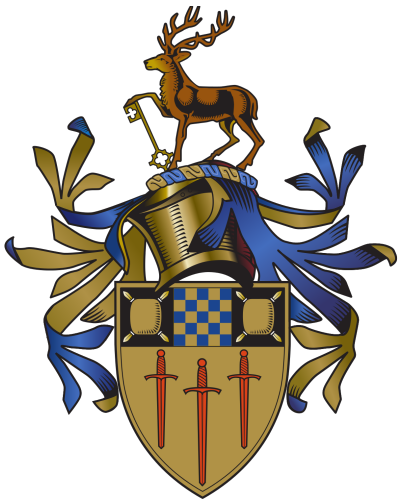
Sentiment Analysis in Social Media Monitoring

Rishi Patel

Computer Science

Project Report

May 18, 2021



Department of Computing

Faculty of Engineering and Physical Sciences

University of Surrey

Project Supervisor: Liqun Chen

Declaration of Originality

I confirm that the submitted work is my own work. No element has been previously submitted for assessment, or where it has, it has been correctly referenced. I have also clearly identified and fully acknowledged all material that is entitled to be attributed to others (whether published or unpublished) using the referencing system set out in the programme handbook. I agree that the University may submit my work to means of checking this, such as the plagiarism detection service Turnitin® UK. I confirm that I understand that assessed work that has been shown to have been plagiarised will be penalised.

Rishi Patel

May 2021

Acknowledgements

I want to thank God first and foremost for getting me through this exciting project. I would like to thank Liqun Chen for the notable guidance and direction from beginning to end. I also thank Dr. Toral Patel for her support and advice that helped me accomplish my goals. Finally, I would like to thank my family and friends who have offered their advice, moral support, and gratitude.

Abstract

Analysis or opinion mining is a technique where a computer is taught to gather sentences, phrases, or opinions and extract subjective information. For example, a sentiment expressed as "I love food" would be considered positive whilst a sentiment expressed as "I hate the cold weather" would be classified as negative. The application of NLP (Natural Language Processing), statistics, and machine learning are some of the most important tools used to extract this information.There are various types of SA such as subjective and objective analysis, sentiment detection and classification, fine-grained analysis and Aspect-Based Sentiment Analysis (ABSA). Although similar work has been carried out in the field of SA which uses the traditional approach of sentiment detection and classification, there still needs to be more applications and techniques combined to determine SA in very fine detail. Also, other work within literature has analysed these sentiments largely through documents, product reviews, and media reviews whilst there remains a challenge of identifying detailed analytics of sentiments with high accuracy for social media. This project demonstrates that more than one technique of SA can be used, advocating a more hybrid approach that can be implemented by combining sentiment classification and ABSA that provides a high-level view of the classification of the text and a low-level view of the entities and features within the text itself. The underlying mechanisms which are used for both techniques of the hybrid approach rely on various machine learning algorithms/models and NLP techniques. This project will additionally compare and contrast these algorithms/techniques with performance metrics such as accuracy of the models and F-1 score.

After evaluation, the best performing model will be embedded within a web application that allows users to search social media such as Twitter. The application will help gather this information in real-time and have detailed SA classifications with key aspects identified returned to the user.

The result of the project should see a business user search for a topic, obtain accurate predictions, and use this information to understand how sentiments are affecting their brand reputation.

Contents

Nomenclature

SA	Sentiment Analysis
ABSA	Aspect Based Sentiment Analysis
ML	Machine Learning
NLP	Natural Language Processing
TF-IDF	Term Frequency-Inverse Document Frequency
POS	Part-of-Speech
API	Application Programming Interface
MVC	Model-View-Controller SQL
Structured Query Language UI	User Interface
TPR	True-Positve Rate
FPR	False-Positive Rate

List of Figures

1 Introduction

1.1 Problem Background

75% of global data and information are largely unstructured. [1] This means businesses generate swaths of data, which is very unorganised, resulting in large amounts of valuable information are being discarded.

The data include emails, customer service logs, product reviews and most importantly social media. Social media is increasingly becoming the forefront of quickly creating extensive amounts of sentiments towards a business in real-time.

Understanding and organising this data with the application of Sentiment Analysis (SA) will help businesses and organisations create more practical solutions. Several techniques can be applied such as gauging customer/user feedback through manual monitoring. This method is unproductive as it can be very monotonous and ineffective as it will take a person enormous amounts of time to sort through the sentiments and classify them.

A more useful technique would be to use machine learning and allow the computer to learn and train on a past data set then use the resulting model to classify sentiments.

Pre-processing will need to be done on the dataset. There are various methods of pre-processing and implementing NLP techniques will be necessary for both the sentiment classification and ABSA. This step is crucial as feeding more accurate data into the models will affect the final accuracy of the model and therefore, the sentiment classifications.

There are plenty of approaches to utilising a machine learning model and applying NLP techniques, however, a thorough analysis and comparison is required to determine the best approach which will produce the most accurate sentiment classifications alongside accurate aspects based on the sentiment.

1.2 Project Description

This project will develop and implement a web-based solution for SA and ABSA in parallel to providing in-depth research of SA techniques and machine learning algorithms. The user will be able to enter a topic on the web application to search which fetches social media sentiment from Twitter. Each sentiment retrieved will be fed through the sentiment classification model and ABSA model.

thorough analysis of the Multinomial Naïve Bayes, Logistic regression, and vectorizer algorithms will be made to determine the best model for the sentiment classification. The ultimate goal would be to reach the most effective model that can handle the correct classification of sentiment and be

performance efficient whilst making predictions so the end-user can minimise the time of waiting for their results. After, the ABSA model with NLP processing will be created to delve deeper into the features of the sentiment and extract key aspects.

Each model will be tested, and evaluation metrics will be compared in the research. The evaluation metrics to consider are accuracy, F-1 score, true positive rate, false-positive rate, and additional metrics such as user acceptance testing. Thereafter, a model will be chosen and implemented for sentiment classification and ABSA.

The project will then explore the results stage of a machine learning web application utilizing both models and subsequentially providing accurate sentiment classifications and feature identifications to the user on the front-end of the web app.

A business will be able to use the sentiment classifications and aspects identified to further adapt or change certain processes that will help monitor and improve their reputation/branding.

1.3 Project Aims

The overall project aims to define how SA can be used effectively and how most companies can add value through understanding customer-based sentiment through:

- Explore and analyse a dataset that will be relevant to Twitter data and use this dataset for the training and testing processes of each machine learning algorithm.
- Research into machine learning algorithms, how Natural Language Processing (NLP) is used for SA.
- Research into the different approaches to SA and try to find the best techniques which can produce the most accurate results with a model achieving an accuracy of at least 80% or above.
- Implement a basic web application to show how a machine learning model using SA as well as ABSA can be used together to enable monitoring of social media sentiment.
- Implement a second model/system within the web application using ABSA on the same data.
- Provide the user with the ability to have the most engaging sentiments returned to them along with their sentiment classifications.
- Conduct thorough testing of the web application functionality and evaluate the results and accuracy of the classifications.
- Draw conclusions on the SA findings, reflect on implementation successes, challenges, and how the development can be improved in the future.

1.4 Expected Outcomes

There are several expected outcomes for this project. Firstly, there will be a literature review section which will cover all the topics for SA and ABSA including web and machine learning technologies being used for the web application. Afterwards, a dataset will be analysed and processed for the machine learning phase in section 3. Once the machine learning analysis and web application design is determined then the implementation will begin based on the functional and non-functional requirements. A thorough evaluation will be performed after the implementation reflecting on the project aims, successes, challenges, and further development considerations. Finally, this project will consider and reflect upon a statement of ethics which will cover the legal, ethical, social, and professional considerations.

2 Literature Review

This chapter will focus on the relevant literature associated with sentiment analysis, aspect-based sentiment analysis, NLP techniques, and machine learning algorithms. This chapter will also compare existing solutions of sentiment analysis for social media monitoring.

2.1 Twitters Evolution in SA

Twitter is a global social media platform commonly used as a "micro-blogging" service created in 2006 and has grown into one of the top 10 social media platforms in the world. [5]. Twitter allows users to create short 280-character messages which are called "Tweets" [6]. Each Twitter user has a set of followers and people they are following. The entities you can follow can be anyone from word leaders and news organisations outlets to businesses and ordinary civilian users. Over the years ever since its launch, Twitter has been gaining popularity within the SA realm mainly because of the real-time messages being sent – once a user makes a Tweet it is made public instantly on the platform for anyone to see (unless security settings are changed for the user). Twitter has many applications covering various subject domains, and it is an essential part of an individual's or business's daily digital life. [4]. The way Twitter has interconnected users around the globe creates a diverse-user base and active participation from users generates a strong foundation for analysing sentiments both qualitatively and quantitatively. [4]

2.2 Sentiment Analysis for text classification Use in Literature

SA is a type of text classification which categorizes text based on the polarity of the text. The polarity can be positive, neutral, or negative. Social networks in general act as a great medium for where opinions and sentiments can be expressed. [2]

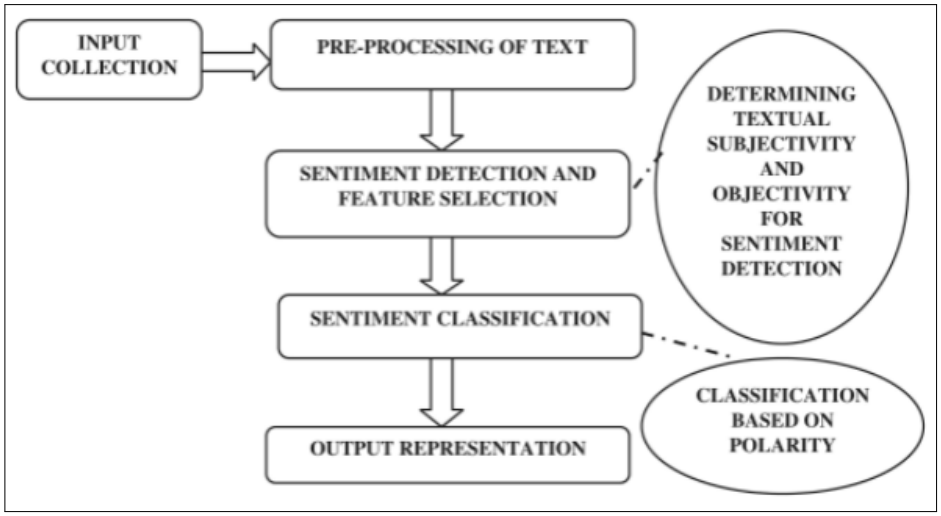


Figure 2-1: Diagram representing the process of sentiment analysis for text [4]

Although it is known that SA can be used within social networks it needs to be analysed further to explore the benefits and incentive for businesses to adopt SA tools.

As there are lots of unstructured data on the web from social media platforms such as Facebook, Instagram, and Twitter – there needs to be a way to extract important information. In the past decade, there has been a driver for business innovation which is called SMAC (Social-media, Mobile, Analytics, and Cloud) [3]. SMAC transforms the functioning environment and user engagement on the web and social media whether it be through your mobile device, data analytics or cloud computing [4]. The effects of SMAC create a shift in people using e-commerce to s-commerce. S-commerce is a child of e-commerce but uses entities such as Twitter, Google, Amazon, and Facebook to drive online marketing and sales. Most importantly, S-commerce has been able to generate more online activity enabling users to discuss, share, analyse, appreciate, and research about different brands and services through social media such as Twitter and Facebook. All the information and sentiments gathered through S-commerce and other online shopping outlets can be analysed further to enable both the consumer and seller/organisation to benefit. For example, data analytics on the sentiments within social media platforms has always been a trend where online texts, posts, reviews, Tweetss, and comments are made into the sentiment-rich knowledgebase. The knowledgebase can be used for effective changes and decision making in the future. An instance of a consumer making wise decisions is utilizing the reviews or posts about other users and creating their own decisions based on other users' sentiment. Conversely, a business example would be for a business or organisation to monitor sentiments from social media and be able to create effective decisions to improve or take note of current user sentiment whether it be positive, neutral or negative. Therefore, we can confidently say that the Internet has extensive amounts of data to be analysed under scrutiny, and sentiment analysis can form a solid foundation. [4]

2.3 ABSA in literature

ABSA is a more fine-grained analysis that relates to the aspects or entities of a text. ABSA comes in handy when classifying the polarity of sentiment does not extract enough information. For example, let's use the following sentiment: "I really enjoyed going to the new tasty seafood restaurant by the ocean, although the service could have been a bit faster." The sentiment has a generally positive sentiment, however, key bits of information such as "service could have been faster" can be extracted and show how the user has found something negative about their experience. Therefore, the need for ASBA and more fine-grained analysis would enable businesses, organisations, and consumers to recognise key pieces of information upon which they can make improvements [7].

2.4 Showcasing different approaches, architectures, and techniques for SA and ABSA

Natural Language Processing (NLP) is an application that explores how computers can be used to understand and orchestrate natural language text or speech to do practical things such as SA or ABSA. [8] The core foundation of NLP contains many different areas of study such as computer and information sciences, linguistics, mathematics, electronic engineering, artificial intelligence and robotics, and psychology. [8] The way NLP can be applied is through artificial intelligence, multilingual and Cross-Language Information Retrieval (CLIR), natural language text processing and summarization, speech recognition, and user interfaces. [8] Most importantly, NLP techniques can both be used for SA text classification and ASBA extraction.

2.4.1 Word Embeddings (Word2Vec)

A popular NLP technique is word embeddings. A word embedding is a numerical vector-based representation of the word based on the context of how the word is being used. [9]

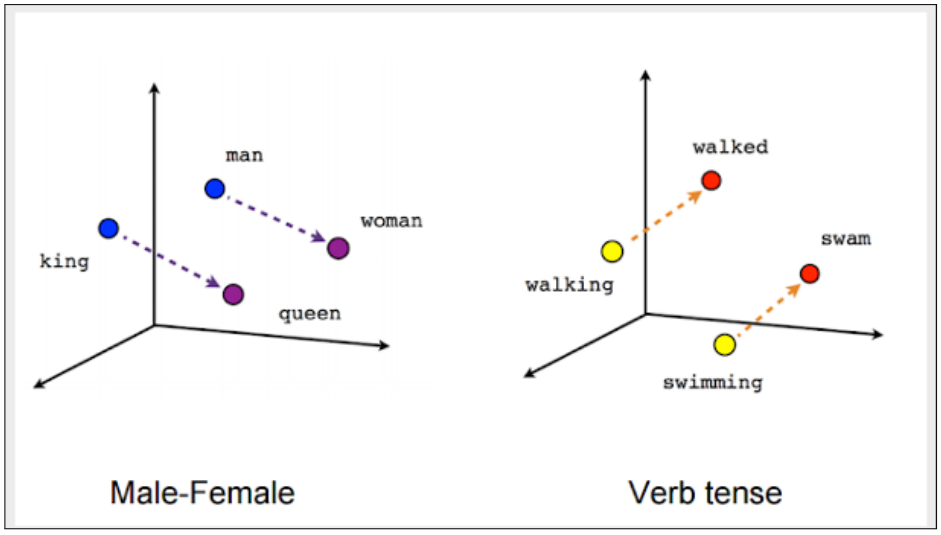


Figure 2-2: diagram representing words in a word vector space [4]

A popular technique for word embedding that will be used is Word2vec. Word2vec is a model which represents words as vectors. The text in a Word2vec model is represented by a vector in a lower dimension of space and this space each vector has a certain orientation. The relationships between these vectors in the lower dimension of space can define relationships amongst the words. [10] As seen in figure 1 there are representations of the words and their respective vectors represented in a 3-d vector space. The way the words are plotted is based on cosine similarity. Cosine similarity is a normalized dot product of two vectors. The result of this product represents the angle between the two vectors. If two vectors happen to have the same cosine similarity of 1, then they have the same orientation. If two vectors are at 90 degrees, then they have a cosine similarity of 0. Finally, if two vectors have an orientation in the opposite direction then they have a cosine similarity of -1 and this is not dependent on their magnitude. [10] Word embeddings are important as they help prepare the data for further machine learning algorithms and classifications.

2.4.2 TF-IDF

Another technique is the Term Frequency – Inverse Document Frequency (TF-IDF). The TF-IDF value is a numeric statistical measure that represents how significant a word is to a particular document that is contained in a collection of written texts or corpus. The numeric score is determined by the number of times a word appears in a document divided by the frequency of the word appearing in the collection of written texts or corpus. Many search engines utilise TF-IDF and in the case of SA, the significance of the word plays a key role in determining polarity. [11]

2.4.3 Part-of-speech tagging

Additionally, another applicable text analysis technique is Part-Of-Speech tagging or POS.POS parses the given text and will identify the POS tag such as nouns, verbs, adjectives, conjunctions,

pronouns, adverbs, and interjections. POS tagging is essential in ABSA since it helps to give an understanding of words that may come before certain words. For example, "I really liked going to the new London bar, however, even the small drinks were expensive" contains the noun "drink" and its neighbour "small" so the aspect extracted would be "small drinks." Another key aspect that could be identified through POS is entities. Using the most recent example, the word "London" represents a proper noun and its neighbour "bar" so the entity "London bar" can be extracted. [12] POS tagging has been used in various tasks such as ASBA, parsing, and informational retrieval.

2.4.4 Dependency Parsing

Dependency parsing is key to ASBA as it helps to identify the relationships between words. Many dependency parsers use a graphical tree method to represent these relations for better visualization. This type of dependency structure is also known as a typed dependency structure since the labels are drawn based on a pre-existing set of grammatical notations.

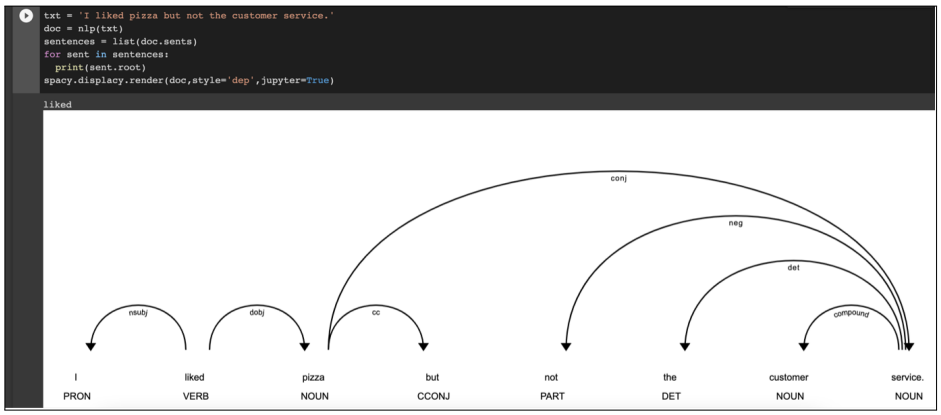


Figure 2-3: Dependency tree structure from spaCy

In Figure 2-3, labelled arcs are representing the dependent relations between each word. The tree structure contains a head and several dependents. The head-dependent relationship is characterised by having a root word (head) and the dependents from that root word. [13] As seen in Figure 2-3 the root word is "liked" and the direct dependents are "I" and "pizza." The headword signifies the centre of the overall dependency structure. For example, if there was a verb phrase or noun phrase the singular verb or noun would be considered the head. However, as mentioned previously, there are many more dependency relations seen in figure 2-4 below. Another important relationship is the subject and object of a sentence or text. As seen in Figure 2-3 the word "I" is the nominal subject and "pizza" is the direct object which relates to the head. However, there are many more notations available because of "Universal Dependencies." Universal Dependencies is a project which aims to have a consistent catalogue of dependency relations that span across multiple languages and helps to create a multilingual development of parsers, the ability for learning across many languages and further parsing research. [14] The basis of the project relies on the advancements made by the

Stanford dependencies, Google universal tag set, and various other tag sets. [14]

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 2-4: Dependency relations [13]

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Figure 2-5: Dependency relations continued [13]

A popular type of dependency parser which is used in various NLP tasks and information extraction is a transition-based dependency parser that is influenced by shift-reduce parsing. This parser at its foundation contains a stack, grammar, and a catalogue of tokens that need a constituent structure assigned by analysing the tokens syntactically. The tokens are placed onto the stack one by one and the first couple elements "are matched against the right-hand side of the rules in the grammar." [13] A matched element will perform a "reduce" function where the element gets replaced onto the stack. The match happens with the "non-terminal from the left-hand side of the rule being matched." However, when dependency parsing transitionally then the grammar can be ignored. Instead, the word at the head of the stack will have a head-dependent relationship determined between itself and the word just below it. Alternatively, a head-dependent relationship can be determined between the word below and the word on the head of the stack. [13]

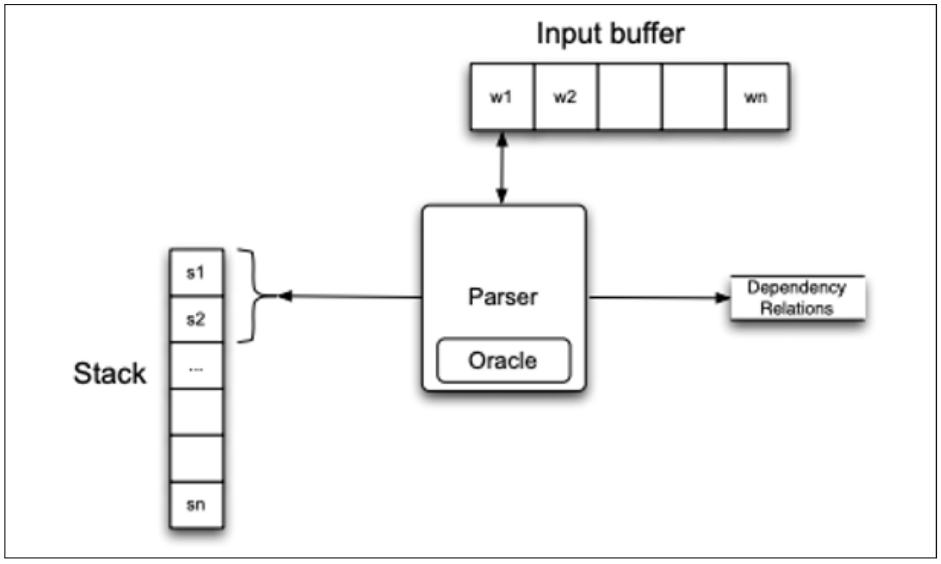


Figure 2-6: An example of a bare bones dependency parser [13]

A benefit of using a dependency-based approach for ASBA is that the head-dependent parsing function enables relations to be explicitly defined between words including the root which makes it crucial for information extraction. [13]

2.4.5 Tokenization

Tokenisation is one of the most crucial first steps in SA and ASBA extraction. Tokenisation breaks sentences or strings of words into smaller pieces. For example, take the following sentence: "I really liked the Computer Science course at the University of Surrey." The tokenised sentence would be a list of each word or punctuation such as the following: ["I", "really", "liked", "the", "Computer", "Science", "course", "at", "the", "university", "of", "Surrey", "."]

In the array above each word is tokenised including the ending period punctuation. However, there are several challenges when tokenising such as knowing how to split the sentence, handling punctuation characters, and language-specific syntax.

A popular method of breaking up the sentence into tokens is applying several rules when tokenising. For example, splitting words where there are white space and splitting contractions.

Language issues occur when the tokenizer cannot understand different languages and their syntax. An instance of this would be the French phrase "au fait", which should not be split. Another would be the German compound noun "Computerlinguistik" which tokenised in English would be "computational" and "linguistics." Therefore, the tokenizer needs to serve a language-specific domain which in this project will be English. [15] A tokenizer that will be analysed in later sections will be the spaCy Python library. This library will be able to handle the above issues of splitting, punctuation,

and language. As seen in Figure 5 the sentence has been tokenised and is ready for POS and dependency parser tagging for ABSA extraction.

```
text = "Also, Mr. Johnson thinks that the girls' arguments about Obama's presidency aren't accurate."
doc = nlp(text)
for token in doc:
  print(token)
```

```
Also
,
Mr.
Johnson
thinks
that
the
girls
'
arguments
about
Obama
's
presidency
are
n't
accurate
.
```

Figure 2-7: Each line is representing a token using the spaCy library

2.4.6 Lemmatising and Stemming

Lemmatising is a technique that reduces the word to its base form to avoid inflectional word forms and other words very similar to a word in its common base form. Stemming does a similar job to lemmatising and gets rid of derivational affixes. However, stemming does not take context or the morphological analysis of the words into consideration and can lead to ambiguous and less accurate word forms. Consider the following:

Case 1:

"Caring", "Cares", "Cared" -> Lemmatization -> "Care"

"Caring" -> Stemming -> "Car"

Case 2:

"Studying", "Study", "Studies" -> Lemmatization -> "Study"

"Studying" -> Stemming -> "Study"

"Studies" -> Stemming -> "Studi"

As seen above in Case 1, lemmatization reduces the word to its base form or "lemma" and considers the context. Stemming does reduce the word; however, the original meaning of the word has changed

from the verb “caring” to the noun “car.” Additionally, case 2 shows that there are different stems for different derivational affixes for a word.

Stemming is much faster, simpler, and does not take into consideration the part-of-speech. Conversely, lemmatization takes into consideration context and must have a part of speech tag determined whilst lemmatising which takes much longer to process. Therefore, since the meaning of a word and its context is very important in this project, lemmatization will be one of the preferred approaches for text pre-processing. [16]

```
word_list = ["be", "am", "is"]
for word in word_list:
    doc = nlp(word)
    for token in doc:
      print(str(token) + " -> Lemmatization -> " + str(token.lemma_))
```

```
be -> Lemmatization -> be
am -> Lemmatization -> be
is -> Lemmatization -> be
```

Figure 2-8: Each line is representing a token using the spaCy library

Further analysis of the performance gains/losses of lemmatisation will be examined and discussed in the pre-processing section of this report.

2.5 Web Frameworks/Production Services

The web application will form the centre of showing the power of SA and ASBA in a logical and clean presentation. Therefore, a reliable web framework that can support fetching Tweets from Twitters API, machine learning, database management, server-side development, and production environment will need to be investigated and selected. All the challenges, such as the performance and size of the web application in its entirety, needs to be able to effectively be handled in a production environment.

2.5.1 Production Hosting Service

A requirement for the production service is it to be free as the web application should not be very large and needs basic components such as a database, task queue, and support for Python or C. Some potential options investigated were Heroku, Amazon Web Services (AWS), Google App Engine, and Digital Ocean.

Heroku was chosen as the primary production hosting service as it enables faster deployment compared to AWS, Google App Engine, and Digital Ocean. The deployment process is more streamlined and quicker with a Git repository or Git CLI commands. Heroku supports both Python and C based

applications to be deployed along with other essential services like a PostgreSQL database and a Redis task queue for asynchronous programming. Heroku offers all these services for free with the only limitations being the entire application size needs to be within a 500mb limit and the web application can have 20 simultaneous open connections. For a free tier service, Heroku offers everything necessary and automatic scaling is provided if needed in the future. [23]

2.5.2 ASP.NET Framework

The ASP.NET Framework is a web framework developed by Microsoft which enables web apps, services, and various .NET components to be built. Microsoft created and deployed C, F, and Visual Basic as their primary programming languages for development within the .NET Framework. More specifically, the popular C programming language within the ASP.NET framework enables powerful and dynamic web applications to be created. As required, choosing ASP would help in hosting a database through the use of MySQL or PostgreSQL. The machine learning component could be covered with ML.NET. ML.NET is Microsoft's machine learning library which integrates various machine learning algorithms and classifiers enabling users to implement data-intensive web applications. However, ML.NET is relatively new and lacks extensive documentation compared to Python machine learning libraries. Besides, ML.NET requires manual set up of data inputs and class construction whilst Python enables much easier data manipulation when selecting inputs for model creation. Another disadvantage of ASP.NET is it takes much longer to configure and can become quite hefty in application size when deploying to a production server. Performance and size need to be considered and are important when deploying to a production server since the end-user needs to have fairly quick responses to web requests. [17]

2.5.3 Flask

A more viable option is the Flask framework. Flask is a Python micro web framework compared to other popular frameworks such as ASP.NET or Django. The way Flask comes right out of the box makes Flask lightweight and easily configurable even for deployment to production servers. Flask helps to create a modular design which means there does not need to be anything included that is not necessary or wanted. This is because Flask encompasses its web server for development, but does not come with any database abstraction layer, authentication, or form validation. Instead, these essentials can be easily installed through various Python packages. Other important packages or libraries which can be installed is PostgreSQL with SQLAlchemy for relational database integration and the sci-kit learn library for a machine learning pipeline.[22] Flask uses Jinja to generate HTML/CSS web pages with JavaScript for the client-side user interface. Python will be the main programming language used in this project for backend development. Python is an interpreted language making it very fast and can handle swift data manipulation/pre-processing using the Pandas and NumPy libraries. [20] [21] Another useful Python library will be Tweepy. Tweepy is a library that handles authentication and retrieval of Tweets through Twitters API. Gathering the Tweets from Twitter

with Tweepy is a very quick process and data privacy ethics are acknowledged and safe. This will increase the performance and reliability when accessing the Twitter API for Tweets. Finally, Flask and Python together both have very strong documentation along with a huge community helping make Flask a strong choice for the web application. [18]

2.6 Machine Learning Pipelines Comparison

The web application and machine learning model need to be tightly coupled in the backend server. This means the server-side code must be able to communicate with the machine learning (ML) model to serve predictions. Training a model can take a very long time and would be very unpractical to make a user wait for their SA or ABSA predictions. A simpler solution would be to load a pre-trained model and use the predict function to generate predictions as soon as a user submits their option from the front-end user interface. There are several machine learning pipelines to consider for this specific task.

2.6.1 ML.NET

mentioned before, ML.NET would be a great choice since it does have its own quick and reliable prediction engine, however, it lacks the interoperability of being able to work with other Python libraries such as Tweepy or Flask. ML.NET would also be too large to efficiently deploy to a production server such as Heroku. [19]

2.6.2 TensorFlow

TensorFlow is a low-level ML and deep learning library created by Google which has powerful tools for generating ML models. TensorFlow can be used within Python. However, TensorFlow tends to focus more on deep learning models such as CNN's and RNN's and require a large amount of GPU power. Therefore, TensorFlow is not a good choice for this project. [22]

2.6.3 Sci-kit Learn

Sci-kit learn is the most reliable, lightweight and efficient library to create an ML pipeline compared to ML.NET and TensorFlow. Sci-kit learn tends to be focused on traditional ML algorithms rather than deep learning algorithms which will usually be more power-intensive and require dedicated GPU processing power. The library also features a TF-IDF vectorizer that should transform the training data into TF-IDF vectors. After, the vectorizer can be fit into various sci-kit learn ML algorithms such as Logistic Regression and Multinomial Naïve Bayes. Another important feature of this library is the ability to tune hyperparameters of these classifiers and see a confusion matrix

and other performance metrics such as True-Positive Rate (TPR) and False-positive Rate (FPR). [23]

2.7 Essential Python Libraries for Web Frameworks/Production Services

This section will cover essential libraries in Python which will help in SA and ABSA extraction.

2.7.1 spaCy

spaCy is a powerful NLP Python library that deals with artificial intelligence and various advanced NLP tasks. spaCy was created in 2015 and developed by a company called Explosion. spaCy has various different "pipelines" which NLP components that can be trained on various languages. For example, spaCy supports many languages including English, Spanish, Italian, Chinese, Dutch, Russian. For this project the English model will be used. The component of each pipeline integrates many different parts such as a tokenizer, POS tagger, dependency parser, and an entity recognition component. Firstly, the text is passed to the NLP pipeline and then a "Doc" object is created which will allow the user to access various pipeline components and analyse the text further. Figure 2-9 and Figure 2-10 are visualise examples which shows this process. Figure 2-10 specifically loads in the English language model denoted by "en_core_web_sm" and a doc variable is created. Afterwards, the a for-loop expression is used to access the various components of the linguistic features. [24]

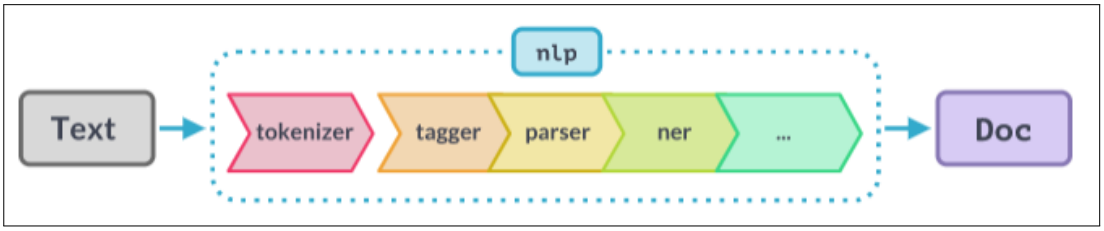


Figure 2-9: spaCy pipeline example [24]

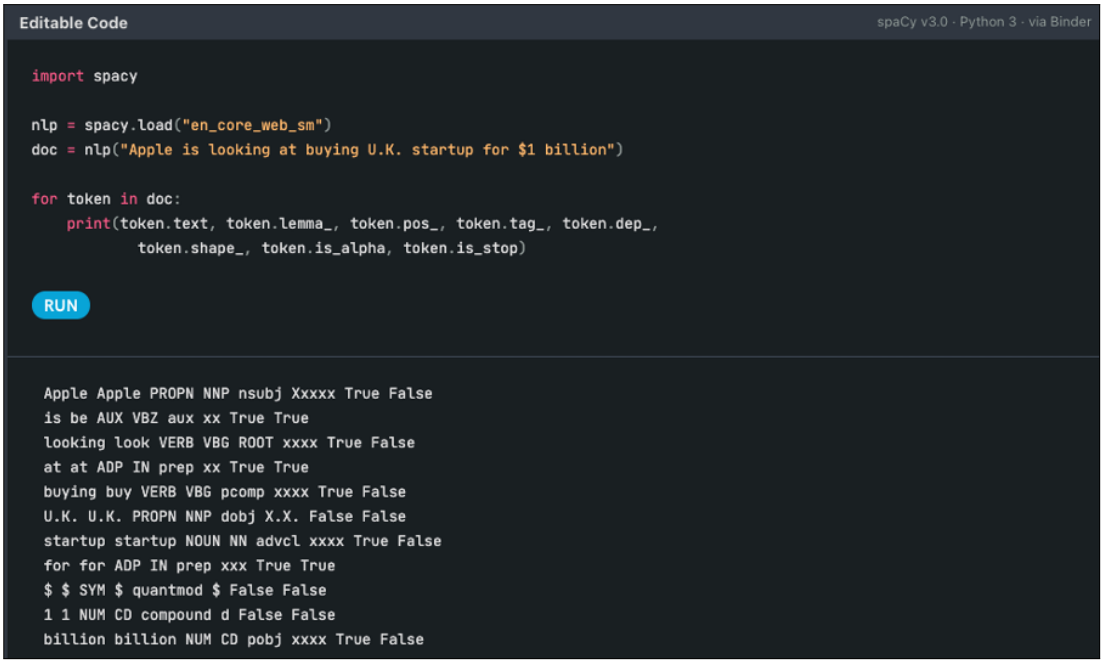


Figure 2-10: Example from spaCy documentation of looping through tokens of the doc object [24]

This project will heavily rely on the tokenizer, tagger, and parser components when pre-processing and analysing the text for the ABSA model.

2.7.2 Pickle

The models created after implementing an ML algorithm can be saved into (.pkl) binary files. The (.pkl) format comes from the Python Pickle library. The library allows compression of the model file which reduces the overall size. The (.pkl) can be loaded later within the web application instantaneously and new sentiments can be given to the model and predictions can be made for the user on the front-end. Pickles quick and efficient compression functions satisfy the requirement of having small model files with easy loading when on the production server. [26]

2.7.3 Tweepy

Tweepy is a Python library which helps access the Twitter API. This library is necessary as it will be the API which enables secure requests to Twitter when trying to retrieve data such as Tweets, Likes, Retweets, Trends, and various social media features. Tweepy uses OAuth to make the requests which is a secure authorisation protocol and Twitter Developer authentication credentials must be generated. Also, the library is written in Python which means it can be easily integrated into the Flask back-end. [27]

2.8 Determining Best Choices for Web Application Development

After a thorough analysis and comparison, the software services and libraries for the web application have been determined. Firstly, the production hosting service will be Heroku for ease of use and high interoperability of services. Heroku is also free and has a command line interface (CLI) which makes deployments simple and effective. Flask will be chosen for the server-side development as it uses Python and many Python libraires including ML libraires can be imported easily into a Flask application. Flask is also very lightweight and is very customisable. The ML library which will be used for the sentiment classifications will be sci-kit learn. Sci-kit learn is written in Python and supports various traditional ML models. As for the ABSA model, spaCy will be used since it contains very powerful NLP components which can be used to create powerful and fast performing analysis of sentiments. Pickle will be used as it is the only Python based library which will compress ML models into binary formats that can be deployed to a Heroku production web server. Using all the aforementioned libraries and services should be sufficient in creating a sentiment analysis web application. [25]

2.9 Machine learning Algorithm Analysis

The basis of the project relies on ML algorithms. The algorithm performance, accuracy, and F-1 score are important factors to consider. There are several types of ML algorithms such as regression models or classifiers. Regression models tend to use data to predict outputs. For example, a regression model can you past stock trading data to determine future trends whether it be stock prices increasing or decreasing.

It is expected that some algorithms will outperform others, however thorough analysis will be made to investigate the differences.

2.9.1 Logistic Regression

A strong model to implement would be a logistic regression model since it does a binary classification which would predict an output to be either true or false and yes or no etc. This is much more important when working with categorical values rather than continuous values. For example, the outputs of this project will mainly need to determine two categories which is positive or negative Tweets. Although there is a third category of neutral, this can be determined by a threshold.

A solution for classification is logistic regression. Instead of fitting a straight line or hyperplane, the logistic regression model uses the logistic function to squeeze the output of a linear equation between 0 and 1. The logistic function is defined as:

$$\text{logistic}(\eta) = \frac{1}{1 + exp(-\eta)}$$

And it looks like this:

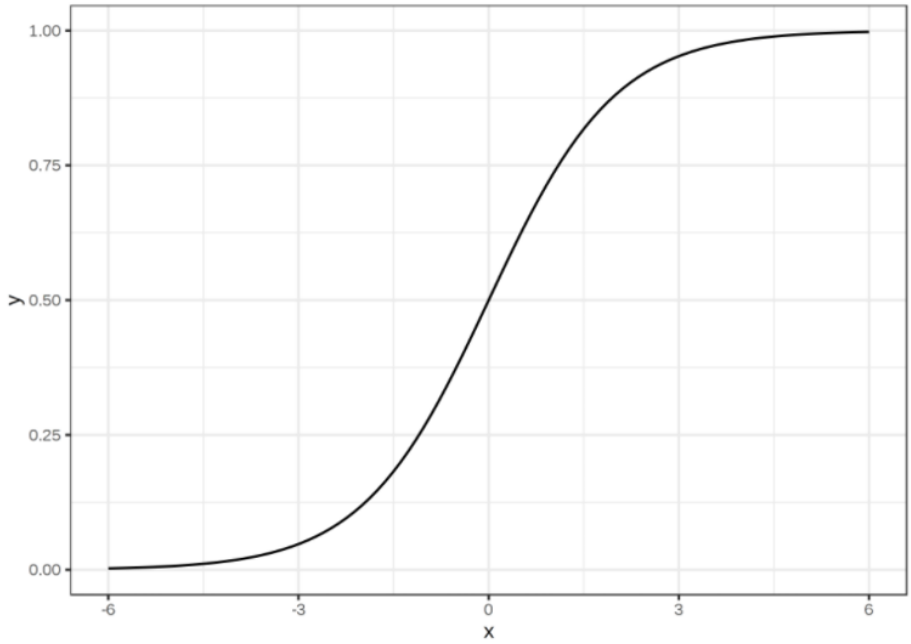


FIGURE 4.6: The logistic function. It outputs numbers between 0 and 1. At input 0, it outputs 0.5.

Figure 2-11: An example of a Logistic Regression curve fitting outputs between 0 and 1. [26]

To understand why logistic regression a statistically good algorithm is we need to understand the advantages it has over linear regression. Linear regression similarly treats classes as 0 and 1, however, linear regression is not suitable for interpreting probabilities of classifications. This is because linear regression algorithms or models try to interject between points and determines a line of best fit. [26]

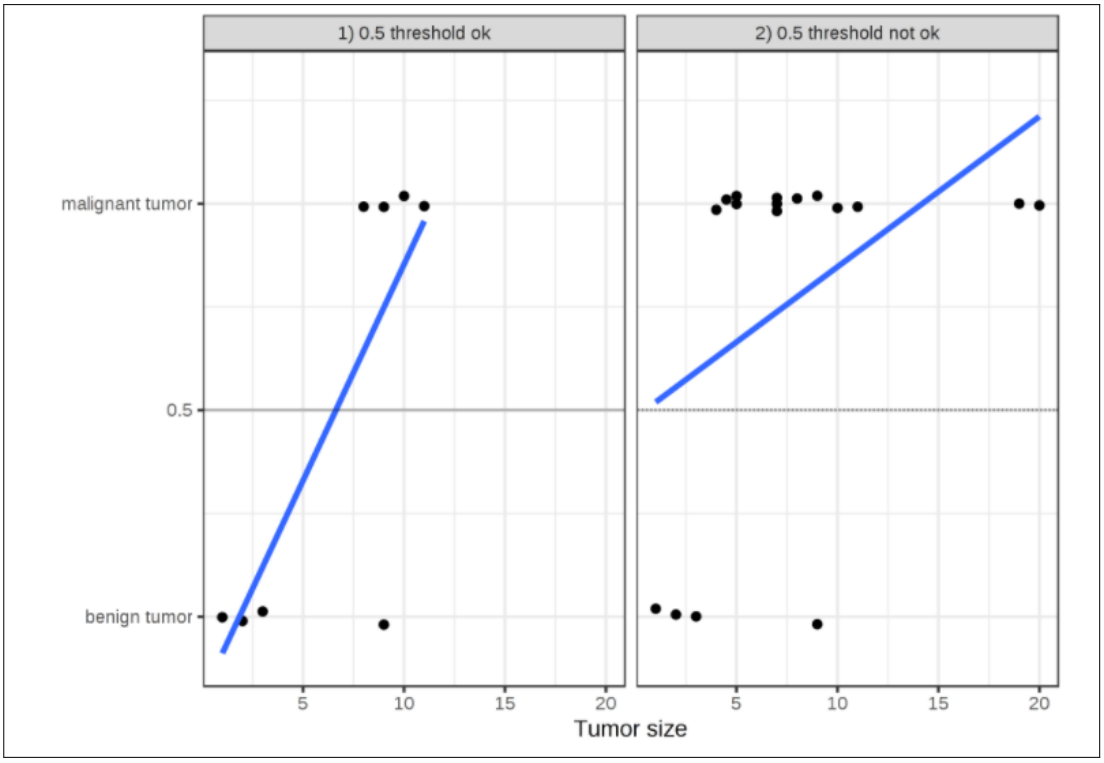


Figure 2-12: An example of a linear model failing to separate classes of tumours. [26]

As seen in Figure 8 the left-hand side of the figure creates a threshold of 0.5, and a line of best fit can be seen which successfully separates the classes. However, on the right side of the figure two more data points were added and the line transforms making the threshold of 0.5 unusable to separate the classes. Another issue with linear regression models is they can derive values above 1 and below 0 making it difficult to determine probabilities of a class being either 1 or 0. [26]

2.9.2 Multinomial Naïve Bayes

Another useful model to implement would be a Multinomial Naïve Bayes model.

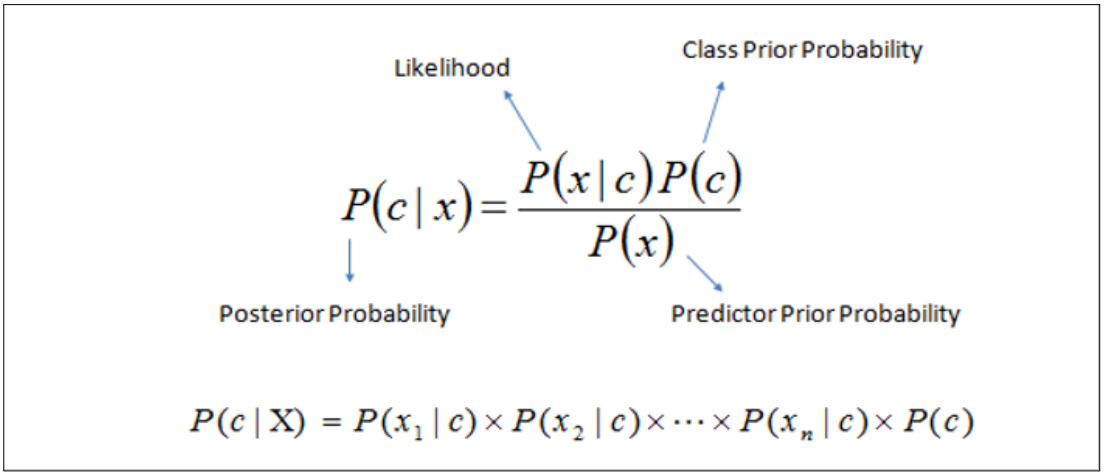


Figure 2-13: Naïve Bayes Formula [26]

The generic Naïve Bayes classifier assumes there is no relation to features based on the appearance of one. It considers prior probabilities determined when training. It then uses these initial probabilities of the words appearing in a positive sentiment multiplies it with a new probability based on how many times a word appears in the sentiment itself. After, a probability score is created which is proportional to the probability of the sentiment being either positive or negative. Whichever class has the higher probability means the sentiment will be classified to that class.

Multinomial Naïve Bayes implements a version of Naïve Bayes which can support multinomially distributed data. Multinomial Naïve Bayes is also used heavily in text classification and TF-IDF word vectors are also known to work well within this algorithm.

`MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{ym})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

Figure 2-14: Multinomial Naïve Bayes Formula [30]

Naïve Bayes is "Naïve" because it ignores all usual grammar rules. There are too many factors and rules for such an algorithm to consider. The Naïve Bayes algorithm also treats all word orders the same. This means the algorithm will have the same probability score and class probability if the sentiment is "if amazing" or "amazing if."

This is also why Multinomial Naïve Bayes is considered to have high bias but low variance. Multinomial Naïve Bayes is also considered for implementation in this project since it tends to be fairly quick and straightforward for classification classifying. [29] [30]

2.9.3 Decision Trees

Decision Trees are another supervised machine learning algorithm which will be implemented for the classification of sentiments. The goal of decision trees is to predict the value of a variable which is chosen as the target. In this case, the target variable would be either a positive or negative sentiment classification. The value will be determined by using a set of rules which are deduced from the input features of the dataset. A visual result of the algorithm shows a tree structure where the rules can be seen and the different features which can be chosen.

Decision tree mathematical formulation can be seen below:

Given training vectors $x_i \in R^n$, i=1,..., l and a label vector $y \in R^l$, a decision tree recursively partitions the feature space such that the samples with the same labels or similar target values are grouped together.

Let the data at node m be represented by Q_m with N_m samples. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ subsets

$$Q_m^{left}(\theta) = \{(x, y) | x_j <= t_m\}$$
$$Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta)$$

The quality of a candidate split of node m is then computed using an impurity function or loss function $H()$, the choice of which depends on the task being solved (classification or regression)

$$G(Q_m, \theta) = \frac{N_m^{left}}{N_m} H(Q_m^{left}(\theta)) + \frac{N_m^{right}}{N_m} H(Q_m^{right}(\theta))$$

Select the parameters that minimises the impurity

$$\theta^* = \operatorname{argmin}_\theta G(Q_m, \theta)$$

Recurse for subsets $Q_m^{left}(\theta^*)$ and $Q_m^{right}(\theta^*)$ until the maximum allowable depth is reached, $N_m < \min_{samples}$ or $N_m = 1$.

Figure 2-15: Multinomial Naïve Bayes Formula [31]

Decision Trees also considers different types of criterion parameters depending on the problem being solved. There is classification criterion and regression criterion, however, since this project is using classification them classification criterion will be analysed. There are 3 types of classification criteria which are Gini, Entropy, and Misclassification as seen in Figure 2-15.

As the Decision Tree splits new nodes are created. The decision tree needs to know when to split and create new nodes. There can be many splits based on certain information or at random. However, understanding when to split in a decision tree is critical to for performance and accuracy. The sci-kit library uses a parameter called “criterion” which either uses a method of “gini” or “entropy” to determine the most optimal split. Gini or the gini impurity will randomly label elements in a dataset. Then it will calculate the number of times any element within the dataset was wrongly mislabelled based on the initial random label assignments. Entropy will measure the amount of in information which will suggest the disorder of features within the dataset against the target. Entropy is similar to the gini impurity method since the optimal split is determined by choosing the feature which has the least entropy. [31]

If a target is a classification outcome taking on values 0,1,...,K-1, for node m, let

$$p_{mk} = 1/N_m \sum_{y \in Q_m} I(y = k)$$

be the proportion of class k observations in node m. If m is a terminal node, `predict_proba` for this region is set to p_{mk}. Common measures of impurity are the following.

Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

Entropy:

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$$

Misclassification:

$$H(Q_m) = 1 - \max(p_{mk})$$

Figure 2-16: Decision Tree criterion [31]

An example of a visual decision tree can be seen in Figure 2-17 where an animal is being classified. Traversing down the tree more rules are introduced and the output of the classification of the leaf nodes can be determined. [31]

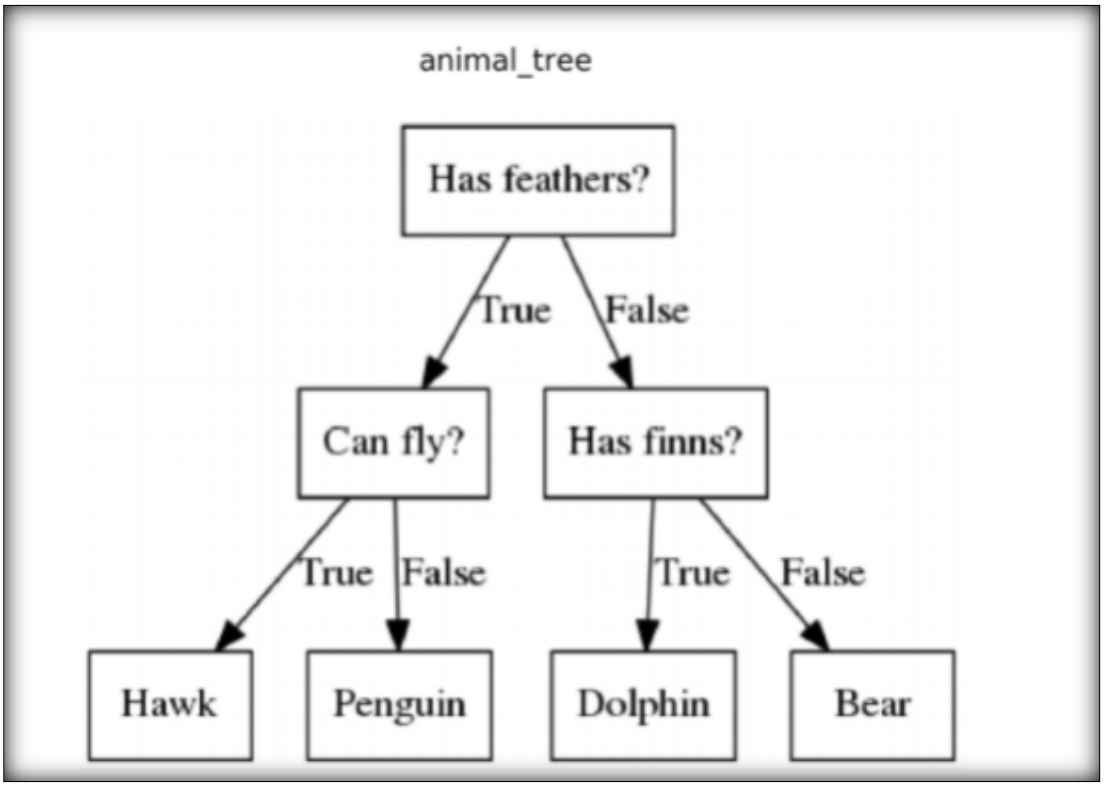


Figure 2-17: Example of an animal decision tree [32]

Benefits of using decision trees are they can handle both continuous and categorical data, does not

require intensive data pre-processing, and they have a simple visual representation. [31]

2.10 Existing SA and ASBA based Software Solutions

SA is not a new field of study by any means and has extensive research and work done in the area. However, there are not many SA software solutions which dissect into granule information past simple classification of sentiments. For example, ABSA solutions are still relatively new.

2.10.1 Master's Thesis

Similar solutions have been created with regards to social media monitoring. One solution proposed was made by Stefan Cristian Tutuianu who is a Master's student at Manchester University. The solution of the master student similarly grabbed Tweets from Twitter and used machine learning algorithms to classify the sentiments. Tutuianu then provides a graphical user interface (GUI) showcasing the average sentiment score and the number of Tweets analysed. [33]

Figure 2-18: Example of Tutuianu's UI [33]

2.10.2 Social Mention

Social mention is a SA tool which provides the user with "mentions" of certain keywords. For example, a user can search for "iPhone 12" and will be returned specific information about where that keyword was mentioned. Social Mention does a search across many social media platforms including Twitter, Instagram, and Facebook. Social mention is a paid service and only allows users a free 7-day trial. [34]

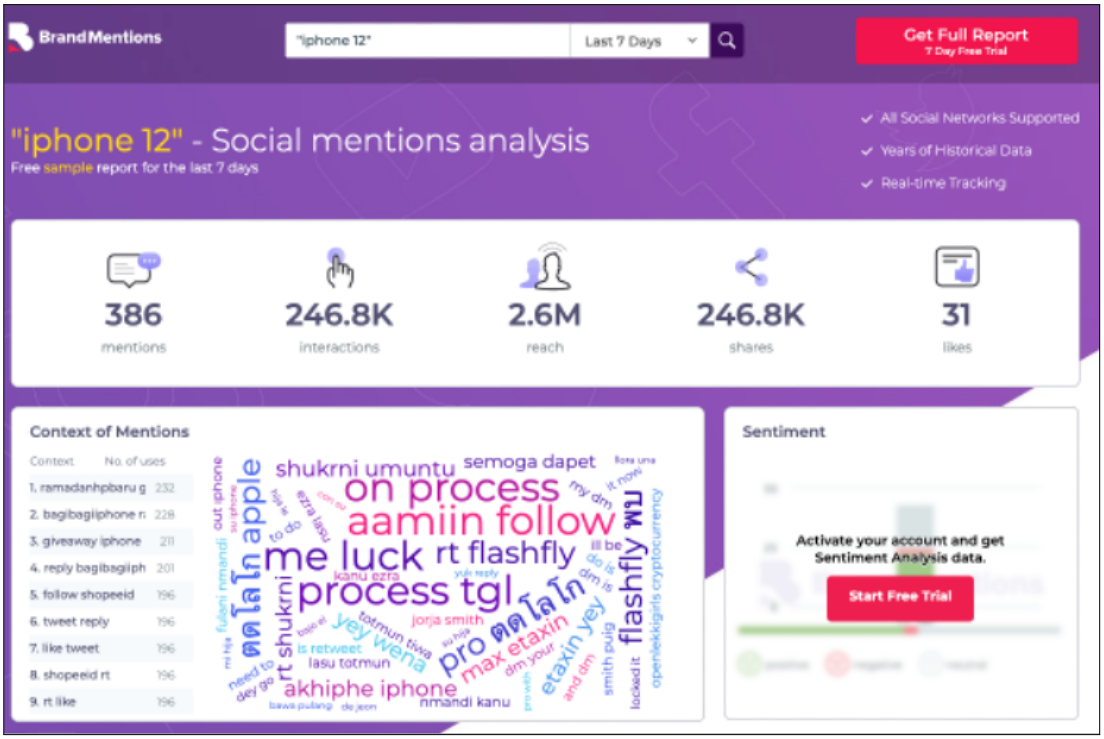


Figure 2-19: Example of Social Mention's SA tool [34]

2.10.3 Social Searcher

Similar to Social Mention, Social Searcher enables social media monitoring across a broad range of social media platforms such as Twitter, Tumblr, YouTube, Reddit, and Vimeo. Social Searcher fetches sentiment information from a much larger range of platforms. Additionally, Social Searcher provides ratios of positive, neutral, and negative sentiments for each platform. After a search is made the user can see keyword information and the number of times the keyword has appeared throughout the social media search. A benefit of Social Searcher is that all their analytical services are free. [35]

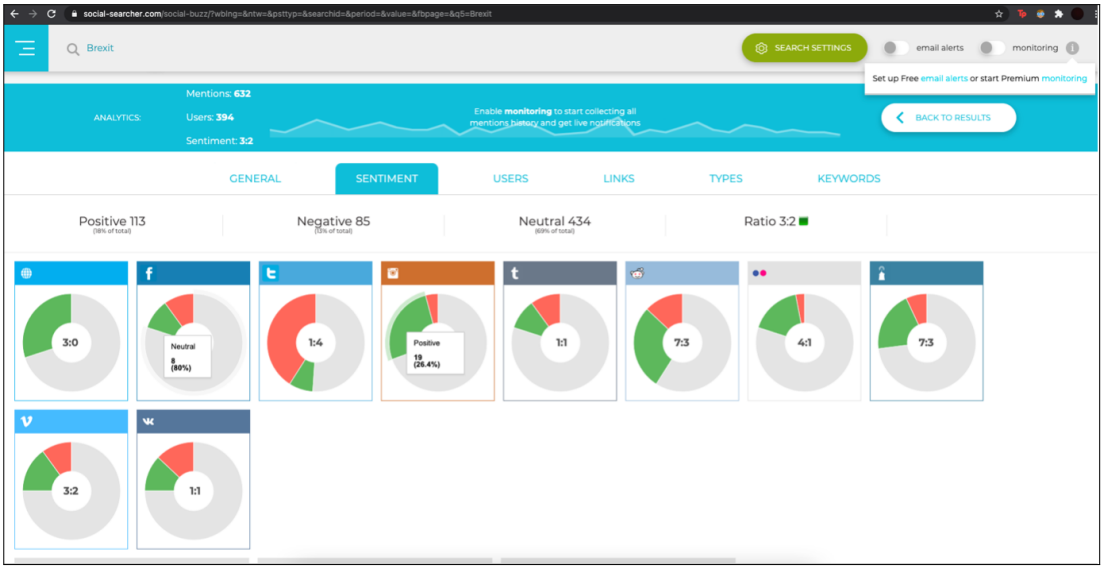


Figure 2-20: Example of Social Searcher's SA tool [35]

2.11 Proposed Solution

The current solutions mentioned are thorough and show deep analytical insights. Most of the solutions are built as web applications as well. However, there are several differences between the solutions proposed by Tutuianu, Social Mention, Social Searcher and the solution proposed in this project.

This project will allow users to preselect the number of Tweets they would like to see, and an individual classification of the Tweets will be made along with a pie chart to show the percentages of Tweets which were either classified as positive, neutral, or negative. Tutuianu and Social Mention does not enable these features. Also, this project will aim to provide ABSA on each Tweet which will show more granularity of the sentiment. These features are not present in Tutuianu or Social Mention solutions. Although keywords are derived from Social Searcher, they lack the visibility of the specific aspects extracted from the Tweet itself. For example, as seen in Figure 2-21 when clicking or viewing a sentiment the Social Searcher only extracts hashtags whilst the aim of the proposed solution is to provide the nominal subject and accompanying adjective which is much more descriptive and granular. The proposed solution provides a free unlimited use of the web application whilst Tutuianu's solution is not publicly available and Social Mention requires a subscription after the free trial period.

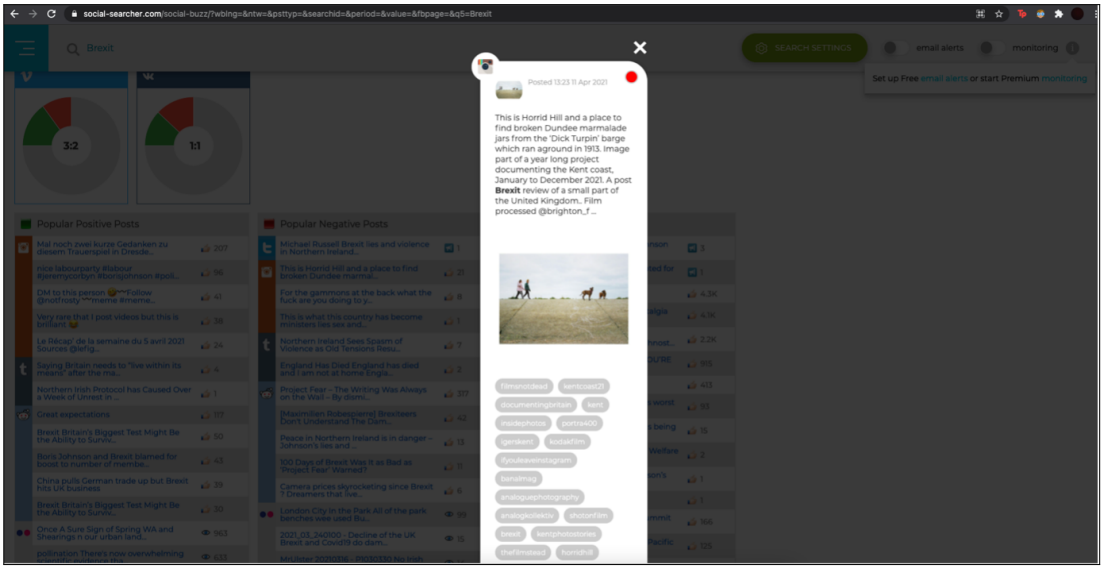


Figure 2-21: Example of Social Searcher's SA tool [35]

3 System Design

This section will cover the main system design where the first step would be an initial dataset selection. As a second step, the dataset will go through pre-processing and a thorough machine learning analysis will be conducted where several machine learning models will be produced. This will be followed by a model comparison which will be carried out to determine the best performing model for the web application. Finally, the web application requirements will be specified.

3.1 Twitter Kaggle Dataset – Sentiment140

Data analysis is a crucial step for this project has it will set the foundation for the construction of the machine learning algorithms/models. The dataset will need to be relative the topic of the project which is analysing, classifying, and extracting information from social media platforms which in this case is Twitter. Therefore, a dataset containing Twitter Tweets will need to be researched.

A dataset containing Tweets has been chosen for this project and is named Sentiment140 dataset with 1.6 million Tweets. The Tweets have been gathered from the official Twitter API. A large dataset was chosen because it can help training data in machine learning models be less prone to margins of error, provides more accurate means values, and the chance to identify outliers which could potentially skew the data. [36]. In addition, having more data for the machine learning algorithm can help the algorithm learn many different cases of positive or negative sentiments. Considering live Tweets from Twitter can be ambiguous or contain many different types of sentence structures, the ML model will benefit if it can understand all the differences when it comes to predicting and classifying. [37]

The dataset contains 6 fields or inputs:

Target: shows the polarity of the Tweet which is either 0 or 4. 0 represents negative and 4 represents positive.

Ids: represent the unique ID of a Tweet in the dataset.

Date: represents the date of the Tweet.

Flag: represents if the Tweet was part of a query.

User: represent the unique user ID of the Twitter user.

Text: represents the Tweet.

Using a part of the CRISP-DM methodology of data exploration it is important to remove unnecessary columns as this can affect the accuracy of the ML models. For example, the Ids, Date, Flag, and User fields have no effect on the Tweet being positive or negative. As they are redundant fields, we can remove them leaving the Target and Text columns. There is a text column in the dataset which represents a positive Tweet as '4' and a negative Tweet as '0.' After a quick analysis a better representation would be 'negative' for negative and 'positive' for positive. Changing the text representation will not have any performance impacts, but it will make it easier to read and classify within the code. Figure 3-1 shows the "loadDataset()" function which will handle all of the corrections mentioned to the dataset. [37]

```
def loadDataset():
    columns  = ["sentimentScore", "id", "date", "no_query", "userName", "text"]
    encoding = "ISO-8859-1"

    # load tweet dataset into pandas dataframe
    dataFrame = pd.read_csv('training.1600000.processed.noemoticon.csv', encoding = encoding, names = columns)

    # remove unnecessary columns - feature selection
    dataFrame = dataFrame.drop('id', 1)
    dataFrame = dataFrame.drop('date', 1)
    dataFrame = dataFrame.drop('no_query', 1)
    dataFrame = dataFrame.drop('userName', 1)

    # convert sentiment values from 0/1 to negative/positve
    dataFrame['sentimentScore'] = dataFrame['sentimentScore'].map({0:'negative', 1:'positive'})

    #even distribution of positive and negative sentiments
    print(dataFrame['sentimentScore'].value_counts())
```

Figure 3-1: The "loadDataset()" function which will contain the code for loading the dataset

The dataset has been analysed and is now ready for the pre-processing phase.

3.2. Pre-processing and using NLP techniques

As mentioned before NLP is Natural Language processing and it is crucial for data pre-processing. This phase is necessary in order to clean the data to get rid of unnecessary information which can affect the accuracy of a prediction. For example, when classifying sentiment, the use of certain words, punctuation, emojis, and other noise in the data will create less effective ML models which may incorrectly predict classifications. Several techniques and functions will be used on the dataset in order to prepare the data for the machine learning algorithm implementation. It is important to note that the pre-processing stage can be revisited as different techniques can produce better accuracy in some models whilst decreasing accuracy in others.

Current dataset preview of first 20 rows:

```
print(dataFrame.head(20))
```

```
   sentimentScore                                               text
0        negative  @switchfoot http://twitpic.com/2y1zl - Awww, t...
1        negative  is upset that he can't update his Facebook by ...
2        negative  @Kenichan I dived many times for the ball. Man...
3        negative    my whole body feels itchy and like its on fire
4        negative  @nationwideclass no, it's not behaving at all....
5        negative                      @Kwesidei not the whole crew
6        negative                                        Need a hug
7        negative  @LOLTrish hey  long time no see! Yes.. Rains a...
8        negative              @Tatiana_K nope they didn't have it
9        negative                          @twittera que me muera ?
10       negative       spring break in plain city... it's snowing
11       negative                         I just re-pierced my ears
12       negative  @caregiving I couldn't bear to watch it.  And ...
13       negative  @octolinz16 It it counts, idk why I did either...
14       negative  @smarrison i would've been the first, but i di...
15       negative  @iamjazzyfizzle I wish I got to watch it with ...
16       negative  Hollis' death scene will hurt me severely to w...
17       negative                                about to file taxes
18       negative  @LettyA ahh ive always wanted to see rent  lov...
19       negative  @FakerPattyPattz Oh dear. Were you drinking ou...
```

Figure 3-2: The output of the cleaned text

3.3 Syntax Cleaning

The following section will cover the NLP processes and various pre-processing techniques.

Each Tweet will be converted to lowercase. This because there are many occasions in the English language where certain words have to be capitalised such as pronouns or certain entities. When converting to lowercase the entire training set will become simpler and removes the need to distinguish words which are capitalised.

Many Tweets will contain links to various images, videos, or websites. Links do not provide meaning and adds unnecessary noise to the data, therefore links will be removed from each Tweet.

Removing punctuation is important since punctuation is used in most of the syntax within a Tweet. This means each Tweet contains a large amount of noise and removing the noise will return Tweets with more meaning. The following piece of code in Figure 3-3 will achieve this.

Usernames on Twitter are represented by a handle which always begins with an "@" symbol. For example, Tim Cook's Twitter handle is represented as "@tim_cook." Many Tweets contain the handle, and the first iteration of pre-processing will try and replace each "@" handle with "user." The reason for this is so that when the ML algorithms try to give a value, weight, or score to a particular word the "user" which is a common word to represent any user on Twitter. Emojis are also common within Tweets but they will also be removed since they do not contain any significant meaning.

words are words which contain minuscule amounts of meaning. Therefore, removing stop words can improve performance and accuracy.

Lemmatisation is the process of normalising the data. Normalising text data is trickier than continuous values since there is a lot of variation. However, the most effective processes to normalise text data would be through lemmatisation and stemming. Lemmatisation helps to normalise by modifying a word to its base form. Lemmatisation is preferred over stemming since lemmatisation considers the morphological context of the word whilst stemming does not.

Figure 3-3 shows the "preProcess()" function which will execute all of the above pre-processing techniques. As the dataset contains 1.6 million rows, a lambda function has been implemented to apply the "preProcess()" function to each Tweet with better performance than an ordinary for-loop expression. The cleaned text is replaced into the same "text" column within a Pandas data frame.

Figure 3-3: The "preProcess()" function which will contain the code for pre-processing the dataset

3.4 Machine Learning Model Analysis with Word2vec and TF-IDF

This section will cover the algorithms and models being implemented along with the performance and accuracy of each model. Each model will compare the results of using a Word2vec and TF-IDF technique mentioned previously. A model can only implement either a Word2vec or TF-IDF technique. The final algorithm chosen needs to achieve an accuracy of 80% or higher as this is one of the main projects aims. Other performance metrics will be investigated such as F1-score, TPR, FPR, and hyper-parameter tuning.

3.5 Splitting Training and Testing/Validation Data

Splitting the pre-processed data into training and testing sets is an important step to prepare the data for ML models. The training data allows the ML model to learn the mappings of inputs to outputs and fit the model. The testing data is used to show the evaluation of the fit of the data to the model. The training data is the independent variable which in this case would be the sentiments or Tweets. The testing data would be the corresponding labels to the sentiments which is either positive or negative. However, Word2vec and TF-IDF will use different data when splitting the data.

This is because Word2vec will transform the "text" column into numerical vectors and TF-IDF will use the string representation of the "text" column. Word2vec will be using the vector representation of the sentiment or Tweets as training data and will use the "sentimentScore" from the data frame as the testing data. TF-IDF will use the "text" column as the training data and "sentimentScore" as the testing data. There will be 4 main categories which will be created when using the "train_test_split" function from the sci-kit learn library which is "X_train", "Y_train", "X_test", and "Y_test". "X_train" is going to hold most of the training data and will be used for model fitting. "Y_train" is going to be used during training and will act as the labels for "X_train. "X_test" is going to be a smaller subset of "X_train" and will be used to make predictions after model fitting. Finally, "Y_test" will be used to validate the predictions made with "X_test."

The Word2vec transformation and TF-IDF and Vectorizer Transformation sections will clarify this in more detail with figures.

3.6 Word2vec Transformation

Using the data from the pre-processing phase a Word2vec model can be created. A very useful library called spaCy will assist in creating word vectors. spaCy is an NLP python library which has various components of a succinct pipeline including a Part-of-speech tagger, Named Entity Recognition, and a Tokeniser. spaCy can load these components quickly making it a very efficient and versatile library. The first step would be to tokenise the text and create a vector from the text. To illustrate, the following code snippet passes the text "I love pizza" into the generateVector function as seen in

Figure 3-4.

```
def generateVector(text):
  token = nlp(text)
  vector = token.vector
  return vector

example_vector = generateVector("I love pizza")

print(example_vector)
```

Figure 3-4: code which will generate a vector for the sentence “I love pizza”.

The output of the above code would be:

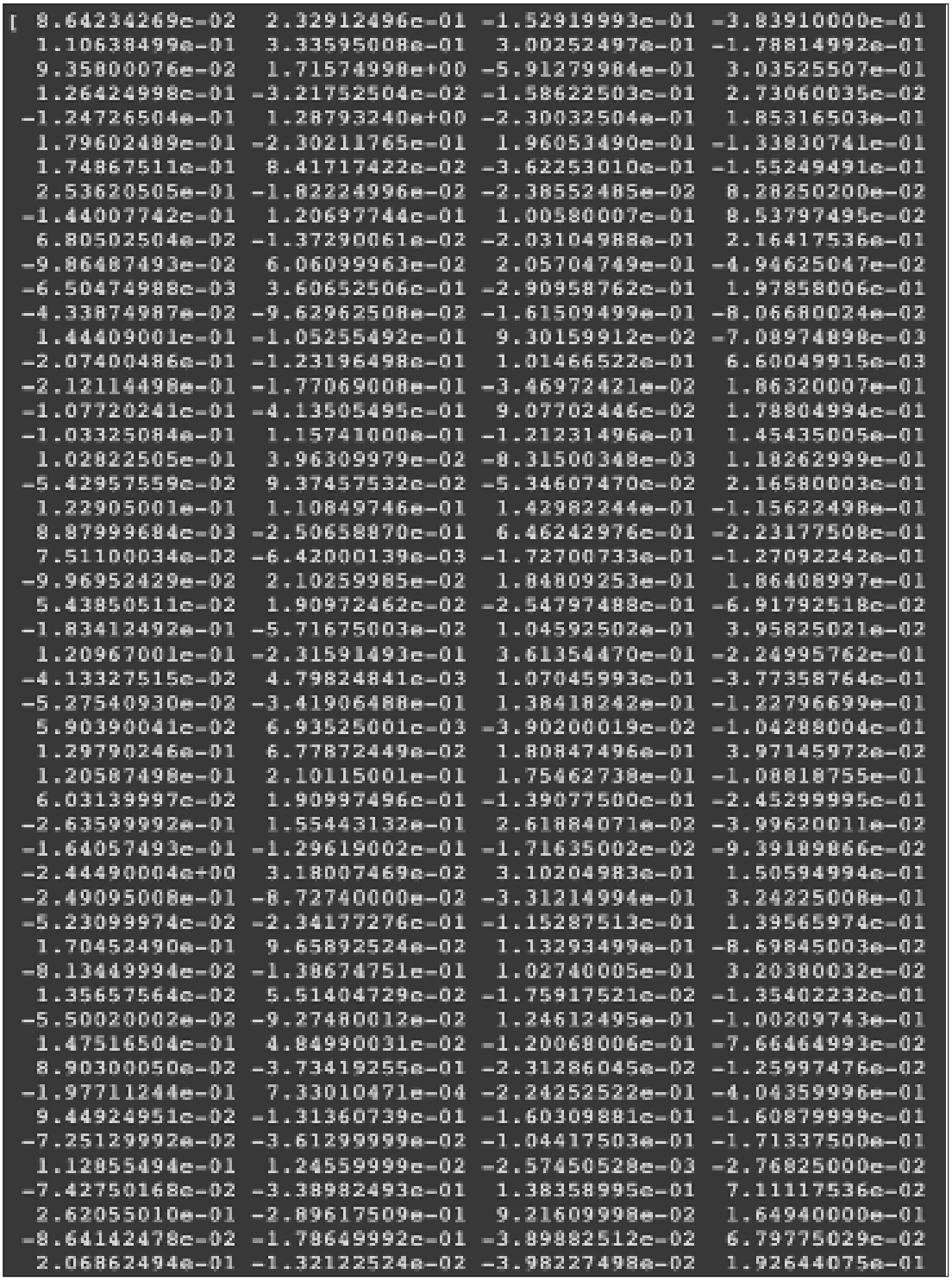


Figure 3-5: Shows the numerical output of generating a vector for "I love pizza."

```
 2.06862494e-01 -1.32122524e-02 -3.98227498e-02  1.92644075e-01
 3.89399976e-02 -8.87510031e-02 -8.26405063e-02  1.56273246e-01
-1.32087007e-01 -6.72969893e-02  1.13910995e-01  6.91442490e-02
 1.14964001e-01 -2.05405265e-01 -7.60597438e-02  3.04495007e-01
 7.21224993e-02  6.08605258e-02 -4.64803241e-02  3.51625010e-02
-8.82519931e-02 -3.63982506e-02 -2.12384999e-01  3.62712555e-02
-5.71599938e-02 -2.37654999e-01 -2.06022501e-01 -2.54877508e-02
 4.88299914e-02 -1.42701253e-01 -9.09230039e-02 -1.95516020e-01
-2.23727494e-01 -8.68972540e-02 -1.33336738e-01  1.87489986e-01
 1.44785002e-01  1.28150746e-01 -1.33224756e-01  1.40280008e-01
 1.48450002e-01 -6.12725057e-02 -1.45399906e-02 -5.11349998e-02
-1.97524011e-01  2.06242517e-01  2.38807499e-01 -1.54217497e-01
-2.50224978e-01 -7.91417509e-02  1.47349998e-01  2.06172511e-01
 8.37427452e-02 -6.55589998e-02 -2.79249996e-01 -2.28384003e-01
-9.40699503e-03  1.28916740e-01 -1.27742514e-01 -1.32355005e-01
-1.03372745e-01  5.47299907e-02 -8.03747475e-02  1.61596254e-01
-3.89874876e-02  2.63067484e-01 -3.74833465e-01  3.30099985e-02
-3.03100526e-01  4.78982478e-02 -2.35397011e-01  1.06673419e-01
-8.07175040e-03 -1.11069255e-01  1.39954358e-01  2.92042494e-01
 2.72329986e-01 -2.62745261e-01 -9.57124680e-03 -1.19110756e-01
-4.49017510e-02  2.50328749e-01  3.90162528e-01  1.48519993e-01
-3.50525081e-02  1.97962765e-02  2.17014998e-01  6.56127557e-02
-2.33104751e-01  5.16210571e-02  4.47075069e-03 -1.38203248e-01
 6.16277531e-02 -7.47669935e-02  6.47950172e-03  3.20374995e-01]
```

Figure 3-6: Shows the numerical output of generating a vector for "I love pizza."

Figure 3-5 and Figure 3-6 shows an example array which was created when generating a vector for a simple sentence. However, a vector representation needs to be generated for each of the 1.6 million Tweets in the pre-processed data frame named "dataFrame" in Figure 3-7. Considering the enormous amounts of vectors needing to be generated a lambda expression was created to generate the vectors more efficiently than a for-loop expression. After the vectors were generated, the resulting output was saved into a new column called "vectorArray" and this will act as the "x_train" and "x_test" data. The final data frame is saved into a pickle file called "w2vUpdate.pkl."

```
# uses spacy to generate a token object from the pipeline and then creates a word vector array for each sentiment
# tokenization will occur as spaCy handles this in one of the pipeline components
def generateVector(text):
  token = nlp(text)
  vector = token.vector
  return vector

# create a new column in the dataframe and add the vector array
dataFrame['vectorArray'] = dataFrame['text'].apply(lambda text: generateVector(text))

# saving word2vector transformation to pickle file
dataFrame.to_pickle('w2vUpdate.pkl')
```

Figure 3-7: The code which shows the new "vectorArray" being created from the "generateVector()" function

The entire process was approximately 30 minutes. The Word2vec model is now complete and ready to be integrated into the ML algorithms. A sample of the first 20 rows of the Word2vec data frame can be seen in Figure 3-8 below.

```
print(dataFrame.head(20))
```

```
   sentimentScore  ...                                        vectorArray
0        negative  ...  [0.005911498, 0.11861346, -0.12657002, -0.0240...
1        negative  ...  [-0.097391695, 0.15376349, -0.13821003, -0.015...
2        negative  ...  [0.059114683, 0.0039304197, -0.11442397, -0.03...
3        negative  ...  [-0.045068603, 0.1058173, -0.1635969, 0.064762...
4        negative  ...  [-0.07272821, 0.10524147, -0.1634731, -0.07032...
5        negative  ...  [0.043939997, -0.064748004, -0.14207941, 0.016...
6        negative  ...  [-0.03159733, -0.057513673, -0.33823332, 0.256...
7        negative  ...  [-0.023867836, 0.13468497, -0.17468788, 0.0006...
8        negative  ...  [-0.03903951, 0.06606667, -0.16810285, -0.2153...
9        negative  ...  [0.038946398, -0.051136, -0.312792, 0.14818001...
10       negative  ...  [0.20994298, 0.19566716, 0.027414862, -0.02134...
11       negative  ...  [0.044541597, 0.17533001, -0.348762, -0.050676...
12       negative  ...  [0.021900246, 0.10879738, -0.23240496, 0.06462...
13       negative  ...  [-0.035812344, 0.22628139, -0.28923357, -0.084...
14       negative  ...  [0.03425105, 0.10765895, -0.18037571, -0.08893...
15       negative  ...  [0.034985594, 0.1949752, -0.25240734, -0.06312...
16       negative  ...  [-0.07921342, 0.09707091, -0.0979074, -0.02006...
17       negative  ...  [-0.16098349, 0.089897506, -0.2354975, -0.1238...
18       negative  ...  [0.11584723, 0.099665925, -0.22821, -0.1086704...
19       negative  ...  [-0.006802831, 0.022217408, -0.18881343, -0.09...
```

Figure 3-8: Example of the output of the first 20 rows after generating word vectors for the sentiments

3.7 TF-IDF, Vectorizer Transformation, and Hyperparameter Tuning

Using the data from the pre-processing phase, a TF-IDF vectorizer can be created. The initial step is to import the training data from the pickle file. The next step would be to split the data into training and testing sets. After, a "Tfidfvectoriser" import from the Sci-kit learn library will be initialised. There are several parameters when initialising the TF-IDF vectorizer as seen in Figure 3-9. The parameters used were ngram_range, max_df, min_df, smooth_idf, use_idf, and max_features. Hyper-parameter tuning needs to be conducted and the best possible accuracy of the TF-IDF vectorizer must be achieved. The following section will go into more detail of tuning parameters and the various comparisons of accuracies.

```
# initialise the tfidf vectorizer
tfidf_vectorizer = TfidfVectorizer(max_df=1.0, min_df=0, ngram_range=(1,4), smooth_idf=True, use_idf=True, max_features = None)
```

Figure 3-9: Code which initialises the TF-IDF vectorizer with selected parameters

3.7.1 N-gram parameter tuning

An n-gram range is the characterisation of transforming the features of the words of a sentence or piece of text into a series of words. The n-gram range can usually consist of unigrams, bigrams, and trigrams. However, four-grams and five-grams also exist. An example can be created out of the sentence "Eating an apple a day keeps the doctor away." A unigram from the aforementioned sentence can be "eating" and a bigram can "eating an." A trigram would be three consecutive words such as "apple a day." The purpose of creating these n-grams is that the vectorizer can create more complicated groupings of words instead of relying on the context of one word or just a "unigram." Utilising different n-gram ranges can also improve the accuracy of a ML model.

The "Tfidfvectoriser" import from the Sci-kit library can have multiple n-gram range inputs so it is important to understand the best input range. The parameter allows defining an n-gram range as (min_n, max_n) where min_n ¡= n ¡= max_n. [38] Figure 3-10 and Figure 3-11 and Figure 3-11 compares the best min_n input ranges. As seen when looking at the TF-IDF accuracy scores the n-gram range of (1,4) yielded the highest accuracy of 81%. Also, the n-gram range of a min_n greater than 0 is more likely to produce a higher accuracy. [38]

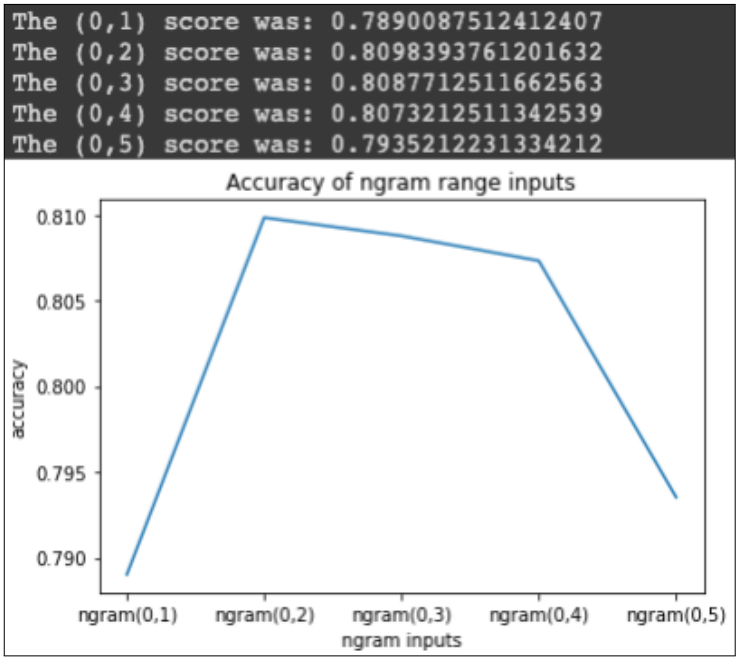


Figure 3-10: Graph showing the accuracy of varying ngram input

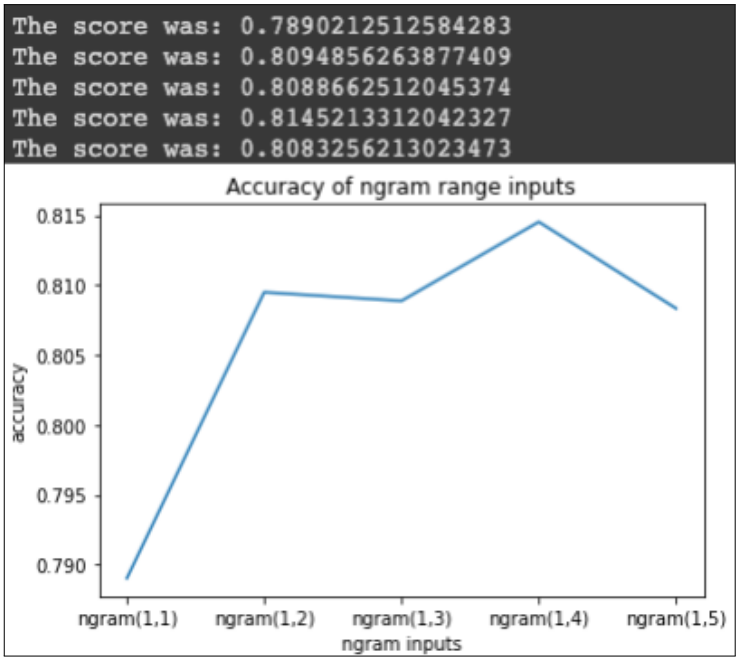


Figure 3-11: Graph showing the accuracy of varying ngram inputs

3.7.2 Max_df/Min_df

Max_df refers to the frequency of words which appear often or with high frequency. For example, if max_df = 0.75 then this means terms which appear in more than 75% of the documents or in our case Tweets will be ignored. On the other hand, min_df is a parameter which removes words which do not appear often or with low frequency. If min_df = 0.1 then this means the vectorizer will ignore words which appear in less than 1% of the total Tweets in the dataset. The max_df has been set to 1 which means they will not ignore any terms. The min_df has been set to 0 which means it will not ignore any terms which appear in less than 0% of documents. Using the "cross_val_score()" function within the sci-kit learn library the best parameter can be tested and comparing the results for different "max_df" inputs as seen in Figure 3-12. [38]

The inputs which were used for comparison for max_df were: [0.25, 0.5, 0.75, 1.0, 1.5]. The inputs which were used for comparison for min_df were: [0, 0.01, 0.05, 0.1, 0.25, 0.3].

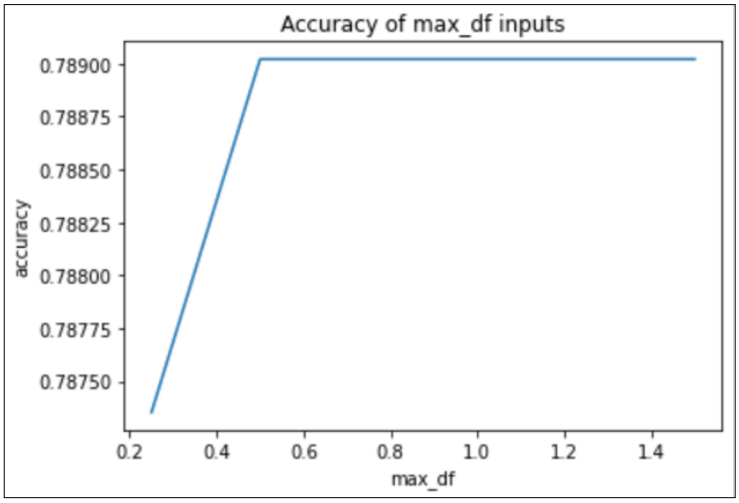


Figure 3-12: max_df parameter input comparison

Similarly, Figure 3-13 illustrates the same function used to find the best performing min_df and 0 was chosen.

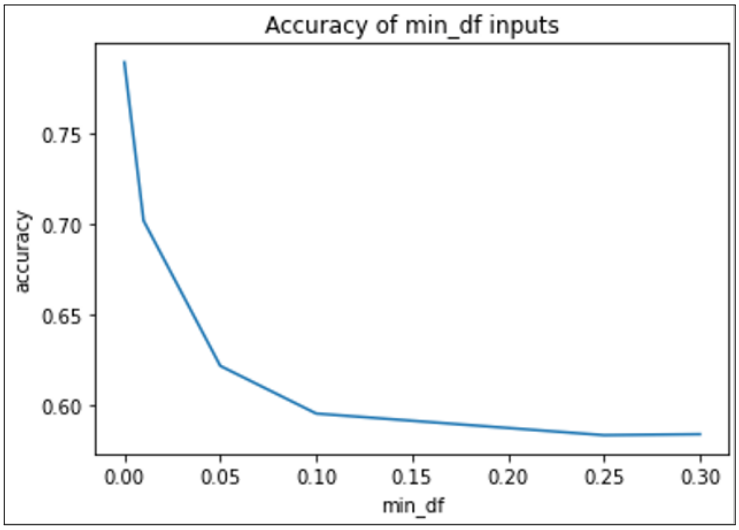


Figure 3-13: Min_df parameter input comparison where 0 was chosen as the input parameter

3.7.3 Smooth_idf/Use_idf

Smooth_idf is a parameter which will help prevent zero divisions and adds one to document frequencies. The significance is that if there is a word that is in training data but not in the test, then the unseen word within the test will still be processed instead of an error being thrown. The default setting for this parameter is true and will be initialised this way. Use_idf controls weather inverse document frequency will be used. The default for this will be true as using inverse document frequency is one of the aims of this project.

3.7.4 Max_features

The max features parameter decides how to create a feature matrix. If set to 20,000 for example, then a feature matrix of the most frequent 20,000 words amongst all the documents will be used. However, in Figure 3-14 the inputs of various inputs for max_features were compared, and the accuracy stays the same at approximately 79% when increasing from 0. However, even when "None" was passed the accuracy remained at 79%. Therefore, "None" will be used for max_features.

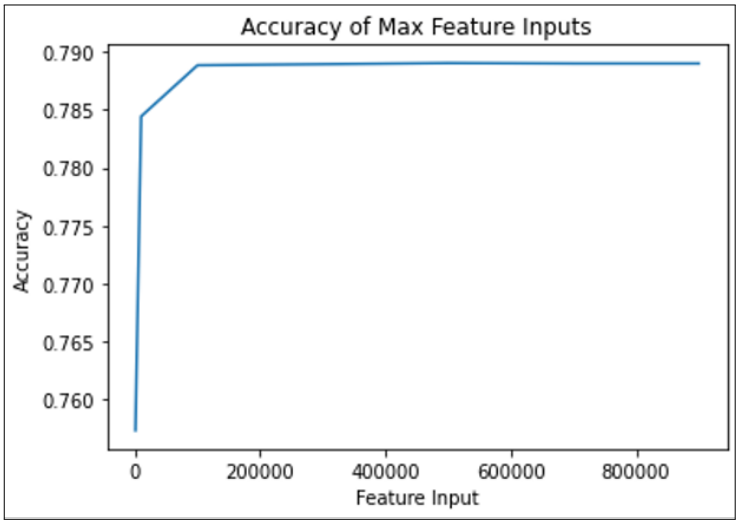


Figure 3-14: Max_features parameter input comparison where None was chosen as the input parameter

3.7.5 Final Vectorizer accuracy

The following code in Figure 3-15 shows the Tfidfvectoriser being initialised with all the tuned hyper parameters.

```
tfidf = TfidfVectorizer(ngram_range=(1,4), min_df=0, max_df=1, use_idf=True, smooth_idf=True max_features=None)

tfidf_x = tfidf.fit_transform(trainX)

# cross val score/ predict
tfidf_score = cross_val_score(lr, tfidf_x, validationY, cv=3)
print(tfidf_score.mean())
```

Figure 3-15: The code which initialises the TF-IDF vectorizer with the chosen parameter's and calculates the accuracy of the vectorizer.

The output of the vectorizer accuracy was 0.8114118763166488 or 81% rounded. Although, the accuracy of the ML model will depend on how well the transformed data from the vectorizer fits onto an ML model.

3.7.6 Transforming the Data

Now that the hyper-parameters for the TF-IDF Vectorizer have been determined, the training data or "train_x" will then be passed into the "fit_transform" function which will create TF-IDF scores for the training data. After, the "x_test" otherwise known as "x_tfidf_val" will be transformed as well.

```
trainX =  dataFrame['text']

validationY = dataFrame['sentimentScore']

x_tfidf_train, x_tfidf_val, y_tfidf_train, y_tfidf_val = train_test_split(trainX, validationY, test_size = 0.15, random_state = 2, stratify=validationY)

# initialise the tfidf vectorizer
tfidf_vectorizer = TfidfVectorizer(max_df=1.0, min_df=1, ngram_range=(1,2), smooth_idf=True, use_idf=True, max_features = 40000)

# fit the vectoriser and then transform the x train data
x_tfidf_train = tfidf_vectorizer.fit_transform(x_tfidf_train)

# only transform the x test data
x_tfidf_val = tfidf_vectorizer.transform(x_tfidf_val)
```

Figure 3-16: The code which initialises the TF-IDF vectorizer with the chosen parameter's and transforms the training data

Finally, the training and testing data has been split, and TF-IDF scores have been generated. The data is now ready to be fitted onto the ML models.

3.8 Logistic Regression Model

Logistic Regression is known to be great for classification tasks. In order to create a Logistic Regression model in Python and fit the model, a logistic regression sci-kit learn library will be imported and initialised.

3.8.1 Hyperparameter Tuning for Logistic Regression

There are several parameters which can be configured upon initial initialisation such as C, n_jobs, and max_iter.

C determines the inverse regularisation strength. Regularisation is using a penalty to reduce the chance of overfitting. C must be a positive integer and if C is kept small then the regularisation will be stronger. However, after testing a value of 4 for C gave the optimal accuracy for this model. N_jobs specifies the number of CPU cores which will be used during the training phase. The default is 1 which means it will just use 1 core processer. However, using n_jobs = -1 will enable the solver to use all available CPU processors. The final parameter is max_iterations which specifies the number of iterations to be set before the algorithm solvers reach a point of convergence. After testing various values, the optimal number iterations chosen was 850. This is because the logistic regression classifier will always converge before the 850th iteration. [39]

3.8.2 Fitting Transformed Data onto Model

After the parameters are selected the "x_train" and "y_train" will be passed into the "fit" method as seen in Figure 3-17. The "fit" method trains a logistic regression classifier and tries to reach a point of convergence. Convergence is when a regression coefficient is meaningful during the iterative process of finding a solution. The iterative process tries to use coefficients to maximise the maximum likelihood estimation. Many times, a logistic regression algorithm may not converge. However, this usually happens if there are too many predictor variables in the dataset and there are only a small number of classes to predict. Another reason may be multicollinearity. Multicollinearity is when the predictor variables are highly correlated. Having high collinearity means the regression coefficients is very high when comparing two predictor variables, for example. this also implies that the predictor variable may be redundant and can be removed as it will not have significance in determining a classification. In this case, there are a very small number of predictor variables so multicollinearity will not be an issue.

Once convergence or the max iterations parameter is satisfied then algorithm will cease execution. After, the "predict" function will be called and the "x_test" data will be passed in which will result in the predictions being made. A confusion matrix which classifies the True-positive, true-negative, false-positive, and false-negative predictions can be visualised as shown in Figure 3-19.

3.8.3 Logistic Regression TF-IDF

```
import pickle
from sklearn.linear_model import LogisticRegression

# initialise the logistic regression classifier
log_model = LogisticRegression(C=4, random_state = 0, verbose = 2, max_iter = 850, n_jobs=-1, solver = 'lbfgs')

# fit the the logistic regression classifier on the x train and y train data
log_model.fit(x_tfidf_train, y_tfidf_train)

predictions = log_model.predict(x_tfidf_val)

conf = confusion_matrix(y_tfidf_val, predictions)
print(conf)

# use y-validation set to determine the correct amount of classifications based on predictions
model_metrics = classification_report(y_tfidf_val, predictions)
print(model_metrics)

#saving the logistic regression model
pickle.dump(log_model, (open('LR.pkl', 'wb')))
```

Figure 3-17: Shows the code which initialising the logistic regression model and fitting the TF-IDF transformed data onto the model and printing the performance metrics

	precision	recall	f1-score	support
negative	0.81	0.85	0.83	120000
positive	0.84	0.79	0.82	120000
accuracy			0.82	240000
macro avg	0.82	0.82	0.82	240000
weighted avg	0.82	0.82	0.82	240000

Figure 3-18: Shows the performance metric output of the TF-IDF Logistic Regression model fit

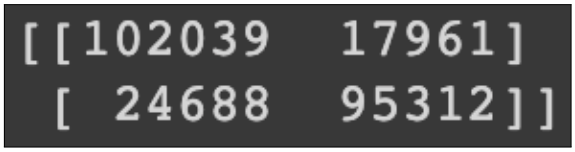


Figure 3-19: Shows the output of confsuion matrix for the TF-IDF model fit

The final accuracy was 82% for Logistic Regression. The training for this model only took approximately 30 minutes which performed very well.

3.8.4 Logistic Regression Word2Vec

The Logistic Regression model will now be fit onto the Word2Vec trained data similar to the TF-IDF implementation. The hyper-parameters were configured the same as TF-IDF. Figure 3-20 shows the performance metrics of the model based on its predictions.

```
CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  5.2min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  5.2min finished
              precision    recall  f1-score   support

    negative       0.76      0.76      0.76    119868
     postive       0.76      0.76      0.76    120132

    accuracy                           0.76    240000
   macro avg       0.76      0.76      0.76    240000
weighted avg       0.76      0.76      0.76    240000
```

Figure 3-20: Shows the accuracy of the Word2Vector model which was at 76%

The Word2Vector transformed data trained on the Logistic Regression model results in an accuracy of 76% which is slightly less accurate than the TF-IDF transformed Logistic Regression model.

3.9 Multinomial Naïve Bayes Model

Similar to Logistic Regression sci-kit learn has a model library for Multinomial Naïve Bayes.

3.9.1 Hyperparameter Tuning for Multinomial Naïve Bayes

The parameters for this model are alpha, fit_prior, and class_prior. However, the default value of fit_prior is set to true and will be retained as fit_prior will try and learn the class probabilities beforehand which is beneficial since there are only two classes. Class_prior has a default value of "None" and will also be retained as adjusting priors before training can lead to less accurate model classifications. [40]

Alpha represents the Laplace or Lidstone smoothing parameter. A thorough test comparing various alpha values can be seen in Figure 3-21. An alpha of 1 returns an accuracy of 80% which is much higher than any other alpha inputs. Also, the higher alpha value chosen shows a trend of decreasing accuracy. [40]

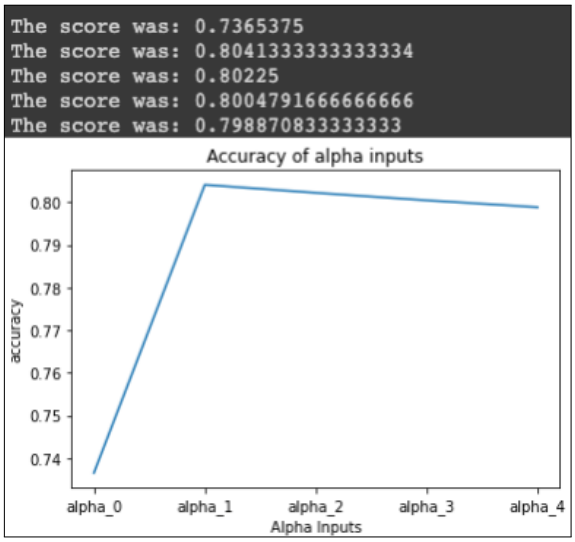


Figure 3-21: Comparison of alpha values where an alpha of 1 yields the highest accuracy.

3.9.2 TF-IDF for Multinomial Naïve Baye

After tuning the hyperparameters the Multinomial Naïve Bayes Classifier is initialised and the TF-IDF transformed data is fitted onto the model.

```
# initialise the logistic regression classifier
MB = MultinomialNB(alpha=1, fit_prior=True, class_prior=None)

# fit the the logistic regression classifier on the x train and y train data
MB.fit(x_tfidf_train, y_tfidf_train)

predictions = MB.predict(x_tfidf_val)

confusion_matrix = confusion_matrix(y_tfidf_val, predictions)
print(confusion_matrix)
```

Figure 3-22: Shows the TF-IDF data being fit onto the Multinomial Naïve Bayes Model

Figure 3-23 shows the confusion matrix and accuracy of the classifier after the predictions were made. The overall accuracy of this model was 80% which was slightly less performant that the Logistic regression TF-IDF model, however, the performance of this model when training was much quicker.

```
[[101879  18121]
 [ 28887  91113]]
              precision    recall  f1-score   support

    negative       0.78      0.85      0.81    120000
    positive       0.83      0.76      0.79    120000

    accuracy                           0.80    240000
   macro avg       0.81      0.80      0.80    240000
weighted avg       0.81      0.80      0.80    240000
```

Figure 3-23: Confusion matrix output for the Multinomial Naïve Bayes Model fit

3.9.3 Word2vec for Multinomial Naïve Bayes

The Word2Vec models transforms the data into numerical vectors as seen in the previous sections. The numerical values can be negative, however, for the Multinomial Naïve Bayes values cannot be negative. This is because Multinomial Naïve Bayes assumes the features will have a multinomial distribution and a multinomial distribution cannot contain a negative value. Figure 3-24 highlights the value error which gets thrown. Due to this the Multinomial Naïve Bayes Model will not be created and will result in N/A shown in Section 3.11 Final Model Comparisons.

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-f869ee03c7ad> in <module>()
     36 print("beginning fit")
     37 # fit the the multinomial naive bayes classifier on the x train and y train data
---> 38 MB.fit(x_w2v_train, y_w2v_train)
     39
     40 predictions = MB.predict(x_w2v_val)

                              2 frames
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py in check_non_negative(X, whom)
    992
    993     if X_min < 0:
--> 994         raise ValueError("Negative values in data passed to %s" % whom)
    995
    996

ValueError: Negative values in data passed to MultinomialNB (input X)

SEARCH STACK OVERFLOW
```

Figure 3-24: Error being thrown from negative input being passed into the Multinomial Naïve Bayes model

3.10 Decision Tree Model

Sci-kit learn has a Decision Tree model library called "DecisionTreeClassifier."

3.10.1 Hyperparameter Tuning for Decision Trees

The parameters available for this classifier include criterion and max_depth. Max_depth accepts a positive integer value. For criterion there are two options to choose from which is "gini" and "entropy."

The max_depth parameter of the Decision Tree Classifier will determine the depth of the final tree. To demonstrate this idea Figure 3-25 will be referenced. Figure 3-26 shows a tree with a depth of

2. Each time there is a split for a rule then the depth will increase. Nevertheless, if the max_depth parameter is not set then the depth of a tree can keep increasing and the tree will most likely overfit the data. Overfitting is something to avoid as it will create a rule for each and every different decision which can make the accuracy suffer as well as becoming extremely performance intensive which means the model takes a long time to train. The max_depth parameter was chosen to be 25. Through comparing several different max_depth parameters as seen in Figure 3-35, the value 25 provided the optimal accuracy. [31]

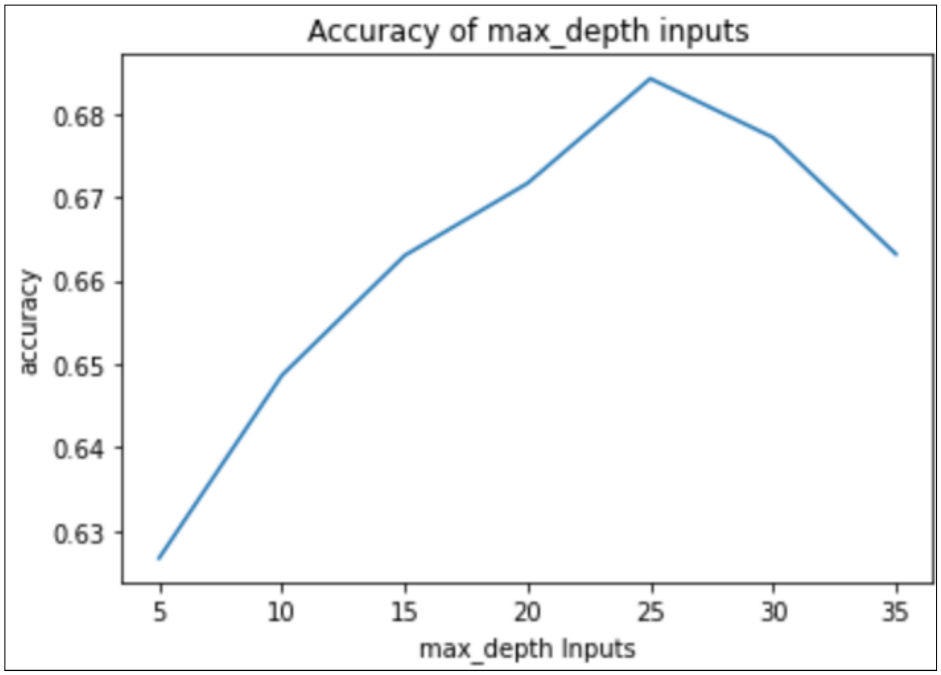


Figure 3-25: Comparison of max_depth inputs where 25 was chosen

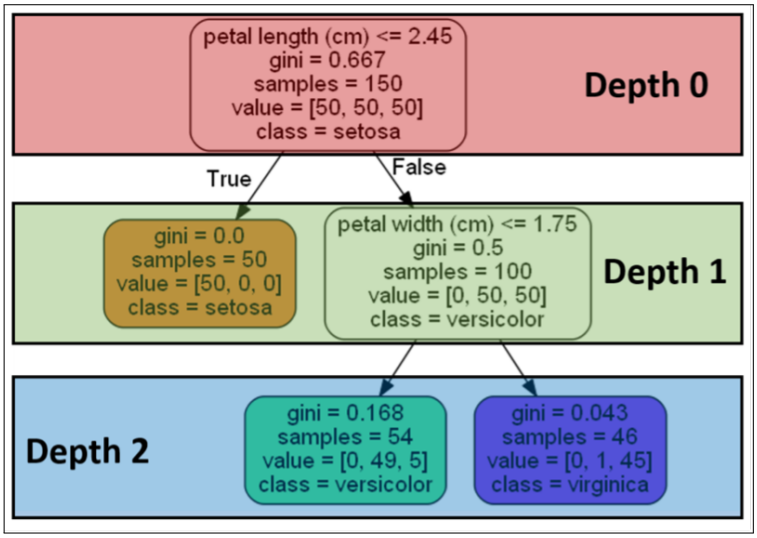


Figure 3-26: Example of an Iris flower dataset decision tree [41]

There are two types of criterion parameters which can be chosen for the criterion which are "gini" or "entropy." As mentioned in section 2.10.3 gini focuses determines the optimal split by calculating the number of times any element within the dataset was wrongly mislabelled based on the initial random label assignments. Entropy will determine the optimal split of a node by the measure the amount of in information which will suggest the disorder of features within the dataset against the target. [41]

Gini was chosen as the criterion as the performance of training is faster and more accurate for example comparing the results in Figure 3-27 and Figure 3-28 a decision tree model with gini as criteria had a slightly higher precision and F1-score for positive sentiments.

```
[[88316 31684]
 [44088 75912]]
              precision    recall  f1-score   support

    negative       0.67      0.74      0.70    120000
    positive       0.71      0.63      0.67    120000

    accuracy                           0.68    240000
   macro avg       0.69      0.68      0.68    240000
weighted avg       0.69      0.68      0.68    240000
```

Figure 3-27: The performance metrics of the decision tree model using the "gini" criterion parameter which performed better than "entropy"

```
[[88421 31579]
 [45088 74912]]
              precision    recall  f1-score   support

    negative       0.66      0.74      0.70    120000
    positive       0.70      0.62      0.66    120000

    accuracy                           0.68    240000
   macro avg       0.68      0.68      0.68    240000
weighted avg       0.68      0.68      0.68    240000
```

Figure 3-28: The performance metrics of the decision tree model using the "entropy" criterion parameter which performed worse than "gini"

3.10.2 TF-IDF for Decision Trees

As highlighted in Figure 3-29 the Decision Tree Classifier was initiated with a criterion of "gini" and max_depth of 25. This classifier is using the TF-IDF transformed data. The final accuracy of the Decision Tree Classifier for TF-IDF was 68%.

```
# creating the calssifier and using the gini appraoch over entropy
DT = DecisionTreeClassifier(criterion='gini', max_depth=25)

# fit the training data onto the model
DT.fit(x_tfidf_train, y_tfidf_train)

# fit the the logistic regression classifier on the x train and y train data
predictions = DT.predict(x_tfidf_val)

acc = accuracy_score(y_tfidf_val, predictions)

print("The accuracy score is " + str(acc))

confusion_matrix = confusion_matrix(y_tfidf_val, predictions)
print(confusion_matrix)

# use y-validation set to determine the correct amount of classifications based on predictions
model_metrics = classification_report(y_tfidf_val, predictions)
print(model_metrics)
```

Figure 3-29: Code which shows the transformed TF-IDF data being fit onto the classifier

```
[[88316 31684]
 [44088 75912]]
              precision    recall  f1-score   support

    negative       0.67      0.74      0.70    120000
    positive       0.71      0.63      0.67    120000

    accuracy                           0.68    240000
   macro avg       0.69      0.68      0.68    240000
weighted avg       0.69      0.68      0.68    240000
```

Figure 3-30: The final output of the TF-IDF decision tree model which has an accuracy of 68%.

3.10.3 Word2vec for Decision Trees

Following the TF-IDF transformation, the Decision Tree Classifier is initialised for Word2vec with the parameters of having criterion set to "gini" and the max_depth set to 25.

The final accuracy of the Decision Tree Classifier was 64%. The accuracy showed reduced performance than the Logistic Regression model and Multinomial Naïve Bayes models. The training time for this model was fairly long taking around 4 hours and the performance of predicting is much slower compared to the Logistic Regression and the Decision Tree Classifier.

	precision	recall	f1-score	support
negative	0.64	0.65	0.65	120000
postive	0.64	0.64	0.64	120000
accuracy			0.64	240000
macro avg	0.64	0.64	0.64	240000
weighted avg	0.64	0.64	0.64	240000

Figure 3-32: The final output of the TF-IDF decision tree model which has an accuracy of 68%.

3.11 Final Model Analysis Comparisons

The following figure shows the final models and their accuracies along with TF-IDF and Word2vec file sizes.

Model	Word2Vector Accuracy	TF-IDF Accuracy	Word2Vector .pkl file size	TF-IDF .pkl file size
Logistic Regression	76%	82%	20.1mb	324kb
Multinomial Naïve Bayes	N/A	80%	N/A	1.3mb
Decision Trees	64%	68%	17.5mb	1.4mb

Figure 3-33: The final output of the TF-IDF decision tree model which has an accuracy of 68%.

3.12 Determining the Final Model

The final model which will be chosen is the Logistic Regression model. This is because Logistic Regression yielded the highest accuracy as well as having a very fast performance. Another important aspect to consider is the size of the model files. Word2vec model files are slightly larger large with some becoming as large as 20mb or higher. Also, when making a prediction the .pkl model file will need to be loaded and loading a large file can lead very slow predictions and a bad user experience. On the other hand, the Logistic_Regrssion.pkl file is only 324kb with a much higher accuracy which also means the predictions can be made faster. The vectoriser will also need to be saved and deployed to the production server as a .pkl file. Still, the vectorizer is quite large at a size of 123.6mb. Although the vectoriser is required alongside a TF-IDF model so it makes it suitable for deployment.

3.13 Web Application Design

This section showcases the web application design detailing the functional and non-functional requirements. The web application utilised new Python files and the previously saved model .pkl files during the Machine Learning Analysis in sections 3.4 – 3.12. The backend architecture uses a Flask Server and PostgreSQL database. The front-end will be designed with HTML5, CSS, and JavaScript.

3.13.1 Functional Requirements/Non-functional Requirements

The figure below specifies the functional and non-functional requirements. Functional requirements begin with the letter 'F' within the requirement ID and non-functional requirements begin with the letter 'U.'

Requirement ID	Requirement Description
F1	Allow a user to type in a topic or hashtag in the search bar for SA
F2	When the user clicks the 'submit' button the Twitter API should be called to fetch Tweets
F3	Predictions should be returned to front-end when the user requests
F4	Aspects should be returned to front-end when the user requests
F5	The system must return the exact number of Tweets specified when submitting
F6	The system must return the classification of a sentiment
F7	Database must store the Tweets collected from API into PostgreSQL database
F8	Database must store the prediction of the sentiment for each Tweet into PostgreSQL database
F9	The web application must specify an aspect for each Tweet
F10	The "Tweets" and "Predictions" table must be deleted each time a user submits a new Tweet via the front-end
U1	The user should be able to select the number of Tweets they want to retrieve from the Twitter API.
U2	The user should be able to scroll and see the all the Tweets which were collected by the Twitter API
U3	The user should be able to expand the Tweets to full view to see the Tweets in a larger scrollable region
U4	The user should be able to expand the Predictions to full view to see the Predictions in a larger scrollable region
U5	The user should be able to expand the Aspects tab to full view to see the Aspects in a larger scrollable region
U6	The user should be able to see the pie chart with the correct percentages of positive, neutral, or negative Tweets
U7	Create a name for the overall web application

Figure 3-34: Requirements Figure

3.13.2 Libraries and Software Being Used

The figure below will list the most important libraries used and their sources for the web application.

Library/Service	Description	Source
Python	Programming language	https://www.python.org/downloads/source/
spaCy	NLP library used for SA and ABSA	https://spacy.io/ https://pypi.org/project/spacy/
Flask	Web server library for back-end development	https://flask.palletsprojects.com/en/2.0.x/ https://pypi.org/project/Flask/
Tweepy	Twitter API Library to create request from the Twitter API	https://github.com/tweepy/tweepy https://pypi.org/project/tweepy/
Pickle	Binary compression library for model files	https://pypi.org/project/pickle5/
Sci-Kit Learn (Sklearn)	Machine Learning library	https://pypi.org/project/scikit-learn/
Redis	Background job worker library	https://pypi.org/project/redis/
RQ	Queue job library	https://pypi.org/project/rq/
SQLAlchemy	Database configuration manager	https://github.com/sqlalchemy https://pypi.org/project/SQLAlchemy/

Figure 3-35: The final model comparison

4 Implementation

This section covers the web application implementation. This includes initial creation to end-result of the web application.

4.1 Initialise Flask Web Application

The initial creation of the web app will contain 3 components. This is the app.py Python file, a Templates folder, and a Static folder. The first step for creating the Flask web application must be a simple app.py file. The app.py file is a Python file that is the starting point of the application. This file is run once the Flask web application is deployed. In order to initiate a Flask server, the Python Flask library needs to be imported and configured. The Flask application is now ready for further development. The chosen name of the web application is "Flask Sentiment Analysis" as stated by

requirement U7 in section 3.13.1.

The app.py file also contains the business logic of the app. For example, fetching Tweets, making predictions, and determining aspects will be made into functions which will be called from app.py. The Templates folder contains the HTML5 files. Templates are what the user interface sees and interact with. The static folder contains all the CSS and JavaScript code. CSS modifies the attributes of the HTML files found within the Templates folder. CSS focuses on styling and formatting of HTML web pages. JavaScript is rendered in the browser when a web page is loaded. JavaScript controls the behaviour of a web page and how the user interacts with it. For example, if a user clicks a button and a loading icon appears, then that is JavaScript in action and controlling that behaviour.

4.2 Routing

Routing is an essential part of a Flask application. The routes are similar to controller actions in a MVC architecture. Routes are the functions which are called whenever the user initiates the function via the HTML web page. They can also be initiated by JavaScript functions. Routes contain the business logic of the web application handling all of the back-end code functions. Routes are denoted by using a '@/' before naming the route. An example can be seen below. Routes are also allowed different REST methods such as POST, GET, PUT, and DELETE. Figure 4-1 shows an example of a route created to fetch Twitter Tweets from the Twitter API.

```
# selects and returns Tweets to the user depending on the amount of Tweets selected
@app.route('/fetchTweets', methods=['POST', 'GET'])
def fetchTweets():

    # clears Tweet and Prediction tables
    db.session.query(Tweets).delete()
    db.session.commit()
    db.session.query(Predictions).delete()
    db.session.commit()

    term = request.form['text']
    tweetAmount = int(request.form.get('tweetAmount'))
    tweetList = api.getTweets(term, tweetAmount)

    return render_template("tweets.html", tweets = tweetList)
```

Figure 4-1: Example of an "@app.route" declaration

4.3 Database configuration

Database configuration is the next step of implementation and this is carried out through the Python SQLAlchemy library. SQLAlchemy allows SQL commands to be created within Python. For this

project Tweets from the Twitter API and predictions made by the ML models need to be stored into database tables. There are two main tables which will be created for this project which are "Tweets" and "Predictions." Allowing data to be stored into tables allows easy access to the data whenever needed at different places in the application. For example, Tweets will need to be stored into the "Tweets" tables. After, a function is needed to create an SQL command to read the Tweets from the table and create predictions. The predictions then can be stored into a "Predictions" database table. Figure 4-2 shows how a table is created with SQLAlchemy in Python.

```python
import tweepy
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Tweets(db.Model):
    __tablename__ = 'Tweets'
    id = db.Column(db.Integer, primary_key=True)
    User = db.Column(db.String(1000))
    Handle = db.Column(db.String(1000))
    TweetText = db.Column(db.String(1000))
    ReplyCount = db.Column(db.Integer())
    RetweetCount = db.Column(db.Integer())
    LikeCount = db.Column(db.Integer())

    def __init__(self, User, Handle, TweetText, ReplyCount, RetweetCount, LikeCount):
        self.User = User
        self.Handle = Handle
        self.TweetText = TweetText
        self.ReplyCount = ReplyCount
        self.RetweetCount = RetweetCount
        self.LikeCount = LikeCount
```

Figure 4-2: Tweets table created with various columns and data type configurations

4.4 Twitter API

In order to fetch Tweets, live from Twitter in real time the Twitter API must be used to make requests. Twitter does not allow web scraping and only allows Tweets to be collected through their official Twitter API. In order to access the Twitter API a developer account is needed. A developer account was created, and Twitter has given access for the Flask web application to make requests to the API. The configuration can be seen in Figure 4-3.

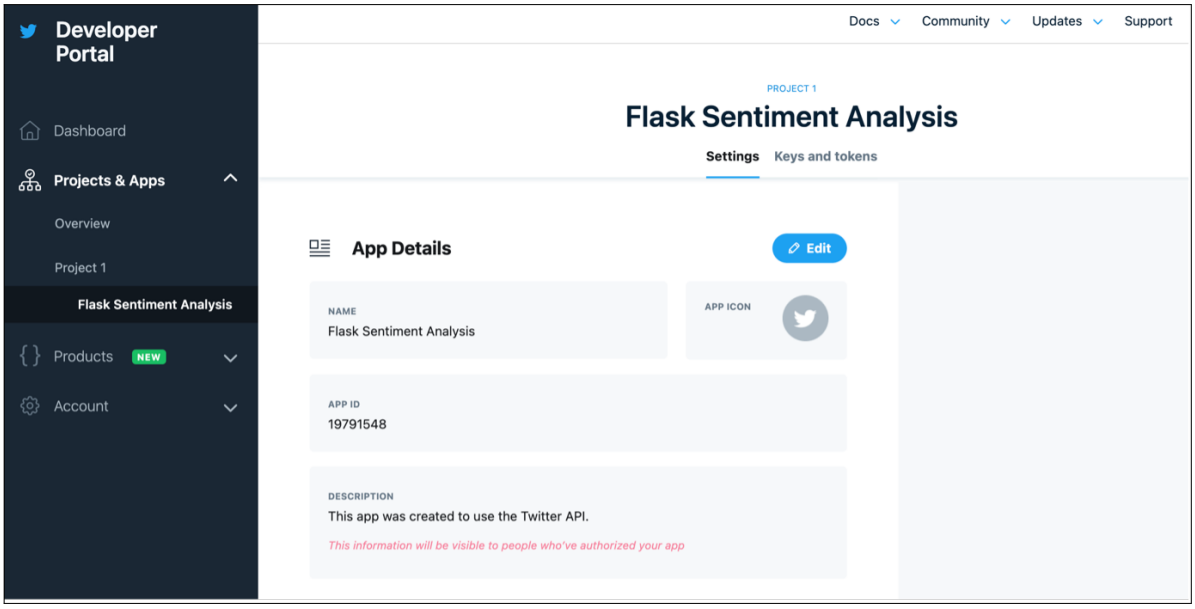


Figure 4-3: Twitter Developer account configuration settings portal

4.5 Add basic HTML and CSS

After being able to retrieve Tweets from Twitter and can store within the database, the front-end UI can be created. Figure 4-4 shows an initial design of the front-end. The design at first is very basic asfurther functionalities need to be built. Figure 4-4 shows how a user can submit a topic, Tweets to be collected from the API, and the Tweets that are stored in the database.

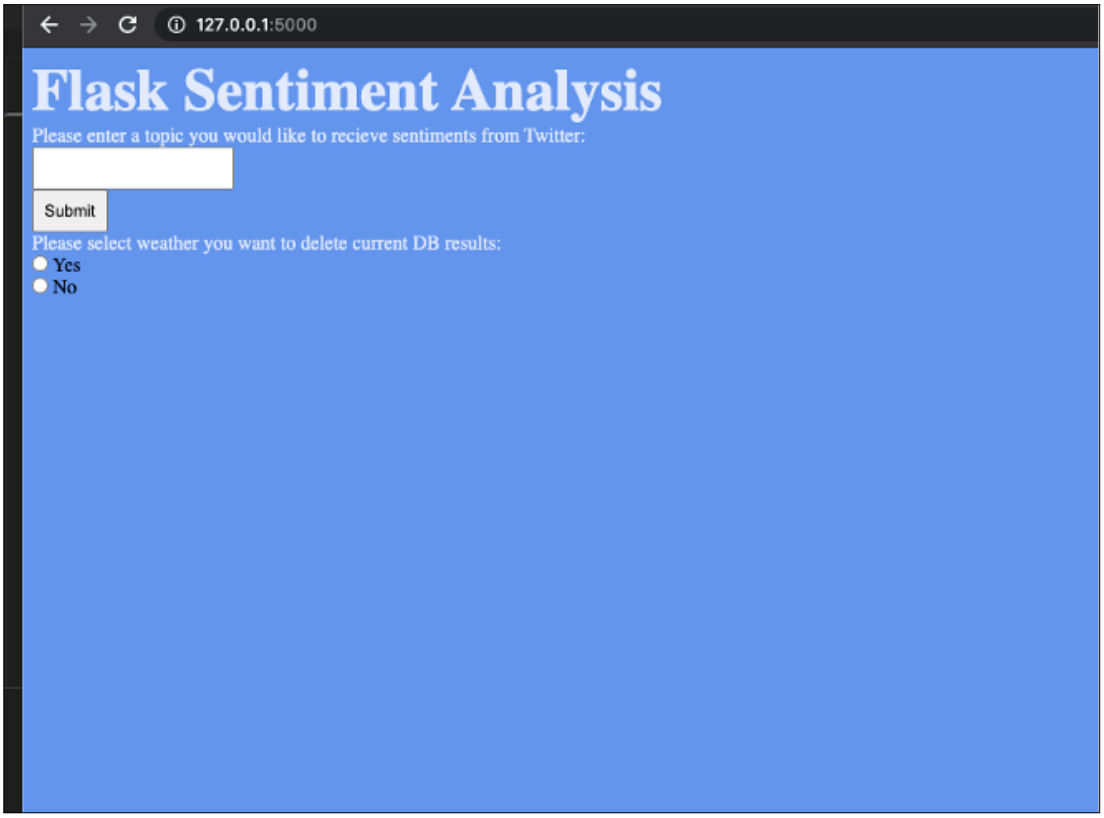


Figure 4-4: Very early first design of the Flask Sentiment Analysis web application

4.6 Generate Predictions

After further functionalities were added such as allowing the user to select a number of Tweets the Tweets can be viewed in the front-end. The user has choice to find predictions for those Tweets. The function that handles this is the "predictSentiment()" function within the app.py file. The function take in the Tweets found as a parameter and passes it to the "Logistic_Regression.get_predictions()" function.

Within this function each Tweet is passed to the preProcess function mentioned previously and then the TF-IDF vectorizer and Logistic Regression .pkl file is loaded. Each Tweet is passed into the vectorizer and then finally to the "log_model.predict()" function which uses the saved Logistic regression model and generates a prediction. Although generally there are two classes which are positive and negative there is also neutral state. A neutral classification will be made if neither the probability of the Tweet being positive or negative is above 65%. Figure 4-5 shows the code to illustrate the prediction process in more clarity.

```
def get_predictions(tweetList):
    # load the logistic regression model
    log_model = pd.read_pickle('Sentiment/TF_IDF_Pickle_Models/LR.pkl')

    # create an empty dataframe
    predictionResults = pd.DataFrame(columns=['text', 'prediction'])

    # instantiate the vectorizer instance
    vectorizer = pickle.load(open('Sentiment/TF_IDF_Pickle_Models/vectorizer.pkl', 'rb'))

    nlp = en_core_web_md.load()

    for tweet in tweetList:

        # pre-process the tweet
        tweet_processed = preProcess(tweet, nlp)
        tweet_processed = [tweet_processed]

        # transform tweet using vectorizer
        tweet_processed = vectorizer.transform(tweet_processed)
        prediction = log_model.predict(tweet_processed)

        positiveProb = (int(log_model.predict_proba(tweet_processed)[0,1] * 100))

        negativeProb = (int(log_model.predict_proba(tweet_processed)[0,0] * 100))

        print("Probability of sentiment being negative: ")
        print(int(log_model.predict_proba(tweet_processed)[0,0] * 100))
        print("Probability of sentiment being postive: ")
        print(int(log_model.predict_proba(tweet_processed)[0,1] * 100))

        # assign correct prediction to new variable

        if positiveProb < 65 and negativeProb < 65:
            prediction = 'neutral'
        elif prediction == ['positive']:
            prediction = "positive"
        else:
            prediction = "negative"

        print(prediction)

        # append each prediction result to dataframe
        predictionResults = predictionResults.append({'text': tweet, 'prediction': prediction}, ignore_index=True)
```

Figure 4-5: Shows the "get_prediction()" function of the Logistic Regression model

4.7 Update the User Interface to Reflect New Functionalities

After updating the back-end functions and allowing the user more functionality, the user interface can be updated. A much more simple and cleaner interface was implemented to allow the user to submit a topic, select the number of Tweets, view the Tweets, and view predictions. Figure 4-6 and Figure 4-7 demonstrate the updates.

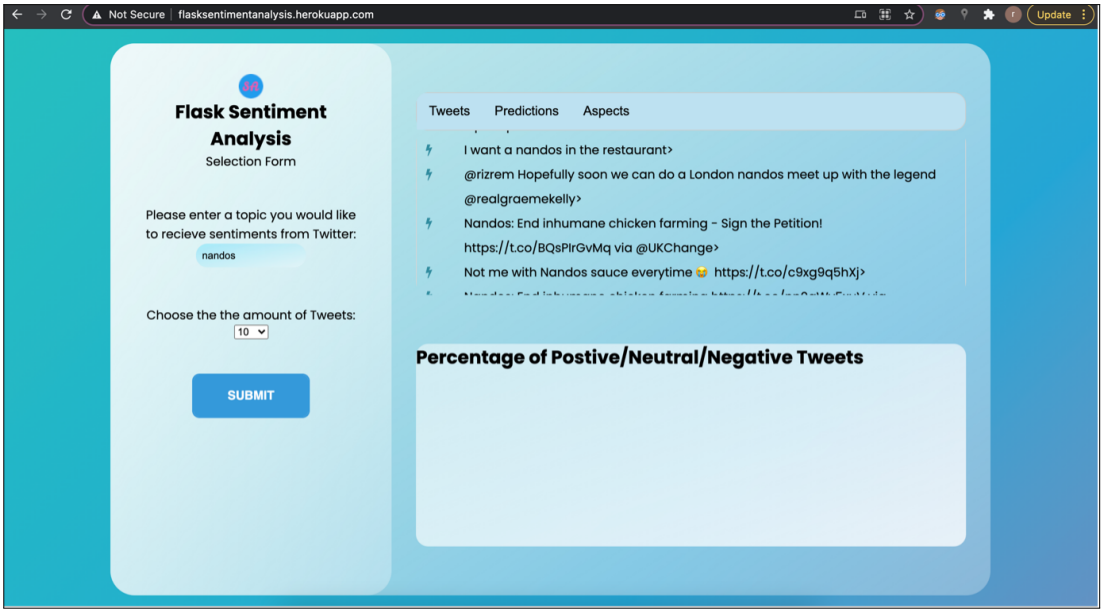


Figure 4-6: Users can view and scroll through Tweets found via "Tweets" tab

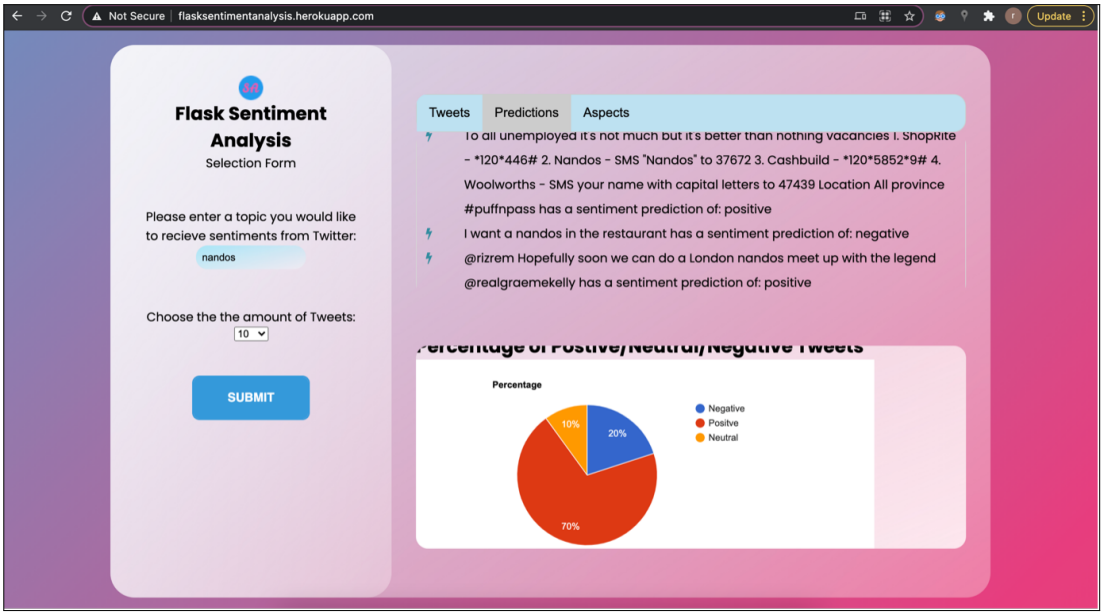


Figure 4-7: Users can view and scroll through Predictions found via "Predictions" tab

4.8 Creating the ABSA Model

The final significant part of the project is to implement the ABSA model. The goal is to use the Tweets stores in the "Tweets" table in the database and determine the entities and aspects of the Tweet. The spaCy model was used extensively for this process. First, in order to extract aspects, the sentence syntax and structure along with parts of speech and dependencies were determined. This was conducted through spaCy's POS tagger and dependency parser.

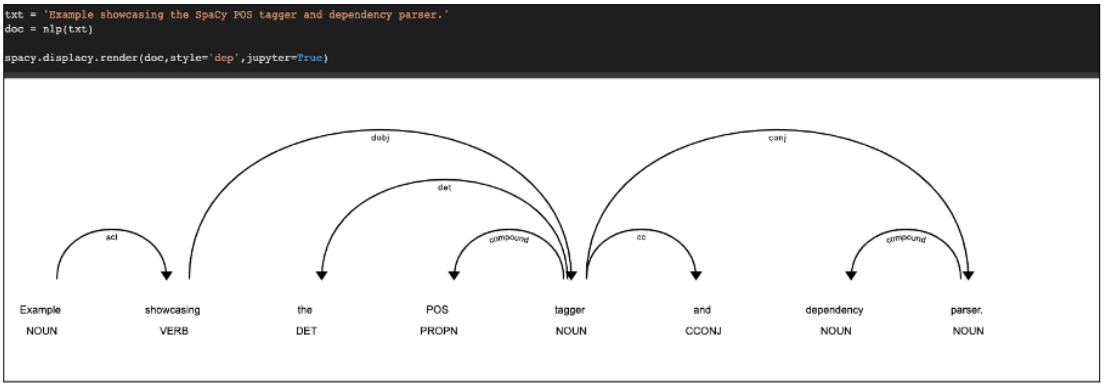


Figure 4-8: spaCy POS tagger and dependency parser

Figure 4-8 the parts-of-speech are denoted under each word and the dependency relation is denoted by the arrows. This is very helpful to understand the syntactical structure of a sentence. The next step was to identify patterns using the POS tagger and dependency relationships.

4.9 How can the ABSA model detect patterns?

When a sentence is passed to the ABSA model file, the sentence is passed to an "nlp()" function which tokenises the sentence and determines the POS and dependency relations. Afterwards, a for-loop function can be created to loop through each word of the sentence allowing to extract import aspects and descriptions.

4.10 Identifying Patterns

The most important dependency relationships to identify is the nominal subjects or direct objects. These subjects and objects contain the core topic of the sentence. For example, in the below figure, the text is "I love the restaurants pizza" which shows the "nsubj" or nominal subject as "I" and the "dobj" or direct object as "pizza." The nominal subjects and direct objects of a sentence can be considered as the aspects.

```
txt = 'I love the restaurants pizza.'
doc = nlp(txt)

spacy.displacy.render(doc,style='dep',jupyter=True)
```

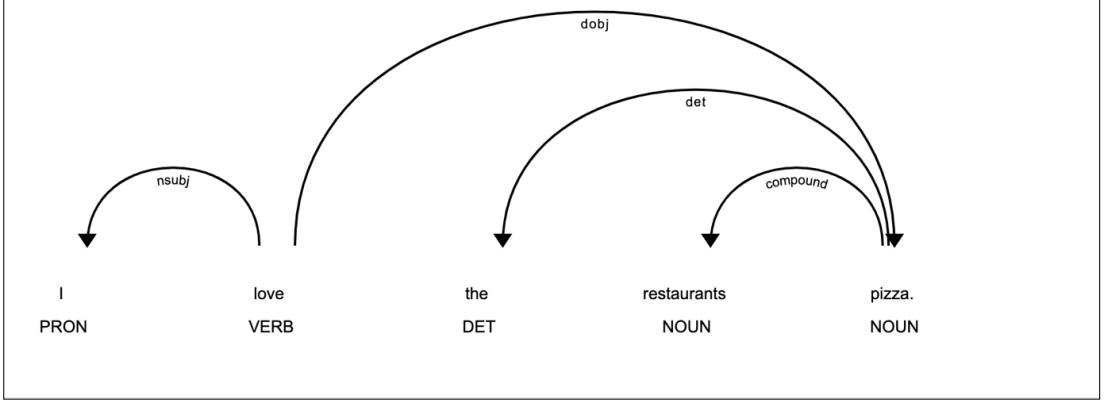


Figure 4-9: spaCy POS tagger and dependency parser on an the example text of "I love the restaurants pizza."

After finding the nominal subject or direct object dependency, the next step was to find the POS relation to the dependency. Considering Figure 4-9, the POS for the nominal subject is "PRON" or pronoun and the POS for the direct object is a noun. Now that the dependency relation and POS have been identified, a deeper look into the structure shows that simply identifying the nominal subject and direct object aspects are not enough andadescription of the aspectsis also required.

A description can come in many forms and a couple common descriptions can be adjectives, adjective modifiers, verbs, and nouns. Also, in Figure 4-9 the description can be identified by the verb which is 'love.' An important thing to notice is the nominal subject dependency and direct object dependency both have 'love' as a child node. Combining the description with the aspects gives "I love pizza." Similarly, in the ABSA model code a pattern for finding this aspect and description pattern is implemented.

Another pattern for identifying aspects is pinpointing POS tags. As illustrated in Figure 4-10 there is a for-loop expression on line 251 which iterates through each sentence or Tweet which was collected from the Twitter API. Within the for-loop expression each token is checked for a dependency relation of nominal subject, and if the POS of the token is a noun, pronoun, or determiner. Specifically, these conditions can be seen in lines 255, 266, and 276. If a token meets any of these conditions, then the appropriate function is called to determine the aspect found.

```
251     for sent in doc.sents:
252       print("The sentence is: " + str(sent))
253       for token in sent:
254         # first condition finding nominal subject - aspect
255         if token.pos_ == 'NOUN' and token.dep_ == 'nsubj':
256           aspect_found = Noun_nsubj(token)
257           if aspect_found == '' or aspect_found is None:
258             aspect_found = ''
259             continue
260           else:
261             print("Aspect found is: " + aspect_found)
262             global aspect_found_flag
263             aspect_found_flag = True
264
265         # finding nominal subject if it's a pronoun - aspect
266         if token.pos_ == 'PRON' or token.pos_ == "PROPN" and token.dep_ == 'nsubj':
267           aspect_found = Pron_nsubj(token)
268           if aspect_found == '' or aspect_found is None:
269             aspect_found = ''
270             continue
271           else:
272             print("Aspect found is: " + aspect_found)
273             aspect_found_flag = True
274
275         # finding nominal subject if it's a determiner - aspect
276         if token.pos_ == 'DET' and token.dep_ == 'nsubj':
277           aspect_found = Det_nsubj(token)
278           if aspect_found == '' or aspect_found is None:
279             aspect_found = ''
280             continue
281           else:
282             print("Aspect found is: " + aspect_found)
283             aspect_found_flag = True
```

Figure 4-10: Showing code which will try to identify POS tokens and dependency relations to identify patterns

After finding the aspect then the next section of the code will try to find the adjective or description linked to the aspect. This is done through the "get_adjective()" function on line 325 in Figure 4-11. The "get_adjective()" function which will check if the token has a negation, left or right adjective modifier, or double adjective. It is very important to check for these conditions in order to find a descriptive description which makes for a logical link to the aspect. In addition, negations, adjective modifiers, and double adjectives must be identified so an accurate descriptions of the aspect can be made. Figure 4-12 illustrates this with an example of code.

```
322         # second condition finding adjective of aspect
323         if token.pos_ == 'ADJ':
324           global adjective_found
325           adjective_found = get_adjective(token)
326           # can remove below line???
327           if adjective_found != '':
328             if adjective_found not in adjective_found_list:
329               adjective_found_list.append(adjective_found)
330               print("ADJ is: " + token.text)
331               print("OK the ADJ is set here as: " + adjective_found)
332               global adjective_found_flag
333               adjective_found_flag = True
```

Figure 4-11: Showing code within the "get_adjective()" function

```
62  # gets adjective with any negations, adjective modifiers, or double adjectives
63  def get_adjective(token):
64    # checks for negation
65    negation = ''
66
67    # checking for negation in ancestor tokens
68    for ancestor in token.ancestors:
69      if ancestor.dep_ == 'neg':
70        negation_position = str(ancestor.nbor(-1) + child.text + str(ancestor.nbor(1)))
71        if negation_position not in negation_found_list:
72          negation_found_list.append(negation_position)
73          negation = ancestor.text + " "
74        continue
75
76      for child in ancestor.children:
77        if child.dep_ == 'neg':
78          negation_position = child.nbor(-1).text + child.text + child.nbor(1).text
79          if negation_position not in negation_found_list:
80            negation_found_list.append(negation_position)
81            negation = child.text + " "
82
83            print("Negation found." + " The negation is: " + negation)
84
85    # checking for negation in child tokens
86    for child in token.children:
87        if child.dep_ == 'neg':
88          negation = child.text + " "
89        for c in child.children:
90          if c.dep_ == 'neg':
91            negation = child.text + " "
92            print("Negation found." + " The negation is: " + negation)
93
94    if negation == '':
95      print("No negation was found")
96
97    left_adjective = ''
98    right_adjective = ''
99
100   # checks for advmods of the adjective
101   if len(list(token.lefts)) > 0:
```

Figure 4-12: Snippet of the "get_adjective()" function which checks for negations, left and right adjective modifiers, and double adjectives."

The ABSA model can extract aspects with descriptions from most sentences if they contain valid nominal subjects or direct objects. However, there are times where the ABSA model does not find any aspects because there is not sufficient information or tokens for the model to identify and create meaningful relationships.

4.11 Example of performance

Figure 4-13 shows there are a few example sentences passed into the ABSA which are shown on line 388. The following output shows the process of the ABSA model searching through each sentence and each token identifying various POS tags and dependency relation patters. The output towards the very end of Figure 4-13 shows the aspects with their descriptions. The ABSA model performs very fast as there were three sentences passed into the model and the output was shown within 1.52 seconds.

```
if __name__ == '__main__':
    import time
    start = time.time()
    print('Timer started')

    tweet_list = ["I love pizza, but I really hated the customer service.", "I love food.", 'Pam has rarely cooked any food.']
    data_dict = main(tweet_list)
    for name in data_dict:
      print("The aspects are: " + str(data_dict[name]))
    end = time.time()
    print("Identifying these aspects took a total of: " + str(round(end - start, 2)) + " seconds")
```

```
Timer started
The tweet is: I love pizza, but I really hated the customer service.
The sentence is: I love pizza, but I really hated the customer service.
Direct object phrase found so will return
The tokens in left are: I
The nsubj found is:I
The dobj found is: pizza
No negation was found
Verb found is: love
Direct object phrase found so will return
The tokens in left are: I
The nsubj found is:I
The tokens in left are: really
The dobj found is: service
No negation was found
Left advmod: really
Verb found is: really hated
The tweet is: I love food.
The sentence is: I love food.
Direct object phrase found so will return
The tokens in left are: I
The nsubj found is:I
The dobj found is: food
No negation was found
Verb found is: love
The tweet is: Pam has rarely cooked any food.
The sentence is: Pam has rarely cooked any food.
Direct object phrase found so will return
The tokens in left are: Pam
The nsubj found is:Pam
The tokens in left are: has
The tokens in left are: rarely
The dobj found is: food
No negation was found
Left advmod: rarely
Verb found is: rarely cooked
The aspects are: [['[The nominal subject and verb with direct object aspect is: I love pizza]', '[The nominal subject and verb with direct object aspect is: I really hated service]']]
The aspects are: [['[The nominal subject and verb with direct object aspect is: I love food]']]
The aspects are: [['[The nominal subject and verb with direct object aspect is: Pam rarely cooked food]']]
Identifying these aspects took a total of: 1.52 seconds
```

Figure 4-13: Output of aspects and descriptions for example sentences being passed into the ABSA model

4.12 Final Touches

Finally, the ABSA model has been created. Now the ABSA model needs to be integrated into the web application. Firstly, a new tab for Aspects needs to be added to the front-end of the web application. Also, a button called “Find aspects for Tweets” will be added to the front-end. When a user clicks the button then the function “aspectPrediction()” within app.py will be called and return the aspects identified to the user. A pie chart showing the percentages of Tweets which were positive, neutral, or negative has also been added for visualisation purposes. Cleaning the UI will also help navigate the web application an provide a seamless experience. Therefore, unnecessary HTML elements such as “Review submissions” and the list of the names of ML models have been removed. The final result can be seen in Figure 4-14.

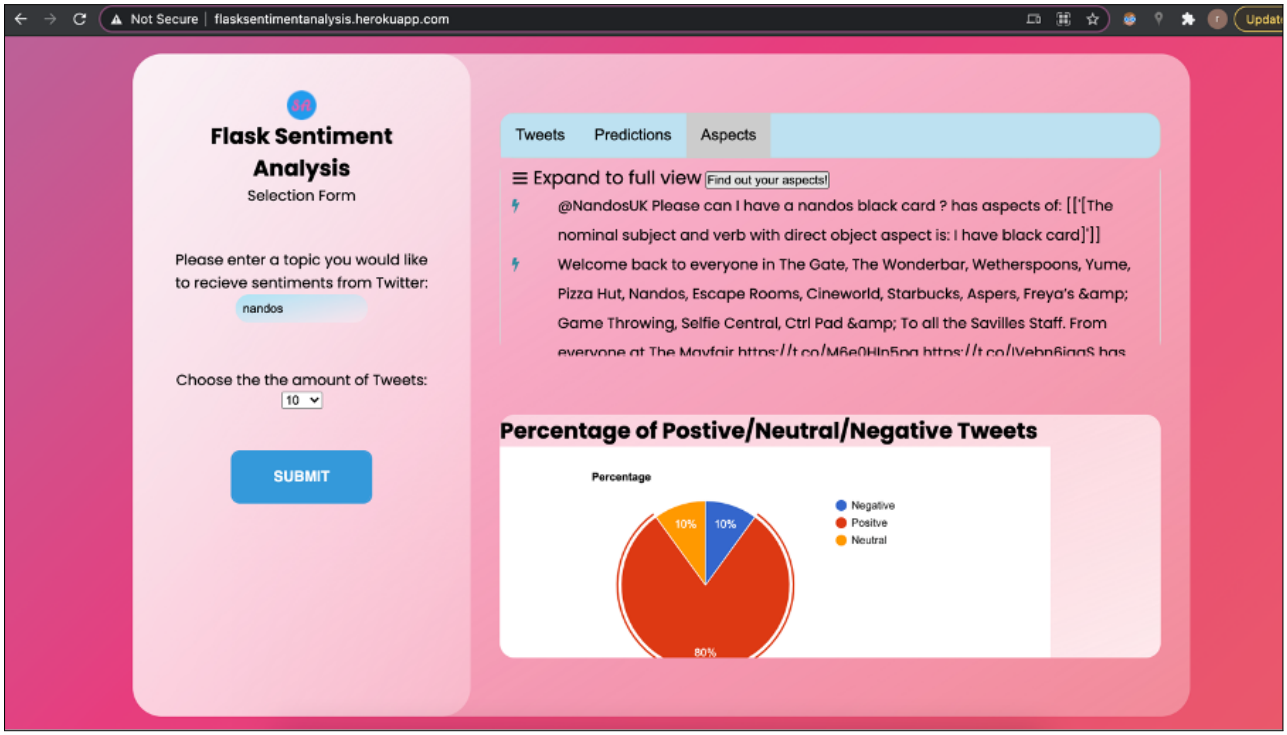


Figure 4-14: Updated Aspects tab with "Find your aspects!" button and removed unnecessary HTML elements from UI

4.13 Deploying to Heroku

The last stage of the web application is to deploy to Heroku. An account was created on Heroku a new application called "flasksentimentanalysis" along with a new Dyno was developed. A Dyno is essentially a cloud hosted web server which allows the deployment of a scalable web application. Once the Dyno is initialised all of the local development files can be pushed to the Heroku Dyno. Fortunately, Heroku offers a command line interface (CLI) to handle deployments of applications to production services which includes Git commands to push code. Using Git commands all development software including the machine learning .pkl files, Python code, HTML, CSS, and JavaScript files were compiled successfully on the Dyno. If a compilation is successful on the Heroku Dyno then this means the web application should perform exactly the same way it does locally, but on a production hosted web service. The URL for the web application should meet requirement U7 from the requirements table from section 3.13.1. Therefore, the web hosted URL is: www.flasksentimentanalysis.herokuapp.com as shown in Figure 4-15.

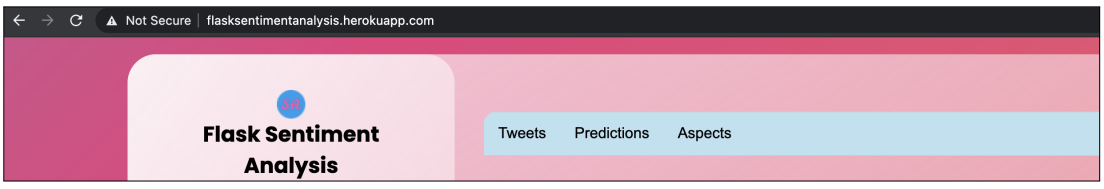


Figure 4-15: The Flask Sentiment Analysis URL: www.flasksentimentanalysis.herokuapp.com

5 Evaluation and Concluding Statement

This section focuses on the overall evaluation of the system including project aims, results, issues, challenges, and future work.

5.1 Requirements Testing

This section goes through testing of the entire application including testing each of the functional and non-functional requirements of the system. Figure 5-1, Figure 5-2, and Figure 5-3 highlights the results. The actual outcome boxes are highlighted green for successfully implemented, yellow for partially implemented and red for unsuccessful implementation.

Please reference demo presentation for testing of code functionality.

Requirement ID	Requirement Description	Expected Outcome	Actual Outcome
F1	Allow a user to type in a topic or hashtag in the search bar for SA	The user can type any string into the search bar and will be considered the topic to retrieve Twitter Tweets from	The user can successfully enter any topic or hashtag into the search bar
F2	When the user clicks the 'submit' button the Twitter API should be called to fetch Tweets	The Twitter API should be called and should retrieve Tweets based on the amount specified	The Twitter API is called each time the user clicks the 'submit' button and it fetches the correct number of Tweets
F3	Predictions should be returned to front-end when the user requests	The predictSentiment function in app.py is called and returns the predictions to the front-end	The predictSentiment() function gets called successfully and returns the predictions of the Tweet if it is positive, neutral, or negative. Also, the Logistic_Regression.pkl and vectorizer.pkl files are called and loaded successfully during the predictions.
F4	Aspects should be returned to front-end when the user requests	The aspectPrediction function in app.py is called and aspects for each Tweet is generated. If no aspect can be found it should let the user know.	The aspectPrediction function gets called successfully and returns a descriptive list of aspects being returned. It does let the user know if there were no aspects found.
F5	The system must return the exact number of Tweets specified when submitting	The web application must return the same number of Tweets in the "Tweets" header with the number of Tweets specified when submitting.	The number of Tweets returned is correct and matches the amount the user initially selected.

Figure 5-1: Requirements Figure

F6	The system must return the classification of a sentiment	The user must be able to see if the Tweet in their search was 'positive', 'neutral', or 'negative'	At the end of the Tweet the sentiment classification is given and can be either positive, neutral, or negative.
F7	Database must store the Tweets collected from API into PostgreSQL database	Depending on the number of Tweets selected, each Tweet should be saved into a 'Tweets' table in the PostgreSQL database.	The "Tweets" table within the database gets populated successfully based on the number of Tweets selected.
F8	Database must store the prediction of the sentiment for each Tweet into PostgreSQL database	Predictions for each Tweet must be stored into a "Predictions" database table	The "Predictions" table within the database gets populated successfully based on the number of Tweets selected.
F9	The web application must specify an aspect for each Tweet	The web application should return an aspect for each Tweet and if there was no aspect identified then	At the end of each Tweet within the "Aspects" tab, the aspects are identified and described as nominal subjects or direct objects which were found. If there are no aspects, then at the end of the Tweet it will tell the user no aspects were identified.
F10	The "Tweets" and "Predictions" table must be deleted each time a user submits a new Tweet via the front-end	The row count for the "Tweets" and "Predictions" table should be zero when submitting a new topic from the front-end	The "Tweets" and "Predictions" table both have records deleted successfully each time the user submits a new Tweet.
U1	The user should be able to select the number of Tweets they want to retrieve from the Twitter API.	There should be a dropdown list on the front-end allowing the user to select number of Tweets	A dropdown with the number of Tweets to fetch from the Twitter API is shown. The available options are 10, 50, 100, and 200.

Figure 5-2: Requirements Figure

U2	The user should be able to scroll and see the all the Tweets which were collected by the Twitter API	After submission the "Tweets" tab should be populated with Tweets	The "Tweets" tab gets populated with Tweets successfully. Also, the user can scroll through the Tweets is there are a large number of Tweets returned.
U3	The user should be able to expand the Tweets to full view to see the Tweets in a larger scrollable region	There should be a button which allows the user to view the predictions in the "Tweets" tab more clearly	There is a button called "expand to full view" which will create an overlay over the entire screen and shows the Tweets in a much larger more clear view.
U4	The user should be able to expand the Predictions to full view to see the Predictions in a larger scrollable region	There should be a button which allows the user to view the predictions in the "Predictions" tab more clearly	There is a button called "expand to full view" which will create an overlay over the entire screen and shows the Predictions in a much larger more clear view.
U5	The user should be able to expand the Aspects tab to full view to see the Aspects in a larger scrollable region	There should be a button which allows the user to view the aspects in the "Aspects" tab more clearly	There is a button called "expand to full view" which will create an overlay over the entire screen and shows the Aspects in a much larger more clear view.
U6	The user should be able to see the pie chart with the correct percentages of positive, neutral, or negative Tweets	The pie chart should be shown as soon as the predictions are returned to the front-end	The pie chart successfully gets rendered below the tab bar and when the user hover overs a section of the chart it will show the percentage of Tweets which were either 'positive', 'neutral', or 'negative.'
U7	Create a name for the overall web application	A valid and logical name for the web application	The name created is "FlaskSentimentAnalysis"

Figure 5-3: Requirements Figure

5.2 Implementation Performance and Successes

The project investigated a dataset and various NLP techniques that were researched and applied to the dataset. Afterwards, the pre-processed dataset was used to train ML models. Throughout the training process several ML models were analysed and results were compared. The goal of reaching a model accuracy above 80% was also achieved. Using a Flask backend with a PostgreSQL database architecture, a web application was created with the Logistic Regression model used for the sentiment classifications. An ABSA model was created using spaCy and NLP techniques and integrated within the web application to quickly extract main aspects from each Tweet for the SA submission. The overall web application was built successfully with each project aim from Section 1.3 being achieved. Addtionally, each of the functional and non-functional requirements from section 3.13.1 were implemented successfully. The web application's main goal is to provide SA to help businesses and organisations understand social media sentiment. Using the web application businesses can create meaningful and impactful solutions dependent on the analytics provided. In addition, a pie chart for predictions was provided to help with visualisation purposes.

5.3 Issues/Challenges

This section will cover various issues and challenges which happened during the entire system implementation.

5.3.1 Model Performance/size

Although the web application system was successful, there were various issues and challenges throughout the project which affected the performance and progress at certain stages during development. Firstly, although there were many models trained, only the Logistic regression model which had TF-IDF transformed vectors was implemented in the production service of the web application. The Word2Vec models were unfortunately too large and could not be deployed to the production web server.

5.3.2 Processing predictions

Also, when processing the predictions on the front-end of the web application the user has to wait for more than a minute for predictions to be returned. This is due to the TF-IDF vectorizer and "NLP" object which needs to be called for each Tweet in the for-loop expression. Since the vectorizer and NLP objects need to be called for each Tweet, the RAM usage within the web application increases and slows down performance significantly during processing of predictions.

5.3.3 Visualisation and user interface

The visualisation of the Tweets, predictions, and aspects can be generated more intuitively. For example, the user manually clicks on a button to find predictions instead of clicking on the 'Predictions' tab which would then call the function in Flask to generate the predictions. The same issue applies to the 'aspects' tab where the user has to click on a button within the tab instead of simply clicking on the tab to generate aspects. The pie chart shows the correct percentage of positive, neutral, and negative Tweets, but the pie chart also requires the user to scroll down the web page in order to see the entire chart instead of showing the pie chart in its entirety without the need to scroll.

5.3.4 Production hosting service

The production hosting service chosen for this project was Heroku. Although Heroku supports the basic functionality of the Flask Sentiment Analysis web application, there were some restrictions. For example, there is a 500mb limit for size of the entire web server. This meant certain model files could not be implemented if needed for a version 2 of the web application. This includes several Word2Vector ML models such as Logistic regression, Multinomial Naïve Bayes, and Decision Trees.

5.4 Future Development

There are several points of development for the future of the Flask Sentiment Analysis web application. The current state of the project can be considered at Version 1. Version 2 would update several areas including the back-end implementation and front-end user interface.

5.4.1 Back-end Considerations

With regards to back-end development and issues considered in section 5.3, the first point of improvement would be the ML model development. There were only three models which were analysed and trained. More models such as Linear SVC, Random Forrest, Convolutional Neural Networks, and RCNN's could provide more insight and varying performance metrics. The performance metrics could then be compared to those implemented in Version 1 of the web application to choose a higher performing and optimised model solution. Other back-end improvements would be to include micro service architecture and Docker. Docker can help separate parts of the web application such as fetching Tweets, making predictions, and generating aspects into their own microservices. In addition, choosing a cloud-based services for a database may also help performance. For example, the current Version 1 of the web application relies on a PostgreSQL service hosted by Heroku, but a service such as Amazon RDS could provide a more scalable solution with faster read and write times from the database.

Providing more data analytics would be helpful for the future. An example would be to consider frequency of a certain aspect appearing within the Tweets. Currently, the user is provided with

aspects from each Tweet, but identifying the most common aspects found within all the Tweets would provide more interesting statistics. These statistics can lead to more valuable insights for an organisation or business which they could harness.

5.4.3 Front-end Considerations

The front-end design could change in several areas from reformatting to adding support for additional features. A major area for improvement would be the interactivity of the tab bar which holds the Tweets, Predictions, and Aspects. The tab bar can be improved to show each section when clicking on the tab name rather than making the user click a button to begin the process of finding predictions or aspects. Another update would be to allow the user to select a ML model and give the user a choice of different models to use. The user interface would also benefit from adding more analytical insights such as graphs, charts, and averages. For example, a graph to help visualise sentiment of Tweets over a certain period of time, a chart to show different distributions of frequently identified aspect words, or other averages and percentages of sentiment.

5.5 Concluding Statement

To conclude, the project has been highly successful. Although there were many issues and challenges along the way the main project aims were achieved . A thorough literature review was undertaken along with an in-depth machine learning algorithm analysis and web application design. The implementation successfully shows the final web application which is able to use SA and ABSA in social media sentiment to help businesses and organisations.

6 Statement of Ethics

This section will cover various ethical considerations and legislation such as the Computer Misuse Act (CMA) and Data Protection Act (DMA). This section will also cover the legal, ethical, social, and professional ethics which were considered and abided by throughout this entire project.

6.1 Computer Misuse Act (CMA)

The CMA is a piece of legislation which was enacted to criminalise any persons who misuse computer systems by modifying data on the computer or not having the consent and permission. Throughout this project there was a heavy amount of research conducted and data analysis. All information and data in this project were referenced and accessed through legal ways and correct permissions. This includes the research conducted in the Literature Review and Data Analysis sections. The software services and libraries used have been referenced or are open source.

6.2 Data Protection Act (DPA)

The DPA is a piece of legislation which controls how personal information is used by a business or organisation and the protection of an individual's privacy. There are several principles which are described within this legislation. For example, the data must be fair and lawful, adequate, relevant, processed securely, not kept longer than absolutely necessary, and accurate and up to date. This project abides by this act in various ways. The data which is being gathered from Twitter is through an official API. The data or "Tweets" of users is securely stored into an encrypted PostgreSQL database. The data only stays inside the database as long as necessary which means the data will be removed each time there is a new submission of a topic from the front-end. Also, processing the data for predictions and visualisations happens securely through the Flask back-end. The server-side code processes data, however, it does not store any of the data on the server which means there is no sensitive data vulnerable to session attacks.

6.3 Legal Considerations

The legal considerations refer to how I was able to keep my conduct and intentions with the project within boundaries set by the law. The legal considerations describe the way I have kept my research, implementation, and conduct set within the legal boundaries of the law. This includes any pieces of literature, software, libraries, or software services subject to copyright. I had correctly referenced each of my sources and summarised literature into my own words to avoid plagiarism. I have also used reference quotes if I used something directly from my academic literature sources. In order to abide by the legal ethics of the law I had only used code and libraries which were open source. I also referenced each software and library which required a reference.

6.4 Ethical Considerations

There are several ethical considerations which cover the specifics of how information and data was handled. In order to act accordingly and ethically I have made sure to only use data which was available through legal and legitimate sources. For example, the Twitter dataset which was used for the ML training and model development was data gathered through the official Twitter API. Also, when trying to gather Tweets from Twitter I have used the official Twitter API. The data collected from the Twitter API is also deleted from the database each time the user submits a new Twitter topic or when the web application is visited. This will ensure that data is only kept for as long as necessary. The database configuration is done through SQLAlchemy and SQL statements which cannot be subject to SQL dependency injections. Also, the application files for the Flask Sentiment Analysis Web Application will only be available to myself and the University of Surrey upon submission. The deployed application code is on a secure hosted version control repository hosted by Heroku which requires the owner's permission to view or modify any files. This means that the access to the code and data is strictly confidential and restricted.

6.5 Social Considerations

The social considerations for this project include various aspects such as consent of others, the effect the project has on peoples' lives, and basic human rights. The data aspect of the project was a concern in terms of peoples' consent. The project's goal is to enable the use of social media monitoring specifically for businesses or organisations. The project will assist by analysing sentiments and providing useful analytics. However, using a dataset and collecting data via the official Twitter API allows me to use data in accordance with Twitter's rules and regulations. The privacy and consent are very important to consider especially when dealing with social media content, however, the web application system only collects public Tweets and data which is available through the Twitter API. The Tweets will be deleted from the database after a new submission. This project does not infringe on anyone's basic human rights and this project does not cause or have any intention of harm.

6.6 Professional Considerations

Professional considerations include conducting research and implementation of the entire project with professional standards which do not cause harm, offense, or illegitimate results being produced. Also, communication with University Administrators must be kept to a high standard. The research of the project used legitimate and credible sources in order to support many of the project's main aims and objectives. This enabled me to continue to the implementation of the web application. When implementing the system, the code was kept in a readable and maintainable structure with comments to describe the functionality of the code. Diagrams and figures were used appropriately only to further illustrate and justify various explanations throughout the report. Throughout the entire process there was effective and clear communication with my supervisor. This included any questions I had about the project and associated challenges. For example, several discussions took place as to how the data was collected and the efficient removal of data when not required. After having many discussions throughout each stage of the project I was able to act honestly and maintain a high moral conduct throughout the project.

6.5 Statement of Ethics Conclusion

I have gone into detail about the many legal, ethical, social, and professional considerations for this project. I have been able to identify areas of concern and examples of where the appropriate use of judgment was used. I believe I have provided work within the report abiding to with a responsibility to uphold high moral ethics and standards along with BCS conduct protocols.

7. References

1 Schneider, C., 2021. The biggest data challenges that you might not even know you have - Watson Blog. [online] Watson Blog. Available at: <https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/> [Accessed 8 February 2021].

2 Devika, M., Sunitha, C. and Ganesh, A., 2021. Sentiment Analysis: A Comparative Study on Different Approaches. [Accessed 8 February 2021].

3 SearchCIO. 2021. What is SMAC (social, mobile, analytics and cloud)? - Definition from WhatIs.com. [online] Available at: <https://searchcio.techtarget.com/definition/SMAC-social-mobile-analytics-and-cloud: :text=SMAC%20(social%2C%20mobile%2C%20analytics%20and%20cloud)%20is%20the,e%2Dbusiness%20to%20digital%20business> [Accessed 8 February 2021].

4 Kumar, A. and Jaiswal, A., 2021. Systematic literature review of sentiment analysis on Twitter using soft computing techniques.

5 Tankovska, H. (2021). Global Social Media Ranking 2019 — Statistic. [online] Statista. Available at: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/ [Accessed 10 Feb. 2021].

6 ResearchGate. 2021. (PDF) The Tweets They are A-Changin: evolution of twitter users and behavior. [online] Available at: <https://www.researchgate.net/publication/291992322_The_Tweets_They_are_A-Changin_evolution_of_twitter_users_and_behavior> [Accessed 10 February 2021].

7 Guha, S., Joshi, A. and Varma, V. (2015). SIEL: Aspect Based Sentiment Analysis in Reviews. [online] Association for Computational Linguistics, pp.759–766. Available at: https://www.aclweb.org/anthology/S15-2129.pdf [Accessed 10 Feb. 2021].

8 Chowdhury, G.G. (2005). Natural language processing. Annual Review of Information Science and Technology, [online] 37(1), pp.51–89. Available at: https://strathprints.strath.ac.uk/2611/1/strathprints002611.pdf [Accessed 11 Feb. 2021].

9 KhanKhattak, F. (2019). A survey of word embeddings for clinical text. Journal of Biomedical Informatics: X, [online] 4, p.100057. Available at: https://www.sciencedirect.com/science/article/pii/S2590177X19300563 [Accessed 11 Feb. 2021].

10 Munesh Lakhey (2019). Word2vec Made Easy. [online] Towards Data Science. Available at: https://towardsdatascience.com/word2vec-made-easy-139a31a4b8ae [Accessed 11 Feb. 2021].

11 Tfidf.com. (2021). Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining. [online] Available at: https://tfidf.com [Accessed 11 Feb. 2021].

12 Tian, Y. and Lo, D. (2015). A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/7081879 [Accessed 11 Feb. 2021].

13 Daniel, J. and Martin, J. (2020). Speech and Language Processing Dependency Parsing. [online] . Available at: https://web.stanford.edu/ jurafsky/slp3/14.pdf [Accessed 11 Feb. 2021].

14 Ginter, F. (n.d.). Universal Dependencies. [online] universaldependencies.org. Available at: https://universaldependencies.org/introduction.html [Accessed 12 Feb. 2021].

15 Cambridge University Press (2009b). Tokenization. [online] Stanford.edu. Available at: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html [Accessed 12 Feb. 2021].

16 Cambridge University Press (2009a). Stemming and lemmatization. [online] Stanford.edu. Available at: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html [Accessed 12 Feb. 2021].

17 Cambridge University Press (2009a). Stemming and lemmatization. [online] Stanford.edu. Available at: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html [Accessed 12 Feb. 2021].

18 Palletsprojects.com. (2010). Welcome to Flask — Flask Documentation (1.1.x). [online] Available at: https://flask.palletsprojects.com/en/1.1.x/ [Accessed 12 Feb. 2021].

19 Microsoft (2020). ML.NET — Machine Learning made for .NET. [online] Microsoft. Available at: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet [Accessed 12 Feb. 2021].

20 pandas.pydata.org. (n.d.). API reference — pandas 1.1.4 documentation. [online] Available at: https://pandas.pydata.org/docs/reference/index.html [Accessed 12 Feb. 2021].

21 numpy.org. (2021). NumPy Reference — NumPy v1.19 Manual. [online] Available at: https://numpy.org/doc/stable/reference/index.html [Accessed 13 Feb. 2021].

22 TensorFlow (2019). TensorFlow. [online] TensorFlow. Available at: https://www.tensorflow.org/ [Accessed 13 Feb. 2021].

23 Scikit-learn (2019). scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/ [Accessed 13 Feb. 2021].

24 spaCy (n.d.). spaCy 101: Everything you need to know · spaCy Usage Documentation. [online] spaCy 101: Everything you need to know. Available at: https://spacy.io/usage/spacy-101whats-spacy [Accessed 14 Feb. 2021].

25 Pitrou, A. (n.d.). pickle5: Backport of the pickle 5 protocol (PEP 574) and other pickle changes. [online] PyPI. Available at: https://pypi.org/project/pickle5/ [Accessed 14 Feb. 2021].

26 Molnar, C. (n.d.). 4.2 Logistic Regression — Interpretable Machine Learning. [online] christophm .github.io. Available at: https://christophm.github.io/interpretable-ml-book/logistic.html [Accessed 14 Feb. 2021].

27 Tweepy (2019). Tweepy. [online] Tweepy.org. Available at: https://www.tweepy.org/ [Accessed 14 Feb. 2021].

28 www.saedsayad.com. (n.d.). Naive Bayesian. [online] Available at: https://www.saedsayad.com /naive_bayesian.htm [Accessed 14 Feb. 2021].

29 Goel, A., Gautam, J. and Kumar, S. (2016). Real time sentiment analysis of tweets using Naive Bayes. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/7877424 [Accessed 14 Feb. 2021].

30 scikit-learn (n.d.). 1.9. Naive Bayes — scikit-learn 0.24.2 documentation. [online] scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/naive$_b ayes.html[Accessed 14 Feb.$2021].

31 Scikit-learn (2009). 1.10. Decision Trees — scikit-learn 0.22 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/tree.html [Accessed 15 Feb. 2021].

32 Thatascience (2018). Gini Index vs Entropy Information gain — Decision Tree — THAT-A-SCIENCE. [online] thatascience.com. Available at: https://thatascience.com/learn-machine-learning/gini-entropy/: :text=Gini%20index%20and%20entropy%20are%20the%20criteria%20for%20calculating%20information%20gain.text=Both%20gini%20and%20entropy%20are [Accessed 15 Feb. 2021].

33 Tutuianu, S. (2016). REAL TIME SENTIMENT ANALYSIS Third Year Project Report. [online] . Available at: http://studentnet.cs.manchester.ac.uk/resources/library/3rd-year-projects/2016/cristian-stefan.tutuianu.pdf [Accessed 16 Feb. 2021].

34 Brand Mention (n.d.). SocialMention — Free real-time social mentions search analysis. [online] brandmentions.com. Available at: https://brandmentions.com/socialmention/ [Accessed 16 Feb. 2021].

35 Social Searcher (n.d.). Social Searcher - Free Social Media Search Engine. [online] www.social-searcher.com. Available at: https://www.social-searcher.com [Accessed 16 Feb. 2021].

36 kaggle.com. (n.d.). Sentiment140 dataset with 1.6 million tweets. [online] Available at: https://www.kaggle.com/kazanova/sentiment140 [Accessed 16 Feb. 2021].

37 Zamboni, J., 2021. The Advantages of a Large Sample Size. [online] Sciencing. Available at: <https://sciencing.com/advantages-large-sample-size-7210190.html> [Accessed 15 March. 2021].

38 Scikit-learn (2018). sklearn.feature_extraction.text.CountVectorizer — scikit-learn 0.20.3 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html [Accessed 15 Mar. 2021].

39 Scikit-learn (2014). sklearn.linear_model.LogisticRegression — scikit-learn 0.21.2 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Accessed 16 May 2021].

40 Scikit-learn (2019). sklearn.naive_bayes.MultinomialNB — scikit-learn 0.22 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html [Accessed 16 May 2021].

41 Ceballos, F. (2020). Scikit-Learn Decision Trees Explained. [online] Medium. Available at: https://towardsdatascience.com/scikit-learn-decision-trees-explained-803f3812290d: :text=max_depth%3A%20This%20determines%20the%20maximum [Accessed 16 May 2021].

8. Appendix

SAGE FORM

SAGE-HDR

Response ID	Completion date
640816-640807-77644220	5 May 2021, 20:27 (BST)

1	**Applicant Name**	Rishi Patel
1.a	**University of Surrey email address**	rp00463@surrey.ac.uk
1.b	**Level of research**	Undergraduate
1.b.i	**Please enter your University of Surrey supervisor's name. If you have more than one supervisor, enter the details of the individual who will check this submission.**	Liqun Chen
1.b.ii	**Please enter your supervisor's University of Surrey email address. If you have more than one supervisor, enter the details of the supervisor who will check this submission.**	liqun.chen@surrey.ac.uk
1.c	**School or Department**	Computer Science
1.d	**Faculty**	FEPS - Faculty of Engineering and Physical Sciences Sciences

1 / 6

2	**Project title**	Sentiment Analysis in Social Media Monitoring

3	**Please enter a brief summary of your project and its methodology in 250 words. Please include information such as your research method/s, sample, where your research will be conducted and an overview of the aims and objectives of your research.**	This project will develop and implement a web-based solution for sentiment analysis and ABSA in parallel to providing in-depth research of sentiment analysis techniques and machine learning algorithms. The user will be able to enter a topic on the web application to search which fetches social media sentiment from Twitter. Each sentiment retrieved will be fed through the sentiment classification model and ABSA model. A thorough analysis of the Multinomial Naïve Bayes, Logistic regression, and vectorizer algorithms will be made to determine the best model for the sentiment classification. The ultimate goal would be to reach the most effective model that can handle the correct classification of sentiment and be performance efficient whilst making predictions so the end-user can minimise the time of waiting for their results. After, the ABSA model with NLP processing will be created to delve deeper into the features of the sentiment and extract key aspects. Each model will be tested, and evaluation metrics will be compared in the research. The evaluation metrics to consider are accuracy, F-1 score, true positive rate, false-positive rate, and additional metrics such as user acceptance testing. After, a model will be chosen and implemented for sentiment classification and ABSA.

		The project will then explore the results stage of a machine learning web application utilising both models and subsequentially providing accurate sentiment classifications and feature identifications to the user on the front-end of the web app. A business will be able to use the sentiment classifications and aspects identified to further adapt or change certain processes that will help monitor and improve their reputation/branding.

4	**Are you making an amendment to a project with a current University of Surrey favourable ethical opinion in place?**	NO

5	**Does your research involve any animals, animal data or animal derived tissue, including cell lines?**	NO

7	**Does your project involve any of the following: human participants (including human data and/or any human tissue*); or is your project linked to engineering and/or the physical sciences?**	NO

8	**Will you be accessing any organisations, facilities or areas that may require prior permission? This includes organisations such as schools (Headteacher authorisation), care homes (manager permission), military facilities etc. If you are unsure, please contact RIGO.**	NO

9	**Will you be working with any collaborators or third parties to deliver any aspect of the research project?**	NO

10	**Is your project a service evaluation or an audit?**	NO

11	**Does your funder, collaborator or other stakeholder require a mandatory ethics review to take place at the University of Surrey?**	NO

12	**Are you undertaking security-sensitive research, as defined in the text above?**	NO

30	**Declarations**	• I confirm that I have read the University's Code on Good Research Practice and ethics policy and all relevant professional and regulatory guidelines applicable to my research and that I will conduct my research in accordance with these. • I confirm that I have provided accurate and complete information regarding my research project • I understand that a false declaration or providing misleading information will be considered potential research misconduct resulting in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies • I understand that if my answers to this form have indicated that I must submit an ethics and governance application, that I will NOT commence my research until a Favourable Ethical Opinion is issued and governance checks are cleared. If I do so, this will be considered research misconduct and result in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies. • I understand that if I have selected any options on the higher, medium or lower risk criteria then I MUST submit an ethics and governance application (EGA) for review before conducting any

		research. If I have NOT selected any of the higher, medium or lower risk criteria, I understand I can proceed with my research without review and acknowledge that my SAGE answers and research project will be subject to audit and inspection by the RIGO team at a later date to check compliance
31	**If I am conducting research as a student:**	• I confirm that I have discussed my responses to the questions on this form with my supervisor to ensure they are correct. • I confirm that if I am handling any information that can identify people, such as names, email addresses or audio/video recordings and images, I will adhere to the security requirements set out in the relevant Data protection Policy • Not applicable, I am not a student

User Guide

This guide will show how to run the Flask Sentiment Analysis Web Application on the production server and locally. In addition, a guide to run individual model training and running the vectorizer parameter tuning will be discussed.

Basic Feature Instructions

Once successfully on the Flask Sentiment Analysis Web Application the user can enter a topic and select amount of Tweets via the submission form which will search Twitter. The Tweets can be seen under the "Tweets" tab. The user can click the "Find our you predictions" button which will return predictions under "Predictions" tab and generate a pie chart. Finally, the user clicks on the "Aspects" tab and the the "Find our your Aspects!" button which will identify the aspects found for each Tweet. Please also reference the demo video for any further assistance or clarification.

Production

In order to run the Flask Sentiment Analysis Web Application on a production server visit the following URL: flasksentimentanalysis.herokuapp.com

Local

PLEASE NOTE ALL CODE WAS CREATED IN A VIRTUAL ENVIRONMENT WHICH WILL REQUIRE LIBRARIES TO BE INSTALLED.

In order to run the Flask Sentiment Analysis app locally, there are prerequisites which must be installed. Firstly, Python 3.7.8 must be installed. Secondly, the requirements from the requirements.txt file. The requirements can be installed by running the following command: 'pip install -r requirements.txt' Also, in order to install the prerequisites a code editor can be used such as VS Code which will allow easy access to the Terminal.

Once the prerequisites are installed then the user must "cd" into the Flask-App directory. In the Terminal a Redis server must be initiated for the background jobs. This can be done by typing "redis-server" into the Terminal.

Once the Redis server is started, a new Terminal must be opened and the command "python worker.py" must be ran. This will make sure the worker thread is started so jobs can be queued.

Finally, in order to start the application the command "python app.py" must be ran inside a new Terminal window. The Terminal will output a localhost address which can be copied into the browser and the Flask Sentiment Web Application can now be used.

Model Training

In order to run the TF-IDF and Word2vec models, the steps within the Local section must be completed. Once completed, the user must "cd" into the "Sentiment/Models" folder within the Flask-App directory. If TF-IDF models are to be ran then the user can "cd" into the "TF_IDF" directory. In order to begin model training the user must type 'python' and the name of file must be specified. Here is an example of the command once in the directory: 'python Multinomial_Naive_Bayes.py'

Before proceeding the "w2vUpdate.pkl" file from Surrey Drop Off must be placed inside the following directory: Flask-App/Sentiment/Models/Word2Vector.

If Word2Vector models are to be ran then the user can "cd" into the "Word2Vector" directory. In order to begin model training the user must type 'python' and the name of file must be specified. Here is an example of the command once inside the directory: 'python Decision_Trees.py'

If the vectoriser input tests need to be ran then the user must cd into the 'Sentiment/Models/TF_IDF' directory and run the following command in the Terminal: 'python Vectoriser_Input_Test.py'