Computer Science and Artificial Intelligence

B221319

# Sentiment Analysis on Twitter

*by*

*Ravi-Shyam Patel*

*Supervisor: Dr A. Soltoggio*

May/June 2016

# Sentiment Analysis on Twitter

by Ravi-Shyam Patel

Student ID: B221319

Loughborough University

15COC251: Computer Science and Artificial Intelligence Project

Supervisor: Dr Andrea Soltoggio

SUBMITTED 26th APRIL 2016

**Abstract**

The growing use of social media has led researchers and businesses to exploit the constant stream of data it has produced. Xeidos LTD have a social engagement product that analyses sentiment of tweets on Twitter using a classifier created on the Azure Machine Learning platform. However, the implementation of features and machine learning classifiers is not an area in which the company has expertise. To validate and gain trust in Xeidos' use of the Azure classifier, this work develops a classifier using the same data, features and machine learning classifier, a support vector machine. The accuracy achieved by the developed classifier is within 0.2% of the Azure classifier, which validates its use. The second objective of this work is to explore additional features that could be implemented and to evaluate their impact on the accuracy of the developed classifier comparing it with a state-of-the-art classifier. Overall, the additional features increased the accuracy of the developed classifier by 4.30%, a considerably useful increase.

## Acknowledgements

I would like to thank Paul Ince and Simon Jenkinson from Xeidos LTD, for proposing this project and for their ideas and support. I would also like to thank Dr Andrea Soltoggio for keeping me pointed in the right direction throughout this project.

# Contents

# 1 Introduction

Others' opinions have always been vital information for society during any decision-making process (Pang and Lee (2008)). Liu (2012) explains that the choices made by individuals and organisations are conditioned upon how others see and evaluate the world. This reliance on opinions is the main reason for the development of the research fields of opinion mining and sentiment analysis. While sentiment analysis can be found in work by Carbonell (1979), the use of statistical analysis methods to mine sentiment is more recognisable in the work of Manning and Schütze (1999). They laid the foundation for how machine learning methods could be applied to the problem of sentiment analysis. After the turn of the century, the web 2.0 was born, opening the door for blogging and social media sites (O'Reilly (2005)). The earliest recognised machine learning techniques for many researchers were developed around this time by Pang et al. (2002). Their use of support vector machines (SVMs), Naïve Bayes, and Maximum Entropy machine learning techniques to mine opinions in on-line movie reviews, were considered the baseline for future research. However, there was no recognised application of machine learning for sentiment analysis in social media until Go et al. (2009) applied the same methods used by Pang et al. (2002) to the social media microblogging platform, Twitter. Since then, there has been an increase in research on sentiment analysis of social media because researchers have realised social media offers a constant stream of highly opinionated data.

## 1.1 Aims and Objectives

Xeidos Ltd. has developed a social media sales engagement product, Thryyve, that allows users to search Twitter for people who are a good match to specific areas of interest, for example, people who might purchase their products or services. The current algorithm used in Thryyve is based on manual as-

sumptions of their expertise in social media. Part of their algorithm uses sentiment analysis to gauge a customer's sentiment towards a certain product. The current implementation of sentiment analysis uses a freely available experiment created on Microsoft's Azure Machine Learning platform (hereinafter called the Azure classifier). This implementation is further discussed in section 1.3. The inner workings of this implementation are not thoroughly explained, and for those who do not have specific expertise in machine learning or natural language processing, it is difficult to trust the results from Azure implementation.
This work aims to:

1. validate the current implementation of sentiment analysis used by Xeidos in Thryyve; and

2. explore features that can improve the accuracy of a sentiment analysis classifier.

The first objective will be achieved by building a similar sentiment analysis tool as the current implementation and testing it with the same data set. The accuracy of the developed classifier will then be compared to that of the Azure classifier. If similar results are produced, this will validate the results produced by the Azure classifier and confirm that its implementation can be trusted.
The second objective, which looks at possible improvements such as adding features to the implementation, will be explored by adding them to the developed classifier. The results from the developed classifier will then be compared to those of a state-of-the-art classifier that uses the implemented features. The comparisons of results will reveal whether the features in the developed classifier have been implemented correctly.
The structure of the paper will be as follows. There will be a brief review of relevant previous work in which the ideas and methods will be evaluated in

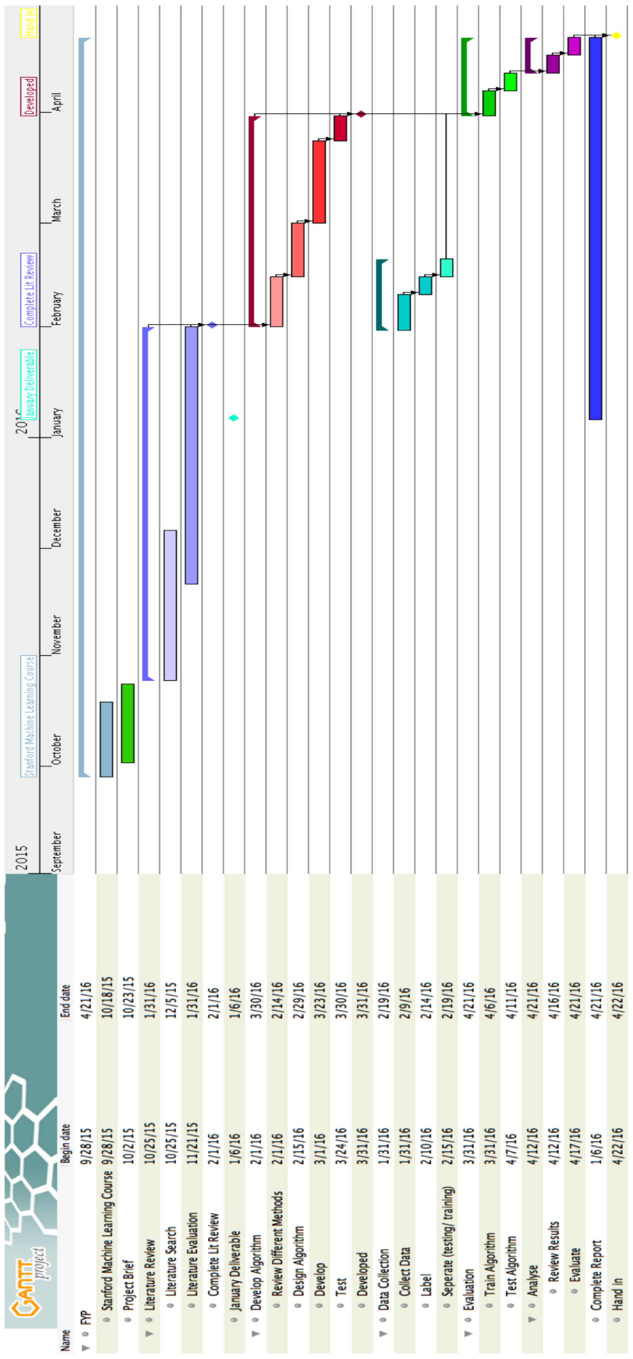
terms of their applicability to a small start-up, Xeidos, that may not have relevant expertise in machine learning. The use of linguistic patterns will also be researched and implemented because preliminary research suggests that combining the two techniques yields higher accuracy. Different methods of feature selection to train the chosen machine learning algorithm will also be discussed. The chosen machine learning algorithm and linguistic rules will be implemented in the Python programming language. The implementation will be described in detail.

The work in this project will benefit Xeidos by validating the current implementation of sentiment analysis in its product. The sentiment analysis tool produced will also be made available as a starting place for future work in this field.

## 1.2 Work Plan

Following a work plan for this project was vital for managing time as best as possible. Figure 1 shows a Gantt chart built using Gantt Project that enabled visualisation of the work plan.

Before starting the project, the basics of Machine Learning and Natural Language Processing were learned through free courses. Having the basic grasp of these topics helped the researcher quickly identify and understand relevant literature. The information reviewed gave some direction to the project in terms of the choice of machine learning classifier, choice of features and methods of extracting those features. With these indications, and after data had been collected, the development of the classifier in Python began. Collecting data before developing the classifier was imperative to allow testing of the classifier throughout its development. Once the classifier was developed, it was tested. The report was written alongside the development process.



| Name | Begin date | End date |
|---|---|---|
| FYP | 9/28/15 | 4/21/16 |
| Stanford Machine Learning Course | 9/28/15 | 10/18/15 |
| Project Brief | 10/2/15 | 10/23/15 |
| Literature Review | 10/25/15 | 1/31/16 |
| Literature Search | 10/25/15 | 12/5/15 |
| Literature Evaluation | 11/21/15 | 1/31/16 |
| Complete Lit Review | 2/1/16 | 2/1/16 |
| January Deliverable | 1/6/16 | 1/6/16 |
| Develop Algorithm | 2/1/16 | 3/30/16 |
| Review Different Methods | 2/1/16 | 2/14/16 |
| Design Algorithm | 2/15/16 | 2/29/16 |
| Develop | 3/1/16 | 3/23/16 |
| Test | 3/24/16 | 3/30/16 |
| Developed | 3/31/16 | 3/31/16 |
| Data Collection | 1/31/16 | 2/19/16 |
| Collect Data | 1/31/16 | 2/9/16 |
| Label | 2/10/16 | 2/14/16 |
| Seperate (testing/ training) | 2/15/16 | 2/19/16 |
| Evaluation | 3/31/16 | 4/21/16 |
| Train Algorithm | 3/31/16 | 4/6/16 |
| Test Algorithm | 4/7/16 | 4/11/16 |
| Analyse | 4/12/16 | 4/21/16 |
| Review Results | 4/12/16 | 4/16/16 |
| Evaluate | 4/17/16 | 4/21/16 |
| Complete Report | 1/6/16 | 4/21/16 |
| Hand in | 4/22/16 | 4/22/16 |

Figure 1: Gantt Chart showing the followed work plan

## 1.3 Related Work

### 1.3.1 Opinion and Sentiment Definition

In this work, an opinion is defined as a regular opinion as described by Liu (2012). This is a review of a generic toaster posted by Anon.

> "(1) I bought this toaster a few months ago to replace my broken one. (2) I like that it was reasonably priced. (3) The different heat settings are good, so I can change the setting depending on what I'm toasting. (4) One thing I should mention is that I don't think the lead is very long."

The review has been divided into numbered sections and consists of a target, the toaster, and several opinions about that toaster (2),(3) and (4). (2) and (3) express positive opinions, whereas (4) expresses a negative opinion. From this we can deduce that an opinion is a tuple (target, sentiment) in which the sentiment can be of three polarities—positive, negative or neutral—and the target is either the toaster or a feature of the toaster. For example, opinion (2) can be decomposed to (price, positive). At this point, the definition opinion used in this work will differ from that used by Liu (2012) because the person who gives the opinion and the time at which the opinion is given are outside the scope of this work.

### 1.3.2 Brief History

As discussed in the introduction, the turn of the century has seen sentiment analysis become a more researched area of computer science (Pang and Lee (2008); Liu (2012)). One motivation for this increased interest is the ability of more people to express their opinions online through mediums such as blogs, social media and online articles. As discussed by Hu and Liu (2004), some method of summarising these opinions on the Web needed to be developed.

Early work carried out by Pang et al. (2002) that dealt with long pieces of online texts such as movie reviews laid the foundation for many of the techniques used today. However, it was not until Go et al. (2009) applied the sentiment analysis techniques used by Pang et al. (2002) to Twitter that interest grew in applying sentiment analysis on social media. Their pioneering use of machine learning to predict user opinion on Twitter led the way for many more researchers to further enhance the techniques.

### 1.3.3 Twitter as a Platform for Sentiment Analysis

Twitter is a microblogging social media platform on which, among other things, users can tweet their opinions about events and products to their followers or other specific users. Many researchers consider Twitter a rich resource for opinionated data (Agarwal et al. (2011); Martínez-Cámara et al. (2014)). Further evidence of this is provided by the yearly competition SemEval, which is held at the National American Chapter of the Association for Computational Linguistics (NAACL). The most popular tasks involve developing techniques to perform sentiment analysis on Twitter and competing with other teams (Rosenthal et al. (2014, 2015)).
Sentiment analysis has been widely used on Twitter. It has been used by Tumasjan et al. (2010) to predict election polls and by Si et al. (2013) to make real-time predictions on the S&P100 index. In both instances, opinions on Twitter gave good indications to what the outcomes might be, and the use of sentiment analysis outperformed other prediction methods. These two examples show the promise of analysing opinions on Twitter and the diverse area of applications.
Prediction is only one of the many uses of sentiment analysis on Twitter. According to Belleghem et al. (2011), more than 50% of social media users followed brands in 2011. Unsurprisingly, this has led to more opinions being shared about brands and their products on social media (Lim and Buntine

(2014)). Therefore, social media has become a rich resource for organisations to obtain unsurveyed feedback to help improve their brand image and products. In Ghiassi et al. (2013), researchers were able to apply sentiment analysis to successfully determine a consumer sentiment about a particular brand. While this case is limited to a specific brand, in this case a famous singer, the theory behind the implementation can be used to train the algorithm to determine the consumer sentiment of any brand.

While it is agreed that Twitter is an excellent resource for opinion mining, the concise and informal nature of social media posts makes it computationally difficult to process the language within them. The difficulties with preforming sentiment analysis on Twitter posts was first explored by Go et al. (2009). They pointed out that the language model of Twitter posts contains many spelling errors, slang words and emoticons. The inconsistency in the language used poses difficulties when training machine learning classifiers because to be accurate, they require consistent data. However, this problem has been overcome by researchers who have developed Twitter-specific dictionaries and lexicons. Examples of these are the NRC Hashtag Emotion Lexicon and the Sentiment 140 Lexicon (Svetlana Kiritchenko and Mohammad (2014)). These resources help alleviate the problems posed by the language model of Twitter by using language commonly used in Twitter. Another drawback of Twitter discussed by Go et al. (2009) is the varied domain of subjects that users of Twitter may talk about. Research has shown that higher accuracy of sentiment analysis can be obtained by focusing on domain specific classifiers, (Ghiassi et al. (2013); Feldman (2013), because a domain specific lexicon can be built for the classifiers. In the case of this work, the classifier cannot be built for a specific domain since the product that it is being built for, Thryyve, needs to analyse the sentiment of many different domains.

### 1.3.4 Machine Learning in Sentiment Analysis

Machine learning classifiers have been the most used tool in Twitter sentiment analysis from early studies Go et al. (2009) to the winning classifier of the most recent SemEval competition (Hagen et al. (2015)). As discussed by Rosenthal et al. (2015) in the evaluation of all the entries for the 2015 SemEval competition, all the teams used some method of supervised learning with classifiers such as a Support Vector Machine, Maximum Entropy and Linear Regression being among the most popular. It is also noted that this is no different from previous years (Rosenthal et al. (2014)). The different classifiers and their ability to improve upon the sentiment analysis method currently used in Thryyve will now be explored.
Support vector machine (SVM) classifiers were first implemented for Twitter sentiment analysis by Go et al. (2009) and proved to be the most accurate classifier out of the three they developed. A SVM is a linear classifier that works by using straight lines to separate the different classes in a feature space, in this case, the polarity of the text (Medhat et al. (2014)). More recently, Mohammad et al. (2013) used more features than Go et al. (2009) to train their SVM. They used a LibSVM package (Chang and Lin (2011)) and a linear kernel classifier. They found that the linear kernel performed better than other classifiers such as Maximum Entropy. Furthermore, in the 2015 SemEval competition, two out of the top five teams for the message level sentiment analysis task used SVMs. The first team, Webis (Hagen et al. (2015)), also was the winner. The team's method was to reproduce four entries from previous years of the competition and use them as an ensemble, which is a method used to combine several classifiers. Each classifier used slightly different sets of features. Bringing them together combined their individual strengths. Two of the classifiers used in the ensemble also used an SVM classifier. While the method of an ensemble produces a better result (Rokach (2010)), the implementation is too complex for any company with-

out in-depth knowledge of each classifier. The implementation of a single SVM, on the other hand is more straightforward, with libraries and support available for commercial use.

As with SVMs, Maximum Entropy (ME) has been a popular method for analysing sentiment on Twitter. ME classifiers take the labelled data as a set of features and calculate a weight for each feature. The weights are then combined to determine the most probable polarity label for a data set (Mcdonald et al. (2009)). Its basic implementation is as complex as SVMs, but the results produced are not as promising.

Another classifier that is increasingly applied to sentiment analysis is neural networks. Last year's runner up for the SemEval competition used a deep convolutional neural network (Severyn and Moschitti (2015)). However, as with the ensemble method, training and implementation are complex compared to that for an SVM. Severyn and Moschitti (2015) also used computers with great power to achieve their results. Few small businesses have the resources to train deep convolutional neural networks, so this type of classifier is not a viable option.

### 1.3.5 Lexicon Based Sentiment Analysis

Machine learning classifiers are not the only tool being used for sentiment analysis. Recent surveys such as that by Medhat et al. (2014) have shown the popularity of Lexicon-based approaches to obtaining sentiment from text. Lexicon-based approaches for sentiment analysis may include using dictionaries with lists of words and their corresponding polarity (Taboada et al. (2011)). Many such dictionaries have been developed, including the automatically created SentiWordNet (Baccianella et al. (2010)) dictionary and the manually created Bing Liu Opinion Lexicon (Hu and Liu (2004)). More recently, Twitter-specific dictionaries have been created such as the NRC Hashtag Sentiment Lexicon (**?**) and the Sentiment 140 Lexicon (Mohammad et al. (2013)). By using these dictionaries, each word in each tweet can be given a polarity and the overall sum of the tweet can be computed to find its polarity. However, this method has drawbacks, since in many cases the same word can have several meanings when used in different contexts. For example, the word 'sick' can be a negative, expressing illness. But in slang, 'sick' can also be a positive word. This can lead to the wrong polarity being applied to a word and a tweet being incorrectly classified.

While the use of lexicon-based sentiment analysis has yielded good results (Medhat et al. (2014)), recent sentiment analysis tools have used both machine learning classifiers and lexicon-based approaches to great effect (Mohammad et al. (2013); Chikersal et al. (2015)). In both studies, the impact that each feature had on the overall accuracy of the classifier was stated. By including lexicon-based approaches, both classifiers showed improvement. Therefore, using lexicons to improve the accuracy of Xeidos' sentiment analysis classifier will be explored in this work.

### 1.3.6 Feature Selection for Sentiment Analysis

The focus of the work thus far has been on different methods of classifying tweets. However, selecting the correct features to train the classifier on is arguably more important than the classifier used. A complex classifier that is trained on poor features will return worse results than a simple classifier trained on good features (Guyon and Elisseeff (2003)). The next section will briefly explain the use of different features for sentiment analysis. Each feature used will be defined in depth in section 2.2.

Early work by Go et al. (2009) used POS (Part of Speech) taggers and n-grams. POS taggers split a sentence into its different parts of speech, such as adjectives and nouns. N-grams split sentences into a set of single words (unigrams), a set of two consecutive words (bigrams) and a set of three consecutive words (trigrams). They found that the best results came from using unigrams and bigrams together. While Go et al. (2009) worked with three different features, Mohammad et al. (2013) used nine different features. They discovered that the most influential features were those based on lexicon-based approaches and n-grams. Small improvements were made by including negation and POS taggers as features. Negation is defined by Mohammad et al. (2013) as adding the suffix '_NEG' to words that follows a negation word, such as "no" and "shouldn't". More recently, Hagen et al. (2015) attributed their success to the depth of features that their ensemble of classifiers used. Features they used, including those introduced by Mohammad et al. (2013), were stemming, length of a tweet and a manually created emoticon and abbreviation dictionary. Each of these features improved the classifier when implemented originally (Günther and Furrer (2013); Proisl et al. (2013)). The choice of features has great impact on the overall accuracy of a classifier. This work aims to find the best combination of features to train the SVM classifier.

## 1.4 Microsoft Azure Machine Learning Classifier

The current classifier used by Xeidos (hereinafter called the Azure classifier) is a two-class support vector machine classifier built on the Microsoft Azure Machine Learning platform. Being two class, the Azure classifier is only able to classify positive and negative tweets, like the classifier built by Go et al. (2009). The similarities continue, as the Azure classifier also uses the same data produced by Go et al. (2009) and only uses uni-grams and bi-grams as features. The classifier is available for anyone with a free Microsoft Azure account, and a tutorial is available (Microsoft (2014)). The Azure classifier is popular with those who have built their products on the Microsoft Azure platform. Azure comes with many out-of-the-box tools for easy integration of different features. However, the tutorial only goes so far in explaining how accuracy is obtained. This work will lift the lid of the classifier to give users

such as Xeidos a better understanding of how accuracy is achieved.
While the Azure classifier does not classify neutral tweets, it uses the confidence scores produced by the binary SVM classifier and a threshold to map tweets to their predicted label. Therefore, the metrics represented in the tutorial are only true for classifying positive and negative tweets. Because one of the objectives of this work is to verify the Azure classifier, the same dataset and machine learning method will be used on both classifiers to reproduce similar accuracy.
The rest of this work will be split into two sections. Firstly, the different aspects of the method such as data collection, feature selection and choice of machine learning classifier will be discussed in great detail in the Method section. Lastly, the results, conclusion and future work will be considered in the Results section.

# 2 Method

In this section, the data collection, feature selection and machine learning classifier are discussed. Figure 2 outlines the methodology used in this work.

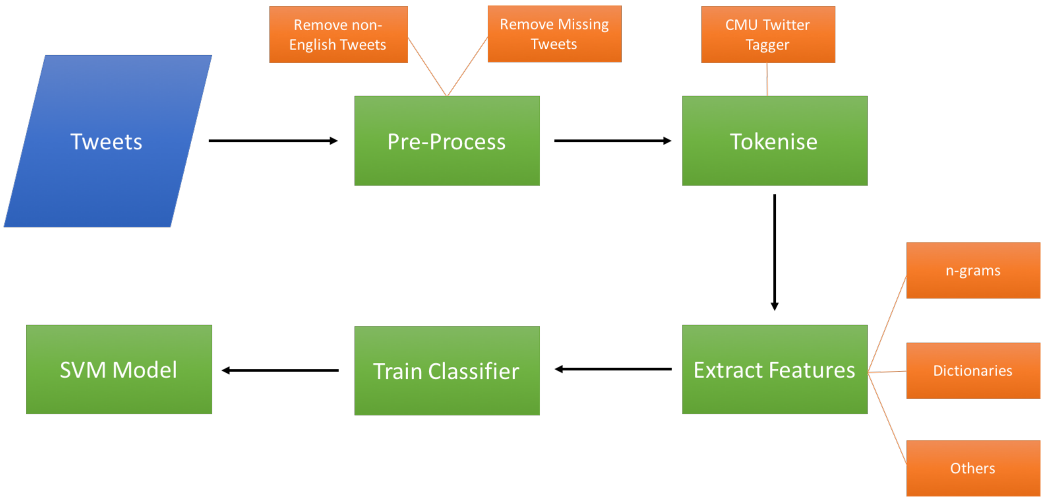


Figure 2: Visual representation of sentiment analysis methodology

## 2.1 Data Collection

Before any Machine Learning method can be applied to a problem, data are required. The ease at which data can be procured from Twitter is the main reason it is chosen as the social media platform for this experiment. Twitter provides an Application Program Interface (API) (Twitter (2016)) that many researchers have used to obtain Twitter data.
Competitors in the SemEval competitions use a provided dataset of tweets. In the 2013 competition, Nakov et al. (2013) explained that one factor that had been slowing down improvement in sentiment analysis in recent years is the lack of reliable data. To remedy this, the committee decided to create a publicly available dataset for future researchers to use. Nakov et al.

(2013) obtained the data set by collecting millions of tweets from January 2012 to January 2013. By extracting the named entities from these tweets, they were able to identify popular topics on a specific date within the time period. Tweets that expressed some sentiment towards those topics were then extracted by filtering them through the sentiment lexical resource SentiWordNet (Baccianella et al. (2010)). However, what made the production of this dataset better than others was use of Amazon's Mechanical Turk Workers (Turkers) to manually label the tweets. The Turkers were chosen based on their prior experience and approval rating of at least 95%.
This work also uses the publicly available dataset created by Nakov et al. (2013) for the 2013 SemEval competition because it contains three polarities of tweets as opposed to Sentiment140's two polarities. The data set can be obtained for free from the SemEval 2013 website. Training, development and testing data sets were used in this work. The development set was used as validation to ensure the testing set would only be used once. To protect the users' identities, the actual tweets were not stored, but the tweet ID and User ID were provided. A provided script was then used to query the Twitter API and retrieve the tweets. Table 1 shows the number of positive, negative and neutral tweets within the SemEval dataset.

Table 1: Polarity of tweets in SemEval dataset

| Data set | Positive | Negative | Neutral |
|---|---|---|---|
| Training | 3,662 | 1,466 | 4,600 |
| Development | 341 | 173 | 418 |
| Test | 1,573 | 601 | 1,640 |

### 2.1.1 Preprocessing

Once the dataset was obtained, it needed to be formatted and pre-processed before it could be passed to the classifier. Firstly, the dataset was formatted to remove unnecessary information such as tweet and user ID. These were removed using regular expressions in a text editor. Due to the length of time that had passed since the tweets had been originally collected by Nakov et al. (2013), some of the users had either deleted their accounts or deleted certain tweets. Consequently, several tweets were missing from the original set and subsequently could not be retrieved. These were displayed in the original set as 'missing' where the text from the tweet should be. Table 2 shows the final data set used for training compared to the original set. The available training set, shown as a percentage of the original training set, makes it clear that none of the three polarities has decreased significantly more than the others. Overall, the data have been reduced by 10.47%, which still leaves a large amount of good quality tweets with which to train the classifier. The missing tweets were removed from the dataset using regular expressions.
Further pre-processing included using the Carnegie Mellon University part-of-speech (POS) tagger (Owoputi et al. (2013)) to split the tweets into tokens and assign each token a POS tag. The CMU tagger has been constructed specifically for online communication methods such as Twitter. It includes Twitter-specific tags such as emoticons and hash-tags, which make it the ideal POS tagger for Twitter sentiment analysis. The tagger was originally built for Java, but wrappers have been created to allow the CMU tagger to be used in Python. The wrapper used in this work was developed by Ian Ozsvald and is free to use under the MIT license. The tagger was used in a Python script that ran each tweet through the CMU tagger before outputting the tokenised tweet to a file along with its POS tags and sentiment. The final format of a tweet passed into the training script would be "<Tokenised tweet><tab><POS tags><tab><Sentiment>".

The sentiment analysis classifier created in this work can only analyse tweets written in English. Tweets in the training set in other languages were removed during pre-processing. One of the tags used by the CMU tagger is the 'Garbage' tag, which is assigned to all nonsensical and non-English words. Tweets written in languages other than English always contained three or more 'garbage' tags in a row. These tweets, which accounted for 4.66% of the original set, were removed from the data set using regular expressions. Table 2 shows a comparison of the final number of tweets used for training and the original set.

Table 2: Difference between Original data set and data set Used

| Data set | Positive | Negative | Neutral |
|---|---|---|---|
| Training - Original | 3,662 | 1,466 | 4,600 |
| Training - Available | 3,049 | 1,205 | 3,930 |
| Percentage of Original | 83.26% | 82.20% | 85.43% |

## 2.2 Feature Selection

As discussed in the introduction, feature selection plays a large role in creating a successful classifier. The features are used to create a feature extractor, which transforms tweets into numerical features ready for machine learning. This section describes the use of n-grams in both the classifier built in this work and the Azure classifier. Part-of-speech tags and sentiment lexicons will also be explored to see how the accuracy of a classifier can be improved.

### 2.2.1 N-Grams

One of the most popular features used to classify sentiment on Twitter is n-grams. They have been used by Mohammad et al. (2013); Chikersal et al. (2015); Günther and Furrer (2013); Proisl et al. (2013) to good effect. N-grams are created by splitting sentences into 'n' consecutive words. Figure 3 shows an example of a sentence split into un-igrams, bi-grams and tri-grams. Certain words and combinations of words are used frequently to convey certain polarities. Using uni-grams leads to the complete loss of sentence structure, since only single words are assigned their sentiment. To remedy this, bi-grams and tr-igrams are used to retain some structure.

The extraction of n-grams from tweets in this work is done by a simple function that splits a tweet into uni-grams, bi-grams and tri-grams. The different occurrences of the n-grams from every tweet in the training set are stored in a feature set. Storing the feature set as a set data type in Python prevents any repetition of words. Each tweet is then represented as 1's and 0's, depending on whether the n-gram is present or absent in the tweet. This representation, also known as a feature vector, is passed to the support vector machine in place of the actual tweet.

Table 3 shows an example of how a tweet would be represented given all the n-grams present in the feature set. In actual practice, the feature set becomes quite large because it contains every unique n-gram from the entire training set. One method to reduce the size of the set was achieved by using a list of non-sentiment bearing words that are removed from the set. These words are taken from the Van stop word list (Van Rijsbergen (1979)). After the feature set is built, any word that is in both the stop word list and the feature set is removed.

Another method used to minimise the size of the feature set is to change the language used in the n-grams to be as regular and uniform as possible. Examples of the irregular use of language in Twitter are words with many repeated letters. For example, the word 'cool' can be entered as an elongated version: 'cooooooool'. Both standard and elongated versions should be classified with the same sentiment by the SVM. To enable better classification, a simple method used in the work of Go et al. (2009); Mohammad et al. (2013) is implemented. Any letter repeated in a word more than twice is reduced to two repetitions. Similarly, it is important for all words to be in lower case and to be stripped of unnecessary punctuation. A simple use of Python's .tolower() function returns the tweets in lower case, and the use of regular expression removes any trailing punctuation in the words in a tweet. After following these steps, the final feature set should contain only unique and uniform n-grams.

Table 3 shows that when only n-grams are used as features, the feature set and the feature vector contain the same number of elements. This is because the feature vector representation of a tweet is created by comparing each n-gram in the tweet to every n-gram in the feature set. A binary representation is used to represent the presence and absence of the n-grams in the feature vector. Therefore the feature vector is mainly a list of 0's with several 1's sparsely distributed, as shown in Table 3. As is discussed in section 2.3, the sparse feature vector is a main reason for choosing the support vector machine as the classifier for this machine learning task.

Table 3: An example to show the usage of n-grams and feature lists to represent tweets

| | |
|---|---|
| **Example feature set** | hasn't, left, further, cool, won't, happy, sad, fishing, going, hate, fortunate, won't be, i hate, help them, for glory, he's left |
| **Example tweet** | It's so sad that he's left the show... It won't be the same. |
| **Example feature vector** | 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 |

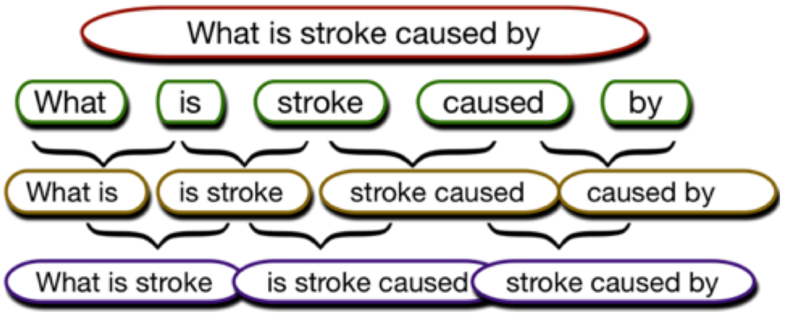


Figure 3: An example how different length n-grams can be constructed from a sentence

### 2.2.2 Part-Of-Speech Tags

The part-of-speech tags (POS) obtained from the CMU tagger are counted and used as features for sentiment analysis. The reason for counting POS tags is the prevalence of different tags depending on the polarity of a tweet. This is discussed in Pak and Paroubek (2010), in which the counts of POS tags in positive and negative tweets are compared. The results showed that tags such as possessive pronouns and adverbs are more common in positive tweets, while tags such as past tense verbs are more common in negative tweets. Furthermore, Pang and Lee (2008); Liu (2012) prove the inclusion of POS tag counts improves the accuracy of the classifier. Using POS tags as a feature is therefore an simple way to improve the classifier, since the POS tags are already available to use.

In section 2.1.1, the format of the tweet passed to the main program is defined, in which the POS tags of each token are included. Within the main program, the POS tags of a tweet are passed to a function to count how many of each of the 25 tokens are present. The count of each POS tag is appended to the end of the feature vector described in section 2.2.1. Table 4 gives an example of how the POS tags are counted from a tweet. The last row of Table 4 shows the running feature vector, which is the feature vector from section 2.2.1 with the POS tag count added. This gives a better understanding as to how the tweet will finally be represented once all the features are added.

Table 4: An example to show how POS tags are used as features. Please note that only non-zero values are shown. In practice, non-zero values will be added to the feature vector. *Key for POS tags: L= Nominal verbal, R= Adverb, A= Adjective, P= Preposition, V= Verb, D= Determiner, N= Noun, ,= Punctuation, O= Pronoun.*

| | |
|---|---|
| **Example tweet** | It's so sad that he's left the show ... It won't be the same. |
| **POS tags** | It's/L, so/R, sad/A, that/P, he's/L, left/V, the/D, show/N, .../, ,It/O, won't/V, be/V, the/D, same/A, ./, |
| **Example feature vector** | 2,1,2,3,2,1,2,1 |

### 2.2.3 Sentiment Lexicons

Another feature that can be used to improve a sentiment analysis classifier is sentiment lexicons. From Mohammad et al. (2013), sentiment lexicons are lists of words with associations to positive and negative sentiments. As discussed in the introduction, several sentiment lexicons, or sentiment dictionaries, are available. These dictionaries fall into two classes, automatically created and manually created. This work uses two automatically created lexicons and one manually created lexicon.
The automatic lexicons that will be used are the NRC Hashtag Emotion Lexicon and the Sentiment 140 Lexicon (Svetlana Kiritchenko and Mohammad (2014)). These dictionaries were chosen because they are Twitter-specific and have been shown to increase the accuracy of an SVM classifier. The Hashtag Lexicon represents a corpus of tweets collected by searching for strongly

positive or negative words. After non-English tweets were removed, the final lexicon has 183,354 entries of possible hashtags. Each entry in the corpus includes a string in the form *'#somehashtag'*, followed by a polarity score between positive infinity and negative infinity. For some entries, the string includes several popular punctuation marks that precede the hashtag, for example *'!!#somehashtag'*.
The Sentiment 140 Lexicon uses the corpus built by Go et al. (2009) and labels words represented as unigrams and bigrams with the polarity of the tweet. In both cases, many tweets from the corpus hade no strong polarity. A further processing step removed all tweets that did not contain any polarity seed words, which are words with strong polarity towards positive or negative.
The third Lexicon used in this work is the MPQA Subjectivity Lexicon (Wilson et al. (2005)), a manual lexicon created by taking subjective phrases from the previously created MPQA Opinion Corpus. The phrases were then manually annotated according to a set of rules and assigned a polarity of positive, negative or both. For the purposes of this work, words that were assigned the polarity of both were removed. The polarity labels were also changed to 1 for positive words and -1 for negative words. The method of using lexicons as features is similar to that implemented in Mohammad et al. (2013). Their system was the highest ranked hybrid system that used lexicons and standard features. Each lexicon will generate its own set of features. For each token $w$ in the tweet that exists in the lexicon, the following features will be calculated:

- the number of tokens $w$ with the polarity score not equal to zero;
- the sum of the positive and negative score of all $w$;
- the maximum score of all $w$; and
- the score of the last token $w$ in the tweet.

Table 5 gives an example of the calculated features for the example tweet using the Sentiment 140 Lexicon. In practice, each lexicon will create its own list of features. Furthermore, each feature and lexicon are removed from the classifier to show the overall impact they have on it. This will be further explained in section 3.1.

Table 5: The calculated scores for an example tweet using the Sentiment140 lexicon

| | |
|---|---|
| **Example tweet** | It's so sad that he's left the show ... It won't be the same. |
| **Number of tokens with a polarity score** | 12 |
| **Sum of POS and NEG scores** | -6.34 |
| **Maximum score of all tokens** | 'so sad': -3.662 |
| **Score of last token** | 'same': -0.146 |
| **Example feature vector** | 12, -6.34, -3.662, -0.146 |

### 2.2.4 Negation

Negation is a lexical tool used in Pang et al. (2002) to alter words in sentences that contain specific negative words. The rewards of this feature are an improvement in the accuracy of their classifier. The implementation of negation in this work is similar to that used in Pang et al. (2002), further described in Potts (2011).

As discussed in section 2.2.1, the feature set of n-grams is created from the training set of tweets. By introducing negation, the words within this feature set are expanded to include words with the suffix '_NEG'. The words that have this suffix appended to them will follow this rule taken from Potts (2011): "Append a _NEG suffix to every word appearing between a negation and a clause-level punctuation mark". Clause level punctuation is defined as one of '.:;!?,' and all the words considered as negation can be seen in Table 6. Negation words end in "n't" and also include those that are missing the punctuation and have been misspelled. An example of the implementation of negation can also be seen in Table 6.

Two features are created with the introduction of negation. Firstly, the feature set is altered to contain words that have negation. This will allow the machine learning classifier to identify tweets with these negated tokens and to conclude that the tweet itself is negative. Secondly, the count of negated tokens will provide a feature, the intuition being that more negated tokens will lead to a tweet being negative. This value will be appended to the feature vector.

Table 6: An example to show the use of negation on a tweet and the words considered to be negation

| | |
|---|---|
| **Negation words** | never, no, nothing, nowhere , noone, none, not, havent, hasnt, hadnt, cant, couldnt, shouldnt, wont, wouldnt, dont, doesnt, didnt, isnt, arent, aint, *n't |
| **Example tweet** | It's so sad that he's left the show ... It won't be the same. |
| **Tokens with negation** | it's, so, sad, that, he's, left, the, show, ..., it, won't_NEG, be_NEG, the_NEG, same_NEG, . |
| **Count of negated tokens** | 4 |

## 2.3 Machine Learning Classifier

Support vector machines (SVMs) are popular in text classification (Pang and Lee (2008)). As discussed in the introduction, many systems entered in the SemEval competition used SVMs (Rosenthal et al. (2015)).
Joachims (1998) explains why SVMs are well suited to text categorisation problems. Firstly, many text categorisation problems use a large number of features. The over-fitting protection within SVMs enables them to handle large feature spaces. Secondly, many of the features are important to the integrity of the information, so feature reduction can harm the accuracy of the classifier. While other classifiers struggle to combine so many features, SVMs do not. Finally, SVMs require a relatively small amount of parameter

tuning because they find good parameter settings automatically, which saves time during the training process.

The SVM library chosen for this work is the LibSVM machine learning library (Chang and Lin (2011)) because of its popularity and the support available. As suggested in Chang and Lin (2011), there is great importance in selecting the right parameters for the classifier. The linear kernel was chosen because literature suggests it produced better results than other kernels (Mohammad et al. (2013)). The only parameter that required tuning was the regularisation parameter, which instructs the SVM about how well to classify the training data. Increasing the regularisation parameter can lead to over-fitting the training data, while leaving it too low can lead to under-fitting. During the experimentation, different values of the regularisation parameter will be tested to find the best result.

The SVM will accept tweets in the form a feature vector, which will be a numerical representation of a tweet with all the features. Figure 4 shows a visual example. With this information, as with many machine learning classifiers, the SVM will find the best method of separating the points created by the feature vectors of each tweet. SVMs differ from other classifiers in that they prioritise separating the most difficult points first with the intuition that the easier points will also become separated (Bennett and Campbell (2000)).

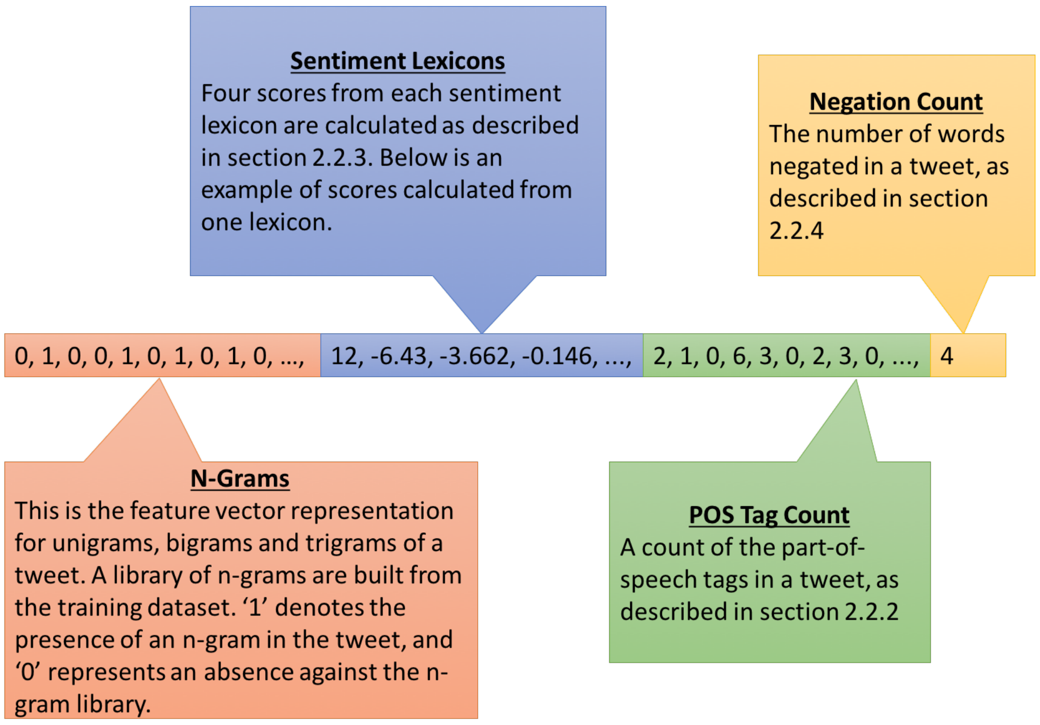

**Sentiment Lexicons**
Four scores from each sentiment lexicon are calculated as described in section 2.2.3. Below is an example of scores calculated from one lexicon.
**Negation Count**
The number of words negated in a tweet, as described in section 2.2.4
0, 1, 0, 0, 1, 0, 1, 0, 1, 0, ...,
12, -6.43, -3.662, -0.146, ...,
2, 1, 0, 6, 3, 0, 2, 3, 0, ...,
4
**N-Grams**
This is the feature vector representation for unigrams, bigrams and trigrams of a tweet. A library of n-grams are built from the training dataset. '1' denotes the presence of an n-gram in the tweet, and '0' represents an absence against the n-gram library.
**POS Tag Count**
A count of the part-of-speech tags in a tweet, as described in section 2.2.2

Figure 4: A visual representation of the feature vector of a tweet, that is passed to the Support Vector Machine.

# 3 Results

In this section, the results obtained from the experiments will be explored. Firstly, the possible values for the regularisation parameter are tested, followed by the outcome of the results of the sentiment analysis. The final model will then be compared to the Azure classifier and a state-of-the-art classifier. Finally, some concluding thoughts will be offered, and possible future work will be discussed.

## 3.1 Experiments

### 3.1.1 Optimising the Regularisation Parameter

As discussed in section 2.3, only the regularisation parameter requires tuning in a support vector machine (SVM). LIBSVM's default regularisation parameter is set to 1. A range of parameters between $10^5$ and $10^{-5}$ was tested on a classifier that used only the n-grams as features. The classifier was tested twice, once for three polarities, Figure 5, and once for two polarities, Figure 6. The regularisation parameter that returned the best value for the classification of all three polarities was 0.1. The regularisation parameter that returned the best value for the classification of only the positive and negative polarities was 10.

A regularisation value that is too high creates a classifier that over-fits the training data and is unable to generalise to the development data. If the value is too low, the classifier under-fits the training data and does not perform well on development data. Both cases result in loss of accuracy. The regularisation value that returns the best accuracy is 0.1 and 10 for all three polarities and only negative and positive polarities respectively. These values are used in the rest of this work.

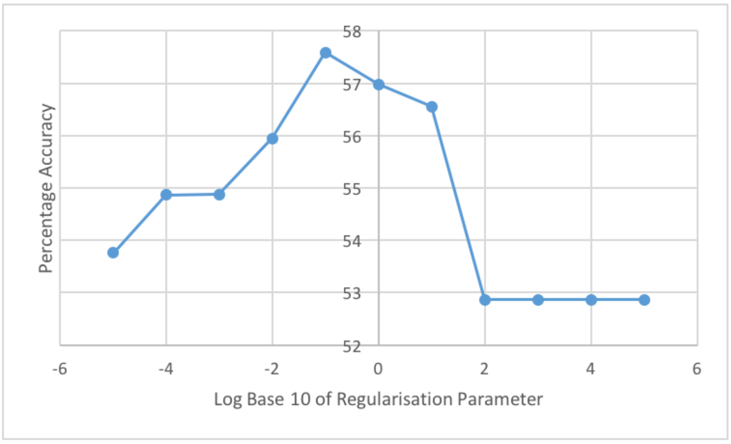


Figure 5: The accuracy on the development data, produced by the classifier when using different values for the regularisation parameter. For the classification of all three polarities of tweets.

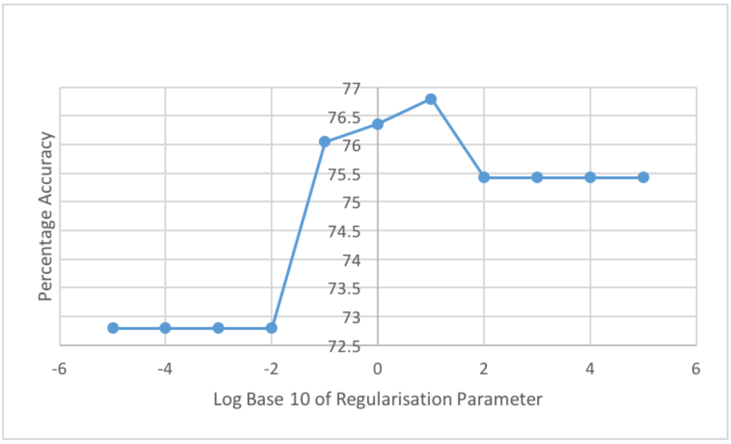


Figure 6: The accuracy on the development data, produced by the classifier when using different values for the regularisation parameter. For the classification of positive an negative tweets.

### 3.1.2 Finding the Best Feature Configuration

It is imperative to find the best configuration of features so the classifier will produce the best accuracy on the test set. To find the best configuration, an experiment was devised that highlighted the effect on the accuracy of the classifier when each feature was removed. These accuracies are then compared to the accuracy of the classifier when all the features are used. The results from this experiment can be seen in Table 7 and are discussed further in section 3.2.1.

From these results, the features that have a negative effect on accuracy can be identified. These features are then removed from the classifier, and the experiment that produced the results in Table 7 is repeated. The results from this experiment can be seen in Table 8 and are further discussed in section 3.2.1.

Table 7: The percentage accuracy of the classifier with all but one feature. The difference is calculated from the accuracy with all features used on the test set.

| Feature | Training Set | Test Set | Difference |
|---|---|---|---|
| All Features | 99.98% | 80.49% | - |
| w/o N-grams | 84.04% | 80.68% | -0.19% |
| w/o uni-grams | 99.98% | 79.61% | 0.88% |
| w/o bi-grams | 99.98% | 80.49% | 0.00% |
| w/o tri-grams | 99.98% | 80.93% | -0.44% |
| w/o Lexicon Dictionaries | 99.98% | 77.99% | 2.50% |
| w/o MPQA Dictionary | 99.98% | 80.93% | -0.44% |
| w/o Hashtag Sentiment Lexicon | 99.98% | 80.11% | 0.38% |
| w/o Sentiment 140 Lexicon | 99.98% | 79.30% | 1.19% |
| w/o POS count | 99.98% | 80.55% | -0.06% |
| w/o Negation | 99.98% | 80.86% | -0.37% |
| w/o Negation Count | 99.98% | 80.55% | -0.06% |

Table 8: Further Fine Tuning of Feature Combination

| Feature | Training Set | Test Set | Difference |
|---|---|---|---|
| All Features | 99.98% | 80.49% | - |
| w/o tri-grams, MPQA Dict | 99.98% | 81.30% | -0.81% |
| w/o tri-grams, MPQA Dict, POS Count | 99.98% | 80.36% | 0.13% |
| w/o tri-grams, MPQA Dict, Negation Count | 99.98% | 80.11% | 0.38% |
| w/o tri-grams, MPQA Dict, Negation | 99.98% | 80.93% | -0.34% |

### 3.1.3 Comparison with Azure Classifier

The first objective of this work is to verify the results produced by the Azure classifier. To do so, the same features that were used to train the Azure classifier are used to train the classifier made in this work. The data used on by the Azure classifier were changed to make sure both classifiers used the same data from the SemEval competition. Only positive and negative tweets were used, because the Azure classifier was a two class SVM. No other changes were made to the Azure classifier. Table 9 shows the comparison of accuracy from the Azure classifier and from the developed classifier.
The second objective of this work is to discover methods that can improve the accuracy of a sentiment analysis classifier. Table 9 shows how the inclusion of extra features affects the overall accuracy of the classifier. This was implemented by adding the POS tag count, lexicon scores and negation to the feature vector, as discussed in section 2.2. The same training set and testing set are used as before. The results are discussed in section 3.2.2.

Table 9: The first two columns show the accuracy of both the classifier developed in this work and the Azure classifier using the same features. The final column shows the increase in accuracy when additional features are included in the developed classifier.

| | Developed Classifier | Azure Classifier | Developed Classifier |
|---|---|---|---|
| | N-grams | N-grams | Additional Features |
| Training Set | 99.80% | 95.10% | 99.80% |
| Test Set | 76.80% | 77.00% | 81.30% |

### 3.1.4 Comparison with state-of-the-art classifier

Because the Azure classifier uses unigrams and bigrams as features, it is impossible to gauge whether the additional features implemented in the developed classifier were used to their full potential. Therefore, the developed classifier is compared to a state-of-the-art classifier, the results of which can be seen in Table 10. The chosen classifier is the NRC-Canada classifier (Mohammad et al. (2013)), which was submitted for the 2013 SemEval Competition. The NRC-Canada classifier was chosen because it was trained and tested on the SemEval dataset, which is the dataset used to develop the classifier created in this work. Positive, n The NRC-Canada classifier uses similar features such as Lexicons, POS tags, n-grams and negation, but also uses additional features not explored in this work. As discussed in section 2.2.3, the scores calculated from the lexicons were inspired by the same team that created the NRC-Canada classifier, making it the ideal comparison for the classifier developed in this work.

As opposed to using the percentage accuracy of the classification on the test set, the SemEval competition uses the f-score to evaluate the submitted classifiers. The f-score is calculated using the method in Nakov et al. (2013). Firstly, the precision (1) and recall (2) for both positive tweets and negative tweets are calculated using equations (1) and (2), respectively. The f-score of each label is then calculated using equation (3), and the average f-score over both labels is calculated using equation (4).
Table 10 shows the effect each feature has on the f-score of the classifier. The classifier was trained with the SemEval 2013 training and development dataset, with different features removed from the feature vector. Removing features highlights the importance of each feature to the classifier. Table 10 also shows the loss or gain in accuracy as each feature is removed compared with having all features in the classifier.
The results in Table 10 are discussed further in section 3.2.3.

$$Precision_{\mathrm{A}} = \frac{total\,correctly\,predicted\,labelA}{total\,predicted\,labelA} \tag{1}$$

$$Recall_{\mathrm{A}} = \frac{total\,correctly\,predicted\,labelA}{total\,labelledA\,in\,test\,set} \tag{2}$$

$$F - score_{\mathrm{A}} = 2\frac{Precision_{\mathrm{A}} * Recall_{\mathrm{A}}}{Precision_{\mathrm{A}} + Recall_{\mathrm{A}}} \tag{3}$$

$$F - score_{\text{classifier}} = \frac{F - score_{\text{A}} + F - score_{\text{B}}}{2} \tag{4}$$

Table 10: F-score of the developed classifier compared to a state-of-the-art classifier. Different features are removed to highlight their impact on the f-score.

| Features | Developed Classifier | Difference | NRC Classifier | Difference |
|---|---|---|---|---|
| All Features | 48.68 | - | 69.02 | - |
| w/o Lexicons | 37.70 | -10.98 | 60.42 | -8.60 |
| w/o N-Grams | 43.25 | -5.43 | 61.77 | -7.25 |
| w/o Negation | 46.40 | -2.28 | 67.20 | -1.82 |
| w/o POS | 47.10 | -1.58 | 68.38 | -0.64 |

## 3.2 Discussion

In this section, the results from the experiments will be discussed with regard to the objectives. Possible areas of future work will also be addressed.

### 3.2.1 Finding the Best Feature Configuration

The purpose of this experiment was to find the configuration of features that produced the best accuracy on the test set. The results from Table 7 show that removing certain features increases the accuracy of the classifier. This proves that not all of the features are useful and that further experimentation is required to determine which combination is best.

Table 8 shows that the best configuration returns an accuracy of 81.30% and uses all the features, with the exception of tri-grams and the MPQA dictionary. As discussed in section 2.2.1, bi-grams and tri-grams are used to capture more information of a tweet that may have been lost when a tweet is represented only as uni-grams. While this is true with bi-grams, the information provided by tri-grams does not prove helpful to the classifier. A possible reason for this is that the feature set created from the training set of tweets can become very large when all possible combinations of tri-grams are included. Also, the probability of three consecutive words appearing in a tweet from the testing set is quite low. These factors together lead to an unnecessarily large feature set with information that is not useful to the classifier. This hampers both the performance and the accuracy of the classifier. The MPQA Dictionary also proves to be detrimental to accuracy. As discussed in section 2.2.3, the sentiment lexicons and dictionaries are used to calculate the score of a tweet given the polarity of the words within them. While the other two lexicons used in this work are Twitter-specific, this dictionary is not. This is one possible reason why it has a negative effect on accuracy. Another reason is the ambiguity of the dictionary. Many of the words are classed as both negative and positive and are repeated many times. This can produce different scores for the same set of words and cause confusion during classification. Furthermore, unlike the other lexicons used in this work, the words are only given a negative, positive or both class. It does not contain varying levels of polarity, which mean words such as 'happy' and 'ecstatic' have the same level of polarity. Therefore, the MPQA Dictionary provides a weaker set of features than the Sentiment 140 and Hashtag lexicons.

### 3.2.2 Comparison with Azure Classifier

The main objective of this work is to validate the Azure classifier currently being used by Xeidos. To do this, this work developed a classifier that used the same features as the Azure classifier and trained it with the same data. Table 9 shows that both the Azure classifier and the classifier developed in this work have very similar accuracies when using the same features. This proves that the Azure classifier is working correctly, and therefore, the results it produces are valid.

The second objective of this work is to look at features that could improve the accuracy of a sentiment analysis classifier. The additional features explored in this work have shown an increase in accuracy of 4.30%. This improvement can translate to far more tweets being classified over time. In the use of a product such as Xeidos' Thryyve, producing a better sentiment classification can help improve sales and customer satisfaction. It is therefore imperative to use extra features such as those suggested in this work.

The classifier developed in this work can in theory benefit companies that use the Azure classifier. However, the integration of Python in the Azure Machine Learning platform is not currently extensive. Therefore, it would not be possible to simply swap the classifier developed in this work with the Azure classifier. Instead, the methods used to obtain the POS tags and lexicon features could be integrated into the Azure classifier.

### 3.2.3 Comparison with NRC-Canada Classifier

The second objective of this work is to explore features that can improve a sentiment analysis classifier. The results in Table 9 proved that the additional features can increase the accuracy of a classifier. However, because the Azure classifier does not use these features, it is impossible to know if the increase in accuracy is justified. As discussed in section 3.1.4, the use of the NRC-Canada classifier solves this problem because it is an excellent classifier to compare with the developed classifier.

Table 10 shows how each feature impacted the f-score of both classifiers. At first glance, the f-scores of the NRC-Canada classifier are far superior to those calculated from the developed classifier. This is expected because the NRC-Canada classifier uses far more features than those used in this work. Furthermore, the levels of development and expertise used to make the NRC-Canada classifier are well outside of the scope of this work. Regardless, the f-scores are still useful because they show how both classifiers are affected by the removal of each feature. The difference that is calculated determines how well the developed classifier is using the extra features.

Table 10 shows that the sentiment lexicons have the highest impact on the developed classifier, which is also true for the NRC-Canada classifier. The features extracted from sentiment lexicons are not exactly the same across both classifiers, and the NRC-Canada classifier is not reliant on just 3 features. For this reason, the lexicons have a greater positive impact on the developed classifier compared to the NRC-Canada classifier. The lexicon features are most useful when classifying negative tweets, which have an impact on the overall f-score because they are averaged. Therefore, the use of lexicons as an additional feature should definitely be considered in any sentiment analysis classifier.

As discussed in section 2.2.4, negation plays an important role by identifying negative words in a tweet. Since the f-score of negative tweets is generally worse than positive tweets (Mohammad et al. (2013)), the use of negation increases the f-score on negative tweets. This in turn increases the f-score of the overall classifier. Compared to the NRC-Canada classifier, the use of negation yields a bigger improvement in the developed classifier because the NRC classifier has more features to depend on. Regardless, negation is the third most influential feature for both classifiers. This proves that negation has been well implemented in the developed classifier.

The addition of POS tags as a feature makes a very small improvement to both classifiers. However, their simple implementation makes them an effortless feature to use. While small, the increase in 1.58 of the f-score can prove to be the difference in correctly classifying a tweet that would otherwise have been misclassified.
Overall, the comparison of f-scores between the the NRC-Canada classifier and the developed classifier have been shown to be useful in evaluating the effectiveness of each feature. This is best represented by the ranking of each feature in regards to its effect on both of the classifiers' f-scores. The features are ranked in the same order for both classifier. It therefore goes some way toward proving that the implementations of the features in the developed classifier are correct.

### 3.2.4 Future Work

A possible area of future work would be to discover a way to implement the Python script developed in this work within the Microsoft Azure Learning Platform. This work has already shown the improvement from including extra features, but until this is implemented so that businesses such as Xeidos can use it, the work can be seen as theoretical. Applying the features to the Azure classifier would be a sensible next step because this work has proved the validity of the results it produces, therefore making it a perfect starting point.

### 3.2.5 Evaluation of Work Plan

The work plan set out in section 1.2 was followed as best as possible. Thankfully, plenty of slack was given to all the tasks, as it was required. Specifically, the development of the classifier took much longer than initially anticipated. This was mainly due to the lack of prior experience in using tools such as LIBSVM, and the sharp learning curve that came with it.

The ordering of tasks proved to be critical. The knowledge acquired from the free courses enabled better understanding of relevant literature, which in turn helped build a successful classifier.
One factor that was not accounted for was the time taken during the testing of the support vector machine with experiments (feature extracting, training and testing), which took up to 30 minutes. Because of this, every effort was made to reduce the time complexity of the code. This involved restructuring and using different data structures to speed up searches.

## 3.3 Conclusion

This work has successfully developed a sentiment analysis classifier for Twitter data. The classifier has used different features, and their impact on the classification of tweets has been explored.
The main aim of this work was to verify the sentiment analysis classifier currently used by Xeidos. By creating a classifier in Python that used the same features and data, an accuracy of within 0.2% was achieved. Therefore, this work has validated the results produced by the Azure classifier.
The second aim of exploring features that can improve a sentiment analysis classifier has also been addressed. The use of POS tags, negation and especially sentiment lexicons showed an increase of 4.30% in the accuracy of the developed classifier.

# Bibliography

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., and Passonneau, R. (2011). Sentiment analysis of twitter data. In *Proceedings of the workshop on languages in social media*, pages 30–38. Association for Computational Linguistics.

Baccianella, S., Esuli, A., and Sebastiani, F. (2010). Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 2200–2204.

Belleghem, V., Eenhuizen, M., and Veris, E. (2011). Social media around the world, 2011. *The report by InSites Consulting.*

Bennett, K. P. and Campbell, C. (2000). Support vector machines: hype or hallelujah? *ACM SIGKDD Explorations Newsletter*, 2(2):1–13.

Carbonell, J. G. (1979). Subjective understanding: Computer models of belief systems. Technical report, DTIC Document.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chikersal, P., Poria, S., and Cambria, E. (2015). Sentu: sentiment analysis of tweets by combining a rule-based classifier with supervised learning. In *Proceedings of the International Workshop on Semantic Evaluation, SemEval*, pages 647–651.

Feldman, R. (2013). Techniques and applications for sentiment analysis. *Communications of the ACM*, 56(4):82–89.

Ghiassi, M., Skinner, J., and Zimbra, D. (2013). Twitter brand sentiment analysis: a hybrid system using n-gram analysis and dynamic artificial neural network. *Expert Systems with applications*, 40(16):6266–6282.

Go, A., Bhayani, R., and Huang, L. (2009). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1:12.

Günther, T. and Furrer, L. (2013). Gu-mlt-lt: Sentiment analysis of short messages using linguistic features and stochastic gradient descent. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 328–332, Atlanta, Georgia, USA. Association for Computational Linguistics.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182.

Hagen, M., Potthast, M., Büchner, M., and Stein, B. (2015). Webis: An ensemble for twitter sentiment detection. In *Proceedings of the 9th International Workshop on Semantic Evaluation*, pages 582–589.

Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM.

Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features.* Springer.

Lim, K. W. and Buntine, W. (2014). Twitter opinion topic model: extracting product opinions from tweets by leveraging hashtags and sentiment lexicon. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1319–1328. ACM.

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*, volume 999. MIT Press.

Martínez-Cámara, E., Martín-Valdivia, M. T., Urena-López, L. A., and Montejo-Ráez, A. R. (2014). Sentiment analysis in twitter. *Natural Language Engineering*, 20(01):1–28.

Mcdonald, R., Mohri, M., Silberman, N., Walker, D., and Mann, G. S. (2009). Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*, pages 1231–1239.

Medhat, W., Hassan, A., and Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113.

Microsoft (2014). Binary classification twitter sentiment analysis. https://gallery.cortanaintelligence.com/Experiment/59e52ed2895144ea964947c950a9c794?r=legacy.

Mohammad, S. M., Kiritchenko, S., and Zhu, X. (2013). Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. *arXiv preprint arXiv:1308.6242*.

Nakov, P., Kozareva, Z., Ritter, A., Rosenthal, S., Stoyanov, V., and Wilson, T. (2013). Semeval-2013 task 2: Sentiment analysis in twitter.

Owoputi, O., O'Connor, B., Dyer, C., Gimpel, K., Schneider, N., and Smith, N. A. (2013). Improved part-of-speech tagging for online conversational text with word clusters. Association for Computational Linguistics.

O'Reilly, T. (2005). O'reilly spreading the knowledge of innovators. *What is web*, 2.

Pak, A. and Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326.

Pang, B. and Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics.

Potts, C. (2011). Symposium tutorial: Linguistic structure. http://sentiment.christopherpotts.net/lingstruc.html#negation.

Proisl, T., Greiner, P., Evert, S., and Kabashi, B. (2013). Klue: Simple and robust methods for polarity classification.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39.

Rosenthal, S., Nakov, P., Kiritchenko, S., Mohammad, S. M., Ritter, A., and Stoyanov, V. (2015). Semeval-2015 task 10: Sentiment analysis in twitter. *Proceedings of SemEval-2015*.

Rosenthal, S., Ritter, A., Nakov, P., and Stoyanov, V. (2014). Semeval-2014 task 9: Sentiment analysis in twitter. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 73–80.

Severyn, A. and Moschitti, A. (2015). Unitn: Training deep convolutional neural network for twitter sentiment classification. In *Proceedings of the*

*9th International Workshop on Semantic Evaluation (SemEval 2015), Association for Computational Linguistics, Denver, Colorado*, pages 464–469.

Si, J., Mukherjee, A., Liu, B., Li, Q., Li, H., and Deng, X. (2013). Exploiting topic based twitter sentiment for stock prediction. *ACL (2)*, 2013:24–29.

Svetlana Kiritchenko, X. Z. and Mohammad, S. M. (2014). Sentiment analysis of short informal texts. 50:723–762.

Taboada, M., Brooke, J., Tofiloski, M., Voll, K., and Stede, M. (2011). Lexicon-based methods for sentiment analysis. *Computational linguistics*, 37(2):267–307.

Tumasjan, A., Sprenger, T. O., Sandner, P. G., and Welpe, I. M. (2010). Predicting elections with twitter: What 140 characters reveal about political sentiment. *ICWSM*, 10:178–185.

Twitter (2016). Rest api. https://dev.twitter.com/rest/public.

Van Rijsbergen, C. J. (1979). *Information retrieval.* Butterworths, London.

Wilson, T., Wiebe, J., and Hoffmann, P. (2005). Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics.

# Appendices

Code for Main

```
from functions import *
from svmutil import *
from datetime import datetime

# Begin timing script run time
startTime = datetime.now()

"""Call function to read in data files. These must be
in the correct format Consult README.txt for more
information"""
# Read in training data
data = read_data("../data/tweets/finalTrainingInput.txt")
# Read in testing data
dataT = read_data("../data/tweets/finalTestingInput.txt")



"""Read in the three Lexicons/dictionaries. These should
be imported into the dictionary type for quicker search.
The key is the n-gram/word and the value is the polarity
score."""

# Read in sentiment 140 lexicon
dictionary={}
with open
("../data/other/sentiment 140 lexicon/unigrams-pmilexicon copy.txt",'r')
```

```
as f:
    for i in f:
        el = i.split("\t")
        dictionary[el[0]] = el[1]
with open
("../data/other/sentiment 140 lexicon/bigrams-pmilexicon copy.txt",'r')
as f:
    for i in f:
        el = i.split("\t")
        dictionary[el[0]] = el[1]

# Read in Hashtag lexicon
dictionary2={}
with open
("../data/other/hashtag lexicon/unigrams-pmilexicon.txt",'r')
as f:
    for i in f:
        el = i.split("\t")
        dictionary2[el[0]] = el[1]
with open
("../data/other/hashtag lexicon/bigrams-pmilexicon.txt",'r') as f:
    for i in f:
        el = i.split("\t")
        dictionary2[el[0]] = el[1]

# Read in MPQA dictionary
dictionary3={}
with open
("../data/other/MPQA/subjclueslen1-HLTEMNLP05.txt",'r') as f:
    for i in f:
        el = i.split("\t")
        dictionary3[el[0]] = el[1]

"""Build the Feature List and get the preprocessed tweets"""
featureList, train_tweets= get_features(data,1)
test_tweets = get_features(dataT,0)

"""Convert feature list into feature set to remove repitions
of n-grams"""
featureList = list(set(featureList))


"""Pass the tweets, feature set and dictionaries to the function
to create feature vector and labels. The LibSVM package is then
used to train an SVM on feature vector and labels"""
print 'Building feature vector'
training_fv = make_fv(train_tweets, featureList, dictionary,
    dictionary2, dictionary3)

print 'Begin training'
classifier = svm_train(training_fv['labels'],
    training_fv['feature_vector'], '-t 0 -c 10 -q')
svm_save_model('classifierDumpFile', classifier)

print 'Accuracy on training set'
t_labels, t_accs, t_vals = svm_predict(training_fv['labels'],
    training_fv['feature_vector'], classifier)
print t_labels
print 'Begin Testing'
# Test the classifier
test_fv = make_fv(test_tweets, featureList, dictionary,
    dictionary2, dictionary3)

print 'Accuracy on Testing'
p_labels, p_accs, p_vals = svm_predict(test_fv['labels'],
    test_fv['feature_vector'], classifier)
print p_labels


print 'Time taken', datetime.now() - startTime
```

Code for Functions

```
"""This file contains the all of functions used in the
project"""
import re



"""This function creates n-grams. The tweets are passed
to it along with the number of n-grams required. A list
of n-grams are returned:
    -Adapted from http://stackoverflow.com/a/13424002
    Accessed:10/02/16"""



def ngrams(tweet, n):
    tweet = tweet.split(' ')
    output = []
    for i in range(len(tweet) - n + 1):
        output.append(tweet[i:i + n])
    return output



"""Pre-processes tweet
    -Replace URLs and user mentions with URL and AT_USER
    repectively
    -Remove the Hastag from a '#word' and return just the
    word
    -Replace repetitions of letters in a word with double
    occurence
    -strip words of unnesecary punctuation
    -Adapted from http://ravikiranj.net/posts/2012/code
    /how-build-twitter-sentiment-analyzer/
    Acessed:20/02/16"""


def preprocess(word):

    word = word.lower()
    # Convert a url to URL
    word = re.sub('((www\.[^\s]+)|(https?://[^\s]+))',
        'URL', word)
    # Convert a username to AT_USER
    word = re.sub('@[^\s]+', 'AT_USER', word)
    # Replace the hashtag without the hasthag
    word = re.sub(r'#([^\s]+)', r'\1', word)
    # used to replace 'cooooooooool' to 'cool'
    word = totwo(word)
    #strip punctuation
    word = word.strip('\'"?,.:;')
    #Remove additional white spaces
    word = re.sub('[\s]+', ' ', word)
    return word


"""Finds which of two numbers is furthest from 0
    -This function is used to get one of the lexicon
    scores"""
```

```
def biggestscore(a, b):
    if abs(a) > abs(b):
        return a
    elif abs(a) < abs(b):
        return b
    else:
        return 0



"""Get lexicon scores
    -4 scores are calculated and returned as a list"""



def scorelexicon(tweet, dictionary, n):
    count = 0
    score = 0.0
    maxscore = []
    b_score = 0
    lastscore = 0.0
    for i in range(len(tweet)):
        if tweet[i] in dictionary:
            sent = float(dictionary[tweet[i]])
            count += 1
            score += sent
            maxscore.append(sent)
            lastscore = sent

    if maxscore != []:
        b_score = biggestscore(max(maxscore),
            min(maxscore))
    else:
        b_score = 0.0

    return [count, score, b_score, lastscore]


"""Replace with letter reptitions from more than 2 to 2,
makes use of
regular expression
    -From: http://ravikiranj.net/posts/2012/code
    /how-build-twitter-sentiment-analyzer/
    Accessed: 20/02/16"""


def totwo(s):
    pattern = re.compile(r"(.)\1{1,}", re.DOTALL)
    return pattern.sub(r"\1\1", s)


"""Negation
    - Tweet is passed to function and words are marked
    with _NEG from the rules given in section
    2.2.4 of the report.
    - A count of how many words are negated is also
    calculated."""

def negation(tweets):
    neg_list =['never', 'no', 'nothing', 'nowhere', 'noone',
    'none', 'not', 'havent', 'hasnt', 'hadnt', 'cant',
    'couldnt', 'shouldnt', 'wont', 'wouldnt', 'dont',
    'doesnt', 'didnt', 'isnt', 'arent', 'aint']

    punct = [',','.',':',';','!','?']
    neg_count =0.0
    tweet=tweets.split()
    for i in range(len(tweet)):
        #print tweet[i]
        word = tweet[i]
        # if word == "no" or word == "not":
        if word in neg_list or "n't" in word:
            #print 'fired'
            for j in range(i,len(tweet)):
                #print j
                if tweet[j] in punct:
                    #print 'break'
                    break
                else:
                    tweet[j] += '_NEG'
                    neg_count+=1.0
    return tweet, neg_count

"""Count the POS tags
    - The number of each POS tag is counted from a tweet."""


def countPOS(pos):
    PUNCT = N = V = PN = P = A = O = R = NUM = C = MEN = 0
```

```
D = L = U = HASH = INT = DIS = VER = EMO = PRO = F = 0
EX = NOM = 0

for i in range(len(pos)):
    if pos[i] == ',':
        PUNCT += 1
    elif pos[i] == 'N':
        N += 1
    elif pos[i] == 'V':
        V += 1
    elif pos[i] == '^':
        PN += 1
    elif pos[i] == 'P':
        P += 1
    elif pos[i] == 'A':
        A += 1
    elif pos[i] == 'O':
        O += 1
    elif pos[i] == 'R':
        R += 1
    elif pos[i] == '$':
        NUM += 1
    elif pos[i] == '&':
        C += 1
    elif pos[i] == '@':
        MEN += 1
    elif pos[i] == 'D':
        D += 1
    elif pos[i] == 'L':
            L += 1
        elif pos[i] == ’U’:
            U += 1
        elif pos[i] == ’#’:
            HASH += 1
        elif pos[i] == ’!’:
            INT += 1
        elif pos[i] == ’~’:
            DIS += 1
        elif pos[i] == ’T’:
            VER += 1
        elif pos[i] == ’E’:
            EMO += 1
        elif pos[i] == ’Z’:
            PRO += 1
        elif pos[i] == ’G’:
            F += 1
        elif pos[i] == ’X’:
            EX += 1
        elif pos[i] == ’S’:
            NOM += 1

    count =[PUNCT, N, V, PN, P, A, O, R, NUM, C, MEN, D, L,
    U, HASH]

    return count


"""Feature extractors:
    -The following 3 functions are used to pr-process different
    length n-grams and build the feature list.
    -Tweets are passed to the functions and split into n-grams.
    -N-grams are processed if processing is required.
    Some lexicons require unprocessed n-grams, this is why the
    option 'process' is a parameter in the function.
    -If the n-gram is made of stop words, begins with a
    non-alphabetic character, then the word is not added to the
    feature list.
    -Adapted from http://ravikiranj.net/posts/2012/code
    /how-build-twitter-sentiment-analyzer/
    Accessed: 02/03/16"""


# start getfeatureVector
def get_unigrams(tweet, stopWords, process):
    unigrams = []
    #split tweet into words
    #   - if negation is not being used in the get_features
    #   function, then use 'words = tweet.split()'
    #   - It negation is used, use 'words = tweet'.

    #words = tweet.split()
    words = tweet

    for i in range(len(words)):
        if process == 1:
           words[i] = preprocess(words[i])
        val = re.search(r"^[a-zA-Z][a-zA-Z0-9]*$", words[i])
        # ignore if it is a stop word, not alphabetic, or only
        # processed not for use in lexicon
        if (words[i] in stopWords or val is None and process == 1):
            continue
        else:
            unigrams.append(words[i].lower())

    return unigrams


def get_bigrams(tweet, stopWords, process):
    tweetB = ngrams(tweet, 2)
    bigrams = []
    # split bigram into two and process each word
    for i in range(len(tweetB)):
        w = tweetB[i][0]
        if process == 1:
            w = preprocess(w)
        w = w.lower()
        tweetB[i][0] = w

        w2 = tweetB[i][1]
        if process == 1:
            w2 = preprocess(w2)
        w2 = w2.lower()
        tweetB[i][1] = w2

        # ignore if stop word and not for use in lexicon
        if tweetB[i][0] in stopWords and tweetB[i][1] in stopWords
        and process == 1:

            continue
        else:
            bigrams.append(' '.join(tweetB[i]))
    return bigrams


def get_trigrams(tweet, stopWords):
    tweetB = ngrams(tweet, 3)
    trigrams = []
    for i in range(len(tweetB)):
        w = tweetB[i][0]
        w = preprocess(w)
        tweetB[i][0] = w

        w2 = tweetB[i][1]
        w2 = preprocess(w2)
        tweetB[i][1] = w2

        w3 = tweetB[i][2]
        w3 = preprocess(w3)
        tweetB[i][2] = w3
        # ignore if stop word
        if tweetB[i][0] in stopWords and tweetB[i][1] in stopWords:
            continue
        elif tweetB[i][1] in stopWords and tweetB[i][2] in stopWords:
            continue
        else:
            trigrams.append(' '.join(tweetB[i]))
    return trigrams




"""Get stop words
    -Reads in stopwords from a given file name
    -Appends 'AT_USER and URL from preprocessing stage
    -From http://ravikiranj.net/posts/2012/code
    /how-build-twitter-sentiment-analyzer/
    Accessed: 02/03/16 """


def getStopWordList(stopWordListFileName):
    # read the stopwords file and build a list
    stopWords = []
    stopWords.append('AT_USER')
    stopWords.append('URL')

    fp = open(stopWordListFileName, 'r')
    line = fp.readline()
    while line:
        word = line.strip()
        stopWords.append(word)
        line = fp.readline()
    fp.close()
    return stopWords
```

```
"""read in data from file
    -used to read in training/testing data.
    -Index of tweet, token and label can be altered
    depending on format of input file
    -By default, the classifier only runs on two
    polarities. If three are required then replace
    'neutral' string in the last if with an empty
    string. i.e '' """


def read_data(filename):
    f = open(filename, 'r')
    data = []
    for i in f:
        if i:
            i = i.split('\t')
            tweet = i[0]
            token = i[1]
            label = i[2].strip('\n')
            if label != 'neutral':
                data.append([tweet, token, label])
    return data


"""Build feature list
    -Code uses earlier functions and extracts the different
    n-grams while building the feature list. See section 2.2.1
    of report for more information of implementation.
    -As default the trigrams are commented out as they cause a
    drop in performance, to use them, the 2 lines under
    'Extract trigrams' need to be uncommented and 'feature_list_tri'
    must be added to the last line of the for loop.
    -The if statement is used to seperate out the training
    and testing data. Only the trainingdata's feature list is used,
    not the testing.
    -Adapted from http://ravikiranj.net/posts/2012/code
    /how-build-twitter-sentiment-analyzer/
    Accessed: 02/03/16"""


def get_features(data, T):
    feature_list = []
    vector = []
    stopWords = getStopWordList('../data/other/stopwords.txt')
    for i in range(len(data)):
        sentiment = data[i][2]
        token = data[i][1]

        #Used to get negated version of tweet
        negated, neg_count = negation(data[i][0])

        # Extract uni-grams
        #   - If negation is used, use the first line
        #   - If negation is not used, use the second line
        #   - Please consult function 'get_unigrams' for more changes
        feature_list_uni = get_unigrams(negated, stopWords, 1)
        #feature_list_uni = get_unigrams(data[i][0], stopWords, 1)
        feature_list.extend(feature_list_uni)

        # Extract bigrams
        feature_list_bi = get_bigrams(data[i][0], stopWords, 1)
        feature_list.extend(feature_list_bi)

        # Extract trigrams
        #feature_list_tri = get_trigrams(data[i][0], stopWords)
        #feature_list.extend(feature_list_tri)

        # Extract unigrams and bigrams for use with certain lexicons
        unigrams = get_unigrams(data[i][0], stopWords, 0)
        bigrams =get_bigrams(data[i][0], stopWords, 0)
        n_grams = unigrams+bigrams
        # remember to add +feature vector 3 below
        vector.append((feature_list_uni + feature_list_bi, token,
            sentiment, n_grams, neg_count))
    if T == 1:
        return (feature_list, vector)
    else:
        return vector


"""Extract features:
    - Extracts the features from the text and stores them in a format
    ready for to be passed to the support vector machine. For more
    information on correct format for LibSVM, please consult the LibSVM
    README.txt.
    - Build the feature vectors as discussed in the report section 2.2.
    - Dictionary 3 is commented out by default as it reduces the performance.
    - Adapted from http://ravikiranj.net/posts/2012/code
    /how-build-twitter-sentiment-analyzer/
    Accessed: 02/03/16"""


def make_fv(tweets, featureList, dictionary1, dictionary2, dictionary3):
    sortedFeatures = sorted(featureList)

    feature_vector = []
    labels = []
    label=0
    counter = 0

    for t in tweets:

        # Initialize empty map to store all n-grams
        map = {}
        for w in sortedFeatures:
            map[w] = 0

        tweet_words = t[0]
        tweet_token = t[1]
        tweet_opinion = t[2]
        tweet_n_grams = t[3]
        tweet_neg_count=int(t[4])

        # Add 1 to map if an n-gram in tweet is in the map of n-grams
        for word in tweet_words:
```

```
    if word in map:
        map[word] = 1

values = map.values()

# Get the lexicon values
#Setiment140 lexicon
values.extend(scorelexicon(tweet_words, dictionary1,1))

#Hashtag lexicon
#   -Makes use of unprocessed unigrams and bigrams
values.extend(scorelexicon(tweet_n_grams, dictionary2,2))

#MPQA lexicon, commented out as it reduces the performance
#values.extend(scorelexicon(tweet_words, dictionary3,3))

# Count POS tags
values.extend(countPOS(tweet_token))

# Count Negation
values.extend([tweet_neg_count])

# Add all values to the feature vector
feature_vector.append(values)

#Add the label of the tweet to the labels
if tweet_opinion == 'negative':
    label = -1
    labels.append(label)
```

```
        elif tweet_opinion == 'positive':
            label = 1
            labels.append(label)
        elif tweet_opinion == 'neutral':
            label = 0
            labels.append(label)

    # return a dictionary of lists of feature_vector and labels
    return {'feature_vector': feature_vector, 'labels': labels}
```