.13 shows the neural network that was designed for this project in order to perform sentiment analysis. The network consists of four layers, the input layer, two hidden layers, and an output layer. The input layer consists of all features that are input for each sample. The first hidden layer and second hidden layers are autoencoders. The activation of the first hidden layer is controlled by the sparsity parameter. This is then used to create values for an autoencoder, in hidden layer 2. Features are created in this layer that are then optimised using backpropagation. This second hidden layer then feeds into the output layer which performs softmax regression to find the probability of each output. Using this probability from softmax regression the output is determined. The output can be very negative, negative, neutral, positive and very positive.

4.3.3 Input Layer

The input layer to the neural network is made up of all the features that are input into the network for each tweet. Each of the features for this neural network were processed using the bag-of-words model. Table 2.1 in section 2.1.2 shows an example of the features that were created from tweets one to ten. Each of these features is then an input into the neural network.

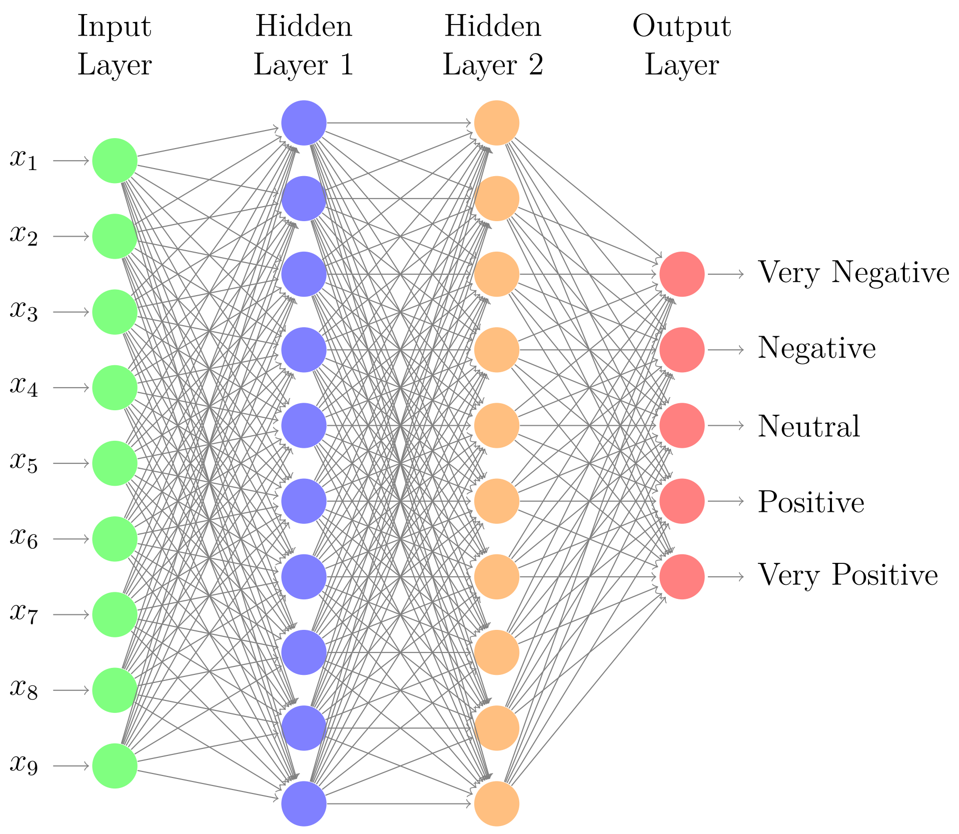


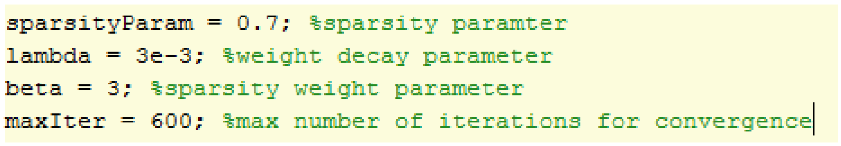
Figure 4.13: Neural Network for Sentiment Analysis

4.3.4 Hidden Layers

The first and second hidden layers create the sparse autoencoder that is used to create the features input into softmax regression. Figure 4.14 shows the parameters used by the autoencoder. The sparsity parameter, $\hat{p_j}$, lambda, λ, and beta, β, are all parameters that are used to determine the cost function. Equation 4.7 shows the cost function with the weight decay and sparsity implemented.

$$J(W,b) = [\frac{1}{m}\sum_{i=1}^{m}(\frac{1}{2}||h_{W,b}(x) - y||^2)] + \frac{\lambda}{2}\sum_{l=1}^{n_t-1}\sum_{i=1}^{s_t}\sum_{j=1}^{s_t+1}(W_{ji}^{l})^2 + \beta\sum_{j=1}^{s_2}KL(p||\hat{p_j}) \quad (4.7)$$

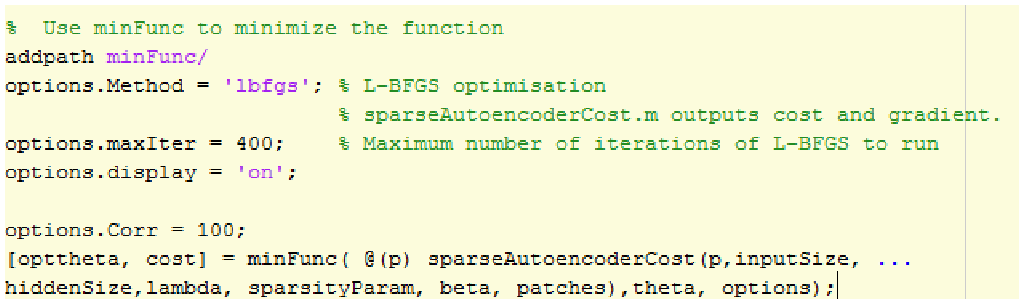
This function is optimised using gradient descent or Newtons Method, as described in section 3.1.2. Optimisation was implemented in this project using the *minFun* library[23]. This is a library that implements the L-BFGS optimisation algorithm, which is the Limited-memory BFGS optimisation algorithm. Similar to Newtons Method this method uses the inverse of the hessian matrix in order to find minimise the cost function. The difference between the L-BFGS algorithm and Newtons method is that, the L-BFGS algorithm is a quasi-newton method where the hessian is not computed at every iteration as it considered too expensive.
Figure 4.15 shows the code used in this project to optimise the parameters used in the sparse autoencoder. There are a maximum of 400 iterations of the function to check for convergence and the function is using the L-BFGS optimisation algorithm.



```
sparsityParam = 0.7; %sparsity paramter
lambda = 3e-3; %weight decay parameter
beta = 3; %sparsity weight parameter
maxIter = 600; %max number of iterations for convergence
```

Figure 4.14: Parameters for the Deep Learning Project

Each of the parameters used as inputs into the sparse autoencoder are from the parameters defined in figure 4.14.



```
%  Use minFunc to minimize the function
addpath minFunc/
options.Method = 'lbfgs'; % L-BFGS optimisation
                          % sparseAutoencoderCost.m outputs cost and gradient.
options.maxIter = 400;    % Maximum number of iterations of L-BFGS to run
options.display = 'on';

options.Corr = 100;
[opttheta, cost] = minFunc( @(p) sparseAutoencoderCost(p,inputSize, ...
hiddenSize,lambda, sparsityParam, beta, patches),theta, options);
```

Figure 4.15: Implementation of the *minFunc* Library to Perform Optimisation on the Sparse Autoencoder

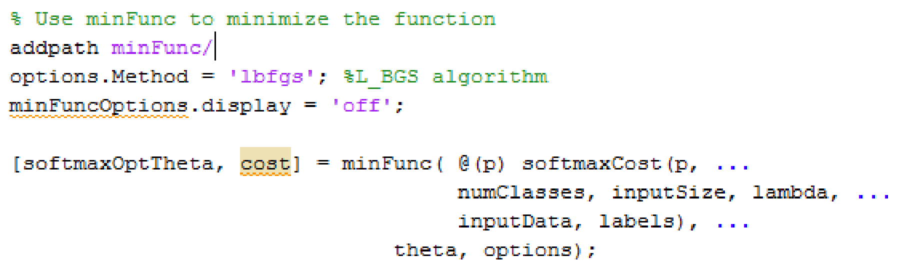
4.3.5 Output Layer

As mentioned earlier softmax regression is used to determine the output of the neural net. In order to perform softmax regression the features that were created in the sparse autoencoder first need to be extracted from the second hidden layer of the neural network. This is computed by calculating the activation of the hidden layer of the autoencoder. The activation of the hidden layer is the features that need to be output.
Once the features output from the sparse autoencoder are found softmax regression can be performed on these features. Softmax regression like the sparse autoencoder is also optimised by minimising the cost function. The cost function for softmax regression is shown in equation 3.36, in section 3.4.4 on softmax regression.[5] This cost function was also optimised using the L-BFGS algorithm, from the *minFunc* Library[23]. Figure 4.16 shows the code used to optimise the softmax regression algorithm.

4.3.6 Testing

Once the softmax regression model has been created and optimised it can be used to predict the sentiment of tweets given the features of the tweets. Figure 4.17 shows the calculation of the prediction for the output using softmax regression. The parameter *theta* is found from the softmax regression model that was optimised.

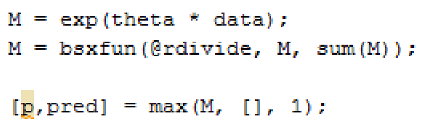
[5]This cost function was created in this project using the code shown in figure 3.28 in section 3.4.4 on softmax regression.



```
% Use minFunc to minimize the function
addpath minFunc/
options.Method = 'lbfgs'; %L_BGS algorithm
minFuncOptions.display = 'off';

[softmaxOptTheta, cost] = minFunc( @(p) softmaxCost(p, ...
                                   numClasses, inputSize, lambda, ...
                                   inputData, labels), ...
                              theta, options);
```

Figure 4.16: Implementation of the *minFunc* Library to perform Optimisation on the Softmax Regression Algorithm

This parameters is then used to find M, the matrix of the input parameters. This matrix divided by its sum gives the probability of each output. This is shown in equation 3.35 in section 3.4.4 on softmax regression. The maximum probability for each testing data is then the prediction of the model.



```
M = exp(theta * data);
M = bsxfun(@rdivide, M, sum(M));

[p,pred] = max(M, [], 1);
```

Figure 4.17: Implementation of the Softmax Regression Model to find a Prediction of the Output for all of the Testing Data

4.3.7 Results

Table 4.2[6] shows the results of the confusion matrix for the testing data for this method of sentiment analysis. The top of the table represents the predicted sentiment of the input tweet from the testing data and the left of the table represents the actual sentiment of the input tweet. The first column of the table shows the number of tweets of each sentiment in the testing data. There are 177 very negative tweets, 139 neutral tweets, 181 very positive tweets and zero negative or positive tweets. Of the 177 very negative tweets 102 tweet were correctly classified, 57.63% accuracy. The remaining 75 tweets were classified as very positive. Of the 139 neutral tweets zero tweets were classified correctly, 0% accuracy. 80 were classified as being very negative and 59 were classified as very positive. Of the 181 very positive tweets, 95 were classified correctly, 52.49% accuracy. The remaining 86 tweets were classified as very negative. Of all the tweets that were identified as being very negative 38.06% were actually very negative. Of all the tweets that were classified as very positive only 41.48% were actually very positive. Of all the tweets 40% were classified correctly. This is the overall accuracy of the sentiment analysis. It is calculated as the number of correctly classified tweets divided by the total number of tweets. Equation 4.8 shows the calculation of the accuracy for these results, as shown in table 4.2.

[6]The following symbols in this table have the meaning: VN - Very Negative, N - Negative, NU - Neutral, P- Positive, VP- Very Positive

		Actual Class					
		VN	N	NU	P	VP	
Total Population	497	177	0	139	0	181	
Predicted Class	VN	102	0	80	0	86	38.06%
	N	0	0	0	0	0	0%
	NU	0	0	0	0	0	0%
	P	0	0	0	0	0	0%
	VP	75	0	59	0	95	41.48%
		57.63%	0%	0%	0%	52.49%	40%

Table 4.2: Confusion Matrix for Deep Learning Implementation

$$PercentageAccuracy = \frac{102+0+0+0+95}{497} * 100 = 40 \tag{4.8}$$

5 Conclusion

5.1 Result Discussion

For the purpose of classifying tweets, the neural network that was trained on tweets for this project performed better than the Stanford model. This shows the advantage of training the classifier on the kind of text that it will be classifying. However this may also illustrate some parts of the model that are open to interpretation. Sentiment is a subjective property and thus can be defined differently by alternative models. In this case the labels very negative and very positive are quite subjective. The Stanford model[25] rarely defines sentiments as very positive or very negative. However the dataset used in this experience defines all tweets with positive emoticons very positive and all tweets with negative tweets as very negative. This definition has a huge effect on the overall performance of the model. From the confusion matrix from the Stanford model it is clear that many tweets that were labelled as very negative were classified by the Stanford model as negative. The same is true for the tweets that were labelled as very positive. The neural network on the other hand, only classifies tweets as very positive or very negative as these were the only labels in the training data. While this improves the accuracy of the model for this dataset, it is clear that the model is not very accurately classifying the tweets. For very negative, neutral and very positive tweets almost half of the tweets are split in all cases between very negative and very positive classes. The model also seems to favour classifying tweets as very negative.
Table 5.2 and 5.1[1] show the confusion matrix from the neural network implementation and the Stanford method for only three classes, positive, negative and neutral. In order to create these tables all the very negative labels were counted as negative

[1]The following symbols in this table have the meaning: N - Negative, NU - Neutral, P- Positive

		Actual Class			
		N	NU	P	
Total Population	497	177	139	181	
Predicted Class	N	126	68	60	49.61%
	NU	35	63	49	42.86%
	P	16	8	72	75%
		71.19%	45.32%	39.78%	53%

Table 5.1: Confusion Matrix for Stanford NLP Sentiment Analysis Implementation[25] for Three Classes

		Actual Class			
		N	NU	P	
Total Population	497	177	139	181	
Predicted Class	N	102	80	86	38.06%
	NU	0	0	0	0%
	P	75	59	95	41.48%
		57.63%	0%	52.49%	40%

Table 5.2: Confusion Matrix for Deep Learning Implementation for Three Classes

and all the very positive labels were counted as positive. The result of the neural network does not change as the only outputs were very negative or very positive. The Stanford model however improves its accuracy greatly, from 21% to 53%. This illustrates the effect of the ambiguous very negative and very positive labels.
Therefore, understanding the tweet in a hierarchical structure in which the meaning of a sentence is maintained improves the ability to classify it. In general, however, both the Stanford model and the neural network model are reasonably ineffective at classifying tweets, even when the labels are reduced to less ambiguous labels. The ability to perform sentiment analysis on a tweet is still not dependable. Determining public opinion using twitter is not perfected to a point where it could be relied upon.

5.2 Future Work

5.2.1 Dataset

Increase the size of the dataset by including tweets taken from the Twitter API. These tweets could be hand labelled. The advantage of including these tweets could be that if the tweets contain emoticons they could be maintained in the dataset, which would allow the model to trained to account for the presence of emoticons. Another dataset that has been labelled using other methods could also be used. Currently there are few other labelled dataset available, however this is likely to change in the future.

5.2.2 Text Processing

There are many additional methods that could be applied to this project in order to improve the performance of the neural network created. Words that have spaces in them were not considered in this model, such as "San Francisco". By creating a corpus of words with spaces a comparison could be made against the input word and this corpus. This would ensure that words with spaces could be characterised correctly in the dictionary created for this model.
A corpus of words with the same word form such as "I'm" and "I am" could be created so that words were either recognised as stop words, as in this case, or entered into the dictionary as a single input. On social media, users can intentionally miss spell words that have the same word form as other words for emphasis. A corpus

containing commonly intentionally miss spelt words on social media would negate this problem.
Due to the fact that all words were case folded certain words were categorised as being the same, while they in fact had different word forms, for example "US" (United States) as appose to "us". By treating words with capitalised letters differently depending on their location in the sentence, the accuracy of the model could be improved. Words with capitalised letters in the middle of a sentence would be entered as they are, without case folding. This would retain the meaning of abbreviations and company names.
These changes require large data bases in order to contain all variations of words that occur in the twitter dataset. This increases the computational time required in order to create dictionaries, but would improve the accuracy of the model.

5.2.3 Further Assessment of the Model

In order to further assess the performance of this model it could be compared against the naïve Bayes method for sentiment analysis. As this method is much simpler to design than the Stanford implementation of Sentiment Analysis from Socher et al.[25] it could be trained using the tweets in the same way that the model implemented in this project was. For this reason the naïve Bayes model would be a good model of comparison for this implementation despite the fact that it is not the most state of the art model in this area.

Bibliography

[1] Bengio, S., Dean, T. and Ng, A. *Machine Learning.* Open Classroom, 2014. `http://openclassroom.stanford.edu`

[2] Bengio, Y. *Learning Deep Architectures for AI.* Dept. IRO, Universite de Montreal, C.P. 6128, Montreal, Qc, H3C 3J7, Canada, 2009. `http://www.iro.umontreal.ca/bengioy`

[3] Bollen, J., Mao, H., and Zeng, X., *Twitter mood predicts the stock market.* J. Comput. Science, 2(1):1–8, 2011.

[4] Bonthous, J., *Twitrratr Archive.* Twitrratr, 2010.

[5] Duggan, M., Ellison, N., Lampe, C., Lenhart, A., and Madden, M., *Demographics of Key Social Networking Platforms.* Pew Research Centre, 2014. `http://www.pewinternet.org/2015/01/09/demographics-of-key-social-networking-platforms-2/`

[6] Ghiassi, M., Skinner, J., and Zimbra, D., *Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network.* Oklahoma State University, 2014.

[7] Go, A., Bhayani, R., and Huang L., *Sentiment140.* Stanford, 2009. `http://help.sentiment140.com/for-students/`

[8] Go, A., Bhayani, R., and Huang, L., *Twitter Sentiment Classification using Distant Supervision.* Stanford University, 2009. `http://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf`

[9] Haykin, S., *Neural Networks.* A Comprehensive Foundation, Second Addition, McMaster University, Hamilton, Ontario, Canada pages 173–178, 1999.

[10] Hernandez, D. *Meet the Man Google Hired to Make AI a Reality.* Wired, 2014. `http://www.wired.com/2014/01/geoffrey-hinton-deep-learning/`

[11] Hinton G. E., *Movies of the neural network generating and recognizing digits.* Basic papers on deep learning, 2006. `http://www.cs.toronto.edu/~hinton/`

[12] Jurafsky D., *Naïve Bayes and Text Classification.* CS 124: From Languages to Information, Stanford, 2015.

[13] Jurafsky, D., and Manning C., *Video Lectures*. Natural Language Processing, Coursera, Stanford, 2015. `https://class.coursera.org/nlp/lecture/preview`

[14] Klein, D., and Manning, C., *Accurate Unlexicalized Parsing*. Computer Science Department, Stanford, 2003.

[15] Klein, D., and Manning, C., *Stanford CoreNLP version 3.5.2.*. The Stanford Natural Language Processing Group., Stanford, 2015.

[16] Mitchell, A., and Hitlin, P., *Twitter Reaction to Events Often at Odds with Overall Public Opinion*. Pew Research Center, 2013. `http://www.pewresearch.org/2013/03/04/twitter-reaction-to-events-often-at-odds-with-overall-public-opinion/`

[17] Munro, R., *Why is sentiment analysis hard?*. Digital Marketing, Natural Language Processing, IDIBON, 2013. `http://idibon.com/why-is-sentiment-analysis-hard/`

[18] Nadkarni, P., Ohno-Machado, L., and Chapman, W., *Natural language processing: an introduction*. Journal of the American Medical Informatics Association, NCBI, 2011. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3168328/`

[19] Ng, A., Ngiam, J., Foo, C., Mai, Y., Suen, C., *UFLDL Tutorial*, Unsupervised Feature Learning and Deep Learning, Stanford, 2013. `http://ufldl.stanford.edu/wiki/`

[20] Nielsen, .M *Neural Networks and Deep Learning*. Free online book, 2014. `http://neuralnetworksanddeeplearning.com/`

[21] Pang, B., Lee, L., and Vaithyanathan, S., *Thumbs up? Sentiment classification using machine learning techniques*. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 79–86, 2002.

[22] Ríos, M., *The geography of Tweets*. Twitter, 2009. `https://blog.twitter.com/2013/the-geography-of-tweets`

[23] Schmidt, M., *minFunc Library*. Unconstrained differentiable multivariate Optimisation in Matlab, 2005. `http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html`

[24] Shure, L. *Analysing Twitter with Matlab*. Matlab Central Blogs, 2014. `http://blogs.mathworks.com/loren/2014/06/04/analyzing-twitter-with-matlab/`

[25] Socher, R., Perelygin, A., Wu.J, Chuang, J., Manning, C., Ng, A., and Potts, C., *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*. Stanford, 2014. `http://nlp.stanford.edu/~socherr`

[26] Socher, R., Pennington, J., Huang, E., Ng, A., Manning, C., *Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions.* Computer Science Department, Stanford University, 2011. `http://www.socher.org/uploads/Main/SocherPenningtonHuangNgManning_EMNLP2011.pdf`

[27] Srivastava, T., *How to avoid Over-fitting using Regularization?.* Learn Everything About Analytics, Analytics Vidhya. `http://www.analyticsvidhya.com/blog/2015/02/avoid-over-fitting-regularization/`

[28] Tumasjan, A., Sprenger, T., Sandner, P., and Welpe, I., *Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment* . Technische Universität München, 2010.

[29] Vryniotis, V., *Tuning the learning rate in Gradient Descent.* Machine Learning Blog and Software Development News, DatumBox. `http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/`

[30] Yu, S. and Kakn, S. *A Survey of Prediction Using Social Media.* Oklahoma State University, 2014.

[31] Blogger "Rocapal" *Polynomial Regression.* Rocapal or Lapacor, Blog. `http://blog.rocapal.org/?page_id=2`

[32] *Gradient Descent in Linear Regression.* Questions, Stack Overflow. `http://stackoverflow.com/questions/21064030/gradient-descent-in-linear-regression`

[33] *Naive Bayes Classifier.* Statisitcs Textbook, DELL Software.

[34] *Penn Treebank II tag set.* Penn Treebank, Computational Linguistics and Psycholinguistics Research Centre. `http://www.clips.ua.ac.be/pages/mbsp-tags`

[35] *The Search API.* Developers, Documentation, REST APIs, Twitter, Inc. 2015. `https://dev.twitter.com/rest/public/search`

[36] *Twitter Usage Statistics.* Internet Live Statistics, 2013.

A
Appendix 1

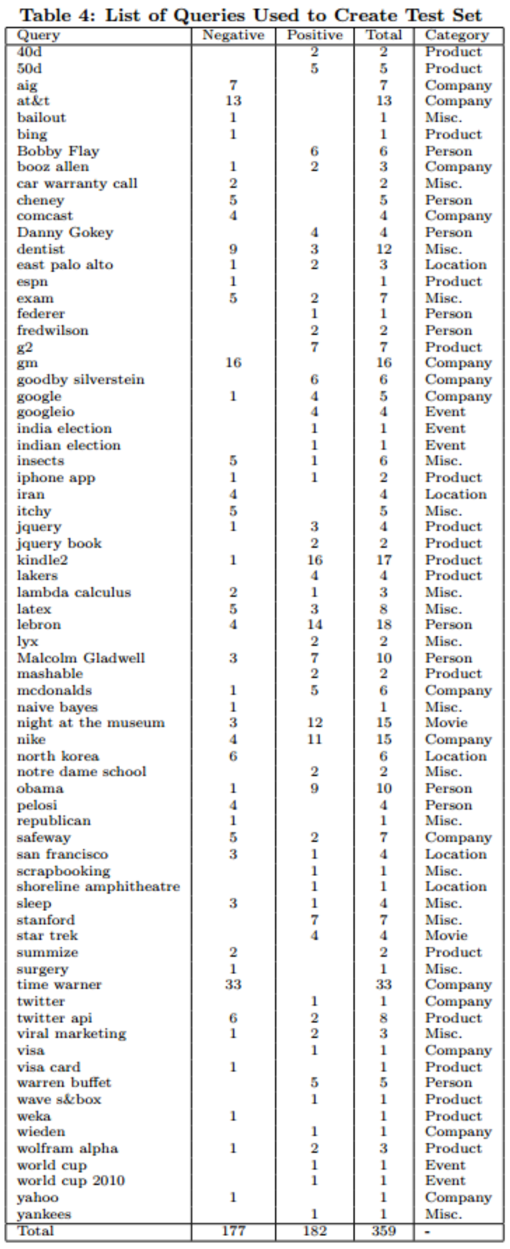


Table 4: List of Queries Used to Create Test Set

Query	Negative	Positive	Total	Category
40d		2	2	Product
50d		5	5	Product
aig	7		7	Company
at&t	13		13	Company
bailout	1		1	Misc.
bing	1		1	Product
Bobby Flay		6	6	Person
booz allen	1	2	3	Company
car warranty call	2		2	Misc.
cheney	5		5	Person
comcast	4		4	Company
Danny Gokey		4	4	Person
dentist	9	3	12	Misc.
east palo alto	1	2	3	Location
espn	1		1	Product
exam	5	2	7	Misc.
federer		1	1	Person
fredwilson		2	2	Person
g2		7	7	Product
gm	16		16	Company
goodby silverstein		6	6	Company
google	1	4	5	Company
googleio		4	4	Event
india election		1	1	Event
indian election		1	1	Event
insects	5	1	6	Misc.
iphone app	1	1	2	Product
iran	4		4	Location
itchy	5		5	Misc.
jquery	1	3	4	Product
jquery book		2	2	Product
kindle2	1	16	17	Product
lakers		4	4	Product
lambda calculus	2	1	3	Misc.
latex	5	3	8	Misc.
lebron	4	14	18	Person
lyx		2	2	Misc.
Malcolm Gladwell	3	7	10	Person
mashable		2	2	Product
mcdonalds	1	5	6	Company
naive bayes	1		1	Misc.
night at the museum	3	12	15	Movie
nike	4	11	15	Company
north korea	6		6	Location
notre dame school		2	2	Misc.
obama	1	9	10	Person
pelosi	4		4	Person
republican	1		1	Misc.
safeway	5	2	7	Company
san francisco	3	1	4	Location
scrapbooking		1	1	Misc.
shoreline amphitheatre		1	1	Location
sleep	3	1	4	Misc.
stanford		7	7	Misc.
star trek		4	4	Movie
summize	2		2	Product
surgery	1		1	Misc.
time warner	33		33	Company
twitter		1	1	Company
twitter api	6	2	8	Product
viral marketing	1	2	3	Misc.
visa		1	1	Company
visa card	1		1	Product
warren buffet		5	5	Person
wave s&box		1	1	Product
weka	1		1	Product
wieden		1	1	Company
wolfram alpha	1	2	3	Product
world cup		1	1	Event
world cup 2010		1	1	Event
yahoo	1		1	Company
yankees		1	1	Misc.
Total	177	182	359	-

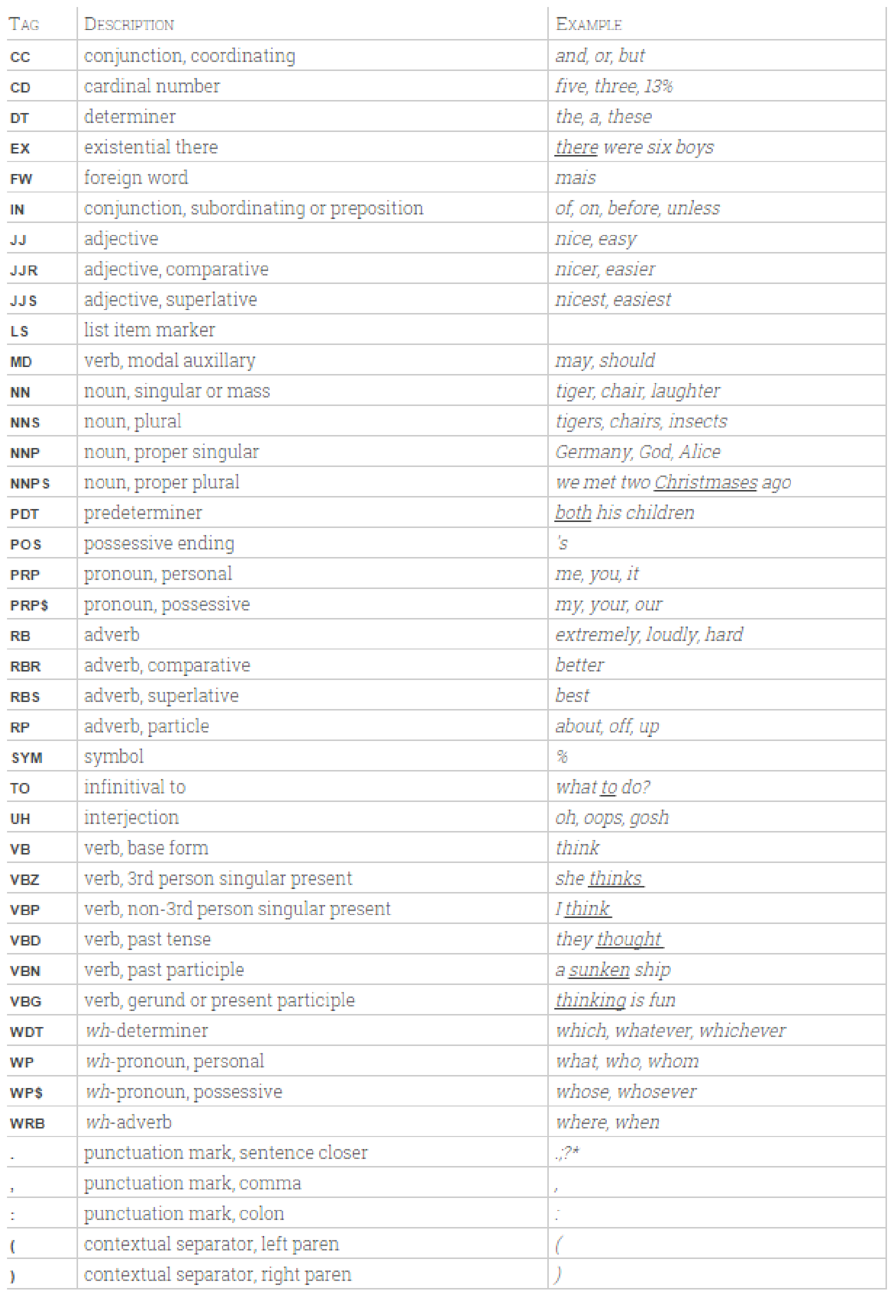
Table A.1: Queries used to create dataset[8]

Label	ID	Date and Time	Topic	User	Tweet
4	3	Mon May 11 2009	kindle2	tpryan	@stellargirl I loooooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.
4	4	Mon May 11 2009	kindle2	vcu451	Reading my kindle2... Love it... Lee childs is good read.
4	5	Mon May 11 2009	kindle2	chadfu	Ok, first assesment of the #kindle2 ...it fucking rocks!!!
4	6	Mon May 11 2009	kindle2	SIX15	@kenburbary You'll love your Kindle2. I've had mine for a few months and never looked back. The new big one is huge! No need for remorse! :)
4	7	Mon May 11 2009	kindle2	yamarama	@mikefish Fair enough. But i have the Kindle2 and I think it's perfect :)
4	8	Mon May 11 2009	kindle2	GeorgeVHulme	@richardebaker no. it is too big. I'm quite happy with the Kindle2.
0	9	Mon May 11 2009	aig	Seth937	Fuck this economy. I hate aig and their non loan given asses.
4	10	Mon May 11 2009	jquery	dcostalis	Jquery is my new best friend.
4	11	Mon May 11 2009	twitter	PJ King	Loves twitter
4	12	Mon May 11 2009	obama	mandanicole	how can you not love Obama? he makes jokes about himself.

Table A.2: 10 tweets taken for testing corpus for Stanford twitter corpus [7]

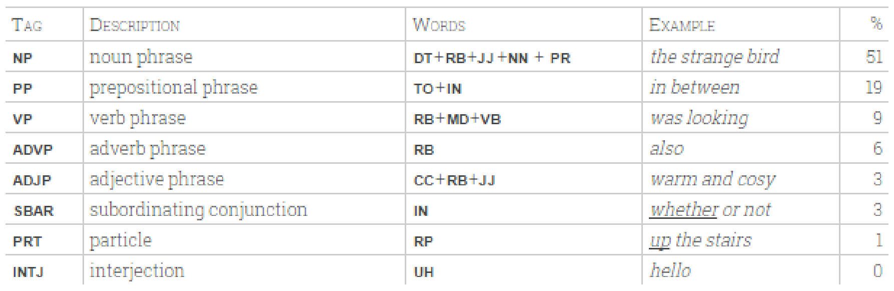
Stop Words			
a	ever	might	them
able	every	most	then
about	for	must	there
across	from	my	these
after	get	neither	they
all	got	no	this
almost	had	nor	tis
also	has	not	to
am	have	of	too
amoung	he	off	twas
an	her	often	us
and	hers	on	wants
any	him	only	was
are	his	or	we
as	how	other	were
at	however	our	what
be	i	own	when
because	if	rather	where
been	in	said	which
but	into	say	while
by	is	says	who
can	it	she	whom
cannot	its	should	will
could	just	since	with
dear	least	so	would
did	let	some	yet
do	like	than	you
does	likely	that	your
either	may	the	
else	me	their	

Table A.3: Stop words used during text processing



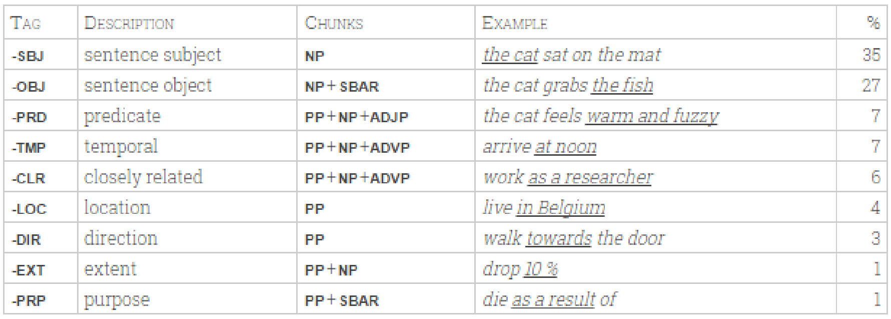
Tag	Description	Example
CC	conjunction, coordinating	*and, or, but*
CD	cardinal number	*five, three, 13%*
DT	determiner	*the, a, these*
EX	existential there	*there were six boys*
FW	foreign word	*mais*
IN	conjunction, subordinating or preposition	*of, on, before, unless*
JJ	adjective	*nice, easy*
JJR	adjective, comparative	*nicer, easier*
JJS	adjective, superlative	*nicest, easiest*
LS	list item marker	
MD	verb, modal auxillary	*may, should*
NN	noun, singular or mass	*tiger, chair, laughter*
NNS	noun, plural	*tigers, chairs, insects*
NNP	noun, proper singular	*Germany, God, Alice*
NNPS	noun, proper plural	*we met two Christmases ago*
PDT	predeterminer	*both his children*
POS	possessive ending	*'s*
PRP	pronoun, personal	*me, you, it*
PRP$	pronoun, possessive	*my, your, our*
RB	adverb	*extremely, loudly, hard*
RBR	adverb, comparative	*better*
RBS	adverb, superlative	*best*
RP	adverb, particle	*about, off, up*
SYM	symbol	*%*
TO	infinitival to	*what to do?*
UH	interjection	*oh, oops, gosh*
VB	verb, base form	*think*
VBZ	verb, 3rd person singular present	*she thinks*
VBP	verb, non-3rd person singular present	*I think*
VBD	verb, past tense	*they thought*
VBN	verb, past participle	*a sunken ship*
VBG	verb, gerund or present participle	*thinking is fun*
WDT	*wh*-determiner	*which, whatever, whichever*
WP	*wh*-pronoun, personal	*what, who, whom*
WP$	*wh*-pronoun, possessive	*whose, whosever*
WRB	*wh*-adverb	*where, when*
.	punctuation mark, sentence closer	*.;?**
,	punctuation mark, comma	*,*
:	punctuation mark, colon	*:*
(	contextual separator, left paren	*(*
)	contextual separator, right paren	*)*

Table A.4: Part of Speech Tags for Penn Treebank [34]



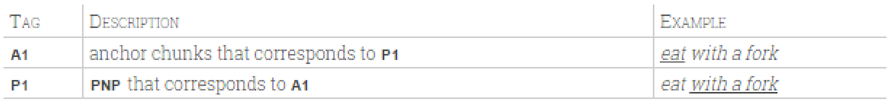
Tag	Description	Words	Example	%
NP	noun phrase	DT+RB+JJ+NN + PR	*the strange bird*	51
PP	prepositional phrase	TO+IN	*in between*	19
VP	verb phrase	RB+MD+VB	*was looking*	9
ADVP	adverb phrase	RB	*also*	6
ADJP	adjective phrase	CC+RB+JJ	*warm and cosy*	3
SBAR	subordinating conjunction	IN	*whether or not*	3
PRT	particle	RP	*up the stairs*	1
INTJ	interjection	UH	*hello*	0

Table A.5: Chunk Tags, groups of words that belong together, for Penn Treebank [34]



Tag	Description	Chunks	Example	%
-SBJ	sentence subject	NP	*the cat sat on the mat*	35
-OBJ	sentence object	NP+SBAR	*the cat grabs the fish*	27
-PRD	predicate	PP+NP+ADJP	*the cat feels warm and fuzzy*	7
-TMP	temporal	PP+NP+ADVP	*arrive at noon*	7
-CLR	closely related	PP+NP+ADVP	*work as a researcher*	6
-LOC	location	PP	*live in Belgium*	4
-DIR	direction	PP	*walk towards the door*	3
-EXT	extent	PP+NP	*drop 10 %*	1
-PRP	purpose	PP+SBAR	*die as a result of*	1

Table A.6: Relation Tags, groups of chunks that belong together, for Penn Treebank [34]



Tag	Description	Example
A1	anchor chunks that corresponds to P1	*eat with a fork*
P1	PNP that corresponds to A1	*eat with a fork*

Table A.7: Anchor Tags, propositional noun phrases attached together, for Penn Treebank [34]