**How to Build a Sentiment Analysis Program**

**Contents**

* **Introduction**
* **Review**
* **Method**
* **Conclusion**
* **Glossary**
* **Appendix**
* **References**

**Introduction**

What is Sentiment Analysis? [1] Sentiment Analysis is the measure of positive and negative language or text. It is a method of gauging written or spoken language to determine if the statement is favourable, unfavourable or neutral and to what extent.[1]

[1] https://www.clarabridge.com/sentiment-analysis/

This essay will be delivered under three subject headlines: Review, Method and Conclusion. Section 1- Review, will explain how and why Sentiment Analysis is used and discuss in depth the impact of Sentiment Analysis. Furthermore, this section will investigate current technological trends in this area and give real-life examples. Finally, the Review will examine present and potential future development in the field of Sentiment Analysis. In Section 2- Method, this document will detail how to build a Python software program to perform Sentiment Analysis and what are the requirements (concepts, approaches, technical stacks). In addition, what are the advantages and disadvantages of Sentiment Analysis platforms? Also, how to present this information to the user and how we verify accurate of results. At the end of this report, Section 3- Conclusion will conclude the report by reiterating the important points of the essay and state the advantages and limitations of Sentiment Analysis and propose any solution.

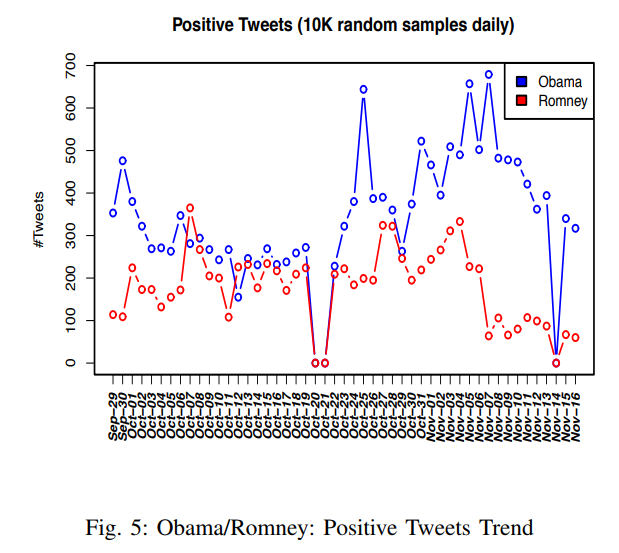
**Section 1- Review**

Sentiment Analysis is used by businesses, companies, groups and even political parties to oversee what the public thinks about their products and ideas. For example, in the 2012 US presidential election Twitter Sentiment Analysis was used to predict the results of the election. [2] According to Twitter, in 2012 Twitter had over 100 million users sending 250 million tweets each day. This is a huge database of information for political parties. Twitter was actively used by 13% of online American adults from May 2011, as the election was beginning. This figure was up from 8% a year previous (Pew Research Center, 2011). This shows that more American voters were becoming more active online during the election, expressing their political opinions. A Sentiment Analysis program was influential in gathering voter’s opinions, using Twitter. [2] [2]<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.383.5491&rep=rep1&type=pdf>

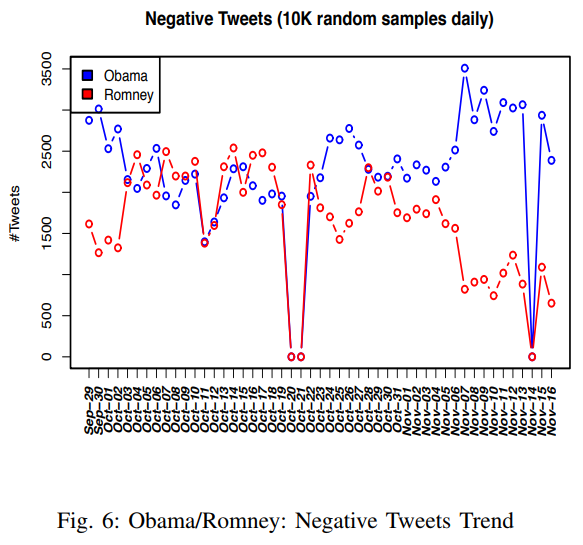
During the 2012 US presidential election Sentiment Analysis was carried out to predict the result of the election, this research was carried out by Cornell University and the paper was submitted in July 2014. [3] The title of the paper was “The Predictive Power of Social Media: On the Predictability of U.S. Presidential Elections using Twitter”. The paper examined the predictability of the US presidential election result using Twitter. In the study, the research team analysed 32 million tweets about the election. To perform the investigation the research team developed an advanced classifier for sentiment analysis to increase accuracy of the analysis of the tweets. The analysis was performed by comparing the Twitter analysis with traditional opinion polls. Furthermore, the researchers used “the Latent Dirichlet Allocation model to extract the underlying topical structure from the selected tweets.” [3] The results of the research showed that they could determine the popularity of the two presidential candidates using Sentiment Analysis. In addition, they could also monitor the popularity of the candidates across the US states using geo-tagged technology. [3]

The graphs below are taken from the “The Predictive Power of Social Media: On the Predictability of U.S. Presidential Elections using Twitter”.

[3] <https://arxiv.org/pdf/1407.0622v1.pdf>



**Figure 1: Obama/Romney: Positive Tweets Trend**



**Figure 2: Obama/Romney: Negative Tweets Trend**

This research shows the importance of social data and the information it can provide. The reason why Sentiment Analysis is so important to businesses and political parties is it can be used to provide information about if the public like or dislike their products or ideas. However, this information is not directly given to the businesses or political parties it involves resources to set-up and run Sentiment Analysis programs. All customer opinions are useful to a business or political party and Sentiment Analysis allows them to quickly know what customers and voters think and gathers vast numbers of customer and voter opinion from a variety of social media platforms. However, for Sentiment Analysis to work it must understand how the customer and voters feels, it is not enough for the business or political party to know what people are writing and talking about. [1]<https://www.clarabridge.com/sentiment-analysis/>

Furthermore, Sentiment Analysis can inform them if there have been any changes in customer opinion about them or their products. In addition, Sentiment Analysis is great for businesses and political parties as it is continually available and giving them current customer opinion.

[4] However, Sentiment Analysis does have weaknesses and is not perfect all the time. It is extremely hard to develop a program to understand all the difficulties and complexities of human language, context and tone. Difficulties include: slang, people misspelling words and sarcasm.

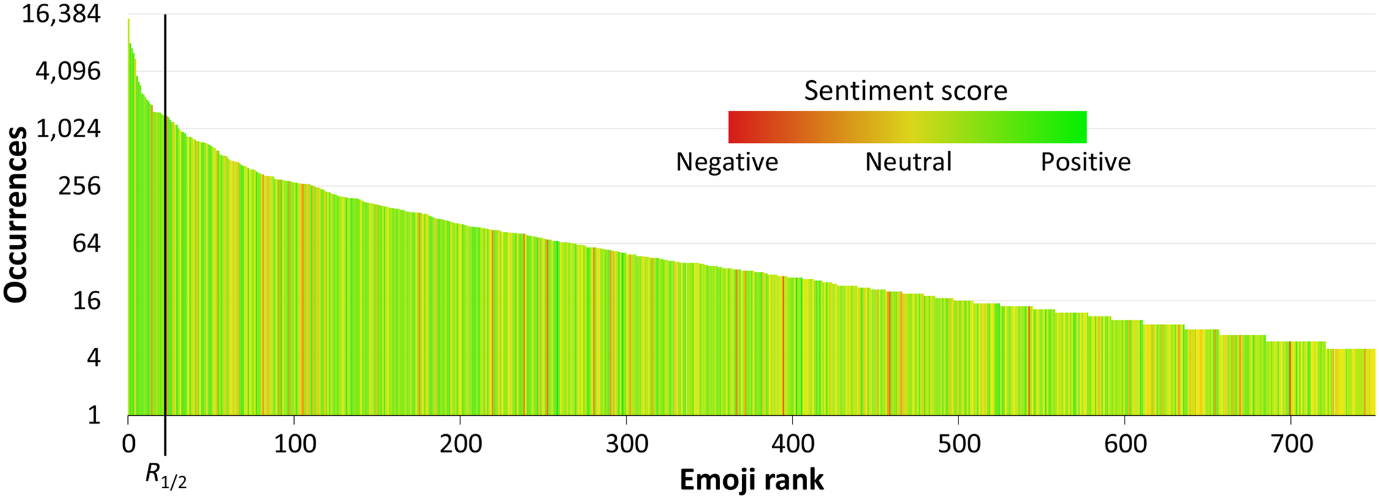
Example of sarcasm: “My train has been cancelled. Amazing”

A person will be able to recognize the sarcasm in the above example, however, a computer program monitoring the above example with no context will not be able too, and would brand the Sentiment as positive; when it is negative in sentiment. [4]

As well as Sentiment Analysis being developed to allow businesses and political parties to analysis the feelings of the public towards them, another promising area of development in Sentiment Analysis is the analysis of emoji’s being used in people’s social media posts on Facebook and Twitter and the Sentiment that they represent. This area of investigation was explored in the implementation of Section 2- Method.

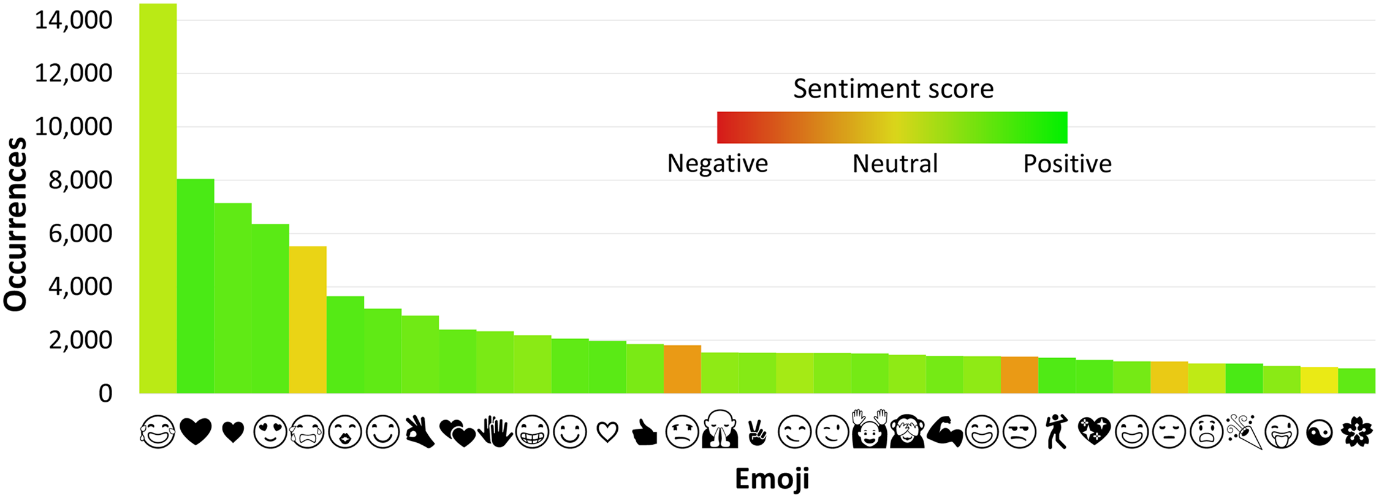
In the past year, there has been many technological trends in Sentiment Analysis. [5] One of these trends has been ISO emoji analytics. Emoji’s are increasingly being used to express emotion and feeling on social media platforms. In the last two years more than 10 billion emoji’s have been used on Twitter. Emoji’s are Unicode graphic symbols, used to express feelings and ideas, without writing text or sentences to convey what you mean or feel. However, what Sentiment do the emoji’s express, as they are just small pictures and ‘smiley faces’. The Sentiment of the emoji’s are generated from the Sentiment of the tweets, they appear in and it turns out that most of the emoji’s are positive, especially the most popular ones. They are easy to use, fun and compact quickly displaying people’s feelings. The future of emoji analysis is exciting and is currently being developed by start-ups like, emogi. [5]

Below on the next page are graphs of emoji analysis from: [5] [5]<http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0144296#pone-0144296-g006>



**Figure 3. Distribution of occurrences and sentiment of the 751 emojis.**

The emojis are ranked by their occurrence (log scale). The column colour indicates the sentiment score. The partitioning into two equally weighted halves’ is indicated by a line at *R*1/2.

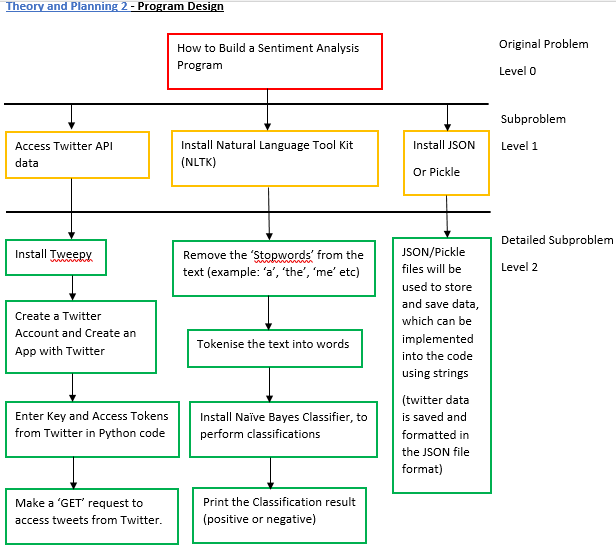


**Figure 4. Top 33 emojis by occurrence.**

Column colour represents the emoji sentiment score.

**Method- Implementation of the Sentiment Analysis Program**

The first step for building the Sentiment Analysis program was planning the project, as from investigation and reading professional papers and online resources, this generated a wealth of knowledge to product the below Program Design Flow Diagram, which divided the project into workable and manageable tasks to result in the completion of a successful and working Sentiment Analysis program.



**Figure 5: Program Design Flow Diagram**– Explanation of the Program and overview of what is needed to successfully run the program

The above Program Design Flow Diagram clearly depicts the requirements, concept and packages required to build the Sentiment Analysis program. The code used to generate the Sentiment Analysis program (which will be explained in detail later in Section 2- Method) was taken from third party libraries and online tutorials and adapted to fulfil the requirements of the Sentiment Analysis program built in this report. This report and program developed show that you do not need to start writing your code from the beginning, you can use third party libraries just make sure any code you do use you understand what it is doing. There is no need to re-invent the wheel.

To build the Sentiment Analysis program detailed below please refer to the User Guide and Start-Up Guide- How to Build a Sentiment Analysis Program, which has been uploaded to Module. (too big to add as an appendix in this report)

Before implementation of code could begin, there was a stage of downloading all the packages needed to carry out Sentiment Analysis using Pycharm (python language interpreter) and Python (programming language) below is the list of packages used and downloaded:

* Tweepy: this program is the official Twitter API client for Python.
* NLTK corpora: Corpora is a large and structured set of texts.
* Matplotlib: import matplotlib allows me to present the live Twitter data in the form of and line graph, measuring the sentiment of the tweets
* NumPy: used for array processing for numbers, strings, records and objects
* SciPy: Scientific library for python

**Stage 1 of Implementation- Training and Testing Naïve Bayes Classifier**

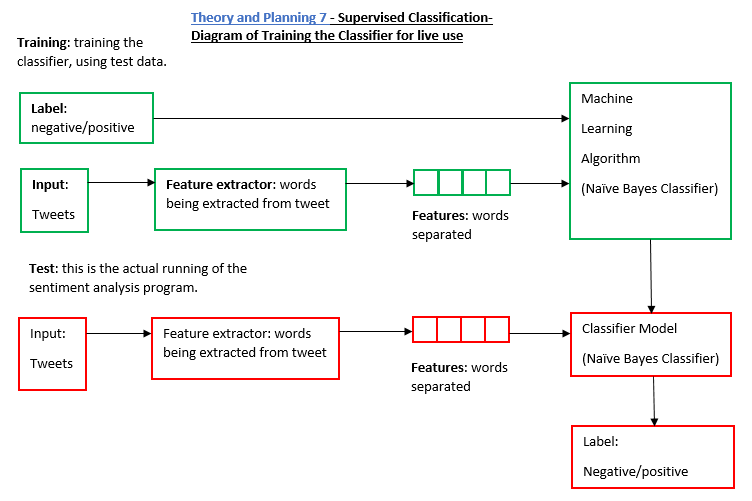
The first piece of code used to build the Sentiment Analysis program was developed and designed to train and test the Naïve Bayes Classifier. The Naïve Bayes Classifier algorithm is used to classify the tweets as positive or negative, it is a popular algorithm for classifying text data. The classifier is trained and tested using a process called supervised classification, which works by separating the data into 2 categorizes training data and test data from the total dataset. For example, the dataset total could be 2000 and this is split into 2 categorizes training data which has 1500 datasets and test data which has 500 datasets.

1500 (training data) + 500 (test data) = 2000 (total dataset)

So, the total dataset is used to train and test the classifier of the program.

“This is called supervised machine learning, because we're showing the machine data, and telling it hey, this data is positive, or this data is negative. Then, after that training is done, we show the machine some new data and ask the computer, based on what we taught the computer before, what the computer thinks the category of the new data is.”

[Appendix 1] <https://pythonprogramming.net/naive-bayes-classifier-nltk-tutorial/>

Below please see the flow diagram showing how the Supervised Classification works within the program.

**Figure 6: Supervised Classification Flow Diagram**

**Please see** **Appendix 1- Code for Training the Naïve Bayes Classifier with Comments**

**Stage 2 of Implementation- Creating a Module for Sentiment Analysis**

This piece of code builds on the code used from the last piece of code for training and testing the classifier. Stage 2 of implementation creates a class for my classifier to run through the dataset and the classifier must classify the data as either positive or negative. Stage 2 also defines, and function called ‘classify’ which can be call upon in future piece of code. Importantly in this piece of code the classifier is being told exactly what type of words to use to classify the data as either positive or negative, the program will be analysing adjectives (which are describing words) to carry out Sentiment Analysis. These words are then tokenized and appended and saved to a pickle file to be called upon later in the code.

**Please see** **Appendix 2- Code for Creating a Module for Sentiment Analysis with Comments**

**Stage 3 of Implementation- Sentiment Module**

Stage 3 of the implementation of the Sentiment Analysis program is very straight forward as all the work needed to carry out this stage has already been done, and all that is being done in this code is all the previously serialized pickle files are being called upon and opened. The only new element to the code is the last three lines, which defines and creates the function for Sentiment Analysis to be carried out on live Twitter data by calling on this function:

def sentiment(text):

feats = find\_features(text)

return voted\_classifier.classify(feats), voted\_classifier.confidence(feats)

**Please see** **Appendix 3- Code for Sentiment Module with Comments**

**Stage 4 of Implementation- Twitter Sentiment Analysis**

To perform live Twitter Sentiment Analysis the code features a line at the top of the code which imports a package called Tweepy. Tweepy is the official Twitter API client for Python and allows the program to retrieve tweets from Twitter API, to access the Twitter API you also need a Twitter account and you must register an app through your own twitter account. Once you have register your app through Twitter you will receive ‘consumer secret’, ‘consumer key’, ‘access token’ and ‘access token secret’. Then these passwords which must be kept secret and private are write into the code and allow access to Twitter data for live Sentiment Analysis. The next step within the code is to stream Twitter data for a specific query and to generate Sentiment Analysis about that query.

**Please see** **Appendix 4- Code for Live Twitter Sentiment Analysis with Comments**

**Review of Results**

* From the results of the Sentiment Analysis program the naïve bayes classifier is more negative than positive and the biggest problem to Sentiment Analysis, which is its inability to detect sarcasm occurs within the results of the code run.
* For example: RT @graham\_williams: Imagine saying 10 years ago that an action comedy with a short-haired Thor would open bigger at the box office than… pos 1.0

**Problems with this result**

* The program has not detected the sarcasm from the author of the tweet, as any human with basic knowledge of comic books would know that ‘Thor’ famously has long blonde hair and the author of the tweet is implying that ‘justice league’ is so bad that a film with a not very stereotypical short haired ‘Thor’ is more successful than justice league.
* Also, this tweet has been categorized has positive when it is a negative tweet about ‘justice league’

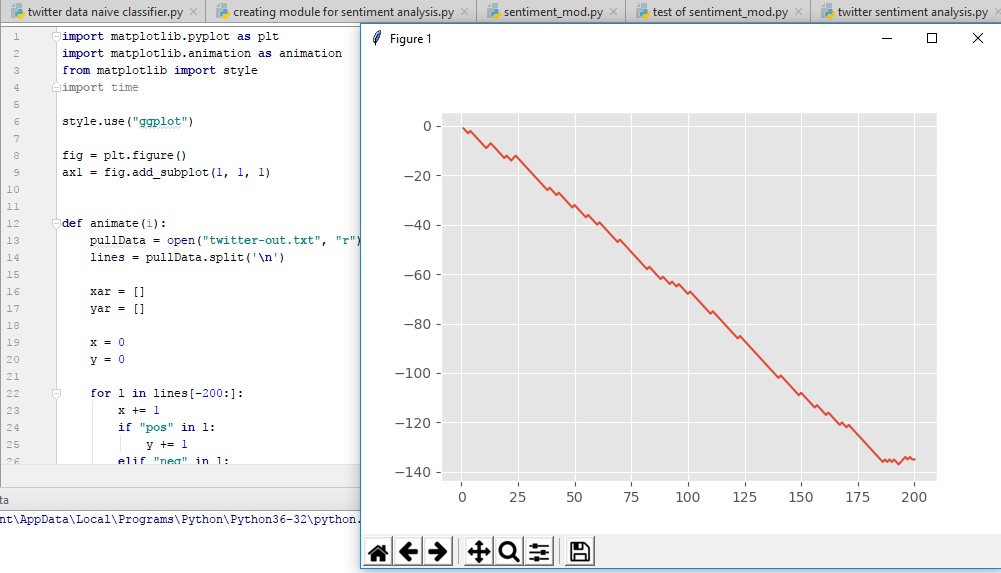
**Stage 5 of Implementation- Graphing Live Twitter Sentiment Analysis Live**

After the program had run and successfully performed Sentiment Analysis on live Twitter data, the next step was to display the Sentiment of the data in a visual and understandable graphical output. The line graph is generated from the ‘twitter-output.txt’ file created in the last piece of code for receiving live tweets from Twitter and was used to produce the graph showed below which depicts the live Sentiment of the first 200 tweets about my search query ‘Justice League’ over the time the program was running, so the graph shows the Sentiment over a specific period. The key to the code used to generate the graph below was line 1 and 2, which imported matplotlib:

import matplotlib.pyplot as plt

import matplotlib.animation as animation

import matplotlib allowed the code to present the live Twitter data in the form of and line graph, measuring the sentiment of the tweets.



**Figure 7**: Graph of live Twitter Sentiment for Justice League

**Please see** **Appendix 5- Code for Graphing Live Twitter Sentiment Analysis with Comments**

**Reflection: About the Overall Project from Beginning to Final Program**

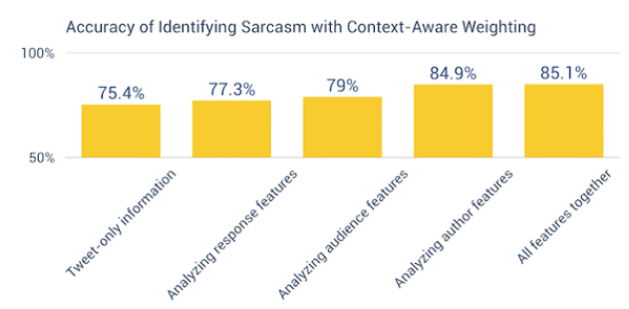
* At the beginning of the project the dataset being used to train and test the Naïve Bayes Classifier was the movie\_reviews dataset from the NLTK corpus, which was too small a dataset to train and test the classifier for live Twitter data. The dataset was only 2000 (1000 negative reviews and 1000 positive reviews), also the movie reviews were too long to train a classifier for Twitter data which only allow 280 characters (recently increased from 140 characters).
* Next during the implementation process Scikit-Learn was installed to access more classifiers to help increase the accurate and reliability of the program as the data would be passed through multiple classifiers instead of just the Naïve Bayes Classifier, at this stage the movie\_reviews corpus was still being used. List of Classifiers used: Naïve Bayes,MultinomialNB, BernoulliNB, LogisticRegression, SGDClassifier, SVC, LinearSVC, NuSVC
* The seven classifiers were used in the code to pass through the training and testing dataset and print an accuracy result for each individual classifier and then a function was created to take the average of the seven classifiers and the final accuracy for the classifier would be the average accuracy result that would be printed out after the code had ran.
* However, when the dataset was changed from the movie\_reviews corpus from the NLTK to the short movie review samples, which was downloaded from online (this dataset was larger 10000 reviews and like the length of real Twitter data) the seven classifiers were not able to run through the dataset and a memory error was produced when the code was run. That’s why the final program only uses the Naïve Bayes Classifier.
* To improve the accuracy of the classifier in the future, the classifier will be trained and tested with a bigger dataset with a bigger range of writing styles, emoji’s, sarcasm, slang, spelling mistakes and language. This would improve the performance and accuracy of the program in future development.

Sentiment Analysis is never 100% accurate. Sometimes, the Sentiment Analysis is only between 50% -70% accurate. [6] Inaccuracy is caused by many factors, one of the reasons for inaccuracy and difficulties for the Sentiment program is not all sentences express emotion or feelings; they are sometime just factual sentences. [6]

[6]<http://brnrd.me/sentiment-analysis-never-accurate/>

[7]But, the biggest source of inaccuracy in the program is the ability to recognise sarcasm, tone and context. This is because most Sentiment Analysis programs are designed to recognise certain words as “positive” or “negative” Sentiment. For example, words such as: “brilliant”, “amazing”, “great” are designed as “positive”. However, words such as: “angry”, “sad”, “evil” are designed as negative. Inaccuracy example, if someone writes on Twitter: “I am happy, I am skint for the week.” The programme will pick-up the word “happy” and will categorize the Sentiment as “positive”, when it is “negative” and the twitter user is being sarcastic. The truth is the solution for sarcasm in Sentiment Analysis program’s is a long way off, because people cannot even successfully determine and identify sarcasm accurately, so how can you expect the program to successfully identify sarcasm. However, there is a potential solution to this problem Context aware weighting. The solution was developed by David Bamman and Noah A. Smith, the solution looks at the context around the tweet. [7]“Rather than just analyzing tweets on their own, the model constructed by Bamman and Noah also looks at attributes of the author (author features), attributes of the intended recipient of a tweet (audience features), and the attributes of responses to potentially sarcastic tweets (response features).” Quote from and image below from:

[7]<http://blog.infegy.com/sentiment-analysis-and-sarcasm>



**Figure 8**: Accuracy of Identifying Sarcasm with Context-Aware Weighting

**Conclusion**

This report has discussed what, how and why Sentiment analysis is important, and introduced the key concepts and understanding required to investigate and experiment further with Sentiment Analysis. Sentiment Analysis is an important field of study and offers crucial information to companies and businesses, as it provides an insight to what customers of their products think of their products and it allows potential future customers to see what existing customers think about the products. [8] Sentiment Analysis has many advantages: it is a faster and lower cost method of receiving customer opinion, it gives businesses the opportunity to reaction to customer feedback and suggestions, it can also highlight a company’s strengths, weaknesses, threats and opportunities. [8] Finally, 80% of a business’s data is expressed in words, so a Sentiment Analysis program is vital. However, Sentiment Analysis does come with limitations and the biggest is, it is never 100% accurate.

In conclusion, sentiment analysis is a relatively new program with massive potential for future development and improvement. Programmers have only scratched the surface of Sentiment Analysis’s full capabilities, as at present it is limited in design as most Sentiment program’s only analysis key words. However, with more companies and analysists working toward the improvement of analysis and investigating emoji’s, reactions, like and unlike and video content across all different platforms such as: Facebook and Twitter etc. So, to conclude Sentiment Analysis will become more accurate and development will improve significantly within the years to come and this program will become more intelligent the more data and information it gathers about the users of social media and the internet.

**Glossary**

**Tweepy** - is the python client for the official Twitter API to gather tweets through Twitter API, to do this we need to register an App through a twitter account.

**Natural Language Toolkit (NLTK)** - is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

**Tokenization** - Tokenizes the tweet, separates each word from the body of text as tokens.

**Supervised Classification** - works by separating the data into 2 categorizes training data and test data from the total dataset. Working example, the dataset total could be 2000 and this is split into 2 categorizes training data which has 1500 datasets and test data which has 500 datasets.

1500 (training data) + 500 (test data) = 2000 (total dataset)

**Random** - This is used to shuffle the dataset the naïve bayes classifier is being trained and tested with, as the dataset is separated into positive and negative is command mixes them up.

**Pickle** - Pickle is used as not all classifiers run in a few minutes some can take hours, so pickle allows us to run the classifier once and save the file as a serialized file that can be loaded into code and won’t take long to run your code, as the process and dataset of the classifier has already been carried out within the loaded pickle file.

**Write in bytes “wb”** - “wb” this is how it will appear in the code, the code is writing the data in binary

**Read in bytes “rb”** - “rb” this is how it will appear in the code; the code is being read in binary because it was in binary.

**Sentiment Analysis** - is the measure of positive and negative language or text. It is a method of gauging written or spoken language to determine if the statement is favourable, unfavourable or neutral and to what extent.

**JSON** - JSON files will be used to store and save data, which can be implemented into the code using strings (twitter data is saved and formatted in the JSON file format)

**Classes** - A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

Definition from: https://www.tutorialspoint.com/python/python\_classes\_objects.htm

**Matplotlib** – Matplotlib is a Python 2D plotting library that can generate tables and graphs displaying statistical data effectively and clearly.

**Matplotlib Animation** - Is used to graph live Twitter data sentiment directly onto a line graph within this program and code used.

**NumPy**: used for array processing for numbers, strings, records and objects

**SciPy:** Scientific library for python

**Function (def)** - The function is defined with 3 components: 1- header ‘def’ followed by name of the function or parameters. 2- optional comment which explains the function. 3- Body of the function which detail the procedures the function will carry out.

**Appendix**

**Appendix 1- Code for Training the Naïve Bayes Classifier with Comments**

import nltk

import random

import pickle

from nltk.tokenize import word\_tokenize

# I will be using the NLTK to assist me in building my naive bayes classifier

# import random will be used to shuffle my training and testing dataset of short movie reviews

# to make the classifier accurate and reliable when processing live tweets

# My dataset has already been labelled as positive and negative, making it possible to train and test with

# See the user guide for instructions on how to download the positive and negative.txt files for training and testing classifier.

# 2 two lines below open the text files and reads the text data contained within.

short\_pos = open("positive.txt", "r").read()

short\_neg = open("negative.txt", "r").read()

documents = []

# documents equals empty list

# r equals review, so for every review split them with a new line

for r in short\_pos.split('\n'):

documents.append((r, "pos"))

for r in short\_neg.split('\n'):

documents.append((r, "neg"))

random.shuffle(documents)

#This will mix up the positive and negative documents

all\_words = []

#all words equals empty list

# The code below will tokenize the words in the positive and negative text files.

short\_pos\_words = word\_tokenize(short\_pos)

short\_neg\_words = word\_tokenize(short\_neg)

for w in short\_pos\_words:

all\_words.append(w.lower())

for w in short\_neg\_words:

all\_words.append(w.lower())

all\_words = nltk.FreqDist(all\_words)

# The line of code above will form a list of the most common words in the text files.

word\_features = list(all\_words.keys())[:5000]

# The above line of code records the most common 5000 words from both text files.

def find\_features(document):

words = word\_tokenize(document)

features = {}

for w in word\_features:

features[w] = (w in words)

return features

# The line of code below does this to all documents, saving the feature existence booleans and the positive or negative categories

featuresets = [(find\_features(rev), category) for (rev, category) in documents]

random.shuffle(featuresets)

#This mixes up the positive and negative featuresets

# The below code follows the process of the supervised classification flow diagram that can be found on page ? of the user guide

# dataset I will train classifier with

training\_set = featuresets[:10000]

# dataset I will test classifier against

testing\_set = featuresets[10000:]

# The below code defines and trains my naive bayes classifier

#classifier = nltk.NaiveBayesClassifier.train(training\_set)

# The 3 lines of code below open the previously saved pickle file to run in the code

classifier\_f = open("naivebayes.pickle", "rb")

classifier = pickle.load(classifier\_f)

classifier\_f.close()

print("Classifier accuracy percent:",(nltk.classify.accuracy(classifier, testing\_set))\*100)

classifier.show\_most\_informative\_features(15)

# The 2 lines of code above give me a list of the 15 most informative words when the code is run and the accuracy of test data

# The code below allows me to save the naive bayes classifier process of running through the dataset

# By inserting import pickle at the top of the code, I can serialize the classifier and load it into my code, this saves time.

#save\_classifier = open("naivebayes.pickle","wb")

#pickle.dump(classifier, save\_classifier)

#save\_classifier.close()

# The above 3 lines of code saved and stored the results of my code in a pickle file, to be accessed at any point in the future.

# Then I commented off the above code with the ‘#’ to allow me to carry out my next lines of code.

# which involved me uploading the saved data from the pickle file straight back into my code, See below for how it works.

# This opens up a pickle file, preparing to write in bytes some data.

# I used pickle.dump() to dump the data.

# The first parameter to pickle.dump() is what are you dumping.

# The second parameter is where are you dumping it.

# Close the file and now I have a pickle file saved.

# Reference for the code: <https://pythonprogramming.net/naive-bayes-classifier-nltk-tutorial/>

**Appendix 2- Code for Creating a Module for Sentiment Analysis with Comments**

import nltk

import random

import pickle

from nltk.classify import ClassifierI

from statistics import mode

from nltk.tokenize import word\_tokenize

# import nltk gives me access to the nltk libraries of data and programs for data analysis

# import random will be used to shuffle my training and testing dataset of short movie reviews

# to make the classifier accurate and reliable when processing live tweets

# My dataset has already been labelled as positive and negative, making it possible to train and test with

# import pickle will insert my previously saved and serialised file of my naive bayes classifier and most common 5000 words

# word\_tokenize will tokenizes the dataset, separating each word from the body of text as tokens

# I imported mode, this will choose the most popular classifier vote (this code was used when I had more classifiers in the code)

# Line classifierI is the classifier being used on the data

# The class below is for my classifier

# The classifier is called VoteClassifier and is inherting ClassifierI

# The classifiers well in this case the naive bayes classifier is programmed to pass through the class to self.classifier

# In the second function 'def classify' I define my classify process, so I can call on it later on.

# The functions below are passing through the classifier and classifying by features

# The classification is being processed as a vote (was more effective when I had more classifiers)

# Finally the class returns the the mode(vote), the most popular classifier (again better when you have more classifiers)

class VoteClassifier(ClassifierI):

def \_\_init\_\_(self, \*classifiers):

self.\_classifiers = classifiers

def classify(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

return mode(votes)

def confidence(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

choice\_votes = votes.count(mode(votes))

conf = choice\_votes / len(votes)

return conf

# confidence function is simply counting up the votes for each classifiers (again working better when you use more classifiers)

# See the user guide for instructions on how to download the positive and negative.txt files for training and testing classifier.

# 2 two lines below open the text files and reads the text data contained within.

short\_pos = open("positive.txt", "r").read()

short\_neg = open("negative.txt", "r").read()

all\_words = []

documents = []

# all\_words equals empty list

# documents equals empty list

# j is adjective, r is adverb, and v is verb

# allowed\_word\_types = ["J","R","V"]

allowed\_word\_types = ["J"]

# I am only looking for adjectives in the dataset

for p in short\_pos.split('\n'):

documents.append((p, "pos"))

words = word\_tokenize(p)

pos = nltk.pos\_tag(words)

for w in pos:

if w[1][0] in allowed\_word\_types:

all\_words.append(w[0].lower())

# The above if statement is saying if the word is an adjective I want to append that word

for p in short\_neg.split('\n'):

documents.append((p, "neg"))

words = word\_tokenize(p)

pos = nltk.pos\_tag(words)

for w in pos:

if w[1][0] in allowed\_word\_types:

all\_words.append(w[0].lower())

# The above if statement is saying if the word is an adjective I want to append that word

# Below I am saving the words in a pickle file

save\_documents = open("documents.pickle", "wb")

pickle.dump(documents, save\_documents)

save\_documents.close()

# The above 3 lines of code saved and stored the results of my code in a pickle file, to be accessed at any point in the future.

all\_words = nltk.FreqDist(all\_words)

# The line of code above will form a list of the most common words in the text files.

word\_features = list(all\_words.keys())[:5000]

# The above line of code records the most common 5000 words from both text files.

save\_word\_features = open("word\_features5k.pickle", "wb")

pickle.dump(word\_features, save\_word\_features)

save\_word\_features.close()

# The above 3 lines of code saved and stored the results of my code in a pickle file, to be accessed at any point in the future.

def find\_features(document):

words = word\_tokenize(document)

features = {}

for w in word\_features:

features[w] = (w in words)

return features

# The line of code below does this to all documents, saving the feature existence booleans and the positive or negative categories

featuresets = [(find\_features(rev), category) for (rev, category) in documents]

random.shuffle(featuresets)

#This mixes up the positive and negative featuresets

print(len(featuresets))

# The line of code above prints the length of the dataset (total number of positive and negative datasets)

# dataset I will test classifier against

testing\_set = featuresets[10000:]

# dataset I will train classifier with

training\_set = featuresets[:10000]

classifier = nltk.NaiveBayesClassifier.train(training\_set)

print("Original Naive Bayes Algo accuracy percent:", (nltk.classify.accuracy(classifier, testing\_set)) \* 100)

classifier.show\_most\_informative\_features(15)

# The above lines of code will print the percentage accuracy of the naive bayes classifier and the 15 most common words

save\_classifier = open("originalnaivebayes5k.pickle", "wb")

pickle.dump(classifier, save\_classifier)

save\_classifier.close()

# The above 3 lines of code saved and stored the results of my code in a pickle file, to be accessed at any point in the future.

# 'open' create a new pickle file

# 'wb' means write in bytes

# I used pickle.dump() to dump the data.

# The first parameter to pickle.dump() is what are you dumping.

# The second parameter is where are you dumping it.

# Close the file and now I have a pickle file saved.

# Reference for code used: <https://pythonprogramming.net/sentiment-analysis-module-nltk-tutorial/>

**Appendix 3- Code for Sentiment Module with Comments**

import random

import pickle

from nltk.classify import ClassifierI

from statistics import mode

from nltk.tokenize import word\_tokenize

# import random will be used to shuffle my training and testing dataset of short movie reviews

# to make the classifier accurate and reliable when processing live tweets

# My dataset has already been labelled as positive and negative, making it possible to train and test with

# import pickle will insert my previously saved and serialised file of my naive bayes classifier and most common 5000 words

# word\_tokenize will tokenizes the dataset, separating each word from the body of text as tokens

# The class below is for my classifier

# The classifier is called VoteClassifier and is inherting ClassifierI

# The classifiers well in this case the naive bayes classifier is programmed to pass through the class to self.classifier

# In the second function 'def classify' I define my classify process, so I can call on it later on.

# The functions below are passing through the classifier and classifying by features

# The classification is being processed as a vote (was more effective when I had more classifiers)

# Finally the class returns the the mode(vote), the most popular classifier (again better when you have more classifiers)

class VoteClassifier(ClassifierI):

def \_\_init\_\_(self, \*classifiers):

self.\_classifiers = classifiers

def classify(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

return mode(votes)

def confidence(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

choice\_votes = votes.count(mode(votes))

conf = choice\_votes / len(votes)

return conf

# confidence function is simply counting up the votes for each classifiers (again working better when you use more classifiers)

# The 3 lines of code below are simply opening the pickle file of the documents, I created in the last piece of code.

documents\_f = open("documents.pickle", "rb")

documents = pickle.load(documents\_f)

documents\_f.close()

# The 3 lines of code below are simply opening the pickle file of the word features, I created in the last piece of code.

word\_features5k\_f = open("word\_features5k.pickle", "rb")

word\_features = pickle.load(word\_features5k\_f)

word\_features5k\_f.close()

# The lines of code below are creating a function (def) to tokenize the words contained within the document

def find\_features(document):

words = word\_tokenize(document)

features = {}

for w in word\_features:

features[w] = (w in words)

return features

# The line of code below does this to all documents, saving the feature existence booleans and the positive or negative categories

featuresets = [(find\_features(rev), category) for (rev, category) in documents]

#This mixes up the positive and negative featuresets

random.shuffle(featuresets)

print(len(featuresets))

# The line of code above prints the length of the dataset (total number of positive and negative datasets)

# dataset I will test classifier against

testing\_set = featuresets[10000:]

# dataset I will train classifier with

training\_set = featuresets[:10000]

# The 3 lines of code below are opening the pickle file of the naive bayes classifier

# The pickle file has saved the classifier, this reduces the run time of the program especially if being used on large dataset

open\_file = open("originalnaivebayes5k.pickle", "rb")

classifier = pickle.load(open\_file)

open\_file.close()

voted\_classifier = VoteClassifier(

classifier)

# The last piece of code is the most important piece of code for the next python files

# The function called 'sentiment' is created and takes the text, analyses the features of the text using find\_features and returns the sentiment (positive or negative)

def sentiment(text):

feats = find\_features(text)

return voted\_classifier.classify(feats), voted\_classifier.confidence(feats)

# Reference for the code used: <https://pythonprogramming.net/sentiment-analysis-module-nltk-tutorial/>

**Appendix 4- Code for Live Twitter Sentiment Analysis with Comments**

from tweepy import Stream

from tweepy import OAuthHandler

from tweepy.streaming import StreamListener

import json

import sentiment\_mod as s

# The above import sentiment\_mod as s line of code imports the sentiment analysis fuction from the last piec of code.

# tweepy is the python client for the official Twitter API to gather tweets through Twitter API

# To receive your own consumer key and secret, access token and secret

# You need to view my user guide for the instructions on how to do this

# Consumer keys and access tokens, used for OAuth (OAuthentication)

# It is used for security and to make sure you have permission to use the Twitter API data

consumer\_key = ""

consumer\_secret = ""

access\_token = ""

access\_secret = ""

# Line 4 of the code 'import json' allows me to use the json module to load the tweet data with the code below 'json.loads(data)'

# The line of code below 'tweet = all\_data["text"]' allows me to target the tweets specifically

# Once I have a tweet I can pass it through the sentiment\_mod

# In the code below the line 'output = open("twitter-out.txt", "a")' and the code that follows will output the tweets into a json file with its sentiment score

class listener(StreamListener):

def on\_data(self, data):

all\_data = json.loads(data)

tweet = all\_data["text"]

sentiment\_value, confidence = s.sentiment(tweet)

print(tweet, sentiment\_value, confidence)

if confidence \* 100 >= 80:

output = open("twitter-out.txt", "a")

output.write(sentiment\_value)

output.write('\n')

output.close()

return True

def on\_error(self, status):

print(status)

# OAuth process, using the keys and tokens

auth = OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_token, access\_secret)

twitterStream = Stream(auth, listener())

twitterStream.filter(track=["justice league"])

# The stream is retreiving all the live tweets about your chosen query, in this case justice league.

# Reference for the code used: <https://pythonprogramming.net/twitter-sentiment-analysis-nltk-tutorial/>

**Appendix 5- Code for Graphing Live Twitter Sentiment Analysis with Comments**

import matplotlib.pyplot as plt

import matplotlib.animation as animation

from matplotlib import style

import time

# import matplotlib allows me to present the live Twitter data in the form of and line graph, measuring the sentiment of the tweets

#

# style 'ggplot' just makes the graph look better and pleasing to the eye

style.use("ggplot")

fig = plt.figure()

ax1 = fig.add\_subplot(1, 1, 1)

# the pullData is the data I will use to construct the graph

# The pullData I will be using is the 'twitter-out.txt' file I created in the last piece of code

# lines equal to pullData split by new line

# 'xar' X array equals empty list

# 'yar' Y array equals empty list

# The line of code 'for l (line) in lines [-200:]:' is saving the graph data when it reaches 200 tweets

# The function below is constructing the graph

# x=0 and y=o are the starting points of the graph

# If the tweet is positive y = plus one and if the tweet is negative y= minus 1

def animate(i):

pullData = open("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()

# Reference for the code used:https://pythonprogramming.net/graph-live-twitter-sentiment-nltk-tutorial/

**References**

[1] Clarabridge. (2017). *Sentiment Analysis | What is Sentiment Analysis?*. [online] Available at: https://www.clarabridge.com/sentiment-analysis/ [Accessed 9 Dec. 2017].

[2] Anon, (2017). [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.383.5491&rep=rep1&type=pdf [Accessed 9 Dec. 2017].

[3] Arxiv.org. (2017). *Cite a Website - Cite This For Me*. [online] Available at: https://arxiv.org/pdf/1407.0622v1.pdf [Accessed 9 Dec. 2017].

[4] Brandwatch. (2017). *Sentiment Analysis: How Does It Work? Why Should We Use It?*. [online] Available at: https://www.brandwatch.com/blog/understanding-sentiment-analysis/ [Accessed 9 Dec. 2017].

[5] Kralj Novak, P., Smailović, J., Sluban, B. and Mozetič, I. (2017). *Sentiment of Emojis*.

[6]Donkor, B. (2017). *Social Sentiment Analysis: Why It's Never 100% Accurate*. [online] Brnrd.me. Available at: http://brnrd.me/sentiment-analysis-never-accurate/ [Accessed 9 Dec. 2017].

[7] Martin, R. (2017). *Can sentiment analysis detect sarcasm on the internet? Oh yeah, totally...*. [online] Blog.infegy.com. Available at: http://blog.infegy.com/sentiment-analysis-and-sarcasm [Accessed 9 Dec. 2017].

[8] Patil (2017). *Introduction to Sentiment Analysis*. [online] Slideshare.net. Available at: https://www.slideshare.net/makrandp/introduction-27376010 [Accessed 9 Dec. 2017].

[Appendix 1] Pythonprogramming.net. (2017). *Python Programming Tutorials*. [online] Available at: https://pythonprogramming.net/naive-bayes-classifier-nltk-tutorial/ [Accessed 9 Dec. 2017].

[Appendix 3 and 2] Pythonprogramming.net. (2017). *Python Programming Tutorials*. [online] Available at: https://pythonprogramming.net/sentiment-analysis-module-nltk-tutorial/ [Accessed 9 Dec. 2017].

[Appendix 4] Pythonprogramming.net. (2017). *Python Programming Tutorials*. [online] Available at: https://pythonprogramming.net/twitter-sentiment-analysis-nltk-tutorial/ [Accessed 9 Dec. 2017].

[Appendix 5] Pythonprogramming.net. (2017). *Python Programming Tutorials*. [online] Available at: https://pythonprogramming.net/graph-live-twitter-sentiment-nltk-tutorial/ [Accessed 9 Dec. 2017].