COM814: Project 2015-16

Dissertation

School of Computing & Information Engineering

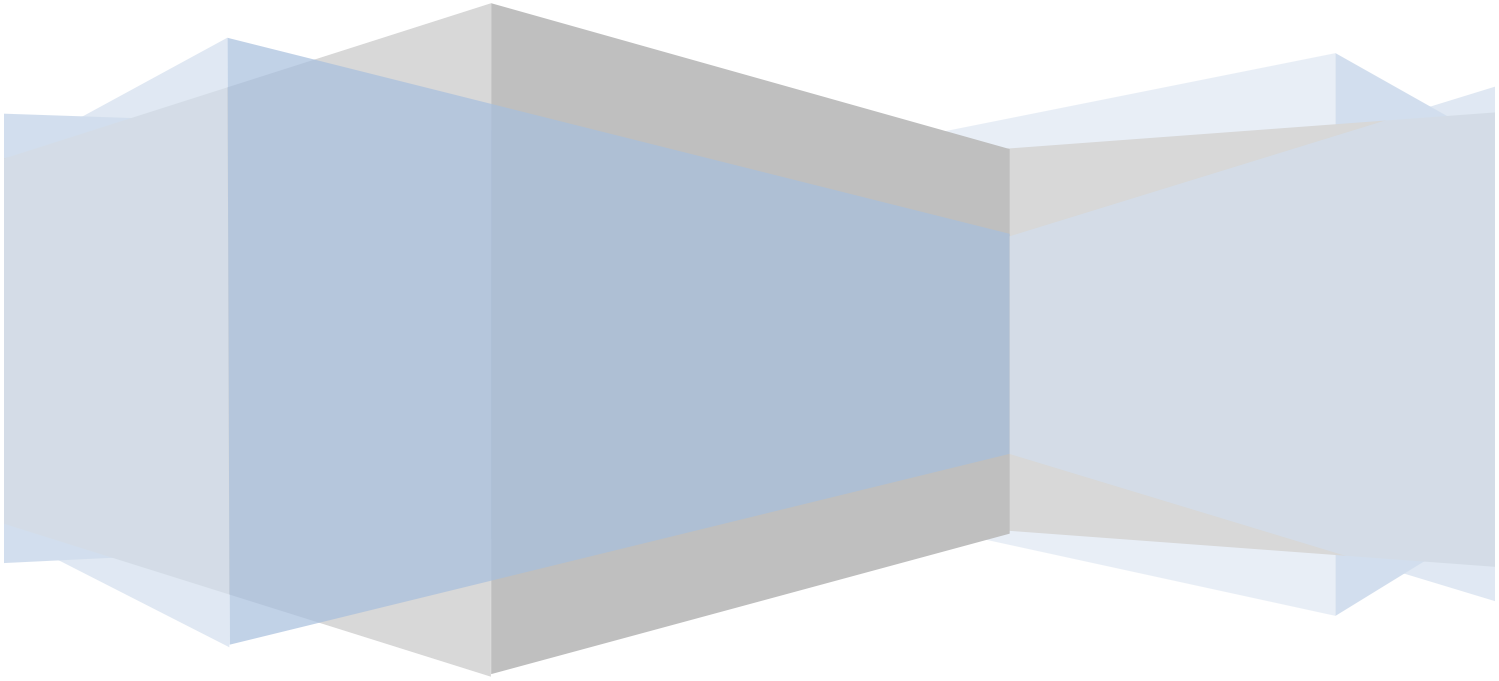
Michael Mc Macken B00702720

Sentiment Analysis and Classification in Python

Supervisor: Dr. Sandra Moffett

Second Marker: Dr. Ignacio Rano

01/09/16



Plagiarism Statement

I declare that this is all my own work and does not contain unreferenced material copied from any other source. I have read the University's policy on plagiarism and understand the definition of plagiarism. If it is shown that material has been plagiarised, or I have otherwise attempted to obtain an unfair advantage for myself or others, I understand that I may face sanctions in accordance with the policies and procedures of the University. A mark of zero may be awarded and the reason for that mark will be recorded on my file.

Signed: _____

Acknowledgements

Thanks to my family and friends for their continuous support and for motivating me to strive for better. Thank you to the staff at the University of Ulster, especially Martin and Janice for giving me a renewed love of programming. Thanks to my supervisor Dr. Sandra Moffett, as well as my second marker Dr. Ignacio Rano, for their constructive feedback on the ever-evolving subject of sentiment analysis.

Contents

Abstract

The ultimate aim of this project is to apply and improve upon already-existing methods of sentiment analysis to create an accurate classifier capable of expressing the polarity (positive vs. negative) of a given piece of text, be it a review or otherwise.

The solution proposed is a tool which accepts input from multiple sources including text entered by the user, documents selected for analysis, websites e.g. Amazon and outputs the overarching sentiment. Using Twitter's Streaming API, the designed tool would also have the capability of performing live sentiment analysis on tweets about a topic specified by the user.

To execute this analysis an underlying architecture reliant on a single classifier, which was to be trained using feature extraction and efficient pre-processing techniques to maximise accuracy, was initially proposed. However, during implementation and testing, it was found that by combining multiple trained classifiers into one that determined the sentiment using a voting system based on each classifier's decision, the overall accuracy as well as the reliability of the process was significantly improved.

The end product is a simple command-line based implementation with each planned feature fully functioning, from accepting different forms of input, to outputting the expressed polarity of the text alongside additional information, for example features identified by the classifier and how confident the classifier's decision was (as a percentage).

1. Introduction

Sentiment analysis refers to the process of extracting subjective information from a piece of text to determine the opinion expressed in relation to the subject of the text; hence it is also known as opinion mining, deriving the attitude of the author towards a particular topic or product. As with the nature of opinions, they are rarely straightforward, however for the most part they can be said to express an overall sentiment that is either positive or negative.

The aim of this project is to develop and implement a tool capable of extracting sentiment from product reviews, for example book or movie reviews. However, the tool should still be able to perform analysis on subjective text that does not fall under the umbrella of reviews (e.g. tweets) with relative accuracy and efficacy. The tool will accept input from a range of different sources, including text that has been typed in by the user, documents that have been selected by the user, URL's from specified websites, and tweets received from Twitter's Streaming API. Before the sentiment of the inputted text is deduced by the tool, the text will undergo a pipeline of pre-processing techniques aimed at stripping the text of its uninformative words and tokens; the most telling features will be extracted from this filtered text to better inform the decision of whether it is given the label of positive or negative. The overall performance of the tool and its underlying architecture will be evaluated against already-existing methods of sentiment analysis and the accuracy of the trained classifier will be assessed through extensive testing.

1.1 Sentiment Analysis Overview

Natural Language Processing (NLP), otherwise known as computational linguistics, is a vast subfield of computer science that has a number of practical purposes and one that is "increasingly being incorporated into consumer products" (Hirschberg & Manning, 2015). Machine Translation (MT) is constantly evolving to aid human-to-human communication; Spoken Dialogue Systems (SDS) are now used daily on mobile devices (Cortana, Siri, Google Now) and have paved the way for Socially Assistive Robots (Fasola & Matarić, 2012); Machine Reading (MR) allows vast amounts of data to be extracted and structured to aid scientific research (Peters et al. 2014).

Sentiment analysis is an area of NLP that has recently been the focus of a great deal of scientific and market research. Given the exponential growth of the internet, and the widespread use of mobile devices and computers with internet connectivity, more and more people are sharing their thoughts online. It is this seemingly endless amount of subjective data that has become important not only for "individual users looking for information to support their everyday purchasing decisions" (Perez-Rosa & Mihalcea, 2013), but for companies to understand more fully customer feedback on products and services. Sentiment analysis aims to take advantage of this vast amount of opinionated web-content by transforming it into useful information relating to whether the sentiment expressed is positive or negative.

Although there has been a relatively recent surge of interest and research activity into the field of sentiment analysis, Pang & Lee (2008) note that "there has been a steady undercurrent of interest for quite a while"; not surprising considering the enormous influence that online consumer voices now held on the opinions of other potential consumers thanks to the growth of the web. Since 2001, there have been "literally hundreds of papers published on the subject", and this was in part, thanks to the following:

- The rise of *Machine Learning* (ML) methods in NLP: Using both supervised and unsupervised methods of ML, information retrieval became more efficient, and unstructured data could be transformed into meaningful information.
- Availability of *datasets*: Thanks to the World Wide Web, and more specifically the introduction of review-aggregation websites, enormous datasets on which to train ML algorithms through supervised learning became more readily available.
- Realisation of potential: Not only realising the interesting and complex challenges of such an area of technology, the potential commercial benefits and intelligence applications were hard to ignore.

As Taboda (2012) points out, the "linguistic expression of emotions and opinions is one of the most fundamental human traits". The variations relating to linguistic expression of opinions such as syntax, vocabulary, slang etc. coupled with the unstructured and cumbersome way in which data is spread across the web ranging from social networks like Facebook to blogs and forums, means that sentiment analysis has had to grapple with a

number of problems, and as such, techniques for extracting and inferring sentiment vary widely. Sentiment analysis is an expansive area of research, and one that continues to show growing promise with deep models of sentiment analysis being introduced to better understand the context of longer sentences in determining sentiment (Socher et al. 2013).

1.2 The Problem

The automatic extraction and accurate classification of sentiment from text-based sources through a tool that accepts input from multiple sources.

The vast number of user-generated text online has "grown dramatically in recent years" (Xia et al. 2016), and as such there are a "wealth of user-generated reviews to inform purchase decisions" (Chua & Banerjee, 2016). However, due to the abundance of these reviews when looking up opinions of a product, a general search will confront the user with an overwhelming number of relevant web pages. A solution to this is to use machine learning algorithms to extract the text automatically.

Sifting through and analysing online reviews to get an overall sentiment is not only time-consuming, but can produce inaccurate or biased results due to human error. The consistency of human analysts would vary on a daily basis, therefore it is advantageous to automate monotonous tasks such as text classification. From a business standpoint, there are financial benefits of automating this process, as it would not only save time, but save money on costly and potentially inefficient methods.

Sentiment analysis can utilise a number of features to address the previously mentioned problems. By extracting features from a piece of text like word frequency and context, a document can be analysed for sentiment in a fragment of the time it would take manually. The process of determining the overall sentiment of a document and assigning labels based on polarity (positive/negative) can be accomplished by utilising a Classifier. Essentially, a classifier takes input data and maps it to a specific category based on its unique features.

It becomes a difficult task to detect sentiment based on relevant information; depending on which features are chosen to represent the text's overall opinion, the results may vary. Some of these features include (Liu, 2012; Rambocas & Gama, 2013):

- **Terms and their frequency:** These can be individual words (unigram) or n-grams with associated frequency counts. Most common features in traditional text classification.

- **Part of speech (POS):** Certain word groups, for example adjectives, have shown to be a good indication of opinion, and as such are treated as special features by several researchers.

- **Opinion words:** Certain words carry more weight with regards to opinion, for example *awesome* and *horrendous*; idioms such as “dead ringer” or “piece of cake” have also been found to play an important role in determining sentiment (Williams et al. 2015)

- **Sentiment shifters:** These kinds of expressions can change the orientation of the sentiment. The most common examples of these are negation words; the phrase “I didn’t like this book” would be classified as positive if negation were not taken into consideration.

The method of sentiment analysis which is most effective in terms of accuracy and consistency is determined by the techniques employed to train the classifier, the processing involved in the preparation phase, and the classifier chosen to be used.

1.3 Software Methodology

Because of the limited timescale of this project, a complimenting methodology will have to be chosen to better reflect the nature of the tool's development. Due to the complexities of NLP and sentiment analysis, the tool is likely to go through various forms of completion before the final product is ready. With this in mind, the tool will be built using the "Agile" methodology, with a focus on iterative development; the figure below is a simplified depiction of the "Agile" process.

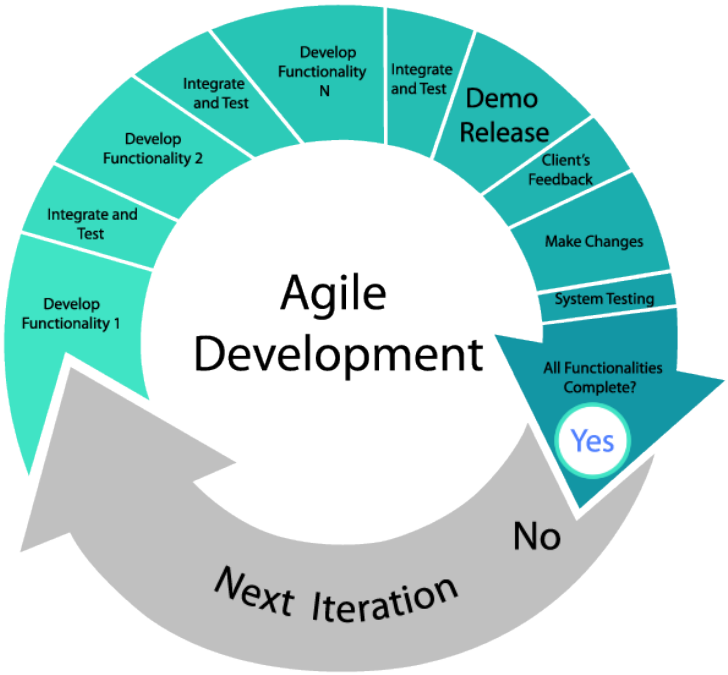


Figure 1.4: The "Agile" methodology

During the phases of development, the functional requirements of the tool may change or evolve, additional features may be added, or tests may reveal redundancies in the functionality. With an "Agile" approach, the tool will be developed in incremental, iterative steps, which "allows the [developer] to adapt quickly to changing requirements" (Cohen, Lindvall & Costa, 2003) and offers the manoeuvrability to make these changes without having to drastically alter the documentation.

1.5 Objectives

The overall objective is to develop a tool that can accurately detect the sentimental polarity of a given piece of text, sourced from various input methods and output that sentiment in a simple and clear manner. Specifically, the tool will provide the user with the ability to paste a URL from an online review, select a document from their device, or freely enter text for analysis, as well as the ability to view the analysis of live tweets based on a topic of their choosing.

1.5.1 Technical Objectives

- To develop and implement a tool capable of performing accurate and efficient sentiment analysis on opinionated text utilising the Natural Language Toolkit (NTK) and scikit-learn libraries, and written using Python, that follows the standard coding conventions of the language (Python Enhancement Proposals, 2016).

- To create a classifier that is able to perform, in terms of accuracy, optimally relative to other projects, surveys and studies in the field of sentiment analysis.

- To provide the user with a fully functioning, simple and easy-to-use command line menu, ensuring the tool's accessibility.

- To test and experiment with the various pre-processing techniques to ascertain their effect on the performance of the classifier, as well as compare the performance of different classifiers included with the scikit-learn library.

- To develop the tool so that it accepts and processes the various planned forms of input before classifying the contents into its respective polarity i.e. positive or negative.

1.5.2 Personal Objectives

- To become competent and comfortable with the Python programming language by using online tutorials and other available resources to develop simple applications.

- To build upon and apply previous experience gained from using the Java language to become more confident with the principles and techniques of programming e.g. Object Oriented Programming.

- To gain a deeper understanding of the field of Natural Language Processing, Machine Learning, and most importantly Sentiment Analysis; how it can be applied to enterprise and how it has advanced in recent years.

- To examine and become familiar with the NLTK and scikit-learn packages, their numerous libraries and how to make use of them within Python.

- Assimilate and ultimately contribute to the field of Sentiment Analysis.

2. Analysis

2.1 Existing Solutions

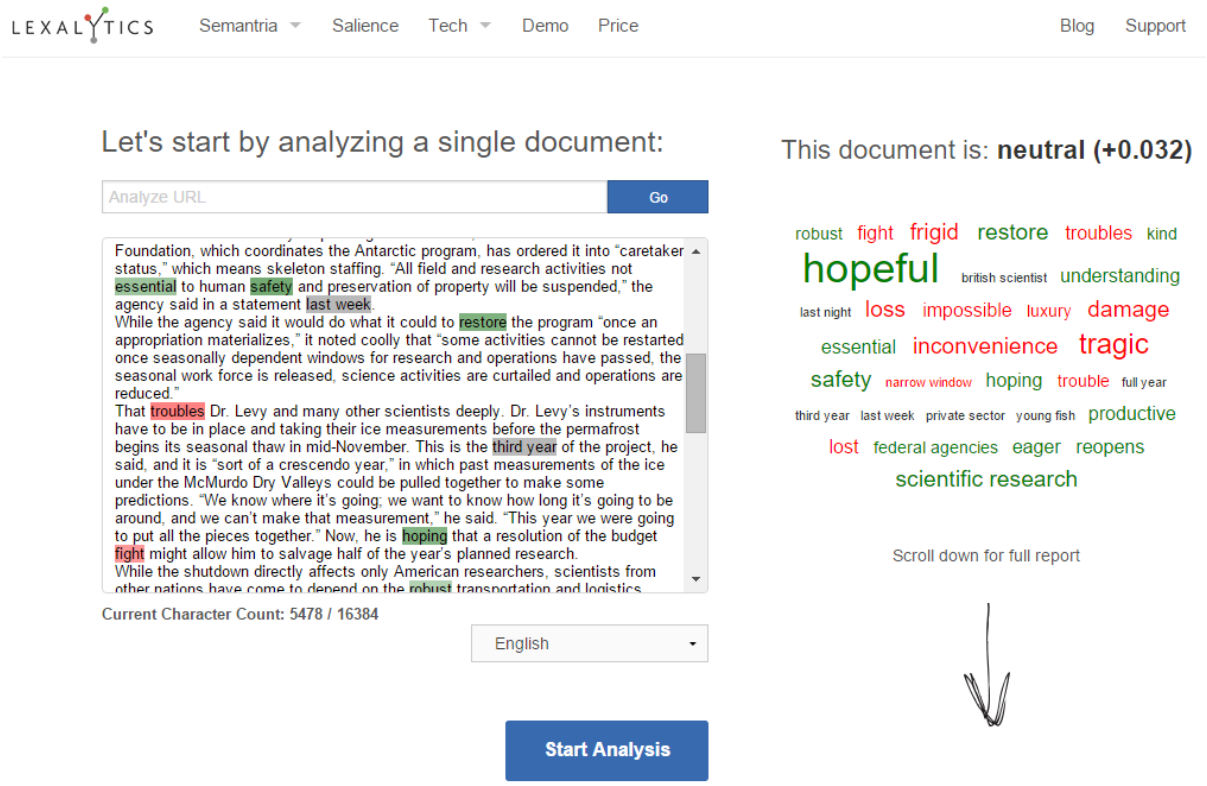


Figure 2.1a Sentiment Analysis Demo - https://www.lexalytics.com/demo

Above is an example of a fully realised sentiment analysis tool. As shown above, the text entered is given an overall polarity score between 0 and 1. As well as this, the program has highlighted positive, negative, and neutral words in the original text, as well as picked out words and represent their intensity by changing their font size. This example also contains a "full report", which includes additional features such as entity extraction (shown below).

Extracted entities	Evidence	Sentiment
Giants	7	+2.61
Coach Tom Coughlin	7	-0.33
Michael Cox	7	+2.95
David Wilson	7	-0.52
Brandon Jacobs	7	-0.54
NICKS	6	+1.60
Minnesota Vikings	5	+0.03
Nov. 10	5	-0.73
$10 million	5	+0.61
Boston	4	-0.26

Figure 2.1b: Full report of sentiment analysis tool

2.2 The Solution

The overall process of this project can be broken down into the following step-by-step process:

1. **Data Collection:** Collecting data in the form of user-generated product (specifically book) reviews. These are essentially raw text with no sentiment classification

2. **Text Preparation:** This step involves "cleaning the extracted data before the analysis is performed" (Rambocas & Gama, 2013), removing any non-textual content as well as irrelevant content. Details of this process will be discussed in the System Architecture section.

3. **Sentiment Detection:** This step utilises the trained classifier to detect the sentiment of the processed text (positive/negative)

4. **Sentiment Classification:** Labelling the overall sentiment of the submitted text.

5. **Visual representation:** The purpose of the analysis is to convert "fragmented text into meaningful information" (Rambocas & Gama, 2013). This might involve pie charts or other appropriate forms of data visualisation.

Figure 2.2a: Sentiment Analysis process simplified

Recent research on sentiment analysis has found that classifiers based on supervised machine learning have become quite adept. Not only this, but Narayanan et al (2013) found that a "highly accurate and fast sentiment classifier can be built using a simple Naïve Bayes model"; the Naïve Bayes model tends to work well with text classifications and takes less time to train that other models like support vector machines (SVM).

The approach of this project will be to utilise the Python programming language along with the NLTK software and utilise the sets of libraries included for processing such as tokenisation, stemming, tagging, parsing etc. The corpus of movie reviews packaged with NLTK will initially be looked into for testing and training the classifier to detect sentiment, and alongside the development of this will be the inclusion of a feature extractor so that only relevant information is being passed on to the classifier. This will be done with thought to sentiment shifters such as negation words as well as an amended list of stopwords to reduce the possibility of inaccuracy.

2.3 System Requirements

2.3.1 Functional Requirements

The functional requirements describe what the tool should do i.e. its behaviour with regards to the system's functionality. As described in the "Problem" section (Chapter 1.3) of this report in greater detail, the underlying architecture of the sentiment analysis tool must be able to perform the following:

- **Selecting** the raw text to be processed and classified, whether from the user directly or a different source.
- **Pre-processing** the text, in preparation for analysis, using techniques designed to break it into its component words (*tokenization*), strip it of its filler (*stopword filtration*), reduce words to their common base form where relevant (lemmatization), and take into account negation phrases (*negation handling*).
- **Extracting** and identifying informative features from the pre-processed text based on a dictionary of features using the most commonly occurring words in the training corpora.
- **Training** multiple classification algorithms included in the NLTK and scikit-learn packages, as well as **testing** the accuracy of each one.
- **Classifying and outputting** the overall sentiment of the input text based on the most accurate, trained classifier.

As this is the underlying architecture on which the core functionality is based, the user, for the most part, will not be aware of the above process as it is executing. They will be presented with a command-line menu detailing their options for selecting text to be analysed, including:

- **Custom text:** entered directly by the user through the console.
- **Document selection:** the tool will provide the user with the option of browsing the directories on their device through a simple GUI dialog box supported by Python's *Tkinter* package.

- **Text parsed from HTML documents/pages:** Using Python's *BeautifulSoup* library, review text can be pulled from the URL of accepted review sites and analysed by the classifier.

Giachanou & Crestani (2016) note that Twitter is "one of the most popular microblogs […] which has managed to attract a large number of users who share opinions, thoughts, and, in general, any kind of information about any topic of their interest". Therefore, in order to tap into this plethora of opinions, the Twitter Streaming API will be utilised:

- Users will have the ability to enter a topic of interest into the console; connecting to Twitter's Streaming API, the tool will then perform live sentiment analysis on tweets related to the chosen topic.

- Alongside this live analysis will be a live graph of the sentimental polarity of these tweets using Python's *matplotlib* package. Given that this is not part of the core functionality of the tool, it may be excluded from the final iteration.

2.3.2 Non-functional Requirements

Where functional requirements detail specific behaviours of a system, non-functional requirements specify criteria that judge the functionality of a system and place constraints on how the tool will perform its functions. The following requirements were incorporated in the design of the tool:

- **Scalability** - The tool should be developed mindful of the fact that additional features may be added further to its completion. This ranges from upgrading the classifier itself, to the addition of supplementary pre-processing techniques, as well as changes to the user's input methods.

- **Compatibility** - The tool will be compatible will the Windows 10 Operating System as well as other Operating Systems with Python 3.5 installed. Required modules or libraries not included with the standard Python distribution will be made clear to the user.

- **Accessibility** – The tool will be amicable towards users who might not be as tech-savvy as others, with straightforward menu options and instructions. Although the underlying architecture will be hidden from the user, a description of how the tool works will be available.

- **Performance** – The sentiment analysis being performed by the tool will be consistently accurate in its execution, the success of which will be assessed in the Testing section of this report. The tool's menu system will be robust in that it will output relevant error messages where appropriate, being free of bugs and major crashes; this robustness will extend to the system's architecture: the code for the classifier should account for unexpected or unacceptable forms of input.

- **Data Integrity** – The tool will not ask for, nor collect any user's sensitive data, and therefore security of such data will not be an issue; the reviews being used to train and test the classifier are non-profitable and available to the public.

2.4 Professional Issues

According to Liu (2012), "our beliefs and perceptions of reality, and the choices we make, are […] conditioned upon how others see and evaluate the world". This applies to online reviews/opinions of different products; thanks to the growth of the Internet, information is no longer controlled by the media or big business. Millions of people all over the world are sharing their opinions and influencing others' decisions through "online word-of-mouth" (Duan et al. 2008).

It is now commonplace for consumers to assess online reviews of a product they are interested in, using sites like Amazon or eBay and indeed research suggests that this has a significant impact on purchase behaviour (comScore 2007; Horrigan 2008; Chen et al. 2003). If the majority of reviews for that product are positive, then a consumer is more likely to buy it; if the reviews are mostly negative, it is unlikely the consumer will spend money on the item. Thus, an organization, business or individual can profit from positive opinions online.

The Natural Language Processing market as a whole is predicted to be worth $13.4 billion by the year 2020 (MarketsandMarkets, 2015), and textual analytics including sentiment analysis will no doubt continue to prove useful for business operations; below highlights the uses of sentiment analysis by exploring its applications.

Applications of Sentiment Analysis

Businesses and organizations

1. Benchmarking products; market research. Companies can "tune in, at scale, to the voice of the customer, patient, public, and market" (Grimes, 2016).
2. Emotion Analytics – a subcategory of sentiment analysis that extracts affective states from images and video via expression analysis.
3. Healthcare – for example, using social media to analyse the moods of patients with cancer (Rodrigues et al, 2015).
4. Businesses pay vast sums of money to conduct and analyse customer satisfaction surveys, despite the ineffectiveness and limitations of this method (Nasukawa & Yi, 2003).

Individuals

1. Aid in the decision making process of buying an item or service.
2. Find opinions on products as well as perform comparisons between similar products and their specific features.

Advertisement

1. Placing ads on blogs based on positive or neutral sentiment (Fan & Chang, 2010).
2. Dissatisfaction-oriented advertising – ads promoting the competing products of those with which the consumer is not satisfied. (Qiu et al. 2010).

Opinion Retrieval

1. For example, attitudes towards political parties, government agencies or specific policies that could inform discussions.
2. Analysing political trends and predicting election results.
3. Identifying bias in the news for example.

It is clear that sentiment analysis has a place from a business standpoint; the ability to "tap the unprompted voice of the customer, via social listening" (Grimes, 2015) is one that has endless practicality.

2.5 Project Risks

It is important to give forethought to potential risks that might arise during the course of the project so that sufficient measures can be taken to pre-empt any such risks from happening.

Physical Risks

Due to the necessity of having to spend prolonged periods of time at a computer in order to develop the tool and consolidate the documentation, physical risks include strained eyes, back pain and repetitive strain injury.

These risks can be avoided by utilising a working environment that is comfortable, which includes a chair with robust lumbar support and wrist rests for mouse and keyboard; frequent breaks and routine physical exercise will also lower the risk of being affected by

the above. In order to avoid strained eyes, a software program ***f.lux***, which "makes the color of your computer's display adapt to the time of day" (f.lux, 2016), will be used.

Technical Risks

The technical risks are low; the software required for sentiment analysis is available to download online for free and tutorials for learning and utilising the Python language, as well as NTLK, are abundant. Not only this, but the choice to use Python as the principal programming language lowers the risk of issues as it has a "very shallow learning curve" and "allows a programmer to rapidly prototype a project […] fit for largescale production" (Madnani, 2013).

The laptop on which the bulk of the work is being done runs a 64-bit version of Windows 10, however it will utilise the 32-bit version of Python due to present problems with the compatibility of Python's 64-bit version and NLTK (Krispil, 2013), which is necessary for the project.

Unique to this project is the risk of erroneous sentiment analysis; while it will never be 100% accurate, steps will be taken during implementation and testing phase to ensure that the classifier performs as intended.

Hardware Risks

All files relating to the project are backed up on an external hard drive, as well as online through Dropbox, where in the case of a "meltdown, your stuff is safe" and "can be restored in a snap" (Dropbox, 2016). This not only reduces the likelihood of data being lost, but also acts as a failsafe to any files accidentally being deleted by tracking changes and storing earlier versions.

Risk Management

The following table identifies the risks associated with the project and gives each a numerical value representing the severity (Points), probability of occurring (Level) and finally that risk's calculated Threat (severity*probability).

2.5.1 Risk Management Table

Risk	Points	Level	Threat	Preventative Actions
Inadequate understanding of project to see its completion	3	2	6	Become familiar with Python and NTLK software using online tutorials and help from staff where necessary. Perform a thorough investigation into the field of sentiment analysis and read up on any recent advancements.
Quality of report writing lacking	4	4	16	Plan to have most if not all work finished before due date with enough time to have it proofread by a third party as well as checked by advisor.
Incomplete Software	4	3	12	Aim to have a functioning prototype of algorithm/system at an early stage.
Data loss/corruption	1	1	1	Store files related to project on a Dropbox folder that is synched across multiple devices and refreshes as long as the device is connected to the internet. Back up files on external hard drive.
Not meeting set deadline	4	2	8	Manage time proficiently by setting realistic goals and utilising project management applications e.g. Trello. Frequently meet with advisor and use project log to stay on top of things.

2.6 Ethical Approval

The ethical considerations of this project are considerably straightforward thanks to the open-sourced nature of the programming language and libraries, as well as the review corpus that is openly available online.

The ethics of this project was approved and deemed to be under Category Z –

This project does not involve invasive interaction with people and does not require a full Ethical review. This includes the use of anonymous questionnaires for testing software etc. I confirm that this project meets the definition for research in the category above.

- All risks and ethical procedural implications have been considered.
- The project will be conducted at all times in compliance with the research description/protocol and in accordance with the University's requirements on recording and reporting.
- This application has not been submitted to and rejected by another committee

3. Design

Given the complexities of Sentiment Analysis and Natural Language Processing, the main focus of this project will be on the core functionality of the tool itself and, as such, the interface between the tool and the user will be a simple command-line menu. Before any design considerations are made in regards to the menu system, however, the underlying architecture of the system must be comprehensively explained.

3.1 Architectural Design

The following diagram illustrates the Sentiment Analysis process on which the tool's functionality will rely:

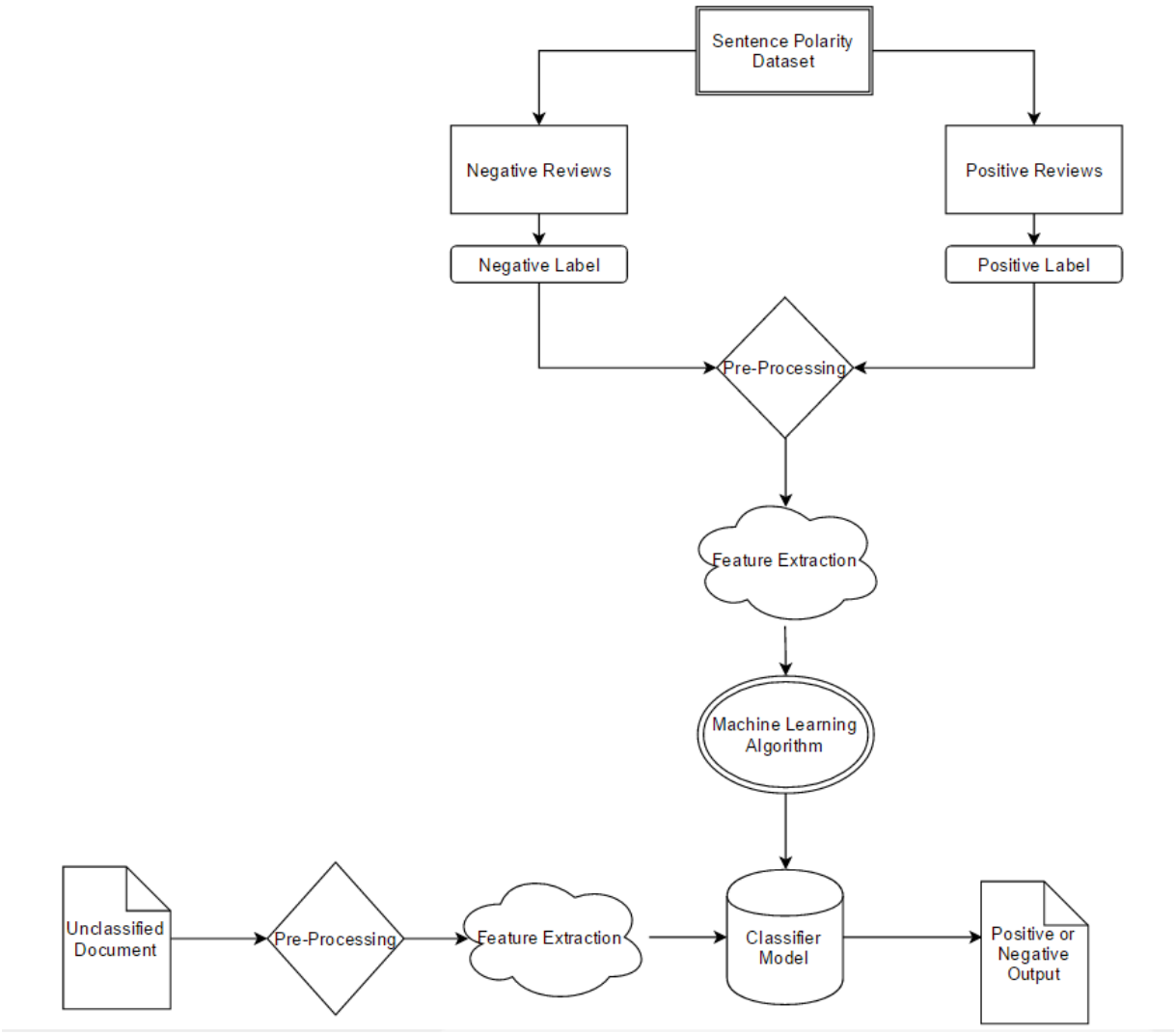


Figure 3.1: Process Diagram of System Architecture for SA tool

3.1.1 Classifier Design

The Natural Language Toolkit comes built in with its own small collection of more widely used classifiers, of which the **Naïve Bayes** classifier model is included; this classifier will be the primary focus for the Sentiment Analysis tool's functionality. That being said, each classifier included with NLTK and the scikit-learn library has its own strengths and weaknesses, as well as unique requirements, and these will be taken into consideration during implementation and testing.

A Naïve Bayes classifier applies Bayes' Probabilistic Theorem to attempt to predict the possible classification for any given text; to do this, it needs a number of previously classified documents of the same or similar type. The full theorem is written as follows:

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

In the context of this project, the goal is to find the probability that a document can be classified into either positive, or negative. One might look specifically at classifying an online review as positive; using Bayes theorem, the probability of the review being positive can be calculated and then compared to the probability of the review being negative. In this context, the theorem can be broken down into its various component parts for the sake of transparency:

$P(A|B)$ ——— This can be read as the probability of A – the classification of the review being positive – given the review (B). This is what is being worked out using the rest of the theorem.

$P(B|A)$ ——— The probability of the review (B) occurring in the given classification of positive (A). Calculated using training dataset.

$P(A)$ ——— The probability of A – the class (positive). This is independent of all other probabilities.

$P(B)$ ——— The probability of B – the review. This is also independent of all other probabilities.

From the above, the theorem can be filled in to reflect the specific application for this example:

$$P(\text{positive} \mid \text{review}) = \frac{P(\text{review} \mid \text{positive}) * P(\text{positive})}{P(\text{review})}$$

Since the probability of the review $P(B)$ is a "normalising constant and helps scale our equation" (Azad, 2016) so that it falls between 0 and 1, it can be disregarded in the calculations, leaving the important calculation between the probability of the review given the class $P(B|A)$ and the probability of $P(A)$ the class:

$$P(\text{positive} \mid \text{review}) = P(\text{review} \mid \text{positive}) * P(\text{positive})$$

The probability of the positive class $P(A)$ is dependent on the number of classes; in the context of this project it will be a binomial classification between positive and negative, meaning the probability $P(A)$ = 0.5.

To calculate the probability of the review occurring given the positive class $P(B|A)$ is where the training dataset comes in. Using a pre-labelled Sentence Polarity dataset included with NLTK's corpus package, the machine learning algorithm forms a basis on which to compute the probability that the review will fall into a specific class. However, given that the chances are extremely low that the entirety of a review text would be found in the training set, it is necessary to split the review up into its individual words, which are then extracted as features. The probability for each feature in the training set is then calculated:

$$P(\text{positive} \mid \text{review}) = P(T1 \mid \text{positive}) * P(T2 \mid \text{positive}) * P(Tn \mid \text{positive})$$

Here, T1 to Tn is all the words in the review. To determine the probability of a specific word falling into the positive class e.g. *P(T1|positive)* the following is needed from the training dataset:

- The number of times T1 occurs in reviews that were marked as positive in the training dataset.
- The total number of words in the positively labelled feature set

For example, if the word **food** occurs in a set of positive tweets 455 times and there are 1211 total words in the positive tweets, the relative probability of **food** occurring in positive category is worked out by dividing the number of occurrences (455) by the total number of words (1211). Therefore the relative probability of **food** occurring in a positive context is 0.376; not surprising considering that food is just as likely to have positive, negative or neutral connotations (du Toit, 2015). Once the probability of each selected feature is calculated, the overall probability of the class given the review can be calculated; the probability of "**I love good food**" being positive can be broken down as follows, assuming that each word's probability has been calculated previously:

*P (positive | review) = P (T1|positive) * P(T2|positive) * P(Tn | positive) * P(positive)*

*= P(**I**|positive) * P(**love**|positive) * P(**good**|positive) * P(**food**|positive) * P(positive)*

*= 0.25 * 0.625 * 0.74 * 0376 * 0.5*

*= **0.0245703125***

Once this has been done, the probability of the same text being negative is calculated in the same way and the two calculations are compared against each other; the text is then categorised into the class with the higher probability.

It is worth noting that the Naïve Bayes classifier is called naïve for a reason: it assumes that the presence of one feature in a class does not depend on the presence or absence of another. The bag of words model goes hand-in-hand with the Naïve Bayes classifier for this reason, as it looks at words from a document in isolation with no regard to context or sentence structure.

Whilst independence is generally a poor assumption to make, the Naïve Bayes classifier is often "remarkably successful in practice, often competing with much more sophisticated techniques" (Rish, 2001), and can still be optimal "even when the assumption is violated by a wide margin" (Domingos & Pazzani, 1997). Even so, it will be important to appraise other classification algorithms, as the Naïve Bayes model is known to have a poor probability estimation in spite of its efficacy (Zhang, 2004).

Included in the **scikit-learn** library is a number of Machine Learning classifiers that are not packaged with NLTK; the Naïve Bayes algorithm alone has three variants, including the Multinomial Naïve Bayes model – "one of two classic Naïve Bayes variants used in text classification" - as well as the Bernoulli and Gaussian Naïve Bayes models. Also included are variants of the Generalized Linear Models, such as the Logistic Regression model – which, despite its name "is a linear model for classification rather than regression" (Pedregosa et al. 2011) - or the Stochastic Gradient Descent model, as well as numerous Support Vector Machine (SVM) models. In its "bleeding-edge version", NLTK comes with a package that implements a wrapper around scikit-learn classifiers, thus making it much "easier to use for NLP people" (Buitinck, 2012).

In his book, *Python 3 Text Processing with NLTK 3 Cookbook*, Jacob Perkins proposes that "one way to improve classification performance is to combine classifiers". The idea is to combine multiple individual classifiers that use different algorithms and through a voting system deduce the overall polarity by choosing the label with the majority vote; ideally the "combination of many [would] compensate for individual bias" because "multiple algorithms are better than one". This technique will be considered, as a voting system may improve the overall reliability of the classification process. However, training multiple classifiers creates room for issues to arise in terms of accuracy; therefore diligence is required, as "combining a poorly performing classifier with better performing classifiers is generally not a good idea" (Perkins, 2014).

3.1.2 Pre-processing

In an attempt to increase the overall accuracy of the classifier, pre-processing will be performed on both the training set and any user input. This process involves removing

unnecessary information irrelevant to the classification of the text as well as breaking the text up by their words or tokens. The following is a dissection of the pre-processing phase:

Normalisation: White space will be cleared from the text and the text will be converted to lower case format.

Tokenisation: This involves splitting the text up into units known as tokens. A token is "an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing" (Manning et al. 2009). Below is an example of the process of tokenization:

Input: "The man in black fled across the desert, and the gunslinger followed."

Output: ["The", "man", "in", "black", "fled", "across", "the", "desert", ",", "and", "the", "gunslinger", "followed", "."]

The text can be tokenised by sentence as well as by word and for the purposes of this tool it will be tokenised by word using a tokenizer included with the NLTK package known as the *WordPunctTokenizer.* This stage makes it easier to extract individual features when it comes to extracting features.

Stemming / Lemmatizing: Stemming reduces a word down to its root so that different forms and variations of a word will be normalised e.g. "ran", "run", "running" will all stem to "run". Stemming is however seen as a "crude heuristic process that chops off the ends of words in the hope of achieving [its] goal" compared to the process of Lemmatizing. Lemmatizing is seen as doing things the proper way, despite not being as fast as stemmers and taking up more resources; the goal is to "remove inflectional endings only and to return the base or dictionary form a word" (Manning et al. 2008), known as a lemma, utilising NLTK's built in WordNet.

Removing Stopwords: Stopwords are extremely common words that offer little to no value in determining the label of a document or piece of text; in this case, they are words deemed as uninformative with regards to the overall sentiment of the text. As a general strategy, stopword lists are created by sorting the words or terms in a document by *collection*

frequency – counting the total number of times a word appears – and then manually filtering the most frequent terms judged to be semantically lacking with regards to content. Although NLTK comes built in with its own list of English stopwords that can be added to, it may be more suitable to create a custom list of stopwords stored in a text file that is then read in by the code. This way, words specific to the territory of movie reviews, book reviews or tweets can be added without having to amend the code; the tool's scalability is also bolstered using this method, as custom stop lists can be added to reflect the domain of the product(s) being processed. Pseudo-code for stopword removal is shown below.

Negation: As stated previously, a bag of words model looks at words/features in isolation and therefore cannot tell the difference between positive words and negated phrases such as "fun" and "not fun"; Perkins proposes a method to handle this known as "antonym replacement" (Perkins, 2014):

```
>> sent = ["let's", "not", "uglify", "our", "code"]
>> replace_negations (sent)
>> ["let's", "beautify", "our", "code"]
```

This involves identifying a word that is preceded by a sentiment shifter like "not", "didn't", or other such words ending in "n't", and replacing it with an unambiguous antonym; it is so-called unambiguous if only one antonym is found when looking up the Synsets of the word using WordNet. If more than one antonym is returned, which can often be the case, then the replacement is ambiguous and does not take place, as the correct antonym cannot be known for sure. Pang et al. (2002) proposed adding the tag "NOT_" to every word between a negation word and the first punctuation mark found after it, so that a sentence like "I do not recommend this movie" becomes:

```
>> ["I", "do", "not", "NOT_recommend", "NOT_this", "NOT_movie"]
```

This technique will be considered, however a combination and slight alteration of both of the above methods may prove to be the most useful, the pseudo-code for which is shown below.

```
FOR individual_word in document_text:
      IF individual_word PRECEDED by negation_words:
            IF unambiguous_antonym:
                  RETURN antonym_word
            ELSE return "NOT_" + individual_word
```

Word length filter: This is done for the same reason that stopwords are removed; words only 2 characters in length or less will not inform the sentiment one way or the other, and therefore these words like "to", "do", "an", "I", etc. will be removed.

Spelling replacement: In everyday language, people are lackadaisical towards the use of grammar and this is especially true for online blogs, reviews and tweets. Often, reviews will contain repeating characters e.g. "I looooooooved this movie" to emphasise their feeling towards something. In other cases – more so in tweets because of the restrictions on length and its informal nature (Giachanou & Crestani, 2016) – words will be spelled incorrectly, or abbreviated, or be some form of slang. Perkins provides methods to handle these kinds of errors through regular expressions and WordNet, whether or not they will be adapted for this project will be decided during implementation.

Part-of-Speech tagging: This is the process of classifying words into their lexical categories (adjectives, nouns, verbs etc.). NLTK comes built with its own Part of Speech tagging function; below is a simplified Part-of-Speech tag set.

Tag	Meaning	English Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, univeristy*

Figure 3.1.3a: Universal POS tag-set (Bird, 2009)

Reading a document: With the option of choosing a document for classification, the tool will need to be able to extract the text from .docx files – the Python module **docx2txt** provides such a utility for doing so with relative ease - or decide to simply read in for example .txt files.

Extracting text from URL: The tool will give the user the option to enter the URL of a Goodreads book review page, or an Amazon product review page as a source of input for sentiment analysis. The method for extracting the targeted text from the specific website's URL will have to be tweaked slightly depending on the website, but it will more or less use the **BeautifulSoup** module to identify the required text found between specific tags in the HTML document. Once the text has been found in the HTML code, the rest of the document can be disregarded, and it is only a matter of saving the required text for pre-processing and analysis. Below shows an example of how the review text of a Goodreads page can be located using HTML tags.

Figure 3.1.3b: Goodreads page for "The Way of Kings" book review and source code (Kogiopsis, 2012)

3.1.3 Feature Extraction

Feature Extraction is "perhaps the most difficult task in SA" because the majority of the works on sentiment analysis do not follow any "consensual" approach, but instead "frequently offer ad-hoc solutions" (Siqueira & Barros, 2010). One frequently used approach that can be "extremely effective in practice", and will be adapted for this project, is to compile all of the words from the corpora into one list – including repeat words – that is then ordered in descending frequency i.e. the most commonly occurring words are at the top. From this list, the top N% are chosen to be features representative of the data set and are used for training; the minor difference for this project being that instead of a percentage, an integer will be specified. Example pseudo-code is shown below.

```
SORT all_words by DESCENDING ORDER
word_features = []
FOR each_word in top(N) FROM all_words:
      ADD each_word to word_features
```

Although there is a risk of losing out on "infrequent but valuable words" that can inform the sentiment of a text, the top N words picked as features are likely to "carry subtle but extremely reliable sentiment information" (Potts, 2011). Using these word features, a method can be written to find features within a given piece of text, the pseudo-code for which elucidates how it might be done:

```
FOR feature_word IN word_features:
      IF feature_word IN document_text:
            features_in_doc CONTAINS feature_word = TRUE
      ELSE features_in_doc CONTAINS feature_word = TRUE
```

What this will do is it will transform the document into a dictionary of the chosen word features, each with a **Boolean** indicating its presence (or lack thereof) in the document. The text "This movie was rubbish, I hated it" might become (assume pre-processing has been performed):

```
[{"love": false},
 {"hate": true},
 {"rubbish": true},
 {"transcendent": false},
                      …}]
```

From there, it is only a matter of finding out which features more commonly occur in positive documents and which appear more frequently in negative documents to create a predictive model on which to perform the classification.

3.1.4 Training the Classifier

The Naïve Bayes classifier, and those previously discussed in this section, fall under the umbrella of **supervised** machine learning algorithms; a classifier is said to be supervised "if it is built based on training corpora containing the correct label for each input" (Bird, 2009). The framework of this method is illustrated in the figure below.

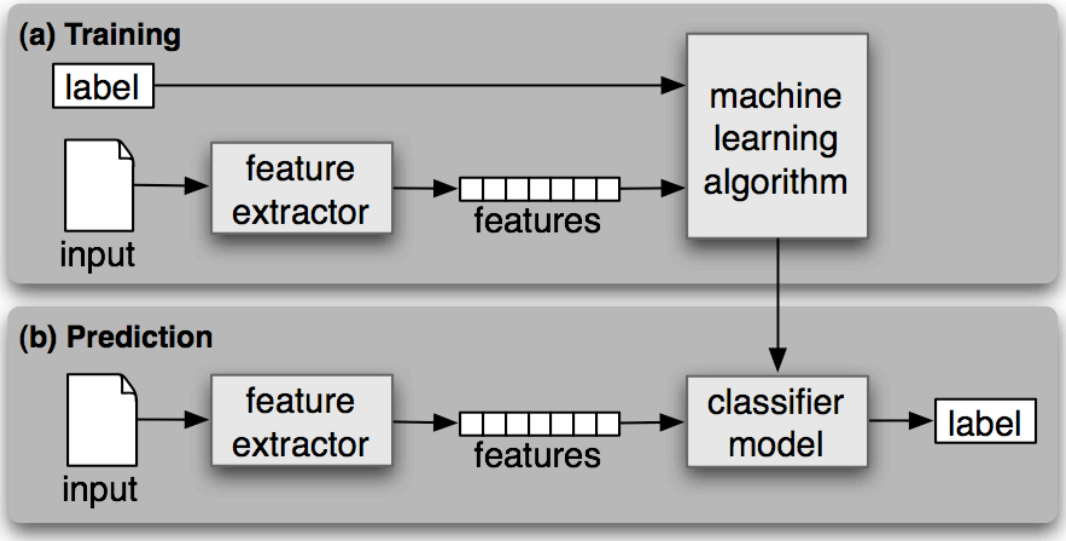


Figure 3.1.5a: Diagrammatic depiction of supervised learning classifier model (Bird, 2009)

Therefore, in order to better train the classifier, thought must be given to the pre-labelled dataset that will be used; indeed, the accuracy of the classifier will directly depend on the choice of dataset. Pre-packaged with NLTK is a movie review dataset first introduced by Pang & Lee (2004) containing 1000 positively labelled reviews and 1000 negatively labelled reviews; a more sizeable sentence polarity dataset, however, is available, containing 5331 positive and 5331 negative "processed sentences" (Pang & Lee, 2005). The dataset that is chosen will be split into a training set (80%), which will be fed into the machine learning algorithm to create a classifier prediction model, and a testing set (20%) used to evaluate the accuracy of the classifier. A method proposed by Bird (2009) is shown overleaf, in which the dataset is separated into a **development set** and a **test set**; the development set is subdivided

into the standard training set, but it also includes a **dev-test** set, which can be used to perform **error analysis** – a "very productive method for refining the feature set". This method will be considered during the implementation phase.

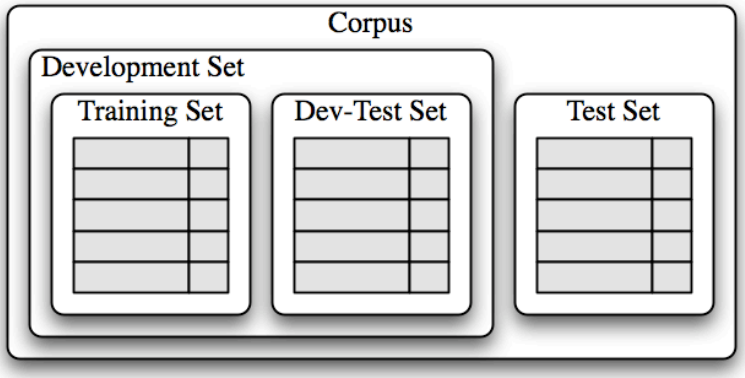


Figure 3.1.5b: Division and subdivision of dataset for supervised classifiers (Bird 2009)

3.1.5 Testing the Classifier

Testing the trained classifier will not only ensure that the accuracy of the algorithm matches the benchmark standards in place, but will allow tweaks to be made to deduce which extracted features are more precise with regards to deducing the sentiment of a document. The "most intuitive and widely used method" to test a classifier is to assess its accuracy, which can be done using the testing set (20%) by dividing the number of correct predictions by the total number of predictions made.

Another form of evaluating will be done on the dataset itself to ensure that the classifier model is not over-performing, or "over-fitting" on the training set relative to the test set; the ideal outcome is "harmony between train-set performance and test-set performance" (Potts, 2011). While this kind of problem is a bigger concern for Maximum Entropy models, it will be taken into consideration during the training and testing of classifiers for this project.

Importantly, the effect of the pre-processing stages will be evaluated by comparing the accuracy of a classifier model trained on the unchanged, raw dataset versus a model trained on the dataset after it has gone through the pre-processing pipeline. Individual pre-processing techniques will be experimented with, and incremental improvements will be made to the finalised classifier model based on their contribution to the overall performance.

3.2 User Stories

End User
Description: Everyday user of the system, who would use the tool to find out the sentiment of the text they have chosen for analysis, be it from directly typing in the text, a document on their device, or a review parsed from an online source.
Example: A customer who is interested in a particular product, be it a book or a movie, and wants to know the sentiment towards the product before making an informed decision on whether or not to purchase.

Developer
Description: Tasked with creating and maintaining the system, keeping it up to date and bug-free. Developers may want to add features to the tool to improve its performance, or to alter the classifier to cater towards another domain.
Example: Technical Staff, Software tester

Story	Title	As a…	I want…	Because	Acceptance Criteria
1	Functionality	Developer /User	The program to be able to detect sentiment from a piece of inputted text and classify the text as positive or negative	This is the system's central function	Can perform these basic functions without error
2	Additional Features	Developer	The system to allow for updates in the future or alterations to the fundamental sentiment analysis algorithm	There are always ways to improve the efficiency of the system	Allows changes such as addition or removal of stopwords, room for additional features to be added and updated
3	Start Program	User	To be able to start up the program without error and receive directions on how to use	The system must be user friendly and this is the initial impression that will be left on the user	System opens without trouble and relevant information is displayed
4	Close Program	User	To have the option to exit the program	Once I am finished I will want to exit	User can terminate the program
5	Input Text/Select Text Document	User	To be able to enter text input as well as choose a document to be analysed by the program.	Input is needed for the program to perform its function	User can enter text, and select a file to be processed
6	Sentiment Analysis	User	To be able to enter a review and have the sentiment displayed in the console	Print statements allow for the process to be broken down and made transparent	Appropriate information displayed in the console with information relating to the sentiment of the text entered as well as the features used to classify

3.3 User Interface Design

As stated in the Analysis & Design report, the functionality of the Sentiment Analysis tool will be the main focus given the complexities of Natural Language Processing, and as such the initial interface will consist of a very simple command line menu that will utilise printed menus that the user will navigate through number choice. This will be done through the use of if, elif and else statements within Python. A simple mock-up of what this command-line based menu might look like is shown in the figure below.

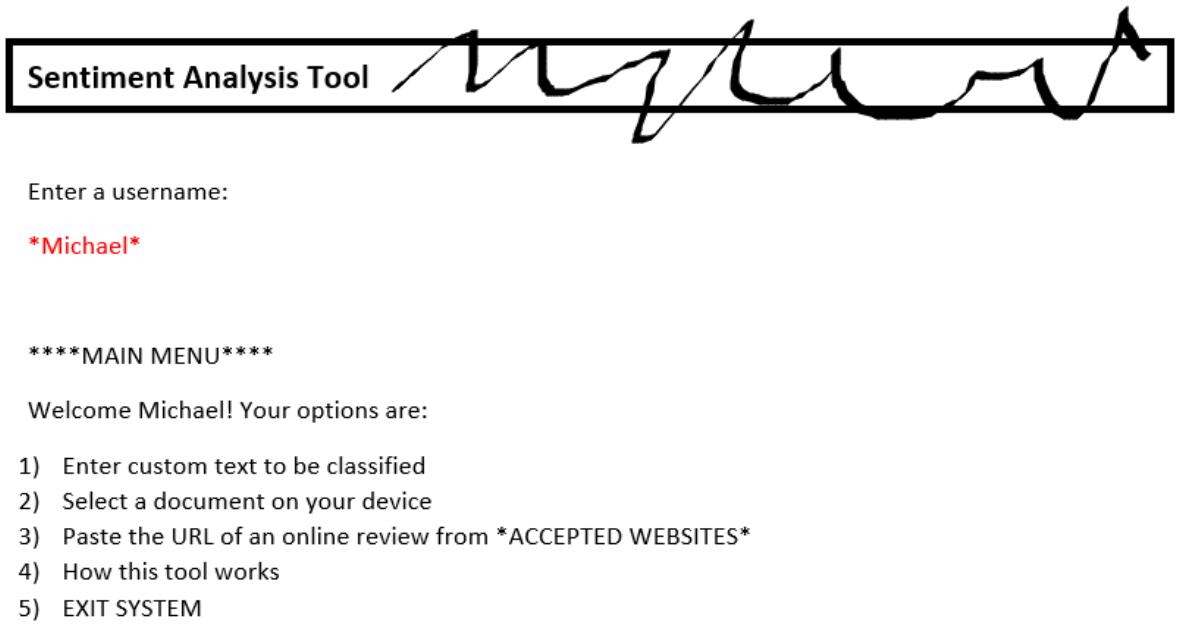


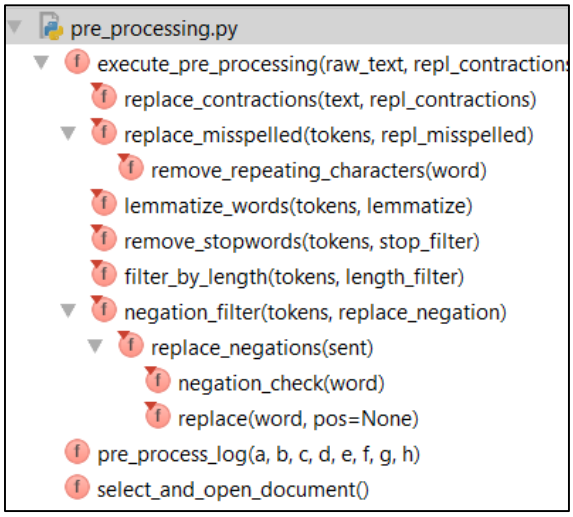
Figure 3.3: Console Menu Design Mock-up

After the tool itself has been built, a simple GUI may be implemented, and given enough time, a simple webpage, however as stated the focus of this project is not on the front-end of the system but the underlying architecture and accuracy of the classifier being trained.

4. Implementation, Testing, and Evaluation

4.1 Implementation

4.1.1 Pre-processing module



As detailed in the Architectural Design section (Chapter 3.1.3) of this report, the pre-processing stage is an important stage in the NLP pipeline for text classification and sentiment analysis. Thus, the early iteration of the tool focused on implementing the pre-processing module; this involved becoming familiar with the NLTK package by reading its documentation, and using tutorials where available. An NLTK tutorial series of particular note entitled "Natural Language Processing w/ Python" by Harrison Kinsley offered invaluable guidance and advice on pre-processing techniques and sentiment analysis (https://pythonprogramming.net/tokenizing-words-sentences-nltk-tutorial/).

With the aim of consolidating all of the pre-processing steps into a concise running order, the pre_processing module's main function, **execute_pre_processing()**, contains nested functions for each stage of pre-processing. This function takes the raw, unfiltered text as an argument, as well as several Boolean values indicating which stages of pre-processing to include or leave out; this ensures that when testing individual pre-processing stages, it is a simple matter of setting the Boolean for that specific step to **False**.

```
def execute_pre_processing(raw_text, repl_contractions=True, repl_misspelled=True,
                           lemmatize=True, stop_filter=True,
                           length_filter=True, replace_negation=True):
```

As shown above, each of the Boolean values are set to True as default. This is so that each time the pre-processing stage is being called, the only required argument is the text to be processed. Within this function is a series of sequential calls to each of the nested pre-processing functions – detailed below – in a logical order; before these calls, however, the text is normalised by removing excess white space and forcing it into a lower case String.

```
string_text = str(raw_text)
normalized_text = string_text.strip().lower()
```

4.1.1.1 Replace_contractions

```
def replace_contractions(text, repl_contractions):
```

This function was adapted from Jacob Perkin's *Python 3 Text Processing with NLTK 3 Cookbook* and is the first function called in the pre-processing pipeline. The parameter `repl_contractions` is a Boolean value indicating whether to run this process or not, and is dependent on the Boolean from the call to `execute_pre_processing`. Using *regular expressions*, a list of replacement patterns – shown below – are compiled; any of these patterns identified in the inputted text are replaced with the corresponding String. For example, the regular pattern `(r'(\w+)\'ll', '\g<1> will')` identifies groups of letters that end in `'ll` and replaces it with the word `will`. Thus "they'll" would become "they will". It was decided that this would be the first stage after normalization, even before tokenization, because using NLTK's *WordPunctTokenizer*, punctuation is separated and thus it would have made this stage redundant.

```
replacement_patterns = [
            (r'won\'t', 'will not'),  # won't → will not
            (r'can\'t', 'cannot'),    # can't → cannot
            (r'i\'m', 'i am'),              # I'm → I am
            (r'isn\'t', 'is not'),    # isn't → is not
            (r'ain\'t', 'is not'),    # ain't →is not
            (r'(\w+)\'ll', '\g<1> will'), # 'll → will
            (r'(\w+)n\'t', '\g<1> not'),  # n't → not
            (r'(\w+)\'ve', '\g<1> have'), # 've → have
            (r'(\w+)\'s', '\g<1> is'),      # 's → is
            (r'(\w+)\'re', '\g<1> are'),  # 're → are
            (r'(\w+)\'d', '\g<1> would')  # 'd → would
        ]
```

Not only does this function make the text cleaner for tokenization, it also helps simplify the process of identifying negation as a lot of negation phrases, for example "I don't like this movie", contain contractions; using the above method, the example becomes "I do not like this movie" and identifying negation is a simple matter of searching for instances of the word "not". Once a call to this function has been made, and contractions have been replaced, the text is broken up into its token components.

```
from nltk.tokenize import WordPunctTokenizer

tokenizer = WordPunctTokenizer()
```

4.1.1.2 Replace_misspelled

```
def replace_misspelled(tokens, repl_misspelled):
```

This function takes as arguments the tokenized text and a Boolean value indicating whether it executes or returns the tokens unchanged. This is another function adapted from Perkins *NLTK Cookbook*; where the initial implementation was a class, here it is a simple function. Within is a nested function `remove_repeating_characters` which accepts a word as a parameter; using regular expressions, repeating characters are removed from said word until its spelling is correct. To ensure that words like "Goose" or "Moose" – perfectly valid spellings containing repeating characters – are not reduced to "Gose" and "Mose", for example, the word is first checked against WordNet to see if it exists; if it is found using WordNet that word is returned, if not the function calls itself recursively until the appropriate word is returned.

```
replaced_tokens = []

if repl_misspelled:
    for token in tokens:
        clean_token = remove_repeating_characters(token)
        replaced_tokens.append(clean_token)
    # END for
# END if
```

Above shows how a `for` loop is used to iterate through each token and passes it through the `remove_repeating_characters` function. Each `clean_token` is then added to a new list of `replaced_tokens` which is then returned at the end of the function. It is worth noting that this function precedes lemmatization and stopword filtration in an attempt to improve the efficiency of these latter steps.

A second nested function was planned within `replace_misspelled` which was going to utilise the *Enchant* library to fix minor spelling errors. However, problems were faced when trying to install the necessary packages along with the Anaconda Python distribution that is being utilised by the tool.

4.1.1.3 Lemmatize_words

```
def lemmatize_words(tokens, lemmatize):
```

Lemmatization was chosen over Stemming, because of the aforementioned reason that Stemming crudely cuts off the end of a word, whereas Lemmatizing always returns a valid word; more specifically it returns the normalized base form of the passed in word.

```
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(w) for w in tokens]
```

Using the *WordNetLemmatizer* algorithm, an instantiation of which can be seen above, each word's lemma – if one is found – is returned. This stage occurs before stopword filtration because logically, normalizing words would increase the efficiency of stopword removal.

4.1.1.4 Remove_stopwords

```
def remove_stopwords(tokens, stop_filter):
```

As mentioned before, stopwords are words that do not inform the sentiment of the text one way or the other, and are therefore removed for more efficient feature extraction and negation identification. The list of stopwords used for this process – a default English stopwords list (http://www.ranks.nl/stopwords) edited to remove negation words and added to incrementally – is read in from a text file in the local directory, which is then used to filter the tokens using a one line command in Python.

```
filtered_tokens = [w for w in tokens if w not in stopwords]
return filtered_tokens
```

A new list of tokens is created, containing only the words that are not in the stopwords list. This aids the process of negation handling in that it has the potential to remove words in between a negation phrase and words that inform the sentiment one way. For example, the phrase "not a good movie", once filtered would become "not good", thus allowing the negation to be accounted for.

4.1.1.5 Negation_filter

```
def negation_filter(tokens, replace_negation):
```

The code for this function has been adapted from Perkins *NLTK Cookbook*, which was initially implemented as a class. As discussed in the Architectural Design section, this function aims attempts to replace words that are preceded by negation phrases with "unambiguous antonyms" (Perkins, 2014); when an unambiguous antonym is not found, the word is instead returned with the prefix "NOT_" as discussed earlier. An alteration to Perkins initial implementation, the nested function `negation_check` is used to identify negation words against a custom list, and determines whether or not the `replace` function is called.

```
negation_words = ['not', 'no', 'nothing', 'never', 'hardly', 'barely', 't']

if word in negation_words:
    return True
```

This list can potentially be expanded upon to catch more negation phrases and poor spelling, however because contractions have been replaced already, identifying negation words is made easier thanks to many of the contractions being replaced by the word "not".

```
if len(antonyms) == 1:
    unambig_antonym = antonyms.pop()
    return unambig_antonym
# END if
else:
    return "NOT_" + word
# END else
```

Once, the `replace` function is called, WordNet's lexical database is searched for the respective word's antonym; if the condition of the `if` statement `len(antonyms) == 1` evaluates down to true, then an unambiguous antonym has been found and the replacement is made. If it evaluates to false, then the word is returned with the prefix "NOT_" to nullify the sentiment of the word following the negation word.

Both the `replace` and `negation_check` functions are consolidated in a simple `while` loop which iterates through the list of tokens, replacing any negation terms as well as the word immediately following.

```
i, l = 0, len(sent)
words = []
while i < l:
    word = sent[i]
    if negation_check(word) == True and i+1 < l:
        ant = replace(sent[i+1])
        words.append(ant)
        i += 2
        continue
```

4.1.1.6 Filter_by_length

```
def filter_by_length(tokens, replace_negation):
```

Words under two characters in length would not offer any informative information in regards to the classification of a document, and as such they are removed in this simple function, making use of the `len(w)` function which evaluates down the length of each String that is passed into it.

```
filtered_tokens = [w for w in tokens if len(w) > 2]
```

4.1.1.7 Select_and_open_document

```
def select_and_open_document():
```

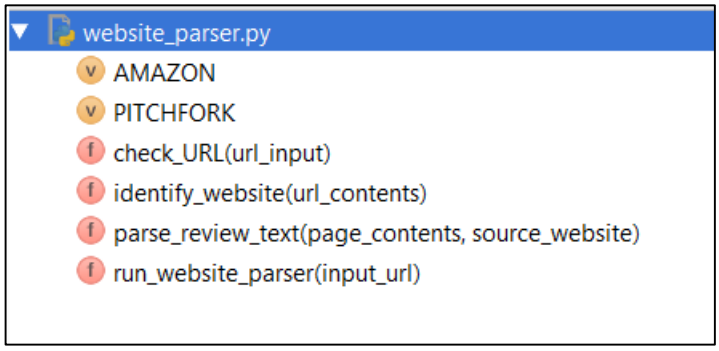
This function exists outside of the above mentioned `execute_pre_processing` function. Using **Tkinter**, Python's standard GUI package, it allows the user to navigate the directories on their device and select a document for sentiment analysis. This is achieved through the `filedialog` modules `askdirectory` and `askopenfilename`, which prompts the user to choose a directory before selecting a file they wish to open.

```
filedialog.askopenfilename(filetypes=[("Text files", "*.txt;*.docx")]
```

Passed in as parameters to the `askopenfilename` function, as shown above, the supported formats are specified as ".txt" and ".docx" file types, and thus will be the only files to appear in the Tkinter dialog window. Once the document is selected, its contents are read in depending on the file format; .txt files are simply read in, whereas .docx files have their text extracted using the **docx2txt** utility before it is written to a temporary .txt file in the project's data folder. The review text is then read in from the temporary file and passed back to the function call, where it will undergo pre-processing and sentiment analysis.

```
if str(file_name).endswith(".docx"):

                text = docx2txt.process(file_name)
                new_file.write(text)
                new_file.close()

review_text = open("data/temp_file.txt", "r").read()
```

4.1.2 Website Parser module



The purpose of this module is to return the review text of a website either pasted or typed in by the user and its main function **run_website_parser()** is called from the console menu module when the appropriate menu option is selected by the user. Once prompted, the user enters the URL and hits enter; this URL is passed in as a parameter to the above mentioned run_website_parser() function, which serves to neatly consolidate the rest of the module's functions. Since each website must be parsed in its own unique way, the choices of supported website sources are limited; at the time of this project's completion, Amazon was the only supported website due to complications with connecting to Goodreads' and Pitchfork's servers.

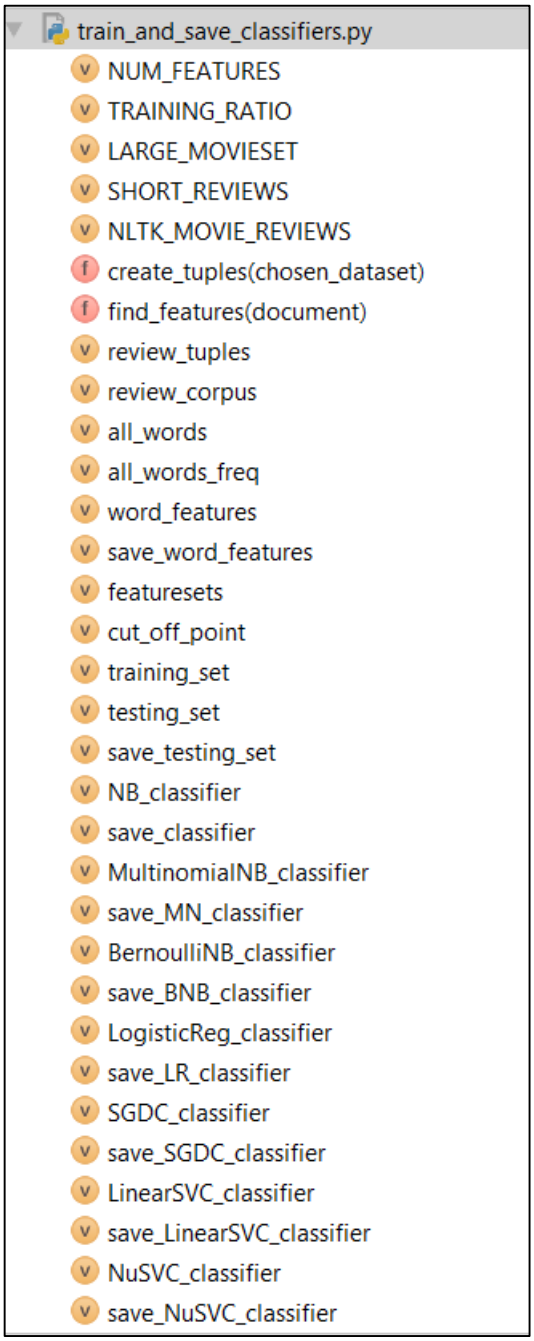
Before any formatting is done, the inputted is URL is validated by the **check_URL()** function, which makes use of regular expression patterns and Python's **re** module:

```
if re.match('https?://(?:www)?(?:[\w-
]{2,255}(?:\.\w{2,6}){1,2})(?:/[\w&%?#-]{1,300})?', url_input):
    url_contents = urlopen(url_input).read()
    return str(url_contents)
```

If the URL entered matches the regular expression pattern, then the raw HTML content is read using the **urllib** package's module `urlopen` to send a request to open the URL. This is then returned as a String value. Once validated, the source of the review is identified through the simple function **identify_website()** which takes the contents of the URL as a parameter and returns the website source as well as the contents if the page is supported, or returns a String informing the user that the website is currently unsupported. The final function, **parse_review_text()** makes use of the **BeautifulSoup** library to scrape the contents from the webpage based on the specific website source. The highlighted code below specifies where to look in the Amazon page source for the relevant review text to be extracted, which is then passed back for pre-processing and analysis.

```
review_text = ""
soup = BeautifulSoup("".join(page_contents), "lxml")
amazon_review = soup.findAll("div","reviewText")
```

4.1.4 Train Classifiers module and Revised Approach



In the Architectural Design phase and first iterations of the tool, the design of the classifier was focused on implemented a simple Naïve Bayes algorithm, with the intention of rigorously testing and improving its accuracy through experimenting with pre-processing and the number of words used as features for extraction. Following this, the **scikit-learn** library was utilised to train a number of classifiers, and using **NLTK**'s **SKLearnClassifier** made it possible to "implement a wrapper around scikit-learn classifiers" (NLTK.org, 2016), allowing them to be treated as NLTK classifiers when calculating accuracy or returning the most informative features for example. It was finally decided that, rather than training these classifiers only to choose the most accurate, they would be combined into one classifier that utilised a voting system, as suggested by Jacob Perkins in his *NLTK Cookbook* and described in the Classifier Design section of this report. By using this approach, resources would not be wasted training the classifiers, and the overall aim was to create an all-encompassing classifier that was more reliable than any one on its own.

Creating Tuples

For training and testing purposes, three sets of data sets were made readily available to train the classifier. Each of these data sets must be read in through different means, as they are each stored uniquely. NLTK's movie review dataset, for example, is conveniently stored in NLTK's corpus library and is read in word at a time, whereas the short reviews –the download link for which can be found here: https://pythonprogramming.net/static/downloads/ - are stored in two .txt files labelled "neg" and "post" in the local data directory and split by the new line escape character as they are read in. Finally, each individual review in the large movie review dataset (Maas et

al. 2011) is stored in its own separate .txt file; the **os** module is used to compile a list of each review's file name, which is then iterated through and the contents of each review is read in one at a time. The **create_tuples()** function takes a String parameter indicating which of the aforementioned datasets to use, and by navigating if/elif/else statements, each review from the chosen dataset is appended to a list of tuples, an example of which is shown below.

```
>> review_tuples[1]
>> [("This movie was truly great","pos")]
```

Once the tuples have been compiled, each review is individually passed through the pre-processing pipeline and a new list is built; this is known as the corpus.

```
for review, tag in review_tuples:
    filtered_review = pre.execute_pre_processing(review)
    review_corpus.append((filtered_review, tag))
    all_words.extend(filtered_review)
# END for
```

Building Features

The word features play a crucial part in training the classifier; those words selected to be features will be used to predict the sentiment of the user's input. As previously discussed, the number of features used would be specified as the top N most frequently occurring words; thus it was necessary to create a frequency distribution based on all words in the dataset.

```
all_words_freq = nltk.FreqDist(all_words)

word_features = []
for w in all_words_freq.most_common(NUM_FEATURES):
    word_features.append(w[0])
# END for
```

The list `word_features` is used to store the most frequently occurring words in the range 0 - `NUM_FEATURES`; this being a global variable which specifies the number of features desired by the developer.

Feature Extraction

The **find_features()** function is used both to create training testing featuresets with which to feed into the classifier model, and to identify features on text inputted by the user when labels for sentiment analysis.

```
def find_features(document):
    words = set(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    # END for
    return features
# END find_features
```

Utilising a for loop, this function iterates through each word in the word features list (as shown above) and returns a dictionary of each word/feature (*key*) and a corresponding Boolean (*value*) indicating the presence or absence of the word.

```
{"love": false, "hate": true, "amazing": false}
```

Training and Testing Featuresets

To produce optimal results, research suggested an 80/20 split of the training and testing sets; to this end, the global variable `TRAINING_RATIO` is utilised to calculate the cut-off point between the two.

```
featuresets = [(find_features(rev), category) for rev, category in review_corpus]

cut_off_point = int(len(featuresets) * TRAINING_RATIO)
training_set = featuresets[:cut_off_point]
testing_set = featuresets[cut_off_point:]
```

Training and Saving the Classifiers

Because of the substantial amount of time it takes to train each of the classifiers – largely due to sifting through and processing the datasets – it was decided that this module would be a once-and-done execution as opposed to a Class. This decision was made upon discovering the **pickle** package, which is "used for serializing and de-serializing a Python object structure" (Yasoob, 2013). This meant that the classifiers could be trained and pickled in this module, and loaded into the **sentiment_module** where they were used to create the **Vote Classifier**, discussed in the next section. This cut the processing time down by an enormous margin;

where it took upwards of 40 minutes to train a classifier now only takes a minute or less to load.

```
MultinomialNB_classifier = SklearnClassifier(MultinomialNB())
MultinomialNB_classifier.train(training_set)
print("Multinomial Classifier % accuracy:",
 (nltk.classify.accuracy(MultinomialNB_classifier, testing_set))*100)
```

Above shows the code used to train the scikit-learn Multinomial Naïve Bayes classifier; as can be seen, an instance of the MultinomialNB() is wrapped using the SklearnClassifier and the **train()** and **accuracy()** methods are used as part of the NLTK **classify** package. Each classifier is trained in the same way before being pickled to a destination folder in the local directory.

```
save_MN_classifier = open("data/pickled_files/MultinomialNaiveBayes.pickle", "wb")
pickle.dump(MultinomialNB_classifier, save_MN_classifier)
save_MN_classifier.close()
```

Because it is an object being saved, the input stream is specified, when open, to "wb" – write bytes. The object being saved as well as the save location are passed in as parameters to pickle's **dump()** function, before the input stream is finally closed.

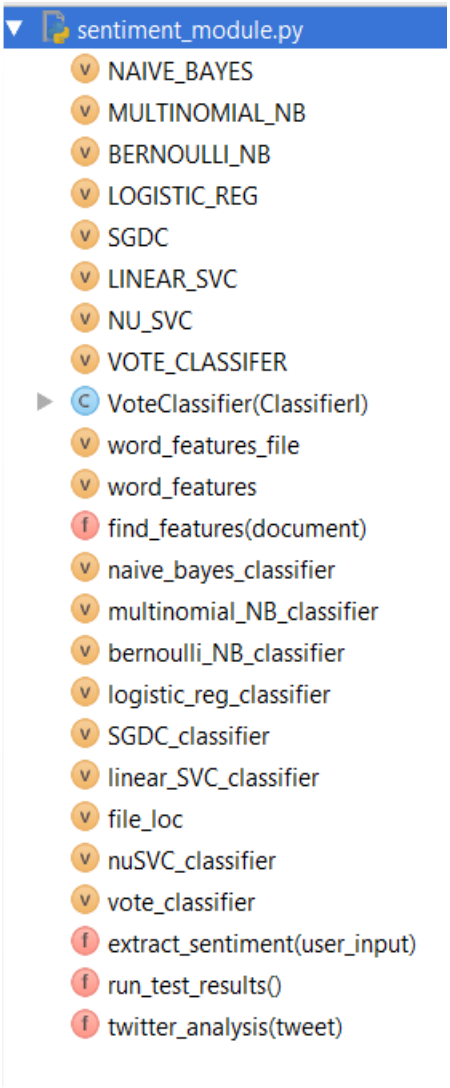
The figure below shows an example print statement displaying the accuracy of the NLTK Naïve Bayes classifier and its 15 most informative features. As can be seen, the several nouns unique to the dataset have not been filtered during stopword removal; though it does not have any great effect on the classifier's accuracy due to the number of features being 5000, the stopword list can be easily amended to pick up domain-distinct words.

```
Naive Bayes % accuracy: 85.08
Most Informative Features
              unwatchable = True              neg : pos    =     30.0 : 1.0
                   paulie = True              pos : neg    =     19.1 : 1.0
                 NOT_save = True              neg : pos    =     17.8 : 1.0
                 flawless = True              pos : neg    =     16.8 : 1.0
               incoherent = True              neg : pos    =     16.7 : 1.0
                  unfunny = True              neg : pos    =     16.6 : 1.0
                     yawn = True              neg : pos    =     14.9 : 1.0
                   mathau = True              pos : neg    =     14.5 : 1.0
                  stinker = True              neg : pos    =     13.2 : 1.0
                   drivel = True              neg : pos    =     12.2 : 1.0
                   gundam = True              pos : neg    =     11.8 : 1.0
                atrocious = True              neg : pos    =     11.7 : 1.0
                 conserve = True              neg : pos    =     11.5 : 1.0
                masterful = True              pos : neg    =     11.4 : 1.0
                 superbly = True              pos : neg    =     11.4 : 1.0
```

Figure 4.1.4 Output from training Naïve Bayes classifier

4.1.5 Sentiment module and Vote Classifier Class

Within this module, the saved classifiers are loaded in from their previously saved pickle files. They are then used to create an instance of the class **VoteClassifier**, which is detailed below. The main functions in this module are the **extract_sentiment()** and **run_test_results()** functions, which are called from the console menu when the appropriate option is chosen. The extract_sentiment() function brings together all of the steps involved the sentiment analysis process, returning appropriate print statements to display:



- The original text entered by the user
- The filtered text after it has been through the pre-processing pipeline.
- The features identified and used to predict the label
- The overall sentiment of the text, as determined by the Vote Classifier.
- Finally, the confidence with which the Vote Classifier made the decision, which is explained in the next section.

The VoteClassifier

```
def __init__(self, *classifiers):
        self._classifiers = classifiers
```

The _init_ method of a class determines what happens when an instance of that class is created; here, the constructor stipulates that multiple classifiers are to be passed in as arguments. They are then stored in a list of classifiers. The Vote classifier class contains three major functions: the **classify()** function, the **confidence()** function, and the **features_present()** function.

```
def classify(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)
        return mode(votes)
```

Based of Perkins' implementation, the classify function takes as a parameter the features that will be used to perform the classification. A for loop is used to iterate through the classifiers, each in turn classifying the features; the label determined by each classifier is appended to a votes list. Using the **mode** function, the most commonly occurring label is returned to where the function was called from. It is worth noting that it is for this reason an odd number of classifiers were used; a voted label will always be returned. The **confidence** function works in much the same way, except for one slight alteration:

```
choice_votes = votes.count(mode(votes))
conf = choice_votes / len(votes)
return conf
```

The above code takes a count of the number of winning votes and divides it by the total number of votes to give a number between 0 and 1 indicating the confidence with which the classifier made the decision. This is then aptly converted to a percentage when being displayed to the user. The features_present function is a simple yet useful method of allowing the user to see the words from their review text that were identified as features from the featureset; the process of sentiment analysis thus becomes more transparent.

```
def features_present(self, features):
      important_features = []
      for word, present in features.items():
          if present == True:
              important_features.append(word)
          # END if
      # END for
```

For every word in the dictionary of features passed in, if the Boolean indicating its presence is true, then it is appended to a list of `important_features`. This list of features is then concatenated to a String and returned to the console menu module to be displayed to the user.

```
string_features = ", ".join(important_features)
```

4.3 Testing

The iterative nature of the project meant that testing was an ongoing process throughout the tool's development. Although test-driven development was not strongly emphasised, potential errors were considered as the tool's development progressed from its early to final stages; to aid in the process, Python's **logging** utility was used to record and keep a log of the pre-processing, training and classification processes. This meant that, despite maybe not being a code-breaking error, any minor oversights could be seen in the log .txt file which is held in the local directory. An error which raised an issue was to do with encoding the large movie review dataset.

```
--- Logging error ---
Traceback (most recent call last):
  File "C:\Users\MickTheHuman\Anaconda3\lib\logging\__init__.py", line 982, in emit
    stream.write(msg)
  File "C:\Users\MickTheHuman\Anaconda3\lib\encodings\cp1252.py", line 19, in encode
    return codecs.charmap_encode(input,self.errors,encoding_table)[0]
UnicodeEncodeError: 'charmap' codec can't encode character '\u014d' in position 227: character maps to <undefined>
```

It was deduced that the reviews being read in needed to be explicitly encoded to the "utf-8" format, and the error was soon resolved. A more simple to fix error that the developer came across was the **UnboundLocalError**, as shown below.

```
Traceback (most recent call last):
    File "C:/Users/MickTheHuman/Documents/Uni Documents/Python Source Code/SentimentToolFinal/test.py", line 2, in <module>
        sentiment_module.test_results("nic")
    File "C:\Users\MickTheHuman\Documents\Uni Documents\Python Source Code\SentimentToolFinal\sentiment_module.py", line 243, in test_results
        observed = classifier.classify(feats)
UnboundLocalError: local variable 'classifier' referenced before assignment
```

Thanks to a thread on StackOverflow (http://tinyurl.com/zpxklcb) the source of the error was determined to be a variable that was being referenced when it was only being assigned a value (and thus bound) if a condition was being met. It was soon resolved by having an assignment statement somewhere that was always going to be executed.

4.3.1 Testing Console Menu with Valid Input

Tests were done each of the menu options to make certain that no errors were thrown when users attempted to enter text, select a document, or paste a URL. The output from the sentiment analysis stage was validated and verified to ensure that there were no major flaws in the classifier's algorithm.

User enters custom text: This is menu option 1, and entering this option functions correctly, as can be seen below. The user is prompted to enter their custom text for analysis, and the output returned displays the overall sentiment correctly for the examples below. The classifier picked up on a number of features that were used to predict the label which are all displayed, as well as the confidence of the vote classifier.

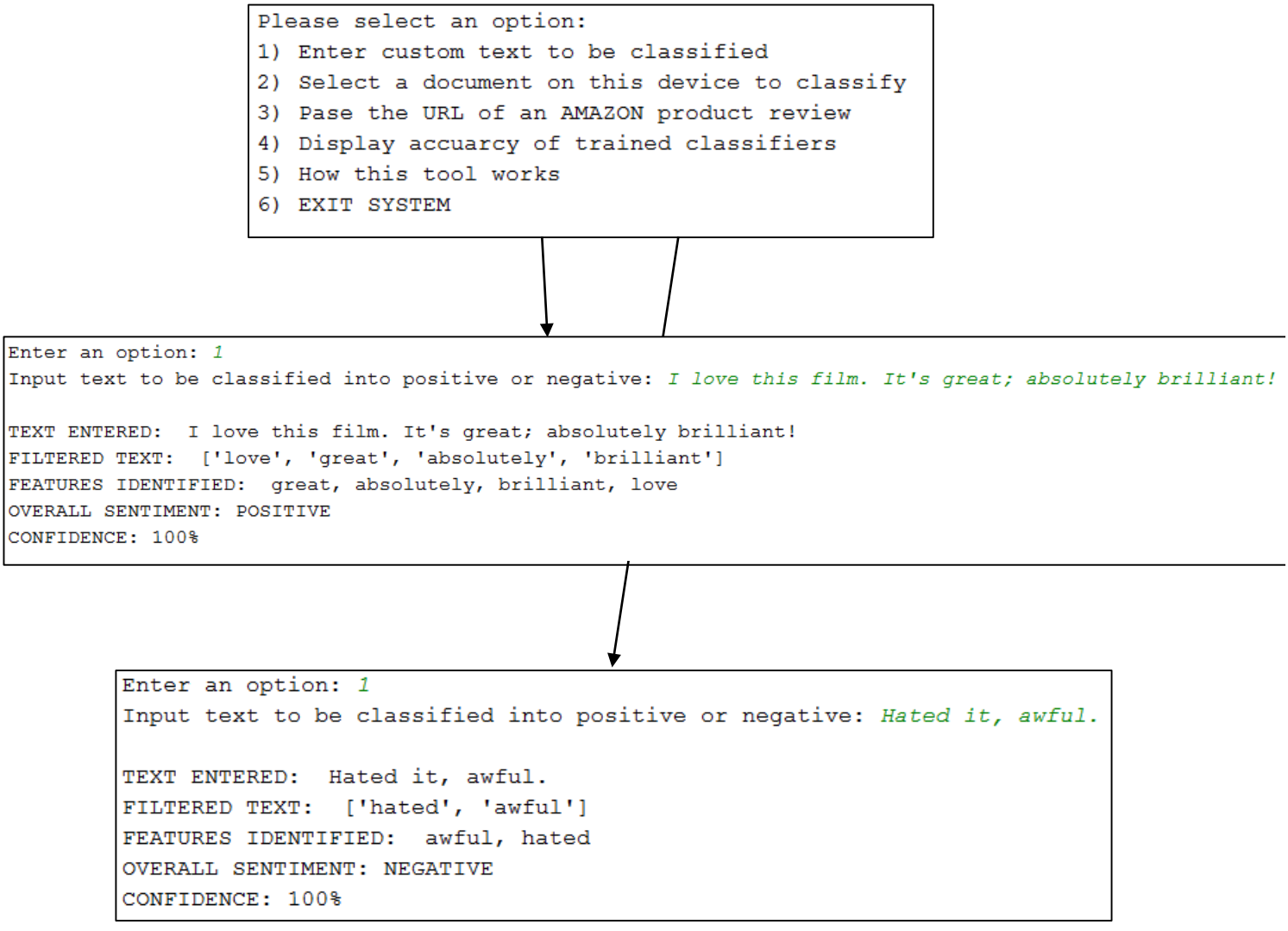


Figure 4.3.1a: Custom Entered Text Menu Option

User Selects Document for Analysis: The relevant browser window prompting the user to select a directory is followed by the file browser window, prompting the user to select a file. The file options available to the user, as specified in the code, are .txt and .docx files. When a docx file is selected, its contents are successfully extracted and the sentiment performed on it is accurate.

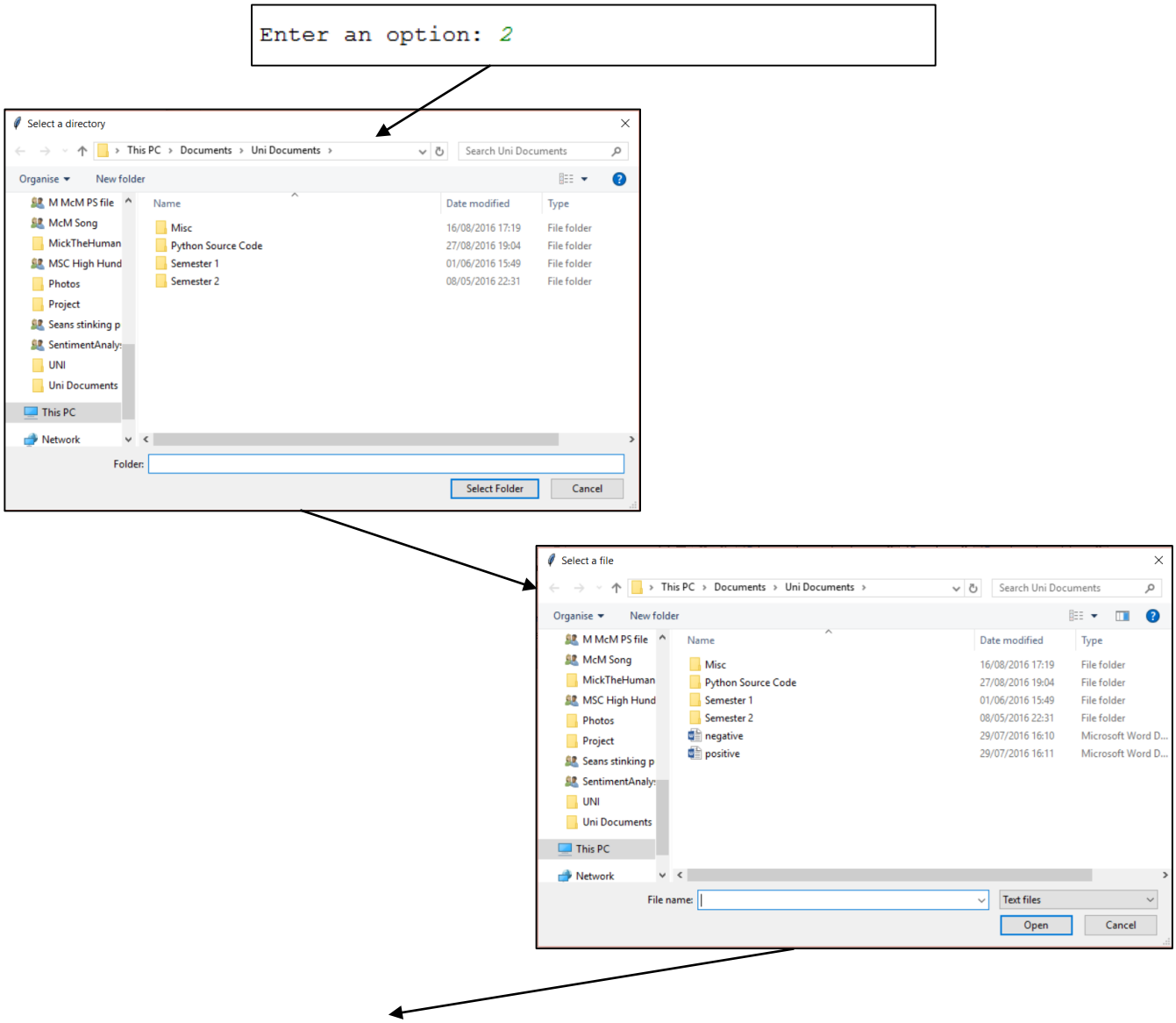


Figure 4.3.1b: User selects document for analysis

User pastes URL of Amazon Product Review: As expected, the URL pasted in by the user has its review text parsed using the website_parser module, which is then fed in to the classifier for sentiment analysis, which is again performed accurately.

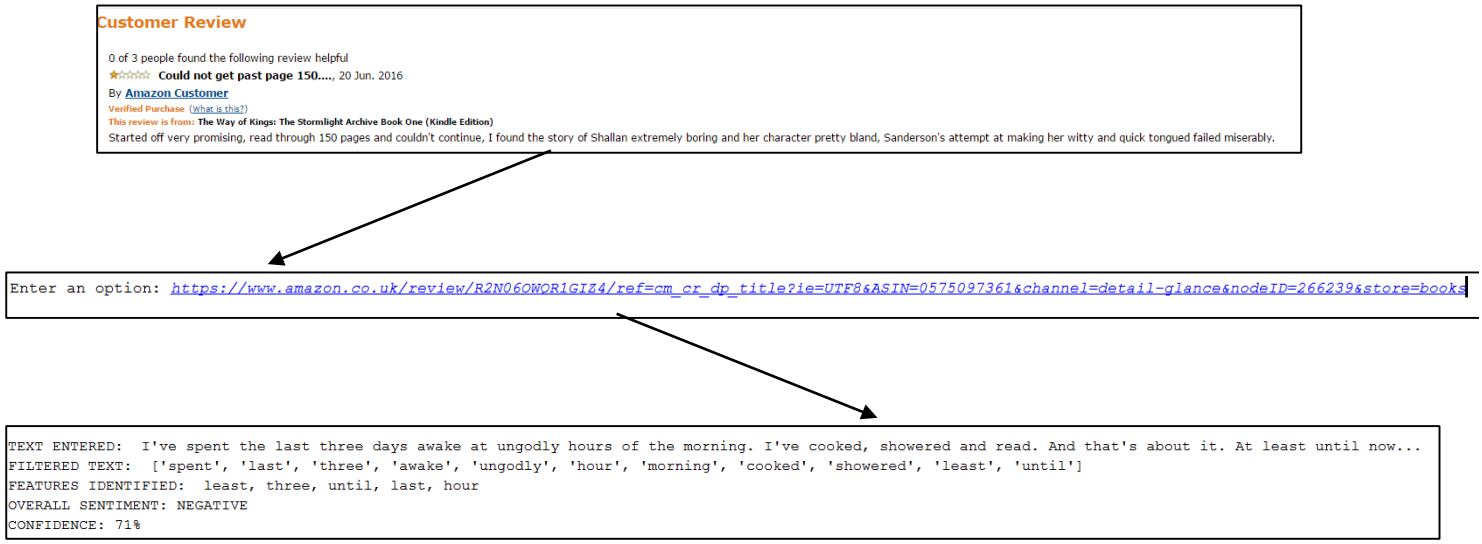


Figure 4.3.1c: User pastes URL of Amazon Review

4.4 Evaluation

4.4.1 Problem Identifying Features in Tweets

```
TWEET: @hof_zaari happy birthday bro 🤘
SENTIMENT: neg
CONFIDENCE: 0.5714285714285714
FOUND FEATURES: No features present

TWEET: happy birthday doll 💝 @PriscilaGamez
SENTIMENT: neg
CONFIDENCE: 0.5714285714285714
FOUND FEATURES: No features present

TWEET: RT @ferriol_gale: Everybody are happy, enjoying the ocean's cool breeze.

TOMIHO TVplusCaravan InCEBU
#PushAwardsToMihos https://t.co/CHU0K…
SENTIMENT: neg
CONFIDENCE: 0.5714285714285714
FOUND FEATURES: No features present

TWEET: RT @DavidRoads: Nobody really cares if you're miserable, so you might as well be happy. -Anonymous
SENTIMENT: pos
CONFIDENCE: 0.7142857142857143
FOUND FEATURES: well

TWEET: RT @hippiecasual: if it makes you happy it doesn't have to make sense
SENTIMENT: neg
CONFIDENCE: 0.5714285714285714
FOUND FEATURES: No features present
```

Figure 4.4.1: No features identified in Tweets

As can be seen from the figure above, a problem was encountered when it came to classifying the live stream of tweets. This was most likely due to the fact that the classifier was trained on a dataset that does not contain features unique to tweets, as well as a stopword list that does not take into account the punctuation and peculiar grammar used on Twitter. As such, the twitter_stream module was excluded from the final version of the tool.

4.4.2 Pre-processing & Number of features evaluation

Pre-processing Step	NLTK Naïve Bayes	Multinomial Naïve Bayes	Bernoulli NB	Logistic Regress	SGDC	Linear SVC	NuSVC
NO PRE-PROCESSING	78.75	78.25	78.75	79.5	76.5	77.25	78.75
Replace Contraction	79	80	79	79	76	77.25	81.25
Stopword Filter	80.25	82.25	79.75	79	77	78.25	81.25
Replace Misspelt Words	78	77	77.75	78.5	77.5	76	81
Replace Negation	77.5	79.25	77.25	79	76.75	75	81.25
Length Filter	79.5	79.5	79.25	78.7	79	78.25	79.5
Lemmatization	79.25	80	79	78.5	80	76	80
ALL PRE-PROCESSING	78.25	81	78	80	78.75	80	81

Figure 4.4.2: Table illustrating the effects of individual stages of pre-processing on the classifier's accuracy

As shown from the results table above, more often than not the removal of stopwords from the dataset produces a trained classifier with the best results. An unexpected outcome is the negative effect the replacement of negations would have on the accuracy of the overall classifier. Perhaps with a larger dataset these results would vary slightly; it was the developers intention to test the effect of pre-processing on each of the datasets available, however due to an oversight and an underestimation of how long it takes to train the classifiers, this was regretfully not the case.

The developer notes that increasing the number of features had a direct correlation on the accuracy of the classifiers until it eventually levelled off at the 10,000 features mark.

4.4.3 Classifier Evaluation

```
Multinomial Classifier % accuracy: 85.86
Bernoulli Classifier % accuracy: 85.08
Logistic Regression Classifier % accuracy: 85.6
SGDC Classifier % accuracy: 84.46000000000001
LinearSVC Classifier % accuracy: 82.54
NuSVC Classifier % accuracy: 87.24

Naive Bayes % accuracy: 85.08
Most Informative Features
              unwatchable = True              neg : pos    =     30.0 : 1.0
                   paulie = True              pos : neg    =     19.1 : 1.0
                 NOT_save = True              neg : pos    =     17.8 : 1.0
                 flawless = True              pos : neg    =     16.8 : 1.0
                incoherent = True              neg : pos    =     16.7 : 1.0
                   unfunny = True              neg : pos    =     16.6 : 1.0
                      yawn = True              neg : pos    =     14.9 : 1.0
                    mathau = True              pos : neg    =     14.5 : 1.0
                   stinker = True              neg : pos    =     13.2 : 1.0
                    drivel = True              neg : pos    =     12.2 : 1.0
                    gundam = True              pos : neg    =     11.8 : 1.0
                 atrocious = True              neg : pos    =     11.7 : 1.0
                  conserve = True              neg : pos    =     11.5 : 1.0
                 masterful = True              pos : neg    =     11.4 : 1.0
                  superbly = True              pos : neg    =     11.4 : 1.0
```

Figure 4.4.3: Accuracy of classifiers and Most Informative Features using the Large Movie Review dataset, 5000 features and all pre-processing steps

With the highest accuracy of an individual classifier reaching 87.24%, the developer notes that the results are comparable to those achieved in the same field. The Vote Classifier, combining the trained classifiers, was able to reach an accuracy of 84%; this success was due to the efficient combination of pre-processing techniques, a high number of features and a comprehensive dataset.

5. Conclusion

Given the limited time frame given to develop the tool, and the little to no experience with the Python programming language and the field of Natural Language Processing, the developer believes that this project, and consequently this tool, was met with relative success. The tool gives the user the option of choosing between a multitude of input methods including entering text directly into the console, selecting a document for analysis, or pasting the URL from an Amazon product review page. Although Amazon is the only currently supported website, this list could easily be added to, as is discussed in the Moving Forward section. However, the performance of the actual classifier itself was a success, for the most part classifying text accurately with a high level of reliability; the analysis itself was made more transparent with the inclusion of the identified features in the output. Overall, the developer believes that the target user, be it those wishing to know the overall sentiment of a review for a product, book or film they are interested in, will find this tool adequate.

During the Analysis and Design phase of the project, the developer was tasked with delving into the intimidating and vast field of Natural Language Processing and Machine Learning; given that this was a hitherto unknown area of computing to the developer, and the abundant amount of research material available online, the developer believes that moderate success was achieved in analysing and bringing together the knowledge gained from research to inform the decision making process during the design and implementation phases. During the design phase, the developer struggled with deciding which method of sentiment analysis to ultimately go for; this proved to have its advantages and disadvantages. While more research was done into the different options before a decision was made, it meant pushing back the implementation phase and the first iterations of the tool.

As mentioned before, the developer before the beginning of this project had no experience with the Python programming language, let alone the NLTK library that was to be used alongside it. For this reason, much of the time coming up to and during the Implementation phase of this project was spent getting comfortable with Python, as well as learning the coding techniques that would be needed to make the tool functional. A considerable amount of time was dedicated to online tutorials on pre-processing techniques in NLTK as well as how to use labelled data to train a classifier. This meant that an early prototype of the tool could not be

produced on-schedule but rather had to be pushed back until July. A revised approach was taken to the classifier design, which proved to be a rewarding decision as the vote based classifier's performance was consistently acceptable.

The Testing phase allowed the developer to validate the functional requirements set out during the Analysis phase by appraising the tool's console menu, ensuring the user's experience was error free and functioned as planned. Lastly, the process of evaluating the tool's classifiers allowed the developer to identify which stages of the pre-processing pipeline were most efficient to include, as well as the number of features that returned the most accurate classifiers.

As mentioned in the Architectural Design section of this report, the bag-of-words feature model looks at words in isolation with no regards to context, save for the identification of negation phrases. This decision meant that the tool would be limited in that it would not be able to handle more linguistically complicated entries such as sarcastic reviews or contextual words that bolster the sentiment of words that follow. Another decision, that of looking at unigrams i.e. single words rather than bi-grams or n-grams, perhaps further limited the scope of the tool, however the developer notes a pragmatic approach was taken and the very limited timescale meant that realistic goals had to be set. Overall, however, the tool functions as it should, and the classifier predicts labels with an accuracy that compares to those within the field of sentiment analysis.

5.1 Lessons Learned

Given the option to tackle the same problem again, the developer would consider an approach to sentiment analysis that utilises a Deep Learning Model that "builds up a representation of whole sentences base on the sentence structure" and "computes the sentiment based on ow words compose the meaning of longer phrases" (Socher et al. 2013). The developer finds the idea of a predictor model like this fascinating, and although neural networks are an intimidating concept to someone new to the filed of computational linguistics, the success with which it detects sentiment would make for a useful application.

The developer learned that adaptability and strong problem solving is key when developing in an Agile environment; when faced with an error, the developer had to think on his feet and

follow the logic of the code to deduce what was causing the problem. The developer also learned the importance of version control and iterative development; as the project progressed to the later stages, having previous versions as safeguards that were fully functional was a relief.

Major Achievements

The developer was happy with how well the Python programming language was picked up, and how comfortable it felt to use; the Java skills learned by the developer played a crucial role in the approach to programming. The developer was happy at the level of understanding achieved in the field of Natural Language Processing in the end up, and counts it as a success that under a tight deadline, a functional tool was implemented successfully.

Major disappointments

- The developer notes that more in-depth evaluations could have been done on the accuracy of the classifiers in the form of plotting on graphs the accuracy vs. the number of features used for example.

- The lack of any kind of GUI, while not the focus of the project, does leave the developer feeling that more could have been done to improve the user's experience as well as the intuitiveness of the tool. It would have also allowed room for more creativity in the design section of this documentation.

- Having to exclude the Twitter Live Sentiment Stream for the final version of the tool has to be looked upon as a failure when evaluating against the functional requirements set out in the Analysis section. This was however necessary due to a problem identified during the evaluation stage.

- Evaluations into the effect of the pre-processing steps could have been broader, and performed on a wider range of datasets.

5.2 Moving Forward

The developer notes that a plethora of features could be added to the tool to increase its overall functionality. This ranges from the simple - such as adding more supported websites or a variety of supported file types - to features to improve and build upon the existing classifier models.

The addition of a GUI would only serve to benefit a tool like this, and with the growth of cross-platform development, the tool could be implemented both across the web and on mobile devices.

The pre-processing pipeline could have additional functions that further improve the accuracy of the classifier; using WordNet's lexical database, a function could be added that does not just replace repeating characters, but fixes minor spelling mistakes too. As mentioned above, Deep Learning Recursive Neural Networks could be used to give meaning to a full sentence, perhaps even being able to recognise sarcasm, colloquialisms and other such lexical occurrences.

To improve upon the classification process, several classifiers could be trained using datasets from a range of domains e.g. twitter or tech blogs, music or science based articles, and using some method of identifying which of these domains the input text belongs to, the appropriate classifier could be loaded and used.

6. References

Azad, K. (2016). *An Intuitive (and Short) Explanation of Bayes' Theorem – BetterExplained*, [online] Available at: https://betterexplained.com/articles/an-intuitive-and-short-explanation-of-bayes-theorem/ [Accessed 22 Jun. 2016]

Bird, S., Klein, E. and Loper, E., 2009. *Natural language processing with Python*. "O'Reilly Media, Inc." Available at: http://www.nltk.org/book/ [Accessed 20 June 2016]

Buitinck, Lars. (2012) Scikit-learn wrapper in NLTK In *Scikit-learn General* [online] Available at: http://sourceforge.net/p/scikit-learn/mailman/scikit-learn-general/thread/20120114195109.GB23476@phare.normalesup.org/ [Accessed 25 July 2016]

Chen, Y., Fay, S. and Wang, Q., (2003) Marketing implications of online consumer product reviews. *Business Week*, *7150*, pp.1-36. Available at: http://plaza.ufl.edu/faysa/review.pdf [Accessed 18 Jan 2016]

Chua, A. Y. K., & Banerjee, S. (2016) Helpfulness of user-generated reviews as a function of review sentiment, product type and information quality. *Computers in Human Behavior*, *54*, 547–554. Available at: http://doi.org/10.1016/j.chb.2015.08.057 [Accessed 20 Jan 2016]

Cohen, D., Lindvall, M. and Costa, P. (2003) Agile software development. *DACS SOAR Report*, *11*. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.2704&rep=rep1&type=pdf [Accessed 20 Aug 2016]

comScore (2007) *Online Consumer-Generated Reviews Have Significant Impact on Offline Purchase Behavior*. [online] Available at: http://www.comscore.com/Insights/Press-Releases/2007/11/Online-Consumer-Reviews-Impact-Offline-Purchasing-Behavior [Accessed 18 Jan. 2016]

Dropbox. (2014) Does Dropbox keep backups of my files? In *Help Center* [online] Available at: https://www.dropbox.com/help/122?path=photos_and_videos [Accessed 19 Jan 2016]

Domingos, P. and Pazzani, M., 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning*, *29*(2-3), pp.103-130. Available at: http://www.cc.gatech.edu/fac/Charles.Isbell/classes/reading/papers/bayes-opt.pdf [Accessed 24 July 2016]

Duan, W., Gu, B. & Whinston, A. B. (2008). Do online reviews matter? — An empirical investigation of panel data. *Decision Support Systems*, *45*(4), 1007–1016. Available at: http://doi.org/10.1016/j.dss.2008.04.001 [Accessed 18 Jan 2016]

du Toit, J. (2015). *A first step into Machine Learning: Building a Bayes Classifier*. [online] Cloud Academy Blog. Available at: http://cloudacademy.com/blog/naive-bayes-classifier/ [Accessed 23 Jul. 2016].

Fan, T., & Chang, C. (2010). Sentiment-oriented contextual advertising. *Knowledge and Information Systems*, *23*(3), 321–344. Available at: http://doi.org/10.1007/978-3-642-00958-7 [Accessed 19 Jan 2016]

Fasola, J., & Matarić, M. J. (2012). Using socially assistive human-robot interaction to motivate physical exercise for older adults. *Proceedings of the IEEE*, *100*(8), 2512–2526. Available at: http://doi.org/10.1109/JPROC.2012.2200539 [Accessed 20 Jan 2016]

F.lux (2016). *f.lux* [online] Available at: https://justgetflux.com/ [Accessed 8 Aug. 2016].

Giachanou, A. and Crestani, F. (2016). Like It or Not: A Survey of Twitter Sentiment Analysis Methods. *CSUR*, 49(2), pp.1-41.

Grimes, S. (2015) Eleven Things Research Pros Should Know About Sentiment Analysis in *Breakthrough Analysis* [online] Available at: http://breakthroughanalysis.com/2015/07/03/eleven-things-research-pros-should-know-about-sentiment-analysis/ [Accessed 18 Jan 2015]

Grimes, S. (2016) Text, Sentiment Analysis & Social Analytics in the Year Ahead: 10 Trends in *Breakthrough Analysis* [online] Available at: http://breakthroughanalysis.com/2016/01/11/10-text-sentiment-social-analytics-trends-for-2016/ [Accessed 19 Jan 2015]

Hirschberg, J. & Manning, C. D. (2015), Advances in natural language processing. *Science*, *349*(6245), 261–266. Available at: http://science.sciencemag.org/content/349/6245/261.abstract [Accessed 15 Jan 2016]

Horrigan, J. A. (2008) Online Shopping: Internet users like the convenience but worry about the security of their financial information, *Pew Internet & American Life Project* [online] Available at: http://www.pewinternet.org/2008/02/13/online-shopping/ [Accessed 18 Jan 2016]

Kogiopsis (2012) Kogiopsis's Review in *Goodreads Reviews* [online] Available at: http://www.goodreads.com/review/show/146662015?book_show_action=true&from_review_page=1 [Accessed 25 July 2016]

Krispil, Mor. (2013) Install NTLK on 64-bit Windows. In *Mor Krispil's Wordpress* [online] Available at: http://morkrispil.wordpress.com/2013/05/04/install-nltk-on-64-bit-windows/ [Accessed 19 Jan 2016]

Liu, B. (2012). Sentiment Analysis and Opinion Mining. *Synthesis Lectures on Human Language Technologies*, *5*(1), 1–167. http://doi.org/10.2200/S00416ED1V01Y201204HLT016 [Accessed 15 Jan 2016]

Maas, A., Daly, R., Pham, P., Huang, D., Ng, A., Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).* [Accessed 25 July 2016]

Madnani, N. (2013) Getting started with Natural Language Processing with Python Available at: http://desilinguist.org/pdf/crossroads.pdf [Accessed 19 Jan 2016]

Manning, C., Raghavan, P., Schutze, H., (2008) Introduction to Information Retrieval, *Stanford Online Resources* [online] Available at: http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html [Accessed 26 July 2016]

MarketsandMarkets (2015) Natural Language Processing Market Worth $13.4 Billion by 2020 [online] Available at: http://www.marketsandmarkets.com/PressReleases/natural-language-processing-nlp.asp [Accessed 19 Jan 2016]

Manning, C.D., Raghavan, P., Schütze, H. (2009) Introduction to Information Retrieval*, Cambridge University Press* [online] Available at: http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf [Accessed July 24 2016]

Narayanan, V. Arora, I. & Bhatia, A. (2013) Fast and accurate sentiment analysis using an enhanced Naïve Bayes Model. Available at: http://arxiv.org/ftp/arxiv/papers/1305/1305.6143.pdf [Accessed 21 Jan 2016]

Nasukawa, T. Yi, J. (2003). Sentiment Analysis : Capturing Favorability Using Natural Language Processing Definition of Sentiment Expressions. *2nd International Conference on Knowledge Capture*, 70–77. Available at: http://doi.org/10.1145/945645.945658 [Accessed 19 Jan 2016]

Nltk.org (2016). *nltk.classify package — NLTK 3.0 documentation*. [online] Available at: http://www.nltk.org/api/nltk.classify.html [Accessed 20 Aug. 2016]

Pang, B., Lee, L. and Vaithyanathan, S. (2002) Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10* (pp. 79-86). Association for Computational Linguistics. Available at: http://arxiv.org/pdf/cs/0205070.pdf [Accessed 26 July 2016]

Pang, B. and Lee, L., (2004) A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on*

Association for Computational Linguistics (p. 271). Association for Computational Linguistics. Available at: http://arxiv.org/pdf/cs/0409058.pdf [Accessed 27 July 2016]

Pang, B. and Lee, L., (2005) Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 115-124). Association for Computational Linguistics. Available at: http://arxiv.org/pdf/cs/0506075.pdf [Accessed 27 July 2016]

Pang, B. & Lee, L. (2008), "Opinion Mining and Sentiment Analysis", *Foundations and Trends in Information Retrieval: Vol. 2: No. 1–2*, pp 1-135. Available at: http://dx.doi.org/10.1561/1500000011 [Accessed 12 Jan 2016]

Pedregosa et al. (2011), 1.9 Naïve Bayes in *Scikit-Learn Modules Documentation* [online] Available at: http://scikit-learn.org/stable/modules/naive_bayes.html [Accessed 24 July 2016]

Perez-Rosas, V., & Mihalcea, R. (2013). Sentiment Analysis of Online Spoken Reviews, (August), 862–866. Available at: https://web.eecs.umich.edu/~mihalcea/papers/perezrosas.interspeech13.pdf [Accessed 20 Jan 2016]

Perkins, J. (2014) *Python 3 Text Processing with NLTK 3 Cookbook*. Packt Publishing Ltd.

Peters SE, Zhang C, Livny M, Ré C (2014) A Machine Reading System for Assembling Synthetic Paleontological Databases. *PLoS ONE 9*(12): e113523. Available at: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0113523 [Accessed 20 Jan 2016]

Potts, Christopher (2011) *Sentiment Symposium Tutorial: Classifiers* [online] Available at: http://sentiment.christopherpotts.net/classifiers.html [Accessed July 26 2016]

Python Enhancement Proposals (2016) *PEP 0 -- Index of Python Enhancement Proposals (PEPs)* [online] Available at: https://www.python.org/dev/peps/ [Accessed 7 Aug 2016]

Qiu, G., He, X., Zhang, F., Shi, Y., Bu, J., & Chen, C. (2010). DASA: Dissatisfaction-oriented Advertising based on Sentiment Analysis. *Expert Systems with Applications*, *37*(9), 6182–6191. Available at: http://doi.org/10.1016/j.eswa.2010.02.109 [Accessed 19 Jan 2016]

Rambocas, M. and Gama, J., 2013. *Marketing research: The role of sentiment analysis* (No. 489). Universidade do Porto, Faculdade de Economia do Porto. Available at: http://wps.fep.up.pt/wps/wp489.pdf [Accessed 18 Jan 2016]

Rish, I., 2001, August. An empirical study of the naive Bayes classifier. In*IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46). IBM New York. Available at: https://www.researchgate.net/profile/Irina_Rish/publication/228845263_An_Empirical_Study_of_the_naive_Bayes_Classifier/links/00b7d52dc3ccd8d692000000.pdf [Accessed 24 July 2016]

Rodrigues, R. G., das Dores, R. M., Camilo-Junior, C. G., & Rosa, T. C. (2015). SentiHealth-Cancer: A sentiment analysis tool to help detecting mood of patients in online social networks. *International Journal of Medical Informatics*, *85*, 80–95. Available at: http://doi.org/10.1016/j.ijmedinf.2015.09.007 [Accessed 19 Jan 2016]

Siqueira, H. and Barros, F., 2010. A feature extraction process for sentiment analysis of opinions on services. In *Proceedings of International Workshop on Web and Text Intelligence*. Available at: http://www.labic.icmc.usp.br/wti2012/IIIWTI_camera_ready/74769.pdf [Accessed 26 July 2016]

Socher, R., Perelygin, A., Wu, J., Jason, C., Manning, C., Ng, A., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1631–1642. Available at: http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf [Accessed 19 Jan 2016]

Taboada, M. (2016). Sentiment Analysis: An Overview from Linguistics. *Annual Review of Linguistics*, *2*(1), 923254456. Available at: http://doi.org/10.1146/annurev-linguistics-011415-040518 [Accessed 18 Jan 2016]

Williams, L., Bannister, C., Arribas-Ayllon, M., Preece, A., & Spasić, I. (2015). The role of idioms in sentiment analysis. *Expert Systems with Applications*, *42*(21), 7375–7385. Available at: http://doi.org/10.1016/j.eswa.2015.05.039 [Accessed 21 Jan 2016]

Xia, R., Xu, F., Yu, J., Qi, Y., & Cambria, E. (2015). Polarity shift detection, elimination and ensemble: A three-stage model for document-level sentiment analysis. *Information Processing & Management*, *52*(1), 36–45. Available at: http://doi.org/10.1016/j.ipm.2015.04.003 [Accessed 20 Jan 2016]

Yasoob, (2013). *What is Pickle in python ?*. [online] Python Tips. Available at: https://pythontips.com/2013/08/02/what-is-pickle-in-python/ [Accessed 25 Aug. 2016].

Zhang, H., 2004. The optimality of naive Bayes. *AA*, *1*(2), p.3. Available at: http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf [Accessed 24 July 2016]

Appendices

A.1 Test Suite

A.8.1 Test Results

NLTK MOVIE REVIEW DATASET

NUM FEATURES: 100 **PRE PROCESSING STEPS:** ALL

```
naive bayes % accuracy:  67.75
pos precision: 0.6985645933014354
pos recall: 0.6886792452830188
post f-measure: 0.6935866983372923
neg precision: 0.6544502617801047
neg recall: 0.6648936170212766
neg f-measure: 0.6596306068601583

multinomial naive bayes % accuracy:  67.75
pos precision: 0.7031963470319634
pos recall: 0.7264150943396226
post f-measure: 0.7146171693735499
neg precision: 0.6280193236714976
neg recall: 0.6914893617021277
neg f-measure: 0.6582278481012659

bernoulli naive bayes % accuracy:  67.75
pos precision: 0.7031963470319634
pos recall: 0.7264150943396226
post f-measure: 0.7146171693735499
neg precision: 0.6280193236714976
neg recall: 0.6914893617021277
neg f-measure: 0.6582278481012659

logistic regression % accuracy:  66.0
pos precision: 0.6835443037974683
pos recall: 0.7641509433962265
post f-measure: 0.7216035634743876
neg precision: 0.6052631578947368
neg recall: 0.7340425531914894
neg f-measure: 0.6634615384615384

stochastic gradient descent % accuracy:  63.5
pos precision: 0.6148867313915858
pos recall: 0.8962264150943396
post f-measure: 0.7293666026871402
neg precision: 0.6025641025641025
neg recall: 0.75
neg f-measure: 0.6682464454976303

linear svc % accuracy:  66.0
pos precision: 0.6148867313915858
pos recall: 0.8962264150943396
post f-measure: 0.7293666026871402
neg precision: 0.6025641025641025
neg recall: 0.75
neg f-measure: 0.6682464454976303

nusvc % accuracy:  64.25
pos precision: 0.6126984126984127
pos recall: 0.910377358490566
post f-measure: 0.7324478178368121
neg precision: 0.583011583011583
neg recall: 0.8031914893617021
neg f-measure: 0.6756152125279642

vote classifier % accuracy:  67.25
pos precision: 0.6126984126984127
pos recall: 0.910377358490566
post f-measure: 0.7324478178368121
neg precision: 0.583011583011583
neg recall: 0.8031914893617021
neg f-measure: 0.6756152125279642
```

```
Most Informative Features
                 NOT_', = False              pos : neg    =      2.7 : 1.0
                    bad = True               neg : pos    =      2.0 : 1.0
                    ',', = False             neg : pos    =      1.6 : 1.0
                 script = True               neg : pos    =      1.6 : 1.0
                    bad = False              pos : neg    =      1.5 : 1.0
                   '!'], = True              neg : pos    =      1.5 : 1.0
                    '*', = True              neg : pos    =      1.5 : 1.0
                 family = True               pos : neg    =      1.5 : 1.0
                   '?'], = False             pos : neg    =      1.4 : 1.0
                    may = True               pos : neg    =      1.4 : 1.0
                  great = True               pos : neg    =      1.4 : 1.0
                 minute = True               neg : pos    =      1.4 : 1.0
                  young = True               pos : neg    =      1.4 : 1.0
                   life = True               pos : neg    =      1.4 : 1.0
                problem = True               neg : pos    =      1.4 : 1.0
                picture = True               pos : neg    =      1.3 : 1.0
               although = True               pos : neg    =      1.3 : 1.0
                   life = False              neg : pos    =      1.3 : 1.0
                   '?'], = True              neg : pos    =      1.3 : 1.0
                   feel = True               pos : neg    =      1.3 : 1.0
```

The most informative features for the NLTK Movie Review dataset presented a problem as the words were being tokenized as they were being read in, and thus punctuation was already separated from the words, making the replace_contractions function redundant.

NUM FEATURES: 1000 **PRE-PROCESSING STEPS:** ALL

```
naive bayes % accuracy:  73.75
pos precision: 0.7487179487179487
pos recall: 0.7227722772277227
post f-measure: 0.7355163727959698
neg precision: 0.7268292682926829
neg recall: 0.7525252525252525
neg f-measure: 0.739454094292804

multinomial naive bayes % accuracy:  77.75
pos precision: 0.7288888888888889
pos recall: 0.8118811881188119
post f-measure: 0.7681498829039813
neg precision: 0.721030042918455
neg recall: 0.8484848484848485
neg f-measure: 0.7795823665893271

bernoulli naive bayes % accuracy:  73.75
pos precision: 0.7288888888888889
pos recall: 0.8118811881188119
post f-measure: 0.7681498829039813
neg precision: 0.721030042918455
neg recall: 0.8484848484848485
neg f-measure: 0.7795823665893271

logistic regression % accuracy:  77.0
pos precision: 0.6964980544747081
pos recall: 0.8861386138613861
post f-measure: 0.7799564270152505
neg precision: 0.7003891050583657
neg recall: 0.9090909090909091
neg f-measure: 0.7912087912087913

stochastic gradient descent % accuracy:  76.0
pos precision: 0.6791044776119403
pos recall: 0.900990099009901
post f-measure: 0.774468085106383
neg precision: 0.6943396226415094
neg recall: 0.9292929292929293
neg f-measure: 0.7948164146868251

linear svc % accuracy:  74.0
pos precision: 0.6739130434782609
pos recall: 0.9207920792079208
post f-measure: 0.7782426778242678
neg precision: 0.6776556776556777
neg recall: 0.9343434343434344
neg f-measure: 0.7855626326963906

nusvc % accuracy:  79.5
pos precision: 0.6702127659574468
pos recall: 0.9356435643564357
post f-measure: 0.78099173553719
neg precision: 0.6751824817518248
neg recall: 0.9343434343434344
neg f-measure: 0.7838983050847459

vote classifier % accuracy:  78.5
pos precision: 0.6702127659574468
pos recall: 0.9356435643564357
post f-measure: 0.78099173553719
neg precision: 0.6751824817518248
neg recall: 0.9343434343434344
neg f-measure: 0.7838983050847459

Most Informative Features
              ridiculous = True              neg : pos    =      5.3 : 1.0
                  awful = True               neg : pos    =      5.1 : 1.0
                  waste = True               neg : pos    =      4.9 : 1.0
                  worst = True               neg : pos    =      4.3 : 1.0
                 stupid = True               neg : pos    =      4.3 : 1.0
                 subtle = True               pos : neg    =      4.2 : 1.0
              memorable = True               pos : neg    =      3.9 : 1.0
                   mess = True               neg : pos    =      3.7 : 1.0
                   dull = True               neg : pos    =      3.6 : 1.0
                terrible = True               neg : pos    =      3.6 : 1.0
                 boring = True               neg : pos    =      3.5 : 1.0
              perfectly = True               pos : neg    =      3.3 : 1.0
              excellent = True               pos : neg    =      3.1 : 1.0
              wonderful = True               pos : neg    =      2.9 : 1.0
                 period = True               pos : neg    =      2.9 : 1.0
                cameron = True               pos : neg    =      2.8 : 1.0
                   dumb = True               neg : pos    =      2.8 : 1.0
              effective = True               pos : neg    =      2.8 : 1.0
                 belief = True               pos : neg    =      2.7 : 1.0
              realistic = True               pos : neg    =      2.7 : 1.0
```

NUM FEATURES: 5000 **PRE-PROCESSING STEPS:** ALL

```
naive bayes % accuracy:  74.5
pos precision: 0.7941176470588235
pos recall: 0.6683168316831684
post f-measure: 0.7258064516129032
neg precision: 0.7086956521739131
neg recall: 0.8232323232323232
neg f-measure: 0.7616822429906542

multinomial naive bayes % accuracy:  78.0
pos precision: 0.7839195979899497
pos recall: 0.7722772277227723
post f-measure: 0.7780548628428927
neg precision: 0.7012448132780082
neg recall: 0.8535353535353535
neg f-measure: 0.7699316628701594

bernoulli naive bayes % accuracy:  74.75
pos precision: 0.7839195979899497
pos recall: 0.7722772277227723
post f-measure: 0.7780548628428927
neg precision: 0.7012448132780082
neg recall: 0.8535353535353535
neg f-measure: 0.7699316628701594

logistic regression % accuracy:  83.0
pos precision: 0.7735042735042735
pos recall: 0.8960396039603961
post f-measure: 0.8302752293577982
neg precision: 0.696969696969697
neg recall: 0.9292929292929293
neg f-measure: 0.7965367965367964

stochastic gradient descent % accuracy:  78.5
pos precision: 0.7593360995850622
pos recall: 0.905940594059406
post f-measure: 0.8261851015801355
neg precision: 0.6891385767790262
neg recall: 0.9292929292929293
neg f-measure: 0.7913978494623655

linear svc % accuracy:  81.75
pos precision: 0.7489878542510121
pos recall: 0.9158415841584159
post f-measure: 0.8240534521158129
neg precision: 0.6838235294117647
neg recall: 0.9393939393939394
neg f-measure: 0.7914893617021277

nusvc % accuracy:  82.5
pos precision: 0.751004016064257
pos recall: 0.9257425742574258
post f-measure: 0.8292682926829269
neg precision: 0.6838235294117647
neg recall: 0.9393939393939394
neg f-measure: 0.7914893617021277

vote classifier % accuracy:  79.5
pos precision: 0.751004016064257
pos recall: 0.9257425742574258
post f-measure: 0.8292682926829269
neg precision: 0.6838235294117647
neg recall: 0.9393939393939394
neg f-measure: 0.7914893617021277

Most Informative Features
                offbeat = True              pos : neg    =     11.7 : 1.0
              ludicrous = True              neg : pos    =     11.3 : 1.0
              strongest = True              pos : neg    =     11.1 : 1.0
            uninvolving = True              neg : pos    =     10.9 : 1.0
            outstanding = True              pos : neg    =     10.8 : 1.0
                idiotic = True              neg : pos    =     10.5 : 1.0
                 temple = True              pos : neg    =     10.4 : 1.0
                 hudson = True              neg : pos    =      9.0 : 1.0
                 regard = True              pos : neg    =      8.6 : 1.0
               marvelous = True              pos : neg    =      8.6 : 1.0
                  finest = True              pos : neg    =      8.6 : 1.0
                   mulan = True              pos : neg    =      8.4 : 1.0
                 winslet = True              pos : neg    =      8.4 : 1.0
              henstridge = True              neg : pos    =      8.3 : 1.0
               uplifting = True              pos : neg    =      8.2 : 1.0
               insulting = True              neg : pos    =      7.8 : 1.0
            wonderfully = True              pos : neg    =      7.2 : 1.0
                sincere = True              pos : neg    =      7.0 : 1.0
                 soviet = True              pos : neg    =      7.0 : 1.0
                mcgowan = True              neg : pos    =      7.0 : 1.0
```

SHORT REVIEW DATASET

NUM FEATURES: 5000 **PRE-PROCESSING STEPS:** ALL

```
naive bayes % accuracy:  75.29301453352086
pos precision: 0.7606330365974283
pos recall: 0.7296015180265655
post f-measure: 0.7447941888619856
neg precision: 0.7459893048128342
neg recall: 0.7757182576459685
neg f-measure: 0.7605633802816901

multinomial naive bayes % accuracy:  75.48054383497421
pos precision: 0.7455399061032864
pos recall: 0.7533206831119544
post f-measure: 0.7494100991033505
neg precision: 0.7447552447552448
neg recall: 0.7896200185356812
neg f-measure: 0.7665317139001351

bernoulli naive bayes % accuracy:  75.80872011251758
pos precision: 0.7448598130841122
pos recall: 0.7561669829222012
post f-measure: 0.7504708097928439
neg precision: 0.7434782608695653
neg recall: 0.7924003707136237
neg f-measure: 0.7671601615074024

logistic regression % accuracy:  74.87107360525081
pos precision: 0.7161716171617162
pos recall: 0.8235294117647058
post f-measure: 0.7661076787290378
neg precision: 0.715647339158062
neg recall: 0.835032437442076
neg f-measure: 0.7707442258340462

stochastic gradient descent % accuracy:  72.90201593999063
pos precision: 0.695852534562212
pos recall: 0.8595825426944972
post f-measure: 0.769100169779287
neg precision: 0.6935975609756098
neg recall: 0.8433734939759037
neg f-measure: 0.7611877875365956

linear svc % accuracy:  71.77684013127052
pos precision: 0.6827637444279346
pos recall: 0.8719165085388995
post f-measure: 0.7658333333333333
neg precision: 0.6774193548387096
neg recall: 0.8563484708063022
neg f-measure: 0.7564469914040114

nusvc % accuracy:  73.23019221753398
pos precision: 0.6788856304985337
pos recall: 0.8785578747628083
post f-measure: 0.7659222497932175
neg precision: 0.6721902017291066
neg recall: 0.8646895273401297
neg f-measure: 0.7563842723956221

vote classifier % accuracy:  75.10548523206751
pos precision: 0.6788856304985337
pos recall: 0.8785578747628083
post f-measure: 0.7659222497932175
neg precision: 0.6721902017291066
neg recall: 0.8646895273401297
neg f-measure: 0.7563842723956221
```

```
Most Informative Features
                  boring = True              neg : pos    =     25.1 : 1.0
                    dull = True              neg : pos    =     22.3 : 1.0
                provides = True              pos : neg    =     15.6 : 1.0
                   fails = True              neg : pos    =     14.7 : 1.0
                   tired = True              neg : pos    =     14.4 : 1.0
              refreshing = True              pos : neg    =     13.6 : 1.0
                     son = True              pos : neg    =     12.9 : 1.0
                mediocre = True              neg : pos    =     12.4 : 1.0
                   waste = True              neg : pos    =     12.4 : 1.0
                   vivid = True              pos : neg    =     11.6 : 1.0
               wonderful = True              pos : neg    =     11.3 : 1.0
                   loses = True              neg : pos    =     11.1 : 1.0
                 generic = True              neg : pos    =     11.1 : 1.0
               obviously = True              neg : pos    =     11.1 : 1.0
            refreshingly = True              pos : neg    =     10.9 : 1.0
               inventive = True              pos : neg    =     10.9 : 1.0
                   shoot = True              neg : pos    =     10.4 : 1.0
             mesmerizing = True              pos : neg    =     10.3 : 1.0
                 culture = True              pos : neg    =     10.3 : 1.0
                  stupid = True              neg : pos    =     10.3 : 1.0
```

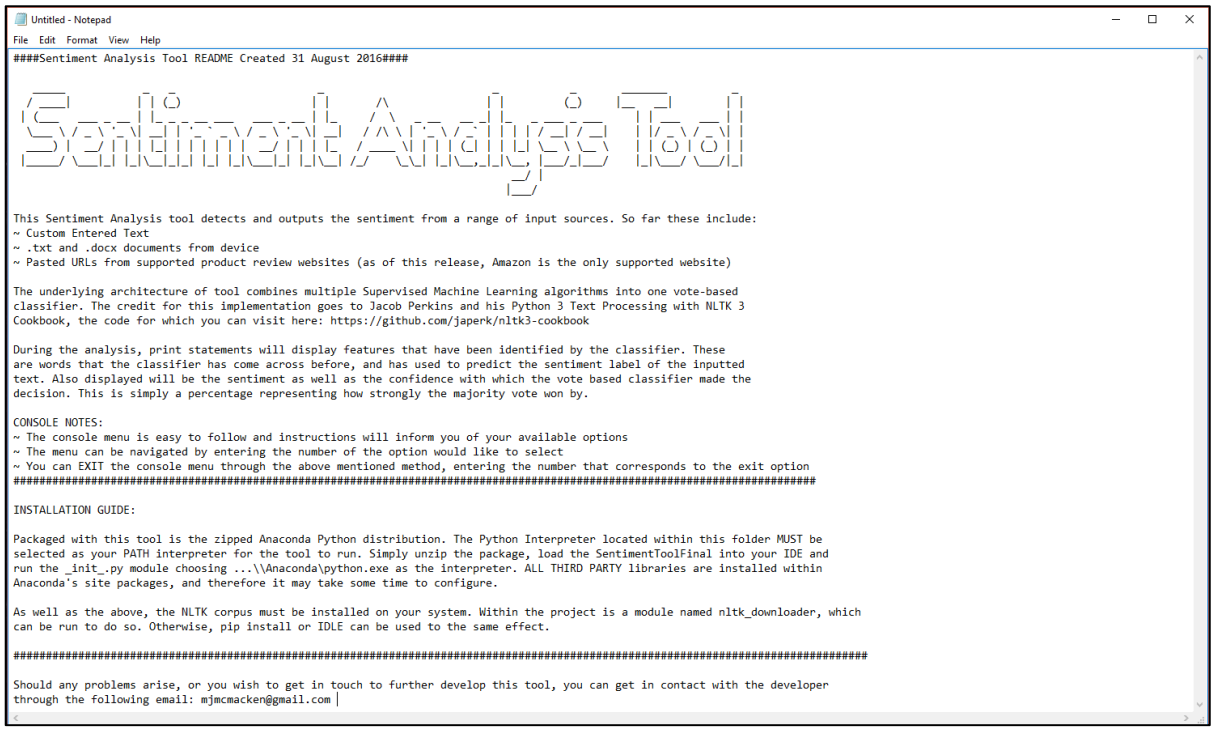
MAAS ACL 2011 LARGE MOVIE REVIEW DATASET

NUM FEATURES: 5000 **PRE-PROCESSING STEPS:** ALL

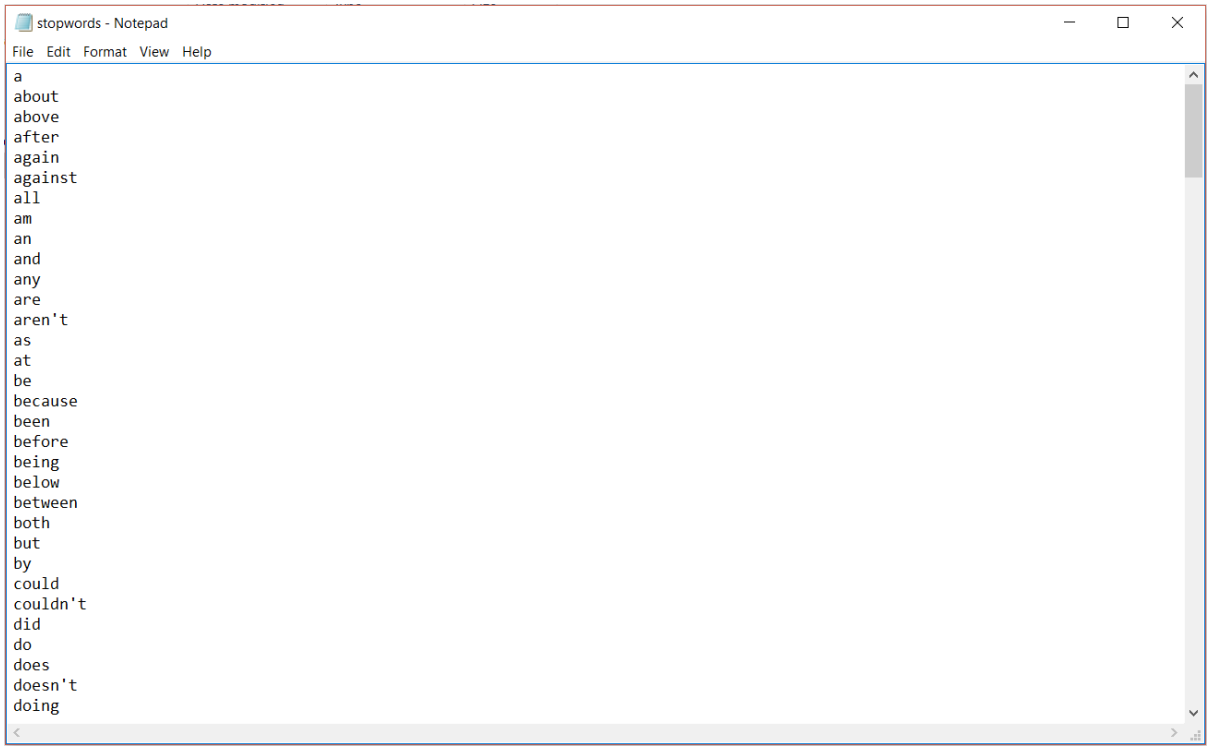
```
Multinomial Classifier % accuracy: 85.86
Bernoulli Classifier % accuracy: 85.08
Logistic Regression Classifier % accuracy: 85.6
SGDC Classifier % accuracy: 84.46000000000001
LinearSVC Classifier % accuracy: 82.54
NuSVC Classifier % accuracy: 87.24

Naive Bayes % accuracy: 85.08
Most Informative Features
              unwatchable = True              neg : pos    =     30.0 : 1.0
                   paulie = True              pos : neg    =     19.1 : 1.0
                 NOT_save = True              neg : pos    =     17.8 : 1.0
                 flawless = True              pos : neg    =     16.8 : 1.0
               incoherent = True              neg : pos    =     16.7 : 1.0
                  unfunny = True              neg : pos    =     16.6 : 1.0
                     yawn = True              neg : pos    =     14.9 : 1.0
                   mathau = True              pos : neg    =     14.5 : 1.0
                  stinker = True              neg : pos    =     13.2 : 1.0
                   drivel = True              neg : pos    =     12.2 : 1.0
                   gundam = True              pos : neg    =     11.8 : 1.0
                atrocious = True              neg : pos    =     11.7 : 1.0
                 conserve = True              neg : pos    =     11.5 : 1.0
                masterful = True              pos : neg    =     11.4 : 1.0
                 superbly = True              pos : neg    =     11.4 : 1.0
```

A.2 Screenshots

A.2.1 Readme.txt



A.2.2 Stopwords.txt



A.2.3 Sentiment Debug Log

sentiment_tool_debug_log - Notepad

File Edit Format View Help

```
DEBUG:root:HOME SCREEN DISPLAYING
DEBUG:root:USERNAME ENTERED: Michael

DEBUG:root:****MAIN MENU RUNNING****
DEBUG:root:OPTION 1 CHOSEN: USER ENTERS CUSTOM TEXT

DEBUG:root:TEXT FOR ANALYSIS: This movie is great. I loved it and this review is positive.

INFO:root:
----PRE-PROCESSING INITIALISED----
DEBUG:root:***TEXT NORMALIZED, WHITE SPACED REMOVED, LOWER CASE FORCED***
DEBUG:root:this movie is great. i loved it and this review is positive.
DEBUG:root:***CONTRACTIONS REPLACED***
DEBUG:root:this movie is great. i loved it and this review is positive.
DEBUG:root:***SPLIT INTO TOKENS***
DEBUG:root:['this', 'movie', 'is', 'great', '.', 'i', 'loved', 'it', 'and', 'this', 'review', 'is', 'positive', '.']
DEBUG:root:***SPELLING CORRECTED***
DEBUG:root:['this', 'movie', 'is', 'great', '.', 'i', 'loved', 'it', 'and', 'this', 'review', 'is', 'positive', '.']
DEBUG:root:***TOKENS LEMMATIZED***
DEBUG:root:['great', '.', 'loved', 'review', 'positive', '.']
DEBUG:root:***STOPWORDS REMOVED***
DEBUG:root:['this', 'movie', 'is', 'great', '.', 'i', 'loved', 'it', 'and', 'this', 'review', 'is', 'positive', '.']
DEBUG:root:***NEGATIONS REPLACED***
DEBUG:root:['great', '.', 'loved', 'review', 'positive', '.']
DEBUG:root:***LENGTH FILTERED***
DEBUG:root:['great', 'loved', 'review', 'positive']
DEBUG:root:****MAIN MENU RUNNING****
DEBUG:root:OPTION 1 CHOSEN: USER ENTERS CUSTOM TEXT

DEBUG:root:TEXT FOR ANALYSIS: I absolutely adore this movie. It's great.

INFO:root:
```

A.2.4 Short Reviews

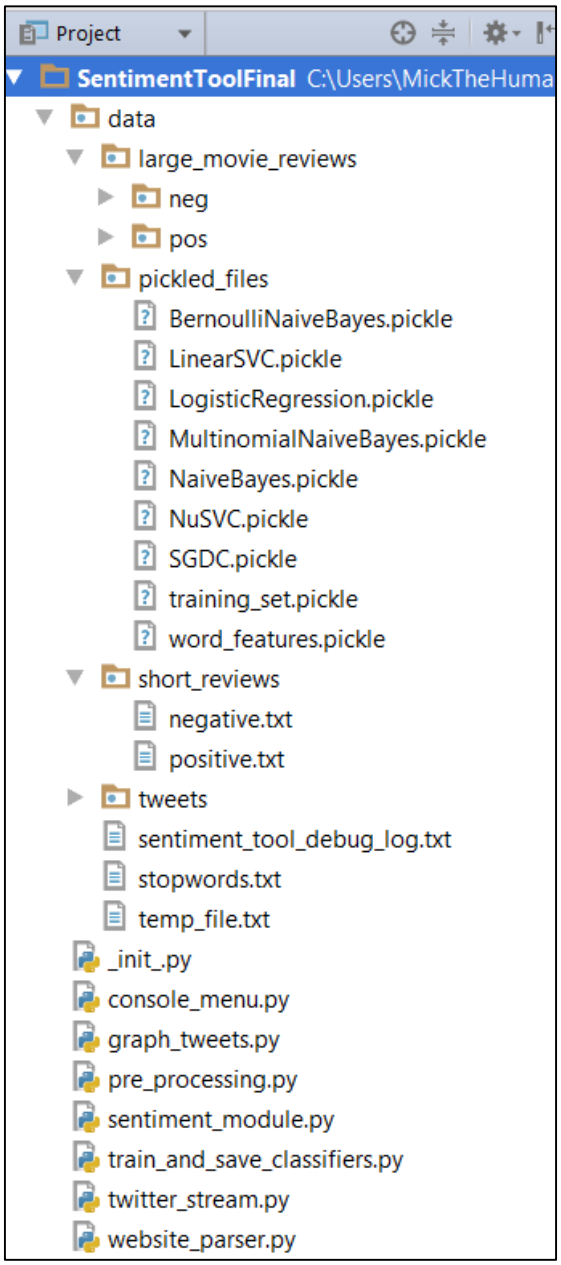
positive - Notepad

File Edit Format View Help

```
the rock is destined to be the 21st century's new " conan " and that he's going to make a splash even greater than arnold schwa
clever you want to hate it . but he somehow pulls it off . take care of my cat offers a refreshingly different slice of asian c
nt film , lovely & amazing involves us because it is so incisive , so bleakly amusing about how we go about our lives . a distı
terful film from a master filmmaker , unique in its deceptive grimness , compelling in its fatalist worldview . light , cute ar
ndish that they border on being cartoonlike . a heavy reliance on cgi technology is beginning to creep into the series . newtor
e glimpse into a culture most of us don't know . tender yet lacerating and darkly funny fable . may be spoofing an easy target
nger/composer bryan adams contributes a slew of songs — a few potential hits , a few more simply intrusive to the story — but t
intense . the invincible werner herzog is alive and well and living in lamorton is a great actress portraying a complex charact
y stylized , is a triumph for its maverick director . at heart the movie is a deftly wrought suspense yarn whose richer shading
fascinating , albeit depressing view of iranian rural life close to the iraqi border . an imaginative comedy/thriller . a few a
raft and uses damon's ability to be focused and sincere . the tenderness of the piece is still intact . katz uses archival foot
ers of a dormant national grief and curiosity that has calcified into chronic cynicism and fear .  . . . a roller-coaster ride
e rules of its own self-contained universe . 4 friends , 2 couples , 2000 miles , and all the pabst blue ribbon beer they can c
his love story . in scope , ambition and accomplishment , children of the century . . . takes kurys' career to a whole new leve
desire to know the 'truth' about this man , while deconstructing the very format of the biography in a manner that derrida woul
t progress . in fessenden's horror trilogy , this theme has proved important to him and is especially so in the finale . it's r
t been dulled by slasher films and gorefests , if you're a connoisseur of psychological horror , this is your ticket . it's a r
of life . waydowntown is by no means a perfect film , but its boasts a huge charm factor and smacks of originality . tim allen
r non-jew husband , feels funny and true . sweet home alabama " is what it is — a nice , harmless date film . . . the year's ha
ck that is a visual tour-de-force and a story that is unlike any you will likely see anywhere else . there are as many misses a
s trials and tribulations . . . the seaside splendor and shallow , beautiful people are nice to look at while you wait for the
ing huppert , a great actress tearing into a landmark role , is riveting . a cop story that understands the medium amazingly we
ocumentary , in which children on both sides of the ever-escalating conflict have their say away from watchful parental eyes ,
d best of all , it lightens your wallet without leaving a sting . it is interesting and fun to see goodall and her chimpanzees
ltzy and thoroughly enjoyable true story . a thoughtful look at a painful incident that made headlines in 1995 . you walk out c
on adolescence feels painfully true . moore's performance impresses almost as much as her work with haynes in 1995's safe . vis
. polished korean political-action film is just as good -- and bad -- as hollywood action epics . is this progress ? elling , p
uy the stuff about barris being a cia hit man . the kooky yet shadowy vision clooney sustains throughout is daring , inventive
is somewhat melodramatic , its ribald humor and touching nostalgia are sure to please anyone in search of a jules and jim for t
detail and nuance and one that speaks volumes about the ability of the human spirit to find solace in events that could easily
accuracy of ms . vardalos' memories and insights . has it ever been possible to say that williams has truly inhabited a charact
t it's hard to know what to praise first . parker holds true to wilde's own vision of a pure comedy with absolutely no meaning
```

B. Source code

Project Structure



The Pycharm IDE allows the developer to view and browse the project's structure with ease. As shown above the code is structured in a logical manner, with clear module names indicating the purpose of individual modules. Additional data is organised into appropriate folders, for example pickled files are all contained within the same folder with the parent directory being the data folder.

B.1 pre_processing.py

```
import logging
from nltk.tokenize import WordPunctTokenizer
from nltk import WordNetLemmatizer
from nltk.corpus import wordnet

from tkinter import *
from tkinter import filedialog
import docx2txt

# sets up logger and specifies log file
logging.basicConfig(filename="data/sentiment_tool_debug_log.txt",filemode='w',level=logging.DEBUG)

'''
To increase the classifier's overall performance, pre-processing will be perfomed on both the training data and any
text input by the user.
'''


def execute_pre_processing(raw_text, repl_contractions=True, repl_misspelled=True,
                           lemmatize=True, stop_filter=True,
                           length_filter=True, replace_negation=True):

    # local variable
    tokenizer = WordPunctTokenizer()    # initialises WordPunctTokenizer object for tokenizing text

    '''
    Based on Jacob Perkins original implementation in NLTK 3 Cookbook p.34
    This function replaces contractions with their expanded forms based on regular expression patterns. For example,
    "can't" is replaced with "cannot" and "would've" is replaced by "would have". Originally a class, but has been
    adapted into a function. (\w+)\'ll recognises preceding groups of letters and only replaces the 'll e.g. "they'll"
    becomes "they will". This makes the text cleaner for tokenization and identifying negation.
    '''
    def replace_contractions(text, repl_contractions):

        replacement_patterns = [
            (r'(\w+)\\\'', '\g<1>\''),
            (r'won\'t', 'will not'),  # won't --> will not
            (r'can\'t', 'cannot'),  # can't --> cannot
            (r'i\'m', 'i am'),  # I'm --> I am
            (r'isn\'t', 'is not'),  # isn't --> is not
            (r'ain\'t', 'is not'),  # ain't --> is not
            (r'(\w+)\'ll', '\g<1> will'),  # cases of 'll become will e.g. They'll --> They will
```

```
            (r'(\w+)n\'t', '\g<1> not'),  # n't --> not
            (r'(\w+)\'ve', '\g<1> have'),  # 've --> have
            (r'(\w+)\'s', '\g<1> is'),  # 's --> is
            (r'(\w+)\'re', '\g<1> are'),  # 're --> are
            (r'(\w+)\'d', '\g<1> would')  # 'd --> would
        ]

        patterns = [(re.compile(regex), repl) for (regex, repl) in replacement_patterns]

        if repl_contractions:
            for (pattern, repl) in patterns:
                text = re.sub(pattern, repl, text)
            # END for
            return text
        # END if

        else:
            return text  # returns original text unchanged
        # END else
    # END replace_contractions

    '''
    The replace_misspelled function is again based on Jacob Perkins implementation in NLTK 3 Cookbook p.37
    Using regular expressions, repeating characters are removed from words, for example "Loooove" is replaced with "love".
    The word is checked first against WordNet to ensure characters are not removed needlessly.

    A second nested function was planned with hopes to utilise the Enchant library to fix minor spelling errors, however
    problems were faced when trying to install the necessary packages.
    '''
    def replace_misspelled(tokens, repl_misspelled):

        def remove_repeating_characters(word):

            repeat_regexp = re.compile(r'(\w*)(\w)\2(\w*)')
            repl = r'\1\2\3'

            if wordnet.synsets(word):  # checks to see if word exists using WordNet
                return word  # if it does, the word is returned
            # END if

            repl_word = repeat_regexp.sub(repl, word)

            if repl_word != word:
                return remove_repeating_characters(repl_word)  # function calls itself until appropriate word returned
```

```
            # END if

            else:
                return repl_word
            # END else
        # END NESTED remove_repeating_characters

        replaced_tokens = []  # empty list to hold the new tokens
        if repl_misspelled:
            for token in tokens:
                clean_token = remove_repeating_characters(token)  # each token is checked for repeating characters
                replaced_tokens.append(clean_token)  # which are then added to new list one by one
            # END for

            return replaced_tokens
        # END if

        else:
            return tokens  # returns original text unchanged
        # END else
    # END replace_misspelled

    '''
    The lemmatize_words function returns the lemma - the base form - of a word if it is found through WordNet. This
    was chosen over the more crude process of Stemming despite being slightly more resource-heavy.
    '''
    def lemmatize_words(tokens, lemmatize):

        lemmatizer = WordNetLemmatizer()  # initialises instance of WordNetLemmatizer object

        if lemmatize:
            lemmatized_tokens = [lemmatizer.lemmatize(w) for w in tokens]  # iterates through words and looks up lemma
            return lemmatized_tokens                                        # through WordNet
        # END if

        else:
            return tokens
        # END else
    # END lemmatize_tokens

    '''
    Stopwords are words that do not inform the sentiment of a text, and are thus removed to allow for more efficient
    feature extraction and negation identification.
    '''
```

```
def remove_stopwords(tokens, stop_filter):

    # custom list of stop words from: http://www.ranks.nl/stopwords
    stopwords = open("data/stopwords.txt", "r").read().splitlines()

    if stop_filter:
        # iterates through each word and only adds those that are NOT in the list of stopwords
        filtered_tokens = [w for w in tokens if w not in stopwords]
        return filtered_tokens
    # END if

    else:
        return tokens  # returns tokens unchanged
    # END else
# END remove_stopwords

'''
Words less than 2 characters long will not be informative, and are therefore filtered from the tokens
'''
def filter_by_length(tokens, length_filter):

    if length_filter:
        filtered_tokens = [w for w in tokens if len(w) > 2] # any words less than 2 characters will be filtered
        return filtered_tokens
    # END if

    else:
        return tokens   # returns tokens unchanged
    # END else
# END filter_by_length

def negation_filter(tokens, replace_negation):

    def replace_negations(sent):

        def negation_check(word):

            # because of replace_contractions function, it is easier to identify negation
            # as most will be normalised to 'not'
            negation_words = ['not', 'no', 'nothing', 'never', 'hardly', 'barely', 't', 'canot', 'isnt', 'wasnt',
                              'cant', 'wont']
            if word in negation_words:
                return True
            # END if
```

```
    else:
        return False
    # END else
# END negation_check

'''
The replace function attempts to replace words following negation words with unambigious antonyms.
'''

def replace(word, pos=None):

    antonyms = set()

    for syn in wordnet.synsets(word, pos=pos):
        for lemma in syn.lemmas():
            for antonym in lemma.antonyms():
                antonyms.add(antonym.name())
            # END for
        # END for
    # END for

    if len(antonyms) == 1:  # if only 1 antonym is found, it is an unambiguous replacement
        unambig_antonym = antonyms.pop()
        return unambig_antonym
    # END if
    else:
        return "NOT_" + word   # appends prefix to indicate the word's negativity
    # END else
# END replace

# CONTINUE replace_negations
i, l = 0, len(sent)
words = []

while i < l:
    word = sent[i]
    if negation_check(word) == True and i+1 < l:
        ant = replace(sent[i+1])
        words.append(ant)
        i += 2
        continue    # continues the next iteration of the loop

    words.append(word)
    i += 1
```

```
            # END while

            return words
        # END replace_negations

        # CONTINUE negation_filter
        if replace_negation:
            filtered_tokens = replace_negations(tokens) # processes text through above function calls
            return filtered_tokens
        # END if
        else:
            return tokens   #returns tokens unchanged
        # END else
    # END negation_filter

    logging.info("\n----PRE-PROCESSING INITIALISED----")

    # SEQUENTIAL calls to the above nested functions makes up the rest of the execute_pre_processing function
    string_text = str(raw_text)
    normalized_text = string_text.strip().lower()

    # contractions are replaced before the text is broken into tokens
    corrected_contractions = replace_contractions(normalized_text, repl_contractions)

    # splits text into its component tokens
    tokens = tokenizer.tokenize(corrected_contractions)

    # repeating characters are removed and minor spelling fixed
    spelling_corrected = replace_misspelled(tokens, repl_misspelled)

    # words are lemmatized where appropriate
    lemmatized_tokens = lemmatize_words(spelling_corrected, lemmatize)

    # stopwords are removed
    stopword_filtered_tokens = remove_stopwords(lemmatized_tokens, stop_filter)

    # checked for negation and replaced where necessary
    negated_tokens = negation_filter(stopword_filtered_tokens, replace_negation)

    # words less than 2 characters removed
    tokens_length_filtered = filter_by_length(negated_tokens, length_filter)

    # log the pre-processing stage
    pre_process_log(normalized_text, corrected_contractions, tokens, spelling_corrected, lemmatized_tokens,
```

```
                    stopword_filtered_tokens, negated_tokens, tokens_length_filtered)

    return tokens_length_filtered

# END execute_pre_processing


'''
The pre_process_log function serves to consolidate the debugging of the pre_processing functions as they execute. This
way it is clear and easier to follow.
'''
def pre_process_log(a,b,c,d,e,f,g,h):

    normalized_text = a
    replace_contractions = b
    tokens = c
    spelling_corrected = d
    stop_filtered = e
    lemmatized_tokens = f
    negated_tokens = g
    length_filtered = h

    logging.debug("***TEXT NORMALIZED, WHITE SPACED REMOVED, LOWER CASE FORCED***")
    logging.debug(normalized_text)

    logging.debug("***CONTRACTIONS REPLACED***")
    logging.debug(replace_contractions)

    logging.debug("***SPLIT INTO TOKENS***")
    logging.debug(tokens)

    logging.debug("***SPELLING CORRECTED***")
    logging.debug(spelling_corrected)

    logging.debug("***TOKENS LEMMATIZED***")
    logging.debug(lemmatized_tokens)

    logging.debug("***STOPWORDS REMOVED***")
    logging.debug(stop_filtered)

    logging.debug("***NEGATIONS REPLACED***")
    logging.debug(negated_tokens)

    logging.debug("***LENGTH FILTERED***")
```

```
    logging.debug(length_filtered)
# END pre_process_log


'''
If the user chooses to select a document for sentiment analysis from their computer, this function is called. It uses
the tkinter module to read in either a simple text file or a docx file using the module docx2txt
'''
def select_and_open_document():
    root_window = Tk()

    root_window.attributes("-topmost", True)
    file_direct = filedialog.askdirectory(parent=root_window, initialdir="/",
                                          title="Select a directory")
    root_window.withdraw()

    if file_direct != "":

        file_name = filedialog.askopenfilename(filetypes=[("Text files", "*.txt;*.docx")],
                                               initialdir=file_direct,
                                               title="Select a file")

        new_file = open("data/temp_file.txt", "w")
        if file_name:
            if str(file_name).endswith(".docx"):

                text = docx2txt.process(file_name)
                new_file.write(text)
                new_file.close()

                review_text = open("data/temp_file.txt", "r").read()
            # END if
            else:
                review_text = open(file_name, "r").read()
            # END else
        # END if
        elif not file_name:
            review_text = "none"
        # END elif
    else:
        review_text = "none"
    # END else
    return review_text
# END select_and_open_document
```

B.2 train_and_save_classifiers.py

```
import nltk
from nltk.corpus import movie_reviews   # 1000 positive & 1000 negative reviews
import random                           # used to shuffle the data as it is all ordered
import pickle                           # used to save the trained classifiers
import os.path
import logging


import pre_processing as pre    # used to perform pre-processing on training data

from nltk.classify.scikitlearn import  SklearnClassifier    # wrapper for the scikit-learn classifiers

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

from nltk.metrics import precision, recall, f_measure

'''
This module is intended as a "once-and-done" execution. It trains multiple classifiers from the NLTK
and scikit-learn libraries, before using pickle to save them.
'''


# Global variables
NUM_FEATURES = 1000
TRAINING_RATIO = .8
LARGE_MOVIESET = "large movie set"
SHORT_REVIEWS = "short reviews"
NLTK_MOVIE_REVIEWS = "nltk_movie_reviews"


def create_tuples(chosen_dataset):

    if chosen_dataset.lower() == "short reviews":
        short_pos = open("data/short_reviews/positive.txt", "r").read()
        short_neg = open("data/short_reviews/negative.txt", "r").read()

        review_tuples = []

        for review in short_pos.split('\n'):
            review_tuples.append((review, "pos"))
```

```
        # END for

        for review in short_neg.split('\n'):
            review_tuples.append((review, "neg"))
        # END for
    # END if

    elif chosen_dataset.lower() == "large movie set":

        review_tuples = []

        pos_path = os.getcwd() + r'\data\large_movie_reviews\pos/'
        neg_path = os.getcwd() + r"\data\large_movie_reviews\neg/"

        pos_list = os.listdir(pos_path) # returns a list of names of pos reviews
        neg_list = os.listdir(neg_path) # returns a list of names of neg reviews


        # each text file is read in and appended to its appropriate list
        for review in pos_list:
            review_text = open((pos_path + review),'r', encoding="utf-8").read()
            review_tuples.append((review_text.encode("utf-8"), "pos"))
        # END for

        for review in neg_list:
            review_text = open((neg_path + review),'r', encoding="utf-8").read()
            review_tuples.append((review_text.encode('utf-8'), "neg"))
        # END for

    #END elif

    else: # to ensure that a dataset is at least chosen, nltk's movie review dataset will be the fallback option
        review_tuples = [(list(movie_reviews.paras(fileid)), category)
                         for category in movie_reviews.categories()
                         for fileid in movie_reviews.fileids(category)]
    # END else

    logging.debug(review_tuples[0])

    random.shuffle(review_tuples) # the tuples are shuffled to reduce the chances of over-fitting
    return review_tuples
# END create_tuples
```

```
def find_features(document):

    words = set(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    # END for

    return features
# END find_features

review_tuples = create_tuples(NLTK_MOVIE_REVIEWS)
review_corpus = []
all_words = []

for review, tag in review_tuples:
    filtered_review = pre.execute_pre_processing(review)
    review_corpus.append((filtered_review, tag))
    all_words.extend(filtered_review)
# END for

all_words_freq = nltk.FreqDist(all_words)   # converts all words into an NLTK frequency distribution

word_features = []
for w in all_words_freq.most_common(NUM_FEATURES):
    word_features.append(w[0])
# END for

save_word_features = open("data/pickled_files/word_features.pickle", "wb")
pickle.dump(word_features, save_word_features)
save_word_features.close()

featuresets = [(find_features(rev), category) for rev, category in review_corpus]
cut_off_point = int(len(featuresets) * TRAINING_RATIO)
training_set = featuresets[:cut_off_point]
testing_set = featuresets[cut_off_point:]

# pickle the testing set
save_testing_set = open("data/pickled_files/training_set.pickle", "wb")
pickle.dump(testing_set, save_testing_set)
save_testing_set.close()


# **********************
```

```
# Naive Bayes Classifier
# **********************
NB_classifier = nltk.NaiveBayesClassifier.train(training_set)
print("Naive Bayes % accuracy:", (nltk.classify.accuracy(NB_classifier, testing_set))*100)
NB_classifier.show_most_informative_features(100)

save_classifier = open("data/pickled_files/NaiveBayes.pickle", "wb")
pickle.dump(NB_classifier, save_classifier)
save_classifier.close()
# ************************************************


# ************************
# Multinomial Naive Bayes
# ************************
MultinomialNB_classifier = SklearnClassifier(MultinomialNB())
MultinomialNB_classifier.train(training_set)
print("Multinomial Classifier % accuracy:", (nltk.classify.accuracy(MultinomialNB_classifier, testing_set))*100)

save_MN_classifier = open("data/pickled_files/MultinomialNaiveBayes.pickle", "wb")
pickle.dump(MultinomialNB_classifier, save_MN_classifier)
save_MN_classifier.close()
# ************************************************


# **********************
# Bernoulli Naive Bayes
# **********************
BernoulliNB_classifier = SklearnClassifier(BernoulliNB())
BernoulliNB_classifier.train(training_set)
print("Bernoulli Classifier % accuracy:", (nltk.classify.accuracy(BernoulliNB_classifier, testing_set))*100)

save_BNB_classifier = open("data/pickled_files/BernoulliNaiveBayes.pickle", "wb")
pickle.dump(BernoulliNB_classifier, save_BNB_classifier)
save_BNB_classifier.close()
# ************************************************


# ********************
# Logistic Regression
# ********************
LogisticReg_classifier = SklearnClassifier(LogisticRegression())
LogisticReg_classifier.train(training_set)
print("Logistic Regression Classifier % accuracy:", (nltk.classify.accuracy(LogisticReg_classifier, testing_set))*100)
```

```
save_LR_classifier = open("data/pickled_files/LogisticRegression.pickle", "wb")
pickle.dump(LogisticReg_classifier, save_LR_classifier)
save_LR_classifier.close()
# ************************************************************

# ***************************
# Stochastic Gradient Descent
# ***************************
SGDC_classifier = SklearnClassifier(SGDClassifier())
SGDC_classifier.train(training_set)
print("SGDC Classifier % accuracy:", (nltk.classify.accuracy(SGDC_classifier, testing_set))*100)

save_SGDC_classifier = open("data/pickled_files/SGDC.pickle", "wb")
pickle.dump(SGDC_classifier, save_SGDC_classifier)
save_SGDC_classifier.close()
# ************************************************************

# **********
# Linear SVC
# **********
LinearSVC_classifier = SklearnClassifier(LinearSVC())
LinearSVC_classifier.train(training_set)
print("LinearSVC Classifier % accuracy:", (nltk.classify.accuracy(LinearSVC_classifier, testing_set))*100)

save_LinearSVC_classifier = open("data/pickled_files/LinearSVC.pickle", "wb")
pickle.dump(LinearSVC_classifier, save_LinearSVC_classifier)
save_LinearSVC_classifier.close()
# ************************************************************


# *****
# NuSVC
# *****
NuSVC_classifier = SklearnClassifier(NuSVC())
NuSVC_classifier.train(training_set)
print("NuSVC Classifier % accuracy:", (nltk.classify.accuracy(NuSVC_classifier, testing_set))*100)

save_NuSVC_classifier = open("data/pickled_files/NuSVC.pickle", "wb")
pickle.dump(NuSVC_classifier, save_NuSVC_classifier)
save_NuSVC_classifier.close()
# ************************************************************
```

B.3 website_parser.py

```
import logging, re
from bs4 import BeautifulSoup
from urllib.request import urlopen
from nltk.tokenize import WordPunctTokenizer
import urllib

'''
This module uses BeautifulSoup to scrape the text from accepted website URLs for sentiment analysis. If the user
enters a valid URL, then the text is parsed based on the specific website entered. This module has the potential
to be scaled up to include many more websites.
'''

AMAZON = "amazon"
PITCHFORK = "pitchfork" # attemping to connect to pitchfork gives a 403 forbidden error

'''
Checks to see if user's URL is valid using regular expressions. Credit for regex goes to John Gruber:
http://daringfireball.net/2010/07/improved_regex_for_matching_urls
'''
def check_URL(url_input):
    if re.match('https?://(?:www)?(?:[\w-]{2,255}(?:\.\w{2,6}){1,2})(?:/[\w&%?#-]{1,300})?', url_input):

        # because some requests might not be available because of a high load, a try except is used here
        try:
            url_contents = urlopen(url_input).read()
            return str(url_contents)
        except urllib.error.HTTPError as e:
            print(e.code)
            print(e.read())
        # END try/except
    # END if

    else:
        return "invalid"
    # END else
# END check_URL


'''
This function identifies the website for the URL's contents
'''
def identify_website(url_contents):
```

```
    page_content = url_contents
    tokenizer = WordPunctTokenizer()

    tokenized_url = tokenizer.tokenize(page_content) # tokenizing the text makes it easier to identify the source

    if AMAZON in tokenized_url:
        return page_content, AMAZON
    # END if
    elif PITCHFORK in tokenized_url:
        return page_content, PITCHFORK
    # END elif
    else:
        return "unsupported"
    # END else
#END identify_website


'''
If this function has been called, then the URL has been accpeted and the source identified. The text is then parsed
using specific tags and container names.
'''
def parse_review_text(page_contents, source_website):

    if source_website == AMAZON:

        review_text = ""
        soup = BeautifulSoup("".join(page_contents), "lxml")
        amazon_review = soup.findAll("div","reviewText")

        for reviews in amazon_review:
            temp_review_text = review_text + " " + reviews.text
            review_text = temp_review_text
        # END for
    # END if

    # elif source_website == PITCHFORK:
    #
    #     review_text = ""
    #     soup = BeautifulSoup("".join(page_contents), "lxml")
    #     pitchfork_review = soup.findAll("div","abstract")
    #
    #     for reviews in pitchfork_review:
    #         temp_review_text = review_text + " " + reviews.text
```

```
    #         review_text = temp_review_text
        # END for
    # END elif

    return str(review_text)
# END parse_review_text


'''
This function serves to consolidate the above functions
'''
def run_website_parser(input_url):

    url_contents = check_URL(input_url)

    if url_contents == "invalid":
        parsed_text = "invalid"
    # END if

    else:
        page_contents, url_source = identify_website(url_contents)
        parsed_text = parse_review_text(page_contents, url_source)
    # END else
    return parsed_text
# END run_website_parser
```

B.4 sentiment_module.py

```
import collections
import nltk
import pickle
from nltk.classify import ClassifierI   # Voted classifier inherits methods from the NLTK classifier class
from statistics import mode             # used to choose which class gets the most votes

import pre_processing as pre            # used to pre-process input from user
from nltk.metrics import precision, recall, f_measure


'''
Within this module is the Class for creating a VoteClassifier object. This uses the previously trained and pickled
classifiers to classify input from the user as positive or negative. Because this class inherits from NLTK's classifier
superclass, it can be treated as a normal classifier e.g. the accuracy can be calculated.
'''

# GLOBAL VARIABLES FOR DISPLAYING TEST RESULTS
NAIVE_BAYES = "naive bayes"
MULTINOMIAL_NB = "multinomial naive bayes"
BERNOULLI_NB = "bernoulli naive bayes"
LOGISTIC_REG = "logistic regression"
SGDC = "stochastic gradient descent"
LINEAR_SVC = "linear svc"
NU_SVC = "nusvc"
VOTE_CLASSIFER = "vote classifier"

class VoteClassifier(ClassifierI):

    def __init__(self, *classifiers):   # a list of classifiers is passed through as arguments
        self._classifiers = classifiers

    def classify(self, features):

        votes = []
        for c in self._classifiers:
            v = c.classify(features)    # each classifier classifiers the features, the result is appended
            votes.append(v)             # to a list of votes
        return mode(votes)              # returns which class (positive or negative) got the most votes
    # END classify

    def confidence(self, features):
```

```
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)

        choice_votes = votes.count(mode(votes))      # takes a count of the number of winning votes
        conf = choice_votes / len(votes)             # returns a number between 0 nd 1 indicating confidence
        return conf                                  # not a % at this stage
    # END confidence

    def features_present(self, features):

        important_features = []

        for word, present in features.items():
            if present == True:
                important_features.append(word)
            # END if
        # END for

        # convert list of features to string for print purposes
        string_features = ", ".join(important_features)

        if string_features == "":
            string_features = "No features present"

        return string_features
        # End if
# END classifier

# Load in pickled word features
word_features_file = open("data/pickled_files/word_features.pickle", "rb")
word_features = pickle.load(word_features_file)
word_features_file.close()

def find_features(document):
    words = set(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    # END for
    return features
# END find_features
```

```
# load all of the pickled classifiers
# Naive Bayes
file_loc = open("data/pickled_files/NaiveBayes.pickle", "rb")
naive_bayes_classifier = pickle.load(file_loc)
file_loc.close()

# Multinomial Naive Bayes
file_loc = open("data/pickled_files/MultinomialNaiveBayes.pickle", "rb")
multinomial_NB_classifier = pickle.load(file_loc)
file_loc.close()

#Bernoulli Naive Bayes
file_loc = open("data/pickled_files/BernoulliNaiveBayes.pickle", "rb")
bernoulli_NB_classifier = pickle.load(file_loc)
file_loc.close()

# Logistic Regression
file_loc = open("data/pickled_files/LogisticRegression.pickle", "rb")
logistic_reg_classifier = pickle.load(file_loc)
file_loc.close()

# SGDC
file_loc = open("data/pickled_files/SGDC.pickle", "rb")
SGDC_classifier = pickle.load(file_loc)
file_loc.close()

# Linear SVC
file_loc = open("data/pickled_files/LinearSVC.pickle", "rb")
linear_SVC_classifier = pickle.load(file_loc)
file_loc.close()

# NuSVC
file_loc = open("data/pickled_files/NuSVC.pickle", "rb")
nuSVC_classifier = pickle.load(file_loc)
file_loc.close()

# create a voting classifier using the class defined above using the pickled classifiers
vote_classifier = VoteClassifier(naive_bayes_classifier,
                                 multinomial_NB_classifier,
                                 bernoulli_NB_classifier,
                                 logistic_reg_classifier,
                                 SGDC_classifier,
                                 linear_SVC_classifier,
```

```
                                nuSVC_classifier)

'''
The following function will be called from the console menu and will return print statements detailing the extracted
sentiment of the user's input, the confidence with which the text was classified, and the features recognised by
the classifier.
'''
def extract_sentiment(user_input):

    filtered_text = pre.execute_pre_processing(user_input)                      # filters text thorough pre-processing
    extracted_features = find_features(filtered_text)                           # creates dictionary of found features
    sentiment = vote_classifier.classify(extracted_features)                    # votes counted for overall sentiment
    confidence = vote_classifier.confidence(extracted_features)                 # confidence of vote
    features_identified = vote_classifier.features_present(extracted_features)  # important features used to classify

    # print statements for the console menu
    print("\nTEXT ENTERED: ", user_input)
    print("FILTERED TEXT: ", filtered_text)
    print("FEATURES IDENTIFIED: ", features_identified)

    if sentiment == 'pos':
        print("OVERALL SENTIMENT: POSITIVE")
    # END if
    else:
        print("OVERALL SENTIMENT: NEGATIVE")
    # END else

    print("CONFIDENCE: " + str(int(confidence*100)) + "%")
# END extract_sentiment


'''
The following function iterates through each classifier and displays test results for each one
'''
def run_test_results():

    classifiers = [naive_bayes_classifier, multinomial_NB_classifier, bernoulli_NB_classifier, logistic_reg_classifier,
                   SGDC_classifier, linear_SVC_classifier, nuSVC_classifier, vote_classifier]

    classifier_names = [NAIVE_BAYES, MULTINOMIAL_NB, BERNOULLI_NB, LOGISTIC_REG, SGDC, LINEAR_SVC,
                        NU_SVC, VOTE_CLASSIFER]


    open_testing_set = open("data/pickled_files/training_set.pickle", "rb")
```

```
    testing_set = pickle.load(open_testing_set)
    open_testing_set.close()

    refsets = collections.defaultdict(set)
    testsets = collections.defaultdict(set)

    loop = 0  # control variable

    for classifier in classifiers:

        for i, (feats, label) in enumerate(testing_set):
            refsets[label].add(i)
            observed = classifier.classify(feats)
            testsets[observed].add(i)
        # END for

        print("\n" + classifier_names[loop] + " % accuracy: ", str((nltk.classify.accuracy(classifier, testing_set))*100))
        print("pos precision:", precision(refsets['pos'], testsets['pos']))
        print("pos recall:", recall(refsets['pos'], testsets['pos']))
        print("post f-measure:", f_measure(refsets['pos'], testsets['pos']))
        print("neg precision:", precision(refsets['neg'], testsets['neg']))
        print("neg recall:", recall(refsets['neg'], testsets['neg']))
        print("neg f-measure:", f_measure(refsets['neg'], testsets['neg']))

        loop += 1
    # END for

    naive_bayes_classifier.show_most_informative_features(20)
# END run_test_results

def twitter_analysis(tweet):

    filtered_tweet = pre.execute_pre_processing(tweet)
    features = find_features(filtered_tweet)

    return vote_classifier.classify(features), vote_classifier.confidence(features), vote_classifier.features_present(features)
# END twitter_analysis
```

B.5 console_menu.py

```
import logging
import sys
import sentiment_module as classifier
import pre_processing as pre
import website_parser


class console_menu:

    def __init__(self):

        logging.debug("HOME SCREEN DISPLAYING")
        self.user_name = ""

    def home_screen(self):

        print("""
```

```
        self.user_name = input("\nPlease enter your name: ")
        print("\nWelcome %s!" % self.user_name)
        logging.debug("USERNAME ENTERED: " + self.user_name + "\n")
    # end home_screen

    def main_menu(self):
        logging.debug("****MAIN MENU RUNNING****")

        print("\n****HOME MENU***")

        print("\nPlease select an option:")

        print("1) Enter custom text to be classified")
        print("2) Select a document on this device to classify")
        print("3) Pase the URL of an AMAZON product review")
        # print("4) Perform live sentiment analysis on tweets (requires internet connection)")
        # print("5) Graph live sentiment analysis of tweets")
```

```
print("4) Display accuarcy of trained classifiers")
print("5) How this tool works")
print("6) EXIT SYSTEM")

user_choice = input("\nEnter an option: ")
if (user_choice == "1"):

    logging.debug("OPTION 1 CHOSEN: USER ENTERS CUSTOM TEXT\n")

    input_text = input("Input text to be classified into positive or negative: ")
    logging.debug("TEXT FOR ANALYSIS: " + input_text + "\n")

    classifier.extract_sentiment(input_text) # call to this function returns appropriate print statements

# end if

elif (user_choice == "2"):

    logging.debug("OPTION 2 CHOSEN: USER TO SELECT DOCUMENT FROM DEVICE")

    document_text = pre.select_and_open_document()

    if document_text != "none" or "":
        classifier.extract_sentiment(document_text)
    # END if

    else:
        print("Document does not contain text!")
    # END else

# END elif

elif (user_choice == "3"):

    logging.debug("OPTION 3 CHOSEN: USER ENTERS URL FOR PARSING")

    user_input = input("Enter or paste URL and hit enter: ")

    parsed_text = website_parser.run_website_parser(user_input)

    if parsed_text == "invalid":
        print("\nINVALID URL ENTERED")
    # END if
    elif parsed_text == "unsupported":
```

```
            print("\nThat website is currently not supported, sorry!")
        # END elif
        else:
            classifier.extract_sentiment(parsed_text)
        # END else

# END elif

elif (user_choice == "4"):

    logging.debug("OPTION 4 CHOSEN: DISPLAY ACCURACY OF CLASSIFIERS")
    classifier.run_test_results()

    print("\nPRESS ENTER TO CONTINUE")
    input()
    self.main_menu()

# END elif

elif (user_choice == '5'):

    logging.debug("MENU OPTION 5: DISPLAYING INFORMATION")

    print("\nThis tool uses Natural Language Processing techniques combined with Supervised Machine Learning"
          " to produce trained algorithms capable of classifying your input as either positive\nor negative.")
    print("\nFirst the text is put through the pre-processing stage, making use of NLTK's library of useful"
          "modules. The text is normalized and stripped of uninformative words before important \nfeatures are"
          "identified to suss out the overarching sentiment.")
    print("\nWhere some methods employ just the one classifier algorithm, this tool combines seven trained "
          "predictor models into one vote-based classifier. The advantage here is that a more \nreliable "
          "classifier is created that is able to produce consistently accurate results.")
    print("\nIf you wish to learn more, a good starting place is Steven Bird's NLTK Book: http://www.nltk.org/book/")
    print("\nPRESS ENTER TO GO BACK")

    input()
    self.main_menu()

# END elif

elif (user_choice == "6"):
    logging.debug("OPTION 6 CHOSEN: PROGRAM TERMINATING")
    sys.exit("Application terminated")
# end elif
```

```
        else:
            logging.error("INVALID INPUT ENTERED")
            print("Not a valid input")
        # end else

        self.main_menu()

    #end main_menu

    def execute_menu(self):
        self.home_screen()
        self.main_menu()
    # end execute_menu

def run():
    start_menu = console_menu()
    start_menu.execute_menu()
```

B.6 twitter_stream.py

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import json
import sentiment_module as s

# consumer key, consumer secret, access token, access secret.
ckey = "AWpfXoESvQqOxAgOtahLre357"
csecret = "ENeVJ4Iadhwf0RBWmaWZXa7uz48LFGVmm9XlHOmNraxMInRiJ6"
atoken = "4898261687-2i1PlUTVbtDyiiYdEp1UWEs9EOjdSVPMS9RJ4Ki"
asecret = "IOGTHF10CJHIGfPQ0yHZH0TSH4KTWQmYqkhuVCeUKEPLU"


class listener(StreamListener):
    def on_data(self, data):
        all_data = json.loads(data)

        tweet = all_data["text"]
        encode_tweet = tweet.encode("utf-8")
        sentiment_value, confidence, features_present = s.twitter_analysis(encode_tweet)
        print("\nTWEET: " + tweet)
        print("SENTIMENT: " + sentiment_value)
        print("CONFIDENCE: " + str(confidence))
        print("FOUND FEATURES: " + features_present)

        if confidence*100 >= 80:
            output = open("data/tweets/twitter-out.txt", "a")
            output.write(sentiment_value)
            output.write('\n')
            output.close()

        return True

    def on_error(self, status):
        print(status)


auth = OAuthHandler(ckey, csecret)
auth.set_access_token(atoken, asecret)

twitterStream = Stream(auth, listener())
twitterStream.filter(track=["happy"])
```

B.7 graph_tweets.py

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import style


'''
This module is intended as a companion module to twitter_stream. It was adapted based on a tutorial by Harrisson Kinsley
on www.pythonprogramming.net
'''
style.use("ggplot")

fig = plt.figure()
ax1 = fig.add_subplot(1, 1, 1)


def animate(i):
    pullData = open("tweets/twitter-out.txt", "r").read()
    lines = pullData.split('\n')

    xar = []
    yar = []

    x = 0
    y = 0

    for l in lines[-200:]:
        x += 1
        if "pos" in l:
            y += 1
        elif "neg" in l:
            y -= 1

        xar.append(x)
        yar.append(y)

    ax1.clear()
    ax1.plot(xar, yar)


ani = animation.FuncAnimation(fig, animate, interval=1000)
plt.show()
```