1. INTRODUCTION

Wellbeing is a complex concept with no singular definition, however the Office for National Statistics (ONS) describes wellbeing as '"how we are doing', as individuals, as communities and as a nation, and how sustainable this is for the future" (Evans et al., 2015). The term 'wellbeing' will assume the description given by the ONS for the remainder of this project.

Whilst traditional economic measures are undoubtedly essential when evaluating a country's performance, national wellbeing is being used increasingly as an additional, wider measure to monitor progress (Shenton et al., 2012).

An important element of national wellbeing is subjective wellbeing. Subjective wellbeing, also known as personal wellbeing, is measured by asking individuals for their evaluation of their life. It provides unique information that is not obtainable from objective measures alone, and it enables a complete assessment to be made on the wellbeing of a person, and thus of an area (Shenton et al., 2012).

Both national and subjective wellbeing are measured in the UK by the ONS. The ONS is the 'UK's largest independent producer of official statistics and the recognised national statistical institute of the UK'. The ONS is responsible for data collection and analysis, and publication of statistics on a variety of domains, including the economy, population and society (ONS, [no date]).

This study will investigate the use of social media data to produce timelier and geographically localised estimates of subjective wellbeing, which can be compared against existing survey-based results.

1.1 Objectives

The objectives of this project are as follows:

- To understand more about wellbeing and research other existing methods for measuring subjective wellbeing.

- To identify the limitations of surveys and discover how social media data can be used to supplement them.
- To research existing work that has been done on this topic and understand the approaches taken.
- To establish an accurate method for analysing social media data.
- To analyse social media data and compare the outcome to existing survey-based results.
- To determine whether subjective wellbeing can be detected through social media.

1.2 Structure

The structure of this project is as follows:

- The *background* will discuss the importance of wellbeing.
- Well-known approaches to wellbeing and methods of measuring subjective wellbeing will be explored in the *literature review*, along with existing research on the use of social media data for predictions and psychological studies.
- The specific approach taken in this project will be described in the *methodology*.
- The results from the methodology will be presented in the *results* section.
- The methodology and results will be discussed and analysed in further detail, along with suggestions for future research in the *discussion*.
- The *conclusion* will provide a brief overview of the project and summarise the findings.

2. BACKGROUND

Gross Domestic Product (GDP), defined as 'the monetary value of all the finished goods and services produced within a country's borders in a specific time period' (Investopedia, [no date]), is arguably one of the most important economic statistics (BBC News, 2011). However invaluable it may be as a financial indicator, it fails to provide a broader perspective of the quality of life in the UK.

In 1972 Bhutan rejected GDP as the sole measure of the country's performance after the 4th King declared 'if the government cannot create happiness for its people, there is no purpose for the government to exist' (Ura et al., 2012). From then onwards, the country's progress was measured by the level of happiness among its citizens, determined by a Gross National Happiness (GNH) index.

Perhaps inspired by Bhutan and this alternative measure of performance, in 2010 David Cameron announced 'it's high time we admitted that, taken on its own, GDP is an incomplete way of measuring a country's progress', and he stated that he wanted to 'start measuring our progress as a country, not just by how our economy is growing, but by how our lives are improving; not just by our standard of living, but by our quality of life' (Cameron, 2010). He therefore requested that the ONS devised a method to measure national wellbeing.

Following this request, the ONS launched a Measuring National Well-being Programme in November 2010, with the aim of developing an accepted and trusted set of statistics that would enable wellbeing to be accurately evaluated in the UK (Shenton et al., 2012). However, before being able to monitor and improve wellbeing, the ONS had to first identify what contributes most to national wellbeing, and therefore what they should measure. They initiated a national debate on 'what matters to you' which took place from November 2010 to April 2011. The ONS set up a variety of methods for data collection to ensure they received as many responses as possible, including events, online surveys, a website, a phone line and pre-paid postcards. This debate generated a record number of responses for the ONS at the time. The five questions included in the debate were as follows:

1. What things in life matter to you?
2. Of the things that matter to you, which should be reflected in measures of national wellbeing?
3. Which of the following sets of information do you think help measure national wellbeing and how life in the UK is changing over time?
4. Which of the following ways would be best to give a picture of national wellbeing?
5. How would you use measures of national wellbeing?

From this national debate, the ONS were able to conclude that there is no singular definition of wellbeing, both national and subjective wellbeing are 'complex and multidimensional issues'. However the key findings across all demographics included the importance of family, friends, health, financial security, equality and fairness in determining wellbeing (Matheson, 2011).

The ONS believed that both subjective and objective elements of wellbeing must be measured in order to evaluate national wellbeing as a whole (Shenton et al., 2012). Therefore a survey was created based on the results of the national debate, encompassing questions across 10 domains, including objective measures such as health, what we do, where we live, personal finance, relationships, education and skills, the economy, governance and the natural environment; and subjective measures, including personal wellbeing (Beaumont, 2011).

The questions exist as part of the Annual Population Survey (APS). The APS is a 'continuous household survey, covering the UK', with a sample size of approximately 320,000 respondents. It has the largest coverage of any household survey in the UK and it provides information for small geographic locations. The information provided includes statistics on employment and unemployment, housing, ethnicity, religion, health and education (ONS, 2012). The specific questions asked on subjective wellbeing generate roughly 165,000 annual responses (ONS, 2015), and will be described in more detail in the literature review.

Social media data is used increasingly as an alternative to responses obtained from original survey-based methods. This project will focus specifically on the use of

Twitter[1] data. Twitter is a well-known social media site that allows users to post updates of no longer than 140 characters, otherwise known as 'tweets'. It is a popular method for users to share messages with others and is also known as a platform for 'microblogging' (Nations, 2017). With approximately 500 million tweets being posted daily (Aslam, 2017), Twitter provides a vast amount of open data which can be analysed. This data, if utilised correctly, has the ability to supplement or even replace traditional survey-based methods.

[1] https://twitter.com/?lang=en-gb

3. Literature Review

3.1 Wellbeing

Wellbeing is commonly misinterpreted as "happiness", which is of course an important contributing factor to a person's wellbeing. However wellbeing is a complex, multidimensional concept which encompasses a wide range of factors (Schneider, 2016). Listed below are some well-known approaches to wellbeing.

3.1.1 Eudaimonic vs Hedonic Wellbeing

The concept of wellbeing dates back over 2,000 years ago. Greek philosopher, Aristippus, coined a theory on "hedonic wellbeing". He believed that a person would experience higher levels of wellbeing if they focused their time predominantly on participating in fun, enjoyable activities, and avoided tedious, unenjoyable tasks (Ryan and Deci, 2001).

However this approach was criticised. It was argued that participating in activities that provide immediate enjoyment does not necessarily improve a person's wellbeing. In fact, participating in such activities could even have long-term harmful effects on wellbeing (i.e. drug or alcohol abuse). Greek philosopher, Aristotle, believed that hedonic wellbeing was "a vulgar idea", and he therefore created an alternative theory, "eudaimonic wellbeing" (Ryan and Deci, 2001). Eudaimonic wellbeing is based on the realisation of one's true potential, and the participation in tasks that are worthwhile (Ryan and Deci 2001; Waterman et al. 2008).

3.1.2 PERMA

Seligman (2011) designed a PERMA model for happiness, based on both hedonic and eudaimonic approaches. The model consists of five essential components that Seligman believes will collectively enable one to experience greater wellbeing.

- P - Positive emotion. A hedonic concept, whereby one experiences more positive and less negative emotion.

- E - Engagement. By being fully engaged in a task, an enjoyable sense of "flow" is experienced, which is known to boost wellbeing.
- R - Positive relationships. Humans thrive for social interaction and having positive relationships can greatly increase happiness.
- M - Meaning. A eudaimonic concept, participation in worthwhile activities creates greater meaning in one's life.
- A - Accomplishment/Achievement. Another eudaimonic notion, by achieving a specific goal in life a person is able to become a better version of themselves.

3.1.3 Hierarchy of Needs

Maslow (1943) created a wellbeing model consisting of five levels of human needs. Similar to PERMA, this model encompasses both hedonic and eudaimonic aspects of wellbeing. It is structured as a hierarchy to demonstrate that once a need has been satisfied, a new need emerges that requires satisfaction, however the new need can only be satisfied upon satisfaction of the previous need. At the base of the hierarchy is "psychological needs" which includes the most basic human needs, i.e., food, water, warmth, rest, etc. Once the psychological needs have been satisfied, one's motivation is then focused on satisfying the next tier, "safety needs", the need for security and safety. Above safety needs are the "love needs", the desire to experience affection and a sense of belonging. Following the love needs are the "esteem needs", the feeling of self-confidence and adequacy. At the top of the hierarchy is "self-actualisation", a purely eudaimonic need which refers to the desire for one to fulfil their full potential.

The first four needs can be described as "deficiency needs". These are desires that grow stronger, the longer they are neglected for. Once a deficiency need has been satisfied, it is no longer desired. For example, the longer a person is left without food, the hungrier they get, and once they then eat, they are no longer hungry. In contrast, self-actualisation is known as a "growth need", a need which, after it has been engaged, continues to be felt and may even grow to be stronger (McLeod, 2016).

Self-actualisation can only be reached once all other needs have been satisfied, however life experiences (such as divorce or job loss, for example) can cause a

disruption in lower levels which can hinder progress. It is said, therefore, that movement through the hierarchy is not always unidirectional, and fluctuations can occur (McLeod, 2016).

3.2 Measuring Wellbeing

Wellbeing can be measured either subjectively or objectively. Alatartseva and Barysheva (2015) describe objective wellbeing as 'external', influenced by factors that affect the quality of life such as health, income, etc. Whereas subjective wellbeing can be thought of as 'internal', determined by emotions and how a person evaluates their life.

Objective indicators can be measured over time and, whilst assumptions can be made based on objective indicators, they may not reflect a person's actual wellbeing. For example, someone that does not earn a good salary may still be highly satisfied with life (Smith and Clay, 2010).

For many years, measuring objective wellbeing was the primary focus, due mainly to the fact that objective indicators are easier to observe and measure than subjective indicators. However, many objective indicators act as proxies for what really matters (Waldron, 2010). Waldron (2010) states that "although objective measures are crucial, they cannot tell the whole story and the array of indicators currently available sometimes complicate the picture". A number of scales have been devised that attempt to measure subjective wellbeing which will be outlined below.

3.2.1 Subjective Wellbeing Scales

Diener et al. (2003) defines subjective wellbeing as "people's emotional and cognitive evaluations of their lives". Subjective wellbeing is a purely hedonic concept which can be divided into three components, life satisfaction, positive affect and negative affect (Albuquerque, 2010). Positive affect refers to the presence of positive emotions and negative affect, the presence of negative emotions, while life satisfaction can be defined as "one's evaluation of satisfaction with life in general" (Singh and Jha, 2008).

Ultimately, subjective wellbeing is the measure of how a person feels about their life, and a number of scales have devised to measure subjective wellbeing based on these three components.

3.2.1.1 Affect Balance Scale (ABS)

Bradburn (1969) created the 'Affect Balance Scale (ABS)' which measures positive and negative affect to assess one's subjective wellbeing. The scale consists of 10 questions, 5 positive and 5 negative:

During the past few weeks did you ever feel…

Positive feelings:

- Pleased about having accomplished something?
- That things were going your way?
- Proud because someone complimented you on something you had done?
- Particularly excited or interested in something?
- On top of the world?

Negative feelings:

- So restless that you couldn't sit long in a chair?
- Bored?
- Depressed or very unhappy?
- Very lonely or remote from other people?
- Upset because someone criticised you?

Both positive and negative affect are measured by adding 1 for every "yes" answer in the positive and negative questions respectively. Affect balance is then calculated by subtracting the negative affect score from the positive affect score.

3.2.1.2 Satisfaction With Life Scale (SWLS)

Diener et al. (1985) created the 'Satisfaction with Life Scale (SWLS)' which measures how satisfied an individual is with life from their perspective.

The SWLS consists of the following five statements:

1. In most ways, my life is close to ideal.
2. The conditions of my life are excellent.

3. I am satisfied with my life.
4. So far I have gotten all the important things I want in life.
5. If I could live my life over, I'd change almost nothing.

The respondents are asked to rank the extent to which they agree with each statement between 1-7, where 1 means strongly disagree, and 7 means strongly agree. The scores from each statement are then added together to create an overall score out of 35 where lower scores indicate that the person is extremely dissatisfied with life and higher scores indicate that the person is extremely satisfied with life.

3.2.1.3 Subjective Happiness Scale (SHS)

Lyubomirsky and Lepper (1999) created a 'Subjective Happiness Scale (SHS)' in an attempt to measure a person's overall subjective wellbeing, rather just than one component of subjective wellbeing (such as the ABS and SWLS). They wanted to create a measure that is 'a global, subjective assessment of whether one is a happy person or not'. The SHS consists of four statements, and the respondent is asked to indicate how much they agree with each statement.

The four statements are as follows:

1. In general, I consider myself:
   (1 - not a very happy person … 7 - a very happy person)
2. Compared to most of my peers, I consider myself:
   (1 - less happy … 7 - more happy)
3. Some people are generally very happy. They enjoy life regardless of what's going on, getting the most out of everything. To what extent does this characterisation describe you?
   (1 - not at all … 7 - a great deal)
4. Some people are generally not very happy. Although they are not depressed, they never seem as happy as they might be. To what extent does this characterisation describe you?
   (1 - not at all … 7 - a great deal)

An individual's score is calculated by reversing the result of question 4, and taking the mean of all four answers.

3.2.1.4 Annual Population Survey (APS)

The ONS measures subjective wellbeing by asking four questions on the Annual Population Survey (APS). The survey includes hedonic questions, based on affect (Q3 and Q4), an evaluative question which asks about life satisfaction (Q1), and, unlike other existing surveys, a eudaimonic question (Q2) (ONS, 2011). The questions are:

1. Overall, how satisfied with you are with your life nowadays?
2. Overall, to what extent do you feel the things you do in your life are worthwhile?
3. Overall, how happy did you feel yesterday?
4. Overall, how anxious did you feel yesterday?

The respondent is asked to give their answer on a scale of 0 to 10, where 0 is 'not at all' and 10 is 'completely' and average scores are formed for each question (ONS, 2017).

3.3 Alternative Methods to Surveys

Self-reported questionnaires are a commonly used tool, particularly in psychological studies. They are beneficial as they can gather information that is otherwise unobtainable, i.e. how someone actually feels, they are cheap, and they can be scaled to large samples. However there are also limitations concerning self-reported questionnaires (Hoskin, 2012):

- The respondents are not always honest with their answers, and may even be unwilling to answer certain questions if it is asking for personal information (i.e. political views). There is no way of knowing whether or not an answer is honest.
- Answers are subject to response bias, the respondent may answer with what they think is 'right' or 'what the interviewer wants to hear', regardless of what they actually think.
- Respondents may misinterpret a question if it is not written clearly.
- Even if the question is written clearly, more abstract questions where you are asked to 'rank' certain things can be interpreted differently. Yes/no questions tend to work more effectively, however they are restrictive.

- Surveys can be prone to sample bias, where the sample being surveyed is not a true representation of the population.

Open data sources such as social media offer an alternative approach to more traditional survey-based methods. The benefits of using such data include that it is plentiful, free and very fast to obtain, and it can measure a 'variety of dimensions of the public mood state' (Bollen et al., 2011). The analysis of social media data has unveiled a variety of interesting and abstract relationships which will be discussed below.

3.3.1 Sentiment Analysis

Sentiment analysis can be defined as "the process of determining the emotional tone behind a series of words, used to gain an understanding of the attitudes, opinions and emotions expressed within an online mention" (Bannister, 2015).

There are two main approaches for sentiment analysis, a lexicon-based approach and a machine learning approach. A lexicon-based approach relies on a lexicon (a vocabulary of words and phrases) to classify a document (Taboada et al., 2011). A machine learning approach involves training a model to classify a document. Machine learning can be further separated into two types, supervised machine learning and unsupervised machine learning. Supervised machine learning involves training models using a pre-classified training dataset, whilst unsupervised machine learning does not rely on a training dataset that is pre-classified. It instead works by finding structures within the training dataset (Brownlee, 2013).

3.3.1.1 Predicting Stock Markets from Social Media

Bollen et al. (2011) explored the relationship between emotions in twitter statuses and large scale social and economic events in the U.S. in 2008. The tweets were processed and analysed before being assigned to one of six mood indicators determined by the 'Profile Of Mood States (POMS)' (these six indicators include tension, depression, anger, vigour, fatigue and confusion). During the period of time being observed there

were multiple big socio-economic events, which were reflected by large fluctuations in public mood on Twitter. Bollen et al. were able to conclude that macroscopic socio-cultural events did affect assorted dimensions of the public's mood, which was assessable using sentiment analysis.

Bollen et al. (2011) later analysed tweets in order to establish a relationship between public mood and the stock market. They hypothesised that although stock market prices primarily depended on new information, such as the news (which is impossible to predict), human decision-making (particularly financial decisions), is driven by mood and emotions. By re-employing the POMS method from their earlier studies, they were able to conclude that shifts in certain moods, in particular 'calmness' of the public, were followed by shifts in the stock market 3-4 days later, therefore the calmness of the public is predictive of the stock market.

Mittal and Goel (2012) built on the work of Bollen et al. (2011) by analysing tweets and assigning them to one of four categories (calm, happy, alert and kind). Their results proved that both calmness and happiness of the public were good predictors of the stock market.

3.3.1.2 Predicting Presidential Elections from Social Media

In 2011, Choy et al. (2011) attempted to predict the result of the Singapore presidential election by analysing tweets. They performed sentiment analysis on the Twitter data using a customised corpus (collection of texts), and were able to successfully predict the narrow margin between the top two (of four) candidates, however they were unsuccessful in predicting the correct candidate.

In 2012, Choy et al. (2012) re-attempted to predict a presidential election, this time in the U.S. By assessing and altering the methods used in their previous attempt, they were able to successfully predict Obama winning the election.

MogAI is a software that predicts election results using Artificial Intelligence. It analyses social media sites such as Twitter, and takes into account location of tweets,

as well as just the context, in order to predict elections. It has correctly predicted every US presidential election since 2004, including the most recent election where almost all predictions suggested that Hillary Clinton was certain to win. MogAI correctly predicted that Donald Trump would win, and by quite a margin (Skula, 2017 and BBC, 2016).

3.3.1.3 Predicting Box-office Revenue from Social Media

Asur and Bernardo (2010) used sentiment analysis in an attempt to predict box-office revenues from Twitter data. They hypothesised that "movies that are well talked about will be well-watched", which they were able to confirm by showing that the more tweets per hour there were about a film, the more revenue the film made after its release, thus they created a predictive model using the rate of tweets (tweet-rate). Using sentiment analysis it was discovered that the sentiment of a movie was higher after its release (i.e. more positive or negative tweets than neutral tweets). They were able to track the effect of an increase in sentiment on the revenue of the movie, thus they used sentiment obtained from the Twitter feed after the release as an additional element in the predictive model. Whilst of the two measurements (tweet-rate and sentiment), tweet-rate was shown to be a better predictor of revenue, the addition of sentiment provided a small improvement to the model, and the accuracy of their predictions outperformed the those of the Hollywood Stock Exchange.

Liu et al. (2016) disagreed with the main hypothesis of the work of Asur and Huberman, arguing 'regardless of how much of the movie is discussed and how perfect the review is rated, the factor that is most related to the movie box-office revenue is how many people are willing to see the movie'. Using similar methods to Asur and Bernardo, they analysed the tweets to determine the 'purchase intention of users for movies from social media', i.e., the number of people on social media who actually expressed interest in wanting to see the movie, rather than just mentioning it. A strong correlation between purchase intention and movie box-office revenues was discovered, and this alteration resulted in significant improvements of the performance of the prediction model.

3.3.1.4 Predicting Personality and Wellbeing from Social Media

As well as sentiment analysis being used to make box office or stock market predictions from social media data, there have been studies demonstrating the use of social media to analyse and predict emotions, personality traits and wellbeing of users.

Mohammad and Kiritchenko (2015) investigated whether they were able to determine fine emotions and personality traits through social media. They created a large corpus of tweets which enabled them to produce a lexicon (vocabulary) of over 15,000 entries where each entry is a word-emotion pair from 585 emotions. This lexicon then was used by a classifier to detect personality from written text using two datasets, Facebook statuses and essays, where the writer had taken a personality test that they used for comparison. They were able to prove that the lexicon containing fine-grained emotion words was very useful in detecting personality.

Liu et al. (2015) explored the relationship between Facebook status updates and subjective wellbeing, specifically the user's self-reported satisfaction with life. By using the 'MyPersonality' Facebook application they were able to access specific people's Facebook data and their corresponding results to the SWLS. LIWC (linguistic inquiry and word count) was used to analyse all Facebook statuses of each user for twelve months prior to them taking the survey. Liu et al. concluded that life satisfaction correlated negatively with negative emotion in the most recent nine months, therefore negative emotional experiences within the past nine months were related to SWB.

Similarly, Schwartz et al. (2016) attempted to predict satisfaction with life by assessing Facebook statuses. A predictive model was built based on both PERMA and satisfaction with life (SWL) scales. They designed two models, a 'message-level model' and a 'user-level model'. The message-level model was created using Amazon's Mechanical Turk to analyse Facebook statuses according to positive and negative aspects of PERMA, and the SWLS, whilst the user-level model was created using the Facebook 'MyPersonality' application to gain user's Facebook statuses and results to the SWLS (note that the Facebook statuses used in each model were

separate). Both models were then used to create a ‘cascaded model’ where message predictions from the message-level model were passed to the user-level model and a mean prediction was made. Their model significantly outperformed other existing predictive models.

Yang and Srinivasan (2016) also analysed Twitter data to determine levels of life satisfaction. A different approach was taken to that of Schwartz et al., by focusing predominantly on user’s expressing life satisfaction or life dissatisfaction in their tweets, rather than relying on results from a survey. They believed that individuals whose tweets are analysed are different than those who respond to the surveys. They created a robust classification method based on the satisfaction with life survey (SWLS) which they tested by selecting 120 ‘satisfied’ tweeters and 120 ‘dissatisfied’ tweeters daily for just under 5 months, and inviting them to take the satisfaction with life survey within 24 hours of their post. The results of their test showed that life satisfaction between the two groups of people was significantly different. Using this classifier, they analysed the sentiment of tweets and classified them as satisfied or dissatisfied, then they took a subset of 120 satisfied and 120 dissatisfied tweets and retrieved all tweets that that user had posted, which resulted in a set of 14,506 satisfied tweets and 14,743 dissatisfied tweets. They were able to analyse the differences between the language used in both sets of tweets and the differences between psychological process categories, as defined by LIWC.

4. Methodology

The Office for National Statistics (ONS) provided a Twitter dataset which consisted of approximately 81 million geo-tagged tweets. The tweets were from the UK during April 2014 to October 2014. A subsection of the tweets were analysed and classified as 'positive' or 'negative' using sentiment analysis. The outcome was then compared to the results from the wellbeing survey in 2014.

Within the literature, supervised machine learning is a commonly used choice of sentiment analysis, it was therefore adopted as the technique for this project. However, before analysing the 2014 Twitter dataset, an accurate method had to be identified. Using a pre-classified Twitter dataset (Naji, 2012) and methods demonstrated by Sentdex (2015), eight different classifiers were assessed and the most accurate classifiers were selected to form a combined classifier. Each classifier was trained, tested and evaluated using a variety of different pre-processing conditions in an attempt to determine which conditions yield the most accurate results.

VADER (2014) is a lexicon and rule-based method for sentiment analysis, designed for use specifically with social media data. It has been proven that VADER outperforms other well-established sentiment analysis lexicons (Hutto and Gilbert, 2014), and so it was used as a benchmark throughout the project, where the aim was to create a model that was at least as accurate as VADER.

All methodology and analysis was performed using Python[2] (version 3.6.2). Python was chosen as it is a logical, flexible programming language with a vast amount of libraries and tools available.

[2] https://www.python.org/

4.1 Classifiers and Pre-Processing Conditions

4.1.1 Classifiers

Supervised machine learning involves training a model on a pre-classified dataset, which thereafter enables the model to classify unseen objects. Eight different classifiers were trained and evaluated: Naive Bayes, Multinomial Naive Bayes, Bernoulli Naive Bayes, Logistic Regression, Support Vector Classifier, Nu Support Vector Classifier, Linear Support Vector Classifier and Stochastic Gradient Descent Classifier. Each one is described briefly below.

4.1.1.1 Naïve Bayes, Multinomial Naive Bayes and Bernoulli Naive Bayes Classifiers

Naïve Bayes is a probabilistic classifier based on Bayes Theorem:

$$P(c|d) = \frac{P(d|c)*P(c)}{P(d)}$$

Where P(c|d) represents the posterior probability of class 'c' given document 'd', P(c) represents the prior probability of class 'c', P(d|c) represents the likelihood which is the probability of document 'd' given class 'c' and P(d) is the prior probability of document 'd' (Sayad, 2017).

Naïve Bayes assumes that the features used in the classifications are conditionally independent. It is a simple, effective classifier that is commonly used in text classification, particularly for spam email detection, language recognition and sentiment analysis (Vryniotis, 2013).

Text classification aims to find the best class for the document, which in the case of Naïve Bayes is the most likely or the "maximum a posteriori (MAP)" class (Manning et al. 2008):

$$c_{map} = \text{argmax}_{c \text{ in } C} (P(c|d))$$

Applying Bayes Theorem:

$$c_{map} = \text{argmax}_{c \text{ in } C} \left(\frac{P(d|c)*P(c)}{P(d)}\right)$$

Which can be rewritten as:

$$c_{map} = \text{argmax}_{c \text{ in } C} (P(d|c)*P(c))$$

The denominator is dropped since P(d) is the same for all classes, therefore it does not affect the argmax (Manning et al. 2008). '$\text{argmax}_{c \text{ in } C}$' simply means for each class 'c' in a set of classes 'C', select the one that produces the highest value (Schmidt, 2016).

Naïve Bayes employs a commonly used 'bag of words' method whereby text from the training document is split into individual words. A vocabulary is formed which consists of the words that appear over a certain amount of times (Raschka, 2014). The vocabulary is required for training the classifier.

Therefore, for a document 'd' that contains n words, which can be expressed as w_1, w_2, …, w_n, the above equation can be rewritten as:

$$c_{map} = \text{argmax}_{c \text{ in } C} (P(w_1, w_2, \ldots, w_n|c)*P(c))$$

$$= \text{argmax}_{c \text{ in } C} (P(c) \prod_{i=1}^{n} P(wi|c))$$

Where P(c) is the prior probability of class 'c' and $P(w_i|c)$ is the conditional probability of word 'w_i' in the document given class 'c' (Vryniotis, 2013). $P(w_i|c)$ can also be interpreted as a measure of how much evidence 'w_i' contributes that 'c' is the correct class (Manning et al. 2008).

To summarise, the Naïve Bayes classifier works by computing the product of the probability of each word belonging to a particular class 'c', and the prior probability of class 'c', for all classes 'c' in 'C'. It then chooses the class that yields the highest probability (Vryniotis, 2013).

Multinomial Naïve Bayes is a variation of the Naïve Bayes classifier that estimates the conditional probability of a word 'w' given a class 'c' as the relative frequency of the word in a document that belongs to a class, i.e. how often the word appears in a given document that belongs to a class 'c' (Vryniotis, 2013). Multinomial Naïve Bayes classifiers replace P(d|c) with $P(w_1, w_2, \ldots, w_n|c)$ where $w_1, w_2, \ldots, w_n$ is the sequence of words as they occur in document 'd', including duplications and excluding the words that are not listed in the vocabulary (Manning et al. 2008).

Bernoulli Naïve Bayes is another variation of the Naïve Bayes classifier. In contrast with the Multinomial Naïve Bayes classifier, which relies on the number of times a word appears in a given class, Bernoulli Naive Bayes examines whether or not words from the vocabulary exist in the current document, by accounting for both occurrences *and* non-occurrences of words (Vryniotis, 2013). Bernoulli Naïve Bayes classifiers replace P(d|c) with $P(e_1, e_2, \ldots, e_n|c)$, where $e_1, e_2, \ldots, e_n$ is a list of words from the vocabulary with a binary vector of 1 or 0 indicating whether or not that word is in the current document (Manning et al. 2008).

4.1.1.2 Logistic Regression Classifier, Gradient Descent and Stochastic Gradient Descent

Simple linear regression predicts an output value (y) based on an input value (x) using two coefficients b_0 and b_1, i.e.:

$$y = b_0 + b_1 * x$$

In linear regression, b_0 is known as the intercept on the y axis, or in machine learning the 'bias', which offsets the result by a certain amount. b_1 is the slope and it determines how x translates to y before the bias is added (Brownlee, 2016).

Logistic regression, also known as maximum entropy, operates under the same principles as linear regression. It uses the same basic formula, except it produces binary, rather than continuous, output values (Brownlee, 2016). The above equation is modified to ensure that the output value is between, and including, 0 and 1. To do this, an exponential function is applied, which will always produce an output value that is greater than or equal to 0:

$$y = \exp(b_0 + b_1 * x)$$

To ensure the output value will be less than or equal to 1, the function is divided by a value that is slightly bigger than the value of itself:

$$y = \frac{\exp(b_0 + b_1 * x)}{\exp(b_0 + b_1 * x) + 1}$$

The coefficients $b_0, \ldots, b_n$ have to be learned from the data using either gradient descent or stochastic gradient descent (Brownlee, 2016).

Gradient descent is an optimisation algorithm that finds coefficients of a function which minimise the 'cost'. The process starts by using initial values for the coefficients that are either equal to 0, or small random values. The 'cost' of the coefficients is derived by calculating y - Y where y is the actual value and Y is the predicted value, obtained by using the initial values for the coefficients. 'Delta' is then obtained by calculating the derivative of the 'cost'. By taking the derivative, the slope of the function at that point is discovered, which gives the direction (sign) that the function needs to move in order to update the coefficients, and gain a lower 'cost' value in the next iteration (Brownlee, 2016). The coefficients are updated by calculating:

$$b_i = b_i - (alpha*delta)$$

For i = 1, …, n. 'Alpha' is known as the learning rate, which varies how fast or how slow the coefficients converge towards the optimal coefficients. The process is repeated until either an acceptable 'cost' is achieved, or a certain number of iterations, has been reached.

Stochastic Gradient Descent (SGD) is a variation of the gradient descent algorithm. Gradient descent can be a lengthy process when training large datasets. Instead of updating the coefficients iteratively, which can be time consuming particularly when training large datasets, all coefficients are updated based on either one item, or a few items in the training dataset (Brownlee, 2016).

4.1.1.3 Support Vector Classifier, Linear Support Vector Classifier, Nu Support Vector Classifier and Stochastic Gradient Descent Classifier

Support vector classifiers are also known as support vector machines. A support vector machine can be defined as "a large-margin classifier: it is a vector space based machine learning method where the goal is to find a decision boundary between two classes that is maximally far from any point in the training data" (Manning et al. 2008). In other words, a support vector machine aims to find a hyperplane that can separate two distinct classes.

The distance between the hyperplane and the closest point is called the 'margin' of the classifier, and the data points it touches are known as the 'support vectors'. When the

data consists of two linearly separable classes, there are many possible hyperplanes that can separate the data. Support vector machines choose the hyperplane that can move the largest distance either side before touching a data point in that class. This is also known as the ‘maximum-margin hyperplane’ (Manning et al. 2008).

By having the biggest margin possible, there are less data points close to the decision classifier. If a data point is close to the classifier, it has a higher chance of being selected for the wrong class (Manning et al. 2008). Therefore a classifier with a large margin will make more accurate classifications.

If the data is not linearly separable, which is often the case with text classification problems, a ‘kernel’ can be applied. A kernel is used to map the data to a higher dimensional space, thus enabling it to be linearly separated by a classifier (Manning et al. 2008).

Linear support vector classifier, support vector classifier and nu support vector classifier (LinearSVC, SVC and NuSVC) are variations of support vector machines in Python’s ‘sklearn’ (2011) library. Both SVC and NuSVC use a ‘radial basis function’ kernel, however NuSVC has a parameter, ‘nu’, that controls the number of support vectors, therefore NuSVC produces very different results to SVC. Conversely LinearSVC uses a ‘linear’ kernel. The stochastic gradient descent classifier (SGDClassifier) implements a linear support vector machine using stochastic gradient descent.

4.1.2 Pre-Processing Conditions

The eight classifiers described above were trained and tested on five different subsets of the pre-classified dataset. Furthermore, the eight classifiers were trained using a variety of pre-processing conditions, in particular, unigrams and bigrams, both with and without part of speech tagging. These conditions will be described in more detail below.

4.1.2.1 Unigrams and Bigrams

'n-grams' are all combinations of consecutive words of length n within an item of text. Experiments were carried out on models that were trained using unigrams (1-grams) and bigrams (2-grams). For example, given the text:

"I am writing my dissertation"

Separating this into unigrams yields:

"I" "am" "writing" "my" "dissertation"

Separating this into bigrams yields:

"I am" "am writing" "writing my" "my dissertation".

4.1.2.2 Part of Speech Tagging

This is a method whereby, after a sentence has been separated into either unigrams or bigrams, only certain items are allowed to be selected for the vocabulary. The classifiers were trained using vocabulary that was formed from unigrams and bigrams *without* part of speech tagging, i.e. all words and punctuation were allowed to be chosen for the vocabulary, and *with* part of speech tagging, i.e. only certain types of words were allowed to be chosen for the vocabulary. The 'allowed word types' for this project were verbs, adverbs and adjectives.

4.2 Training the Classifiers

Each classifier was trained and tested, using methods demonstrated by Sentdex (2015), on five different subsets of the pre-classified dataset. Each subset of the dataset was used to test all classifiers, under each pre-processing condition before being reselected. This ensured that the comparisons between models were fair and meaningful.

4.2.1 Processing the Pre-Classified Dataset

A pre-classified Twitter dataset (Naji, 2012) was used to train and test each classifier. The dataset consisted of approximately 1.6 million tweets, each with a corresponding sentiment score of positive or negative. It is important to note that the dataset contained

roughly twice as many positive than negative tweets, so extra care had to be taken to ensure that the datasets selected to train the classifiers contained an even number of positive and negative tweets.

Before the classifiers could be trained, the dataset first had to be converted into an appropriate format. The original dataset was a csv file consisting of four columns: item id, sentiment (0 or 1 where 0 represents negative and 1 represents positive), sentiment source and sentiment text. In Python, classifiers require training datasets to be in the form of a list of tuples. These tuples must contain the text first, followed by the sentiment of the text.

The code for formatting the data is as follows:

```
1 import csv
2 import random
3 import pickle
```

Built-in Python modules ‘csv’, ‘random’ and ‘pickle’ were imported. The ‘csv’ module enables interaction between Python and csv files, which was the format in which the pre-classified dataset was saved. Python’s ‘random’ module has many features, the particular feature of interest was that it allows lists to be randomly shuffled. Python’s ‘pickle’ module converts objects in python to byte-streams and back, which essentially allows objects created in python to be saved and reused.

```
4 all_data = []
5
6 with open('Sentiment Analysis Dataset.csv') as f:
7    read = csv.reader(f, delimiter=',')
8    for row in read:
9         newrow = (row[3], row[1].replace('0', 'neg').replace('1', 'pos'))
10        all_data.append(newrow)
11 all_data.pop(0)
12
13 all_data_save = open('all_data.pickle', 'wb')
14 pickle.dump(all_data, all_data_save)
15 all_data_save.close()
```

The dataset (Sentiment Analysis Dataset.csv) was accessed and for each row in the csv file, the text (row[3]) and the sentiment score (row[1]) were selected and arranged into a tuple. The ‘.replace’ function replaced ‘0’ with ‘neg’ and ‘1’ with ‘pos’ in order to avoid any risk of confusion. Each tuple was then added to the list ‘all_data’. ‘all_data.pop(0)’ removed the first item from the list, which contained the column

headings. The list of all data was saved using pickle, so it could be easily accessed and reused.

```
positive = []
negative = []

for item in all_data:
   if item[1] == "pos":
       positive.append(item)
   if item[1] == "neg":
       negative.append(item)
```

The entire dataset was then divided into two separate lists of positive and negative tweets. By separating the dataset, it was easy to ensure that the number of positive and negative tweets in the training dataset was even.

```
random.shuffle(positive)
random.shuffle(negative)

random_pos = positive[:6000]
random_neg = negative[:6000]

train_data = []
test_data = []

for item in random_pos[:5000]:
   if item not in train_data:
       train_data.append(item)
for item in random_neg[:5000]:
   if item not in train_data:
       train_data.append(item)

for item in random_pos[5000:]:
   if item not in train_data:
       if item not in test_data:
           test_data.append(item)
for item in random_neg[5000:]:
   if item not in train_data:
       if item not in test_data:
           test_data.append(item)

train_data_save = open("train_data.pickle", "wb")
pickle.dump(train_data, train_data_save)
train_data_save.close()

test_data_save = open("test_data.pickle", "wb")
pickle.dump(test_data, test_data_save)
test_data_save.close()
```

The lists of positive and negative tweets were shuffled using Python's 'random' module, and the first 6,000 items in both lists were selected. From those 6,000 items, the first 5,000 were added to a list for training the classifiers (called 'train_data'), and the remaining 1,000 were added to a list for testing the classifiers (called 'test_data'). The 'if statements' ensured that there were no duplicates in either set, which could have potentially altered the accuracy of the classifiers.

This resulted in 10,000 items for training the classifiers and 2,000 items for testing the classifiers (providing no items were removed due to duplication). The training and test sets were then saved using pickle, so that they could be used later for the classifiers.

The datasets created were used for training and testing the classifiers (under each pre-processing condition) before being re-selected. This process was repeated a total of five times so that a different subset of training and test data was selected each time. The objective was to determine which pre-processing conditions would yield results with the highest accuracy for the classifiers, and which of the classifiers performed best.

4.2.2 Training the Classifiers using Unigrams without Part of Speech Tagging

The code for training the classifiers using unigrams, without part of speech tagging, is as follows:

```
1 import pickle
2 import nltk
3 import random
4
5 from nltk.classify.scikitlearn import SklearnClassifier
6 from nltk.tokenize import word_tokenize
7
8 from sklearn.naive_bayes import MultinomialNB, BernoulliNB
9 from sklearn.linear_model import LogisticRegression, SGDClassifier
10 from sklearn.svm import SVC, LinearSVC, NuSVC
```

Python's built-in modules 'pickle' and 'random' were again imported, along with the natural language toolkit module, 'nltk' (Bird et al., 2009), and various elements from the sci-kit learn module, 'sklearn' (Pedregosa et al., 2011). 'nltk' contains tools for classification, part-of-speech tagging and separating text into unigrams and bigrams, and 'sklearn' encompasses a number of classification algorithms.

```
11 test_data_f = open('test_data.pickle', 'rb')
12 test_data = pickle.load(test_data_f)
13 test_data_f.close()
14
15 train_data_f = open('train_data.pickle', 'rb')
16 train_data = pickle.load(train_data_f)
17 train_data_f.close()
```

The training and testing datasets that were previously created were ‘unpickled’, i.e. retrieved.

```
18 all_words = []
19
20 for item in train_data:
21    words = word_tokenize(item[0])
22    for word in words:
23        all_words.append(word.lower())
```

The ‘word_tokenize’ function from the nltk module was used to separate each item of text from the training dataset into unigrams (recall that the text is the first item in the tuple, ‘item[0]’). The unigrams were then added, in lower case, to a list of all words (named ‘all_words’). Lower case was consistently used so that words could be recognised by the classifiers regardless of capital letters.

```
24 all_words = nltk.FreqDist(all_words)
25
26 vocab = []
27 words_for_vocab = all_words.most_common()
28 for item in words_for_vocab:
29    if item[1] >= 2:
30        vocab.append(item[0].lower())
```

‘FreqDist’, a tool from the ‘nltk’ module, created a dictionary with each word from the list of all words (‘all_words’) as the key, and the total number of times that word occurred in the list as the value. A vocabulary was then constructed, which consisted of every item that appeared at least twice in the list of all words, i.e. every word in the dictionary that had a value of at least two.

```
31 def dictionary_of_words(text):
32    words = word_tokenize(text.lower())
33    Dictionary = {}
34    for word in vocab:
35        Dictionary[word] = (word in words)
36    return Dictionary
```

A function was created that would take a piece of text and create a dictionary with each item in the vocabulary as the key, and a Boolean expression (True or False), depending on whether that word existed in the given text as the value. This function was used to find items from the vocabulary, in the positive and negative training documents. It was also necessary for use later on with text from the testing document, and unseen text, to search for items from the vocabulary that existed in the text.

```
test_set = [(dictionary_of_words(item), sentiment) for (item, sentiment)
in test_data]

train_set = [(dictionary_of_words(item), sentiment) for (item, sentiment)
in train_data]
```

The ‘dictionary_of_words’ function was then applied to the text (the first item of every tuple) for each item in the training and testing datasets. The first item was therefore converted from a sentence, to a dictionary, and the second item remained as the sentiment for the text. Two new lists were created containing these tuples (called ‘test_set’ and ‘train_set’). These new lists were in the correct format to be used by the classifiers.

```
### CLASSIFIERS

# Original Naive Bayes
NB_Classifier = nltk.NaiveBayesClassifier.train(train_set) # Train set
print("Original Naive Bayes Classifier accuracy:
",  (nltk.classify.accuracy(classifier, test_set))*100, "%") # Test set

# Multinomial Naive Bayes
MNB_Classifier = SklearnClassifier(MultinomialNB())
MNB_Classifier.train(train_set)
print("Multinomial Naive Bayes Classifier accuracy:
",  (nltk.classify.accuracy(MNB_Classifier, test_set))*100, "%")

# Bernoulli Naive Bayes
BNB_Classifier = SklearnClassifier(BernoulliNB())
BNB_Classifier.train(train_set)
print("Bernoulli Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(BNB_Classifier, test_set))*100, "%")

# Logistic Regression
LR_Classifier = SklearnClassifier(LogisticRegression())
LR_Classifier.train(train_set)
print("Logistic Regression Classifier accuracy:
",  (nltk.classify.accuracy(LR_Classifier, test_set))*100, "%")

# Support Vector Classifier
SV_Classifier = SklearnClassifier(SVC())
SV_Classifier.train(train_set)
print("Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(SV_Classifier, test_set))*100, "%")

# Nu Support Vector Classifier
NuSV_Classifier = SklearnClassifier(NuSVC())
NuSV_Classifier.train(train_set)
print("Nu Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(NuSV_Classifier, test_set))*100, "%")

# Linear Support Vector Classifier
LSV_Classifier = SklearnClassifier(LinearSVC())
LSV_Classifier.train(train_set)
print("Linear Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(LSV_Classifier, test_set))*100, "%")

# Stochastic Gradient Descent Classifier
SGD_Classifier = SklearnClassifier(SGDClassifier())
SGD_Classifier.train(train_set)
print("Stochastic Gradient Descent Classifier accuracy: ",
(nltk.classify.accuracy(SGD_Classifier, test_set))*100, "%")
```

The remaining code trained and tested each of the eight classifiers. The classifiers used were elements from the 'sklearn' and 'nltk' modules that were imported in lines 1-10. Each classifier was trained and tested using the lists created in lines 37-39. The classifier is tested by attempting to classify items from the set of testing data, after it has been trained on the set of training data. After training and testing was complete for each classifier, the accuracy was printed. The accuracy was calculated as the percentage of items from the testing dataset that were correctly labelled by the classifier.

4.2.3 Training the Classifiers using Unigrams with Part of Speech Tagging

Similarly to the previous code, these classifiers were trained using unigrams. However part of speech tagging was applied, therefore after the sentence was separated into unigrams, not all items were valid for use in the vocabulary, only certain word types (verbs, adverbs and adjectives). To implement this, lines 18-23 of the code shown in section 4.2.2 was replaced with:

```
all_words = []
allowed_word_types = ["J", "R", "V"] # adjective, adverb, verb

for item in train_data:
   words = word_tokenize(item[0])
   part_of_speech = nltk.pos_tag(words)
   for word in part_of_speech:
       if len(word[0]) > 3:
           if word[1][0] in allowed_word_types:
               all_words.append(word[0].lower())
```

The 'pos_tag' function, a tool from the 'nltk' module was applied to each unigram, after the text had been separated. This function created a tuple consisting of the item followed by the 'part of speech tag', i.e. the 'type' that the item was (verb, punctuation, noun, etc.). If the item was shorter than four characters, it was immediately discarded. This ensured that all punctuation and smaller, meaningless words such as 'it', 'the', 'and', etc. were ignored. If the item was at least four characters long, the part of speech tag was compared to the list of 'allowed word types' (called 'allowed_word_types'). Only the first letter of the part of speech tag was analysed, which enabled a variety of adjectives (i.e. adjectives, comparative adjectives, and superlative adjectives), verbs and adverbs to all be considered. If the first letter of the part-of-speech tag was the

same as a letter in the list of allowed word types, the item was then added to the list of all words.

The remainder of the code was identical to the previous code displayed in section 4.2.2, a vocabulary was created using the most common words in the list of all words, the 'dictionary_of_words' function was applied to both the training and testing datasets, and the classifiers were trained and tested.

4.2.4 Training the Classifiers using Bigrams without Part of Speech Tagging

The code used for this model was another adaptation of the code shown in section 4.2.2. However instead of separating sentences into unigrams, they were separated into bigrams. To execute this, lines 18-23 of the original code displayed in section 4.2.2 was replaced with:

```
all_words = []

for item in train_data:
   bigrm = list(nltk.bigrams(item[0].split()))
   for i in bigrm:
       all_words.append(i[0].lower() + " " + i[1].lower())
```

The 'bigrams' function, a tool from the 'nltk' module, was applied to each sentence in the training dataset. This created a list of tuples, each tuple consisting of two separate items, both an individual word, i.e.:

```
[('is', 'with'), ('with', 'Seth'), ('Seth', 'and'), ('and', 'tylerrrr')]
```

However, this was the wrong format since the items were required to be pairs of words, not tuples containing the words individually. Therefore, the 'for loop' retrieved each item in the tuple, and combined them (in lowercase) with a space in between, before adding them to the list of all words in the correct format, i.e.:

```
[..., 'is with', 'with Seth', 'Seth and', 'and tylerrrr']
```

The vocabulary was created using the same method described in section 4.2.2, from the most common items in list of all words. However, the 'dictionary_of_words' function was changed slightly, lines 36-41 from section 4.2.2 was replaced with:

```
def dictionary_of_words(text):
   words = []
   bigrm = list(nltk.bigrams(text.split()))
   for item in bigrm:
       words.append(item[0].lower() + " " + item[1].lower())
   Dictionary = {}
   for word in vocab:
       Dictionary[word] = (word in words)
   return Dictionary
```

This altered the function by ensuring that birgams were added to the dictionary instead of unigrams, and that they were added in the correct format (pairs of words rather than tuples containing each individual word, using the same method as described above).

4.2.5 Training the Classifiers using Bigrams with Part of Speech Tagging

The alterations made for this model were mostly similar to the alterations shown in section 4.2.4. However, bigrams were only added to the list of all words if the first word was an 'allowed word type'. Lines 18-23 of the code displayed in 4.2.2 was therefore replaced with:

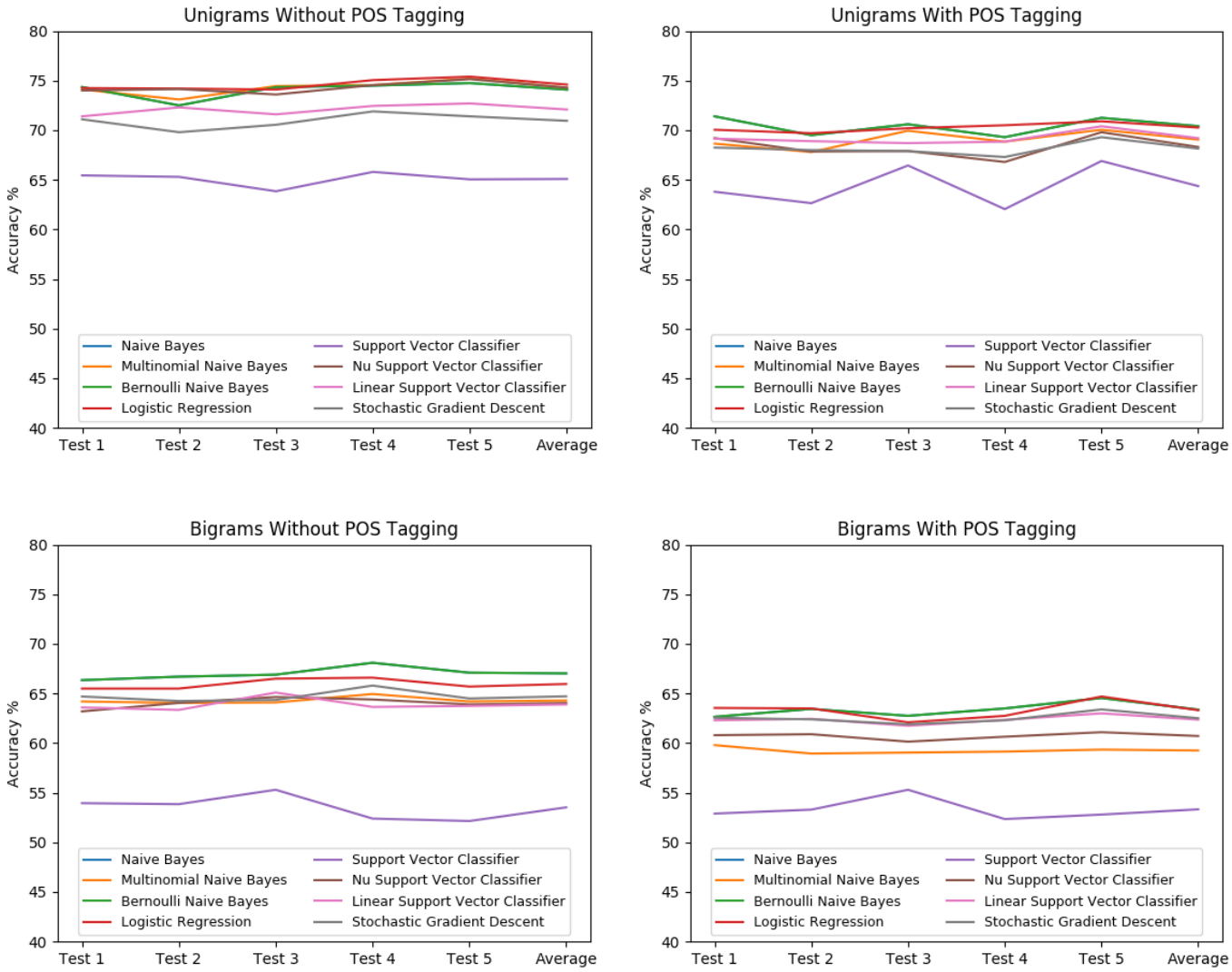
```
all_words = []
allowed_word_types = ["J", "R", "V"] # adjective, adverb, verb

for item in train_data:
   bigrm = list(nltk.bigrams(item[0].split()))
   for i in bigrm:
       POS = nltk.pos_tag(i)
       for word in POS:
           if len(word[0]) > 3:
               if word[1][0] in allowed_word_types:
                   all_words.append(i[0].lower() + " " + i[1].lower())
```

If the first item in the pair was at least four characters long, the 'nltk' part of speech tagging function was applied. The bigram would be added to the list of all words if the first word was an 'allowed word type'. The remaining code for this model was the same as the code described in section 4.2.4.

4.3 Classifier Results

After all classifiers had been trained and tested using each pre-processing condition, for five different subsets of the pre-classified dataset, the results were analysed. Figure 1 presents the outcome.

Figure 1: Results for training and testing the classifiers.



The accuracy, displayed on the y axis, is the percentage of items that were correctly labelled by the classifier. Note that the y axis ranges from 40% to 80%. The x axis shows the five tests, each one performed using a different subset of the training data, along with the average, which displays the mean result for each classifier over all five tests. Each line represents the performance of a classifier.

From the results, three things are clear:

1. Classifiers trained without part-of-speech tagging yield more accurate results than those trained with part-of-speech tagging.
2. Classifiers trained using unigrams produce more accurate results than those trained using bigrams.
3. The support vector classifier generates consistently poor results, relative to the other classifiers.

4.4 Creation of the 'Combined Classifier'

A combined classifier was created, using methods demonstrated by Sentdex (2015), which incorporates multiple pre-trained classifiers to label an item of text. The combined classifier works by passing an item of text through each pre-trained classifier, and returning the mode (most common) result, as well as a confidence score, calculated as the number of classifiers that 'agreed' on the final result.

The individual classifiers that were used to create the combined classifier were those that produced the most accurate results, i.e., those trained using unigrams without part of speech tagging. The Support Vector Classifier was not included for two reasons:

1. It performed consistently poorly relative to the other classifiers.
2. An odd number of classifiers was required in order for the mode result to be selected.

A new subset of 10,000 items was created from the pre-classified dataset, and each classifier, except for the Support Vector Classifier, was trained for a final time. The code used was identical to the code displayed in section 4.2.2, except after training, each classifier was saved using pickle, as was the vocabulary that was created using that particular subset of training data. Pickling the vocabulary and the classifiers meant that they could be reused easily, without the classifiers having to be retrained. Therefore they would be ready for use in the combined classifier.

The code used to create the combined classifier was as follows:

```
import pickle
from statistics import mode

from nltk.classify import ClassifierI
from nltk.tokenize import word_tokenize
```

Python's built-in modules 'pickle' and 'statistics' were imported, along with the 'nltk' 'word_tokenize' and 'ClassifierI' functions. 'ClassifierI' is a feature of the 'nltk' library which supports multiple classifier operations and enables classifiers to be combined.

```
class Vote_Classifier(ClassifierI):
    def __init__(self, *classifiers):
        self.classifiers = classifiers

    # Takes a list of all results and returns the mode, i.e. most common
    def classify(self, features):
        votes = []
        for classifier in self.classifiers:
            vote = classifier.classify(features)
            votes.append(vote)
        return mode(votes)

    # Returns the % of classifiers that chose that result
    def confidence(self, features):
        votes = []
        for classifier in self.classifiers:
            vote = classifier.classify(features)
            votes.append(vote)

        choice = votes.count(mode(votes))
        confidence_ = choice / len(votes)
        return confidence_
```

A class, 'Vote_Classifier', was created that takes a list of classifiers and returns both the mode (most common) result and the confidence of the result (the number of classifiers that 'agreed') after passing an item of text through each classifier.

```
vocab_f = open("vocab.pickle", "rb")
vocab = pickle.load(vocab_f)
vocab_f.close()

def dictionary_of_words(text):
    words = word_tokenize(text.lower())
    Dictionary = {}
    for word in vocab:
        Dictionary[word] = (word in words)
    return Dictionary
```

The vocabulary was retrieved, and the 'dictionary_of_words' function was copied over.

```
open_file = open("Original_NB.pickle", "rb")
NB_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("MNB.pickle", "rb")
MNB_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("BNB.pickle", "rb")
BNB_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("LR.pickle", "rb")
LR_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("NuSVC.pickle", "rb")
NuSV_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("LSV.pickle", "rb")
LSV_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("SGDC.pickle", "rb")
SGD_Classifier = pickle.load(open_file)
open_file.close()

vote = Vote_Classifier(
    NB_Classifier,
    BNB_Classifier,
    LR_Classifier,
    MNB_Classifier,
    NuSV_Classifier,
    LSV_Classifier,
    SGD_Classifier)
```

Each pre-trained classifier was ‘unpickled’ and applied to the ‘Vote_Classifier’ class. Therefore these were the classifiers that would be used to analyse an item of text.

```
def sentiment(text):
    features = dictionary_of_words(text)
    return vote.classify(features), vote.confidence(features)
```

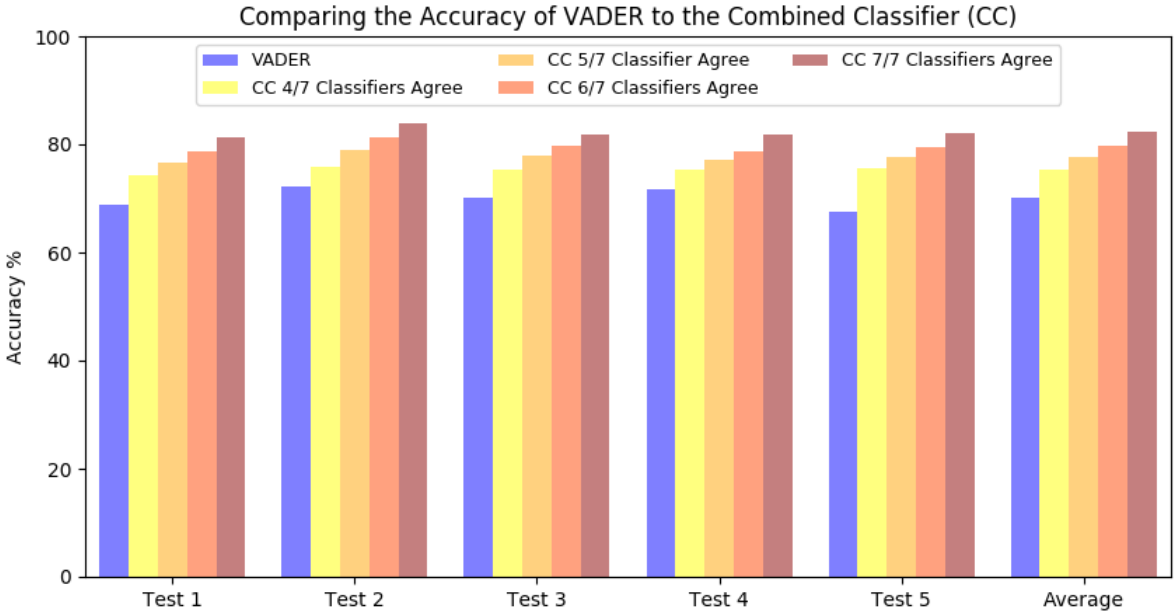
One last function, ‘sentiment’ was then created. This function passes the text through the ‘dictionary_of_words’ function, before applying it to the ‘Vote_Classifier’ class, which feeds the text through all seven classifiers before returning the results.

4.5 Testing the Combined Classifier against VADER Results

To ensure that the results produced by the combined classifier were beyond satisfactory, the combined classifier was compared against VADER (2014), a well-established lexicon and rule-based classifier designed for use specifically with social media text. Five test sets each containing 2,000 items were created from the pre-

classified dataset. Both the combined classifier and VADER were trialled on each test set. The results are illustrated below in figure 2.

Figure 2: Results of the combined classifier tested against VADER.



The accuracy, displayed on the y axis, is the percentage of items that were correctly labelled by the both VADER and the combined classifier. The x axis shows the five tests, each performed using a different subset of the pre-classified dataset, as well as the average result from all five tests. Each bar represents the accuracy of either VADER or the combined classifier at various levels of confidence, for each test set.

Two things to be taken into consideration:

1. VADER classifies text slightly differently to the combined classifier, text can be labelled as either positive, negative or neutral, rather than just positive or negative (it also returns a numerical score indicating strength of the sentiment). When evaluating the accuracy of VADER, the results that were classified as 'neutral' were discarded. The accuracy was based entirely on whether VADER classified a positive item as positive, and a negative item as negative, in order to obtain a meaningful comparison.
2. By observing the results from the combined classifier, it is evident that accuracy and confidence are strongly linked. That is, the higher the confidence of the combined classifier, the higher the accuracy of the results. It is important

to note that by only accepting results with high confidence scores, data classified with a lower confidence score gets discarded. Table 1 shown below demonstrates the number of tweets that were used for each confidence score, from the experiments against VADER.

Table 1: The number of tweets analysed for each level of confidence.

	Test 1	Test 2	Test 3	Test 4	Test 5	Average
CC 4/7 Agree	2000	2000	2000	2000	2000	2000 (100%)
CC 5/7 Agree	1801	1790	1803	1802	1803	1800 (90%)
CC 6/7 Agree	1624	1602	1633	1644	1633	1628 (81%)
CC 7/7 Agree	1301	1310	1321	1356	1288	1315 (66%)

As table 1 shows, the higher the confidence score, the less data was used. When analysing the 2014 Twitter dataset, it had to be decided whether it was better to have higher accuracy or slightly more data.

4.6 2014 Twitter Dataset

After establishing an accurate classification method, the 2014 Twitter dataset that was provided by the ONS was analysed.

4.6.1 Separating the Dataset

The dataset consisted of approximately 81 million geo-tagged tweets, stored in MongoDB[3]. MongoDB is a database used for storing large amounts of data, which enables relatively fast access to the data. The dataset was divided first into geographic regions (based on the regions from the Annual Population Survey), and then into months. Separating the dataset enabled much faster access to specific tweets, as opposed to searching through the entire datasets for tweets with particular dates or locations.

[3] https://www.mongodb.com/

4.6.1.1 Separating the Dataset by Location

Each item in the dataset contained a variety of information, including the coordinates and the postcode of the tweet. By consulting Wikipedia's (2017) list of postcode areas in the United Kingdom, each item in the dataset was assigned to a region based on its postcode. The code below shows demonstrates how this was done.

```
1 import pymongo
2 from pymongo import MongoClient
3
4 client = MongoClient()
5 db = client.twitter
6 collection = db.tweets
```

The module 'pymongo' was imported, which enables access to the mongo database from Python. Line 4 demonstrates how a connection was established with MongoDB. Line 5 shows how the database, named 'twitter', was accessed, and line 6 illustrates interaction with the specific collection, named 'tweets', within the database.

```
7 London = ['W1', 'W2', 'W3', 'W4', 'W5', 'W6', 'W7', 'W8',
8           'W9', 'SW', 'SE', 'WC', 'EC', 'E1', 'E2', 'E3',
9           'E4', 'E5', 'E6', 'E7', 'E8', 'E9', 'N1', 'N2',
10           'N3', 'N4', 'N5', 'N6', 'N7', 'N8', 'N9', 'NW',
11           'W0', 'E0', 'N0']
12
13 NorthernIreland =  ['BT']
14
15 Scotland = ['AB', 'DD', 'DG', 'EH', 'FK', 'G1', 'G2', 'G3',
16            'G4', 'G5', 'G6',  'G7', 'G8', 'G9', 'HS', 'IV',
17            'KA', 'KW', 'KY', 'ML', 'PH', 'PA', 'ZE', 'G0']
18
19 NorthEast = ['DH', 'NE',  'SR', 'TD', 'TS']
20
21 NorthWest = ['BB', 'BL', 'CA', 'CW', 'CH', 'FY', 'L1', 'L2',
22             'L3', 'L4', 'L5', 'L6', 'L7', 'L8', 'L8', 'L9',
23             'L0', 'M0', 'M1', 'M2', 'M3', 'M4',  'M5', 'M6',
24             'M7', 'M8', 'M9', 'OL', 'PR', 'WN',  'SK',  'WA',
25             'LA']
26
27 YorkshireHumber =  ['DL', 'BD', 'DN', 'HD', 'HG', 'HU', 'HX',
28                     'LS', 'S0', 'S1', 'S2', 'S3', 'S4', 'S5',
29                     'S6', 'S7',  'S8', 'S9', 'WF', 'YO']
30
31 EastMidlands = ['DE', 'LE', 'LN', 'NG', 'NN']
32
33 WestMidlands = ['B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7',
34                 'B8', 'B9', 'CV', 'DY', 'HR', 'ST', 'SY', 'TF',
35                 'WR', 'WV', 'WS']
36
37 East = ['CB', 'SS', 'CM', 'CO', 'RM', 'AL', 'EN', 'SG', 'WD',
38        'IP', 'NR', 'PE']
39
40 SouthEast = ['OX', 'MK', 'BN', 'BR', 'CR', 'CT', 'DA', 'GU',
41             'HP', 'KT', 'ME', 'PO', 'RG', 'RH', 'SL',  'SM',
42             'SO', 'TN', 'TW', 'UB']
43
44 SouthWest = ['BA', 'BH', 'BS', 'DT', 'EX', 'GL', 'PL', 'SN',
45             'SP', 'TA', 'TQ', 'TR']
```

```
46
47 Wales = ['CF', 'LD',  'LL', 'NP', 'SA']
```

A list was created for every region, and the first two letters of every postcode in the UK were added to a list, based on the location of the postcode.

```
48 for item in collection.find():
49    postcode = item['tweet']['address']['postcode']
50    if postcode[0:2] in London:
51        db.London.insert(item)
52    elif postcode[0:2] in NorthernIreland:
53        db.NorthernIreland.insert(item)
54    elif postcode[0:2] in Scotland:
55        db.Scotland.insert(item)
56    elif postcode[0:2] in NorthEast:
57        db.NorthEast.insert(item)
58    elif postcode[0:2] in NorthWest:
59        db.NorthWest.insert(item)
60    elif postcode[0:2] in YorkshireHumber:
61        db.YorkshireHumber.insert(item)
62    elif postcode[0:2] in EastMidlands:
63        db.EastMidlands.insert(item)
64    elif postcode[0:2] in WestMidlands:
65        db.WestMidlands.insert(item)
66    elif postcode[0:2] in East:
67        db.East.insert(item)
68    elif postcode[0:2] in SouthEast:
69        db.SouthEast.insert(item)
70    elif postcode[0:2] in SouthWest:
71        db.SouthWest.insert(item)
72    elif postcode[0:2] in Wales:
73        db.Wales.insert(item)
74    else:
75        db.Unknown.insert(item)
```

The code demonstrates how Python iterated through every item in the dataset and accessed the item's postcode. The first two letters of each postcode were compared against the lists of postcodes, to establish which region the postcode belonged to. The 'insert' function inserted the current item into a collection based on the postcode. This function created a new collection if it did not already exist. Unfortunately the dataset did not contain any tweets from Northern Ireland, an area that is monitored by the APS.

4.6.1.2 Separating the Locations by Months

The collections for each region were then further divided into months. An example for separating one region is as follows:

```
1 import pymongo
2 from pymongo import MongoClient
3 client = MongoClient()
4
5 db = client.twitter
6 LondonCollection = db.London
```

‘pymongo’ was again imported and a connection was established. Each collection that was made based on the location was then accessed in turn. Line 6 shows a connection being established with London’s collection.

```
7 for item in LondonCollection.find():
8    month = item['time']['month']
9    if month == 'Apr':
10        db.LondonApr.insert(item)
11    elif month == 'May':
12        db.LondonMay.insert(item)
13    elif month == 'Jun':
14        db.LondonJun.insert(item)
15    elif month == 'Jul':
16        db.LondonJul.insert(item)
17    elif month == 'Aug':
18        db.LondonAug.insert(item)
19    elif month == 'Sep':
20        db.LondonSep.insert(item)
21    elif month == 'Oct':
22        db.LondonOct.insert(item)
23    else:
24        db.LondonUnknown.insert(item)
```

The ‘if statement’ iterated through each item in the collection, obtained the month of the tweet, and inserted the item into a collection based on both the region and the month, using a similar method to before.

4.6.2 Applying the Combined Classifier to the Twitter Dataset

The combined classifier was then applied to the tweets. Due to the time frame of the project and the fact that the combined classifier was quite slow, only a subsection of the data was analysed. 700 tweets were selected per day from each region and passed through the combined classifier. The code below demonstrates how the tweets for the month of April in London were classified:

```
1 import pymongo
2 from pymongo import MongoClient
3 client = MongoClient()
4
5 import Twitter_Sentiment_Module as TSM
6 import pickle
7
8 import csv
9 import collections
10
11 db = client.twitter
```

'pymongo' was imported and a connection was established. The combined classifier, 'Twitter_Sentiment_Module', was imported as well as 'pickle' and 'csv'. A built-in python module called 'collections' was also imported, which was used as a counter for both the number of tweets per day that had been classified (to ensure 700 were analysed for each day), and the number of tweets that were labelled as positive.

```
12 ##### LONDON
13
14 ### April
15
16 LondonAprCollection = db.LondonApr
17
18 LondonApr_day_count = collections.defaultdict(int)
19 LondonApr_pos_count = collections.defaultdict(int)
20
21 for item in LondonAprCollection.find().batch_size(50):
22    date = item['time']['date']
23    text = item['tweet']['text']
24    if len(text) >= 40:
25        if LondonApr_day_count[date] < 700:
26            if TSM.sentiment(text)[1] >= 0.8:
27                LondonApr_day_count[date] += 1
28                if TSM.sentiment(text)[0] == 'pos':
29                    LondonApr_pos_count[date] += 1
30
31 print('London April')
32 for key, value in sorted(LondonApr_day_count.items()):
33    print(key, value, LondonApr_pos_count[key])
34    items = ("{0}, {1}, {2}".format(key, value, LondonApr_pos_count[key]))
35    f = open('London.csv', 'a')
36    f.write(items)
37    f.write('\n')
```

The 'for loop' accessed items from London's April collection, stored in MongoDB, in batch sizes of 50. Note that a batch size of 50 was used instead of the default batch size, because interactions with large datasets using the default setting can cause it to 'timeout' after 10 minutes. Both the date and the text were accessed from each item. The first 'if statement' checked whether the text was at least 40 characters long. This was done to ensure there was enough text for the item to be properly classified. The second 'if statement' ensured that the number of tweets analysed for each day of the month reached 700, and did not surpass 700. The third 'if statement' applied the combined classifier to the text and checked whether the confidence was over 0.8, i.e. at least 6/7 classifiers agreed. If the confidence was over 0.8, the counter for the number of tweets analysed per day was incremented by one. The fourth 'if statement' increased the positive counter by one if the tweet was classified as positive.

Once the day counter had reached 700 for every date in that month, i.e. 700 tweets per day had been analysed, the process stopped and the results for each day (the date,

number of tweets that had been analysed (700), and the number of tweets that were classified as positive) were printed and then added to a csv file for the region. This process was repeated until every month within each region had been analysed.

5. Results

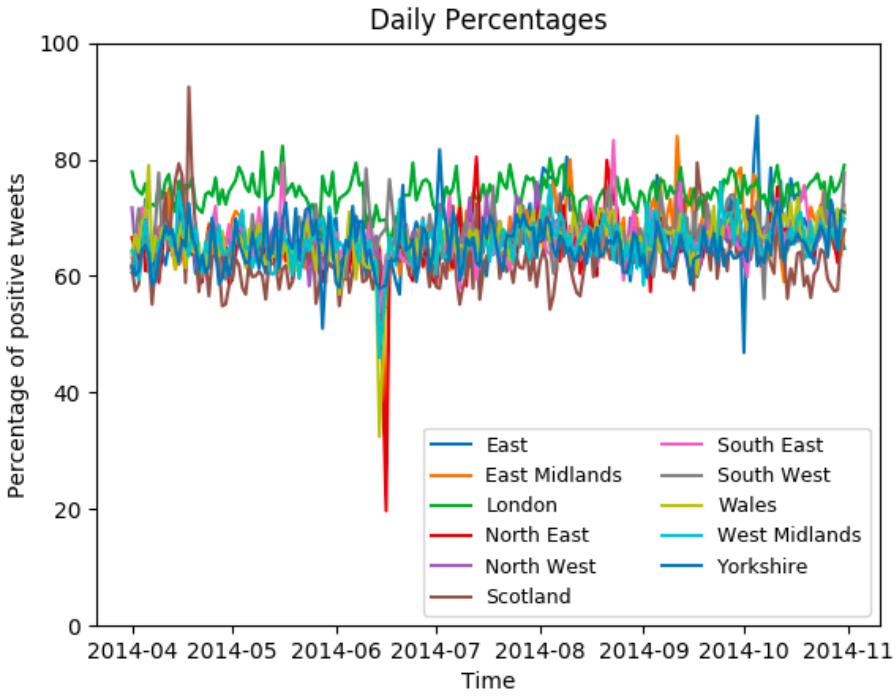
5.1 Twitter Results

After 700 tweets had been classified per day for each region, the percentage of positive tweets per day and per week were computed and analysed.

5.1.1 Percentage of Positive Tweets per Day

Figure 3 illustrates the percentage of tweets analysed per day, for each region, which were labelled as positive by the combined classifier.

Figure 3: Percentage of positive tweets per day.



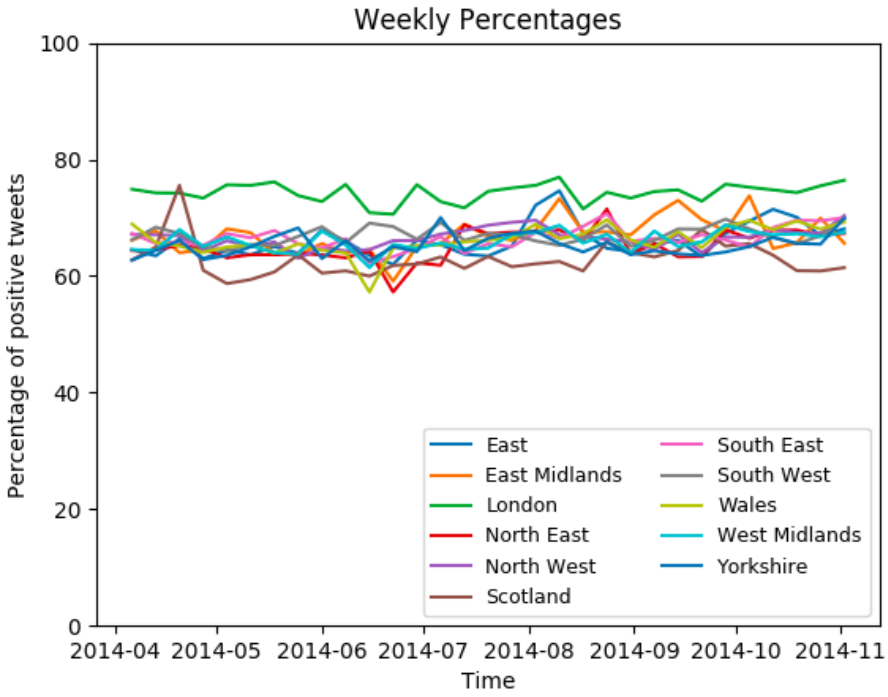
The percentage of positive tweets, displayed on the y axis, represents the proportion of tweets analysed daily that were labelled as positive. The x axis shows the period of time (the start of April 2014 to the end of October 2014) under analysis. Most regions have approximately 60-70% positive tweets per day. Large fluctuations can be identified in the graph, some that are particularly noticeable include: the sharp increase in positive tweets in the first month for Scotland, and the abrupt decrease in positive

tweets in June for the West Midlands, Wales and the North East. It is possible that these fluctuations could be random, however they could also be determined by socio-economic events, or even the results of well-known activities such as sports matches. Whilst viewing the data daily enables fluctuations to be analysed, it is difficult to identify other trends due to the noisiness of the data.

5.1.2 Percentage of Positive Tweets per Week

Figure 4 illustrates the percentage of tweets analysed per week, for each region, which were labelled as positive by the combined classifier.

Figure 4: Percentage of positive tweets per week.

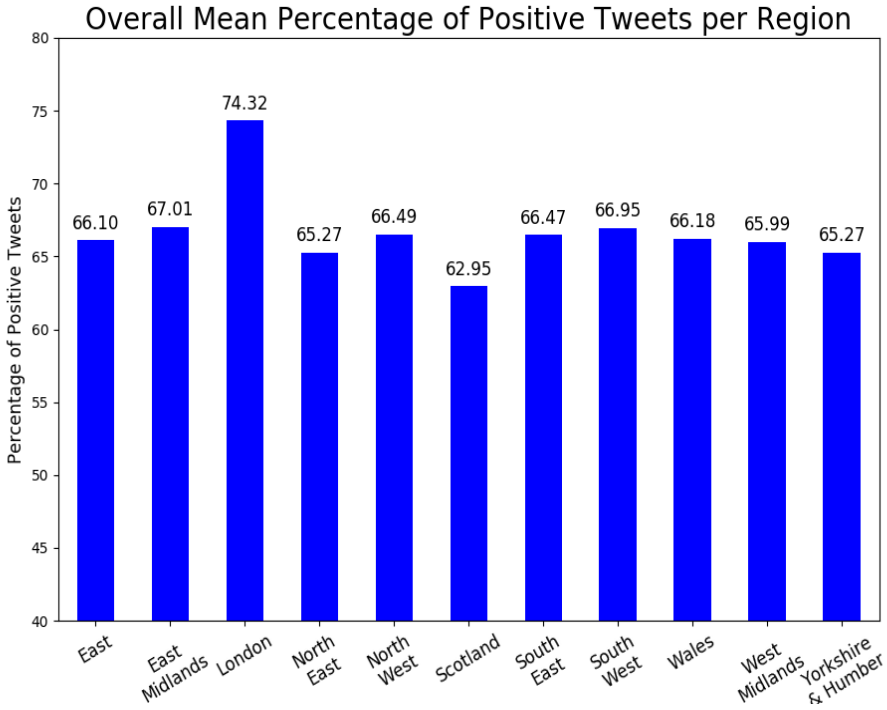


The percentage of positive tweets, displayed on the y axis, represents the proportion of tweets analysed per week that were labelled as positive. The x axis again displays the time scale. Viewing the data weekly instead of daily allows trends to be established more clearly. From this graph it is easier to observe that London has a higher percentage of positive tweets than all other regions. In contrast, Scotland appears to have the lowest percentage of positive tweets. The remaining regions have a similar percentage of positive tweets, approximately 65%. The fluctuations that were identified previously in figure 3 can still be seen, however they are less prominent.

5.1.3 Mean Percentage of Positive Tweets per Region

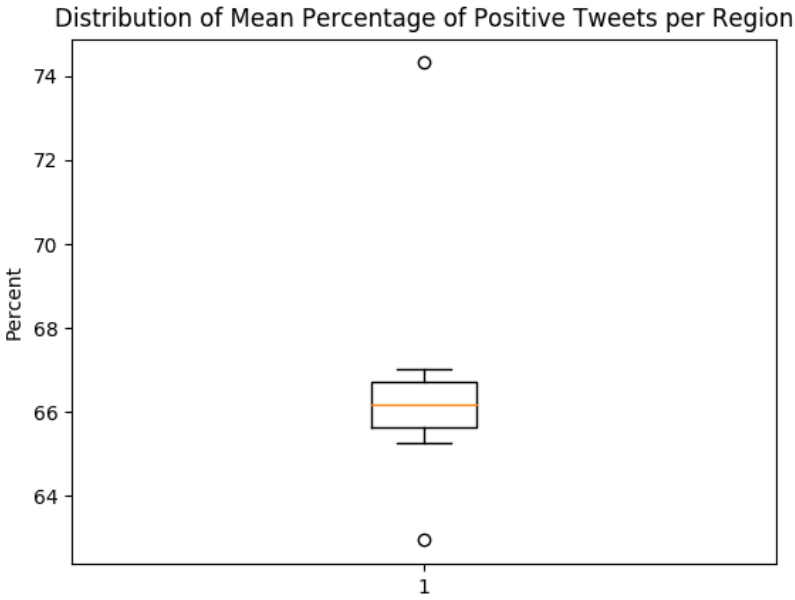
To compare the Twitter results to the results from the Annual Population Survey (APS), the mean value was taken for each region. The results for the mean percentage of positive tweets can be seen below in figure 5.

Figure 5: Mean percentage of positive tweets per region.



The y axis displays the overall mean percentage of positive tweets, note that it ranges from 40-80%, and the x axis represents each region that was analysed. The graph verifies that London has a relatively high percentage of positive tweets (approximately 74%), compared with other regions. Conversely, Scotland has the lowest percentage of positive tweets (approximately 63%). Figure 6 below confirms that London and Scotland are significant outliers within the data.

Figure 6: Boxplot of mean percentage of positive tweets per region.



This boxplot shows the distribution of the mean percentage of positive tweets per region. The orange line in the middle of the boxplot represents the median result, i.e. 50% of the data is greater than this value, and 50% of the data is smaller than this value. The median value is approximately 66%. The top and bottom lines of the box represent the third and first quartiles respectively, where 25% of the data is greater than the third quartile, and less than the first quartile. The whiskers above and below the box extend to 1.5*IQR (inter-quartile range), and show the locations of the minimum and maximum values, excluding any outliers. The outliers, illustrated by circles on the graph, are greater than 1.5*IQR above the third quartile or below the first quartile. The circle displayed at the top of the graph represents London, and the circle at the bottom of the graph represents Scotland.

5.2 APS Wellbeing Results

The results from the Annual Population Survey wellbeing questions, for each region, are displayed below.

Recall, the four wellbeing questions asked on the APS are:

1. Overall, how anxious did you feel yesterday?
2. Overall, how happy did you feel yesterday?
3. Overall, how satisfied are you with your life nowadays?

4. Overall, to what extent do you feel the things you do in your life are worthwhile?

Figure 7: APS anxiety results.

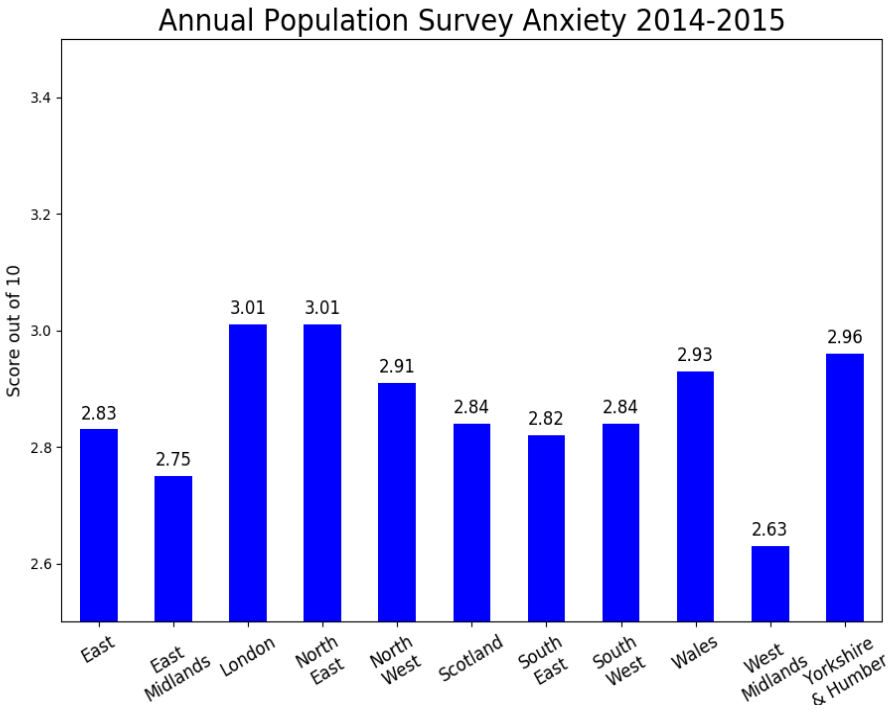


Figure 8: APS happiness results.

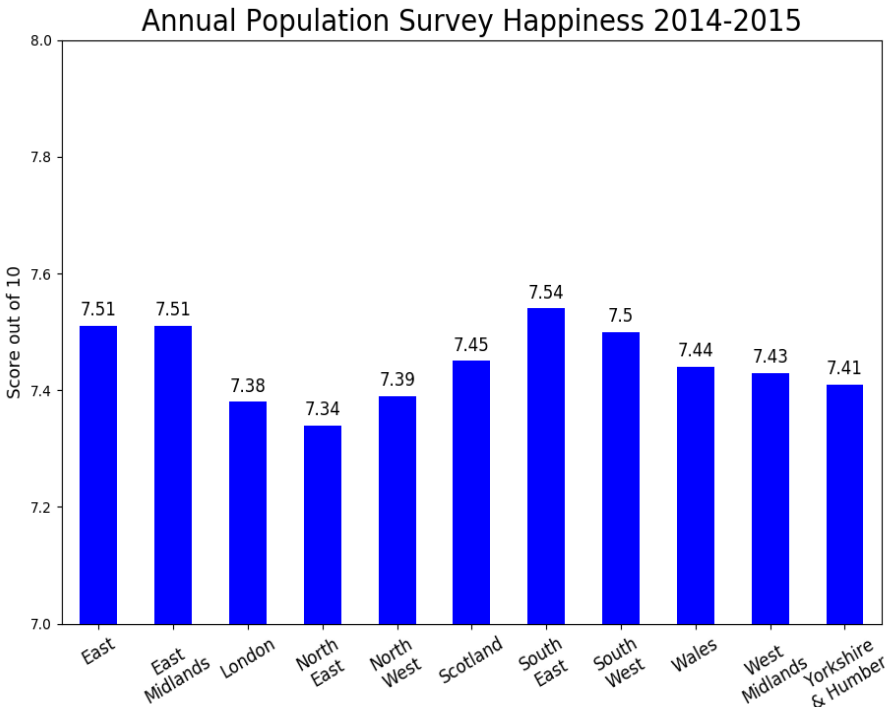


Figure 9: APS life satisfaction results.

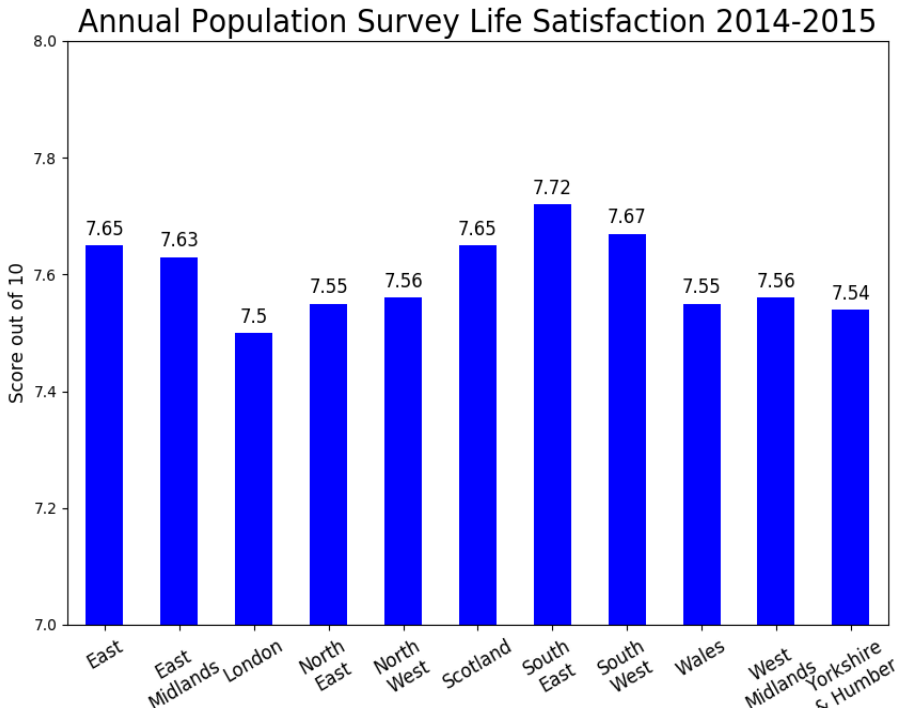
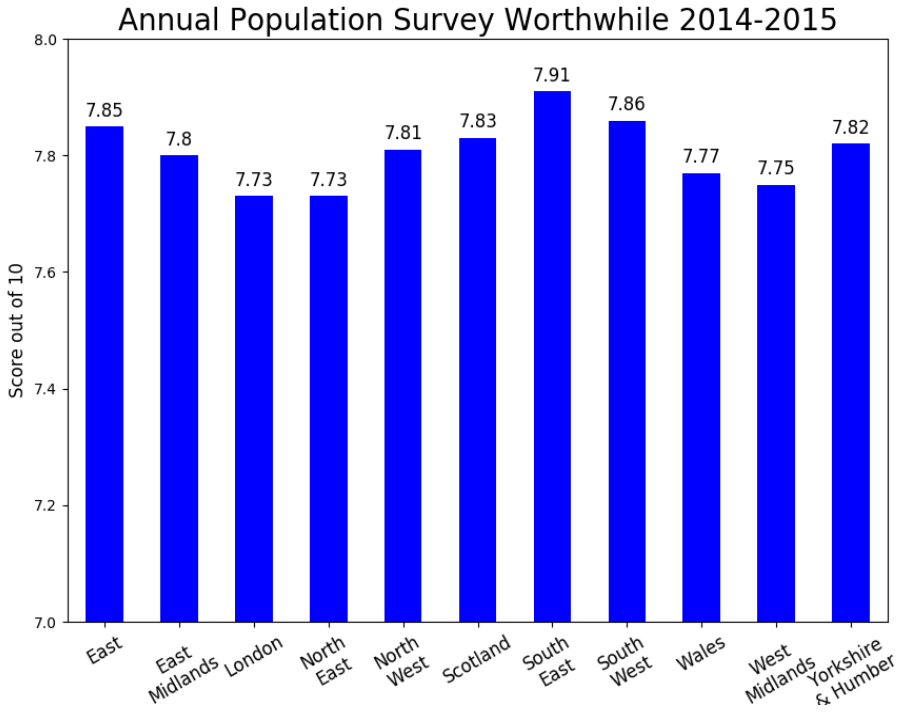


Figure 10: APS worthwhile results.



Figures 7-10 display the results from each wellbeing question in 2014. The y axis represents the mean score out of 10 for each region, and the x axis indicates the region.

Note that the y axis is between 7 and 8 for happiness, life satisfaction and worthwhile, and between 2.5 and 3.5 for anxiety. Higher scores for all questions, except anxiety, represent higher wellbeing, whereas a lower score for anxiety represents higher wellbeing. The results show that the West Midlands has the lowest score for anxiety, and that both London and the North East have the highest anxiety scores. The South East has the highest levels of happiness, life satisfaction and worthwhile, and the North East and London have the lowest.

The graphs below display the distribution of results from each wellbeing question.

Figure 11: Boxplot of APS anxiety results.

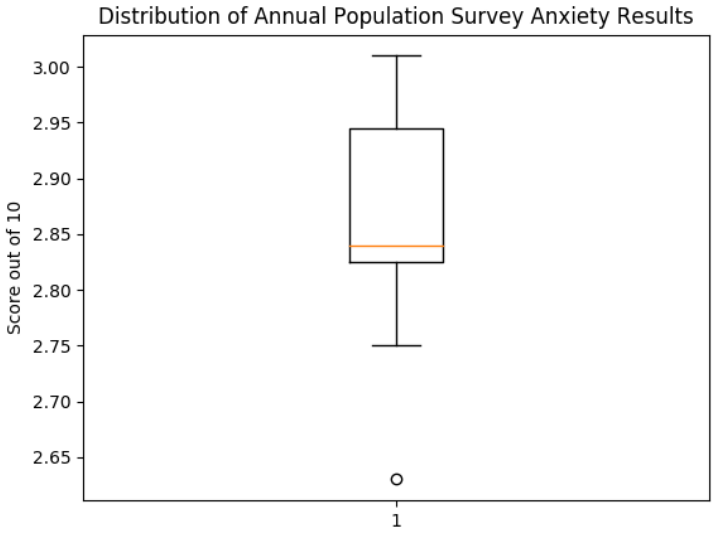


Figure 12: Boxplot of APS happiness results.

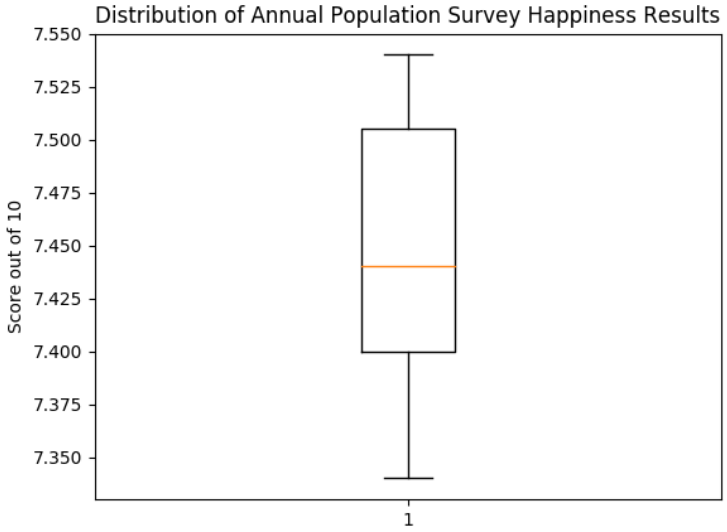


Figure 13: Boxplot of APS life satisfaction results.

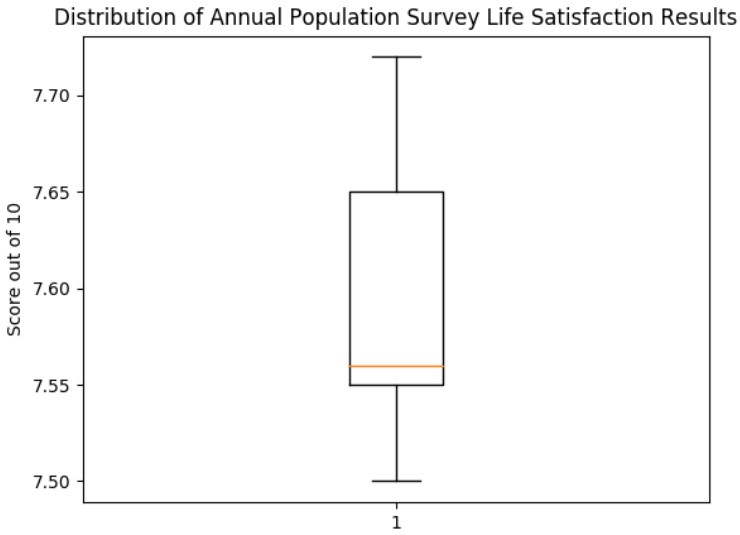
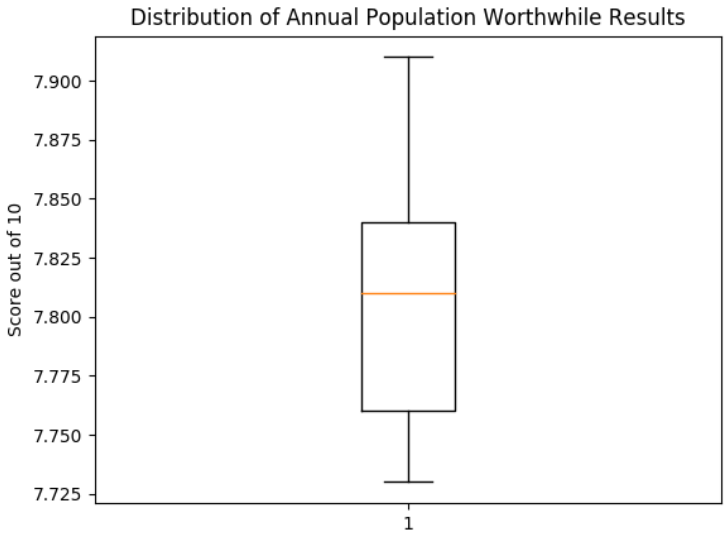


Figure 14: Boxplot of APS worthwhile results.



Figures 11-14 aimed to identify any outliers from the results of the Annual Population Survey. The only outlier, which appeared in the results from the anxiety question, was the West Midlands. In contrast with the outcome of the Twitter analysis, London and Scotland were not outliers.

5.3 Comparisons between Twitter and the APS Wellbeing Results

The mean, median and standard deviation of the Twitter sentiment results were calculated and compared to the results from each of the four APS questions, using Pearson's r correlation. The results are as follows:

5.3.1 Anxiety

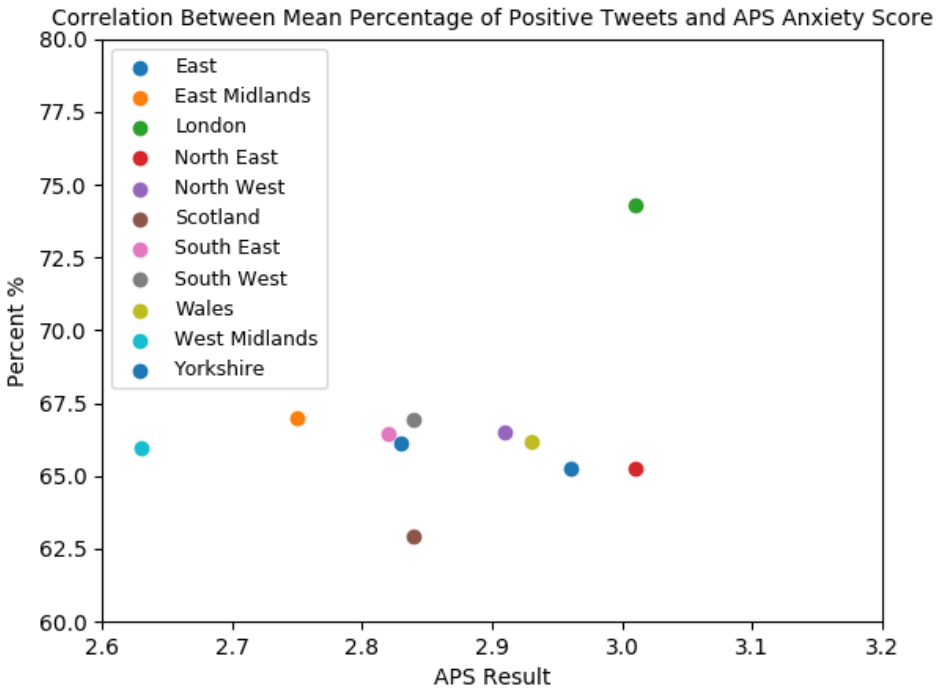
Table 2: Correlation between Twitter and APS anxiety results.
Correlation (p-value)

	Mean	Median	Standard Deviation
All Regions	0.305 (0.361)	0.306 (0.361)	-0.095 (0.782)
No London or Scotland	-0.484 (0.186)	-0.396 (0.291)	0.199 (0.608)
No London, Scotland or West Midlands	-0.836 (0.009)*	-0.598 (0.117)	0.006 (0.989)

* Statistically significant at the 1% level

Only after all outliers were removed from the data, which in this case was London, Scotland (outliers from the Twitter sentiment dataset) and the West Midlands (an outlier from the anxiety results on the APS), was there a significant correlation between the Twitter results and the results from the anxiety question. The correlation, significant at the 1% level, is negative (-0.836) which suggests that regions with a higher percentage of positive tweets have lower levels of anxiety. Figure 15 illustrates this correlation.

Figure 15: Correlation between Twitter and APS anxiety results.



The y axis displays the percentage of positive tweets, note that it ranges from 60%-80%. The x axis represents the APS anxiety results, which ranges from 2.6 to 3.2. Each data point shown on the graph represents a region. It is clear that without Scotland, London or the West Midlands there is a negative correlation between the remaining regions.

5.3.2 Happiness

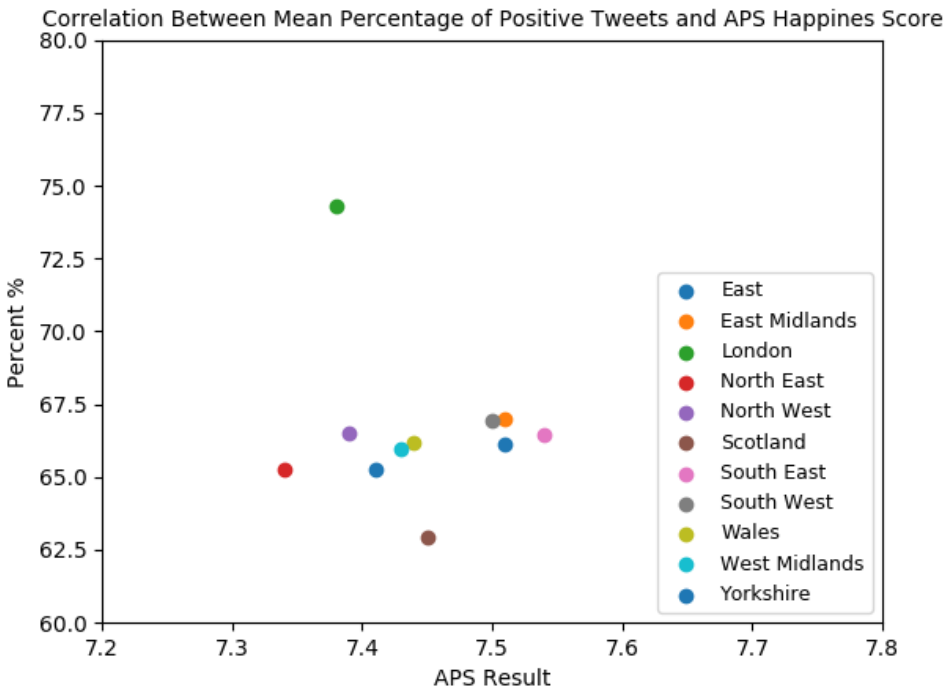
Table 3: Correlation between Twitter and APS happiness results.
Correlation (p-value)

	Mean	Median	Standard Deviation
All Regions	-0.179 (0.599)	-0.201 (0.553)	0.150 (0.659)
No London or Scotland	0.689 (0.040) **	0.474 (0.197)	-0.028 (0.942)

** Statistically significant at the 5% level

Again, only after the outliers were removed from the data (London and Scotland), was there a significant correlation between the Twitter data and results from the happiness question. The correlation, significant at the 5% level, is positive (0.689) which suggests that regions with higher percentages of positive tweets have higher levels of happiness. Figure 16 illustrates this correlation.

Figure 16: Correlation between Twitter and APS happiness results.



The y axis displays the percentage of positive tweets, note that it ranges from 60%-80%. The x axis represents the APS happiness results, which ranges from 7.2 to 7.8. Each data point shown on the graph represents a region. It is clear that without Scotland, or London, there is a positive correlation between the other regions.

5.3.3 Life Satisfaction

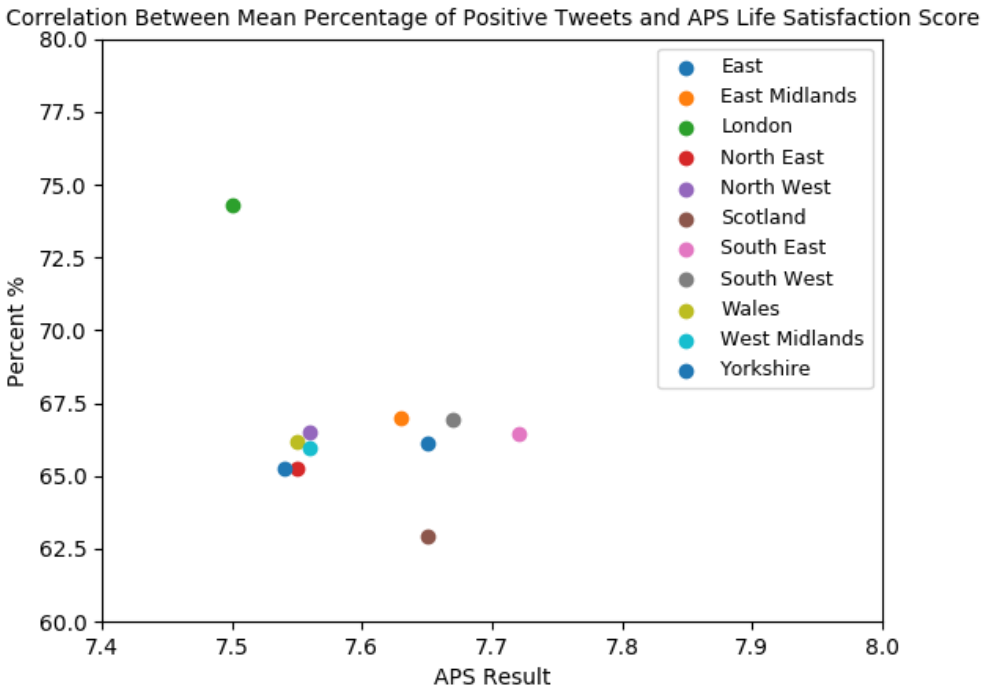
Table 4: Correlation between Twitter and APS life satisfaction results.
Correlation (p-value)

	Mean	Median	Standard Deviation
All Regions	-0.399 (0.224)	-0.421 (0.197)	0.327 (0.327)
No London or Scotland	0.620 (0.075) ***	0.420 (0.260)	-0.008 (0.983)

*** Statistically significant at the 10% level

After London and Scotland were removed a correlation emerged that was significant at the 10% level, but not the 5% level. Thus, depending on the significance level this shows that there is a positive correlation (0.620) between life satisfaction and the Twitter results. Figure 17 illustrates this relationship.

Figure 17: Correlation between Twitter and APS life satisfaction results.



The y axis displays the percentage of positive tweets, note that it ranges from 60%-80%. The x axis represents the APS life satisfaction results, which ranges from 7.4 to 8.0. Each data point shown on the graph represents a region. Again, without London or Scotland a relationship between outcomes can be identified. However, the correlation is only significant at the 10% level.

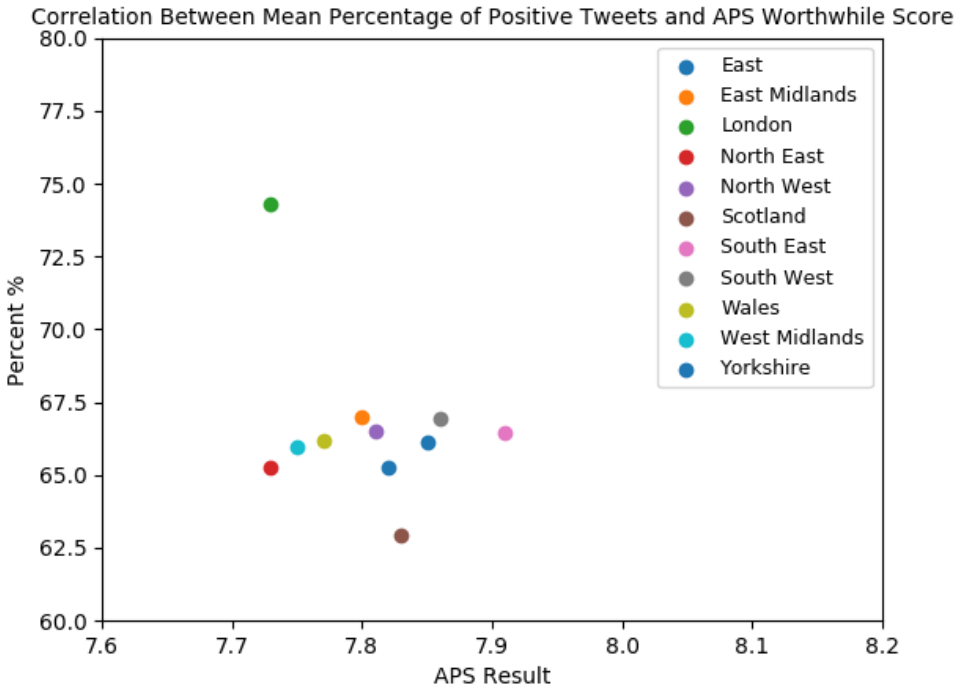
5.3.4 Worthwhile

Table 5: Correlation between Twitter and APS worthwhile results.
Correlation (p-value)

	Mean	Median	Standard Deviation
All Regions	-0.354 (0.286)	-0.373 (0.258)	0.075 (0.827)
No London or Scotland	0.495 (0.214)	0.286 (0.455)	-0.219 (0.447)

Even after removing all outliers from the data, there were no significant correlations between the Twitter results and the results of the worthwhile question. This suggests that a person's worthwhile, as measured by the Annual Population Survey, cannot be detected through Twitter. Figure 18 illustrates this.

Figure 18: Correlation between Twitter and APS worthwhile results.



The y axis displays the percentage of positive tweets, note that it ranges from 60%-80%. The x axis represents the APS worthwhile results, which ranges from 7.6 to 8.2. Each data point shown on the graph represents a region. Even without London or Scotland, it is clear that there is no correlation between the results for the regions.

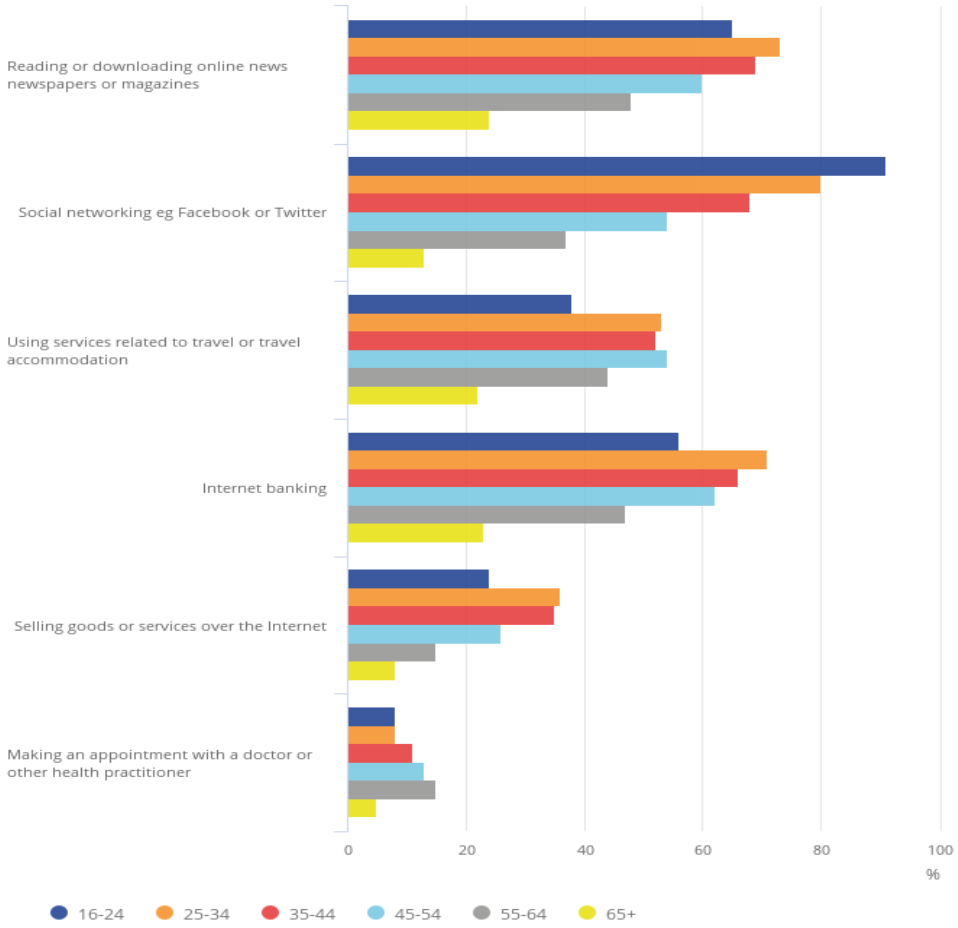
6. Discussion

This study was designed to investigate whether wellbeing can be detected from social media data, in order to develop a timelier and more dynamic measure of both national and localised well-being that can supplement results from the Annual Population Survey (APS). The outcome confirms that aspects of subjective wellbeing can be detected from social media, as other studies have also shown (Liu et al., 2015, Schwartz et al., 2016, and Yang and Srinivasan, 2016).

Using geo-tagged Twitter data enabled the tweets to be analysed according to regions, and compared to the corresponding regional results from the APS. A major benefit of using such data, is that much smaller areas can be analysed in comparison to the smallest areas (local authorities) measured by the APS. It also provides the opportunity to target specific areas for analysis, whilst still ensuring that sufficient data will be available.

Figure 19 demonstrates how in 2014, social networking was the most popular internet activity for people aged between 16-34, with other age groups still using it frequently relative to other activities. This emphasises how widely used social media was, particularly for younger ages.

Figure 19: Internet activities by age, 2014. Source: ONS, 2014.



Twitter has proven to be more addictive than both alcohol and cigarettes (Szalavitz, 2012). This combined with the growing popularity of internet usage 'on the go' (ONS, 2014) are major contributing factors to approximately 6,000 tweets being posted every second (Soulo, 2014). A large number of these updates contain user's thoughts and opinions on their everyday life which can act as direct indicators for their wellbeing. However, an invaluable aspect of sentiment analysis is that it does not have to analyse updates that are directly related to how a person feels, in order to gain an indication of their personality or wellbeing. For example, studies show that extroverts frequently use the word 'party' whilst introverts frequently use the word 'computer', and that emoticons of frowny faces are associated with neuroticism, whilst more stable people post mostly about sports (Eichstaedt, 2014).

Since tweets are being posted so regularly, they are likely to reflect more short-term opinions. Therefore, it is unsurprising that the two questions which showed the

greatest relationship with the Twitter data were the questions referring to positive and negative affect, which are the questions that ask users to indicate their happiness or anxiety *yesterday*, rather than the questions which ask for an evaluation of *life on the whole*.

6.1 Benefits of Social Media Data

Asides from the vast amount of data that social media provides in comparison with self-reported surveys, there are further, alternative benefits to using social media data, particularly to aid psychological studies. These benefits include:

- There is no response bias. It is unlikely that users whose tweets are being analysed, posted their update knowing that it would act as an indicator for their wellbeing. However, it is likely that they posted it as a reflection of their thoughts and opinions at the time. Therefore, it can be assumed that the updates are an accurate representation of the user's true feelings.
- Whilst it can be argued that there are certain social pressures people may feel the need to conform to (to show their life in a more positive, glamorous way), this could mean that negative tweets are more likely to accurately reflect lower levels of happiness or satisfaction than conventional measures. There has been research on detecting suicides from social media updates for this reason (Jashinsky et al., 2014 and O'Dea et al., 2015). This is possibly why the strongest correlation was between Twitter sentiment and the results from the APS anxiety question, as a negative emotion may be easier to detect.
- Surveys that require an answer as a 'scale' are highly subjective. For example, what one person may rank as 7 for a question, another may rank as 5. By using sentiment analysis to interpret the text, a fair evaluation is made in every instance. Additionally, there is no opportunity for a question to be misinterpreted and answered wrong.

6.2 Discussion of Methodology

Compared with other studies reported in the literature review, the approach taken in this project was unique since it relied on multiple classifiers to label an item of text. This resulted in a reliable model that outperformed an existing, well-established classification method (VADER), which ensured that the accuracy of the results were beyond satisfactory. It is evident from the graphs displaying the percentage of positive tweets per region (figures 3 and 4), that the classifier was able to identify regions with consistently higher and lower percentages of positive tweets, such as London and Scotland, proving that the classification method was relatively dependable.

From testing the classifiers using various pre-processing conditions, it was determined that classifiers trained using unigrams produced more accurate results than those trained using bigrams, and classifiers trained without part of speech tagging produced more accurate results than those trained with part of speech tagging. These results contradict research that suggests both bigrams and part of speech tagging produce more accurate classifiers (Naji, 2013). Possible reasons for this conflict could be:

- Punctuation is often used to express opinions on social media, i.e. ‘:)’, ‘:(‘, ‘!!!!’, etc., and words and acronyms that are most likely not recognised by part of speech tagging, such as ‘lol’, ‘haha’, ‘smh’, etc., tend to have a lot of meaning. Implementing part of speech tagging could actually be preventing this useful information from being included in the vocabulary, which consequently prevents the classifiers from learning what could be powerful indications for the sentiment of the text.
- Words are frequently misspelt on social media, whether it is by mistake or on purpose, for example they may be elongated, i.e. ‘happpppppyyyyyy’, or abbreviated, i.e. ‘how r u’. The likelihood that two consecutive words in a tweet are spelt correctly is probably quite low, certainly relatively lower than a document such as a newspaper article. It is also quite unlikely that two consecutive words will be misspelt the same. Therefore, the chances of two words in a sentence being recognised as being the same two words which are in the vocabulary, may be lower, which potentially could be a reason for a decrease in accuracy with classifiers trained using bigrams.

6.3 Discussion of Results

There were significant correlations (at the 5% level) between the Twitter results and two of the APS wellbeing questions. However, these correlations only emerged after the two outliers (London and Scotland) were removed.

6.3.1 London

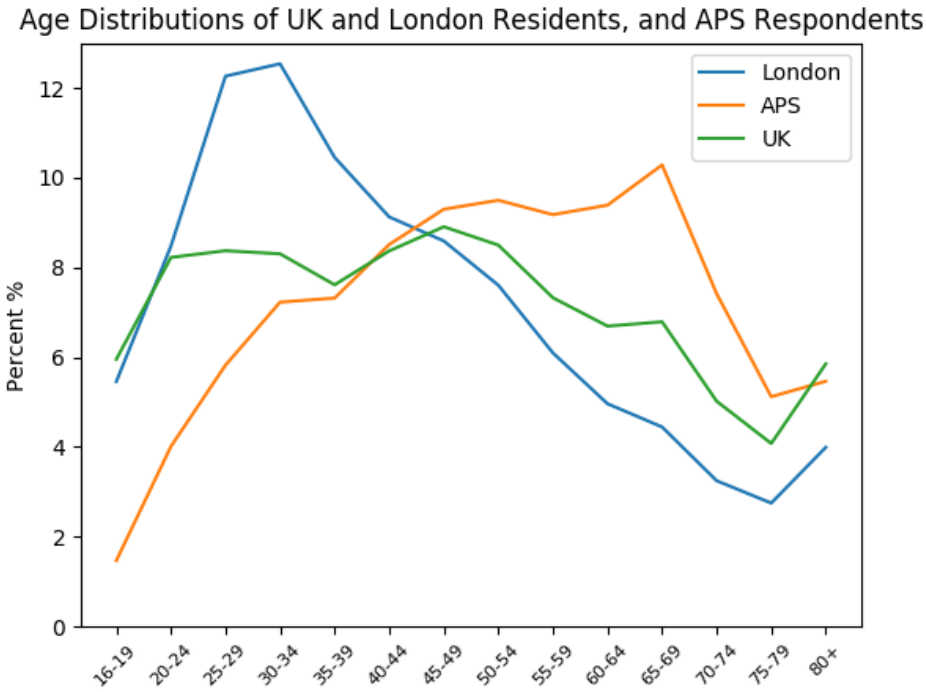
Not only was the mean percentage of positive tweets for London much higher than for all other regions, it was also very inconsistent with the results from the APS which indicated that London had one of the lowest overall wellbeing scores.

This discrepancy between results may potentially be a result of London being a big tourist attraction, which in turn could mean that a large number of tweets analysed were from tourists enjoying their visit, rather than the residents of London whose wellbeing is accurately reflected by the APS results.

Another possible explanation could be the age distribution of London citizens. Figure 20[4] illustrates the percentage of age groups of both London and UK residents in 2014, as well as the percentage of age groups of APS respondents in 2014.

[4] Data obtained from the ONS website.
Demographics of APS respondents available at:
https://www.ons.gov.uk/peoplepopulationandcommunity/wellbeing/datasets/personalwellbeingestimatespersonalcharacteristics [Accessed: 23 August 2017]
Demographics of UK residents available at:
https://www.ons.gov.uk/peoplepopulationandcommunity/populationandmigration/populationestimates/datasets/populationestimatesforukenglandandwalesscotlandandnorthernireland [Accessed: 23 August 2017]

Figure 20: Age distributions of UK and London residents, and APS respondents.



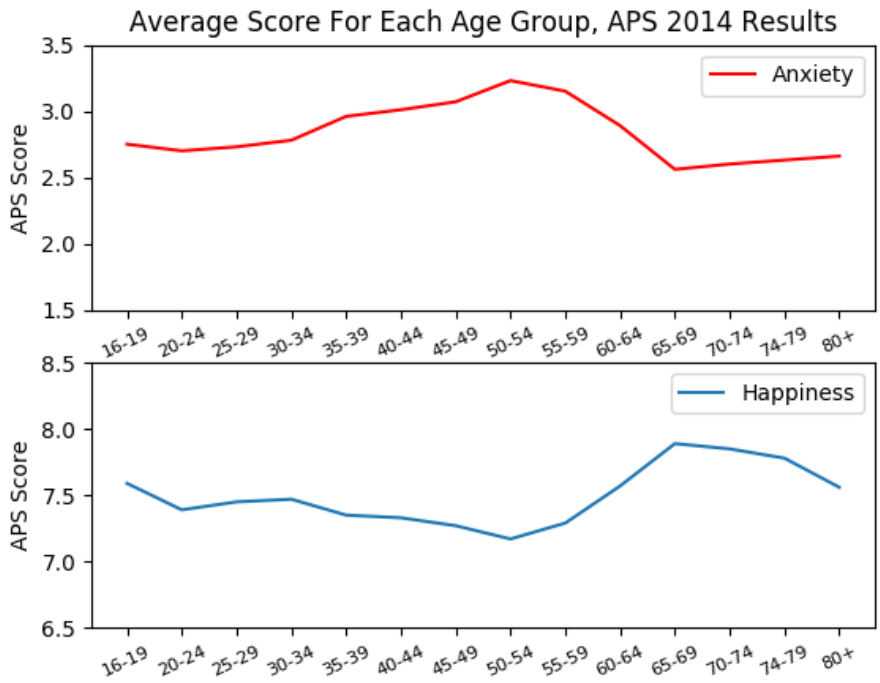
As figure 20 shows, in 2014 approximately 50% of London's population were aged between 16 and 39. This is comparatively more than the UK (approximately 38%), and it is double the proportion of people (approximately 25%) aged between 16 and 39 that responded to the APS in 2014. Additionally as figure 19 indicates, in 2014 at least 80% of people between the ages of 16 and 34, and approximately 70% of people between the age of 35 and 44, used the internet for social media. This suggests that the difference in results between Twitter and the APS for London could be explained by the fact that Twitter is measuring the wellbeing for mostly young people, who make up a large proportion of London's population, and that the APS does not account for as many young people.

Furthermore, as shown in figure 21[5], the 2014 APS results reveal that people between the ages of 16 and 34, consequently the same people who use social media the most frequently, have lower levels of anxiety and higher levels of happiness than older ages, who make up a larger percentage of the APS respondents (until the approximate age of 60, however according to figure 19 far fewer people over the age of 60 use the

[5] Data obtained from the ONS website. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/wellbeing/datasets/personalwellbeingestimatespersonalcharacteristics [Accessed : 23 August 2017]

internet for social media). This complements the hypothesis that Twitter and the APS may be measuring the wellbeing of different age groups for the case of London.

Figure 21: Average score for each age group, APS 2014 results.

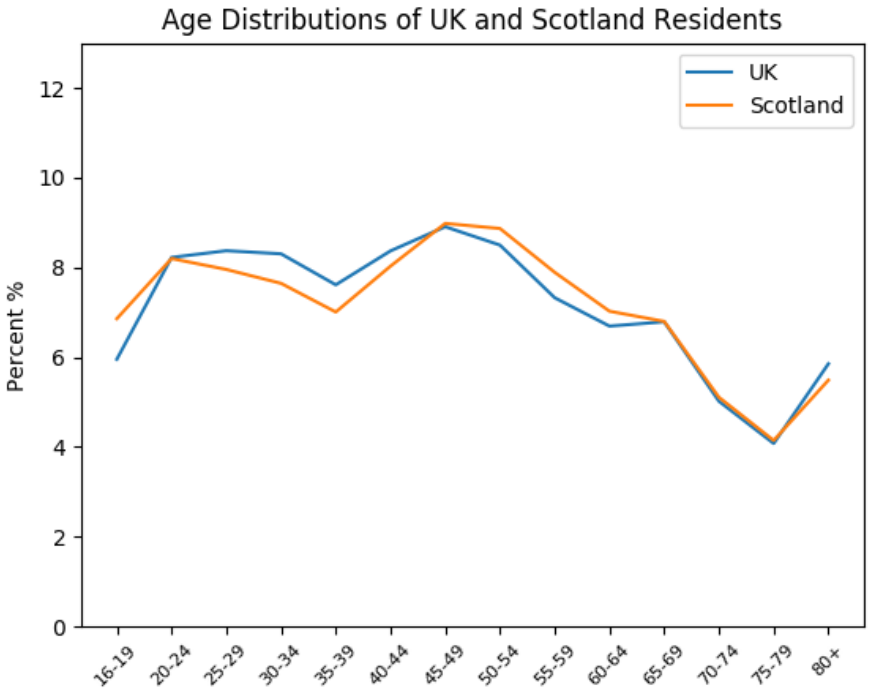


6.3.2 Scotland

Scotland was another outlier in the Twitter analysis results. However, in contrast to London, the mean percentage of positive tweets in Scotland was the lowest out of all regions. This was inconsistent with the results from the APS which indicated that Scotland had relatively high wellbeing results across all four questions. Figure 22[6] illustrates the age distribution of people living in Scotland, compared to the UK, in 2014.

[6] Data obtained from the ONS website. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/populationandmigration/populationestimates/datasets/populationestimatesforukenglandandwalesscotlandandnorthernireland [Accessed: 23 August 2017]

Figure 22: Age distributions of UK and Scotland residents.



It is evident that the age distributions in the UK and Scotland are very similar. Therefore it is unlikely that the difference in outcomes between the Twitter and APS results are caused by a variation in age.

A potential reason for Scotland having a lower percentage of positive tweets could be caused by the fact that 2014 was the year of the Scottish referendum. This may have influenced an increase in users expressing negative opinions via Twitter, which would then have resulted in a higher percentage of negative tweets in 2014.

Additionally, is important to realise that within Scotland there are multiple dialects, each consisting of different words, phrases and punctuation (Scots Language Centre, [no date]). There is a high probability that Scottish tweets contain words and phrases from these dialects, which perhaps may have been undetectable by the classifiers used (recall that the classifiers rely on the vocabulary created from the pre-classified dataset).

6.4 Limitations and Suggestions for Future Research

Whilst social media data provides a variety of benefits over survey-based data, there were a number of limitations which are discussed below, along with suggested future research that could potentially help to overcome these limitations.

It has been shown that sarcasm is extremely difficult to detect using sentiment analysis (Rajadesingan et al., 2015 and Martin, R. 2016), therefore there is a chance that tweets were misclassified as a result of sarcasm not being detected. Additionally, over dramatic updates may have been misclassified, for example, 'I'm so depressed that love island has finished' is most likely an over-exaggeration, rather than a true indication of depression. By creating a classifier that can detect such tweets, although it would be difficult, would increase the accuracy of the results.

Additionally, after more in depth investigations into Scotland and London to identify the cause of discrepancy, adaptations could be made to the classifier to accommodate for the differences. For example, the classifier could be tailored to recognise Scottish jargon, or to account for the surplus of young people or tourists posting tweets in London.

Furthermore, even if there is no reason for these differences in results, the analysis of Twitter data could potentially be used as a totally independent method for measuring wellbeing. For example, comparing regional Twitter results to previous corresponding Twitter results would enable changes to be identified over for that region. This would be particularly useful to analyse how wellbeing, determined by Twitter, changes for short-term events such as rugby matches, festivals, tragedies such as the Grenfell fire, etc.

Without analysing Twitter data directly from the APS respondents, it is difficult to make precise comparisons between a person's self-reported wellbeing and the sentiment of their tweets. Taking a similar approach to Yang and Srinivasan (2016) and analysing people's tweets who have completed the survey, would allow for a more accurate analogy.

Due to both the time limit of the project and the absence of a budget, the classifier that was created was, although accurate, quite simple. By using a service such as Amazon's Mechanical Turk, a similar approach to Schwartz et al. (2016) could be employed where tweets could be hand-classified according to aspects of the wellbeing questions asked in the APS, i.e. does the tweet show happiness, anxiety, etc. This would enable classified tweets to be more comparable to the APS results.

Again due to the time limit, only 700 tweets per day were analysed (for each region), which was approximately 2% of the dataset. By analysing a higher proportion of the dataset, new and more accurate relationships may emerge. Additionally, by separating the Twitter dataset up into smaller areas which match the areas monitored by the APS, such as local authorities, there would be more data to compare to the APS results which could provide more of an insight. For instance, perhaps there is only one area of London or Scotland causing these results to be outliers.

Lastly, the Twitter dataset that was used consisted of tweets recorded over seven months. This was compared to an entire year of APS results. Furthermore, the seven months of Twitter data that were analysed, corresponded to the first seven months of the year monitored by the APS. Therefore almost half of the results in the APS were measured at a completely different time to the tweets. Comparing results over the same time period would have been a more accurate method. Future work could therefore include obtaining the remaining five months to analyse and re-comparing the outcomes.

6.5 Summary

Whilst speculations can be made on each outlier from the data, there are a variety of factors which contribute to the complex and multidimensional issue that is wellbeing, which are simply undetectable through social media. For example, education, marital status, income, participation in exercise and access to greenery are all examples of elements that have been shown to influence subjective wellbeing (Dolan et al., 2008 and Pretty et al., 2007).

Therefore it must be realised that while social media data enables aspects of wellbeing to be detected, similar to most other survey-based methods, on its own it does not allow for a complete assessment of national or subjective wellbeing. However, by using social media as an additional measure, it could provide a more complete overview. With national wellbeing being of utmost importance, the more information that can be obtained, the better.

7. Conclusion

This project aimed to identify whether subjective wellbeing can be determined by social media data. The importance of measuring national wellbeing has been emphasized, and existing methods for measuring subjective wellbeing using both traditional survey-based approaches and modern approaches have been reviewed.

The particular approach taken for this project was machine learning. By creating a combined classifier which incorporates multiple different classifiers, an accurate model was established. After applying this model to a Twitter dataset from 2014, the percentage of positive tweets were determined for each region of the UK. These results were then compared to the results from the Annual Population Survey (APS) in 2014.

The outcome indicates that aspects of subjective wellbeing can be determined from Twitter data for the majority or regions within the UK, but not all. London and Scotland were significant outliers within the Twitter data, and were inconsistent with the results from the APS. Speculations were made on these discrepancies, and further research was suggested in order to establish exactly why these inconsistencies occurred.

Whilst replacing traditional methods for measuring wellbeing may not be a viable solution, a number of suggestions were proposed in this study that show how social media data can be used to supplement existing methods. The use of social media data would enable a timelier and more geographically localised method for measuring wellbeing.

References

Alatartseva, E. and Barysheva, G. 2015. *Well-being: subjective and objective aspects*. Procedia - Social and Behavioural Sciences 166, pp. 36-42.

Albuquerque, B. 2010. *What is Subjective Well-Being? Understanding and Measuring Subjective Well-Being.* [Online]. Available at: http://positivepsychology.org.uk/subjective-well-being/ [Accessed: 6 August 2017]

Aslam, S. 2017. *Twitter by the Numbers: Stats, Demographics & Fun Facts.* [Online]. Available at: https://www.omnicoreagency.com/twitter-statistics/ [Accessed: 5 August 2017]

Asur, S. and Huberman, B.A. 2010. *Predicting the future with social media.* Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference 1, pp. 492-499.

Bannister, K. 2015. *Understanding Sentiment Analysis: What It Is & Why It's Used.* [Online]. Available at: https://www.brandwatch.com/blog/understanding-sentiment-analysis/ [Accessed: 3 August 2017]

BBC. 2016. *Can social media be used to predict election results?* [Online]. Available at: http://www.bbc.co.uk/news/election-us-2016-37942842 [Accessed: 24 August 2017]

BBC News. 2011. *Q&A: What is GDP?.* [Online]. Available at: http://www.bbc.co.uk/news/business-13200758 [Accessed: 23 July 2017]

Beaumont, J. 2011. *Measuring National Well-being: A discussion paper on domains and measures.* [Online]. Available at: http://webarchive.nationalarchives.gov.uk/20160106195224/http://www.ons.gov.uk/ons/dcp171766_240726.pdf [Accessed: 7 August 2017]

Bird, S., Klein, E. and Loper, E. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Bollen, J., Mao, H. and Pepe, A., 2011. *Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena.* ICWSM 11, pp. 450-453.

Bollen, J., Mao, H. and Zeng, X. 2011. *Twitter Mood Predicts the Stock Market*. Journal of Computational Science 2(1), pp. 1-8.

Bradburn, N.M. 1969. *The structure of psychological well-being.* Chicago: Aldine Pub. Co.

Brownlee, J. 2013. *A Tour of Machine Learning Algorithms.* [Online]. Available at: http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ [Accessed: 22 August 2017]

Brownlee, J. 2016. *Linear Regression Tutorial Using Gradient Descent or Machine Learning.* [Online]. Available at: http://machinelearningmastery.com/linear-regression-tutorial-using-gradient-descent-for-machine-learning/ [Accessed: 25 July 2017]

Brownlee, J. 2016. *Logistic Regression for Machine Learning.* [Online]. Available at: http://machinelearningmastery.com/logistic-regression-for-machine-learning/ [Accessed: 25 July 2017]

Brownlee, J. 2016. *Logistic Regression Tutorial for Machine Learning.* [Online]. Available at: http://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learning/ [Accessed: 25 July 2017]

Brownlee, J. 2016. *Gradient Descent for Machine Learning.* [Online]. Available at: http://machinelearningmastery.com/gradient-descent-for-machine-learning/ [Accessed: 25 July 2017]

Cameron, D. 2010. '*PM Speech on Wellbeing', transcript.* Available at: https://www.gov.uk/government/speeches/pm-speech-on-wellbeing [Accessed: 23 July 2017]

Choy, M., Cheong, M.L., Laik, M.N. and Shung, K.P. 2011. *A sentiment analysis of Singapore Presidential Election 2011 using Twitter data with census correction.* arXiv preprint arXiv:1108.5520.

Choy, M., Cheong, M., Laik, M.N. and Shung, K.P. 2012. *US presidential election 2012 prediction using census corrected Twitter model*. arXiv preprint arXiv:1211.0938.

Day, C. 2015. *The Importance of Sentiment Analysis in Social Media Analysis.* [Online]. Available at: https://www.linkedin.com/pulse/importance-sentiment-analysis-social-media-christine-day [Accessed: 3 August 2017]

Diener, E., Emmons, R.A, Larsen, R.J and Griffin, S. 1985. *The Satisfaction With Life Scale*. Journal of Personality Assessment 49(1), pp. 71-75.

Diener, E., Oishi, S. and Lucas, R.E., 2003. *Personality, culture, and subjective well-being: Emotional and cognitive evaluations of life*. Annual review of psychology 54(1), pp. 403-425.

Dolan, P., Peasgood, T. and White, M. 2008. *Do we really know what makes us happy? A review of the economic literature on the factors associated with subjective well-being.* Journal of economic psychology 29(1), pp. 94-122.

Eichstaedt, J. 2014. '*What Social Media Communicates about the Well-being of Society', transcript.* Available at: http://knowledge.wharton.upenn.edu/article/social-media-and-social-well-being/ [Accessed: 28 August 2017]

Evans, J., Macrory, I. and Randall, C. 2015. *Measuring national well-being: Life in the UK, 2015.* Office for National Statistics.

Hern, A. 2014. *Science Reveals Why You're So Addicted To Twitter.* [Online]. Available at: http://uk.businessinsider.com/science-reveals-why-youre-so-addicted-to-twitter-2014-11?r=US&IR=T [Accessed: 27 August 2017]

Hoskin, R. 2012. *The dangers of self-report.* [Online]. Available at: http://www.sciencebrainwaves.com/the-dangers-of-self-report/ [Accessed: 8 August 2017]

Hutto, C.J. and Gilbert, E. 2014. *Vader: A parsimonious rule-based model for sentiment analysis of social media text*. AAAI Publications. Eighth international AAAI conference on weblogs and social media.

Investopedia. [no date]. *Gross Domestic Product - GDP.* [Online]. Avavilable at: http://www.investopedia.com/terms/g/gdp.asp [Accessed: 23 July 2017]

Jashinsky, J., Burton, S.H., Hanson, C.L., West, J., Giraud-Carrier, C., Barnes, M.D. and Argyle, T. 2014. *Tracking suicide risk factors through Twitter in the US*. Crisis 35, pp. 51-59.

Liu, P., Tov, W., Kosinski, M., Stillwell, D.J. and Qiu, L. 2015. *Do Facebook Status Updates Reflect Subjective Well-Being?* Cyberpsychology, Behavior, and Social Networking 18(7), pp. 373-379.

Liu, T., Ding, X., Chen, Y., Chen, Y. and Guo, M. 2016. *Predicting movie Box-office revenues by exploiting large-scale social media content.* Multimedia Tools and Applications 75(3), pp. 1509-1528.

Lyubomirsky, S. and Lepper, H.S. 1999. *A Measure of Subjective Happiness: Preliminary Reliability and Construction Validation.* Social Indicators Research 46(2), pp. 137-155.

Manning, C. et al. 2008. *Introduction to information retrieval.* Cambridge [u.a.]: Cambridge University Press.

Martin, R. 2016. *Can sentiment analysis detect sarcasm on the internet? Oh yeah, totally….* [Online]. Available at: http://blog.infegy.com/sentiment-analysis-and-sarcasm [Accessed: 27 August 2017]

Maslow, A.H. 1943. *A theory of human motivation.* Psychological Review 50(4), pp. 370-396.

Matheson, J., 2011. *National Statistician's Reflections on the National Debate on Measuring National Well-being.* Office for National Statistics.

McLeod, S. 2016. *Maslow's Hierarchy of Needs*. [Online]. Available at: https://www.simplypsychology.org/maslow.html [Accessed: 24 July 2017]

Mittal, A. and Goel, A., 2012. *Stock prediction using twitter sentiment analysis.* Stanford University Working Paper.

Mohammad, S.M. and Kiritchenko, S. 2015. *Using hashtags to capture fine emotion categories from tweets.* Computational Intelligence 31(2), pp. 301-326.

Naji, I. 2012. *Twitter Sentiment Analysis Training Corpus (Dataset).* Available at: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/ [Acessed: 16 August 2017]

Naji, I. 2013. *10 Tips to Improve your Text Classification Algorithm Accuracy and Performance.* [Online]. Available at: http://thinknook.com/10-ways-to-improve-your-classification-algorithm-performance-2013-01-21/ [Accessed: 16 August 2017]

Nations, D. 2017. *What is Microblogging?* [Online]. Available at: https://www.lifewire.com/g00/what-is-microblogging-3486200?i10c.referrer=https%3A%2F%2Fwww.google.co.uk%2F [Accessed: 5 August 2017]

O'Dea, B., Wan, S., Batterham, P.J., Calear, A.L., Paris, C. and Christensen, H. 2015. *Detecting suicidality on Twitter*. Internet Interventions 2(2), pp. 183-188.

ONS. 2014. *Internet Access - Households and Individuals: 2014.* [Online]. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/householdcharacteristics/homeinternetandsocialmediausage/bulletins/internetaccesshouseholdsandindividuals/2014-08-07 [Accessed: 27 August 2017]

ONS. 2011. *Initial investigation into Subjective Wellbeing from the Opinions Survey.* Office for National Statistics.

ONS. 2017. *Personal well-being in the UK: Oct 2015 to Sept 2016.* [Online]. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/wellbeing/bulletins/measuringnationalwellbeing/oct2015tosept2016 [Accessed: 9 August 2017]

ONS. [no date]. *About us.* [Online]. Available at: https://www.ons.gov.uk/aboutus [Accessed: 8 August 2017]

ONS. 2012. *Quality and Methodology Information (QMI): Annual population survey (APS) QMI.* [Online]. Available at: https://www.ons.gov.uk/employmentandlabourmarket/peopleinwork/employmentandemployeetypes/qmis/annualpopulationsurveyapsqmi [Accessed: 8 August 2017]

ONS. 2015. *Measuring National Well-being: Personal Well-being in the UK, 2014-2015*. [Online]. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/wellbeing/bulletins/measuringnationalwellbeing/2015-09-23 [Accessed: 29 August 2017]

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J. 2011. *Scikit-learn: Machine learning in Python*. Journal of Machine Learning Research, pp. 2825-2830.

Pretty, J., Peacock, J., Hine, R., Sellens, M., South, N. and Griffin, M. 2007. *Green exercise in the UK countryside: Effects on health and psychological well-being, and*

implications for policy and planning. Journal of environmental planning and management 50(2), pp. 211-231.

Rajadesingan, A., Zafarani, R. and Liu, H. 2015. *Sarcasm detection on twitter: A behavioral modeling approach*. Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, pp. 97-106.

Raschka, S. 2014. *Naïve Bayes and Text Classification*. [Online]. Available at: http://sebastianraschka.com/Articles/2014_naive_bayes_1.html [Accessed: 22 July 2017]

Roberts, K., Roach, M.A., Johnson, J., Guthrie, J. and Harabagiu, S.M, 2012. *EmpaTweet: Annotating and Detecting Emotions on Twitter*. LREC 12, pp. 3806-3813.

Ryan, R.M. and Deci, E.L. 2001. *On Happiness And Human Potentials: A Review of Research on Hedonic and Eudaimonic Well-Being.* Annual Review of Psychology 52(1), pp. 141-166.

Sayad, S. 2017. *Naïve Bayesian*. [Online]. Available at: http://www.saedsayad.com/naive_bayesian.htm [Accessed: 22 July 2017]

Scots Language Centre. [no date]. *The Main Dialects of Scots.* [Online]. Available at http://www.scotslanguage.com/Scots_Dialects_uid117/The_Main_Dialects_of_Scots [Accessed: 27 August 2017]

Schmidt, M. 2016. *Argmax and Max Calculus*. [Online]. Available at: https://www.cs.ubc.ca/~schmidtm/Documents/2016_540_Argmax.pdf [Accessed: 22 July 2017]

Schneider, N. 2016. *Happiness vs Well Being.* [Online]. Available at: http://www.globalnlptraining.com/blog/happiness-vs-well-being/ [Accessed: 6 August 2017]

Schwartz, H.A., Sap, M., Kern, M.L., Eichstaedt, J.C., Kapelner, A., Agrawal, M., Blanco, E., Dziurzynski, L., Park, G., Stillwell, D. and Kosinski, M. 2016. *Predicting individual well-being through the language of social media*. Biocomputing 2016: Proceedings of the Pacific Symposium, pp. 516-527.

Seligman, M. 2011. *Flourish*. New York: Free Press.

Sentdex. 2015. *NLTK with Python 3 for Natural Language Processing.* [video online]. Available at: https://www.youtube.com/playlist?list=PLQVvvaa0QuDf2JswnfiGkliBInZnIC4HL [Accessed: 16 August 2017]

Shenton, C., Siegler, V., Tinkler, L. and Hicks, S. 2012. *Analysis of Experimental Subjective Well-being Data from the Annual Population Survey, April to September 2011*. Office for National Statistics.

Singh, K. and Jha, S.D. 2008. *Positive and negative affect, and grit as predictors of happiness and life satisfaction.* Journal of the Indian Academy of Applied Psychology 34(2), pp. 40-45.

Skula, S. 2017. *How an Indian AI startup predicted Trump victory.* [Online]. Available at: https://www.geospatialworld.net/article/artificial-intelligence-system-mogia/ [Accessed: 24 August 2017]

Smith, C. and Clay, P. 2010. *Measuring Subjective and Objective Well-being: Analyses from Five Marine Commercial Fisheries.* Human Organization 69(2), pp. 158-168

Soulo, T. 2014. *What Do People Tweet About & The Surprising Truth About What Drives Them.* [Online]. Available at: http://bloggerjet.com/what-do-people-tweet-about/ [Accessed: 27 August 2017]

Szalavitz, M. 2012. *Is Twitter Really More Addictive than Alcohol? The Vagaries of Will and Desire.* [Online]. Available at: http://healthland.time.com/2012/02/07/is-

twitter-really-more-addictive-than-alcohol-the-vagaries-of-will-and-desire/ [Accessed: 27 August 2017]

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

Ura, K., Alkire, S., Zangmo, T. and Wangdi, K. 2012. *A Short Guide to Gross National Happiness Index*. The Centre for Bhutan Studies.

Vryniotis, V. 2013. *Machine Learning Tutorial: The Naïve Bayes Text Classifier | Datumbox.* [Online]. Available at: http://blog.datumbox.com/machine-learning-tutorial-the-naive-bayes-text-classifier/ [Accessed: 22 July 2017]

Waldron, S. 2010. *Measuring subjective wellbeing in the UK*. Office for National Statistics.

Waterman, A.S, Schwartz, S.J. and Conti R. 2008. *The Implications of Two Conceptions of Happiness (Hedonic Enjoyment and Eudaimonia) for the Understanding of Intrinsic Motivation.* Journal of Happiness Studies 9(1): 41-79.

Wikipedia. 2017. *List of postcode areas in the United Kingdom.* [Online]. Available at: https://en.wikipedia.org/wiki/List_of_postcode_areas_in_the_United_Kingdom [Accessed: 22 August 2017]

Yang, C. and Srinivasan, P. 2016. *Life satisfaction and the pursuit of happiness on Twitter.* PLoS ONE 11(3): e0150881.

Appendix 1: Formatting the Pre-Classified Dataset Code

```
import csv
import random
import pickle

all_data = []

with open('Sentiment Analysis Dataset.csv') as f:
   read = csv.reader(f, delimiter=',')
   for row in read:
       newrow = (row[3], row[1].replace('0', 'neg').replace('1',
'pos'))
       all_data.append(newrow)
all_data.pop(0)

all_data_save = open('all_data.pickle', 'wb')
pickle.dump(all_data, all_data_save)
all_data_save.close()

positive = []
negative = []

for item in all_data:
   if item[1] == "pos":
       positive.append(item)
   if item[1] == "neg":
       negative.append(item)

random.shuffle(positive)
random.shuffle(negative)

random_pos = positive[:6000]
random_neg = negative[:6000]

train_data = []
test_data = []

for item in random_pos[:5000]:
   if item not in train_data:
       train_data.append(item)
for item in random_neg[:5000]:
   if item not in train_data:
       train_data.append(item)

for item in random_pos[5000:]:
   if item not in train_data:
       if item not in test_data:
           test_data.append(item)
for item in random_neg[5000:]:
   if item not in train_data:
       if item not in test_data:
           test_data.append(item)

train_data_save = open("train_data.pickle", "wb")
pickle.dump(train_data, train_data_save)
train_data_save.close()
```

```
test_data_save = open("test_data.pickle", "wb")
pickle.dump(test_data, test_data_save)
test_data_save.close()
```

Appendix 2: Unigrams without Part of Speech Tagging Code

```
import pickle
import nltk
import random

from nltk.classify.scikitlearn import SklearnClassifier
from nltk.tokenize import word_tokenize

from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC

test_data_f = open('test_data.pickle', 'rb')
test_data = pickle.load(test_data_f)
test_data_f.close()

train_data_f = open('train_data.pickle', 'rb')
train_data = pickle.load(train_data_f)
train_data_f.close()

all_words = []

for item in train_data:
   words = word_tokenize(item[0])
   for word in words:
       all_words.append(word.lower())

all_words = nltk.FreqDist(all_words)

vocab = []
words_for_vocab = all_words.most_common()
for item in words_for_vocab:
   if item[1] >= 2:
       vocab.append(item[0].lower())

def dictionary_of_words(text):
   words = word_tokenize(text.lower())
   Dictionary = {}
   for word in vocab:
       Dictionary[word] = (word in words)
   return Dictionary

test_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in test_data]
train_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in train_data]

### CLASSIFIERS
```

```
# Original Naive Bayes
NB_Classifier = nltk.NaiveBayesClassifier.train(train_set) # Train
set
print("Original Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(NB_Classifier, test_set))*100, "%") # Test
set

# Multinomial Naive Bayes
MNB_Classifier = SklearnClassifier(MultinomialNB())
MNB_Classifier.train(train_set)
print("Multinomial Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(MNB_Classifier, test_set))*100, "%")

# Bernoulli Naive Bayes
BNB_Classifier = SklearnClassifier(BernoulliNB())
BNB_Classifier.train(train_set)
print("Bernoulli Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(BNB_Classifier, test_set))*100, "%")

# Logistic Regression
LR_Classifier = SklearnClassifier(LogisticRegression())
LR_Classifier.train(train_set)
print("Logistic Regression Classifier accuracy: ",
(nltk.classify.accuracy(LR_Classifier, test_set))*100, "%")

# Support Vector Classifier
SV_Classifier = SklearnClassifier(SVC())
SV_Classifier.train(train_set)
print("Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(SV_Classifier, test_set))*100, "%")

# Nu Support Vector Classifier
NuSV_Classifier = SklearnClassifier(NuSVC())
NuSV_Classifier.train(train_set)
print("Nu Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(NuSV_Classifier, test_set))*100, "%")

# Linear Support Vector Classifier
LSV_Classifier = SklearnClassifier(LinearSVC())
LSV_Classifier.train(train_set)
print("Linear Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(LSV_Classifier, test_set))*100, "%")

# Stochastic Gradient Descent Classifier
SGD_Classifier = SklearnClassifier(SGDClassifier())
SGD_Classifier.train(train_set)
print("Stochastic Gradient Descent Classifier accuracy: ",
(nltk.classify.accuracy(SGD_Classifier, test_set))*100, "%")
```

Appendix 3: Unigrams with Part of Speech Tagging Code

```
import pickle
import nltk
import random

from nltk.classify.scikitlearn import SklearnClassifier
```

```
from nltk.tokenize import word_tokenize

from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC


test_data_f = open('test_data.pickle', 'rb')
test_data = pickle.load(test_data_f)
test_data_f.close()

train_data_f = open('train_data.pickle', 'rb')
train_data = pickle.load(train_data_f)
train_data_f.close()

all_words = []
allowed_word_types = ["J", "R", "V"] # adjective, adverb, verb

for item in train_data:
   first_item = item[0]
   words = word_tokenize(first_item)
   POS = nltk.pos_tag(words)
   for word in POS:
       if len(word[0]) > 3:
           if word[1][0] in allowed_word_types:
               all_words.append(word[0].lower())

all_words = nltk.FreqDist(all_words)

vocab = []
words_for_vocab = all_words.most_common()
for item in words_for_vocab:
   if item[1] >= 2:
       vocab.append(item[0].lower())

def dictionary_of_words(text):
   words = word_tokenize(text.lower())
   Dictionary = {}
   for word in vocab:
       Dictionary[word] = (word in words)
   return Dictionary

test_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in test_data]
train_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in train_data]

### CLASSIFIERS

# Original Naive Bayes
NB_Classifier = nltk.NaiveBayesClassifier.train(train_set) # Train
set
print("Original Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(NB_Classifier, test_set))*100, "%") # Test
set

# Multinomial Naive Bayes
MNB_Classifier = SklearnClassifier(MultinomialNB())
```

```
MNB_Classifier.train(train_set)
print("Multinomial Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(MNB_Classifier, test_set))*100, "%")

# Bernoulli Naive Bayes
BNB_Classifier = SklearnClassifier(BernoulliNB())
BNB_Classifier.train(train_set)
print("Bernoulli Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(BNB_Classifier, test_set))*100, "%")

# Logistic Regression
LR_Classifier = SklearnClassifier(LogisticRegression())
LR_Classifier.train(train_set)
print("Logistic Regression Classifier accuracy: ",
(nltk.classify.accuracy(LR_Classifier, test_set))*100, "%")

# Support Vector Classifier
SV_Classifier = SklearnClassifier(SVC())
SV_Classifier.train(train_set)
print("Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(SV_Classifier, test_set))*100, "%")

# Nu Support Vector Classifier
NuSV_Classifier = SklearnClassifier(NuSVC())
NuSV_Classifier.train(train_set)
print("Nu Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(NuSV_Classifier, test_set))*100, "%")

# Linear Support Vector Classifier
LSV_Classifier = SklearnClassifier(LinearSVC())
LSV_Classifier.train(train_set)
print("Linear Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(LSV_Classifier, test_set))*100, "%")

# Stochastic Gradient Descent Classifier
SGD_Classifier = SklearnClassifier(SGDClassifier())
SGD_Classifier.train(train_set)
print("Stochastic Gradient Descent Classifier accuracy: ",
(nltk.classify.accuracy(SGD_Classifier, test_set))*100, "%")
```

Appendix 4: Bigrams without Part of Speech Tagging Code

```
import pickle
import nltk
import random

from nltk.classify.scikitlearn import SklearnClassifier
from nltk.tokenize import word_tokenize

from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC

test_data_f = open('test_data.pickle', 'rb')
test_data = pickle.load(test_data_f)
test_data_f.close()
```

```
train_data_f = open('train_data.pickle', 'rb')
train_data = pickle.load(train_data_f)
train_data_f.close()

all_words = []

for item in train_data:
   bigrm = list(nltk.bigrams(item[0].split()))
   for i in bigrm:
       all_words.append(i[0].lower() + " " + i[1].lower())

all_words = nltk.FreqDist(all_words)

vocab = []
words_for_vocab = all_words.most_common()
for item in words_for_vocab:
   if item[1] >= 2:
       vocab.append(item[0].lower())

def dictionary_of_words(text):
   words = []
   bigrm = list(nltk.bigrams(text.split()))
   for item in bigrm:
       words.append(item[0].lower() + " " + item[1].lower())
   Dictionary = {}
   for word in vocab:
       Dictionary[word] = (word in words)
   return Dictionary

test_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in test_data]
train_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in train_data]

### CLASSIFIERS

# Original Naive Bayes
NB_Classifier = nltk.NaiveBayesClassifier.train(train_set) # Train
set
print("Original Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(NB_Classifier, test_set))*100, "%") # Test
set

# Multinomial Naive Bayes
MNB_Classifier = SklearnClassifier(MultinomialNB())
MNB_Classifier.train(train_set)
print("Multinomial Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(MNB_Classifier, test_set))*100, "%")

# Bernoulli Naive Bayes
BNB_Classifier = SklearnClassifier(BernoulliNB())
BNB_Classifier.train(train_set)
print("Bernoulli Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(BNB_Classifier, test_set))*100, "%")

# Logistic Regression
LR_Classifier = SklearnClassifier(LogisticRegression())
```

```
LR_Classifier.train(train_set)
print("Logistic Regression Classifier accuracy: ",
(nltk.classify.accuracy(LR_Classifier, test_set))*100, "%")

# Support Vector Classifier
SV_Classifier = SklearnClassifier(SVC())
SV_Classifier.train(train_set)
print("Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(SV_Classifier, test_set))*100, "%")

# Nu Support Vector Classifier
NuSV_Classifier = SklearnClassifier(NuSVC())
NuSV_Classifier.train(train_set)
print("Nu Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(NuSV_Classifier, test_set))*100, "%")

# Linear Support Vector Classifier
LSV_Classifier = SklearnClassifier(LinearSVC())
LSV_Classifier.train(train_set)
print("Linear Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(LSV_Classifier, test_set))*100, "%")

# Stochastic Gradient Descent Classifier
SGD_Classifier = SklearnClassifier(SGDClassifier())
SGD_Classifier.train(train_set)
print("Stochastic Gradient Descent Classifier accuracy: ",
(nltk.classify.accuracy(SGD_Classifier, test_set))*100, "%")
```

Appendix 5: Bigrams with Part of Speech Tagging Code

```
import pickle
import nltk
import random

from nltk.classify.scikitlearn import SklearnClassifier
from nltk.tokenize import word_tokenize

from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC

test_data_f = open('test_data.pickle', 'rb')
test_data = pickle.load(test_data_f)
test_data_f.close()

train_data_f = open('train_data.pickle', 'rb')
train_data = pickle.load(train_data_f)
train_data_f.close()

all_words = []
allowed_word_types = ["J", "R", "V"] # adjective, adverb, verb

for item in train_data:
   bigrm = list(nltk.bigrams(item[0].split()))
   for i in bigrm:
       POS = nltk.pos_tag(i)
```

```
        for word in POS:
            if len(word[0]) > 3:
                if word[1][0] in allowed_word_types:
                    all_words.append(i[0].lower() + " " +
i[1].lower())

all_words = nltk.FreqDist(all_words)

vocab = []
words_for_vocab = all_words.most_common()
for item in words_for_vocab:
   if item[1] >= 2:
       vocab.append(item[0].lower())

def dictionary_of_words(text):
   words = []
   bigrm = list(nltk.bigrams(text.split()))
   for item in bigrm:
       words.append(item[0].lower() + " " + item[1].lower())
   Dictionary = {}
   for word in vocab:
       Dictionary[word] = (word in words)
   return Dictionary

test_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in test_data]
train_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in train_data]

### CLASSIFIERS

# Original Naive Bayes
NB_Classifier = nltk.NaiveBayesClassifier.train(train_set) # Train
set
print("Original Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(NB_Classifier, test_set))*100, "%") # Test
set

# Multinomial Naive Bayes
MNB_Classifier = SklearnClassifier(MultinomialNB())
MNB_Classifier.train(train_set)
print("Multinomial Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(MNB_Classifier, test_set))*100, "%")

# Bernoulli Naive Bayes
BNB_Classifier = SklearnClassifier(BernoulliNB())
BNB_Classifier.train(train_set)
print("Bernoulli Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(BNB_Classifier, test_set))*100, "%")

# Logistic Regression
LR_Classifier = SklearnClassifier(LogisticRegression())
LR_Classifier.train(train_set)
print("Logistic Regression Classifier accuracy: ",
(nltk.classify.accuracy(LR_Classifier, test_set))*100, "%")

# Support Vector Classifier
SV_Classifier = SklearnClassifier(SVC())
SV_Classifier.train(train_set)
```

```
print("Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(SV_Classifier, test_set))*100, "%")

# Nu Support Vector Classifier
NuSV_Classifier = SklearnClassifier(NuSVC())
NuSV_Classifier.train(train_set)
print("Nu Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(NuSV_Classifier, test_set))*100, "%")

# Linear Support Vector Classifier
LSV_Classifier = SklearnClassifier(LinearSVC())
LSV_Classifier.train(train_set)
print("Linear Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(LSV_Classifier, test_set))*100, "%")

# Stochastic Gradient Descent Classifier
SGD_Classifier = SklearnClassifier(SGDClassifier())
SGD_Classifier.train(train_set)
print("Stochastic Gradient Descent Classifier accuracy: ",
(nltk.classify.accuracy(SGD_Classifier, test_set))*100, "%")
```

Appendix 6: Saving the Classifiers to Create the Combined Classifier Code

```
import pickle
import nltk
import random

from nltk.classify.scikitlearn import SklearnClassifier
from nltk.tokenize import word_tokenize

from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC

test_data_f = open('test_data.pickle', 'rb')
test_data = pickle.load(test_data_f)
test_data_f.close()

train_data_f = open('train_data.pickle', 'rb')
train_data = pickle.load(train_data_f)
train_data_f.close()

all_words = []

for item in train_data:
   words = word_tokenize(item[0])
   for word in words:
       all_words.append(word.lower())

all_words = nltk.FreqDist(all_words)

vocab = []
words_for_vocab = all_words.most_common()
```

```
for item in words_for_vocab:
   if item[1] >= 2:
       vocab.append(item[0].lower())

save_vocab = open("vocab.pickle", "wb")
pickle.dump(vocab, save_vocab)
save_vocab.close()

def dictionary_of_words(text):
   words = word_tokenize(text.lower())
   Dictionary = {}
   for word in vocab:
       Dictionary[word] = (word in words)
   return Dictionary

test_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in test_data]
train_set = [(dictionary_of_words(item), sentiment) for (item,
sentiment) in train_data]

### CLASSIFIERS

# Original Naive Bayes
classifier = nltk.NaiveBayesClassifier.train(train_set) # Train set
print("Original Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(classifier, test_set))*100, "%") # Test set

save_classifier = open("Original_NB.pickle", "wb")
pickle.dump(classifier, save_classifier)
save_classifier.close()

# Multinomial Naive Bayes
MNB_Classifier = SklearnClassifier(MultinomialNB())
MNB_Classifier.train(train_set)
print("Multinomial Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(MNB_Classifier, test_set))*100, "%")

save_classifier = open("MNB.pickle", "wb")
pickle.dump(MNB_Classifier, save_classifier)
save_classifier.close()

# Bernoulli Naive Bayes
BNB_Classifier = SklearnClassifier(BernoulliNB())
BNB_Classifier.train(train_set)
print("Bernoulli Naive Bayes Classifier accuracy: ",
(nltk.classify.accuracy(BNB_Classifier, test_set))*100, "%")

save_classifier = open("BNB.pickle", "wb")
pickle.dump(BNB_Classifier, save_classifier)
save_classifier.close()

# Logistic Regression
LR_Classifier = SklearnClassifier(LogisticRegression())
LR_Classifier.train(train_set)
print("Logistic Regression Classifier accuracy: ",
(nltk.classify.accuracy(LR_Classifier, test_set))*100, "%")

save_classifier = open("LR.pickle", "wb")
```

```
pickle.dump(LR_Classifier, save_classifier)
save_classifier.close()

# Nu Support Vector Classifier
NuSV_Classifier = SklearnClassifier(NuSVC())
NuSV_Classifier.train(train_set)
print("Nu Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(NuSV_Classifier, test_set))*100, "%")

save_classifier = open("NuSVC.pickle", "wb")
pickle.dump(NuSV_Classifier, save_classifier)
save_classifier.close()

# Linear Support Vector Classifier
LSV_Classifier = SklearnClassifier(LinearSVC())
LSV_Classifier.train(train_set)
print("Linear Support Vector Classifier accuracy: ",
(nltk.classify.accuracy(LSV_Classifier, test_set))*100, "%")

save_classifier = open("LSV.pickle", "wb")
pickle.dump(LSV_Classifier, save_classifier)
save_classifier.close()

# Stochastic Gradient Descent Classifier
SGD_Classifier = SklearnClassifier(SGDClassifier())
SGD_Classifier.train(train_set)
print("Stochastic Gradient Descent Classifier accuracy: ",
(nltk.classify.accuracy(SGD_Classifier, test_set))*100, "%")

save_classifier = open("SGDC.pickle", "wb")
pickle.dump(SGD_Classifier, save_classifier)
save_classifier.close()
```

Appendix 7: Creation of the Combined Classifier Code

```
import pickle
from statistics import mode

from nltk.classify import ClassifierI
from nltk.tokenize import word_tokenize

class Vote_Classifier(ClassifierI):
   def __init__(self, *classifiers):
       self.classifiers = classifiers

   # Takes a list of all results and returns the mode, i.e. most
common
   def classify(self, features):
       votes = []
       for classifier in self.classifiers:
           vote = classifier.classify(features)
           votes.append(vote)
       return mode(votes)

   # Returns the % of classifiers that chose that result
   def confidence(self, features):
```

```
        votes = []
        for classifier in self.classifiers:
            vote = classifier.classify(features)
            votes.append(vote)

        choice = votes.count(mode(votes))
        confidence_ = choice / len(votes)
        return confidence_

vocab_f = open("vocab.pickle", "rb")
vocab = pickle.load(vocab_f)
vocab_f.close()

def dictionary_of_words(text):
   words = word_tokenize(text.lower())
   Dictionary = {}
   for word in vocab:
       Dictionary[word] = (word in words)
   return Dictionary

open_file = open("Original_NB.pickle", "rb")
NB_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("MNB.pickle", "rb")
MNB_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("BNB.pickle", "rb")
BNB_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("LR.pickle", "rb")
LR_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("NuSVC.pickle", "rb")
NuSV_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("LSV.pickle", "rb")
LSV_Classifier = pickle.load(open_file)
open_file.close()

open_file = open("SGDC.pickle", "rb")
SGD_Classifier = pickle.load(open_file)
open_file.close()

vote = Vote_Classifier(
   NB_Classifier,
   BNB_Classifier,
   LR_Classifier,
   MNB_Classifier,
   NuSV_Classifier,
   LSV_Classifier,
   SGD_Classifier)

def sentiment(text):
```

```
    features = dictionary_of_words(text)
    return vote.classify(features), vote.confidence(features)
```

Appendix 8: Testing the Combined Classifier Code

```
import Twitter_Sentiment_Module as TSM
import pickle

test_data_f = open("test_data.pickle", "rb")
test_data = pickle.load(test_data_f)
test_data_f.close()

score_100 = 0
length_100 = 0
score_80 = 0
length_80 = 0
score_70 = 0
length_70 = 0
score_50 = 0
length_50 = 0

for item in test_data:
    if TSM.sentiment(item[0])[1] == 1.0:
        length_100 += 1
        if TSM.sentiment(item[0])[0] == item[1]:
            score_100 += 1
    if TSM.sentiment(item[0])[1] >= 0.8:
        length_80 += 1
        if TSM.sentiment(item[0])[0] == item[1]:
            score_80 += 1
    if TSM.sentiment(item[0])[1] >= 0.7:
        length_70 += 1
        if TSM.sentiment(item[0])[0] == item[1]:
            score_70 += 1
    if TSM.sentiment(item[0])[1] >= 0.5:
        length_50 += 1
        if TSM.sentiment(item[0])[0]== item[1]:
            score_50 += 1

print("100% certain: ", (score_100/length_100)*100, "% ... cases:",
length_100)
print("Over 80% certain: ", (score_80/length_80)*100, "% ...
cases:", length_80)
print("Over 70% certain: ", (score_70/length_70)*100, "% ...
cases:", length_70)
print("Over 50% certain: ", (score_50/length_50)*100, "% ...
cases:", length_50)
```

Appendix 9: Testing VADER Code

```
import pickle
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
```

```
test_data_f = open("test_data.pickle", "rb")
test_data = pickle.load(test_data_f)
test_data_f.close()

score = 0
length = 0
for item in test_data:
   if analyser.polarity_scores(item[0])['compound'] > 0:
       length += 1
       if item[1] == "pos":
           score += 1
   if analyser.polarity_scores(item[0])['compound'] < 0:
       length += 1
       if item[1] == "neg":
           score += 1
print("Not including 'neutral' classifications: ",
(score/length)*100, "%")
print("Including 'neutral' classifications: ",
(score/len(test_data))*100, "%")
```

Appendix 10: Separating the 2014 Twitter Dataset by Region Code

```
import pymongo
from pymongo import MongoClient

client = MongoClient()
db = client.twitter
collection = db.tweets

London = ['W1', 'W2', 'W3', 'W4', 'W5', 'W6', 'W7', 'W8',
         'W9', 'SW', 'SE', 'WC', 'EC', 'E1', 'E2', 'E3',
         'E4', 'E5', 'E6', 'E7', 'E8', 'E9', 'N1', 'N2',
         'N3', 'N4', 'N5', 'N6', 'N7', 'N8', 'N9', 'NW',
         'W0', 'E0', 'N0']

NorthernIreland =  ['BT']

Scotland = ['AB', 'DD', 'DG', 'EH', 'FK', 'G1', 'G2', 'G3',
           'G4', 'G5', 'G6',  'G7', 'G8', 'G9', 'HS', 'IV',
           'KA', 'KW', 'KY', 'ML', 'PH', 'PA', 'ZE', 'G0']

NorthEast = ['DH', 'NE',  'SR', 'TD', 'TS']

NorthWest = ['BB', 'BL', 'CA', 'CW', 'CH', 'FY', 'L1', 'L2',
            'L3', 'L4', 'L5', 'L6', 'L7', 'L8', 'L8', 'L9',
            'L0', 'M0', 'M1', 'M2', 'M3', 'M4',  'M5', 'M6',
            'M7', 'M8', 'M9', 'OL', 'PR', 'WN',  'SK',  'WA',
            'LA']

YorkshireHumber =  ['DL', 'BD', 'DN', 'HD', 'HG', 'HU', 'HX',
                   'LS', 'S0', 'S1', 'S2', 'S3', 'S4', 'S5',
                   'S6', 'S7',  'S8', 'S9', 'WF', 'YO']

EastMidlands = ['DE', 'LE', 'LN', 'NG', 'NN']
```

```
WestMidlands = ['B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7',
                'B8', 'B9', 'CV', 'DY', 'HR', 'ST', 'SY', 'TF',
                'WR', 'WV', 'WS']

East = ['CB', 'SS', 'CM', 'CO', 'RM', 'AL', 'EN', 'SG', 'WD',
       'IP', 'NR', 'PE']

SouthEast = ['OX', 'MK', 'BN', 'BR', 'CR', 'CT', 'DA', 'GU',
            'HP', 'KT', 'ME', 'PO', 'RG', 'RH', 'SL',  'SM',
            'SO', 'TN', 'TW', 'UB']

SouthWest = ['BA', 'BH', 'BS', 'DT', 'EX', 'GL', 'PL', 'SN',
            'SP', 'TA', 'TQ', 'TR']

Wales = ['CF', 'LD',  'LL', 'NP', 'SA']

for item in collection.find():
   postcode = item['tweet']['address']['postcode']
   if postcode[0:2] in London:
       db.London.insert(item)
   elif postcode[0:2] in NorthernIreland:
       db.NorthernIreland.insert(item)
   elif postcode[0:2] in Scotland:
       db.Scotland.insert(item)
   elif postcode[0:2] in NorthEast:
       db.NorthEast.insert(item)
   elif postcode[0:2] in NorthWest:
       db.NorthWest.insert(item)
   elif postcode[0:2] in YorkshireHumber:
       db.YorkshireHumber.insert(item)
   elif postcode[0:2] in EastMidlands:
       db.EastMidlands.insert(item)
   elif postcode[0:2] in WestMidlands:
       db.WestMidlands.insert(item)
   elif postcode[0:2] in East:
       db.East.insert(item)
   elif postcode[0:2] in SouthEast:
       db.SouthEast.insert(item)
   elif postcode[0:2] in SouthWest:
       db.SouthWest.insert(item)
   elif postcode[0:2] in Wales:
       db.Wales.insert(item)
   else:
       db.Unknown.insert(item)
```

Appendix 11: Separating the 2014 Twitter Dataset by Month Code

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

db = client.twitter

LondonCollection = db.London

for item in LondonCollection.find():
   month = item['time']['month']
   if month == 'Apr':
       db.LondonApr.insert(item)
   elif month == 'May':
       db.LondonMay.insert(item)
   elif month == 'Jun':
       db.LondonJun.insert(item)
   elif month == 'Jul':
       db.LondonJul.insert(item)
   elif month == 'Aug':
       db.LondonAug.insert(item)
   elif month == 'Sep':
       db.LondonSep.insert(item)
   elif month == 'Oct':
       db.LondonOct.insert(item)
   else:
       db.LondonUnknown.insert(item)

ScotlandCollection = db.Scotland

for item in ScotlandCollection.find():
   month = item['time']['month']
   if month == 'Apr':
       db.ScotlandApr.insert(item)
   elif month == 'May':
       db.ScotlandMay.insert(item)
   elif month == 'Jun':
       db.ScotlandJun.insert(item)
   elif month == 'Jul':
       db.ScotlandJul.insert(item)
   elif month == 'Aug':
       db.ScotlandAug.insert(item)
   elif month == 'Sep':
       db.ScotlandSep.insert(item)
   elif month == 'Oct':
       db.ScotlandOct.insert(item)
   else:
       db.ScotlandUnknown.insert(item)

NorthEastCollection = db.NorthEast

for item in NorthEastCollection.find():
   month = item['time']['month']
   if month == 'Apr':
       db.NorthEastApr.insert(item)
```

```
    elif month == 'May':
        db.NorthEastMay.insert(item)
    elif month == 'Jun':
        db.NorthEastJun.insert(item)
    elif month == 'Jul':
        db.NorthEastJul.insert(item)
    elif month == 'Aug':
        db.NorthEastAug.insert(item)
    elif month == 'Sep':
        db.NorthEastSep.insert(item)
    elif month == 'Oct':
        db.NorthEastOct.insert(item)
    else:
        db.NorthEastUnknown.insert(item)

NorthWestCollection = db.NorthWest

for item in NorthWestCollection.find():
    month = item['time']['month']
    if month == 'Apr':
        db.NorthWestApr.insert(item)
    elif month == 'May':
        db.NorthWestMay.insert(item)
    elif month == 'Jun':
        db.NorthWestJun.insert(item)
    elif month == 'Jul':
        db.NorthWestJul.insert(item)
    elif month == 'Aug':
        db.NorthWestAug.insert(item)
    elif month == 'Sep':
        db.NorthWestSep.insert(item)
    elif month == 'Oct':
        db.NorthWestOct.insert(item)
    else:
        db.NorthWestUnknown.insert(item)

YorkshireHumberCollection = db.YorkshireHumber

for item in YorkshireHumberCollection.find():
    month = item['time']['month']
    if month == 'Apr':
        db.YorkshireHumberApr.insert(item)
    elif month == 'May':
        db.YorkshireHumberMay.insert(item)
    elif month == 'Jun':
        db.YorkshireHumberJun.insert(item)
    elif month == 'Jul':
        db.YorkshireHumberJul.insert(item)
    elif month == 'Aug':
        db.YorkshireHumberAug.insert(item)
    elif month == 'Sep':
        db.YorkshireHumberSep.insert(item)
    elif month == 'Oct':
        db.YorkshireHumberOct.insert(item)
    else:
        db.YorkshireHumberUnknown.insert(item)

EastMidlandsCollection = db.EastMidlands

for item in EastMidlandsCollection.find():
```

```
    month = item['time']['month']
    if month == 'Apr':
        db.EastMidlandsApr.insert(item)
    elif month == 'May':
        db.EastMidlandsMay.insert(item)
    elif month == 'Jun':
        db.EastMidlandsJun.insert(item)
    elif month == 'Jul':
        db.EastMidlandsJul.insert(item)
    elif month == 'Aug':
        db.EastMidlandsAug.insert(item)
    elif month == 'Sep':
        db.EastMidlandsSep.insert(item)
    elif month == 'Oct':
        db.EastMidlandsOct.insert(item)
    else:
        db.EastMidlandsUnknown.insert(item)

WestMidlandsCollection = db.WestMidlands

for item in WestMidlandsCollection.find():
    month = item['time']['month']
    if month == 'Apr':
        db.WestMidlandsApr.insert(item)
    elif month == 'May':
        db.WestMidlandsMay.insert(item)
    elif month == 'Jun':
        db.WestMidlandsJun.insert(item)
    elif month == 'Jul':
        db.WestMidlandsJul.insert(item)
    elif month == 'Aug':
        db.WestMidlandsAug.insert(item)
    elif month == 'Sep':
        db.WestMidlandsSep.insert(item)
    elif month == 'Oct':
        db.WestMidlandsOct.insert(item)
    else:
        db.WestMidlandsUnknown.insert(item)

EastCollection = db.East

for item in EastCollection.find():
    month = item['time']['month']
    if month == 'Apr':
        db.EastApr.insert(item)
    elif month == 'May':
        db.EastMay.insert(item)
    elif month == 'Jun':
        db.EastJun.insert(item)
    elif month == 'Jul':
        db.EastJul.insert(item)
    elif month == 'Aug':
        db.EastAug.insert(item)
    elif month == 'Sep':
        db.EastSep.insert(item)
    elif month == 'Oct':
        db.EastOct.insert(item)
    else:
        db.EastUnknown.insert(item)
```

```
SouthEastCollection = db.SouthEast

for item in SouthEastCollection.find():
   month = item['time']['month']
   if month == 'Apr':
       db.SouthEastApr.insert(item)
   elif month == 'May':
       db.SouthEastMay.insert(item)
   elif month == 'Jun':
       db.SouthEastJun.insert(item)
   elif month == 'Jul':
       db.SouthEastJul.insert(item)
   elif month == 'Aug':
       db.SouthEastAug.insert(item)
   elif month == 'Sep':
       db.SouthEastSep.insert(item)
   elif month == 'Oct':
       db.SouthEastOct.insert(item)
   else:
       db.SouthEastUnknown.insert(item)

SouthWestCollection = db.SouthWest

for item in SouthWestCollection.find():
   month = item['time']['month']
   if month == 'Apr':
       db.SouthWestApr.insert(item)
   elif month == 'May':
       db.SouthWestMay.insert(item)
   elif month == 'Jun':
       db.SouthWestJun.insert(item)
   elif month == 'Jul':
       db.SouthWestJul.insert(item)
   elif month == 'Aug':
       db.SouthWestAug.insert(item)
   elif month == 'Sep':
       db.SouthWestSep.insert(item)
   elif month == 'Oct':
       db.SouthWestOct.insert(item)
   else:
       db.SouthWestUnknown.insert(item)

WalesCollection = db.Wales

for item in WalesCollection.find():
   month = item['time']['month']
   if month == 'Apr':
       db.WalesApr.insert(item)
   elif month == 'May':
       db.WalesMay.insert(item)
   elif month == 'Jun':
       db.WalesJun.insert(item)
   elif month == 'Jul':
       db.WalesJul.insert(item)
   elif month == 'Aug':
       db.WalesAug.insert(item)
   elif month == 'Sep':
       db.WalesSep.insert(item)
   elif month == 'Oct':
       db.WalesOct.insert(item)
```

```
   else:
       db.WalesUnknown.insert(item)
```

Appendix 12: Applying the Combined Classifier to the 2014 Twitter Dataset Code

Appendix 12.1: East

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### EAST

### April

EastAprCollection = db.EastApr

EastApr_day_count = collections.defaultdict(int)
EastApr_pos_count = collections.defaultdict(int)

for item in EastAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastApr_pos_count[date] += 1

print('East April')
for key, value in sorted(EastApr_day_count.items()):
   print(key, value, EastApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastApr_pos_count[key]))
   f = open('East.csv', 'a')
   f.write(items)
   f.write('\n')

### May

EastMayCollection = db.EastMay
```

```
EastMay_day_count = collections.defaultdict(int)
EastMay_pos_count = collections.defaultdict(int)

for item in EastMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastMay_pos_count[date] += 1

print('East May')
for key, value in sorted(EastMay_day_count.items()):
   print(key, value, EastMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastMay_pos_count[key]))
   f = open('East.csv', 'a')
   f.write(items)
   f.write('\n')

### June

EastJunCollection = db.EastJun

EastJun_day_count = collections.defaultdict(int)
EastJun_pos_count = collections.defaultdict(int)

for item in EastJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastJun_pos_count[date] += 1

print('East June')
for key, value in sorted(EastJun_day_count.items()):
   print(key, value, EastJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastJun_pos_count[key]))
   f = open('East.csv', 'a')
   f.write(items)
   f.write('\n')

### July

EastJulCollection = db.EastJul

EastJul_day_count = collections.defaultdict(int)
EastJul_pos_count = collections.defaultdict(int)

for item in EastJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
```

```
        if EastJul_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                EastJul_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    EastJul_pos_count[date] += 1

print('East July')
for key, value in sorted(EastJul_day_count.items()):
   print(key, value, EastJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastJul_pos_count[key]))
   f = open('East.csv', 'a')
   f.write(items)
   f.write('\n')

### August

EastAugCollection = db.EastAug

EastAug_day_count = collections.defaultdict(int)
EastAug_pos_count = collections.defaultdict(int)

for item in EastAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
        if EastAug_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                EastAug_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    EastAug_pos_count[date] += 1

print('East August')
for key, value in sorted(EastAug_day_count.items()):
   print(key, value, EastAug_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastAug_pos_count[key]))
   f = open('East.csv', 'a')
   f.write(items)
   f.write('\n')

### September

EastSepCollection = db.EastSep

EastSep_day_count = collections.defaultdict(int)
EastSep_pos_count = collections.defaultdict(int)

for item in EastSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
        if EastSep_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                EastSep_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    EastSep_pos_count[date] += 1

print('East September')
```

```
for key, value in sorted(EastSep_day_count.items()):
   print(key, value, EastSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastSep_pos_count[key]))
   f = open('East.csv', 'a')
   f.write(items)
   f.write('\n')

### October

EastOctCollection = db.EastOct

EastOct_day_count = collections.defaultdict(int)
EastOct_pos_count = collections.defaultdict(int)

for item in EastOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastOct_pos_count[date] += 1

print('East October')
for key, value in sorted(EastOct_day_count.items()):
   print(key, value, EastOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastOct_pos_count[key]))
   f = open('East.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.2: East Midlands

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### EAST MIDLANDS

### April

EastMidlandsAprCollection = db.EastMidlandsApr

EastMidlandsApr_day_count = collections.defaultdict(int)
EastMidlandsApr_pos_count = collections.defaultdict(int)
```

```
for item in EastMidlandsAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMidlandsApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastMidlandsApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastMidlandsApr_pos_count[date] += 1

print('East Midlands April')
for key, value in sorted(EastMidlandsApr_day_count.items()):
   print(key, value, EastMidlandsApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastMidlandsApr_pos_count[key]))
   f = open('EastMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### May

EastMidlandsMayCollection = db.EastMidlandsMay

EastMidlandsMay_day_count = collections.defaultdict(int)
EastMidlandsMay_pos_count = collections.defaultdict(int)

for item in EastMidlandsMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMidlandsMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastMidlandsMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastMidlandsMay_pos_count[date] += 1

print('East Midlands May')
for key, value in sorted(EastMidlandsMay_day_count.items()):
   print(key, value, EastMidlandsMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastMidlandsMay_pos_count[key]))
   f = open('EastMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### June

EastMidlandsJunCollection = db.EastMidlandsJun

EastMidlandsJun_day_count = collections.defaultdict(int)
EastMidlandsJun_pos_count = collections.defaultdict(int)

for item in EastMidlandsJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMidlandsJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
```

```
                EastMidlandsJun_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    EastMidlandsJun_pos_count[date] += 1

print('East Midlands June')
for key, value in sorted(EastMidlandsJun_day_count.items()):
   print(key, value, EastMidlandsJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastMidlandsJun_pos_count[key]))
   f = open('EastMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### July

EastMidlandsJulCollection = db.EastMidlandsJul

EastMidlandsJul_day_count = collections.defaultdict(int)
EastMidlandsJul_pos_count = collections.defaultdict(int)

for item in EastMidlandsJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMidlandsJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastMidlandsJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastMidlandsJul_pos_count[date] += 1

print('East Midlands July')
for key, value in sorted(EastMidlandsJul_day_count.items()):
   print(key, value, EastMidlandsJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastMidlandsJul_pos_count[key]))
   f = open('EastMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### August

EastMidlandsAugCollection = db.EastMidlandsAug

EastMidlandsAug_day_count = collections.defaultdict(int)
EastMidlandsAug_pos_count = collections.defaultdict(int)

for item in EastMidlandsAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMidlandsAug_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastMidlandsAug_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastMidlandsAug_pos_count[date] += 1

print('East Midlands August')
for key, value in sorted(EastMidlandsAug_day_count.items()):
   print(key, value, EastMidlandsAug_pos_count[key])
```

```
   items = ("{0}, {1}, {2}".format(key, value,
EastMidlandsAug_pos_count[key]))
   f = open('EastMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### September

EastMidlandsSepCollection = db.EastMidlandsSep

EastMidlandsSep_day_count = collections.defaultdict(int)
EastMidlandsSep_pos_count = collections.defaultdict(int)

for item in EastMidlandsSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMidlandsSep_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastMidlandsSep_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastMidlandsSep_pos_count[date] += 1

print('East Midlands September')
for key, value in sorted(EastMidlandsSep_day_count.items()):
   print(key, value, EastMidlandsSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastMidlandsSep_pos_count[key]))
   f = open('EastMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### October

EastMidlandsOctCollection = db.EastMidlandsOct

EastMidlandsOct_day_count = collections.defaultdict(int)
EastMidlandsOct_pos_count = collections.defaultdict(int)

for item in EastMidlandsOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if EastMidlandsOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               EastMidlandsOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   EastMidlandsOct_pos_count[date] += 1

print('East Midlands October')
for key, value in sorted(EastMidlandsOct_day_count.items()):
   print(key, value, EastMidlandsOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
EastMidlandsOct_pos_count[key]))
   f = open('EastMidlands.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.3: London

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### LONDON

### April

LondonAprCollection = db.LondonApr

LondonApr_day_count = collections.defaultdict(int)
LondonApr_pos_count = collections.defaultdict(int)

for item in LondonAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if LondonApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               LondonApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   LondonApr_pos_count[date] += 1

print('London April')
for key, value in sorted(LondonApr_day_count.items()):
   print(key, value, LondonApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
LondonApr_pos_count[key]))
   f = open('London.csv', 'a')
   f.write(items)
   f.write('\n')

### May

LondonMayCollection = db.LondonMay

LondonMay_day_count = collections.defaultdict(int)
LondonMay_pos_count = collections.defaultdict(int)

for item in LondonMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if LondonMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               LondonMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
```

```
                    LondonMay_pos_count[date] += 1

print('London May')
for key, value in sorted(LondonMay_day_count.items()):
   print(key, value, LondonMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
LondonMay_pos_count[key]))
   f = open('London.csv', 'a')
   f.write(items)
   f.write('\n')

### June

LondonJunCollection = db.LondonJun

LondonJun_day_count = collections.defaultdict(int)
LondonJun_pos_count = collections.defaultdict(int)

for item in LondonJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if LondonJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               LondonJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   LondonJun_pos_count[date] += 1

print('London June')
for key, value in sorted(LondonJun_day_count.items()):
   print(key, value, LondonJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
LondonJun_pos_count[key]))
   f = open('London.csv', 'a')
   f.write(items)
   f.write('\n')

### July

LondonJulCollection = db.LondonJul

LondonJul_day_count = collections.defaultdict(int)
LondonJul_pos_count = collections.defaultdict(int)

for item in LondonJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if LondonJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               LondonJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   LondonJul_pos_count[date] += 1

print('London July')
for key, value in sorted(LondonJul_day_count.items()):
   print(key, value, LondonJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
LondonJul_pos_count[key]))
```

```
    f = open('London.csv', 'a')
    f.write(items)
    f.write('\n')

### August

LondonAugCollection = db.LondonAug

LondonAug_day_count = collections.defaultdict(int)
LondonAug_pos_count = collections.defaultdict(int)

for item in LondonAugCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if LondonAug_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                LondonAug_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    LondonAug_pos_count[date] += 1

print('London August')
for key, value in sorted(LondonAug_day_count.items()):
    print(key, value, LondonAug_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
LondonAug_pos_count[key]))
    f = open('London.csv', 'a')
    f.write(items)
    f.write('\n')

### September

LondonSepCollection = db.LondonSep

LondonSep_day_count = collections.defaultdict(int)
LondonSep_pos_count = collections.defaultdict(int)

for item in LondonSepCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if LondonSep_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                LondonSep_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    LondonSep_pos_count[date] += 1

print('London September')
for key, value in sorted(LondonSep_day_count.items()):
    print(key, value, LondonSep_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
LondonSep_pos_count[key]))
    f = open('London.csv', 'a')
    f.write(items)
    f.write('\n')

### October
```

```
LondonOctCollection = db.LondonOct

LondonOct_day_count = collections.defaultdict(int)
LondonOct_pos_count = collections.defaultdict(int)

for item in LondonOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if LondonOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               LondonOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   LondonOct_pos_count[date] += 1

print('London October')
for key, value in sorted(LondonOct_day_count.items()):
   print(key, value, LondonOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
LondonOct_pos_count[key]))
   f = open('London.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.4: North East

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### NORTH EAST

### April

NorthEastAprCollection = db.NorthEastApr

NorthEastApr_day_count = collections.defaultdict(int)
NorthEastApr_pos_count = collections.defaultdict(int)

for item in NorthEastAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthEastApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthEastApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthEastApr_pos_count[date] += 1
```

```
print('North East April')
for key, value in sorted(NorthEastApr_day_count.items()):
   print(key, value, NorthEastApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthEastApr_pos_count[key]))
   f = open('NorthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### May

NorthEastMayCollection = db.NorthEastMay

NorthEastMay_day_count = collections.defaultdict(int)
NorthEastMay_pos_count = collections.defaultdict(int)

for item in NorthEastMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthEastMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthEastMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthEastMay_pos_count[date] += 1

print('North East May')
for key, value in sorted(NorthEastMay_day_count.items()):
   print(key, value, NorthEastMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthEastMay_pos_count[key]))
   f = open('NorthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### June

NorthEastJunCollection = db.NorthEastJun

NorthEastJun_day_count = collections.defaultdict(int)
NorthEastJun_pos_count = collections.defaultdict(int)

for item in NorthEastJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthEastJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthEastJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthEastJun_pos_count[date] += 1

print('North East June')
for key, value in sorted(NorthEastJun_day_count.items()):
   print(key, value, NorthEastJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthEastJun_pos_count[key]))
   f = open('NorthEast.csv', 'a')
```

```
    f.write(items)
    f.write('\n')

### July

NorthEastJulCollection = db.NorthEastJul

NorthEastJul_day_count = collections.defaultdict(int)
NorthEastJul_pos_count = collections.defaultdict(int)

for item in NorthEastJulCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if NorthEastJul_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                NorthEastJul_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    NorthEastJul_pos_count[date] += 1

print('North East July')
for key, value in sorted(NorthEastJul_day_count.items()):
    print(key, value, NorthEastJul_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
NorthEastJul_pos_count[key]))
    f = open('NorthEast.csv', 'a')
    f.write(items)
    f.write('\n')

### August

NorthEastAugCollection = db.NorthEastAug

NorthEastAug_day_count = collections.defaultdict(int)
NorthEastAug_pos_count = collections.defaultdict(int)

for item in NorthEastAugCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if NorthEastAug_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                NorthEastAug_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    NorthEastAug_pos_count[date] += 1

print('North East August')
for key, value in sorted(NorthEastAug_day_count.items()):
    print(key, value, NorthEastAug_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
NorthEastAug_pos_count[key]))
    f = open('NorthEast.csv', 'a')
    f.write(items)
    f.write('\n')

### September

NorthEastSepCollection = db.NorthEastSep
```

```
NorthEastSep_day_count = collections.defaultdict(int)
NorthEastSep_pos_count = collections.defaultdict(int)

for item in NorthEastSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthEastSep_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthEastSep_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthEastSep_pos_count[date] += 1

print('North East September')
for key, value in sorted(NorthEastSep_day_count.items()):
   print(key, value, NorthEastSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthEastSep_pos_count[key]))
   f = open('NorthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### October

NorthEastOctCollection = db.NorthEastOct

NorthEastOct_day_count = collections.defaultdict(int)
NorthEastOct_pos_count = collections.defaultdict(int)

for item in NorthEastOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthEastOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthEastOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthEastOct_pos_count[date] += 1

print('North East October')
for key, value in sorted(NorthEastOct_day_count.items()):
   print(key, value, NorthEastOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthEastOct_pos_count[key]))
   f = open('NorthEast.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.5: North West

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle
```

```
import csv

import collections

db = client.twitter

##### NORTH WEST

### April

NorthWestAprCollection = db.NorthWestApr

NorthWestApr_day_count = collections.defaultdict(int)
NorthWestApr_pos_count = collections.defaultdict(int)

for item in NorthWestAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthWestApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthWestApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthWestApr_pos_count[date] += 1

print('North West April')
for key, value in sorted(NorthWestApr_day_count.items()):
   print(key, value, NorthWestApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthWestApr_pos_count[key]))
   f = open('NorthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### May

NorthWestMayCollection = db.NorthWestMay

NorthWestMay_day_count = collections.defaultdict(int)
NorthWestMay_pos_count = collections.defaultdict(int)

for item in NorthWestMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthWestMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthWestMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthWestMay_pos_count[date] += 1

print('North West May')
for key, value in sorted(NorthWestMay_day_count.items()):
   print(key, value, NorthWestMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthWestMay_pos_count[key]))
   f = open('NorthWest.csv', 'a')
   f.write(items)
```

```
   f.write('\n')

### June

NorthWestJunCollection = db.NorthWestJun

NorthWestJun_day_count = collections.defaultdict(int)
NorthWestJun_pos_count = collections.defaultdict(int)

for item in NorthWestJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthWestJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthWestJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthWestJun_pos_count[date] += 1

print('North West June')
for key, value in sorted(NorthWestJun_day_count.items()):
   print(key, value, NorthWestJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthWestJun_pos_count[key]))
   f = open('NorthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### July

NorthWestJulCollection = db.NorthWestJul

NorthWestJul_day_count = collections.defaultdict(int)
NorthWestJul_pos_count = collections.defaultdict(int)

for item in NorthWestJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthWestJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthWestJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthWestJul_pos_count[date] += 1

print('North West July')
for key, value in sorted(NorthWestJul_day_count.items()):
   print(key, value, NorthWestJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthWestJul_pos_count[key]))
   f = open('NorthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### August

NorthWestAugCollection = db.NorthWestAug
```

```
NorthWestAug_day_count = collections.defaultdict(int)
NorthWestAug_pos_count = collections.defaultdict(int)

for item in NorthWestAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthWestAug_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthWestAug_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthWestAug_pos_count[date] += 1

print('North West August')
for key, value in sorted(NorthWestAug_day_count.items()):
   print(key, value, NorthWestAug_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthWestAug_pos_count[key]))
   f = open('NorthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### September

NorthWestSepCollection = db.NorthWestSep

NorthWestSep_day_count = collections.defaultdict(int)
NorthWestSep_pos_count = collections.defaultdict(int)

for item in NorthWestSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if NorthWestSep_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               NorthWestSep_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   NorthWestSep_pos_count[date] += 1

print('North West September')
for key, value in sorted(NorthWestSep_day_count.items()):
   print(key, value, NorthWestSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthWestSep_pos_count[key]))
   f = open('NorthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### October

NorthWestOctCollection = db.NorthWestOct

NorthWestOct_day_count = collections.defaultdict(int)
NorthWestOct_pos_count = collections.defaultdict(int)

for item in NorthWestOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
```

```
        if NorthWestOct_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                NorthWestOct_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    NorthWestOct_pos_count[date] += 1

print('North West October')
for key, value in sorted(NorthWestOct_day_count.items()):
   print(key, value, NorthWestOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
NorthWestOct_pos_count[key]))
   f = open('NorthWest.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.6: Scotland

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### SCOTLAND

### April

ScotlandAprCollection = db.ScotlandApr

ScotlandApr_day_count = collections.defaultdict(int)
ScotlandApr_pos_count = collections.defaultdict(int)

for item in ScotlandAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if ScotlandApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               ScotlandApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   ScotlandApr_pos_count[date] += 1

print('Scotland April')
for key, value in sorted(ScotlandApr_day_count.items()):
   print(key, value, ScotlandApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
ScotlandApr_pos_count[key]))
   f = open('Scotland.csv', 'a')
   f.write(items)
   f.write('\n')
```

```
### May

ScotlandMayCollection = db.ScotlandMay

ScotlandMay_day_count = collections.defaultdict(int)
ScotlandMay_pos_count = collections.defaultdict(int)

for item in ScotlandMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if ScotlandMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               ScotlandMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   ScotlandMay_pos_count[date] += 1

print('Scotland May')
for key, value in sorted(ScotlandMay_day_count.items()):
   print(key, value, ScotlandMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
ScotlandMay_pos_count[key]))
   f = open('Scotland.csv', 'a')
   f.write(items)
   f.write('\n')

### June

ScotlandJunCollection = db.ScotlandJun

ScotlandJun_day_count = collections.defaultdict(int)
ScotlandJun_pos_count = collections.defaultdict(int)

for item in ScotlandJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if ScotlandJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               ScotlandJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   ScotlandJun_pos_count[date] += 1

print('Scotland June')
for key, value in sorted(ScotlandJun_day_count.items()):
   print(key, value, ScotlandJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
ScotlandJun_pos_count[key]))
   f = open('Scotland.csv', 'a')
   f.write(items)
   f.write('\n')

### July

ScotlandJulCollection = db.ScotlandJul

ScotlandJul_day_count = collections.defaultdict(int)
```

```
ScotlandJul_pos_count = collections.defaultdict(int)

for item in ScotlandJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if ScotlandJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               ScotlandJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   ScotlandJul_pos_count[date] += 1

print('Scotland July')
for key, value in sorted(ScotlandJul_day_count.items()):
   print(key, value, ScotlandJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
ScotlandJul_pos_count[key]))
   f = open('Scotland.csv', 'a')
   f.write(items)
   f.write('\n')

### August

ScotlandAugCollection = db.ScotlandAug

ScotlandAug_day_count = collections.defaultdict(int)
ScotlandAug_pos_count = collections.defaultdict(int)

for item in ScotlandAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if ScotlandAug_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               ScotlandAug_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   ScotlandAug_pos_count[date] += 1

print('Scotland August')
for key, value in sorted(ScotlandAug_day_count.items()):
   print(key, value, ScotlandAug_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
ScotlandAug_pos_count[key]))
   f = open('Scotland.csv', 'a')
   f.write(items)
   f.write('\n')

### September

ScotlandSepCollection = db.ScotlandSep

ScotlandSep_day_count = collections.defaultdict(int)
ScotlandSep_pos_count = collections.defaultdict(int)

for item in ScotlandSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if ScotlandSep_day_count[date] < 700:
```

```
            if TSM.sentiment(text)[1] >= 0.8:
                ScotlandSep_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    ScotlandSep_pos_count[date] += 1

print('Scotland September')
for key, value in sorted(ScotlandSep_day_count.items()):
   print(key, value, ScotlandSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
ScotlandSep_pos_count[key]))
   f = open('Scotland.csv', 'a')
   f.write(items)
   f.write('\n')

### October

ScotlandOctCollection = db.ScotlandOct

ScotlandOct_day_count = collections.defaultdict(int)
ScotlandOct_pos_count = collections.defaultdict(int)

for item in ScotlandOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if ScotlandOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               ScotlandOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   ScotlandOct_pos_count[date] += 1

print('Scotland October')
for key, value in sorted(ScotlandOct_day_count.items()):
   print(key, value, ScotlandOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
ScotlandOct_pos_count[key]))
   f = open('Scotland.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.7: South East

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### SOUTH EAST
```

```
### April

SouthEastAprCollection = db.SouthEastApr

SouthEastApr_day_count = collections.defaultdict(int)
SouthEastApr_pos_count = collections.defaultdict(int)

for item in SouthEastAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthEastApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthEastApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthEastApr_pos_count[date] += 1

print('South East April')
for key, value in sorted(SouthEastApr_day_count.items()):
   print(key, value, SouthEastApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthEastApr_pos_count[key]))
   f = open('SouthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### May

SouthEastMayCollection = db.SouthEastMay

SouthEastMay_day_count = collections.defaultdict(int)
SouthEastMay_pos_count = collections.defaultdict(int)

for item in SouthEastMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthEastMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthEastMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthEastMay_pos_count[date] += 1

print('South East May')
for key, value in sorted(SouthEastMay_day_count.items()):
   print(key, value, SouthEastMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthEastMay_pos_count[key]))
   f = open('SouthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### June

SouthEastJunCollection = db.SouthEastJun

SouthEastJun_day_count = collections.defaultdict(int)
SouthEastJun_pos_count = collections.defaultdict(int)
```

```
for item in SouthEastJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthEastJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthEastJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthEastJun_pos_count[date] += 1

print('South East June')
for key, value in sorted(SouthEastJun_day_count.items()):
   print(key, value, SouthEastJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthEastJun_pos_count[key]))
   f = open('SouthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### July

SouthEastJulCollection = db.SouthEastJul

SouthEastJul_day_count = collections.defaultdict(int)
SouthEastJul_pos_count = collections.defaultdict(int)

for item in SouthEastJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthEastJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthEastJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthEastJul_pos_count[date] += 1

print('South East July')
for key, value in sorted(SouthEastJul_day_count.items()):
   print(key, value, SouthEastJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthEastJul_pos_count[key]))
   f = open('SouthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### August

SouthEastAugCollection = db.SouthEastAug

SouthEastAug_day_count = collections.defaultdict(int)
SouthEastAug_pos_count = collections.defaultdict(int)

for item in SouthEastAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthEastAug_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
```

```
                SouthEastAug_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    SouthEastAug_pos_count[date] += 1

print('South East August')
for key, value in sorted(SouthEastAug_day_count.items()):
   print(key, value, SouthEastAug_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthEastAug_pos_count[key]))
   f = open('SouthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### September

SouthEastSepCollection = db.SouthEastSep

SouthEastSep_day_count = collections.defaultdict(int)
SouthEastSep_pos_count = collections.defaultdict(int)

for item in SouthEastSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthEastSep_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthEastSep_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthEastSep_pos_count[date] += 1

print('South East September')
for key, value in sorted(SouthEastSep_day_count.items()):
   print(key, value, SouthEastSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthEastSep_pos_count[key]))
   f = open('SouthEast.csv', 'a')
   f.write(items)
   f.write('\n')

### October

SouthEastOctCollection = db.SouthEastOct

SouthEastOct_day_count = collections.defaultdict(int)
SouthEastOct_pos_count = collections.defaultdict(int)

for item in SouthEastOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthEastOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthEastOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthEastOct_pos_count[date] += 1

print('South East October')
for key, value in sorted(SouthEastOct_day_count.items()):
   print(key, value, SouthEastOct_pos_count[key])
```

```
   items = ("{0}, {1}, {2}".format(key, value,
SouthEastOct_pos_count[key]))
   f = open('SouthEast.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.8: South West

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### SOUTH WEST

### April

SouthWestAprCollection = db.SouthWestApr

SouthWestApr_day_count = collections.defaultdict(int)
SouthWestApr_pos_count = collections.defaultdict(int)

for item in SouthWestAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthWestApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthWestApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthWestApr_pos_count[date] += 1

print('South West April')
for key, value in sorted(SouthWestApr_day_count.items()):
   print(key, value, SouthWestApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthWestApr_pos_count[key]))
   f = open('SouthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### May

SouthWestMayCollection = db.SouthWestMay

SouthWestMay_day_count = collections.defaultdict(int)
SouthWestMay_pos_count = collections.defaultdict(int)

for item in SouthWestMayCollection.find().batch_size(50):
```

```
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if SouthWestMay_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                SouthWestMay_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    SouthWestMay_pos_count[date] += 1

print('South West May')
for key, value in sorted(SouthWestMay_day_count.items()):
    print(key, value, SouthWestMay_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
SouthWestMay_pos_count[key]))
    f = open('SouthWest.csv', 'a')
    f.write(items)
    f.write('\n')

### June

SouthWestJunCollection = db.SouthWestJun

SouthWestJun_day_count = collections.defaultdict(int)
SouthWestJun_pos_count = collections.defaultdict(int)

for item in SouthWestJunCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if SouthWestJun_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                SouthWestJun_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    SouthWestJun_pos_count[date] += 1

print('South West June')
for key, value in sorted(SouthWestJun_day_count.items()):
    print(key, value, SouthWestJun_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
SouthWestJun_pos_count[key]))
    f = open('SouthWest.csv', 'a')
    f.write(items)
    f.write('\n')

### July

SouthWestJulCollection = db.SouthWestJul

SouthWestJul_day_count = collections.defaultdict(int)
SouthWestJul_pos_count = collections.defaultdict(int)

for item in SouthWestJulCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if SouthWestJul_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                SouthWestJul_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
```

```
                    SouthWestJul_pos_count[date] += 1

print('South West July')
for key, value in sorted(SouthWestJul_day_count.items()):
   print(key, value, SouthWestJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthWestJul_pos_count[key]))
   f = open('SouthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### August

SouthWestAugCollection = db.SouthWestAug

SouthWestAug_day_count = collections.defaultdict(int)
SouthWestAug_pos_count = collections.defaultdict(int)

for item in SouthWestAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthWestAug_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthWestAug_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthWestAug_pos_count[date] += 1

print('South West August')
for key, value in sorted(SouthWestAug_day_count.items()):
   print(key, value, SouthWestAug_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthWestAug_pos_count[key]))
   f = open('SouthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### September

SouthWestSepCollection = db.SouthWestSep

SouthWestSep_day_count = collections.defaultdict(int)
SouthWestSep_pos_count = collections.defaultdict(int)

for item in SouthWestSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthWestSep_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthWestSep_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthWestSep_pos_count[date] += 1

print('South West September')
for key, value in sorted(SouthWestSep_day_count.items()):
   print(key, value, SouthWestSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthWestSep_pos_count[key]))
```

```
   f = open('SouthWest.csv', 'a')
   f.write(items)
   f.write('\n')

### October

SouthWestOctCollection = db.SouthWestOct

SouthWestOct_day_count = collections.defaultdict(int)
SouthWestOct_pos_count = collections.defaultdict(int)

for item in SouthWestOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if SouthWestOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               SouthWestOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   SouthWestOct_pos_count[date] += 1

print('South West October')
for key, value in sorted(SouthWestOct_day_count.items()):
   print(key, value, SouthWestOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
SouthWestOct_pos_count[key]))
   f = open('SouthWest.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.9: Wales

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### WALES

### April

WalesAprCollection = db.WalesApr

WalesApr_day_count = collections.defaultdict(int)
WalesApr_pos_count = collections.defaultdict(int)

for item in WalesAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
```

```
    if len(text) >= 40:
        if WalesApr_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                WalesApr_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    WalesApr_pos_count[date] += 1

print('Wales April')
for key, value in sorted(WalesApr_day_count.items()):
    print(key, value, WalesApr_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
WalesApr_pos_count[key]))
    f = open('Wales.csv', 'a')
    f.write(items)
    f.write('\n')

### May

WalesMayCollection = db.WalesMay

WalesMay_day_count = collections.defaultdict(int)
WalesMay_pos_count = collections.defaultdict(int)

for item in WalesMayCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if WalesMay_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                WalesMay_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    WalesMay_pos_count[date] += 1

print('Wales May')
for key, value in sorted(WalesMay_day_count.items()):
    print(key, value, WalesMay_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
WalesMay_pos_count[key]))
    f = open('Wales.csv', 'a')
    f.write(items)
    f.write('\n')

### June

WalesJunCollection = db.WalesJun

WalesJun_day_count = collections.defaultdict(int)
WalesJun_pos_count = collections.defaultdict(int)

for item in WalesJunCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if WalesJun_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                WalesJun_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    WalesJun_pos_count[date] += 1
```

```
print('Wales June')
for key, value in sorted(WalesJun_day_count.items()):
   print(key, value, WalesJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WalesJun_pos_count[key]))
   f = open('Wales.csv', 'a')
   f.write(items)
   f.write('\n')

### July

WalesJulCollection = db.WalesJul

WalesJul_day_count = collections.defaultdict(int)
WalesJul_pos_count = collections.defaultdict(int)

for item in WalesJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WalesJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WalesJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WalesJul_pos_count[date] += 1

print('Wales July')
for key, value in sorted(WalesJul_day_count.items()):
   print(key, value, WalesJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WalesJul_pos_count[key]))
   f = open('Wales.csv', 'a')
   f.write(items)
   f.write('\n')

### August

WalesAugCollection = db.WalesAug

WalesAug_day_count = collections.defaultdict(int)
WalesAug_pos_count = collections.defaultdict(int)

for item in WalesAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WalesAug_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WalesAug_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WalesAug_pos_count[date] += 1

print('Wales August')
for key, value in sorted(WalesAug_day_count.items()):
   print(key, value, WalesAug_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WalesAug_pos_count[key]))
   f = open('Wales.csv', 'a')
   f.write(items)
```

```
   f.write('\n')

### September

WalesSepCollection = db.WalesSep

WalesSep_day_count = collections.defaultdict(int)
WalesSep_pos_count = collections.defaultdict(int)

for item in WalesSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WalesSep_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WalesSep_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WalesSep_pos_count[date] += 1

print('Wales September')
for key, value in sorted(WalesSep_day_count.items()):
   print(key, value, WalesSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WalesSep_pos_count[key]))
   f = open('Wales.csv', 'a')
   f.write(items)
   f.write('\n')

### October

WalesOctCollection = db.WalesOct

WalesOct_day_count = collections.defaultdict(int)
WalesOct_pos_count = collections.defaultdict(int)

for item in WalesOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WalesOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WalesOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WalesOct_pos_count[date] += 1

print('Wales October')
for key, value in sorted(WalesOct_day_count.items()):
   print(key, value, WalesOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WalesOct_pos_count[key]))
   f = open('Wales.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.10: West Midlands

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### WEST MIDLANDS

### April

WestMidlandsAprCollection = db.WestMidlandsApr

WestMidlandsApr_day_count = collections.defaultdict(int)
WestMidlandsApr_pos_count = collections.defaultdict(int)

for item in WestMidlandsAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WestMidlandsApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WestMidlandsApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WestMidlandsApr_pos_count[date] += 1

print('West Midlands April')
for key, value in sorted(WestMidlandsApr_day_count.items()):
   print(key, value, WestMidlandsApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WestMidlandsApr_pos_count[key]))
   f = open('WestMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### May

WestMidlandsMayCollection = db.WestMidlandsMay

WestMidlandsMay_day_count = collections.defaultdict(int)
WestMidlandsMay_pos_count = collections.defaultdict(int)

for item in WestMidlandsMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WestMidlandsMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WestMidlandsMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
```

```
                    WestMidlandsMay_pos_count[date] += 1

print('West Midlands May')
for key, value in sorted(WestMidlandsMay_day_count.items()):
   print(key, value, WestMidlandsMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WestMidlandsMay_pos_count[key]))
   f = open('WestMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### June

WestMidlandsJunCollection = db.WestMidlandsJun

WestMidlandsJun_day_count = collections.defaultdict(int)
WestMidlandsJun_pos_count = collections.defaultdict(int)

for item in WestMidlandsJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WestMidlandsJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WestMidlandsJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WestMidlandsJun_pos_count[date] += 1

print('West Midlands June')
for key, value in sorted(WestMidlandsJun_day_count.items()):
   print(key, value, WestMidlandsJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WestMidlandsJun_pos_count[key]))
   f = open('WestMidlands.csv', 'a')
   f.write(items)
   f.write('\n')

### July

WestMidlandsJulCollection = db.WestMidlandsJul

WestMidlandsJul_day_count = collections.defaultdict(int)
WestMidlandsJul_pos_count = collections.defaultdict(int)

for item in WestMidlandsJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WestMidlandsJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WestMidlandsJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WestMidlandsJul_pos_count[date] += 1

print('West Midlands July')
for key, value in sorted(WestMidlandsJul_day_count.items()):
   print(key, value, WestMidlandsJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WestMidlandsJul_pos_count[key]))
```

```
    f = open('WestMidlands.csv', 'a')
    f.write(items)
    f.write('\n')

### August

WestMidlandsAugCollection = db.WestMidlandsAug

WestMidlandsAug_day_count = collections.defaultdict(int)
WestMidlandsAug_pos_count = collections.defaultdict(int)

for item in WestMidlandsAugCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if WestMidlandsAug_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                WestMidlandsAug_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    WestMidlandsAug_pos_count[date] += 1

print('West Midlands August')
for key, value in sorted(WestMidlandsAug_day_count.items()):
    print(key, value, WestMidlandsAug_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
WestMidlandsAug_pos_count[key]))
    f = open('WestMidlands.csv', 'a')
    f.write(items)
    f.write('\n')

### September

WestMidlandsSepCollection = db.WestMidlandsSep

WestMidlandsSep_day_count = collections.defaultdict(int)
WestMidlandsSep_pos_count = collections.defaultdict(int)

for item in WestMidlandsSepCollection.find().batch_size(50):
    date = item['time']['date']
    text = item['tweet']['text']
    if len(text) >= 40:
        if WestMidlandsSep_day_count[date] < 700:
            if TSM.sentiment(text)[1] >= 0.8:
                WestMidlandsSep_day_count[date] += 1
                if TSM.sentiment(text)[0] == 'pos':
                    WestMidlandsSep_pos_count[date] += 1

print('West Midlands September')
for key, value in sorted(WestMidlandsSep_day_count.items()):
    print(key, value, WestMidlandsSep_pos_count[key])
    items = ("{0}, {1}, {2}".format(key, value,
WestMidlandsSep_pos_count[key]))
    f = open('WestMidlands.csv', 'a')
    f.write(items)
    f.write('\n')

### October
```

```
WestMidlandsOctCollection = db.WestMidlandsOct

WestMidlandsOct_day_count = collections.defaultdict(int)
WestMidlandsOct_pos_count = collections.defaultdict(int)

for item in WestMidlandsOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if WestMidlandsOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               WestMidlandsOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   WestMidlandsOct_pos_count[date] += 1

print('West Midlands October')
for key, value in sorted(WestMidlandsOct_day_count.items()):
   print(key, value, WestMidlandsOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
WestMidlandsOct_pos_count[key]))
   f = open('WestMidlands.csv', 'a')
   f.write(items)
   f.write('\n')
```

Appendix 12.11: Yorkshire and the Humber

```
import pymongo
from pymongo import MongoClient
client = MongoClient()

import Twitter_Sentiment_Module as TSM
import pickle

import csv

import collections

db = client.twitter

##### YORKSHIRE

### April

YorkshireHumberAprCollection = db.YorkshireHumberApr

YorkshireHumberApr_day_count = collections.defaultdict(int)
YorkshireHumberApr_pos_count = collections.defaultdict(int)

for item in YorkshireHumberAprCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if YorkshireHumberApr_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               YorkshireHumberApr_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   YorkshireHumberApr_pos_count[date] += 1
```

```
print('Yorkshire and the Humber April')
for key, value in sorted(YorkshireHumberApr_day_count.items()):
   print(key, value, YorkshireHumberApr_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
YorkshireHumberApr_pos_count[key]))
   f = open('YorkshireHumber.csv', 'a')
   f.write(items)
   f.write('\n')

### May

YorkshireHumberMayCollection = db.YorkshireHumberMay

YorkshireHumberMay_day_count = collections.defaultdict(int)
YorkshireHumberMay_pos_count = collections.defaultdict(int)

for item in YorkshireHumberMayCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if YorkshireHumberMay_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               YorkshireHumberMay_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   YorkshireHumberMay_pos_count[date] += 1

print('Yorkshire and the Humber May')
for key, value in sorted(YorkshireHumberMay_day_count.items()):
   print(key, value, YorkshireHumberMay_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
YorkshireHumberMay_pos_count[key]))
   f = open('YorkshireHumber.csv', 'a')
   f.write(items)
   f.write('\n')

### June

YorkshireHumberJunCollection = db.YorkshireHumberJun

YorkshireHumberJun_day_count = collections.defaultdict(int)
YorkshireHumberJun_pos_count = collections.defaultdict(int)

for item in YorkshireHumberJunCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if YorkshireHumberJun_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               YorkshireHumberJun_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   YorkshireHumberJun_pos_count[date] += 1

print('Yorkshire and the Humber June')
for key, value in sorted(YorkshireHumberJun_day_count.items()):
   print(key, value, YorkshireHumberJun_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
YorkshireHumberJun_pos_count[key]))
   f = open('YorkshireHumber.csv', 'a')
```

```
   f.write(items)
   f.write('\n')

### July

YorkshireHumberJulCollection = db.YorkshireHumberJul

YorkshireHumberJul_day_count = collections.defaultdict(int)
YorkshireHumberJul_pos_count = collections.defaultdict(int)

for item in YorkshireHumberJulCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if YorkshireHumberJul_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               YorkshireHumberJul_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   YorkshireHumberJul_pos_count[date] += 1

print('Yorkshire and the Humber July')
for key, value in sorted(YorkshireHumberJul_day_count.items()):
   print(key, value, YorkshireHumberJul_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
YorkshireHumberJul_pos_count[key]))
   f = open('YorkshireHumber.csv', 'a')
   f.write(items)
   f.write('\n')

### August

YorkshireHumberAugCollection = db.YorkshireHumberAug

YorkshireHumberAug_day_count = collections.defaultdict(int)
YorkshireHumberAug_pos_count = collections.defaultdict(int)

for item in YorkshireHumberAugCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if YorkshireHumberAug_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               YorkshireHumberAug_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   YorkshireHumberAug_pos_count[date] += 1

print('Yorkshire and the Humber August')
for key, value in sorted(YorkshireHumberAug_day_count.items()):
   print(key, value, YorkshireHumberAug_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
YorkshireHumberAug_pos_count[key]))
   f = open('YorkshireHumber.csv', 'a')
   f.write(items)
   f.write('\n')

### September

YorkshireHumberSepCollection = db.YorkshireHumberSep
```

```
YorkshireHumberSep_day_count = collections.defaultdict(int)
YorkshireHumberSep_pos_count = collections.defaultdict(int)

for item in YorkshireHumberSepCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if YorkshireHumberSep_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               YorkshireHumberSep_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   YorkshireHumberSep_pos_count[date] += 1

print('Yorkshire and the Humber September')
for key, value in sorted(YorkshireHumberSep_day_count.items()):
   print(key, value, YorkshireHumberSep_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
YorkshireHumberSep_pos_count[key]))
   f = open('YorkshireHumber.csv', 'a')
   f.write(items)
   f.write('\n')

### October

YorkshireHumberOctCollection = db.YorkshireHumberOct

YorkshireHumberOct_day_count = collections.defaultdict(int)
YorkshireHumberOct_pos_count = collections.defaultdict(int)

for item in YorkshireHumberOctCollection.find().batch_size(50):
   date = item['time']['date']
   text = item['tweet']['text']
   if len(text) >= 40:
       if YorkshireHumberOct_day_count[date] < 700:
           if TSM.sentiment(text)[1] >= 0.8:
               YorkshireHumberOct_day_count[date] += 1
               if TSM.sentiment(text)[0] == 'pos':
                   YorkshireHumberOct_pos_count[date] += 1

print('Yorkshire and the Humber October')
for key, value in sorted(YorkshireHumberOct_day_count.items()):
   print(key, value, YorkshireHumberOct_pos_count[key])
   items = ("{0}, {1}, {2}".format(key, value,
YorkshireHumberOct_pos_count[key]))
   f = open('YorkshireHumber.csv', 'a')
   f.write(items)
   f.write('\n')
```