



Callum Haine

CM3203: One semester individual project

Supervised by Jose Camacho Collados

---

# Final Report

---

‘Comparing machine learning techniques for analysing and tracking the  
*sentiment of Tweets*’

## 1 Contents

|       |   |    |
|-------|---|----|
| 2     | Table of figures.....                       | 4  |
| 3     | Abstract.....                               | 5  |
| 4     | Acknowledgments .....                       | 5  |
| 5     | Introduction .....                          | 5  |
| 6     | Background.....                             | 7  |
| 6.1   | Twitter API.....                            | 7  |
| 6.1.1 | Twitter API overview .....                  | 7  |
| 6.1.2 | Tweepy.....                                 | 7  |
| 6.1.3 | Other tools.....                            | 8  |
| 6.2   | Sentiment analysis.....                     | 8  |
| 6.2.1 | Sentiment analysis overview.....            | 8  |
| 6.2.2 | Sentiment definition.....                   | 8  |
| 6.2.3 | Sentiment analysis goals .....              | 9  |
| 6.2.4 | Applications of sentiment analysis .....    | 9  |
| 6.3   | Machine learning.....                       | 10 |
| 6.3.1 | Machine learning overview.....              | 10 |
| 6.3.2 | Supervised vs unsupervised approaches ..... | 10 |
| 6.3.3 | Classification vs regression.....           | 10 |
| 6.3.4 | Fitting.....                                | 10 |
| 6.3.5 | Features .....                              | 11 |

|        |   |    |
|--------|---|----|
| 6.3.6  | Training, validation, and test sets ..... | 12 |
| 6.3.7  | Cross validation .....                    | 12 |
| 6.3.8  | Evaluation measures .....                 | 13 |
| 6.3.9  | Logistic regression .....                 | 14 |
| 6.3.10 | Optimising logistic regression .....      | 15 |
| 6.3.11 | Support Vector Machine .....              | 15 |
| 6.4    | Pre-processing .....                      | 16 |
| 6.4.1  | Overview .....                            | 16 |
| 6.4.2  | Stemming and lemmatization .....          | 16 |
| 6.4.3  | Standardisation.....                      | 17 |
| 6.5    | Additional python libraries.....          | 17 |
| 6.5.1  | Scikit-learn.....                         | 17 |
| 6.5.2  | NLTK.....                                 | 17 |
| 6.5.3  | matplotlib.....                           | 18 |
| 6.5.4  | Pickle.....                               | 18 |
| 7      | Data collection process .....             | 18 |
| 7.1    | Identifying brands .....                  | 18 |
| 7.2    | Tweepy .....                              | 19 |
| 7.3    | Subjective vs objective Tweets .....      | 19 |
| 7.4    | Method for annotating data .....          | 20 |
| 7.5    | Criteria for annotating data .....        | 20 |
| 7.5.1  | Positive sentiment .....                  | 21 |
| 7.5.2  | Negative sentiment.....                   | 21 |
| 7.5.3  | Neutral sentiment.....                    | 22 |
| 7.6    | Increasing dataset size.....              | 23 |
| 8      | Implementation.....                       | 23 |
| 8.1    | Pre-processing .....                      | 23 |

|       |   |    |
|-------|---|----|
| 8.2   | Model selection .....                         | 25 |
| 8.2.1 | Classifier advantages and disadvantages ..... | 25 |
| 8.2.2 | Model requirements .....                      | 26 |
| 8.2.3 | Methodology for testing model.....            | 26 |
| 8.2.4 | Logistic regression model.....                | 28 |
| 8.2.5 | SVM model.....                                | 31 |
| 8.2.6 | Logistic regression vs SVM.....               | 33 |
| 8.3   | Full pipeline including live tracking .....   | 34 |
| 8.3.1 | Methodology for final pipeline .....          | 34 |
| 8.3.2 | Implementation of pipeline .....              | 35 |
| 9     | Evaluation.....                               | 39 |
| 9.1   | Evaluation of pre-processing.....             | 39 |
| 9.2   | Evaluation of fact/opinion classifier .....   | 39 |
| 9.3   | Evaluation of different datasets.....         | 40 |
| 9.4   | Evaluation of final classifier .....          | 40 |
| 9.5   | Evaluation of full pipeline.....              | 40 |
| 10    | Conclusion .....                              | 41 |
| 11    | Future expansion.....                         | 42 |
| 11.1  | Data collection.....                          | 43 |
| 11.2  | Model testing and selection.....              | 43 |
| 11.3  | Full pipeline.....                            | 43 |
| 12    | Reflection.....                               | 44 |
| 13    | References.....                               | 45 |

## 2 Table of figures

|   |    |
|---|----|
| Figure 1, visual representation of the concept of fitting [9] ..... | 11 |
| Figure 2, visual representation of k-fold cross validation.....     | 12 |

|  |    |
|--|----|
| Figure 3, function used to pre-process tweets.....   | 25 |
| Figure 4, a graph showing how the training set size affected accuracy of the model. ....       | 28 |
| Figure 5, comparison of standard and custom pre-processing techniques.....                     | 29 |
| Figure 6, effect of using scaled and unscaled data.....  | 29 |
| Figure 7, flowchart of full pipeline .....   | 34 |
| Figure 8, loading the subjectivity and sentiment classifiers .....                             | 35 |
| Figure 9, 'Sentiment' function used in final pipeline .....                                    | 36 |
| Figure 10, 'Subjectivity' function used in final pipeline .....                                | 36 |
| Figure 11, conditional statement used to filter tweets in real time .....                      | 36 |
| Figure 12, log function used to save sentiment averages periodically .....                     | 37 |
| Figure 13, average sentiment towards the brand 'playstation' plotted at 5 minute intervals.... | 38 |

### 3 Abstract

The social media platform ‘Twitter’ has been steadily growing in popularity since being founded over a decade ago in 2006. With over 330 million active monthly users, and 500 million tweets being sent every day [1], it provides a constant stream of user generated data which is openly available through the twitter API. Twitter is a platform on which users freely share their opinions on everything, from newly released products to famous public figures. For this project, I intend to analyse this data source as a means of gauging user’s sentiments on different brands, and tracking how they change over time. This will be done by collecting, hand-labelling, and pre-processing tweets before using them to train a range of logistic regression and SVM machine learning algorithms. The most effective machine learning model will then be utilized to build a program capable of recording and tracking twitter sentiment for any brand in real time.

### 4 Acknowledgments

I would like to thank my supervisor Jose for his continued support and advice throughout the project.

### 5 Introduction

The phrase ‘social media’ has become omnipresent in everyday life. More users than ever are sharing their thoughts, opinions, and beliefs online, anywhere, and in a multitude of ways. Statista predicts that there will be a steady increase in the number of social media users [2] for

many years to come. This makes social media presence a crucial aspect of brand awareness for virtually every company in 2020.

Twitter is a unique social media platform, which incorporates the concept of ‘micro-blogging’. Its users make short, frequent posts of no more than 280 characters in length. These posts are commonly opinionated, and can be directed towards a certain user, or categorized with a ‘hashtag’. For this reason, there is a wealth of subjective data which is often conveniently marked with keywords, and with a clear target, being uploaded to the site.

If a company is getting mentioned, specifically mentioned in a positive light, this can have great implications for the success of the brand. If the sentiment with which Twitter Users mention a brand could be classified in an automated manner, then valuable information could be extracted, which a company could use as a method of measuring reception to new product releases, announcements, and decisions.

This leads to the question, how could the subjective opinions directed towards a company or brand on twitter be collected and classified automatically? Further breakdown of the problem leads to two sub problems. How can subjective tweets containing opinions be identified and differentiated from those that don’t, and how can the sentiment of the subjective tweets then be classified?

The solution proposed in this project is machine learning.

Machine learning describes a set of techniques in which a system can constantly learn and improve given experience. The phrase covers a broad spectrum of techniques, from simple mathematical models to fully fledged neural networks. For this project, two suitable algorithms were identified and tested thoroughly, logistic regression and support vector machine, in order to build the most capable model possible. Two models will be trained, one capable of classifying the subjectivity of a tweet, and one which classifies sentiment.

Once capable models have been trained, their use will then be demonstrated by implementing them into a system which is able to stream tweets directly from twitter, filter out subjective tweets, and then classify them in real time.

Once this is achieved, a demonstration of the use of the results of this will be implemented, in which average sentiment is plotted periodically on an animated graph.

The final implementation of this project intends to be one which would be invaluable to any company that wishes to gain insight into customers perception towards them.

## 6 Background

### 6.1 Twitter API

#### 6.1.1 Twitter API overview

The twitter API was used to build a raw dataset of tweets, which would then be labelled and pre-processed for machine learning. In order to access the Twitter API, a Twitter ‘developer account’ is required. Once an account is created, authentication credentials are generated which can be used to stream data from Twitter without the need to manually access the website. The 4 credentials generated are:

- ‘Access token’
- ‘Access token secret’
- ‘Consumer key’
- ‘Consumer secret’

Using these, the API can be interacted with in a variety of different ways, with tools available for both developers and non-developers. For the purpose of the project, it was important to compare some of these options.

#### 6.1.2 Tweepy

Tweepy is a python library designed to access the twitter API for simple automated tasks. It can be used to establish a connection with Twitter, and stream data. Tweets gathered using Tweepy are in JavaScript Object Notation, or ‘JSON’ format. JSON is a file format in which data is represented in a human-readable object with attribute-value pairs and arrays. A Tweet in Json format can have over 150 attributes, in addition to the text itself. Some of the attributes a Tweet object may have include:

- “*created\_at*”, which value will be the exact date and time the Tweet was posted.
- “*id\_str*”, which value will be a string which contains a unique number which can be used to identify the Tweet.
- “*text*”, which contains the actual contents of the tweet.

- “*extended\_tweet*”, which contains additional attributes for tweets over 140 characters in length.

These are just some examples of data included in a Tweet object. For the purpose of collecting the dataset for the project, the fields of “*text*”, “*created\_at*”, “*extended\_tweet*” are of greatest importance.

### 6.1.3 Other tools

‘Tweepy’ is a simple developer-focussed tool for streaming Twitter data. In addition to Tweepy, there are many other tools which can be used for the same purpose. These include:

- IFTT, ‘If this then that’. This is a platform which allows Twitter data to be streamed with no technical knowledge, through a GUI. [2]
- ‘Zapier’. A platform which allows different webapps to be collected seamlessly, without the need for technical knowledge. Using this platform Tweets can be easily collected and collated. [3]
- ‘Tweet download’. This is a tool which allows brands or individuals to download all the Tweets associated with their own account. [4]

Although these tools have the same functionality as ‘Tweepy’, they are more consumer-based, and do not offer as simplistic a solution. It is for this reason that Tweepy was deemed the most suitable for the task at hand.

## 6.2 Sentiment analysis

### 6.2.1 Sentiment analysis overview

Sentiment analysis is used to gain an understanding of emotions, opinions, and attitudes present in a piece of text. The sentiment of text can be classified automatically using a range of techniques, including machine learning algorithms and natural language processing.

### 6.2.2 Sentiment definition

A sentiment is a subjective expression which describes an individuals’ feelings towards a particular subject or topic. For the purpose of this project, the expressions are tweets, the subject being the brands or companies they represent. What differentiates sentiment from a simple emotion is that sentiment is directed towards a specific target, whilst emotion describes a physiological response. [5]



Sentiment polarity describes the orientation of sentiment for a piece of text. Sentiment polarity can be either positive, negative, or neutral.

### 6.2.3 Sentiment analysis goals

The task of sentiment analysis can be broken down into two sub-tasks: subjectivity classification and sentiment polarity classification.

Subjectivity classification allows opinions to be extracted from a larger dataset. A subjective sentence is defined as being ‘based on or influenced by personal feelings, tastes, and opinions’ [6]. This makes subjective phrases valuable for sentiment classification. Alternatively, objectivity is defined as ‘not influenced by personal feelings or opinions in representing facts’, meaning they have no value for sentiment analysis.

Polarity classification refers to the classification of subjective opinions into either positive, negative, or neutral sentiment.

### 6.2.4 Applications of sentiment analysis

Sentiment analysis has a wide range of use cases and can be invaluable to brands wanting to gain insight into the views of their customers.

One application of sentiment analysis for brands is the analysis of the perception of customers towards them. By parsing social media data, an idea can be gained of what customers are saying in real time. This knowledge can be utilized to track responses to new product releases, brand announcements, and the effectiveness of any advertising campaigns run by a brand. By acknowledging the causes of negative sentiment, a brand can adapt its techniques and marketing strategies to improve their status. This use case is most relevant to the current project.

Another widely recognized application of sentiment analysis is in dealing with customer complaints and enquiries. If queries can be passed and then classified by emotional tone, then they can be addressed in order of urgency, instead of on a first-come-first-served basis. This will lead to an improvement in overall customer satisfaction.

## 6.3 Machine learning

### 6.3.1 Machine learning overview

Machine learning is a variation of artificial intelligence that provides a system with the ability to continually learn and improve given experience. Machine learning algorithms take input data and use it to learn for themselves, without human intervention.

In classical programming, the user provides a set of rules and input data and will receive an answer based on these. Machine learning systems differ from this because the rules are not provided by the user. Instead, data is input alongside the expected answers, to generate a set of rules which can then be applied to new data.

### 6.3.2 Supervised vs unsupervised approaches

In supervised machine learning, input data is already ‘labelled’. This means that items in the dataset have been annotated with the correct answers. The data and annotations are then used by the system to generate a predictive model which can label unseen data.

In unsupervised learning, the input data is unlabeled. The system separates the data into categories based on its features without being taught by a human user. It is effective for finding unknown patterns in data, and no manual annotation is required.

### 6.3.3 Classification vs regression

Classification and regression are two types of supervised machine learning system. They differ in the way data is labelled.

In classification, data is mapped to discrete labels. For example, for the task of sentiment analysis, labels could be ‘positive’, ‘negative’, or ‘neutral’. In regression, labels are continuous. As such, regression models are usually utilized when predicting quantities or dimensions. [7] For the purpose of this project, only classification models are used.

### 6.3.4 Fitting

For a predictive model to be effective, it is important that it is neither overfitted nor underfitted. ‘Fitting’ refers to the degree of flexibility of the model.

When examining the fitting of a model, two properties should be considered: variance and bias. Variance refers to how dependent a model is on the data it was trained with. If a model has low variance, then it can be thought of as paying less attention to the training data compared to a model with high variance. Bias refers to simplifying assumptions made by the model that make

the target function easier to learn. A model with high bias has made more assumptions about the target function than one with low. [8]

Ideally, a model would have low variance and low bias. However, in practice, any predictive model will have to have a tradeoff between these. If a model is overfitted, it is said to have low bias and high variance. This means that it fits the training data very closely, and its performance on unseen data may vary.

In turn, an underfitted model has high bias and low variance. This means that it does not take much account of the training data and is unlikely to be a good approximation of the target function.

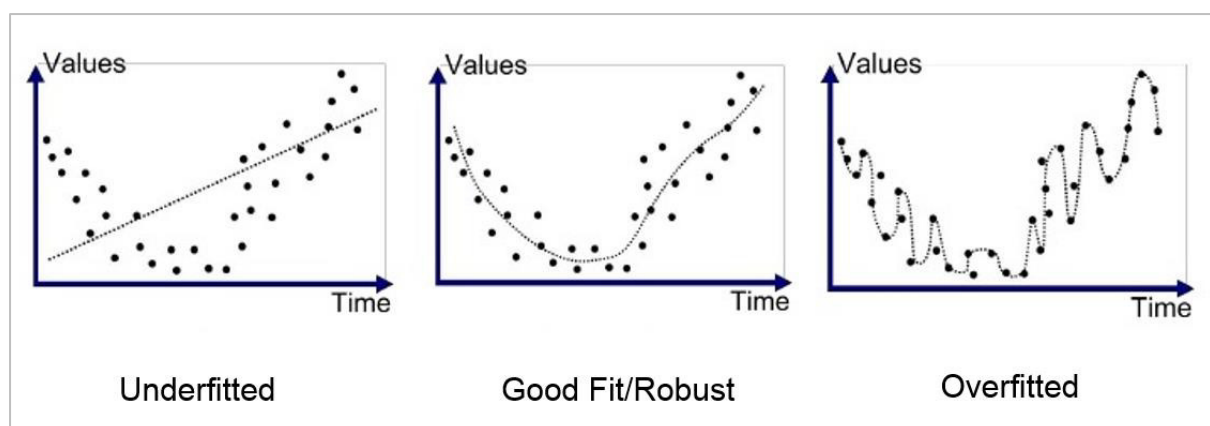


Figure 1, visual representation of the concept of fitting [9]

### 6.3.5 Features

In many cases, data cannot be fed into the model directly. It must first be transformed into a feature vector, an n-dimensional vector which acts as numerical representation of the data.

‘Dimensionality’ is a phrase used to describe the number of features in a dataset. If the number of features in a dataset becomes very large in comparison to the number of observations (high dimensionality), then certain machine learning algorithms can suffer from the ‘curse of dimensionality’. [10] This reduces the system’s ability to train an effective model.

There are two methods for reducing the dimensionality of data: feature selection and feature extraction. The aim of both techniques is to reduce the number of features per observation or label.

Feature selection is the process of selecting a subset of significant features from the entire pool. By only selecting certain features, the system can train a model more quickly and with fewer resources. Selecting features is also beneficial because it reduces overfitting for the model. [11]

A variance threshold is used as a filter to disregard features for which there is little change between observations. It is a method of feature selection for reducing the dimensionality of a dataset without risking losing meaning. Variance thresholds need to be set manually, meaning they rely on human intuition and there is the possibility for error. It is also unlikely to reduce the dimensionality to a sufficient level by itself. [10]

Feature extraction is a similar concept to feature selection, however instead of choosing from existing features, it creates new ones.

### 6.3.6 Training, validation, and test sets

Before it can be used to build a predictive model, a dataset must first be split into training, validation, and test sets. The way the whole dataset is split between these subsets varies, but generally the training set will be the largest.

The training set contains the data and labels used to train the model itself, whilst the validation set is used to change the parameters of the model. The test set is then used to evaluate performance.

### 6.3.7 Cross validation

Cross validation is a technique for evaluating machine learning models. In cross validation, a series of models are trained on different subsets of the data, then tested using the complementary set. These models are then compared to test for overfitting of the data.

k-fold cross validation is a technique in which k subsets of the dataset are taken. These subsets are referred to as folds. The model is then trained on k-1 of the subsets, with the final subset

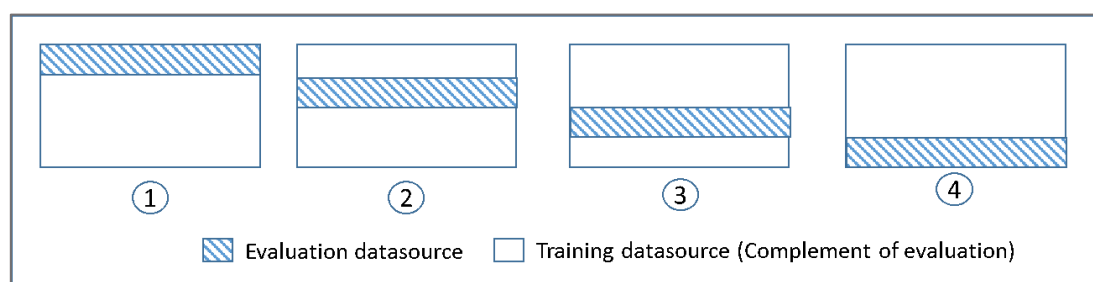


Figure 2, visual representation of k-fold cross validation

used for evaluation. This occurs k times, with a different subset being used as a test set each iteration. [12]

The final result for cross-validation is the average performance of all of the folds. Cross-validation is particularly effective when working with small datasets, or when the distribution of the dataset is skewed.

### 6.3.8 Evaluation measures

There are several different measures used to evaluate the effectiveness of models generated through machine learning. When effectiveness is measured, it is always against a set of already labelled data, often referred to as the gold standard.

Accuracy is a measure of evaluation which describes the number of correctly classified answers compared to the gold standard.

A confusion matrix can be utilized to visualize how a binary classification model performs. In a confusion matrix, expected values can be compared to predicted values:

Table 1, an example of a confusion matrix for a simple pos/neg classifier

| N=235         | Predicted<br>pos | Predicted<br>neg |
|---------------|------------------|------------------|
| Actual<br>pos | 100              | 34               |
| Actual<br>neg | 22               | 79               |

A confusion matrix displays the number of ‘true positive’, ‘false positive’, ‘true negative’, and ‘false negative’ predictions made by a classifier. These are all measures which together provide insight into the effectiveness of a model:

- A ‘true positive’ (TP) describes when a classifier predicts a positive, and it was correct in its classification.
- A ‘false positive’ (FP) describes when a classifier incorrectly predicts a positive.
- A ‘true negative’ (TN) describes when a classifier predicts a negative result correctly.
- A ‘false negative’ (FN) describes when a classifier incorrectly predicts a negative.

This can be applied to any binary classification problem. Using these, other measures of evaluation can be derived:

$$Precision = \frac{TP}{TP + FP}$$

Precision describes the percentage of positive predictions which the model classifies correctly.

$$Recall = \frac{TP}{TP + FN}$$

Recall describes the percentage of positive predictions that are predicted as positive.

$$Specificity = \frac{TN}{TN + FP}$$

Specificity describes the percentage of negative cases that are predicted as negative.

Using precision and recall, an ‘F1 score’ can be calculated. The F1 score describes the harmonic mean between precision and recall. [11] If precision and recall are perfect, the F1 score will be equal to 1. It is defined as:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

For the purposes of this project, classification is not binary. Evaluation measures for multi-class classifiers take the same concepts as binary classification but then calculate either a micro-average, or a macro-average.

To take a macro-average, the metric is computed independently for each class before an average score is taken. In this way, all classes are treated equally. A micro-average is taken by aggregating the contributions of all classes together before calculating a metric.

### 6.3.9 Logistic regression

Logistic regression is a statistical technique used in machine learning, which is highly effective for binary classification.

Logistic regression gets its name for the function that it centres around, the logistic function. This is a function which takes any real number and maps it to a number between 0 and 1. It is given by the equation:

$$y = \frac{1}{1 + e^{-x}}$$

When plotted, this equation forms an S shaped curve. Logistic regression uses an equation based on the logistic function, in which the input value is combined linearly with different coefficient values to produce an output:

$$y = \frac{e^{b_0 + b_1 \cdot x}}{(1 + e^{b_0 + b_1 \cdot x})}$$

Where  $y$  is the output value,  $b_0$  is the intercept term (the point at which the regression line meets the vertical axis),  $b_1$  is a coefficient, and  $x$  is the input value. Logistic regression produces a binary output, e.g.  $Y = 0$  or  $1$ . [13] When the function is used in a computational environment, the  $b$  values (or coefficients) are stored to save the model.

Directly, the logistic regression function returns a probability value for the answer, given the input. For example, when considering sentiment analysis of a string, logistic regression provides an estimate that the probability will be positive:

$$P(\textit{Sentiment} = \textit{Positive} | \textit{String})$$

This probability can then be used to classify a string, e.g. if  $P > 0.5$  then string has positive sentiment, and if  $p < 0.5$  then it is classed as negative.

It is clear that the output of the logistic regression function is dependent on the beta ( $b$ ) values. These values are estimated from the training data. Often, the ‘Maximum likelihood estimation’ algorithm is used to determine this.

#### 6.3.10 Optimising logistic regression

When creating a model using logistic regression, there are optimisations to the data which can be made to improve performance. These include:

- Noise reduction. Logistic regression assumes that all labels in the training set are without error. Before using the dataset, this should be checked. If there is noise in the dataset, the model will fit to this.
- Removal of correlated inputs. If there is a large number of very similar data inputs, then there is a risk that the model will be overfitted.
- Binary classification. Logistic regression works when there are only two possible output variables.

#### 6.3.11 Support Vector Machine

The Support Vector Machine (SVM) is another machine learning algorithm. Although it can be used for regression, generally it is used for classification. SVM is a favourable algorithm to many because it provides significant accuracy for less computational power.

The objective of a support vector machine is to find a hyperplane in N-dimensional space, where N is the number of features, that distinctly classifies all data points.

Separating data points can usually be done with an assortment of different hyperplanes. To maximise the effectiveness of classification, the hyperplane with the greatest margin should be picked. The margin describes distance between data points of different classes.

Hyperplanes can be viewed like a barrier between two groups of datapoints. Either side of this barrier will be the two classes. The dimension of the hyperplane will always be one dimension greater than the number of features in the dataset. When the classes are not linearly separable, SVM uses external functions to map the original data to a new feature space where the data becomes separable. [14]

A support vector is a datapoint which lies close to the hyperplane, and has an influence over its orientation and position. The support vectors can therefore be utilised to maximise the margin of the classifier. [15]

Despite its popularity, SVM has been criticised for underperforming on raw data, and requiring an expert level of feature extraction to classify effectively.

## 6.4 Pre-processing

### 6.4.1 Overview

Pre-processing describes transformations applied to data before it is fed into a machine-learning algorithm. It involves a collection of techniques which convert raw data into a clean and optimised dataset useable by a machine learning system. [16]

### 6.4.2 Stemming and lemmatization

Stemming and lemmatization are two techniques for pre-processing text. It is a common feature of many written languages to have words which are derived from other words. A language in which words that are derived from others are used differently depending on their context is known as inflected language. [17] Different languages have different levels of inflection.

Inflected words will have a 'root form'. The normalisation of text describes mapping inflected words to their root forms, e.g.:

Saying, says, said → Say

Trying, tried, tries → Try



Stemming and lemmatization are two methods of normalising words to their root form. Algorithms for stemming and lemmatization have been developed in computer science since the 1960s. What differentiates stemming from lemmatization is that the root word extracted when stemming, the ‘stem’, may not be a real word, whilst the ‘lemma’ extracted using lemmatization will always be.

Lemmatization is more suitable for this project, as using this word are normalised properly to their actual root. Lemmatization is useful as it reduces the vocabulary of the model, making it more streamlined and efficient.

### 6.4.3 Standardisation

Standardisation is a common requirement for many machine learning classifiers. If vectorized are not close to standard normally distributed data (Gaussian with zero mean and unit variance) then it can have a negative impact on the classifier. To account for this, ‘sklearn’ has a built-in ‘standardize’ function.

## 6.5 Additional python libraries

### 6.5.1 Scikit-learn

Also referred to as ‘sklearn’, this is a python library which contains a range of out-of-the-box machine learning tools.

It contains implementations of both SVM and Logistic Regression machine learning algorithms, alongside pre-processing and vectorisation functions. Vectorization functions in sklearn are used to transform raw text into a format useable by a machine learning classifier. In this project, two sklearn vectorizer will be used, ‘Tfidf’ and ‘Count’.

Vectorization in Sklearn is done in a number of steps. One notable step is pre-processing. This can be manually overwritten with a customised function, as will be done later in the project.

The tools in sklearn are simple efficient, and easily accessible. It is based on the NumPy, SciPy, and matplotlib python libraries. It is also open source and entirely free to use. [18]

### 6.5.2 NLTK

NLTK is another python library, which contains a range of useful tools for the computational processing of human-language data. It provides access to over 50 corpora and lexical resources such as wordnet, alongside several text-processing libraries for tasks such as stemming and lemmatization.

For the project, the main function of NLTK which will be utilised is wordnet's 'lemmatizer'. This is a function which takes an input string, and reduces it to its lemmatized root form. [19]

### 6.5.3 matplotlib

Matplotlib is a commonly used python library for creating data visualisations in python. The library includes tools for both static and dynamic visualisations.

In regard to the project, the ability to produce animated graphs with matplotlib makes it invaluable when visualising sentiment data in real time

### 6.5.4 Pickle

Pickle is a python module which can be utilised to serialise and de-serialise objects. When an object hierarchy is 'pickled', it is converted into a byte stream which can then be 'unpickled' to inverse the operation and return the object.

Pickle can be used to serialise and deserialize machine-learning models, so they can be easily saved and then implemented elsewhere.

## 7 Data collection process

### 7.1 Identifying brands

The aim of the project is to build a classifier able to determine consumer's opinions on brands, based on tweets directed towards them. When collecting data, it was important that Tweets were therefore directed at a brand. For training and testing of the model, tweets from a sample of five brands were chosen. These brands were:

- Playstation (@Playstation)
- Netflix (@Netflix)
- Android (@Android)
- SpaceX (@SpaceX)
- Starbucks(@Starbucks)

This was because these brands represent the five of the most followed companies on the platform. It was important that the brands chosen for training had a large presence on twitter, so that a large enough volume of data could be collected. It is important to emphasize that although these brands were chosen for training purposes, the final model would still be generalised to fit any brand, due to specific pre-processing steps carried out later on in the project.

## 7.2 Tweepy

After identifying the five initial brands which would be used, a sample of over 300 Tweets were collected for each.

The Python library ‘Tweepy’ was used to stream Tweets which contain each of the chosen keywords and save them to a CSV file. ‘Tweepy’ is a commonly used library which allows the Twitter streaming API to be accessed given valid credentials. When streaming data with Tweepy, parameters can be set to stream specific tweets. The filters set when collecting Tweets for the brands were:

- A keyword filter, which allowed only Tweets associated with the brand to be collected. Multiple keywords could be applied at the same time in an array. For each brand, the keyword would be equivalent to the brand name.
- A filter which excludes any tweets with links. This was done by discarding any Tweets containing the characters ‘http’. This was used because link-containing Tweets are unlikely to contain useful sentiment.
- A filter which excludes Tweets with too many ‘@s’ ‘hashtags’ or ‘retweets’. This was useful as Tweets with an excessive number of these are more likely to be produced automated bots, therefore unimportant for analysing sentiment.

## 7.3 Subjective vs objective Tweets

On first observation of the gathered tweets, it became clear that many contained no explicit sentiment. This was due to the objective nature of many of the tweets. After collection of the first sample of tweets, over 30% were found to be objective in nature, and therefore of little use for the task of sentiment analysis.

Because of this, before further collection of data, a method for classifying subjective and objective text was implemented.

To implement this, a machine learning approach was taken. The goal was to create a classifier capable of differentiating objective sentences from subjective ones. To achieve this goal, a suitable dataset was first required.

The most effective pre-existing dataset for this task easily available was a dataset consisting of short snippets of both movie reviews, and movie plot synopses. Naturally, the phrases extracted from movie reviews were entirely subjective, whilst those taken from plot synopses were objective.

The data was used to train a machine learning model which utilises Logistic Regression. Objective sentences were given the classification label 'O', whilst subjective sentences 'S'. The text was left intact without removing stop words in pre-processing, because when classifying subjectivity, the presence of common stop words, such as 'I' or 'They', can provide valuable information.

Once trained, the model was used to read the CSV containing all the streamed tweets, and write any subjective tweets to a new file. When written to the new file, all extra attributes included in the original file, such as 'created\_at', were also recorded.

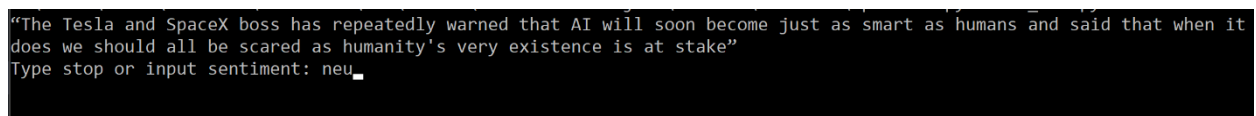
## 7.4 Method for annotating data

After collecting tweets and filtering out opinions, the remaining tweets could be hand-labelled to build a dataset useful for machine learning. When labelling the tweets, three labels were used: 'pos', 'neg', and 'neu'. These represented tweets with a positive sentiment, tweets with negative sentiment, and tweets with a neutral sentiment, respectively.

To label the tweets, an interactive python script, 'read\_CSV.py' was written. The script functioned as follows:

- 1) Read in CSV containing unlabelled tweets, producing a 2-dimensional array of tweets.
- 2) For each tweet in array, display tweet.
- 3) After displaying the tweet, wait for user input.
- 4) Check user input, if it is a valid sentiment (string of either 'pos', 'neg', or 'neu'), then append to the tweet
- 5) When the end of the tweets is reached, or the user enters 'stop', save the array of annotated tweets to a new CSV file.

The annotation process was repeated for each of the five CSV files, containing the tweets for each brand respectively.



```
"The Tesla and SpaceX boss has repeatedly warned that AI will soon become just as smart as humans and said that when it
does we should all be scared as humanity's very existence is at stake"
Type stop or input sentiment: neu_
```

Figure 3, interface produced by 'Read\_CSV.py', through which tweets were annotated

## 7.5 Criteria for annotating data

In text, sentiment can be both explicit and implicit. Explicit sentiment describes instances in which opinion is expressed directly, while implicit describes when an opinion is simply

implied. When labelling the dataset, tweets with explicit sentiment generally more straightforward to classify, whilst those in which sentiment was implicit more challenging.

When labelling sentiment, it was important that a clear set of guidelines was followed, to ensure consistency in the final labelled dataset. The following guidelines were referenced when hand-labelling. All example tweets have been taken from the final dataset used.

### 7.5.1 Positive sentiment

The following points were considered when classifying a tweet as being positive in sentiment.

- There is description of a positive emotion, e.g. ‘love’, ‘enjoy’, ‘like’ targeted at the brand or product associated with the brand. An example from the dataset:

"My former boss got me hooked onto Selling Sunset on @netflix ! Love the outfits, drama, and luxe properties! 🍷🌴"

- The tweet describes a positive experience with the brand, or a product associated with the brand. E.g. ‘customer service was helpful’. An example tweet from the dataset would be:

“to the woman at my drive thru in starbucks u made my day by being so sweet to me”

- The tweet describes a user engaging positively with a product or service provided by the brand. An example from the dataset is this:

“I got my @Starbucks color changing reusable cold cups 🍷”

### 7.5.2 Negative sentiment

The points considered when classifying a tweet as negative were as follows:

- There is a clear negative emotion directed towards the brand or product associated with the brand. For example:

*“Lol.. Nd yet android is trash. Smh”*

- The tweet describes a negative experience or fault with a product/service provided by the brand, e.g.:

*“my twitter for android is broken so i'm using my pc 🙄))s”*

- The tweet describes a lack of engagement with a brand or product. This means they are stating they do not use a brand, or would not purchase a product or service from them. An example of a tweet for which this applies would be:

*“You have a PlayStation? Oop-”*

### 7.5.3 Neutral sentiment

Finally, a tweet was labelled as neutral if it could not be defined as either positive or negative. The following points were considered when labelling a tweet as neutral.

- Objective tweets have a neutral sentiment. In some cases, this would be a result of the inaccuracy of the subjectivity filter, resulting in an objective phrase being in the dataset. An example of an objective tweet in the final dataset, with a neutral sentiment:

*“The Tesla and SpaceX boss has repeatedly warned that AI will soon become just as smart as humans and said that when it does we should all be scared as humanity's very existence is at stake”*

- In other cases, sentiment was marked as neutral if there was a combination of positive and negative sentiment in the tweet, with no clear polarity either way. For example:

*“This is really good but I would get annoyed so fast online after the 2nd time 🤔🤔”*

- Finally, there were cases in which the tweet simply represented a question directed towards the brand or about the brand, with no discernible sentiment. These were classified as neutral. An example:

*“@Irocket Hey Tom, you got any spacex stock you want to sell or know of anyone that is will to sell?”*

## 7.6 Increasing dataset size

After collection and annotation of data, a total of 1245 tweets existed in the dataset. Out of these:

- 529 were labelled positive.
- 243 labelled negative.
- 373 labelled neutral.

When training a classifier, it is beneficial to have a larger dataset. Furthermore, it is important that classes in a dataset are well balanced. Imbalanced datasets can lead to models appearing to perform excellently, with high accuracy, when in actual fact this is simply the result of an imbalanced class distribution. This is known as the accuracy paradox. [20]

In order to avoid this, the hand-labelled dataset was combined with another dataset of tweets before training.

The dataset used was the ‘sentiment140’ dataset. [21] This dataset consists of over 10,000 tweets with labelled sentiment. Sentiment is labelled either, ‘4’ for positive, ‘2’ for neutral, or ‘0’ for negative.

In addition to the hand labelled tweets, 1071 positive, 1227 neutral, and 1257 negative tweets from the sentiment 140 dataset were included in the final dataset. This brought the total number of tweets to:

- 1,600 positive
- 1,600 neutral
- 1,600 negative

The overall dataset now consisted of 4,800 labelled tweets.

## 8 Implementation

### 8.1 Pre-processing

Before training the classifiers using the annotated dataset of tweets, it was first important to implement a method with which raw text could be pre-processed. A customized pre processing function was written, which would then be compared to the built in preprocessing available in sklearn.

Several steps were taken to pre-process tweets. These were:

- 1) Removal of line breaks, paragraph breaks, and extra whitespace between words in the tweet. This was removed as it has little to no impact on the sentiment of the tweet but increases the length and may lead to problems extracting features further along in the process.
- 2) Removal of all punctuation and non-standard characters. Once again, punctuation does not provide any information relating to a tweet's sentiment.
- 3) Removal of any specific mentions of the brand, any of these were replaced simply with the word 'brand'. This improves the generalization of the model. If specific brands are mentioned in the training data, there is a risk the model becomes overfitted to these. Replacing brand names with a generic word will reduce this overfitting. This also reduces the vocabulary of the model, streamlining it.
- 4) Lemmatization of all words, using NLTK and wordnet's lemmatize function. The intention of this is to reduce the vocabulary for the model, and improve generalization.

The aim with all of the above pre-processing steps is to improve the generalizability of the final model. Data in real-world scenarios is often imperfect, and can contain missing values, useless values, and other unnecessary data. By removing this noise, the accuracy of the final model will improve. A custom function was designed mainly because of step 3, in which brand mentions are removed. This is something that the standard preprocessor would not do, which had the potential to negatively effect the generalizability of the model.

The function used to pre-process data was written in python and takes a string as an argument. It returns a new string which has been preprocessed.



```

def pre_process(tweet):
    brands = ['playstation', 'starbucks', 'netflix', 'android', 'spacex']
    valid_chars = 'a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 0 :) :('
    valid_chars.append(' ')
    #Remove unnecessary whitespace and convert to lowercase
    tweet = ' '.join(tweet.lower().split())
    #Remove punctuation and other characters which are not useful
    for char in tweet:
        if char not in valid_chars:
            tweet = tweet.replace(char, '')
    #remove brand names so tweets are more generalised
    for brand in brands:
        tweet = tweet.replace(brand, 'brand')
    words = tweet.split()
    #Lemmatize if Lemmatization is set to L
    for word in words:
        word = lemmatizer.lemmatize(word)
    tweet = ' '.join(words)
    return(tweet)

```

Figure 4, function used to pre-process tweets

## 8.2 Model selection

### 8.2.1 Classifier advantages and disadvantages

‘Sklearn’ provides a toolkit over 10 machine learning algorithms which could have been used for the current project. After comparing these models, the two which were tested further are logistic regression and support vector machine.

Firstly, logistic regression was selected based on the following strengths :

- It is known to be very resource-efficient
- Does not require scaling of data
- Requires little to no tuning
- Can be regularized easily

These advantages, amongst other, make logistic regression algorithms widely used for simple classification problems. Although it is generally used for binary classification, it can be expanded to work on multi-class classification problems, such as the one at-hand.

The support vector machine algorithm was chosen because:

- Known to work well on unstructured or semi-structured data
- Scales well for large datasets
- Good generalization, low risk of overfitting

Both models also have the advantage of having simple implementation in ‘Sklearn’. It was next important to consider any potential disadvantages of the two models. Potential drawbacks of logistic regression algorithms include:

- Inability to solve non-linear problems

Disadvantages of SVM algorithms include:

- Potential for long training time with larger datasets
- Has the potential to be computationally intensive

### 8.2.2 Model requirements

Before testing different classifiers, a list of requirements was set. These requirements describe the accuracy that were required from the model in order for classifications to be meaningful, and the model used further along the pipeline. These requirements were:

- **Must** have a test set accuracy score of over 0.50. A score of 0.50 or over shows the model has at least some ability to classify sentiment with an accuracy greater than if classified at random. If labels were randomly classified, the expected accuracy would be 0.33.
- **Should** have a test set accuracy of over 0.6. This would represent a 81% improvement on random classification and yield meaningful results.

### 8.2.3 Methodology for testing model

To build the most accurate classifier possible, a range of model configurations were trained, their performance measured, and then compared. All models were trained using the ‘sklearn’ python library, using the data collected and annotated previously. Two different machine learning models were tested for each configuration: logistic regression and SVM.

The models were tested, and performance recorded for the following configurations:

- A variety of train/test splits, ranging from 50/50, to 90/10.
- With standard preprocessing.

- With customized preprocessing.
- With and without scaling features.
- With both Tfidf and count vectorizers.

Once tested, the best performing model would then be used in the full pipeline. To test each configuration, a script was used which would take the parameters as input, train the specified model, and output a saved version of the model alongside a ‘.txt’ file which included:

- Accuracy
- Precision
- Recall
- F1 score

As this is a multi-class classifier, the micro, macro, and weighted measures were all recorded for precision, recall, and F1 scores. Although not considered for every configuration, these would be taken into consideration when comparing the most effective logistic regression and SVM models.

In order to make experimentation and data collection as seamless as possible, a python script was written, ‘test.py’ in which a series of variables could be edited in order to build classifiers of different types and with different configurations.

These variables consisted of:

- `model_type`, which could be set to ‘SVM’ in order to train a Support Vector Machine model
- `trainsize`, which was used to set the training/test split for the dataset.
- `preprocess`, which could be set to ‘p’ to train the model using the aforementioned preprocessing function, or ‘np’ in order to train using sklearn’s built-in preprocessing.
- `Scaling`, which when set to ‘s’ will scale the data with sklearn’s standard scaler
- `Vectorizer_type`, which can be set to ‘cv’ to implement a count vectorizer, or ‘tfidf’ to implement a Tfidf vectorizer.

The script would use these variables to build the desired model, before using pickle to serialize and save it, alongside the results.

### 8.2.4 Logistic regression model

The first type of model tested was a logistic regression classifier. To begin testing the model, accuracy scores were taken with standard pre-processing, scaling, using a count vectorizer. Accuracy scores were taken for training/test splits ranging from 50/50 to 90/10. The purpose of this was to observe how the training/test split affects accuracy.

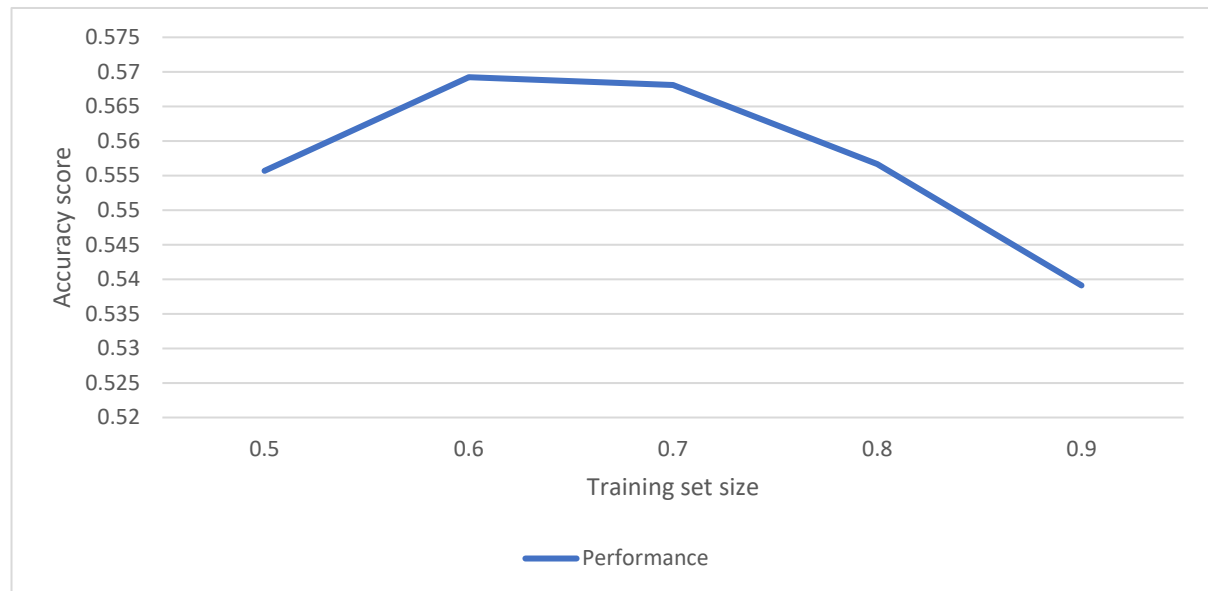


Figure 5, a graph showing how the training set size affected accuracy of the model.

It is evident from this that the accuracy of the logistic regression classifier reaches its maximum at a training size of 0.6. When a training set of larger than 0.7 is used, a decrease in accuracy is observed.

Next, the customized pre-processing function described earlier was compared to the built-in pre-processor implemented by sklearn vectorizers. The results of this can be seen below:

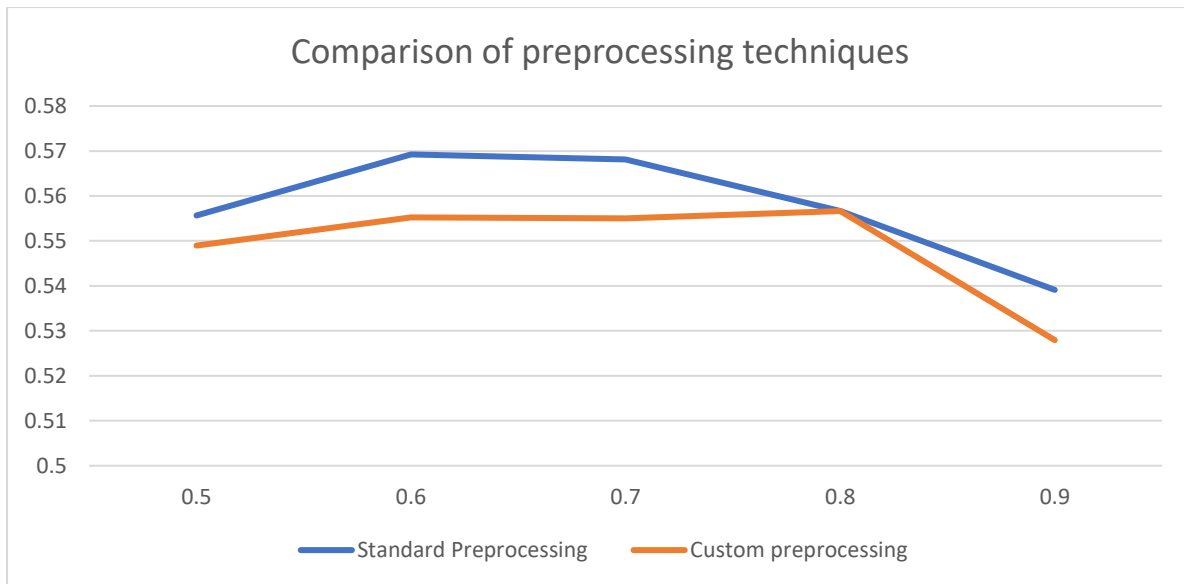


Figure 6, comparison of standard and custom pre-processing techniques

It is evident from these results that the model performs most effectively when sklearn's built-in preprocessing is used. Although it can be noted that the model trained with an 80/20 train test split had an equal accuracy compared to the built-in function.

The next parameter of the model to be tested was the effect of scaling the dataset. Once again this was tested with training/test splits ranging from 0.5 to 0.9.

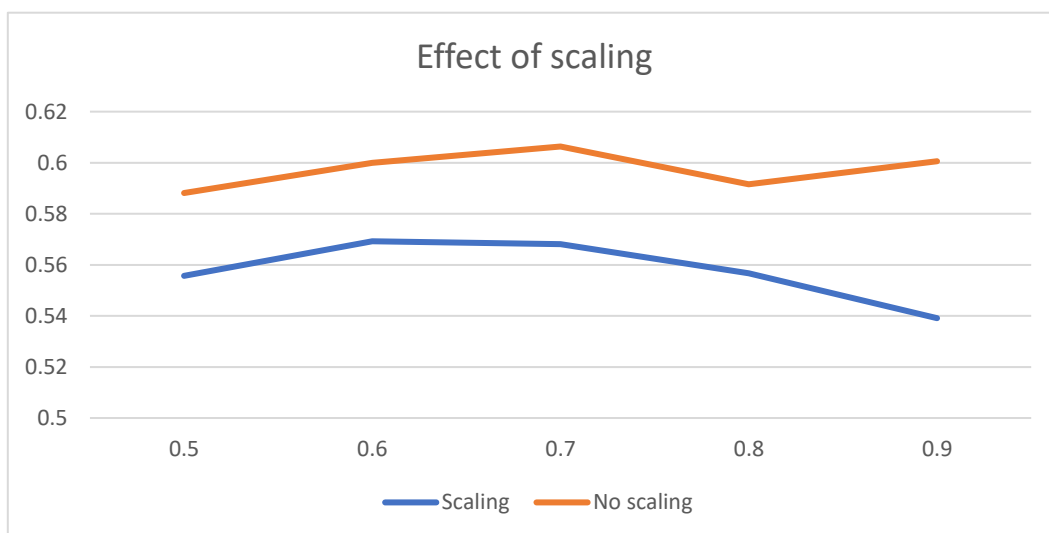
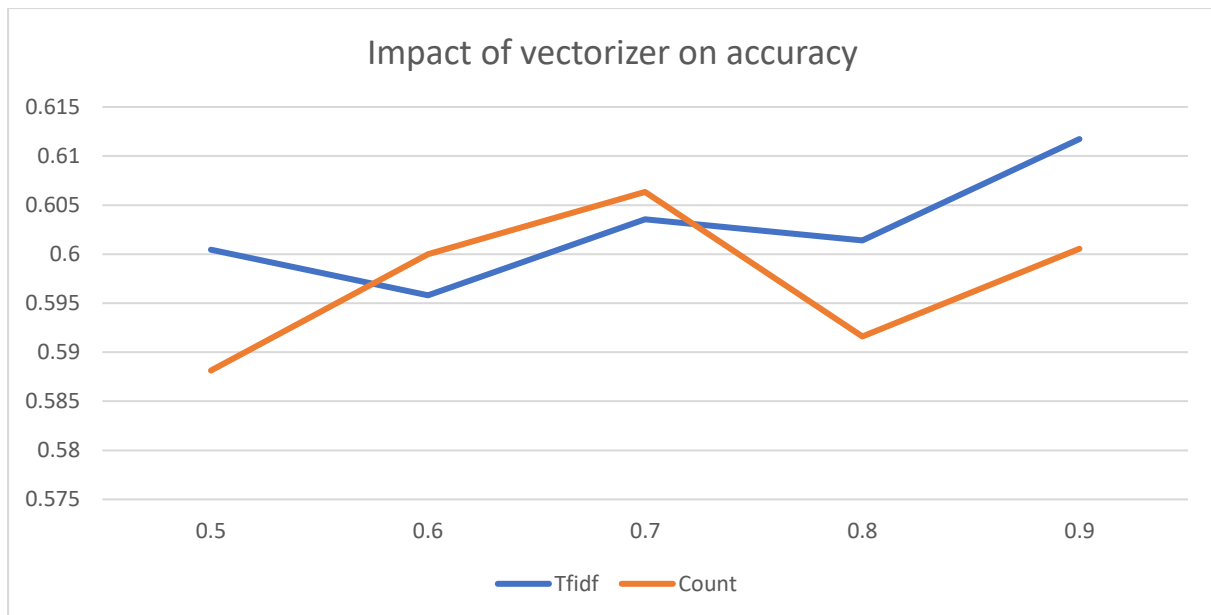


Figure 7, effect of using scaled and unscaled data

The results showed a distinct increase in accuracy when data was left unscaled. This was true for every model trained, from 50/50 train test split, up to 90/10.

The next aspect of the model to be considered was the vectorizer used. The two vectorizers tested with the logistic regression model were 'Tfidf', and 'Count' vectorizers. The results can be observed below:



The outcome of these tests was mixed. Although the count vectorizer appears to yield a more accurate model at 70/30 splits and below, above this Tfidf appears more effective. This indicates that Tfidf performs better when more training data is used.

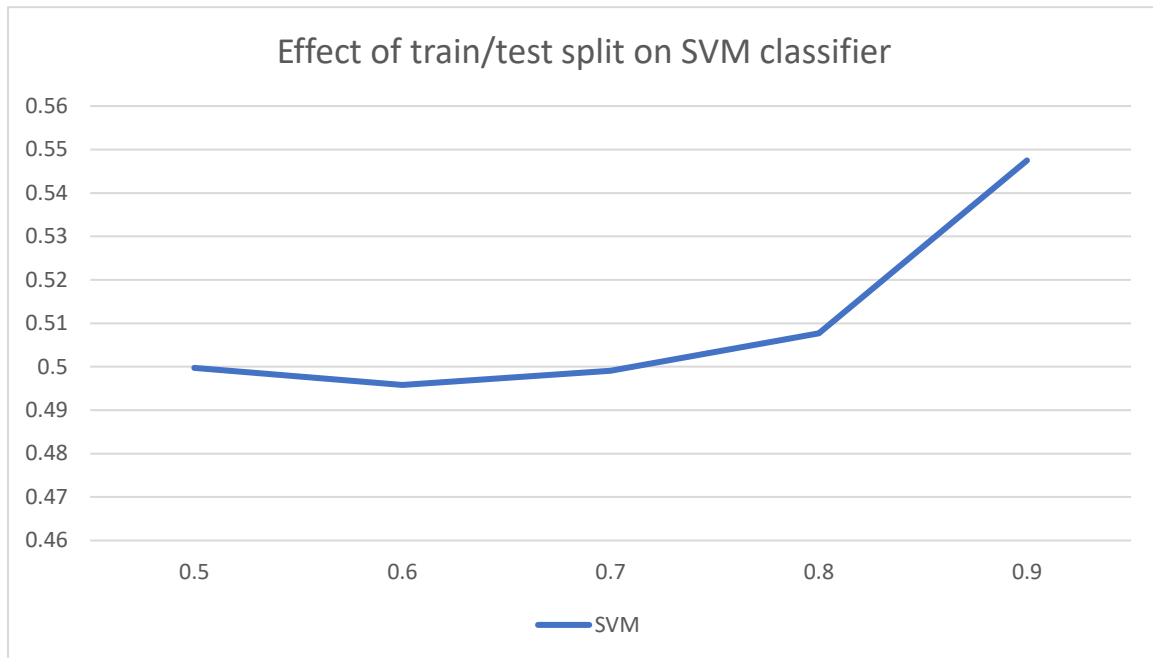
Out of 35 models trained, the Logistic regression classifier with the greatest accuracy had the following configuration:

- 90/10 train test split
- Non-customized pre-processing
- No scaling of data
- 'Tfidf' vectorizer

This classifier has an accuracy of 0.61, rounded to 2 decimal places. This accuracy score met the requirements set out before testing began and would therefore be considered further.

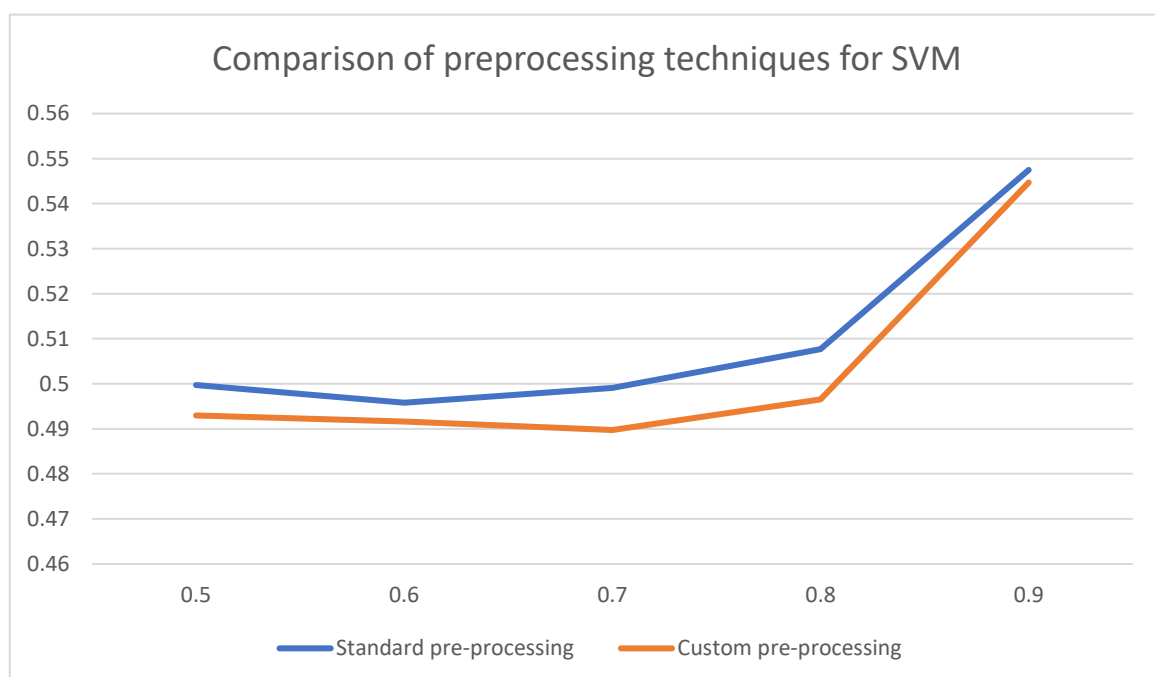
### 8.2.5 SVM model

After completing testing of the Logistic regression classifier, the same configurations were then tested on a Support Vector Machine.

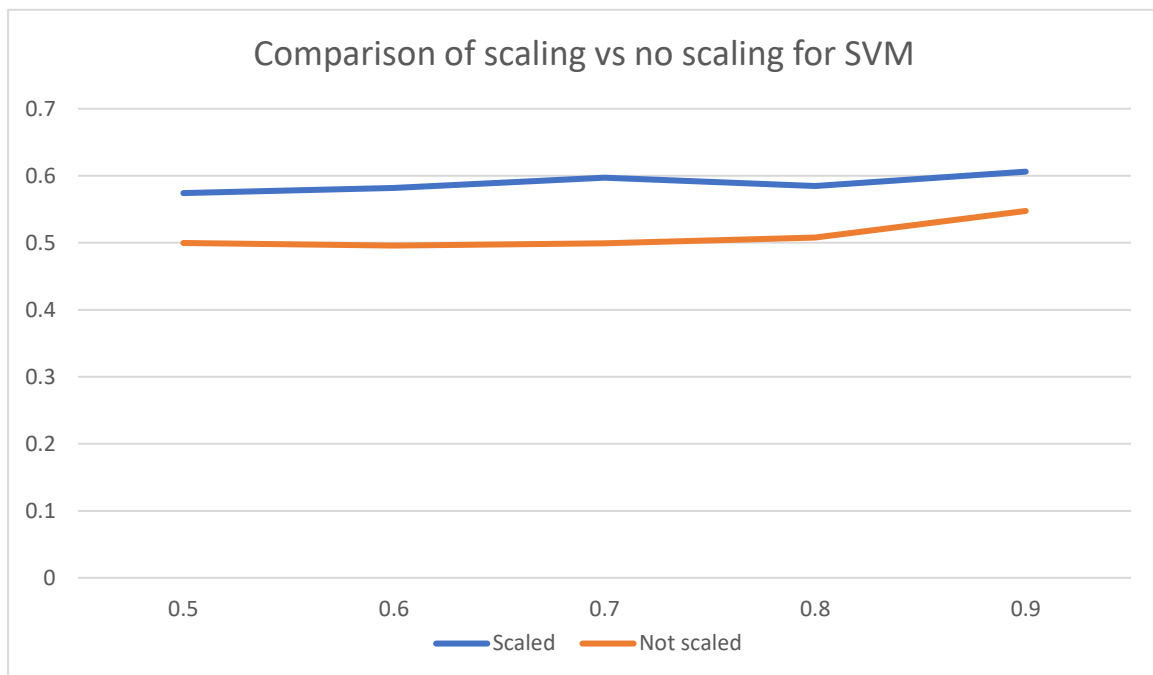


Interestingly, as opposed to Logistic Regression, the SVM's accuracy appears to increase as the size of the training set is increased, and the size of the test set is reduced.

Following on from this observation, the two preprocessing techniques were once more compared.

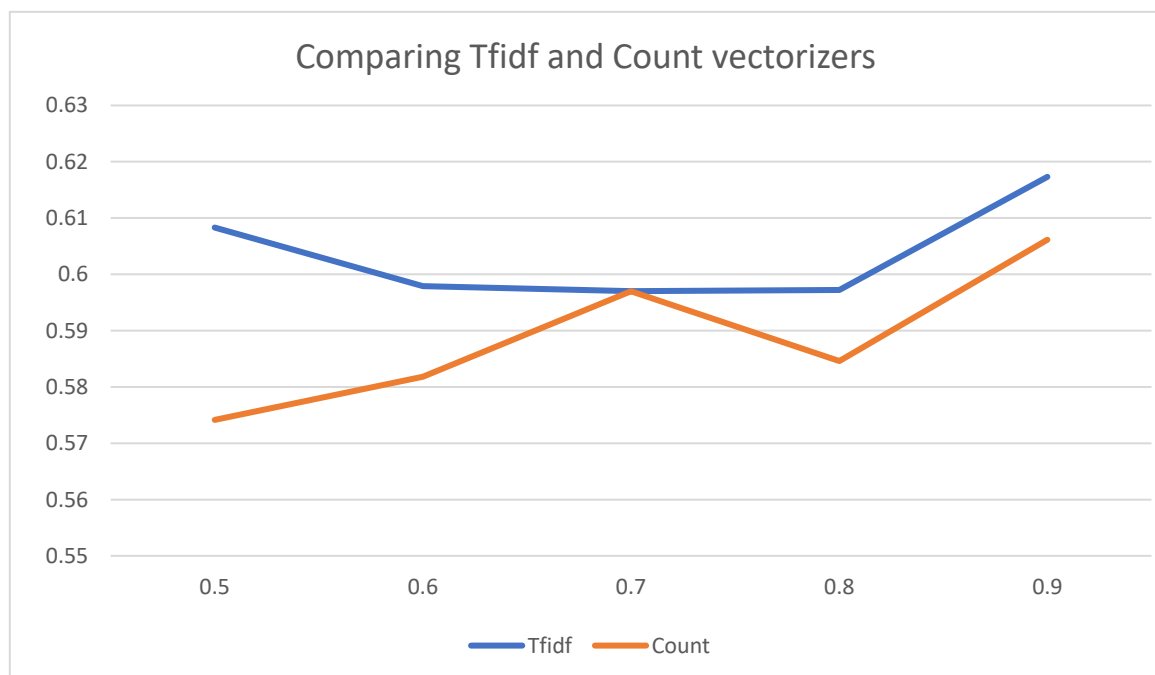


Once again, sklearn's built in preprocessing function outperformed the custom function. In every test case, for each classifier, the model performed more accurately when this function was not used.



For all SVM models trained, when data is left unscaled, the classifier predicts with greater accuracy.

Finally, the effect of each vectorizer was compared.





The results of the comparison show that the ‘Tfidf’ vectorizer outperforms the count vectorizer for every model with which it is trained. This means that once again, the most effective SVM model would utilize ‘Tfidf’ over count Vectorizer.

The highest accuracy achieved with SVM was 0.61 The configuration which achieved this was:

- 50/50 train test split
- ‘Tfidf’ vectorizer
- Standard preprocessing
- Unscaled data

### 8.2.6 Logistic regression vs SVM

After collecting results for a range of different models, the two most accurate models for SVM and Logistic regression, identified in the previous section, were examined further.

Firstly, the other evaluation scores were considered:

|                      | Logistic Regression | Support Vector Machine |
|----------------------|---------------------|------------------------|
| Accuracy             | 0.61                | 0.61                   |
| Precision (weighted) | 0.60                | 0.62                   |
| Recall (weighted)    | 0.61                | 0.57                   |
| F1 Score (weighted)  | 0.58                | 0.54                   |

Although both models had an equal accuracy, SVM boasted marginally higher precision, while Logistic Regression had better Recall and F1 scores.

As it had proved to be the more effective classifier given the dataset, the logistic regression model was chosen to be implemented into the full pipeline.

## 8.3 Full pipeline including live tracking

### 8.3.1 Methodology for final pipeline

After acquiring a final model which maximized performance, it could now be used to collect and visualize sentiment data. A full sentiment analysis pipeline was implemented, beginning with streaming and filtering of twitter data, and producing a live-plotted graph of average sentiment periodically.

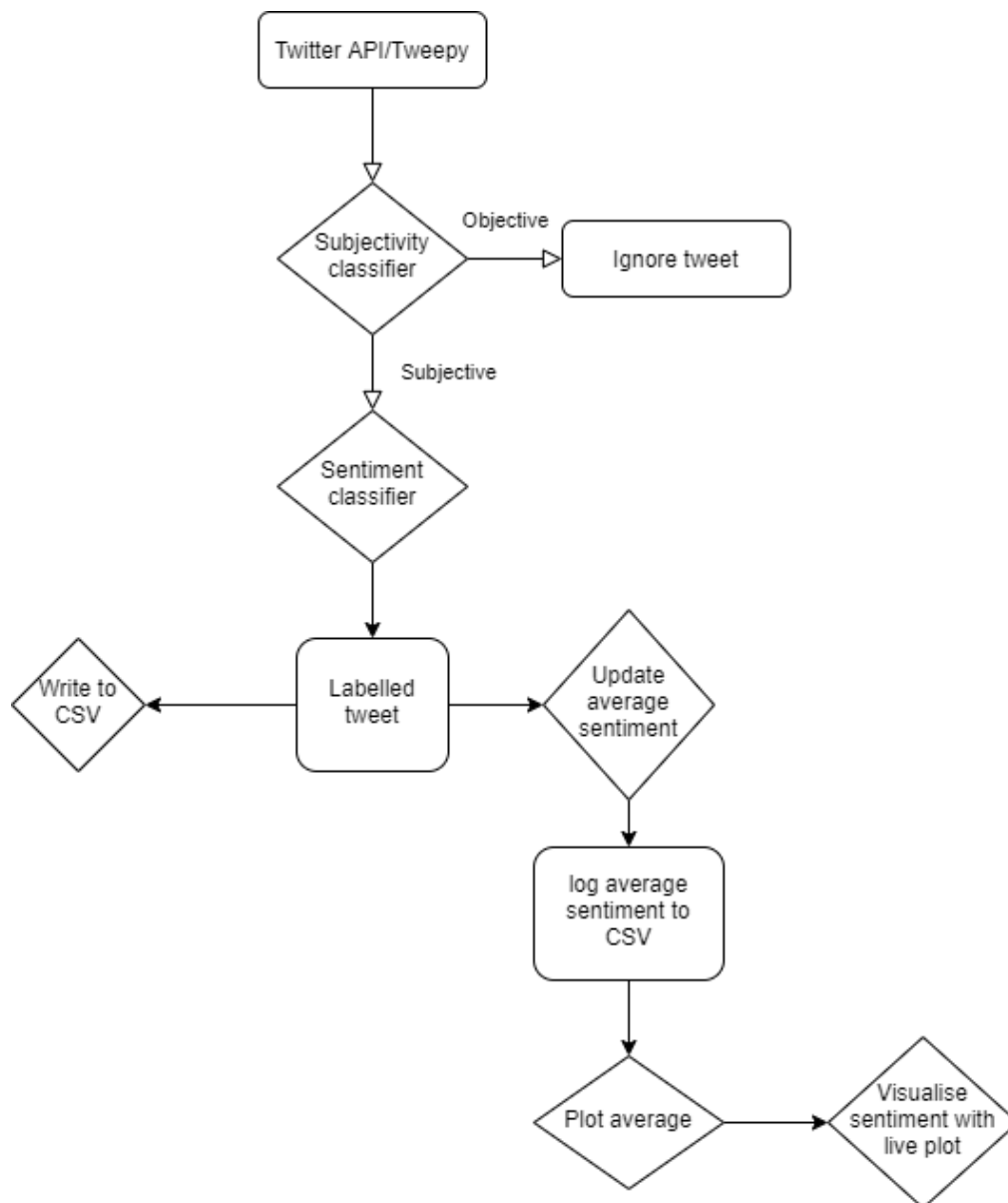


Figure 8, flowchart of full pipeline

Before the pipeline was implemented, a set of requirements was also formulated, so that it could be evaluated in the final stages of the project. These requirements which must be met were as follows:

- The implementation of the pipeline **must** be generalized for any brand. The keyword used to filter tweets should be changeable without having any effect on the result.
- The implementation **must** be flexible and be capable of using any of the classifiers trained and tested previously. This is important as by having this flexibility, the pipeline can be improved easily if a more effective model is built in the future.
- The implementation **must** store a record of average sentiment, as well as the timestamp at which the average is calculated, in such a way this can be plotted using matplotlib.
- The implementation **must** filter and annotate tweets in real time, or close to real time.
- The implementation **must** produce an animated line graph, with a Y axis indicating sentiment, and an X axis indicating time.

The requirements that should be met, but are not compulsory were:

- The graph produced by the implementation **should** be well designed, with labelled axis and appropriate ranges for the X and Y axis
- Every single tweet which meets the keyword criteria set **should** be analyzed.
- The brand for which tweets are to be analyzed **should** be easily specified in a user-friendly manner.

### 8.3.2 Implementation of pipeline

The final classifier model, 'LR\_0.9\_np\_ns\_c\_tfidf\_' was first exported using pickle, along with the subjectivity classifier 'SubjObj.sav'. The vectorizers used to process strings were also exported. These classifiers were then loaded into a new script, 'getTweets.py'.

```
#Load subjectivity classifier
subj_model = pickle.load(open('SubjObj.sav', 'rb'))
subj_vectorizer = pickle.load(open('SubjVectorizer.sav', 'rb'))
#Load sentiment classifier
sent_model = pickle.load(open('LR_preprocessed_augmented0.7_.sav', 'rb'))
sent_vectorizer = pickle.load(open('LR_preprocessed_augmented0.7_vectorizer.sav', 'rb'))
```

Figure 9, loading the subjectivity and sentiment classifiers

After loading the classifiers, the functions 'Subjectivity' and 'Sentiment' were written. These functions take a string argument and use the classifiers to return a predicted label for the string given by the classifier. For example, assuming no error in classification:

- Sentiment('I love this!') would return the string 'pos'
- Subjectivity('He walked to the shop') would return the string 'o'

```

4
5 def Subjectivity(data):
6     array = [data]
7     feature = subj_vectorizer.transform(array)
8     return subj_model.predict(feature)
9

```

Figure 11, 'Subjectivity' function used in full pipeline

```

0
1 def Sentiment(data):
2     array = [data]
3     feature = sent_vectorizer.transform(array)
4     return sent_model.predict(feature)
5

```

Figure 10, 'Sentiment' function used in final pipeline

After defining these functions, a function was defined which would stream twitter data, with a filter added so that only subjective tweets were returned. This was done using a modified version of the script used to save tweets used in the original dataset. To filter the stream of tweets in real-time, the following conditional statement was used.

```

# Filter out tweets with large number of retweets, tweets with L
if not any((( 'RT @' in tweet, 'RT' in tweet,
              tweet.count('@') >= 2, tweet.count('#') >= 3,
              'https' in tweet, Subjectivity(tweet) == 'o'))):

```

Figure 13, conditional statement used to filter tweets in real time

Note the addition of the statement “Subjectivity(tweet) == ‘o’”, allowing the system to ignore subjective tweets. This statement also filtered out any ‘retweets’, tweets with too many mentions and tweets with too many hashtags, for reasons described previously.

Streamed tweets were then saved to a CSV file, with an additional attribute for each tweet describing its sentiment polarity, as classified by the classifier loaded previously. The structure of each line of the CSV was as follows:

[Tweet, sentiment, keyword matched, date of tweet]

Although the keyword matches were not used in the final build of the pipeline, they were still recorded as they would potentially be useful if the implementation was expanded in the future, to track multiple sentiments simultaneously.

In addition to producing a CSV file with annotated tweets, a second CSV was also produced named 'brandname\_log.csv'. This file would be written to periodically, and would contain two columns. The first variable of each row of the file contained a floating point number which describes the average sentiment for all tweets gathered in the time period. This number was calculated by assigning each of the three possible sentiment polarities a value. Positive sentiment was equal to 1, neutral 0, and negative -1. The sum of the sentiment was calculated, before being divided by the number of tweets gathered in the time period in order to calculate average sentiment. The structure of the log file can be seen below:

[average sentiment, datetime calculated]

This was implemented with a function called 'log'. Using the 'timer' function found in Python's threading module, the 'log' would execute periodically. For the purpose of testing, the time period for which average sentiment was logged was set to 5 minutes, so that a large number of points could be tested when plotting the graph.

```
def log():
    #Set time period for log
    threading.Timer(300, log).start()
    #Append to Log CSV
    with open(filename+'_log.csv', 'a', encoding='utf-8') as log_file:
        #Get global variables for total sentiment and number of tweets
        global total_sent
        global tweet_count
        #Write average sentiment and current time to log
        log_writer = csv.writer(log_file)
        log_writer.writerow([total_sent/tweet_count, datetime.datetime.now()])
        #Reset these values for the next time period
        total_sent = 0
        tweet_count = 0
        print('point logged')
```

Figure 14, log function used to save sentiment averages periodically

Next, a second python script, 'track.py' was made. This would read the log produced by 'getTweets.py', and use 'matplotlib' to produce an animated line graph of the data. This was achieved by creating the following 'animate' function:

```

5 def animate(i):
6     csvfile = open( 'Playstation_log.csv', 'r+')
7     point_log = csv.reader(csvfile, delimiter = ',')
8     scores = []
9     time_str = []
10    time_stamps = []
11    for point in point_log:
12        if len(point) > 0:
13
14            scores.append(float(point[0]))
15            time_str.append(point[1])
16
17
18    ax.clear()
19    for time_string in time_str:
20        date = dt.datetime.strptime(time_string, '%Y-%m-%d %H:%M:%S.%f')
21        time_stamps.append(date)
22
23
24    times = matdate.date2num(time_stamps)
25    plt.plot(time_stamps, scores, linewidth=1.0)
26

```

By plotting average sentiment against time, the sentiment towards the brand can be visualised as it changes. Finally, in order to begin gathering and plotting data, the scripts 'gettweets.py' and 'track.py' must be placed in the same directory, and run. The result of this is an animated graph such as the one below.

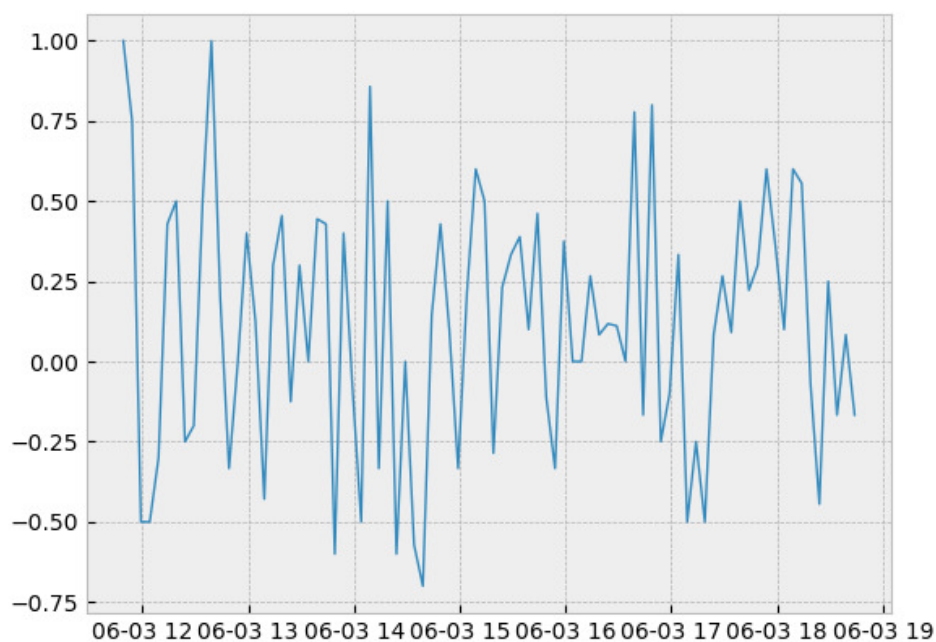


Figure 15, average sentiment towards the brand 'playstation' plotted at 5 minute intervals

## 9 Evaluation

### 9.1 Evaluation of pre-processing

Whilst testing model configurations, it became evident that the customized pre-processing function written was ineffective in comparison to sklearn's built-in function. Evidence of this can be seen in sections 7.2.1 and 7.2.2.

This could have been caused by one of several decisions taken when designing the function. For one, the decision to leave stop words within the tweet may have led to a decrease in accuracy, as the inclusion of these words leads to the classifier having a larger vocabulary, with less focus on words which define the sentiment explicitly. In sklearn's built in preprocessor, stop words are removed by default.

Although using the customized preprocessor reduced accuracy, there is a chance this is a result of the enhanced generalization it provides. Generalization of the model may be increased because of the preprocessor replacing every instance of a brand name in the dataset with the string 'brand'. Doing so makes the model applicable to any tweet which mentions a brand, so long as the brand name is also transformed from the unseen tweet before it is classified.

Due to project time constraints, this was not explored and tested further, however the assumption that this was true was made then deciding to implement the custom preprocessor in the full pipeline and when training the final model.

### 9.2 Evaluation of fact/opinion classifier

The fact/opinion classifier trained and implemented into the final pipeline had an accuracy score of 0.87. It must be noted that this was when tested using the dataset of movie reviews and plot synopses aforementioned, and how well this accuracy would generalize to classifying tweets is difficult to determine without further testing, which was not permitted due to time constraints.

The fact opinion classifier was implemented in the final pipeline despite this because of the fact that even if some tweets were classified as subjective incorrectly, so long as more were classified correctly than incorrectly (over 50%), then the model has been at least somewhat effective in reducing the number of objective tweets that reach the sentiment classifier.

### 9.3 Evaluation of different datasets

When over 1000 tweets are annotated by-hand, there is always the risk of human error. Although utmost care was taken to mitigate this risk, upon evaluation some tweets have been found to have been classified incorrectly. An example of one such tweet can be seen below:

‘That ""Dastardly"" trophy though 🤔😏’

Although this tweet has been annotated as positive, it is arguably of negative sentiment, as it appears to fit the criteria of:

“The tweet describes a negative experience or fault with a product/service provided by the brand”

Set out in the guidelines for annotation.

Although some human error is inevitable when annotating such a large volume of tweets, I do believe that it was kept to an acceptable level. This is evidenced by the effectiveness of the final classifier trained. If a significant number of tweets were annotated incorrectly, the final model would not have produced an accuracy score of 0.61.

Another aspect of the dataset which was not considered was the risk of producing an overfitted model. Due to time constraints, the data which was hand-labelled contained tweets from only five brands. This meant there was a risk that the model trained would be overfitted, and only accurate for the brands considered. This risk was mitigated when the dataset was combined with the ‘sentiment 140’ dataset, as the tweets contained in these were not brand specific.

### 9.4 Evaluation of final classifier

The comparisons of SVM and Logistic regression models made indicated that performance was similar between them. The highest accuracy score achieved for each was 0.61.

This accuracy met the requirements set; however, it is by no means perfect. Given a larger dataset, or more effective preprocessing, the model could undoubtedly have been improved further.

### 9.5 Evaluation of full pipeline

Comparing the full pipeline to the set of requirements listed before implementation, the following was noted:



- The implementation was generalized for any brand, as changing the keywords in the keyword list at the top of the script would result in all instances of these in the tweet being replaced with the word 'brand', as done when training the model. Requirement met
- The implementation was flexible, and different models could be used by simply dropping them into the same directory as 'getTweets.py' and changing the file name that is unpickled at the beginning of the code. Requirement met
- The implementation calculated and stored average sentiment periodically, as planned. Requirement met
- All tweets were classified by both the sentiment and subjectivity classifiers in real time. Requirement met
- A live line graph providing up-to-date visualization of this data was produced. Requirement met

From this, the final pipeline appears to meet all the minimum requirements set out. This does not mean, however, that this implantation was perfect with no room for improvement. Some of the non-compulsory requirements were unfortunately not met:

- The graph produced was not labelled, and the scale for the Y axis was fitted to the highest and lowest sentiment values, e.g. if the lowest average sentiment recorded was -0.2 then the axis would begin at this. In order to improve this, the Y axis should have been between -1 and 1, as this would accommodate all possible average sentiments and improve visualization.
- Due to the limitations of Twitter's search API, there is no guarantee that every single tweet which meets the keyword criteria is streamed into the pipeline. To overcome this, an alternative API such as 'firehose' API could be utilized instead.
- Although the target brand could be changed, it was only through modifying the keyword list in the code itself. This is not especially user friendly. To improve this, keywords could be entered through command the line or a GUI instead.

## 10 Conclusion

The goals of the project were to:

- Collect and label a dataset of tweets which could be used for training a sentiment analysis machine learning model.

- Implement a Logistic Regression classifier capable of differentiating subjective and objective tweets.
- Test a wide range of machine learning classifiers and obtain a model with maximum performance.
- Use this model, alongside the subjectivity classifier, to build a full sentiment analysis pipeline and visualize results.

The first of these goals was achieved. Tweets were gathered using the twitter stream API and saved to a csv, and then labelled using a simple python script. Tweets were labelled using a carefully laid out set of guidelines, and I believe that few were labelled incorrectly. The annotated tweets were used to build an effective model.

The second of these goals was also achieved. The subjectivity classifier had an accuracy score of 0.87 when tested on the dataset of movie reviews and synopses, which is more than acceptable.

The third of these goals was achieved to an acceptable degree. Many different configurations of SVM and Logistic regression models were tested, and yielded accuracy scores ranging from 0.45 up to 0.61. The results of these tests were visualized, and the effects of changing certain aspects of the model understood. Although not perfect, the performance of the model was good enough as to be used in a real application.

Finally, a pipeline which implemented the aforementioned models was built. This pipeline took data from twitter in real time, processed it, and saved a copy of each annotated tweet, as well as a calculation of average sentiment. This average sentiment was then used to plot an animated graph in matplotlib. Although the implementation used 5-minute intervals for proof-of - concept, far larger intervals could be utilized in order to visualize meaningful sentiment changes over time.

This project has proven the ability of machine learning models for classifying the subjectivity and sentiment of tweets, and given an example of a real-world application for these classifiers.

## 11 Future expansion

Although the result of this project is somewhat effective classifier and a working, live sentiment tracker, there is significant room for this to be built upon and improved. Every point from data collection to model training and testing to the final pipeline has limitations which could be

overcome with further work. In some cases, code was written specifically so that if improvements are made to an aspect of the system, they can be implemented with ease.

### 11.1 Data collection

Although the dataset collected and annotated was capable of training a classifier, it was not perfect. Many machine learning models benefit from a larger dataset, and by increasing the number of tweets collected and annotated, the models produced further along in the project could have been more effective.

As evidenced in the dataset evaluation, there was also cases of human error in the annotation of tweets. To reduce this error, future annotations could be checked by another independent party, and only kept in the final dataset if both parties are in agreement.

### 11.2 Model testing and selection

Although Logistic Regression and SVM models were covered in this project, for reasons mentioned previously in section 5.2.1, there are a multitude of other classifiers which could have been utilized. These include Naïve Bayes, Decision trees, Bayesian regression, and neural networks amongst others.

It is likely that given enough time, an alternative machine learning algorithm could be found which would be capable of classifying sentiment more effectively. In recent years, neural networks have become an area of interest, and have begun to overtake more traditional techniques. It is not unlikely that a well selected, trained, and tuned neural network would perform more effectively than the final model used. If this were the case, it could be implemented into the full pipeline easily, and produce more meaningful results.

### 11.3 Full pipeline

The script which allows the live-plotting of sentiment worked on a basic level, however there are many tweaks and improvements that could be made.

Firstly, the method used to stream tweets could be improved. The current build utilizes the twitter streaming API. The slow rate with which tweets matching the stream criteria are received means that often in a given time period, the average sentiment is based only on a handful of tweets. To bypass this limitation, the twitter ‘firehose’ API could instead be used. This is an API similar to Twitter’s streaming API, the key difference being that firehose guarantees delivery of every single tweet which matches a given criteria. This means that a larger volume of tweets can be collected and analyzed.

Although the time period for each calculation of average sentiment was 5 minutes, due to time and technical constraints, by increasing the time period to once per hour or once per day, more meaningful changes in sentiment could potentially be observed.

The live tracking feature implemented in the final pipeline could be used for several purposes in a number of ways. One use case for live sentiment tracking is for brands to monitor customer reactions to a new product release, advertising campaign, or update. If sentiment is seen to dip soon after a product is unveiled, it is an indicator of a negative consumer reaction. The inverse is also true. If sentiment was plotted on a daily basis instead of in a period of minutes, this change would become more easily visible.

Furthermore, the animated graphs could be embedded into webpages, alongside a method for filtering by keyword, in order to make sentiment data easily accessible, and understandable to anyone.

In addition to this, further information could also be extracted from the tweets and displayed to the user. For example, the most positive and most negative tweets could be found by recording the confidence of each classification. The tweets which the classifier has the highest confidence in predicting could be displayed to the user to provide further insight.

## 12 Reflection

From start to finish, this project presented a considerable challenge. It required me to become familiar with a topic I was previously unfamiliar with in a short space of time, to the point where my knowledge could be used to implement a full, working system.

I feel that by dividing the project into sub tasks, e.g. collecting dataset, testing models, implementing most accurate model, I was able to work more effectively. At times I had to consider if I was spending too much time on one aspect of the project, and as such there was room left for further improvements throughout.

I also believe I mismanaged the time taken for working on the actual code and model testing, and so did not leave long enough to write a report with as much detail as I would have liked. Having said this, I am not unhappy with the report, I just think that more supplementary content would have improved it.

The most satisfying part of the project was implementing the full pipeline, and watching actual data be processed and visualized in real time. I was unsure if I would be able to do this in the

time given, and although the output was more of a proof of concept, I believe it could have valuable real-world applications.

## 13 References

- [ “oberlo.co.uk,” Oberlo, 30th November 2019. [Online]. Available:  
1 <https://www.oberlo.co.uk/blog/twitter-statistics>.  
]
- [ IFTT, “Stream tweets with IFTT,” [Online]. Available: <https://ifttt.com/twitter>.  
2  
]
- [ Zapier, “How it works,” [Online]. Available: <https://zapier.com/how-it-works>.  
3  
]
- [ Tweet download, “Tweet download,” [Online]. Available:  
4 <https://www.tweetdownload.net/>.  
]
- [ lionbridge, “essential guide to sentiment analysis,” [Online]. Available:  
5 <https://lionbridge.ai/articles/the-essential-guide-to-sentiment-analysis/>.  
]
- [ O. languages, “Oxford Languages definition of subjectivity,” [Online]. Available:  
6 <https://www.lexico.com/en/definition/subjective>.  
]
- [ “Regression vs classification,” medium.com, 11 August 2018. [Online]. Available:  
7 [https://medium.com/quick-code/regression-versus-classification-machine-learning-whats-](https://medium.com/quick-code/regression-versus-classification-machine-learning-whats-the-difference-345c56dd15f7)  
] [the-difference-345c56dd15f7](https://medium.com/quick-code/regression-versus-classification-machine-learning-whats-the-difference-345c56dd15f7).
- [ J. Brownlee, “Machine learning mastery,” 18 March 2016. [Online]. Available:  
8 [https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-](https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/)  
] [machine-learning/](https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/).

[ A. Bhande, “What is underfitting and overfitting in machine learning and how to deal with it,” medium, 18 March 2018. [Online]. Available: <https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76>. ]

[ “Dimensionality and reduction algorithms,” elite data science, [Online]. Available: <https://elitedatascience.com/dimensionality-reduction-algorithms>. ]

[ J. C. Collados, “Applied Machine Learning Slides”. ]

[ Amazon, “cross-validation,” Amazon AWS, [Online]. Available: <https://docs.aws.amazon.com/machine-learning/latest/dg/cross-validation.html>. ]

[ “logistic regression,” Machine learning mastery, 12 August 2019. [Online]. Available: <https://machinelearningmastery.com/logistic-regression-for-machine-learning/>. ]

[ W. H. P. A. M. Sandra Vieira, “Science direct,” 4 Jan 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0149763416305176>. ]

[ “Support vector Machines,” Towards data science, [Online]. Available: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>. ]

[ g. f. geeks, “data preprocseing python,” [Online]. Available: <https://www.geeksforgeeks.org/data-preprocessing-machine-learning-python/>. ]

6

]

[ H. Jabeen, “Stemming and Lematization in python,” 23 Oct 2018. [Online]. Available:  
1 [https://www.datacamp.com/community/tutorials/stemming-lemmatization-](https://www.datacamp.com/community/tutorials/stemming-lemmatization-python?utm_source=adwords_ppc&utm_campaignid=9942305733&utm_adgroupid=100189364546&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adposition=&utm_creative=255798340456&utm_tar)  
7 [python?utm\\_source=adwords\\_ppc&utm\\_campaignid=9942305733&utm\\_adgroupid=1001](https://www.datacamp.com/community/tutorials/stemming-lemmatization-python?utm_source=adwords_ppc&utm_campaignid=9942305733&utm_adgroupid=100189364546&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adposition=&utm_creative=255798340456&utm_tar)  
] [89364546&utm\\_device=c&utm\\_keyword=&utm\\_matchtype=b&utm\\_network=g&utm\\_a](https://www.datacamp.com/community/tutorials/stemming-lemmatization-python?utm_source=adwords_ppc&utm_campaignid=9942305733&utm_adgroupid=100189364546&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adposition=&utm_creative=255798340456&utm_tar)  
dposition=&utm\_creative=255798340456&utm\_tar.

[ Scikit-learn, “Scikit-learn,” [Online]. Available: <https://scikit-learn.org/stable/>.

1

8

]

[ N. l. toolkit, “NLTK,” [Online]. Available: <https://www.nltk.org/>.

1

9

]

[ M. l. mastery, “Accuracy paradox,” [Online]. Available:  
2 [https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-](https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/)  
0 [machine-learning-dataset/](https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/).

]

[ Sentiment140, “Sentiment 140 dataset,” [Online]. Available:  
2 <http://help.sentiment140.com/for-students>.

1

]