**SOCIAL MEDIA SENTIMENT ANALYSIS USING MACHINE LEARNING**

**ANALYSIS**

**PROBLEM**

Social media sentiment analysis is the process of extracting the underlying sentiment (emotion) directed towards a person, product, organisation or some other entity. In today’s interconnected world, a vast quantity of data is generated daily which cannot be analysed using manual techniques. It is clear that analysis of this data can be very lucrative for corporations, who by identifying trends in user’s sentiments towards particular products, can make more profitable decisions in response. This is analysis of consumer sentiment. Simultaneously, many people have casual interest in the general sentiment of others but direct analysis of this sentiment can be difficult to perform in a decently low timeframe. For this reason, machine-learning backed sentiment analysis can be a useful and interesting tool.

Sentiment can be classified into positive, negative or neutral; the strength of the sentiment may also be classified or ranked on a numerical scale. This project may deal with simple binary classification or it may also recognise neutral statements, label tweets with their emotion (anger, joy, sadness) and put polarity of sentiment on a scale.

Twitter is an ideal platform for extracting sentiments for analysis. There are in the order of millions of tweets daily and Twitter has over 300 million users, which generates a large dataset. Furthermore, the sentiment statements themselves are ideal for the machine learning approach; the character limit of 140 characters means users write concise tweets often with a clear/single sentiment, and so using Tweets guarantees structurally similar training examples. Other examples of where the texts would be of similar length to Tweets are Youtube comments, Reddit comments and Google reviews.

**WHY SOCIAL MEDIA SENTIMENT ANALYSIS LENDS ITSELF TO A COMPUTATIONAL APPROACH**

The fundamental reason why social media sentiment analysis is best performed via a computational approach is the large dataset which must be analysed. Manual analysis is time-consuming, prone to human error and biases and limited in scope; the power of computation allows a vast quantity of data to be analysed efficiently and in a manner which makes the process cost-effective and the investment worth it. The most computation-hungry processes in machine learning are the iterative gradient descent and cost calculations on arrays of hundreds or thousands of items.

*Decomposition*

The production of a machine-learning algorithm which classifies sentiment may be roughly decomposed into the following steps:

1. Collation of a labelled dataset.
2. Cleaning of the data (correction of spelling, potential removal of emoticons and punctuation or replacement of slang terms).
3. Selection of a machine learning hypothesis (eg: logistic regression, SVM, etc.)
4. Selection of a learning algorithm (batch gradient descent, etc.)
5. Training of learning algorithm.
6. Application of learning algorithm to test data.
7. Evaluation of learning algorithm in terms of precision and recall.

The steps 3 to 7 may be repeated as many times as is necessary to arrive at a classifier which has the desired level of accuracy. Each step may further be decomposed into functions or modules (for example, with step 2, a function may be created to remove punctuation).

*Abstraction*

Abstraction is involved in two major facets of this problem. Firstly, abstraction is used when creating the machine learning algorithm, and secondly, it is used on the user end of the system.

Machine learning for natural language processing (which Twitter-based sentiment analysis is a subset of) is a clear example of abstraction. This is because the words, sentences, emoticons and punctuation which contain meaning comprehensible by a human, must be reduced to a vector of numerical features which an algorithm can process. Numbers are used to represent aspects of each sentence (word length, frequency of certain words, appearance of emoticons) and unnecessary details may be removed, which are classic features of abstraction.

Furthermore, in the user’s experience of the final product, the underlying calculations the classifier performs, and the individual Tweets which are being analysed are not displayed; instead a simplified graphical representation of the overall sentiment, for example, will be shown. Therefore, unnecessary information which may detract from the user experience (and which certainly doesn’t enhance it) is removed.

*Divide and conquer*

The steps which were mentioned in *Decomposition* can be further decomposed, and these resultant modules can be easily dealt with. The solutions to these modules can be combined and so divide and conquer is used. It is difficult to imagine divide and conquer techniques in code (such as recursion) being used in the classifier algorithm at this stage, but it has previously been used in conjunction with Support Vector Machines (a large-margin classifier), and so it is a possibility.

*Caching*

In gradient descent, at each iteration of the optimisation function, the coefficients of the parameters of the hypothesis are saved so as to calculate the cost of the hypothesis by multiplying the hypothesis vector by the feature vectors and use this to calculate the derivative of the cost. In a more general way, once I have trained the classifier on training data, its parameters will be stored in memory, so that when a user requests sentiment analysis, the classifier will not have to be retrained.

*Pattern recognition*

On a higher level than the mathematics of machine learning, the fundamental task of the classifier is to spot patterns in the Tweets which give clues as to what the sentiment of the Tweets is. Mathematical trends (regression) in the feature vectors are identified. It is, however, perhaps more fitting to call pattern recognition a feature of unsupervised learning (which this project is unlikely to involve) which takes a large quantity of unlabelled data and using clustering algorithms or otherwise, finds order in it.

**STAKEHOLDERS**

There are two major categories of stakeholders.

In the first category are internet users who have a particular interest in a certain topic, person, product, organisation or idea can use the sentiment analysis machine to gauge the overall Twitter opinion on said entity. This type of stakeholder is very common and includes a vast majority of internet users, whose use of my solution will be mainly recreational. They will be casual users of the product. However, I also anticipate people using my solution to gather evidence for their personal research projects; for example, a social sciences student wanting to analyse the current sentiment directed towards a politician, and compare it with the sentiment after a law is passed, will be able to utilise the sentiment analysis machine for this purpose.

In the second category are marketing branches at companies. They will find my solution useful as small company decisions like alterations to a product design can result in large outcomes in profitability. Modern social media platforms have changed the relationship between corporation and consumer. Nowadays, consumer sentiment has a significant bearing on marketing decisions and so understanding the sentiment and emotion directed towards the company (perhaps analysing the sentiment at crucial moments, such as after the release of the latest version of the company’s software) can be vital to staying profitable.

I have the following stakeholders in the first category:

* Stakeholder 1. She would use the final product for two reasons. Firstly, she has a casual interest in Twitter sentiment directed towards a variety of products (clothes, makeup, music albums) as she wants to understand people’s opinions before buying them. Secondly, she is further intrigued to see if an artificial intelligence can indeed classify emotions and sentiment.
* Stakeholder 2. She studies GCSE Geography, in which one of the areas of study is the human geography of Lagos (Nigeria). For her own depth of understanding and interest, it would be useful for her to get an overview of current sentiment toward new development projects in the city, such as Eco-Atlantic, and compare them quickly without spending a long time on Twitter etc..

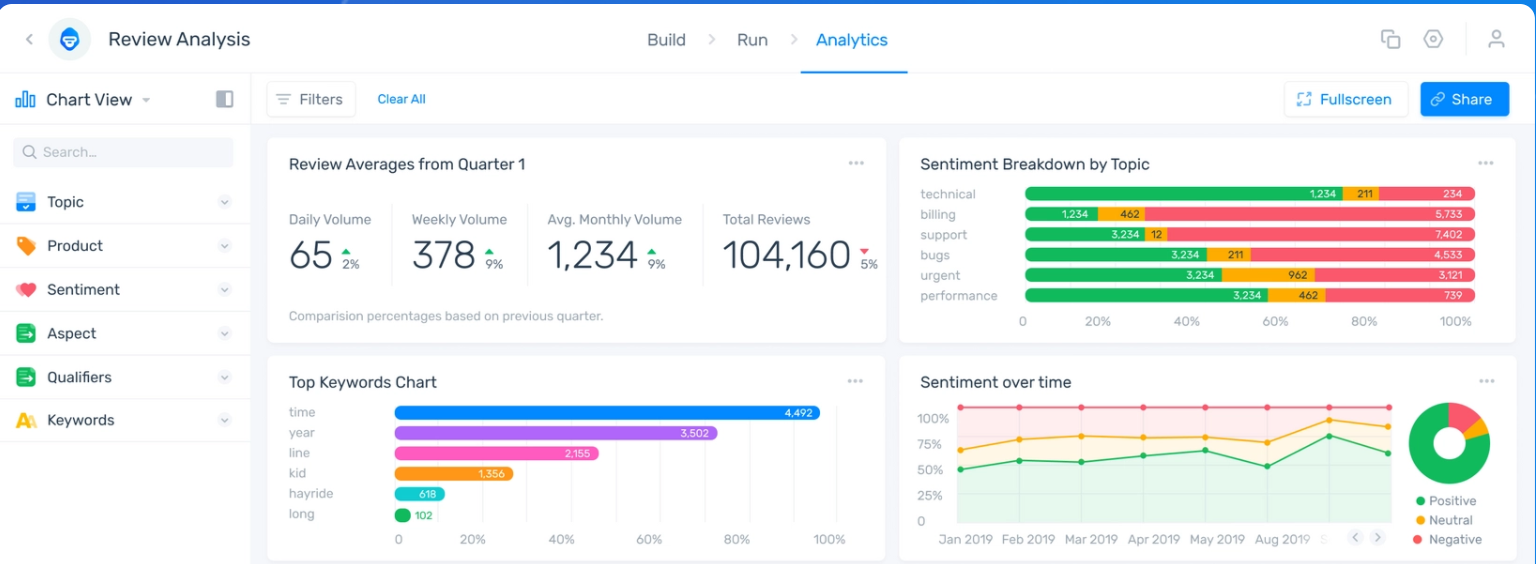
I have the following stakeholder in the second category:

* Stakeholder 3. She runs a small app development company and would like to monitor mentions of her company on social media, and see whether they are positive or negative. She would also like to identify key parts of her products that customers are talking about online. However, she does not have the time to conduct this manually and due to the small size of her company, there aren’t enough employees to do this for her. This is why she would like to utilise my solution to run a quick, automatic analysis.

**EXISTING SOLUTIONS**

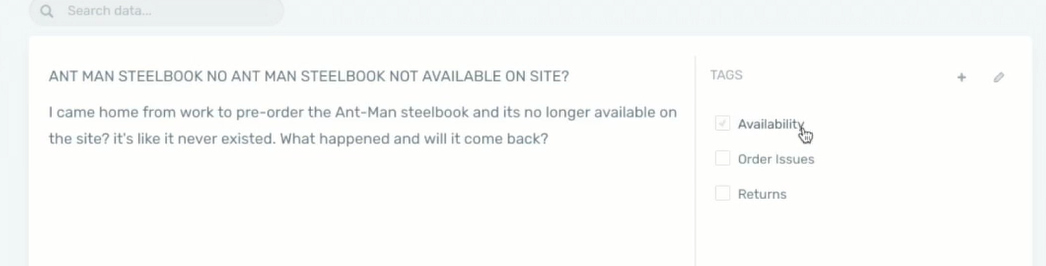
*MonkeyLearn*

Monkey Learn is a typical analysis tool marketed towards companies for identification of weak links in products and business processes through analysis of reviews, surveys and customer feedback. The solution is in the form of a studio (an application) which allows users to upload data and then view a dashboard of visualisations as shown above. Rather than using a pre-trained classifier, users may create their own models in a no-code environment.



I was not able to use MonkeyLearn myself as it requires a subscription fee but viewing YouTube demonstrations, reading the website and trying out the sentiment classifier myself, I am able to evaluate this software.

Monkey Learn has several functionalities which are outside the scope of my project; for example, I believe allowing the users to train their own models detracts from the purpose of my project, which is creating a classifier myself. Moreover, the training process is extremely tedious and requires users to label their own data as below.



For these reasons, I will avoid giving the option to users of creating their own model and focus on making a very good pre-trained model.

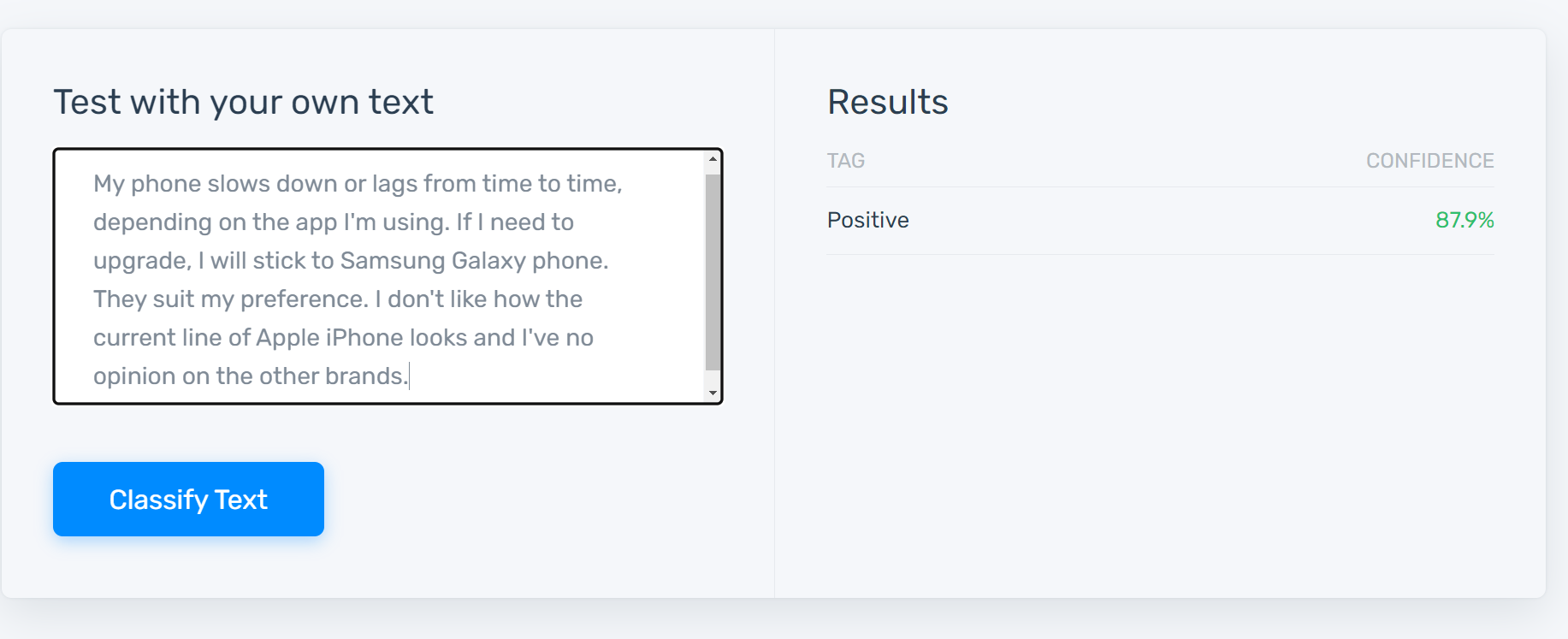
Another aspect of Monkey Learn which I will not include in my solution, simply because it is outside the field of what I would like to achieve and not directly related to Twitter sentiment analysis is allowing users to upload text in the form of a CSV or Excel file. I would prefer for my final solution to scrape Tweets automatically, rather than have users collect Tweets, for ease of use, a more pleasant user experience and accessibility even to those who are unfamiliar with coding.

One of the more relevant-to-my-solution features of Monkey Learn is the dashboard which provides a sentiment breakdown by topic, sentiment analysis over time and a keywords chart. Due to the increase in complexity it would demand, I would like to avoid having my solution provide an analysis of sentiment with time, though if time allows, this is a possibility. However, I like the idea of extracting keywords from data and presenting a sentiment breakdown by topic to the user as this makes it clearer to them opinion on the specific aspects of the product, person, etc., and provides much more intuition than simply returning a percentage indicating the polarity of the sentiment.

Monkey Learn also displays to the user specific examples of positive or negative sentiment, or examples of text regarding a specific topic, from the user-uploaded data. I would aim to allow users of my solution to be able to see example Tweets which the model has classified by sentiment or topic, allowing them to measure the precision of the model for themselves.

I would also aim to carry forward the visual simplicity of the user interface, such as the colour schemes and simple bar charts and minimal text.

Finally, the core of my project and of Monkey Learn is sentiment classification using machine learning. I tried their online demo version of the sentiment analysis tool using Tweets. The tool performed well, even recognising neutral statements, but was unable to classify Tweets comprising many sentiments, such as the one below. This will be a challenge when I produce my solution, but seeing as there is no source code available, I cannot comment on how I could alter the code to deal with this.



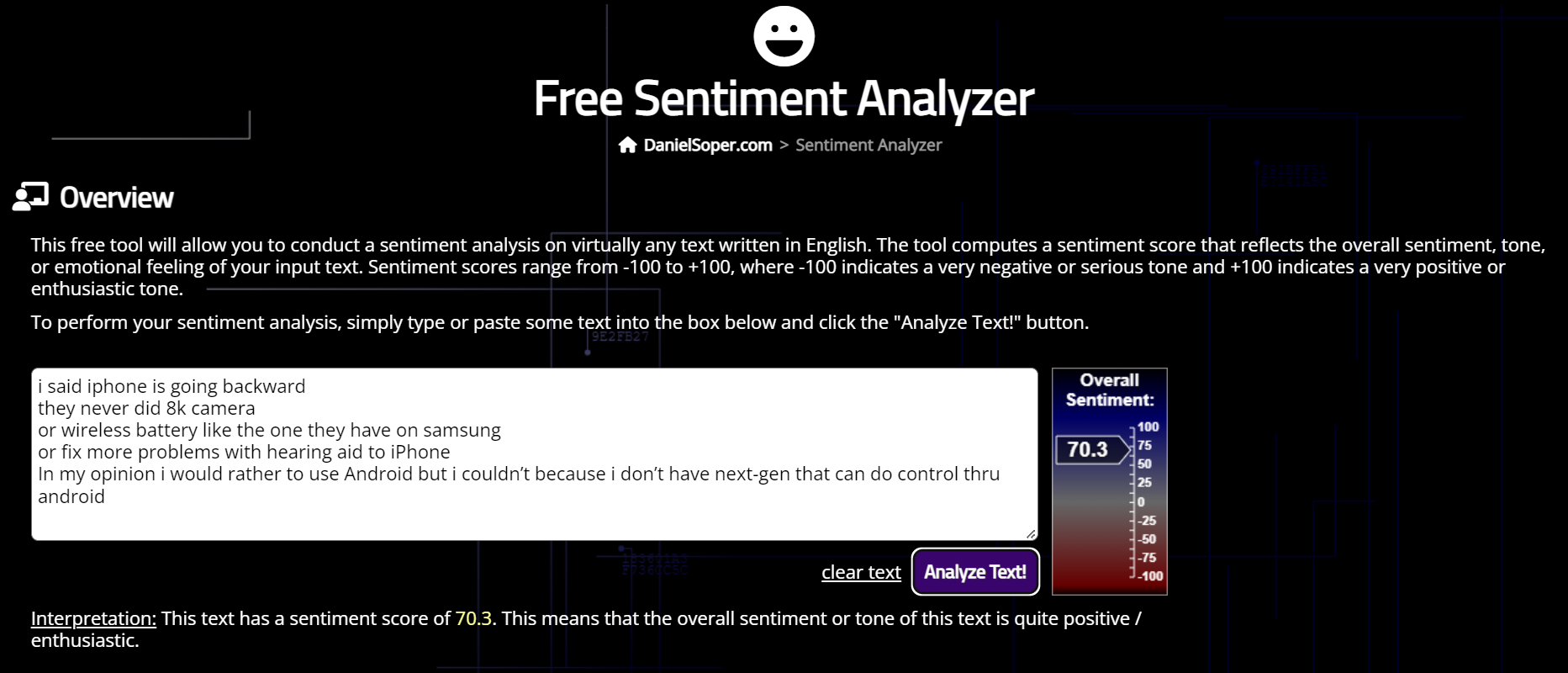
Features I aim to include in my solution:

* Simple, clean, and visually intuitive user interface
* Classification of topic (not only sentiment)
* Displaying examples of Tweets containing a particular sentiment or topic

*Free Sentiment Analyser. DanielSoper.com*

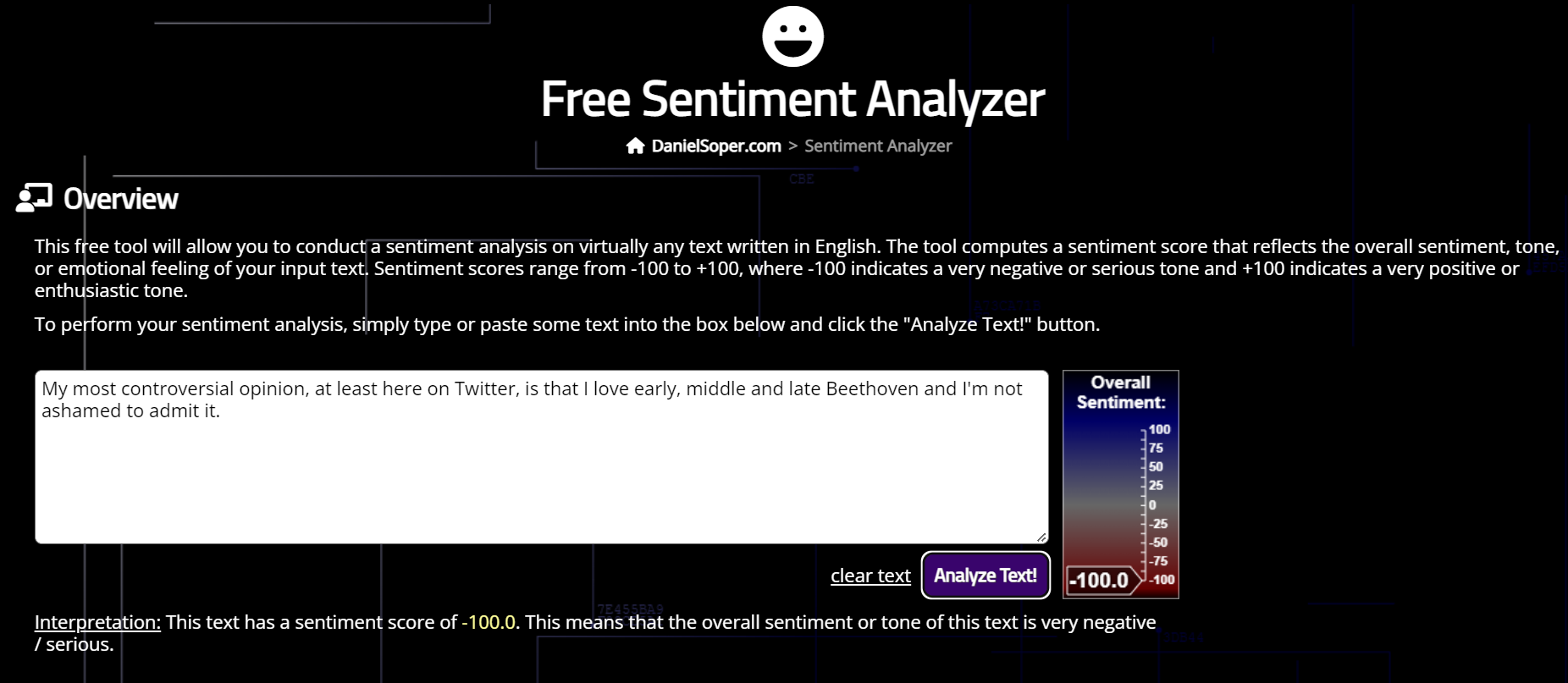
This free sentiment analysis tool scores sentiment on a scale of -100 to 100, where -100 correlates to completely negative sentiment and 100 to completely positive. It allows the user to simply copy and paste the text into the box and click Analyse Text. A score is returned within less than a second. This is the only analysis performed.

In my solution, I would like to provide a more comprehensive analysis than this, as a number does not reveal much about underlying sentiment. Furthermore, I feel it is unnecessary to give a single Tweet a sentiment score which is to three decimal places of accuracy, as the training data itself will not be labelled to that degree of precision. However, overall, the user of my solution may be presented with a number to 3 significant figures which would be the average sentiment of all Tweets analysed (not just a single Tweet, as in this analyser).



The ‘Interpretation’ section at the bottom of the screenshot is certainly useful as it allows the user to understand the meaning of the score in qualitative terms. I will carry this forward to my solution.

In the screenshot, the text I have copied into the box is a Tweet about iPhones, which clearly has a negative sentiment. However, this classifier believes the text is fairly positive. I would aim for my classifier to have a better accuracy than this one. The website explains that this machine learning model was trained on 8000 samples of long texts, so I understand that for my model to have good performance analysing Tweets, my training dataset should be comprised of Tweet-length texts, not longer samples. Furthermore, having more than 8000 training examples may be beneficial if my model suffers from high bias. On the other hand, using less than 8000 training examples would be better If this model has high variance, which is more likely considering this classifier is successful in many cases, but unsuccessful despite high confidence, in others (like the one in the screenshot above).



(The above Tweet clearly has a positive opinion directed towards Beethoven but the classifier believes the opinion is entirely negative. This is most likely because the training examples were much longer documents – therefore, I will train my classifier on short pieces of text.)

Finally, this tool brings to light the issue of a single piece of text containing both positive and negative sentiment on two or more different subjects. This can be an issue for my tool as many Tweets are comparative, displaying positive opinion on one product and negative opinion on a competing product. I will have to deal with this during collation of the dataset, perhaps by omitting Tweets with multiple subjects. In any case, I would aim for my classifier to, unlike this one, provide an accurate analysis of sentiment directed towards the subject which the user of my solution has interest in, and not just the overall sentiment of the Tweet.

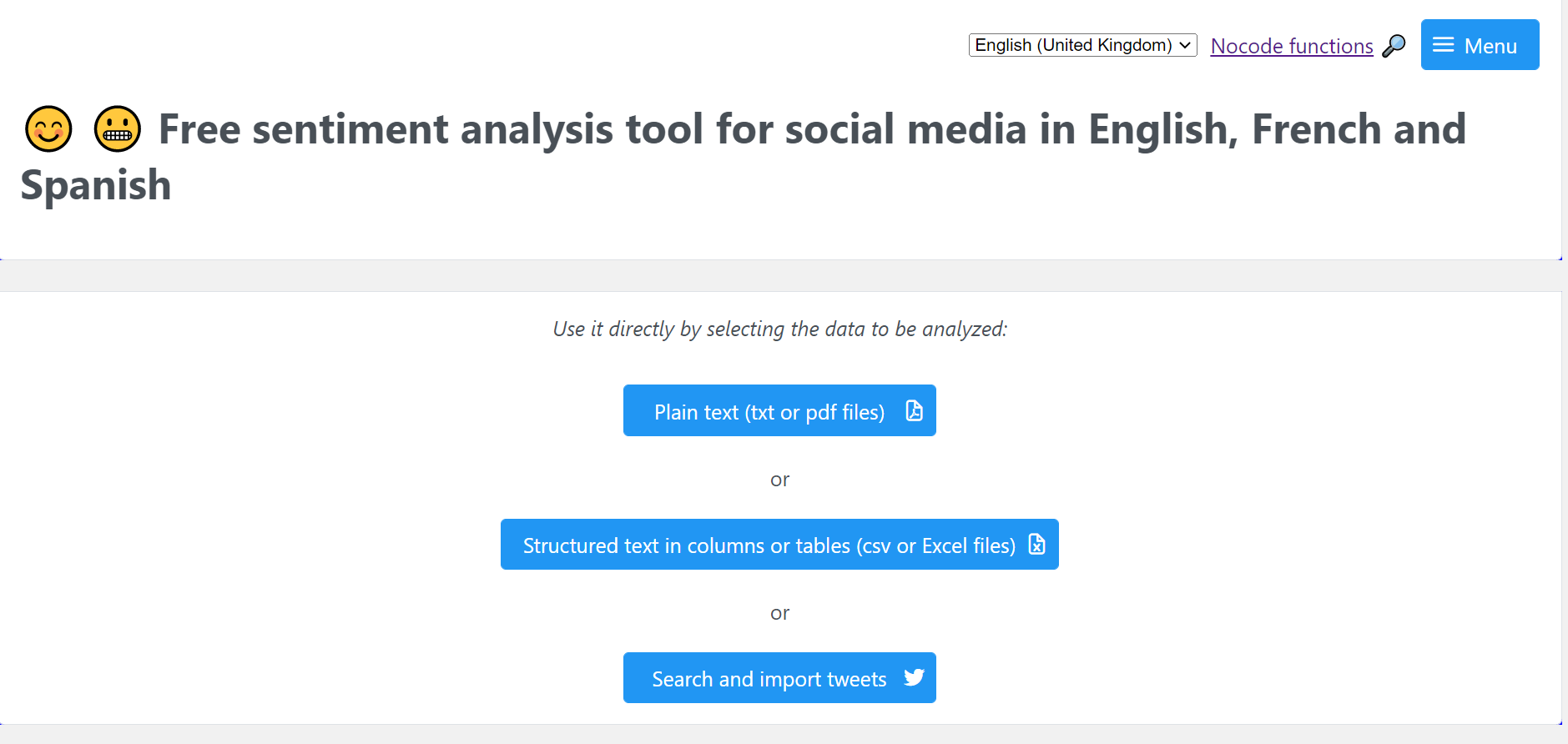
Features I aim to include in my solution:

* Qualitative interpretation/explanation of results

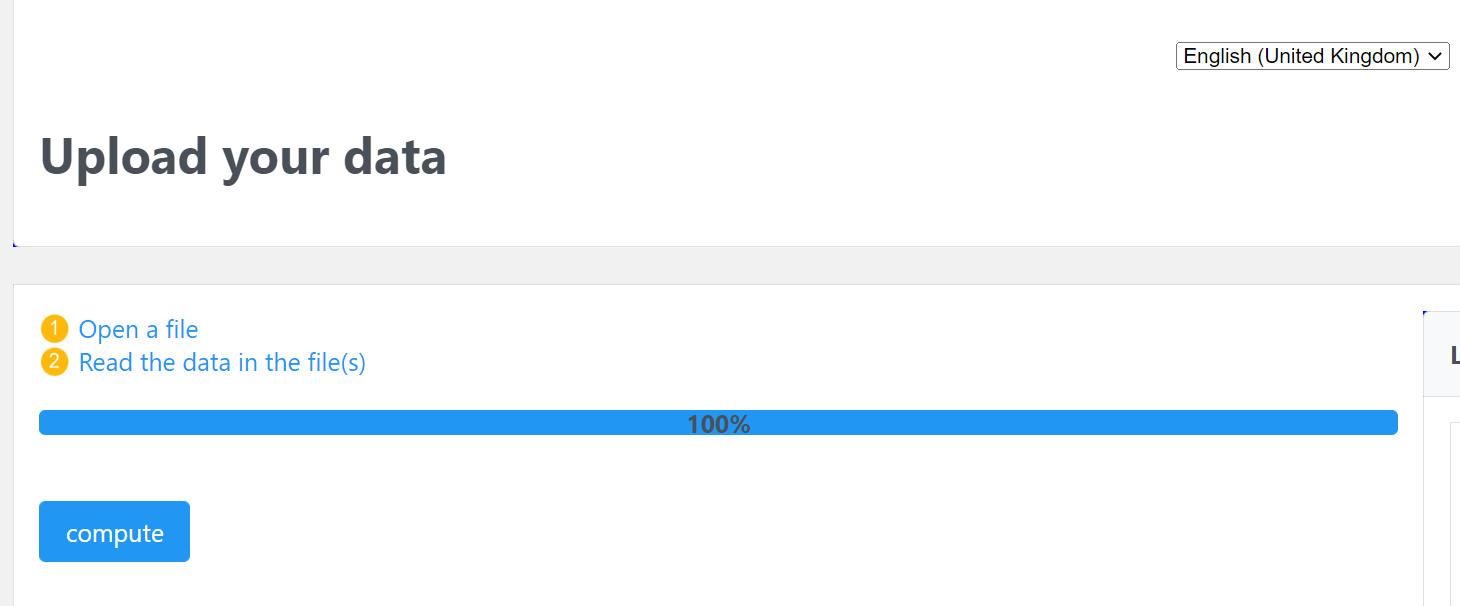
*Umigon*

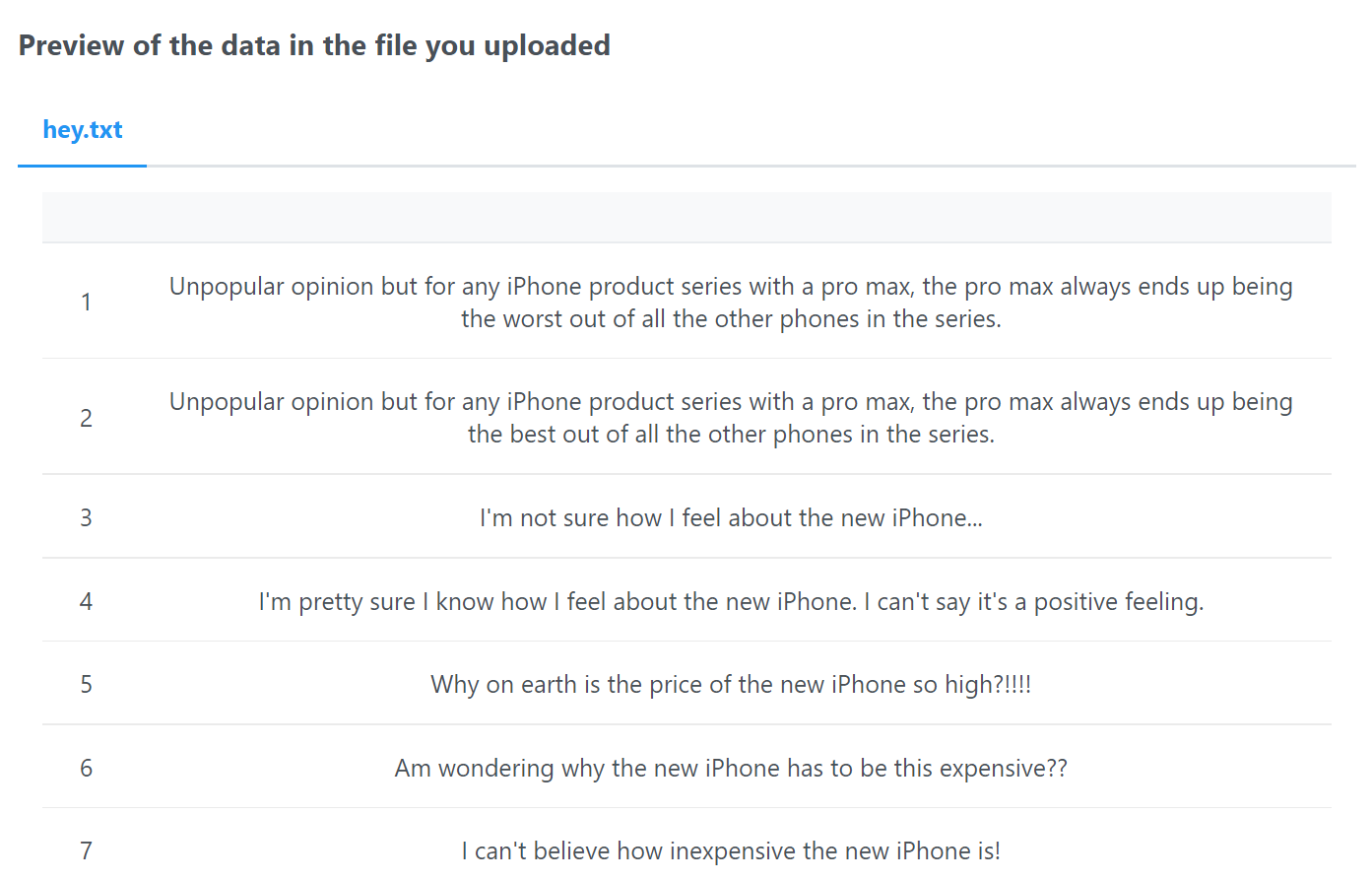
Umigon is a sentiment classifier which is powered by a series of logical functions, rather than machine learning. It allows the user to upload a text file, csv/Excel file, or Tweets using an API and decomposes the file into a series of records for each of which it classifies the sentiment into positive, neutral or negative. Up to 10000 pieces of data can be analysed.

Navigation of this process begins with the following screen, allowing the user to pick their method of upload.

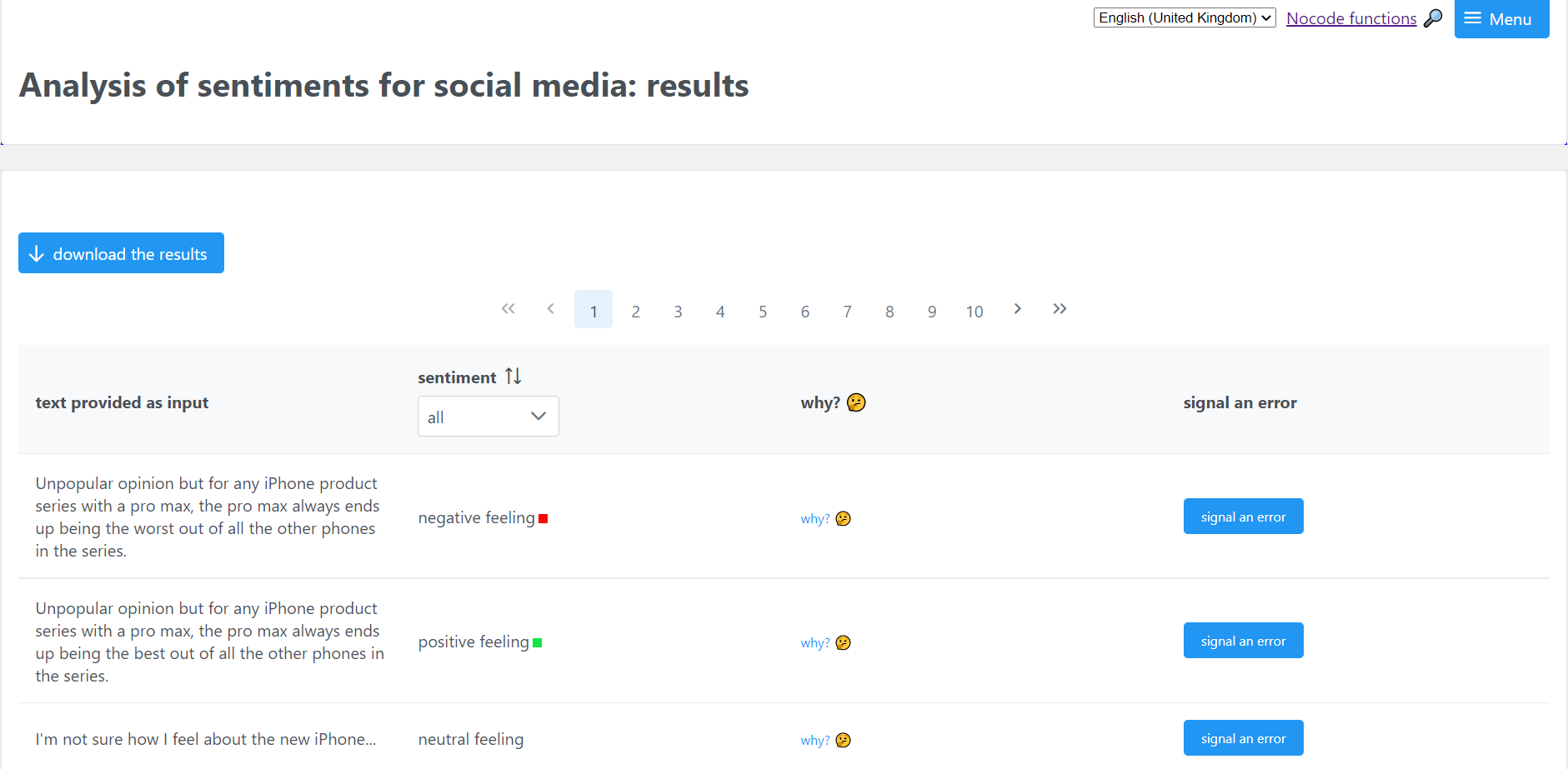


The user is then allowed to upload a file and view a preview of the texts identified in the file.





Finally a results page displays each piece of text alongside the result of classification, a button labelled ‘why?’ and a button labelled ‘signal an error’. On pressing the button labelled ‘why?’ an explanation as to how the sentiment was chosen is given, and on pressing the button labelled ‘signal an error’, a report is sent to the maker of the website to prompt further tweaks to the algorithm.

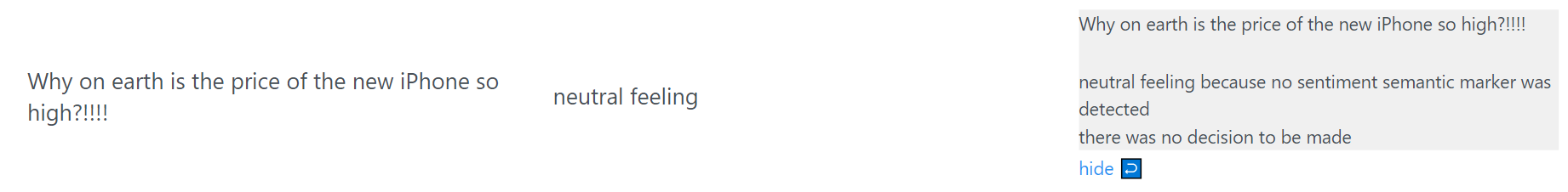


Any potential user would appreciate the array of options given for uploading data. However, I will not allow users the option to upload their own data in the text file format due to the increase in complexity of code. Something I have not yet considered is the option of letting users upload their own csv files; this would be preferable to the classifier simply analysing 50-500 top Tweets which have been extracted using the Twitter API, as this is a very rigid sample of data and may not provide the user with analysis of the data they want. Therefore, I will aim for my solution to allow users to upload their own files, albeit in a very specific format.

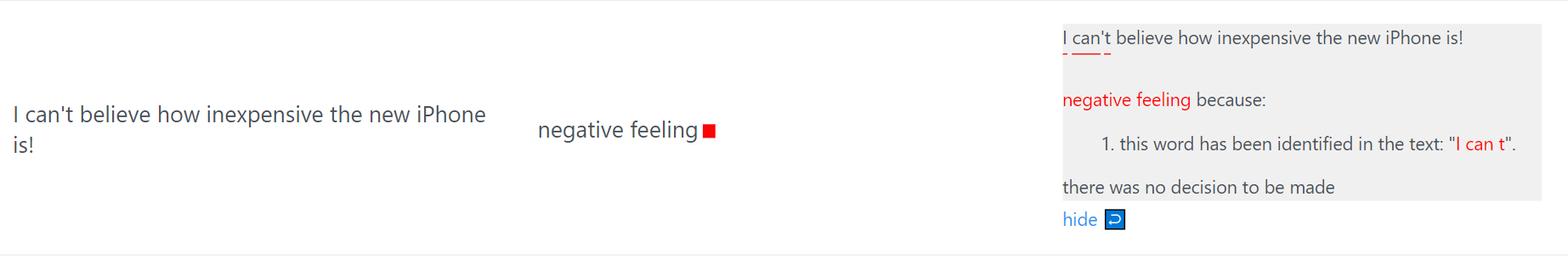
Many other features of this website, such as displaying the preview of uploaded data and allowing users to signal errors, will make my project increasingly difficult considering time constraints, even though it would be ideal to include these.

Having tested specific pieces of text, I find this classifier to work very well when analysing sentences which contain words indicating an obvious sentiment and it understands the rules of grammar (negation). However, it fails to classify several examples correctly, as shown below. I believe these to be a limitation of its overly-rigid design and limited dictionary of sentiment words, which indicates to me that a machine learning approach is better suited.

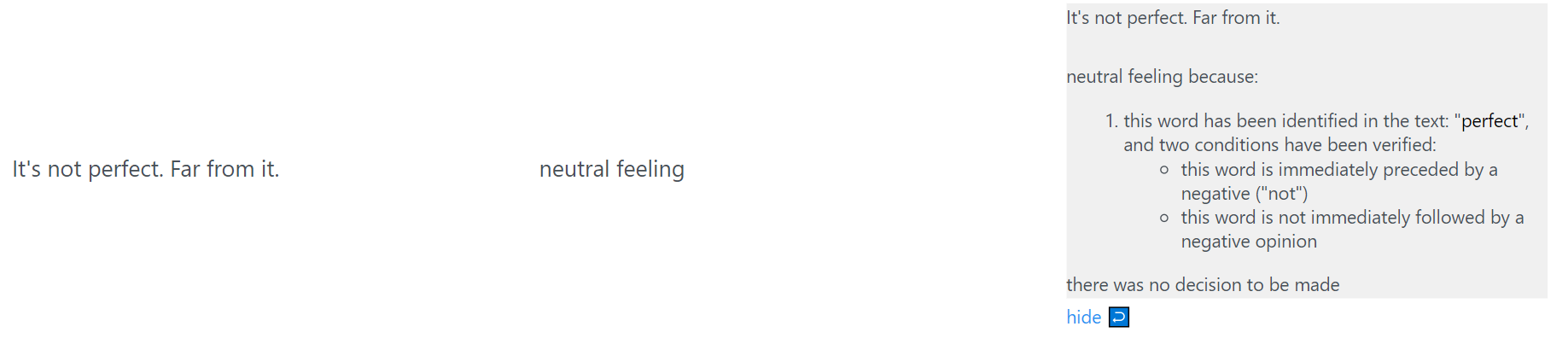
This first Tweet showcasing the common manner of speech in which a rhetorical question is used to express outrage or confusion, has strong negative sentiment, yet the classifier, unable to recognise any ‘sentiment semantic markers’, marks it as a neutral statement.



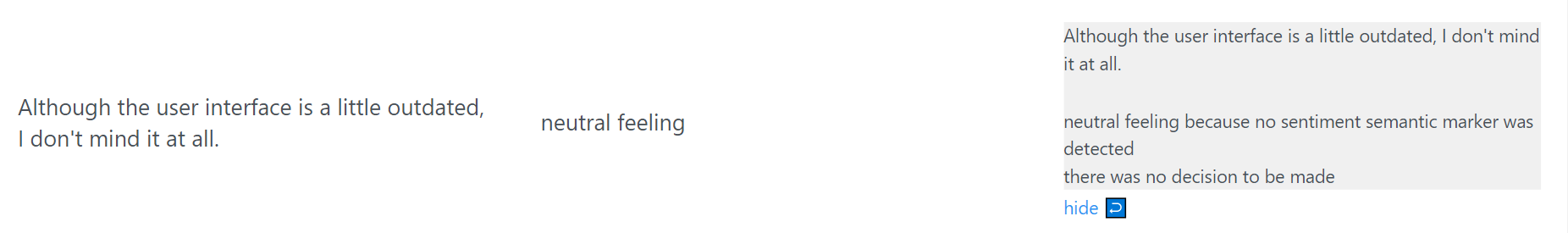
I have manufactured the following ‘Tweet’ to avoid positive or negative statements where the ‘new iPhone’ is the direct subject of the statement, instead making the first person the subject. The classifier does not recognise the obvious positive feeling (pleasant surprise) at the low price of the new iPhone. This language is once again typical of a Tweet, so this reinforces my vision of using machine learning for accurate analysis of real Tweets.



The downfall of the classifier in this third example is its inability to connect ‘Far from it’ to the adjective ‘perfect’ in the previous sentence. It overlooks the fact that ‘Far from it’ is indeed a reinforcement of the milder negative opinion in the first sentence: ‘It’s not perfect’. This tells me that a rule-based analysis approach would limit the capabilities of the solution.



Here the figure of speech ‘I don’t mind it at all’ which should imply mild positive sentiment, is not picked up as a sentiment semantic marker. I believe machine learning is therefore the better method, as a machine learning algorithm would identify a connection between the words ‘I don’t mind’, notice a trend in the training data that this phrase is associated with positive sentiment, and therefore adjust the hypothesis to recognise this in test data.



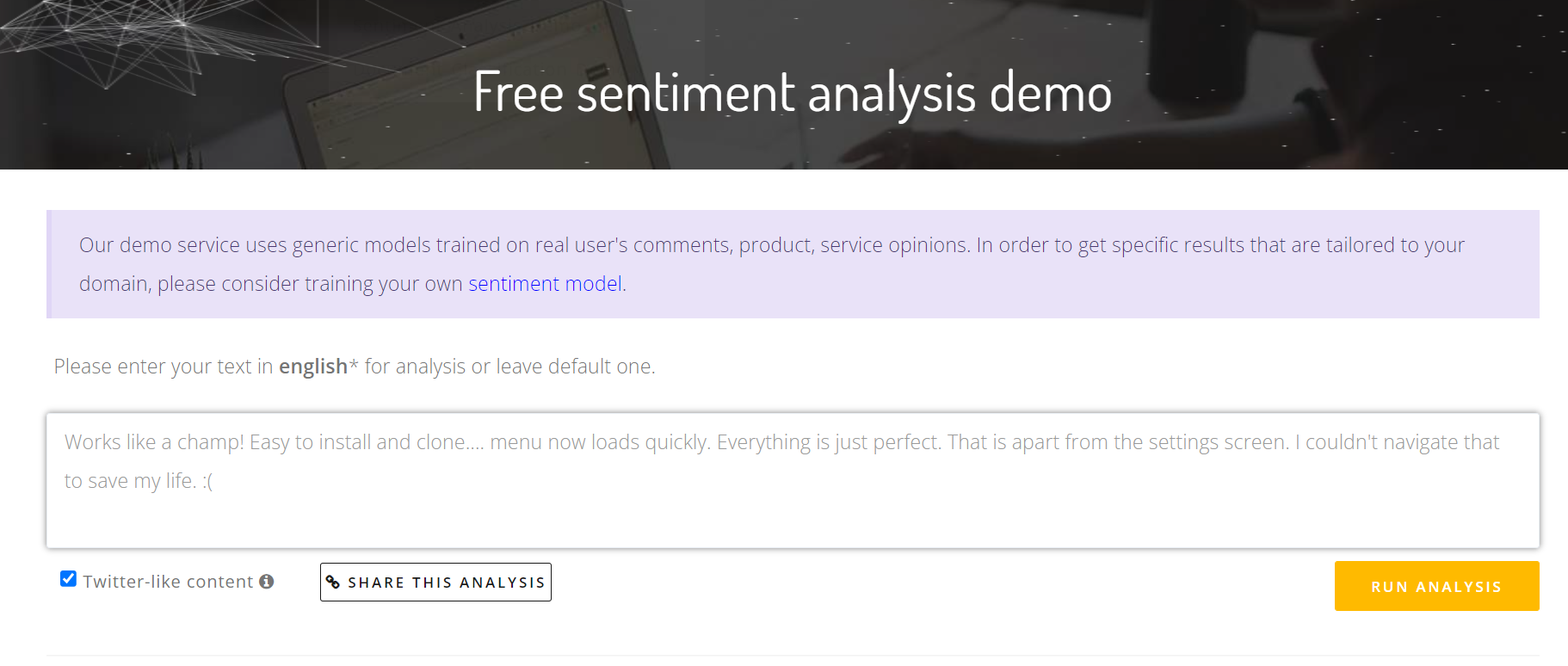
However, the use of a dictionary in conjunction with machine learning may prove useful in optimising the outputs. A dictionary-based approach, where certain words in Tweets are flagged positive or negative, and the preceding markers are analysed, may improve accuracy of my solution. Therefore, I would seriously consider this when designing my solution.

Features I would aim to include in my solution:

* Users can upload their own data
* Data is extracted from Twitter via the API
* Dictionary-based/rules-based approach in conjunction with ML

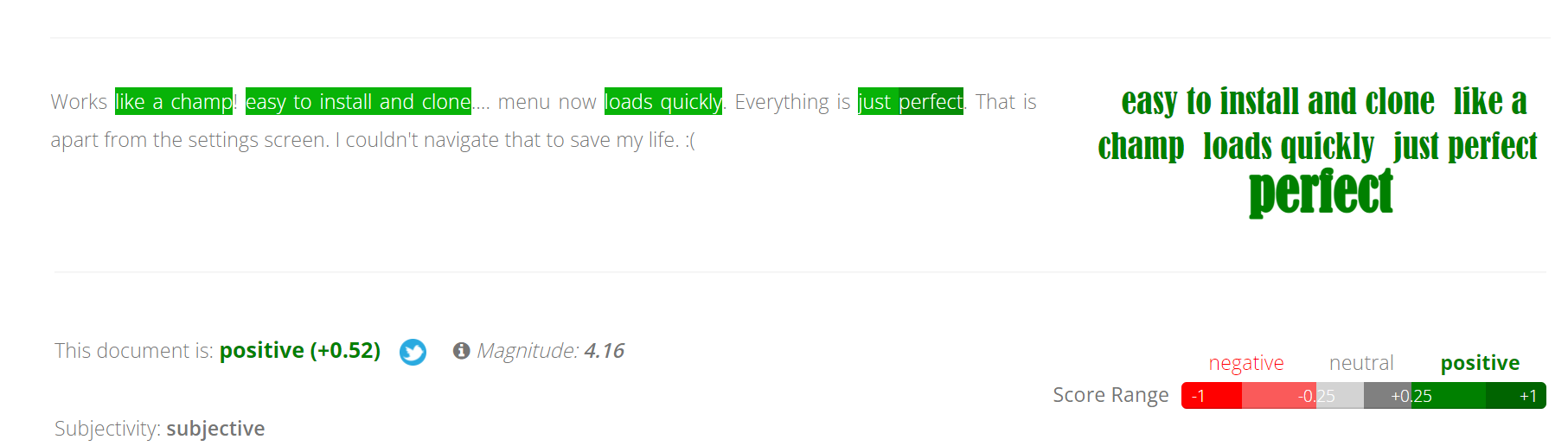
*Text2Data.com Free sentiment analysis demo*

This online, AI-powered classifier allows any length of text to be entered into the box. It displays a thorough analysis of the sentiment, not only scoring it from -1 to 1 based on positivity/negativity, but also scoring it on subjectivity, highlighting key phrases and producing a word cloud from these, detecting themes, keywords and categories, and scoring key phrases in the text based on sentiment individually.

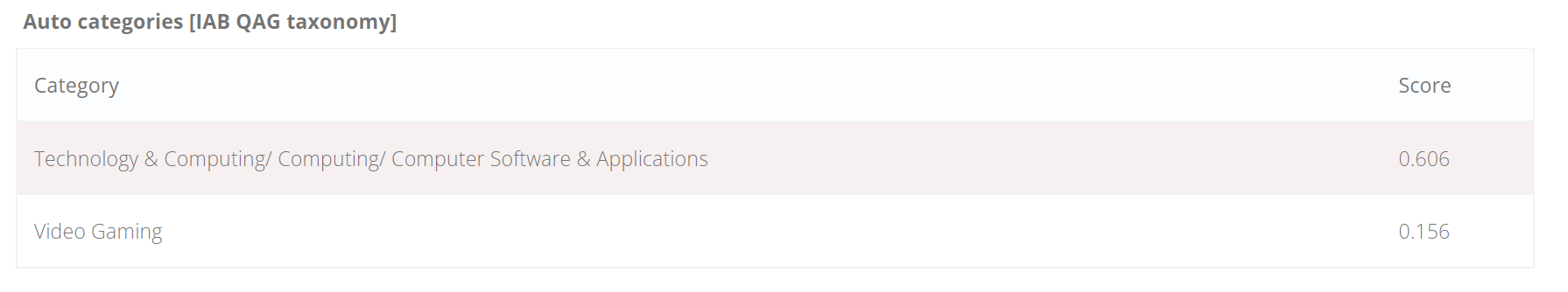


On clicking RUN ANALYSIS, the following graphs and tables are displayed.







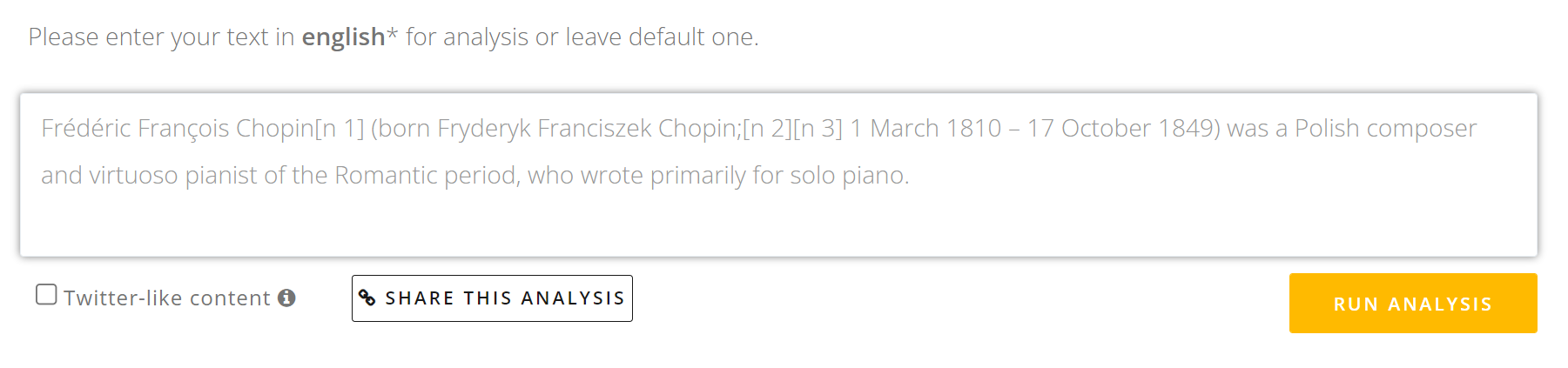


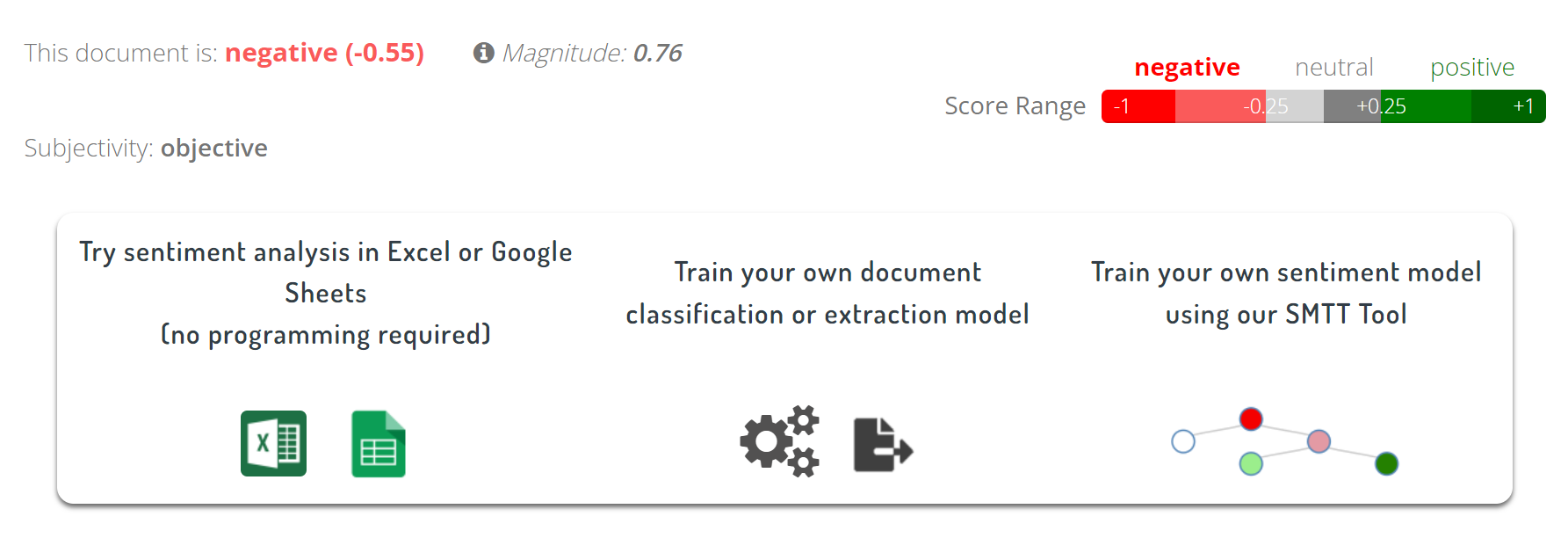
By reading the Tweet and analyses, it is clear that this classifier is the most accurate out of all previous solutions. I will not include the ‘Auto categories’ section in my solution, as these are highly general, and while identifying more specific subjects in each Tweet (such as the camera in a Tweet about iPhones) may be useful, these categories are too broad for that; the user of the classifier will already know the topics likely to arise in the Tweets they have uploaded/requested from Twitter.

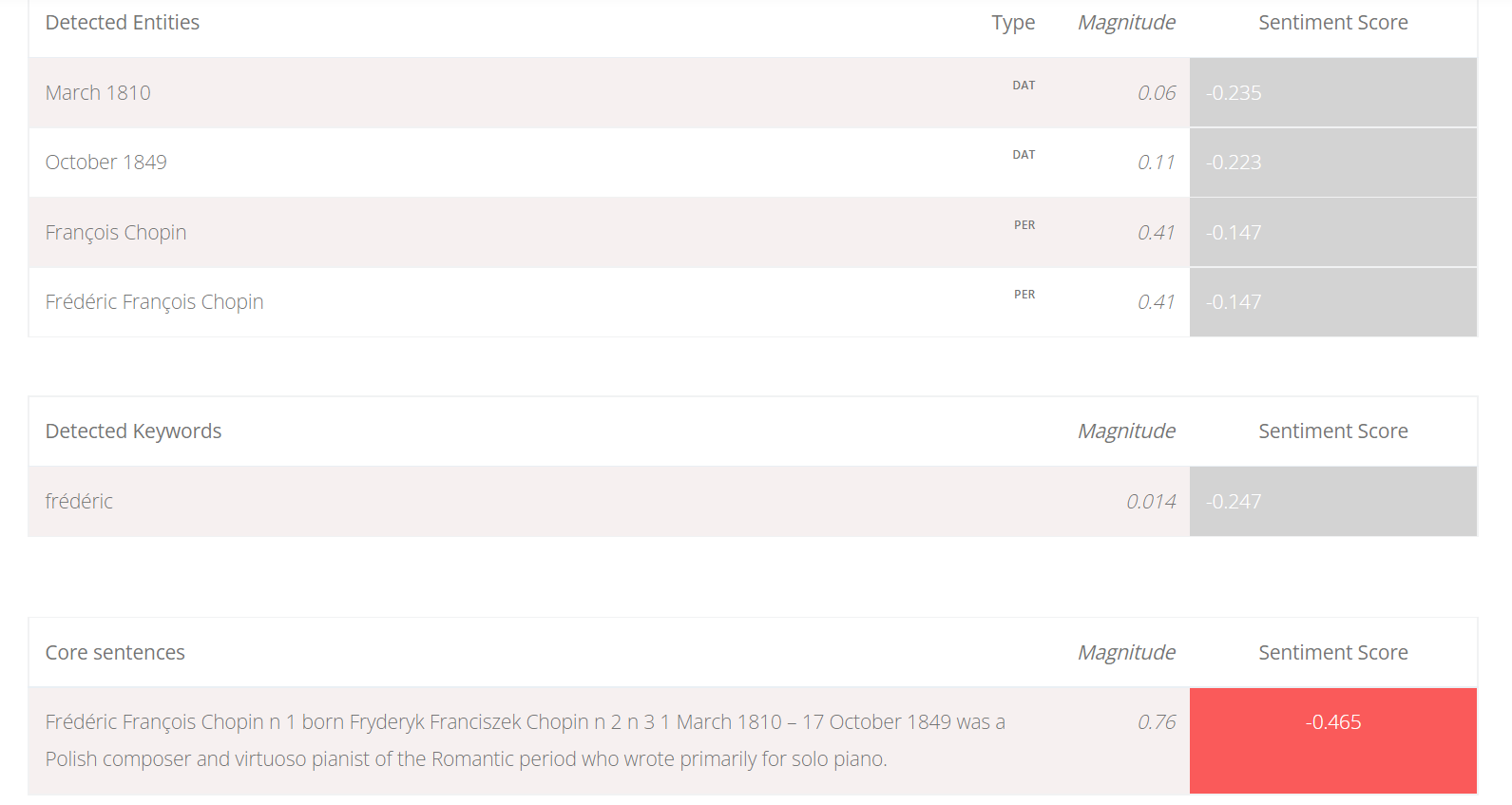
I would like to replicate the ability to detect keywords in Tweets, but not to display them individually as this website has done: instead I would aim to display the most common keywords across all Tweets analysed. The purpose of my system will be to give an overview of sentiment regarding a topic by analysing a large number of Tweets; a user would not have difficulty recognising sentiment in a single Tweet by taking the time to read it, and so I choose not to implement some other features from this solution as well, like the identification of ‘core sentences’ and ‘theme’ for each Tweet.

Another feature of this system that I may carry across to mine is the subjectivity analysis. I decided to test this feature further by copying the very objective first line of the Wikepedia article on the 19th century composer Chopin. The system correctly identified this text as objective, but unfortunately seemed to detect a moderately negative sentiment despite the fact that the sentiment was purely neutral. This implies that sentiment analysis can be difficult when it comes to objective Tweets (which may comprise a majority of the Tweets my system ends up analysing, depending on the extraction method).

In fact, objectivity and sentiment are almost mutually exclusive. Whilst an objective statement like a news report on a natural disaster may have a negative topic, it doesn’t contain negative sentiment. This means I will probably need to eliminate objective Tweets before conducting sentiment analysis; this does not, however, mean I have to forgo subjectivity/objectivity analysis.







A final feature of this website I believe would be good to include in my sentiment analysis project is a numerical indicator of the magnitude of sentiment. This system accurately recognised the first Tweet as having a much stronger sentiment (magnitude 4.16) compared to the Wikepedia text (magnitude 0.76). Implementation of a measure of the polarity of sentiment could increase the development time unreasonably, as I will have to collate a dataset where magnitude of feeling is a label for the data. A sufficiently large dataset of this type is unlikely to be freely available online and manual labelling is far too time consuming. However, ideally, this feature will be part of my system. Rather than displaying the magnitude of each individual Tweet as this system does, I may only display a single number to the user giving them an overview of the entire dataset. This is much more appropriate when the purpose of the solution is to analyse overall sentiment.

Features I aim to include in my solution:

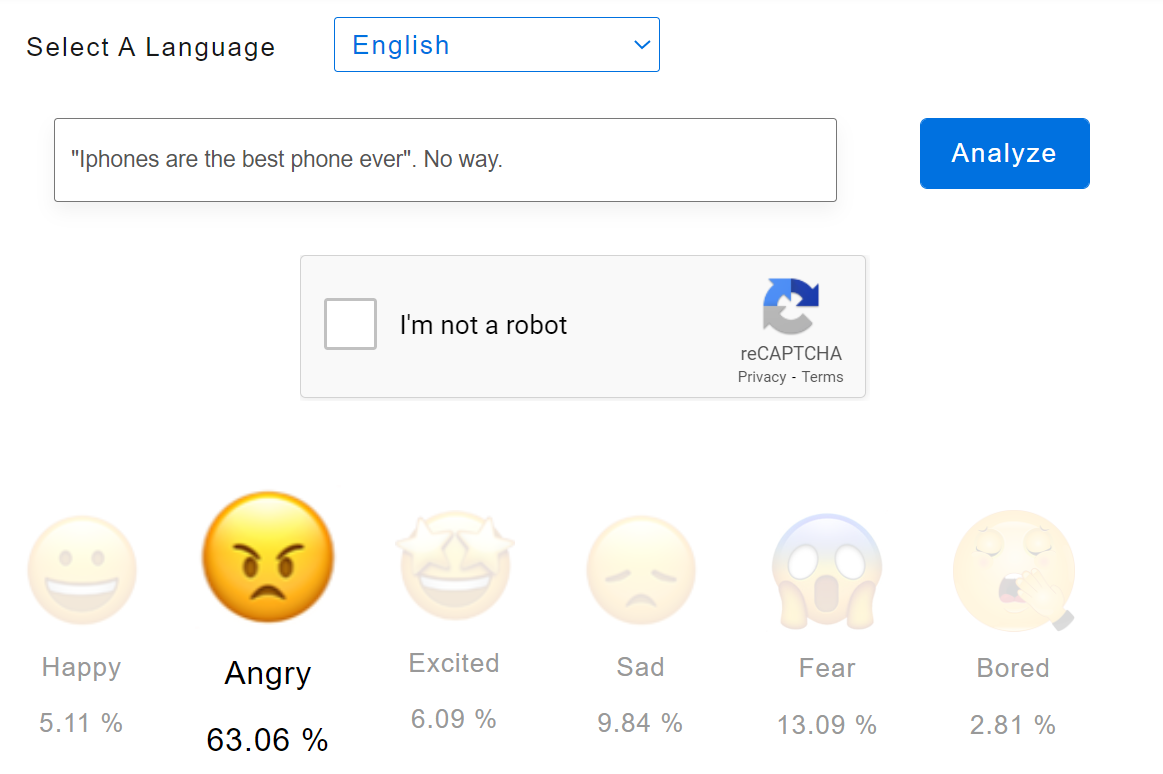
* Detecting and displaying keywords in a visually appealing way (possibly word cloud)
* Binary subjectivity analysis (subjective/objective)
* Displaying numerical magnitude of sentiment feeling

*Komprehend*

This website has a suite of classifiers, some powered by supervised learning algorithms (eg: long short term memory) and others by deep-learning models. Other than sentiment analysis, there is also intent, emotion and semantic analysis and sarcasm detection.

Access to the full range of tools is very expensive, but there are free demonstrations on their website. The system is very similar to Monkey Learn and is geared towards businesses.

The emotion analysis tool is what distinguishes Kcomprehend from the previous solutions, as so far, sentiment has only been classified as positive or negative. When text is entered into the box, the emotion analysis tool returns a percentage corresponding to each of the following emotions: happiness, sadness, anger, fear, excitement and boredom. I am interested in emotion analysis for my solution, as it provides a more comprehensive understanding of sentiment on the user’s chosen topic of interest. Emotion analysis is suited for both casual and business users of my potential solution.



Kcomprehend uses convolutional neural networks to classify emotion. The algorithm is trained on labelled datasets, which means I will have to find or make a dataset labelled with emotions to attain a similarly accurate level of classification. This could be a difficult task, especially considering the fact that most datasets I am likely to find online will either be on very generic and common topics, or highly specific topics (containing a lot of subject-specific knowledge which could easily bias the classifier). However, an ideal solution would include emotion analysis alongside sentiment analysis, so I would aim to carry this feature of Kcomprehend across to my solution.

Features I aim to include in my solution:

* Emotion analysis

*Overview of existing systems*

Now that I have observed and used 5 existing solutions for social media sentiment analysis, I will shortlist the features I aim to include in my solution below.

|  |  |
| --- | --- |
| 1 | Simple, clean, and visually intuitive user interface |
| 2 | Classification of topic |
| 3 | Displaying examples of text containing a particular sentiment or topic |
| 4 | Qualitative interpretation/explanation of results |
| 5 | Users can upload their own data |
| 6 | Data is extracted from websites/apps automatically via the API |
| 7 | Detecting and displaying keywords in a visually appealing way |
| 8 | Subjectivity analysis |
| 9 | Magnitude of feeling analysis |
| 10 | Emotion analysis |

*Table 1*

**STAKEHOLDER SURVEYS**

Having analysed existing solutions for social media sentiment analysis to get an overview of the potential features I could include in my solution, I have prepared questions for each of my stakeholder to allow me to understand what they want from an ideal solution. The questions will have a direct correspondence with the 10 features I aim to include, which I have summarised in Table 1. With this survey, I can evaluate the necessity of the aforementioned features.

These are my first set of questions. After reading the answers I will approach my stakeholders again with follow-up questions.

For all stakeholders:

1. **How interesting do you find it to read other people’s opinions online? Choose one option.**

Very interesting,

Quite interesting

I don’t read other’s opinions online

Uninteresting to the point of being annoying

1. **How often do you read other people’s opinions and sentiments? Choose one option.**

Everyday, for over an hour

Everyday, for less than an hour

A few times a week

Only occasionally; not habitually

1. **Which social media do you read opinionated/sentiment-driven comments/texts on the most? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others

1. **Would you prefer automatic sentiment analysis or to read and evaluate text yourself? Choose one option.**

Automatic only

I would find automatic useful/fun but I have the time to evaluate text myself

I don’t find significant benefits in automatic analysis compared to manual

1. **Which social media would you find it helpful to conduct sentiment analysis on? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others

1. **Which topics/people/products/hashtags would you be interested in for sentiment analysis? Write as much as you wish.**
2. **Would you be willing to collect text related to your chosen topic of interest yourself and upload it for sentiment analysis or would you prefer text to be collected automatically by the final product? Choose one option.**

I can collect text myself

I can collect text myself, but I would prefer automatic collection as an option too

I wouldn’t want to collect text myself; I would prefer only automatic collection

1. **In which ways would you like your data to be analysed? Choose one or more.**

Sentiment present in the text; positive/neutral/negative

Magnitude (strength) of feeling

Emotions present in the text

Objectivity/subjectivity of the text

Common words/phrases in the text

Topics in the text

1. **How would you like the analysis to be presented to you? Choose one or more options or write others.**

Graphs (pie chart, bar graphs, etc.)

Word clouds

Colour scales

Percentages/numerical values

1. **Would you like my program to display examples of text which contain a particular emotion/sentiment/topic/keyword/subjectivity/strength of emotion?**

Yes (please indicate emotion/sentiment/topic/keyword/subjectivity/strength of emotion

No, I am fine with just a general analysis

Only for stakeholders in the second category (business users):

1. **Do you think social media sentiment analysis will benefit your business? Choose one.**

Yes, very much

Yes, it may be of some benefit

I don’t know

No, it wouldn’t make a difference

1. **Would you find sentiment/emotion analysis on reviews or answers to surveys useful? Choose one.**

Yes, very much

Yes, it may be of some benefit

I don’t know

It wouldn’t make a difference

1. **On which platforms exist the texts you wish to analyse the sentiment of? Choose one or more.**

Twitter

Facebook

Instagram

Review websites

Survey applications

Emails

Other (please name)

These questions allow me to get an initial idea of what a user may want from my program.

Questions 1, 2 and 4 determine the necessity of the solution; if it is found through the answers that reading other people’s sentiment on particular topics is a frequent and common activity then a basic need for the solution is established.

Questions 3 and 5 will be used to find out the common social media platforms which users would like sentiment analysis for. This will let me design my solution to fit the social media necessary. These correspond to Feature 6 in Table 1: automatic extraction of text from social media sites.

Questions 6 allows me to work out the approximate size of the dataset my sentiment classifier will likely have to analyse. If the topics provided as answers to the question are narrow and highly specific, then the number of data items may be very small. If the topics are broad, then I must determine a strategy to select a suitably unbiased sample of data items (Tweets/comments). Moreover, if the users give a wide variety of topics altogether, it further reinforces the demand and interest in my project.

Question 7 gives me information on the requirements of my program. If users are willing to collate their own datasets and upload them, I will have to implement a mechanism for this. To inform the requirements further, I will follow up this question by asking which formats the users are comfortable with using (csv, Excel, txt file, etc). However, if users only want automatic data collection, all I will have to implement is use of an API for that particular social media app. This corresponds to Feature 5 in Table 1: the option of allowing users to upload their own data.

Question 8 and 9 tell me what the users want from the most fundamental aspect of my project; analysis of sentiment. This will allow me to see the scope of analysis required. I will follow up this question to obtain more detailed requirements, such as whether the user wants analysis of individual data items and the exact emotions they would like the classifier to identify. These questions correspond to Features 2, 7, 8, 9 and 10 in Table 1.

Question 10 corresponds to Feature 3 in Table 1. From the answers, I will know whether it will be necessary to display specific examples of text which has been labelled a certain way by the classifier to the user.

Questions 11 and 12 have the same role as Questions 1 and 2 but this time they are geared towards business users. Question 13 has more options than Question 3, as business users may wish to analyse text beyond social media.

I gave this survey to my stakeholders and obtained the following responses.

**Stakeholder 1**

1. **How interesting do you find it to read other people’s opinions online? Choose one option.**

Very interesting,

Quite interesting

I don’t read other’s opinions online

Uninteresting to the point of being annoying

1. **How often do you read other people’s opinions and sentiments? Choose one option.**

Everyday, for over an hour

Everyday, for less than an hour

A few times a week

Only occasionally; not habitually

1. **Which social media do you read opinionated/sentiment-driven comments/texts on the most? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others (Reddit)

1. **Would you prefer automatic sentiment analysis or to read and evaluate text yourself? Choose one option.**

Automatic only

I would find automatic useful/fun but I have the time to evaluate text myself

I don’t find significant benefits in automatic analysis compared to manual

1. **Which social media would you find it helpful to conduct sentiment analysis on? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others

1. **Which topics/people/products/hashtags would you be interested in for sentiment analysis? Write as much as you wish.**

#avatar2 #bts #straykids #JYP, plants, Franz Liszt, actors

1. **Would you be willing to collect text related to your chosen topic of interest yourself and upload it for sentiment analysis or would you prefer text to be collected automatically by the final product? Choose one option.**

I can collect text myself

I can collect text myself, but I would prefer automatic collection as an option too

I wouldn’t want to collect text myself; I would prefer only automatic collection

1. **In which ways would you like your data to be analysed? Choose one or more.**

Sentiment present in the text; positive/neutral/negative

Magnitude (strength) of feeling

Emotions present in the text

Objectivity/subjectivity of the text

Common words/phrases in the text

Topics in the text

1. **How would you like the analysis to be presented to you? Choose one or more options or write others.**

Graphs (pie chart, bar graphs, etc.)

Word clouds

Colour scales

Percentages/numerical values

1. **Would you like my program to display examples of text which contain a particular emotion/sentiment/topic/keyword/subjectivity/strength of emotion?**

Yes (please indicate emotion/sentiment/topic/keyword/subjectivity/strength of emotion

No, I am fine with just a general analysis

**Analysis of Stakeholder 1’s responses**

Stakeholder 1’s answers to the first two questions confirms that there is a customer-base for my product, as she is very interested in sentiment analysis and reads opinionated text often.

She believes sentiment analysis would be useful on almost all the options provided, which doesn’t narrow down the requirements for me, but she has suggested that she spends the most time on Reddit.

She has provided a wide variety of topics which she is keen on, which means a general purpose classifier would be necessary rather than a specialised on trained on specific terms. However, some of the hashtags she has provided are quite specific, so my solution must be able to give an accurate analysis even when only a few data items are available.

She wouldn’t be willing to collect text herself, which means it is necessary for an automatic collection system to be present in my solution. It also implies that I won’t need to provide the option for users to upload their own data, though this depends on the subsequent stakeholders’ responses.

The stakeholder is interested in analysis of strength of feeling and emotions present in the text. I will now have to follow up by asking the number of emotions she would like the system to be able to recognise. Her responses mean that I will probably include a numerical metric indicating magnitude of emotion in the text (a measure of how emotional the text is) and some visualisation of the emotions present. She is interested in all the listed methods of data representation, so I will endeavour to include these all in the final product. She also wants to be able to see examples of text labelled by sentiment

**Stakeholder 2**

1. **How interesting do you find it to read other people’s opinions online? Choose one option.**

Very interesting,

Quite interesting

I don’t read other’s opinions online

Uninteresting to the point of being annoying

1. **How often do you read other people’s opinions and sentiments? Choose one option.**

Everyday, for over an hour

Everyday, for less than an hour

A few times a week

Only occasionally; not habitually

1. **Which social media do you read opinionated/sentiment-driven comments/texts on the most? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others (Reddit)

1. **Would you prefer automatic sentiment analysis or to read and evaluate text yourself? Choose one option.**

Automatic only

I would find automatic useful/fun but I have the time to evaluate text myself

I don’t find significant benefits in automatic analysis compared to manual

1. **Which social media would you find it helpful to conduct sentiment analysis on? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others

1. **Which topics/people/products/hashtags would you be interested in for sentiment analysis? Write as much as you wish.**

Actors, songs, clothes, makeup, cinemas, restaurants, nightclubs, drinks etc.

1. **Would you be willing to collect text related to your chosen topic of interest yourself and upload it for sentiment analysis or would you prefer text to be collected automatically by the final product? Choose one option.**

I can collect text myself

I can collect text myself, but I would prefer automatic collection as an option too

I wouldn’t want to collect text myself; I would prefer only automatic collection

1. **In which ways would you like your data to be analysed? Choose one or more.**

Sentiment present in the text; positive/neutral/negative

Magnitude (strength) of feeling

Emotions present in the text

Objectivity/subjectivity of the text

Common words/phrases in the text

Topics in the text

1. **How would you like the analysis to be presented to you? Choose one or more options or write others.**

Graphs (pie chart, bar graphs, etc.)

Word clouds

Colour scales

Percentages/numerical values

1. **Would you like my program to display examples of text which contain a particular emotion/sentiment/topic/keyword/subjectivity/strength of emotion?**

Yes (please indicate emotion/sentiment/topic/keyword/subjectivity/strength of emotion

No, I am fine with just a general analysis

**Analysis of stakeholder 2’s responses**

From the answers to the first and second questions it seems stakeholder 2 is less interested in reading subjective text, but there is still an interest in the solution.

Her answer to questions 4 and 7 imply that conducting sentiment analysis in a short timeframe, and in an efficient and low-energy way is important to her, as she indicated she hasn’t the time for manual collection of data and manual analysis. It seems that the main use of my product for her is recreational and from the topics of interest provided, she will most likely use this classifier for fun.

As she has indicated all the options for social media she is interested in, and the social media she spends most time on are different to stakeholder 1’s, this doesn’t narrow down the requirements for which applications my solution should be able to interact with and collect text from.

She is interested in almost all areas of analysis: sentiment, strength of emotion, emotion, subjectivity and common theme analysis, and she, like the first stakeholder, likes all the methods of data representation. The response to question 8 is the only significant response as her other answers mostly match the first stakeholder’s answers; it means my solution should be able to display an analysis for all of these metrics. She also would like examples of text by subjectivity to be returned as an output of the system.

**Stakeholder 3**

1. **How interesting do you find it to read other people’s opinions online? Choose one option.**

Very interesting,

Quite interesting

I don’t read other’s opinions online

Uninteresting to the point of being annoying

1. **How often do you read other people’s opinions and sentiments? Choose one option.**

Everyday, for over an hour

Everyday, for less than an hour

A few times a week

Only occasionally; not habitually

1. **Which social media do you read opinionated/sentiment-driven comments/texts on the most? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others

1. **Would you prefer automatic sentiment analysis or to read and evaluate text yourself? Choose one option.**

Automatic only

I would find automatic useful/fun but I have the time to evaluate text myself

I don’t find significant benefits in automatic analysis compared to manual

1. **Which social media would you find it helpful to conduct sentiment analysis on? Choose one or more.**

Twitter

Instagram

Tiktok

Youtube

Facebook

Others

1. **Which topics/people/products/hashtags would you be interested in for sentiment analysis? Write as much as you wish.**

#mycompanyproducts

1. **Would you be willing to collect text related to your chosen topic of interest yourself and upload it for sentiment analysis or would you prefer text to be collected automatically by the final product? Choose one option.**

I can collect text myself

I can collect text myself, but I would prefer automatic collection as an option too

I wouldn’t want to collect text myself; I would prefer only automatic collection

1. **In which ways would you like your data to be analysed? Choose one or more.**

Sentiment present in the text; positive/neutral/negative

Magnitude (strength) of feeling

Emotions present in the text

Objectivity/subjectivity of the text

Common words/phrases in the text

Topics in the text

1. **How would you like the analysis to be presented to you? Choose one or more options or write others.**

Graphs (pie chart, bar graphs, etc.)

Word clouds

Colour scales

Percentages/numerical values

1. **Would you like my program to display examples of text which contain a particular emotion/sentiment/topic/keyword/subjectivity/strength of emotion?**

Yes (please indicate emotion/sentiment/topic/keyword/subjectivity/strength of emotion

No, I am fine with just a general analysis

Only for stakeholders in the second category (business users):

1. **Do you think social media sentiment analysis will benefit your business? Choose one.**

Yes, very much

Yes, it may be of some benefit

I don’t know

No, it wouldn’t make a difference

1. **Would you find sentiment/emotion analysis on reviews or answers to surveys useful? Choose one.**

Yes, very much

Yes, it may be of some benefit

I don’t know

It wouldn’t make a difference

1. **On which platforms exist the texts you wish to analyse the sentiment of? Choose one or more.**

Twitter

Facebook

Instagram

Review websites

Survey applications

Emails

Other (please name)

**Analysis of stakeholder 3’s responses**

Stakeholder 3’s responses to the general questions are in line with what the other two have answered; she is very interested in automatic sentiment analysis on a variety of different social media and would like an analysis of almost all the metrics offered.

Reading this final response to Question 8 confirms, however, that none of the stakeholders are interested in topic identification, perhaps because it is very similar to keyword identification.

While she believes sentiment analysis would benefit her company, she is not keen on analysis on review websites or survey responses, which eliminates the need for me to interact with non-social-media websites or provide the option for users to upload their own data.

**General analysis**

It seems that whilst there is no need for my product to allow users to upload their own data, my solution may need to interact with a variety of different social media websites which the stakeholders have indicated interest in. This may be difficult for me due to the limitations imposed by development time. Therefore, I have decided to interview my stakeholders to see if analysis of a smaller range of social media (one or two) would meet their requirements. Furthermore I must follow up on Question 8 with Stakeholder 1 to find out which emotions she would like the system to be able to recognise.

**Interview with Stakeholders**

* **Which emotions would you like the classifier to be able to recognise?**

Stakeholder 1: “Anger, sadness, happiness, fear, excitement. That’s all I can think of but I think it gives a good overview of whatever topic you might search up.”

* **How do you feel about the product analysing text from only one social media site? Which social media site would this be?**

Stakeholder 1: “It would be preferable if it did more than one. But I’m alright with that. If there’s only one I would like it to be Twitter.”

Stakeholder 2: “I think that is great! Besides I only use Youtube. [*interviewer: “But the previous stakeholder is keen on Twitter.”*] Well, that doesn’t really change my mind…

Stakeholder 3: “Twitter is fine with me. I would like it if you could also do sentiment analysis on pictures. [*interviewer: “Sorry, that can’t be done…”*] OK, that’s fine I guess.

**Analysis of Interview with Stakeholders**

Stakeholder 1’s response to the first question has given me a minimum list of emotions to classify.

The answers to the other questions seem to indicate that I will need to interact with more than one social media to meet the user’s requirements. To keep my task from becoming too complex whilst still meeting user requirements, I will conduct sentiment analysis on text from Twitter posts, Facebook posts and Youtube comments. These three platforms will give a good number and range of text examples. They are also less likely to contain an extreme amount of slang and modern internet colloquialisms, making them easier to analyse.

**REQUIREMENTS**

The following table details the requirements of my sentiment analysis system.

*Stakeholder requirements*

|  |  |
| --- | --- |
| Feature | Explanation |
| Simple main window containing: | |
| Introduction and instructions on how to use system | These will likely detail the format the user should use when entering their search query into the search bar and what to expect as an output from the analysis. This will give unfamiliar users a clear understanding of how to use the system. |
| Search bar in a prominent place (middle, top) of the window | This will allow users to input the topic on which they would like text to be extracted and analysed. |
| Checkbox list of analyses the system should return | The checklist will allow the user to choose which analyses the system should perform; users can choose as many as they like from sentiment, magnitude, emotion, keyword and subjectivity. |
| Checkbox list of social media platforms the text should be extracted from | The checklist will let users choose as many types of text as they like from the following options: Twitter posts, Facebook posts and Youtube comments. This will let the user personalise their experience and only see an analysis of text they are interested in. |
| Button labelled ‘RUN ANALYSIS’ | On clicking this button, with the condition that the search query is valid, and the checklists are completed with at least one box ticked in each, the analysis will run and be displayed to the user. |
| If there are errors in the users inputs, simple text pops on screen detailing: | |
| Explanation of the error | This explanation will help users correct their mistake and will contain an instruction: “[Fix error and] retry analysis”. The error could be that the search term was not in the correct format, and so an explanation of the correct format will be (re-)provided. It may simply be that no boxes in the checklists were ticked, so an instruction to tick at least one will be provided. |
| Once analysis has been performed, in a different window the following are displayed to the user: | |
| A pie chart titled sentiment breakdown with categories positive, neutral and negative | The pie chart summarises the results of sentiment classification of all the extracted text for the user. Percentages will be displayed beside the labels to give the user numerical feedback. |
| A line: “Magnitude of feeling: [number]” | The number will indicate the strength of the sentiment in the text to the user, with numbers close to zero indicating calm, neutral text and higher numbers indicating strong emotions. |
| A bar chart with categories: happy, sad, angry, surprise, fear, disgust | The bars themselves will be labelled with percentages indicating the proportion of the text examples which contained that emotion. The axis will be labelled. |
| A line: “Subjectivity: [number]” | The number indicates to the user how subjective (opinionated) the analysed text was, with higher numbers corresponding to more opinionated text. |
| A word cloud | The word cloud is comprised of the 5-15 most common keywords found in the text, with the size of each keyword corresponding to its frequency of appearance. This gives the user an understanding of the main points people talk about related to their topic. |
| Beneath each analysis (line of writing or visual explanation), there is an icon labelled “What does this mean?” | On clicking this icon, the user will receive a brief explanation as a popup of what the classifier has analysed the text for, and if necessary, how to interpret the results. For example, under the line about subjectivity, an explanation of what the number means and a qualitative tag for each range of numbers (eg: 0.3-0.7) will be provided. |
| Below the sentiment pie chart, there will be three buttons labelled: “See positive text”, “See neutral text”, and “See negative text”. Similarly, there will be buttons appropriately labelled by emotion, subjectivity and magnitude of feeling. | On clicking these buttons a single example of text containing the corresponding sentiment which the machine has classified on that topic will be displayed as a popup. Reclicking the button displays a different example. This allows the user to see which comments/posts the classifier recognises as ‘happy’ or ‘subjective’, for example. |
| Each keyword in the keyword word cloud will be a button. | Clicking a keyword will display a popup to the user giving an example of extracted text which contained that keyword to the user. |
| Visually pleasing, simple and intuitive design | The screen will be divided into five or so clearly distinct sections which each contain a result of some classification. Colour coding will make obvious positive and negative sentiment (green and red). The design will be minimalistic so as to not overwhelm the user and make the system user to use. Each popup has a clear “x” button in the corner to close it. |
| A button labelled: “Back to search” | Clicking this button will take the user back to the main window, where they may once again type a topic into the search box. |

*Limitations of solution*

Due to the additional complexity and time required to interact with many different websites and training classifiers on datasets of text which is in the style of social media posts/comments other than those found on Youtube, Twitter and Facebook, I cannot expand the scope of the solution to deal with other social media. This is an acceptable limitation for my stakeholders however, as they have indicated the three social media my system will scrape text from to be an acceptable number.

Another limitation which is imposed by time constraints is the fact that my website is for one-time use. Essentially, I am not allowing the option for users to download the outputted results, or otherwise save them to an account. Also, users will not be able to see all posts/comments which the system has scraped from Youtube, Twitter and Facebook, and so will not be able to judge the accuracy of classification themselves. The reason my solution cannot deal with this is the complexity of displaying the csv files (or other type of file) to the user and allowing them to download it. Also, creating an application/website which has user accounts (backend database work) will take too long considering I have to train around 5 separate machine learning algorithms.

These are the main limitations of the solution. However, I believe they are acceptable limitations considering the timeframe I have for this project, and considering the fact that the proposed features meet all user requirements specified in the interviews and surveys. I met with my stakeholders once again to show them the table of features, which they were happy with and expressed no concerns about.

*Hardware and software requirements*

I need a computer with 4GB RAM and a 1GHz processor which is a reasonable amount of memory for a machine learning project so that the algorithms which act on every example in the dataset and the machine learning algorithms run in a reasonable amount of time.

For software I need a good IDE with debugging features and a terminal which can display parts of a csv file (which will be used in training the algorithms). That means Visual Studio Code is the ideal IDE for me, as I have experience with it already. I will be coding in Python 3 and I will need libraries for the user interface, natural language processing, matrix operations, reading from csvs, writing to csvs, interacting with Twitter, and making graphs (which will require tkinter, nltk, numpy, pandas, csv, tweepy and matplotlib respectively).

**SUCCESS CRITERIA**

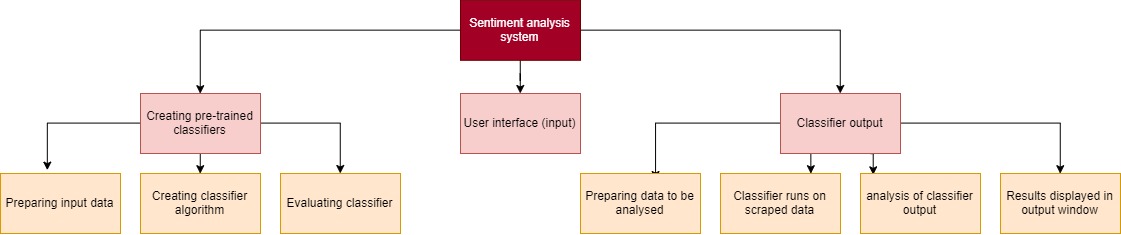
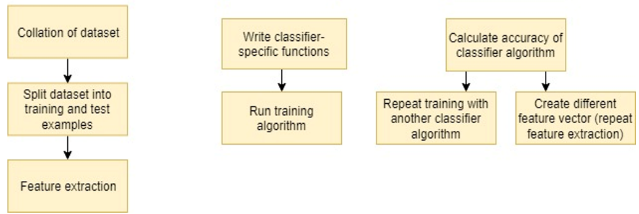
There are two major facets of this solution. One is the side of the system which is displayed to the user. However, for the analysis itself, machine learning algorithms must be created and pre-trained on datasets for use in the system. Success Criteria 1-10 are for the user-side of the solution and Success Criteria 11-12 are for creating the machine learning algorithm which powers the system.

|  |  |  |
| --- | --- | --- |
| **Number** | **Criterion** | **Justification** |
| 1 | Main window has instructions on usage: format in which search query must be inputted, minimum number of boxes to tick in checklist. | Users must understand how to use the system. |
| 2 | Users can input one or more search terms (eg: topics, names, hashtags) into the search box for which they would like related text to be extracted. | This allows the user to conduct sentiment analysis on the topics they are interested in. |
| 3 | Users choose one or more platforms from Youtube, Facebook and Twitter from which they would like text to be collected. | Users should be able to personalise their use of the system. |
| 4 | Users can choose to conduct one or more of the following: sentiment, emotion, magnitude of feeling, subjectivity and keyword analysis. | Users can get an in-depth understanding of the sentiment related to their topics of interest. |
| 5 | Data items (posts or comments) are extracted from the social medium/a users specified and related to the topic users specified and saved as records. | These will become the data items which the machine learning algorithm will run on. |
| 6 | The classifier algorithms corresponding to the ones users picked are run on the text. | The outputted values are converted to relevant averages/required calculations are performed on the large number of classifier-assigned labels. |
| 7 | A visual/textual representation of the output of the classifier is displayed to the user.   1. Sentiment pie chart 2. Emotions bar chart 3. Strength of sentiment metric (a single number) 4. Subjectivity metric (a single number) 5. Word cloud containing 5-15 most common keywords | The user can gain a visually appealing, intuitive understanding of the sentiment, etc., related to their topics of interest. |
| 8 | User has the option to have a post/comment labelled in any way by the algorithm (eg: ‘happy’, ‘objective’, ‘neutral’) which was extracted from social media in that run of the system to be displayed to them. | This lets the user judge the accuracy of the system and explore which posts/comments were classified as having a particular sentiment/emotion etc. |
| 9 | Users have the option of an explanation shown of how to interpret each graph/quantitative metric which was outputted. | This lets users understand what is meant by each label and what each number actually means. |
| 10 | Users can return to the main window to enter new searches/rerun the analysis as many times as required. | The system is not for one-time use. Users should be able to navigate it naturally. |
| 11 | Collate files with records of training examples (short pieces of text) labelled by:   1. Sentiment 2. Emotion 3. Magnitude of feeling 4. Subjectivity | These files contain the training examples for each of the 4 necessary classifiers. |
| 12 | Train all 4 classifiers on the datasets. | This will result in the pre-trained classifiers which will form the core of the final product. |

**DESIGN**

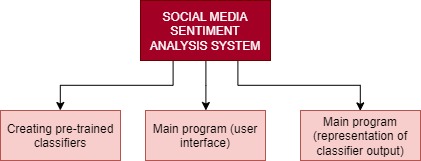
**SYSTEMS DIAGRAMS**

Below is my structure diagram for this project.



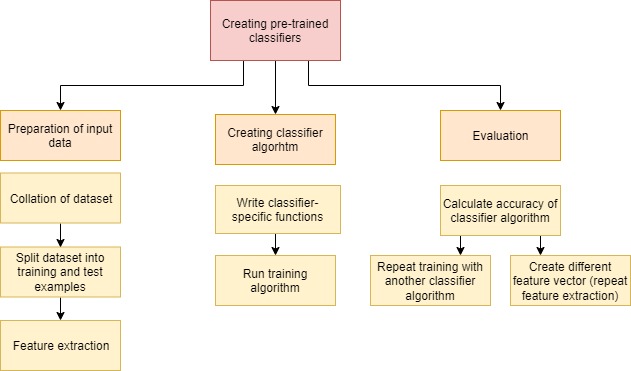
So that all of the detail in these diagrams is visible, I will split the overall systems diagram into separate diagrams. Each diagram will be following by an explanation of why I have decomposed the larger problem that way.

*Overall diagram*

**

|  |  |  |
| --- | --- | --- |
| **Decomposed section** | **Why I decomposed it into that section** | **Explanation of section** |
| Creating pre-trained classifiers | The classifiers are the vital part of the sentiment analysis system. The class which contains the classifier algorithm and its methods must be a separate module to the main program so that I can test it separately and make sure it is ready for deployment in the main program. Since this class will be used throughout the main program, I have chosen this step to be a decomposed section of the solution. | This step will in fact involve creating 4 different classifiers (for sentiment, emotion, subjectivity and magnitude of feeling). This doesn’t necessarily mean that 4 separate datasets are required, but text examples labelled by sentiment, emotion, etc. will be needed. The text must be converted into a mathematical form. Furthermore, it is not certain which algorithm will be used for the classification; several may have to be tested and evaluated to see which has the highest accuracy. |
| Main program (user-interface) | The user interface is like a skeleton which uses the outputs of the other two sections for functionality. However, none of the code for the interface itself requires methods or data within the other two steps (it only requires their outputs), so it can be a separately decomposed section due to its independence from the functions of the other steps. By developing the interface separately, I can make sure it is simple and intuitive to use as the stakeholders require. | The user interface involves taking search terms as the main input of the system. The output will be a number of graphs summarising the algorithms findings on sentiment. |
| Main program (representation of classifier output) | This part of the program is a distinct stage as none of the methods within will be used in the other stages. This module will make use of the classifiers trained in the first stage. It is necessary that the output of this stage is correct before being displayed to the user via the user interface, so I should be able to build and test this module separately. | Data will be scraped from the user’s preferred social media website and saved in a file. The data is then converted into mathematical form (vectors) in the same way as the training data was, and the classifier algorithms corresponding to the user’s desired areas for analysis are deployed on these vectors. The output of the classifiers must be summarised by taking percentages, or calculating means, etc. |

*Creating pre-trained classifiers*

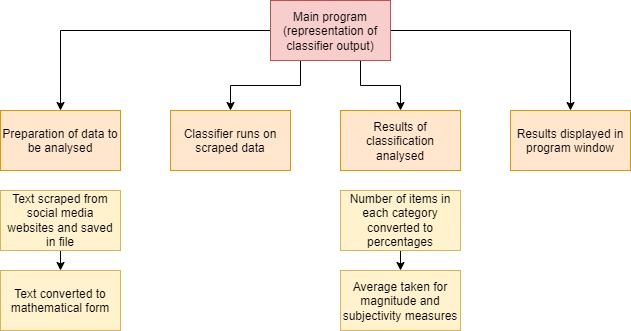


|  |  |  |
| --- | --- | --- |
| **Decomposed section** | **Why I decomposed it into that section** | **Explanation of section** |
| Preparation of input data | Feature extraction (which is part of this step) must be tested thoroughly to make sure the mathematical forms produced are a good representation of the text examples before the classification stage can make use of these forms. Therefore, I have decomposed this into its own stage to ensure testing can occur separately before the output is used in the next stages. | I will find datasets from the internet which contain text examples labelled by sentiment, emotion, subjectivity and strength. I will combine these into 4 datasets, which will be split in a roughly 4:1 ratio of training and test examples. The text must be converted into mathematical form; this could be done for example using a vector which represents word frequency. |
| Creating classifier algorithm | To allow greater flexibility in engineering the system, the classifier model will be its own class. This is so that when testing I can see whether inaccuracies in the model’s predictions are due to the model itself or the feature extraction method (which is a different decomposed part). | The main two algorithms are the cost calculation algorithm (which differs based on the model used) and the learning algorithm (which fits the model to the training data). |
| Evaluation | Evaluation is its own stage as if I were to include the functions for calculating F1 score (which is an evaluation metric) for example in the previous class (classifier algorithm) it would get too convoluted and have too many purposes. I want to be able to conduct testing on these evaluation functions work separately on a far smaller amount of data to that being used in the other stages, so it has to be its own stage. If there are errors in the output to the above decomposed sections, I would like to be sure it is due to issues in the ML model or feature extraction, not with the evaluation function, hence, it is separated from the other functions. | Various metrics such as precision and recall may be used to evaluate a machine learning model’s accuracy. If the model I initially used wasn’t accurate enough, I may use a different model (by changing constants in the model, or otherwise choosing a different ML algorithm entirely) or change the way in which I converted my text to mathematical form. |

*Main program (user interface)*

I have not decomposed the user interface part of the main program any further as all its elements may be developed simultaneously and tested together. The problem doesn’t need to be broken down any further, as this section is essentially just about building the layout and design of the windows in my program. There is no significant coding element to it.

*Main program (representation of classifier output)*

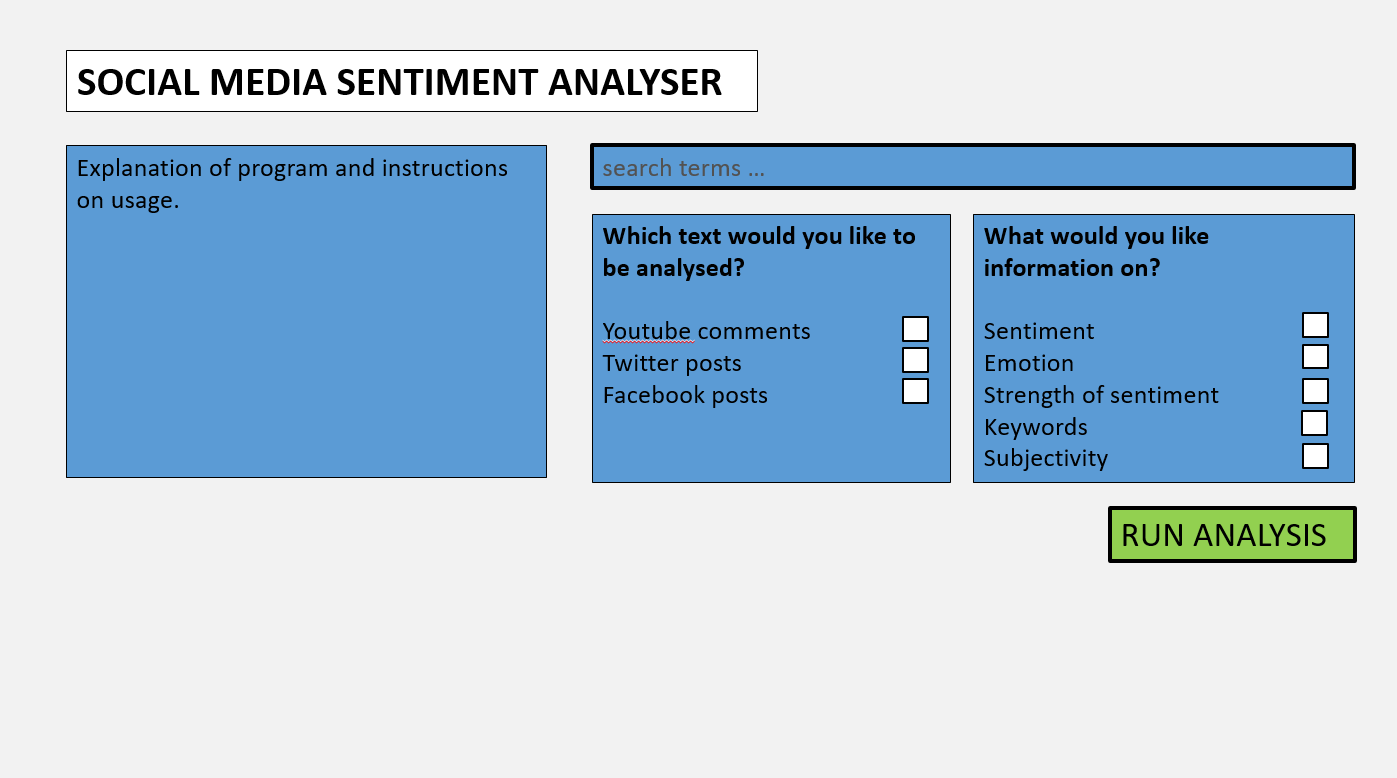


|  |  |  |
| --- | --- | --- |
| **Decomposed section** | **Why I decomposed it into that section** | **Explanation of section** |
| Preparation of data to be analysed | I have decomposed this into its own stage because I would like to be able to develop and test the code I write for the API to interact with the websites independently to the remaining program. | Text examples (Tweets/Facebook posts/Youtube comments) are scraped from the relevant websites using the API to collect only those pieces of text which match the search terms the user has inputted. The text is saved in a file and then each piece of text is converted into vector form using the same feature extraction method as was used to train the algorithm. |
| Classifier runs on scraped data | It makes sense for this to be its own stage as the inner workings of the classifier algorithm are separate to the rest of the program and no functions/data which are used in the classification process will be required by the rest of the program, other than the data outputted. This makes this stage a perfect ‘black-box’. | The relevant classifiers (chosen by the user) are run on the data. The results are saved in another file. |
| Results of classification analysed | The output of this module may be required several times throughout the running of the program (when the user runs analysis several times) so it is good to develop these functions in their own module to prevent duplication of code. Also, I would like to test the averaging and counting processes separately to the remaining program as it makes debugging easier. | For sentiment analysis, the classifier will class each example as positive, negative or neutral, so the number of each class will be counted and converted to a percentage. For emotion analysis, a similar process will take place (this time with emotion as the output variable). For strength and subjectivity, the classifier will output a floating point number so a suitable average will be taken. |
| Results displayed in program window | This is once again a reusable module in the program which may be developed separately. I would also like to test the process of creating the pie charts etc. separately from the remaining program and on simpler data to make troubleshooting an easier task. | The class and percentage for sentiment and emotion can be converted into a pie chart and bar chart, respectively, using a library. The subjectivity and magnitude measures will be presented with a small explanation. A word cloud of key terms will be displayed. |

**SCREEN DESIGNS**

*Main program window*

This is the page users will see when they first open the program. It is from here that all user input into the program is obtained. The colour scheme may be different in the final project.



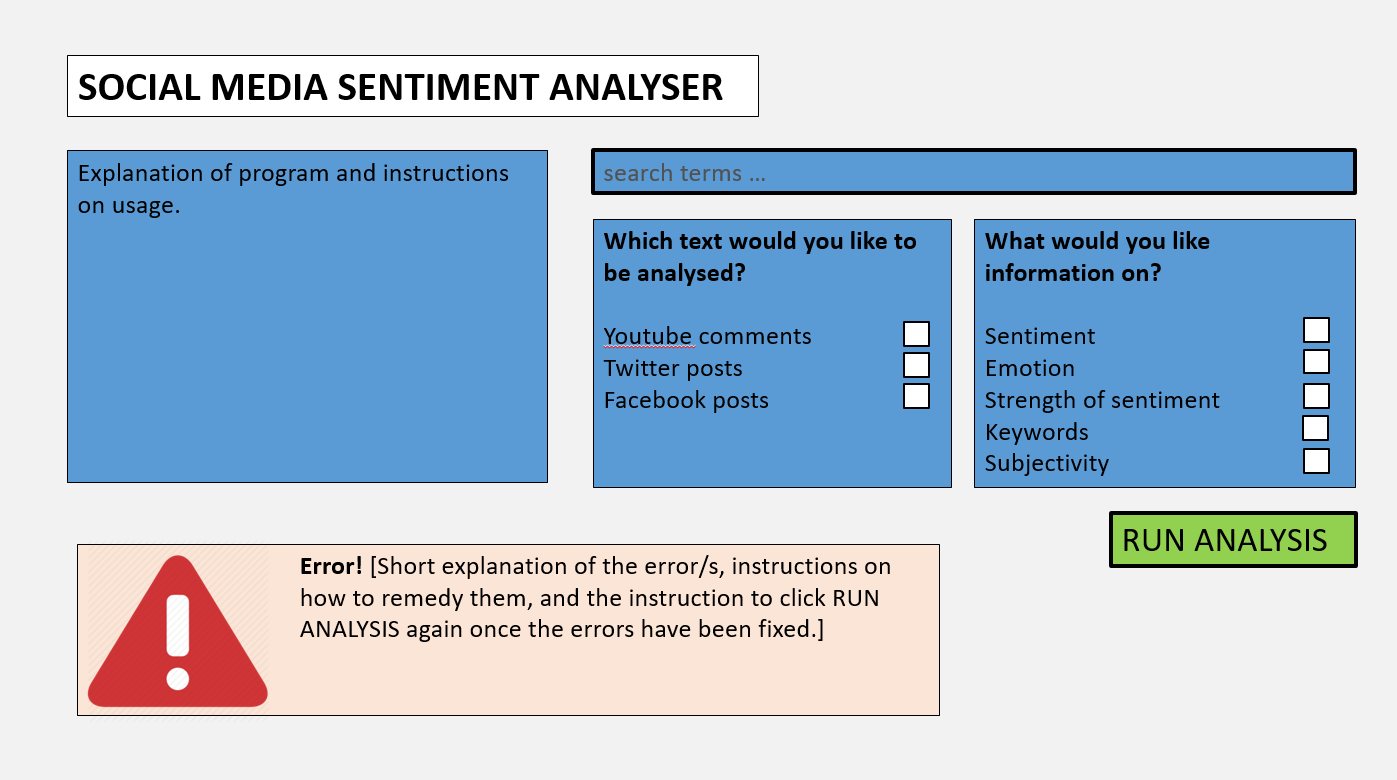
The successful development of this screen design as well as the functionalities behind it will lead to me meeting success criteria 1, 2, 3 and 4.

Usability features:

* The layout is clear to users as there are only 6 major divisions of the screen, of which the user only has to interact with 4, which are the search box, two checklists and the run button. This ensures the design is easy to navigate and use as everything that requires user input is on one screen in close proximity. It also means the page has a minimalist, uncluttered design which makes for a better user experience.
* The questions of the two checklists are written in simple language which abstracts away the underlying complexities of the machine learning process. For example, ‘classifier machine’ isn’t mentioned in the second question. This is to meet requirements for users who don’t have a technical background and are just using the program out of personal interest or amusement.
* The explanation of the program allows users to understand what exactly the program aims to achieve, which is helpful for first-time users. It also allows them to understand what ‘sentiment’ and ‘subjectivity’ mean in this context, as these terms carry a very specific meaning when it comes to the classifier giving labels.
* The instructions on usage will ensure the user inputs search terms in the correct format and knows to tick one or more boxes for each of the questions. This means the user clearly understands how to use the program; entering search terms could be difficult without prior knowledge of what the system will and won’t accept, especially if the user wants to search for something niche.
* The RUN ANALYSIS button is located in a prominent place, has larger text than the rest of the screen, is capitalised, and the colour is different to the rest of the divisions. This is an intuitive design aiming to help users understand that clicking the button is the final input required.

*Main program window (input error version)*

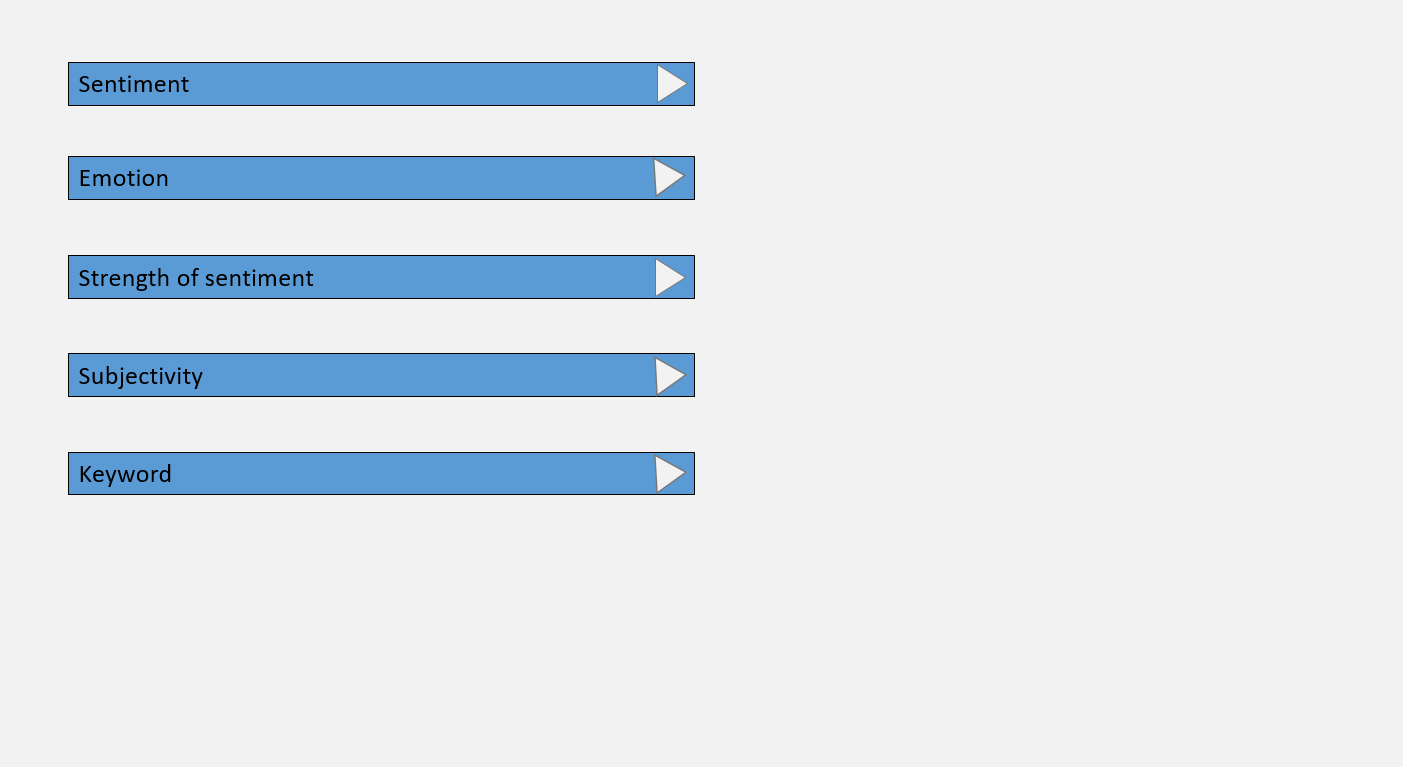
In the case that the user clicks RUN ANALYSIS, but not all user input have been taken/are in the wrong format (concretely, if the user enters search terms in the wrong format or have not ticked at least one box in each of the questions), popups will come up on screen as shown below.



Usability features:

* The error popup is clear due to its red colour and the error sign. This reinforces the intuitive design, helping users understand that the program has met with an error.
* The error popup is on the same screen as the main window, which helps users see both the error and the rest of the screen at the same time, rather than unhelpfully taking them to another screen and making them use the back button to return to the main window. This provides users with an efficient experience of the program.
* The error text will give precise instructions on how to remedy the error, which minimises chances of users getting confused and struggling to enter valid inputs.

*Results window*



Back to search

B

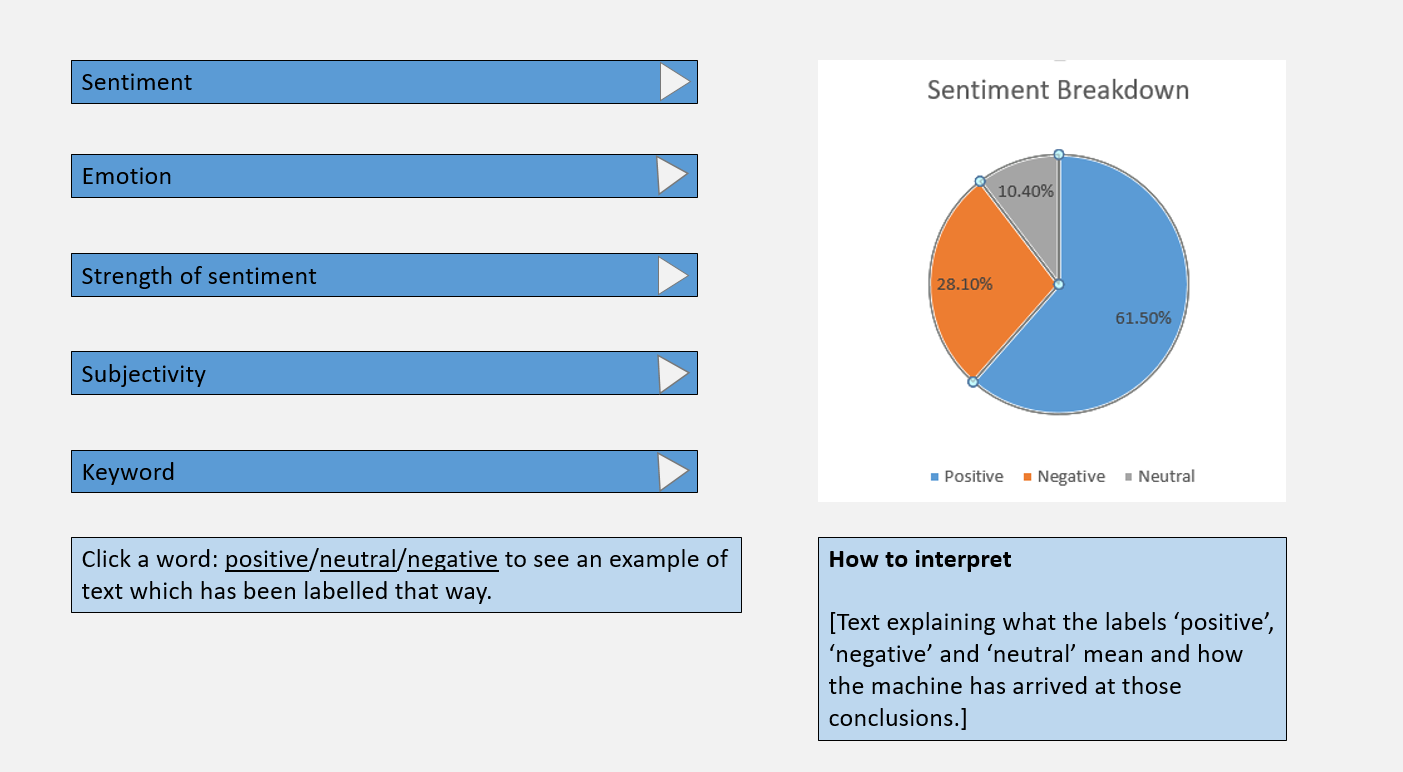
Once the user clicks RUN ANALYSIS the program switches to the following window which has the following bars with labels and arrows. Clicking the arrow will display the results of the classification process for that particular label. Since the user has chosen which classification results they are interested in using the checklist in the previous window, only those results will be displayed. The above screenshot shows the results window if the user chose all the areas of analysis.

Usability features:

* Rather than displaying all the charts/information on screen at once, which would make the screen cluttered and confusing to understand, only the subtitles, eg: ‘Sentiment’, are displayed initially. This allows me to make a clean and minimalist design, meeting the user requirements for ease of use.
* The clear back button lets users easily navigate between this window and the main window.

*Results window (sentiment analysis)*

Clicking the triangle arrow next to sentiment leads to the following screen. It is essentially a popup in the same window.



Back to search

B

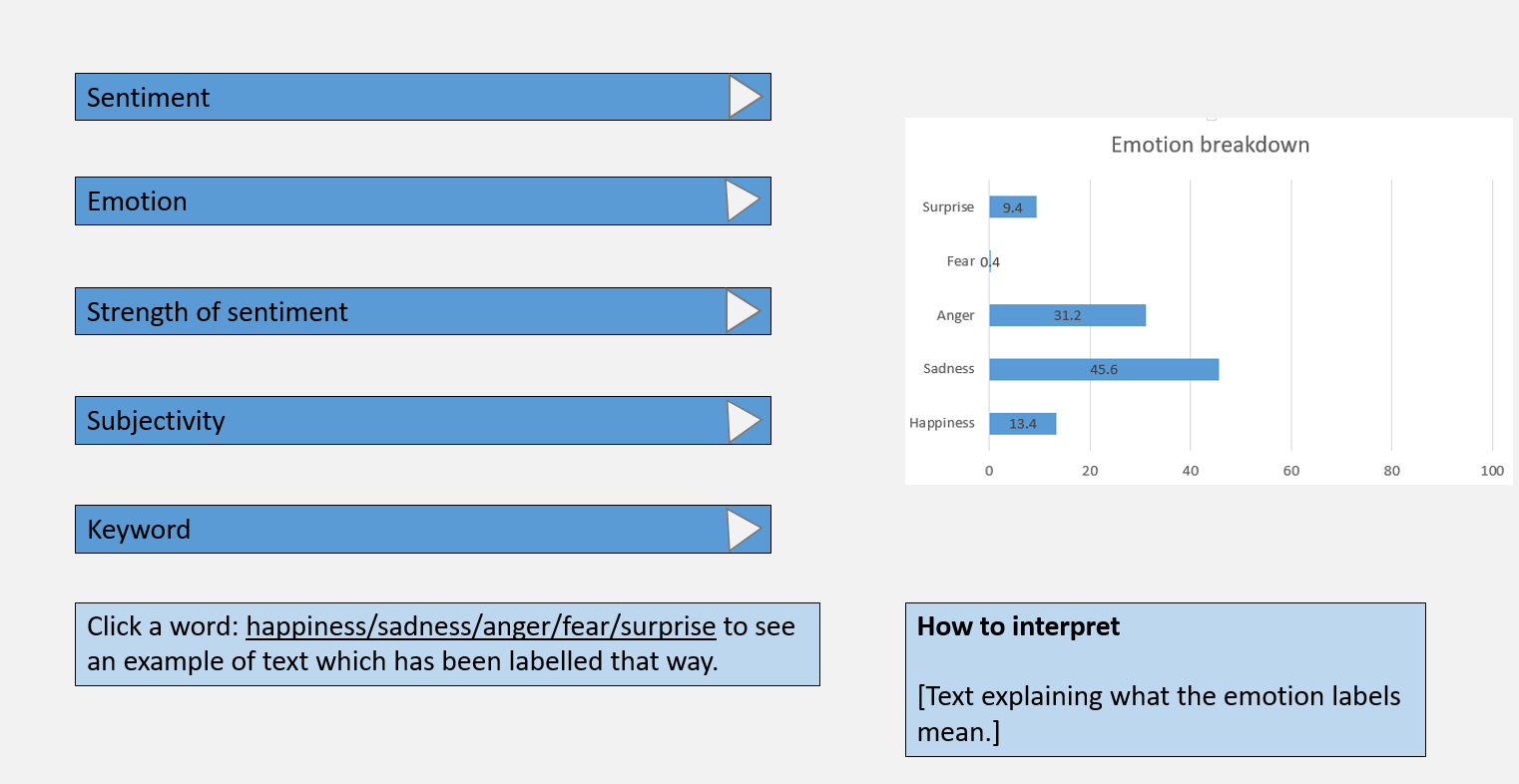
Usability features:

* The pie chart is an intuitive visual representation of the classifier’sdata. It is much easier to understand the overall sentiment by viewing the chart compared to by reading the massive file of individual labels the machine has assigned. However, there is still sufficient detail provided in the analysis as the percentages are displayed to 1 d.p. on the chart, allowing users to get a thorough analysis.
* The How to interpret section helps users understand what the labels mean and the process behind the classification which ensures users don’t get confused.

The successful development of this screen and the functionalities behind it will complete success criteria 7a, 9 and 10.

*Results window (emotion analysis)*

If the user clicks the arrow next to the ‘Emotion’ bar the rest of the screen besides the five horizontal bars and back arrow will be replaced with the following:



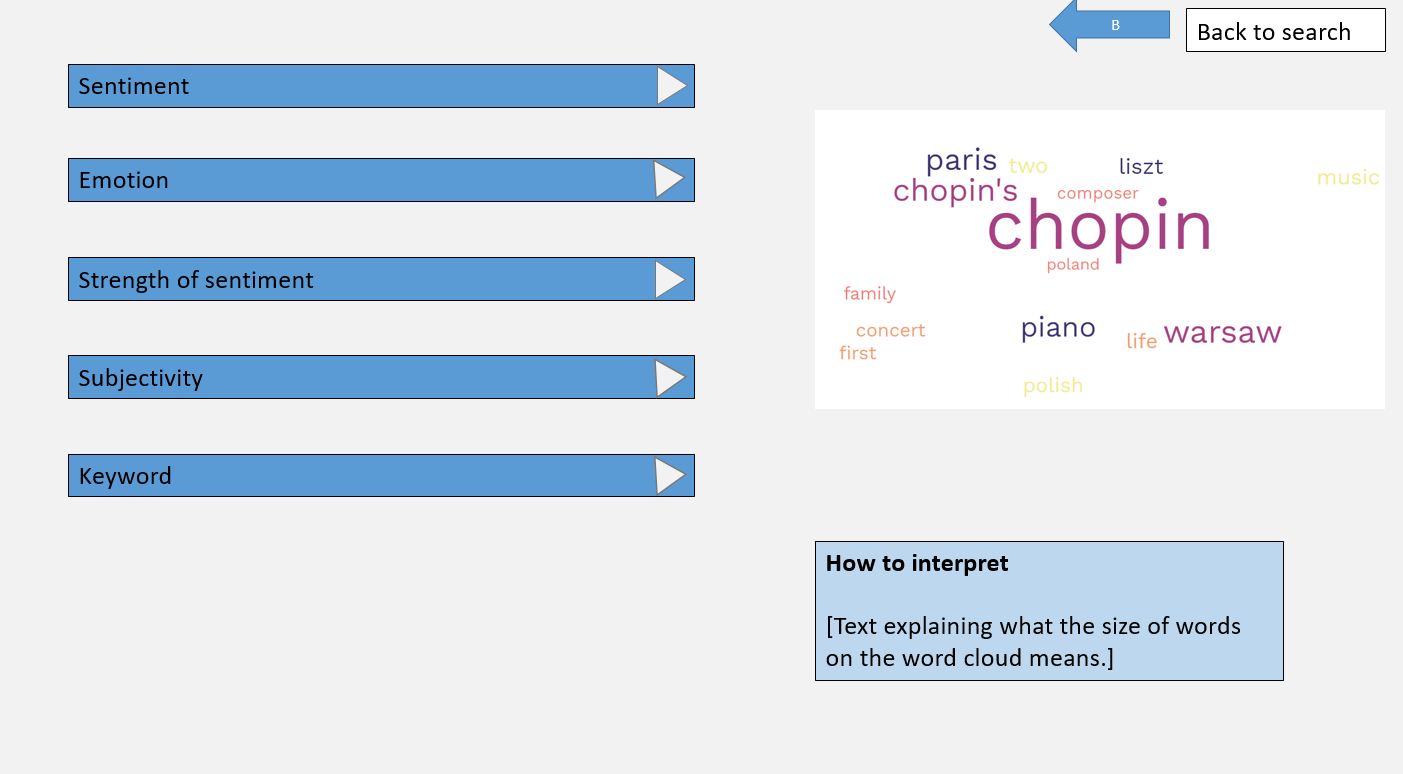
B

Back to search

This screen corresponds to criteria 7b, 9 and 10.

*Results window (Keyword)*

If the user clicks the arrow next to the ‘Keyword’ bar the rest of the screen besides the five horizontal bars and back arrow will be replaced with the following:



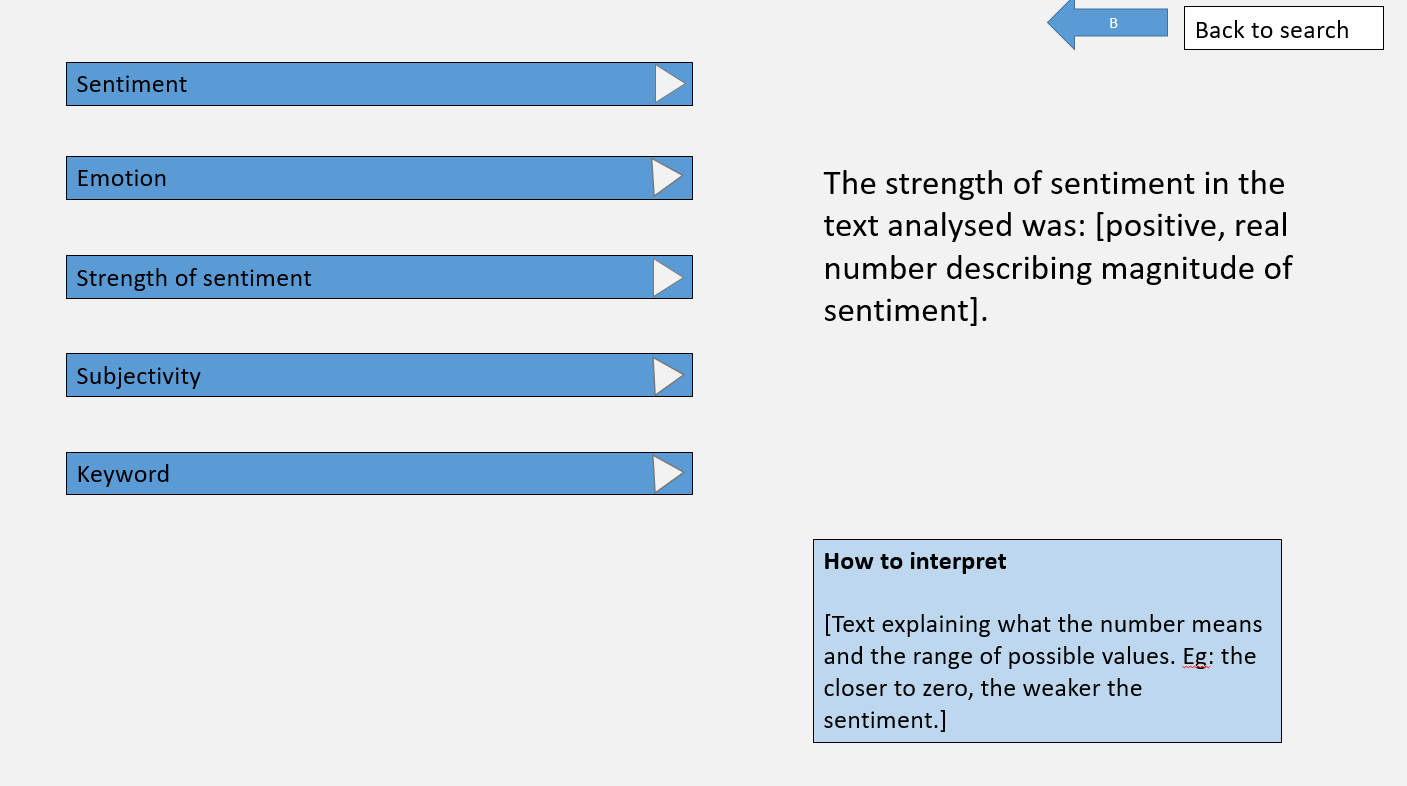
B

Back to search

This screen corresponds to criteria 7e, 9 and 10.

*Results window (strength of sentiment)*

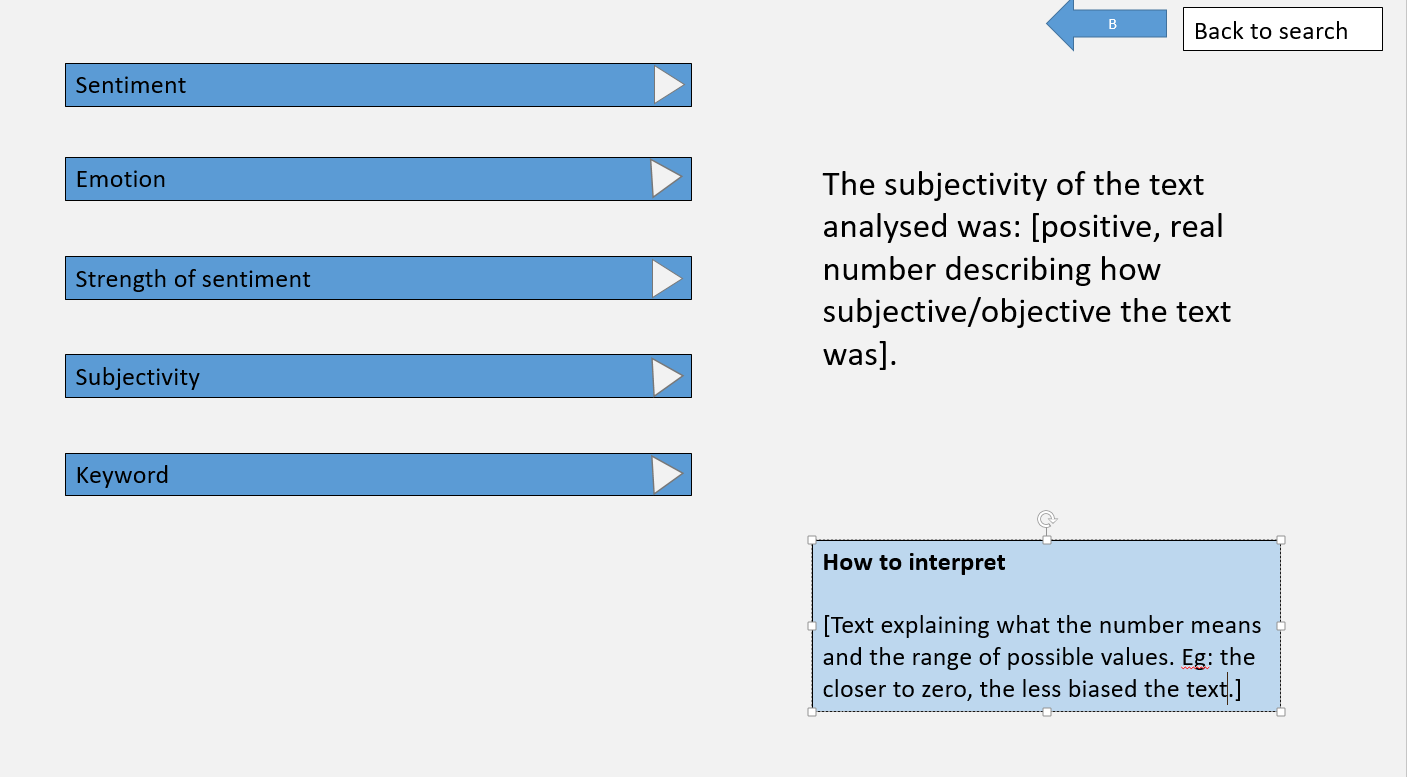
If the user clicks the arrow next to the ‘strength of sentiment’ bar the rest of the screen besides the five horizontal bars and back arrow will be replaced with the following:



This screen corresponds to criteria 7c, 9 and 10.

*Results window (subjectivity)*

If the user clicks the arrow next to the ‘Subjectivity’ bar the rest of the screen besides the five horizontal bars and back arrow will be replaced with the following:



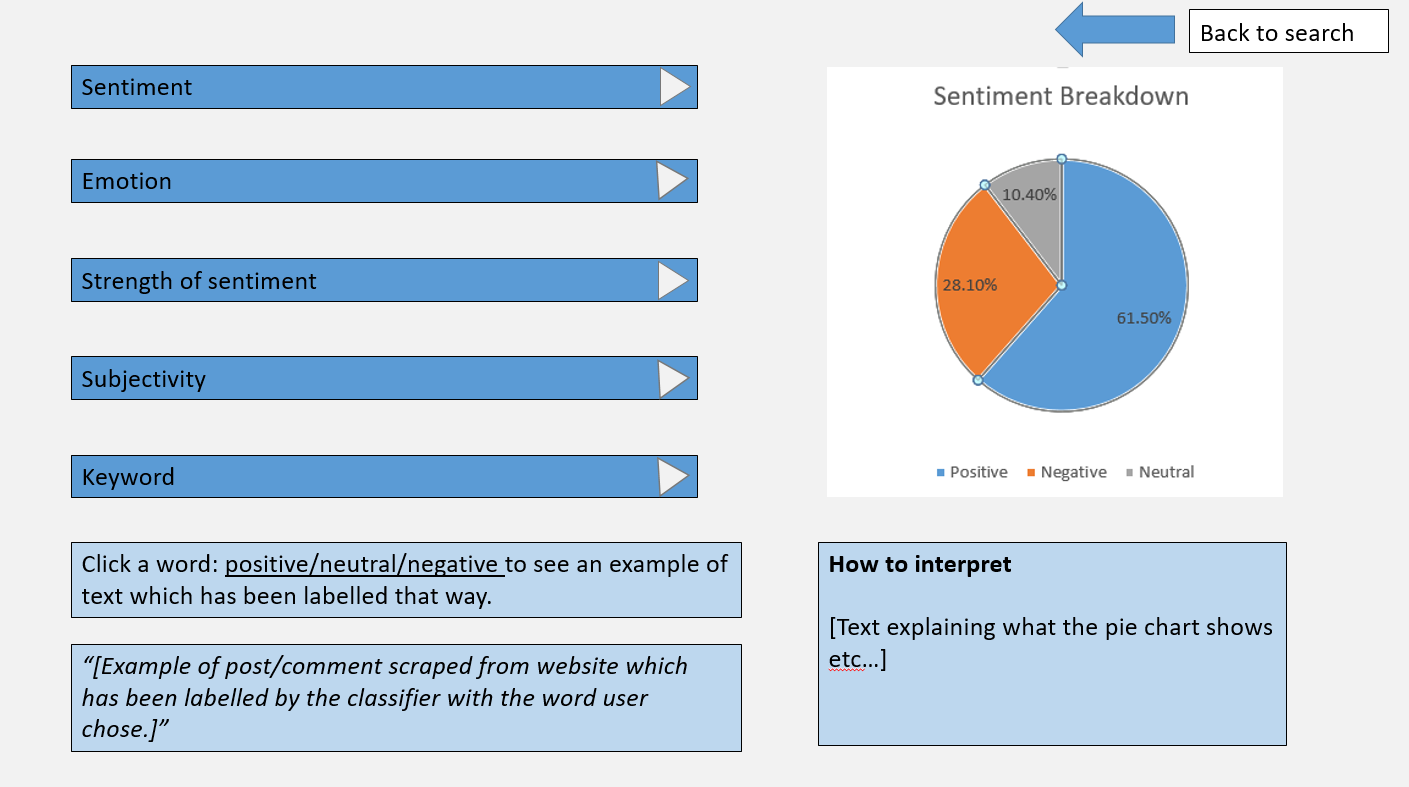
This screen corresponds to criteria 7d, 9 and 10.

The last four screens have the same usability features as the ones explained for the ‘sentiment’ analysis screen. Some additional usability features include:

* The text is large to ensure good accessibility which allows any user to read the text.
* The more difficult metrics to interpret (the quantitative ones) are explained in the How to interpret section.

*Results window (example text)*

For sentiment and emotion, there is an option near the bottom of the screen for the user to click a word and see text which has been labelled with that word (which is a sentiment or emotion descriptor). If the user clicks a word the popup comes up on the same screen as shown below. This completes success criterion 9.



Usability features:

* The text returned to the user will only be one example of a post/comment the machine has scraped. This keeps the user interface uncluttered. Also, the user will have an easier chance digesting just one Tweet, for example, rather than lots of them. The position of the text on the page is so that it doesn’t overlap with any other sections, making it easy to interact with the rest of the page even when the example text is on screen.

**ALGORITHMS**

Here I will outline the key algorithms necessary for my project, organised by section of project identified by the structure diagram earlier. The following algorithms will form a complete solution to the problem as they cover every module displayed in the structure diagram. The output/s of each module are inputs to another module so the solution is connected.

I will explain in a basic manner how the following algorithms will describe the solution fully at the end of the algorithms section.

**Creating pre-trained classifiers**

*Collation of dataset*

There will be some trivial algorithms here to split the dataset into training and test examples simply by splitting the dataset using indexing. Other trivial algorithms involve those to narrow the dataset down into just text data and sentiment data, by removing any other columns there may be.

Cleaning the text data will require many similar algorithms to:

* Turn the text lowercase
* Remove punctuation
* Remove stopwords
* Remove URLs.
* Etc.

The below is a demonstration of how one of these tasks (removing stopwords) could be accomplished.

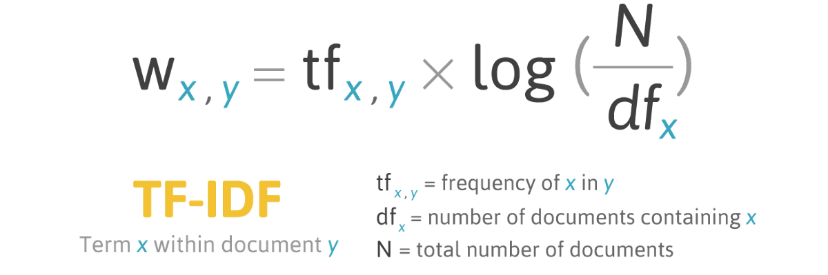
|  |  |
| --- | --- |
| pseudocode | justification |
|  | This is a function so that I can test it separately to the rest of the code and call it from any point in the code I need.  A list of stopwords is declared within the function as it isn’t needed in the rest of the program.  The function iterates through each piece of text in the dataset, and through each word in the text, checking if that word is in the list of stopwords. As long as it isn’t, the word is appended to newtextlist, which is joined into a new piece of text where the stopwords have all been filtered out.after iterating through each word in the original text. Finally the newtext variable replaces the original text.  The reason I chose the approach of appending words to a new list then joining the list into a sentence is because I am very familiar with Python’s join and split functions. This is preferable to learning new syntax for a library like regex which could accomplish the same task, but in a less intuitive way.  The reason I am not showing other cleaning algorithms is because they all involve the same iterative steps as this one; the only difference is the feature of text to be removed (in this case a list of stopwords, but in other functions it could be a list of punctuation/numbers) |

|  |  |
| --- | --- |
| Pseudocode | Justification |
|  | The purpose of this function is to remove any consecutive repeated characters, so ‘I feel so happy!!’ would become ‘I fel so hapy!’ The same iteration through the dataset as the last function is involved in this one, but this time I am iterating through each character in the text, not each word.  The IF statement this time asks for each character in the list, is it equal to the next character, and if not, it can be appended to the new text. Also, the last character must be appended to the text as it is impossible for it to be a repeating character as the previous character will have been filtered out anyway if that were the case.  Once again, I have chosen a logical, non-library approach as it is often difficult to learn library syntax and it may not do the specific task required here. |

*Feature extraction*

This requires me to turn the text into mathematical data which a machine can analyse. The approach I will use here is a tf-idf vectoriser. This turns the entire corpus (dataset) into a matrix where the columns each represent a single word. There will be a column for every single word which appears in the dataset. Each row represents a piece of text. The values in the matrix are the tf-idf score of the word considering the context of the piece of text it is in and the document as a whole. Tf-idf score is a product of tf score and idf score, where tf score equals the number of occurrences of that word in the piece of text divided by the number of words in the piece of text and idf score equals the logarithm base 10 of the number of piece of text divided by the number of pieces of text which contain that word.

The td-idf score indicates to the machine learning algorithm how important a specific word is to the process of sentiment analysis. For example, if a word occurs many times in a tweet it will have a high tf score. Furthermore, if this word occurs very few times in all other tweet, it will have a high idf score, as idf score is inversely proportional to number of occurrences. This means the number representing this word in the matrix is high, so it will have a higher weighting in the learning process.



However, there is more to do before the data is in its final mathematical format. Since each document in the corpus (each piece of text in the dataset) will only contain a tiny percentage of all the possible words in the corpus, the majority of the tf-idf matrix is 0s. These consume a lot of memory and it is preferable to convert this rather sparse matrix into a more memory-efficient format. This is done by converting the matrix into CSR format. Its workings can be demonstrated via an example.

Consider the 4x4 matrix [[0,0,0,1],[3,2,0,0],[1,0,0,0],[0,0,0,0]]. CSR format of this matrix is [[1,3,2,1],[3,0,1,0],[0,1,1,2]]. The first vector [1,3,2,1] of this matrix is simply an array of all the non-zero values in the original matrix. The second vector [3,0,1,0] is a representation of the column of each of the values in the first array. For example, the first value from the first array of the CSR matrix is 1, which is at column 3 (using 0-indexing) of the original matrix. The third and final vector of the CSR matrix represents the row of each of the values in the first array. Using the same example, the first non-zero value 1, is in row 0 of the original matrix.

Here are the functions I will use to turn my dataset into a tf-idf matrix:

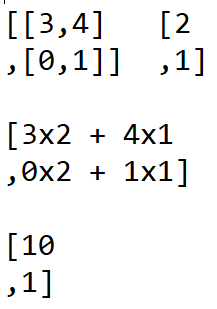
|  |  |
| --- | --- |
| pseudocode | justification |
|  | I have defined the few vectors I will require for all the upcoming functions at the top.  My first task is to make a vector of all the words in the entire dataset, which I have done by iterating through each piece of text, and each word in each piece of text, and appending said word to the list allwords if it isn’t in allwords already. I have chosen this way as it is a clean and efficient approach, which doesn’t require knowledge of casting between lists and sets. Also, the same order of words will be created each time this code is run, unlike the case where the set datatype is used. |
|  | This algorithm will make a matrix whose rows each correspond to a text and columns each correspond to a word in the allwords vector. The values will be the tf scores. It iterates through each line of text, creating a new row for each. Each text is converted into a dictionary where the keys are unique words and the values are their frequency; I am using this approach as the dictionary data structure in Python is a powerful way of manipulating lists and finding word frequencies. The tf score for each word is calculated on the 6th line this score is appended to the correct index in the matrix. |
|  | This algorithm creates a matrix of idf scores for the entire corpus. Similarly to the above algorithm, this one iterates through each text in the corpus and creates a new row for each. It then iterates through each word in the text. For each word, it iterates once again through the corpus, counting how many texts contain that word, incrementing wordtextcount accordingly. On line 10, the idf score is calculated and after the code runs for each text, the row is appended to the matrix.  I have used iteration through the dataset as Python makes it easy to navigate a csv file using the library Pandas. Also, 2d arrays in Python are very good at representing matrices with its [row][column] indexing. |
|  | This final algorithm multiplies the two matrices element-wise to give the tf-idf matrix. First I iterate through each row, then through each column, and each element in the tf matrix is multiplied by the corresponding one in the idf matrix and appended to the correct position in the final matrix.  I used this technique of element-wise multiplication as by the nature of the two original matrices, they have the same dimensions and so are conformable for element-wise multiplication. I could alternatively use the numpy library to execute this algorithm in a single line of code. |
|  | I must convert the tfidf matrix (which contains a massive amount of 0s taking up memory) into a more dense format known as CSR.  I simply iterate though each element of each row checking that it is not equal to 0 before appending the value to the values array, the column to the columns array, etc. Finally, I append the 3 arrays to the final CSR matrix.  I have used this iterative approach as Python makes it easy to iterate through 2d arrays using for loops and indexing in Python also allows me to store the values of the rows and columns. Arrays in the form of lists can easily be appended as rows in other lists. |

*Creating classifier-specific functions*

In machine learning, multiplication of matrices and vectors is always required. Therefore I need an algorithm which can multiply a matrix and vector (provided they are multiplication conformable, which they will be by design). This would be an easy task if my matrix were a regular mxn matrix, but instead I am storing my matrices in CSR format as shown in the latest algorithm above, so I need an algorithm which multiplies a CSR matrix and a vector.

I will now provide an explanation of matrix-vector multiplication in the case that we are multiplying an ‘m’ by ‘n’ matrix with an ‘m’ by 1 vector. The product of this multiplication will be another ‘m’ by 1 vector. The ‘n’th element in the product vector is obtained by summing the product of the ‘k’th element of the ‘l’th row of the matrix with the ‘k’th element of the original vector for k between 1 and n and l between 1 and m.

As an example, consider matrix multiplication of the below 2x2 matrix and 2x1 vector:



The 1st element of the product vector is obtained by multiplying the first element of the first row of the matrix and adding it to the product of the second element of the first row of the matrix.

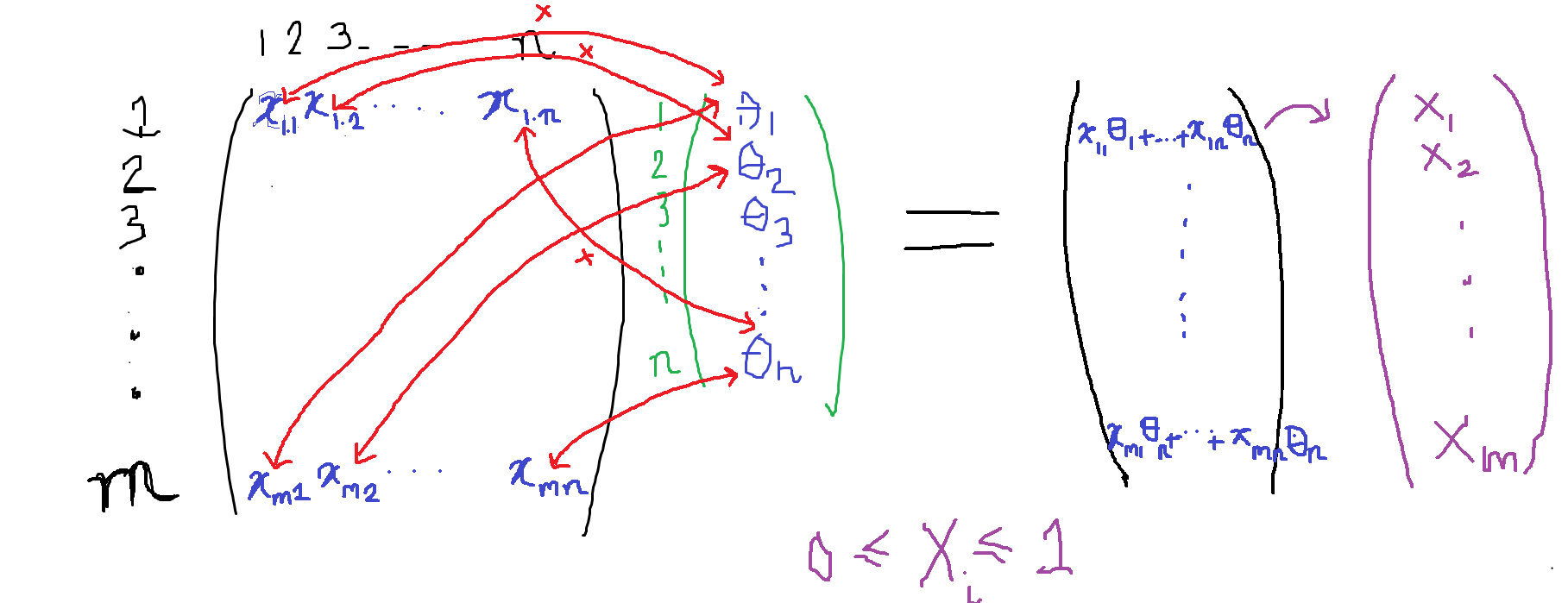
The 2nd element of the product vector is obtained by multiplying the first element of the second row of the matrix and adding it to the product of the second element of the second row of the matrix.

|  |  |
| --- | --- |
| Algorithm | Justification |
|  | This algorithm multiplies a matrix in CSR format with a vector. The first for loop initialises the output vector by filling it with the appropriate number of 0s. The second for loop transposes the CSR matrix (i.e.: reflects the matrix along the diagonal beginning in the first element of the first row). Therefore, since there are 3 rows in any CSR matrix and ‘n’ columns, the CSTRtranspose matrix will contain ‘n’ rows each with 3 columns. The final for loop performs the matrix multiplication. This is done by iterating through the transposed matrix. The number of elements in the transposed matrix equals the number of non-zero values in the entirety of the original matrix. Consider the expression in the line in the for loop. CSRtranspose[count] obtains the 3 element vector at position count in the transposed matrix whose 0th element is the non-zero value, 1st element is the number of the row of that non-zero value is found in the original matrix and 2nd element is the number of the column that non-zero value is found in the original matrix. Therefore, since the product vector’s nth element must be equal to the sum of the products of the elements of the nth row with their corresponding elements in the column vector, CSRtranspose[count][0] (which is the non-0 value) is multiplied by the element at the index of the column the non-0 value is found at (which is at CSRtranspose[count][2]). This number is added to the current value at the index of the product vector which matches the row the non-zero value was found in (which is row CSRtranspose[count][1]).  I have chosen the approach of transposing the original matrix as this makes it far easier to navigate the values in the CSR format matrix and iterate through them to add them to the product vector. I could have accomplished this matrix-vector multiplication using a library like numpy which does accommodate the CSR data structure. However, in numpy the CSR matrix has a slightly different format to mine (where the 2nd array has only unique ‘marker’ elements rather than storing every row) which makes it impossible for this type of multiplication algorithm to work in a small number of lines of code. Therefore, I will use my own custom algorithm. |

*Logistic regression algorithms*

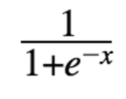
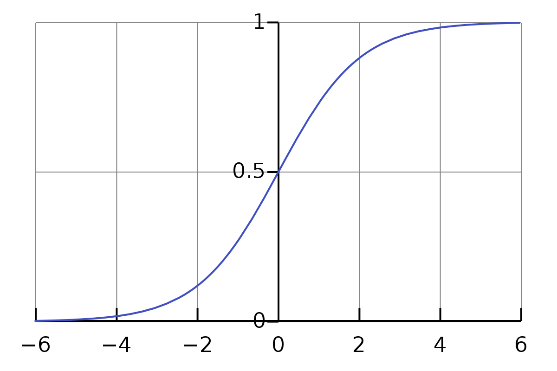
Logistic regression is a statistical model which is used in training binary outputs classifiers. The fundamental algorithms for logistic regression are: the sigmoid function, the cost function and gradient descent. A supplementary algorithm which improves performance of the classifier is standardisation (a form of feature scaling).

To explain the above algorithms, observe the dataset below. Each row represents one piece of text. Each column represents a unique word, and there is a column for each unique word over all pieces of text. The value at a specific row and column is the td-idf score for the word in context to the entire dataset and that specific piece of text. Essentially, the entire dataset has been converted into a massive matrix.



A hypothesis vector whose length is equal to the number of columns (so that there is a parameter for each unique word) can be seen to represent the ‘weighting’ the classfier gives to each word in the dataset. By multiplying the massive matrix by the hypothesis vector, we get as an output another vector whose length is the number of rows in the massive matrix (the number of text examples in the dataset). This vector represents our predictions for the sentiment of each piece of data. However, to convert each number in the outputted vector into a binary prediction (positive or negative), we must use the sigmoid function which maps any real number (any number from negative to positive infinity) onto a number between 0 and 1.

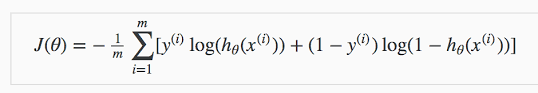
The sigmoid function is shown below.



|  |  |
| --- | --- |
| Algorithm | Justification |
|  | I have used the array data structure, because its equivalent ‘list’ structure in Python is easy to append items to and there are one line expressions which can accomplish this task. Python lists can easily represent the input and output vectors. |

Once the sentiment predictions for all the pieces of text have been made, they must be compared against the actual sentiment (1 or 0 for negative or positive) using the logistic regression cost function, which calculates the price of an incorrect prediction. The cost function is given below, with h(x) = the prediction, m equal to the number of text examples and y equal to the actual sentiment (1 or 0).

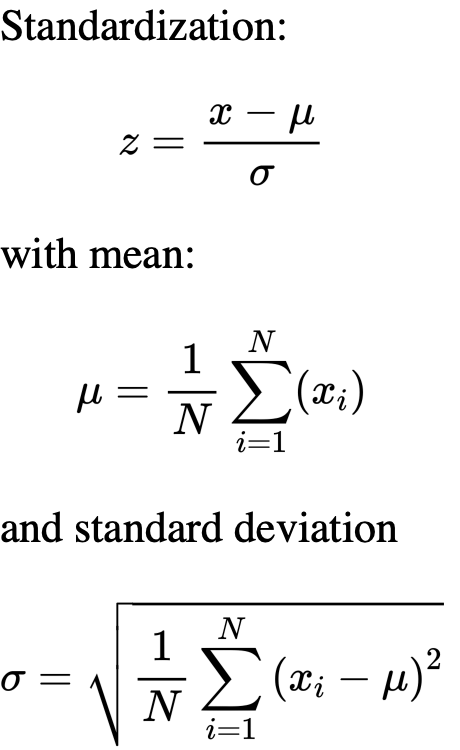
It can be explained as follows. If the actual sentiment is 0 (positive) then the cost function simplifies to -log(1-h(x)). Therefore, the closer the predicted value to 0, the closer the log() function gets to 0 and so the smaller the cost of prediction. However, as the predicted value tends towards 1 (which would be an incorrect prediction of negative rather than positive sentiment), the log(1-h(x)) tends to infinity, so the cost of incorrect prediction becomes very high. Vice versa applies if the actual sentiment was 1 (negative).



An algorithm for calculating cost given an input vector of predictions is given below.

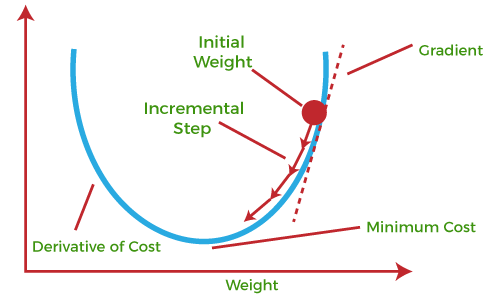
|  |  |
| --- | --- |
| Algorithm | Justification |
|  | I have used arrays for the same reasons as given above for the sigmoid algorithm. Furthermore, this should be an easy implementation of an algorithm in Python as the numpy library has an inbuilt function for calculating logarithms. |

The supplementary function of standardising the values in the massive matrix become necessary as the ML algorithm performs better when all values in the predictions vector are in a small range. The following algorithms calculate the mean and standard deviation of all non-zero values in the CSR matrix (the compressed version of the massive matrix) and return a standardised version of all the values using this equation:



|  |  |
| --- | --- |
| Algorithm | Justification |
|  | This algorithm initialises variables to hold the sum of all elements in the values array in the CSR matrix, the sum of squares of all elements, and the number of elements. After the iteration, the mean and standard deviation are calculated. The final for loop appends the standardised values to the new CSR matrix, which is outputted.  I chose the approach of creating a new matrix and outputting that, rather than standardising the values in the original matrix in place, as if I were to use that second approach, I will have lost the original CSR matrix in the rest of the code, as it has been permanently changed and I have may use it later for different functions. I have also decided to calculate mean and standard deviation as well as normalising the values within the same function because I will not need the means or standard deviations for any other part of the code so they don’t need to be in a function of their own. |

The learning algorithm for logistic regression which I will employ is gradient descent. By analysing a graph of cost against theta for one parameter in the hypothesis vector, we can see that taking a step in the opposite direction of the gradient at any point on this function will bring us closer to the minimum of the function (the minimum cost, and hence the most accurate prediction).



To implement gradient descent, an algorithm to calculate the gradient of the cost, where the derivative is given by:



…will be required, as well as an update algorithm which updates the values of parameters in the theta vector with each iteration of gradient descent, according to the following rule:



|  |  |
| --- | --- |
| Algorithm | Justification |
| https://lh6.googleusercontent.com/OWw4-QB0s34IayMfJDROCEaOTNyO1H96DZf8a9ii26RjSYE7BLCesI8uIe1YiCjfaLHKD7fltf01JoqQopMYBnfDBSgC8MKbkie-qw4Ro9YB4DHxVQ_Wsdf9dOagx8QJjG2SqwPnU7YL | (Note: this function takes the feature matrix as an input in its sparse form, not CSR).  (Also note: the third and fifth lines should iterate to len(thV)-1 and len(yV)-1 not len(thV) and len(yV)).  The algorithm takes the hypothesis vector, feature matrix, actual values vector, prediction vector and learning rate, alpha as inputs in that order.  It begins by initialising a new hypothesis vector as an empty array. Then it iterates through each prediction coefficient in the prediction vector, initialising a variable costsum for each, to hold the sum of costs of incorrect prediction for each value in the feature matrix corresponding to that prediction coefficient, j. The cost for a singular prediction is calculated on line 6 and this is added to the costsum. After iterating through each row in the feature matrix (same as the number of training examples, and the number of actual values), the equation shown above this table is used. The new hypothesis vector is outputted.  I have chosen this approach as Python allows easy iteration through arrays using for loops and the matrices and vectors which are passed into this function by value can be represented easily using 2D and 1D Python lists respectively. |
| https://lh3.googleusercontent.com/NDGxguC5-7q72wtvxNVlu2UlxVKoRYod9tv5n-WA2ABUke5YSZ09rcco35sTO7PZlG_cVBaPndDZRk6lM2nLiqgthkQiZcKLbchouftehlXo5o5hKaxUtZNSsPDwtwWTygr64FEyCaR3 | This function simply runs the above algorithm for the number of times the input specifies as iterations.  I have used this algorithm as Python allows the output of functions to be assigned to variables, so the output of gradient descent can be assigned to the new theta vector value, which is in turn passed into the gradient descent function again. This count controlled approach is much preferable to the recursive approach, which would take up far too much memory by the call stack, especially since thousands of iterations are expected for gradient descent. |

At this stage, I realise that running linear regression (another type of classifier function) for subjectivity/strength analysis is not feasible in the time frame I have. Therefore, I will instead calculate the strength metric using averages of the sigmoid function from logistic regression. Also I will have to find a dataset with binary subjective/objective labels for each text example rather than a numerical value (which would have been necessary for linear regression).

*Evaluating classifier output*

The classifier’s output in the form of a vector and the actual sentiments in the form of a vector can be assessed to work out the accuracy of the classification. The confusion matrix is a good way to visualise true and false positives and negatives.

Its calculation is described by the pseudocode below. (Note in the pseudocode, 1 denotes a prediction of positive and 0 denotes a prediction of negative).

|  |  |
| --- | --- |
| Algorithm | Justification |
| https://lh3.googleusercontent.com/XAVlDLUjH1rm37ZpDXE2FF7aZepsOz9jbiYRCDm1kJsB_FgqSkbvLBSSUtXBY8zd9BoeiBcGXMSO38O7vE3NTmXcAAfYORgCMYEdveDh6VXQ3SkmsPqlMH2ZVrqnUUzRhR7F_GbYc-Oq | Python has good facilities for iteration through lists, which can be used to represent the 2 input vectors, so this algorithm is most efficiently coded in this way. |

**Main Program (User interface)**

There are very few algorithms required for this section. I will just be using Tkinter to make the user interface. The widgets that the user interacts with will only be the two questions which have checkboxes, the search box and the run analysis button.

The one algorithm required is the validation algorithm to ensure that the user has indeed ticked at least one box for each question and has entered something into the search box.

|  |  |
| --- | --- |
| Algorithm | Justification |
|  | The function requires all o1 – o8 which represent the state of their checkboxes as inputs (1 representing ticked and 0 representing unticked). I used selection as the key structure in this function using the logical operator AND to check for the case that all the boxes are unchecked, in which case I output the relevant error message. There is however a quicker method which involves setting a count variable to 0 and adding the value of each of o1-o3 to this variable (and repeating for the second question, which has 5 options), then checking the condition that count is still equal to 0. Although I could do this in the final program, it is only slightly more efficient and saves only a small amount of time. It is more intuitive for someone reading the code to see this method. |
|  | This function is necessary to return the user choices to me after the main input window is closed. I used a 2D array for the data structure as using Python it is easy to unpack and navigate one. |

**Main Program (Representation of Classifier Output)**

*Text scraped and stored in file*

Without the use of libraries, this section of the program would require a lot of code. Nevertheless, the number of algorithms required is very low, as the only task which is not accomplished by the library is storing the pieces of text in a csv, which can be done by iterating through the list of scraped Tweets/etc. and writing them to the csv. For this reason, I will not show the pseudocode for this, as the algorithm is 2 lines long.

*Text converted to mathematical form*

The same algorithms (tf-idf vectoriser) are required for this section as were detailed previously under *Feature Extraction*, so they will not be repeated here.

*Averaging*

Several types of averages may be taken when the classifier outputs the vector of binary values or sigmoid values. These could include a simple mean and multiplying it by a large number to give an easy-to-interpret number. Depending on my judgement after testing the classifier, I will decide which algorithms to use and so I won’t display pseudocode here.

*Representation of output*

Very simple algorithms (such as those iterating through a list of classifier outputs and incrementing the relevant label, eg: positive, negative) for creation of bar charts and pie charts are required. The pseudocode is simple so won’t be displayed here. Another simple algorithm involves taking the users button click on an emotion/keyword/sentiment etc., searching the list of predictions for row values in the file of scraped text which matched that emotion/keyword/sentiment and outputting an example of the relevant text to the user. Once again, the pseudocode is simple and doesn’t need to be shown here.

**Overview of solution described by algorithms**

The basic flow of the solution is explained below. As you can see, the output of each stage is the input of the stage below it. If there is more than one output, it also becomes the input of a different stage too.

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Input | Process | Output |
| 1 | Training dataset | Cleaning algorithms (all lowercase, no punctuation, no hashtags, no @s, consecutive repeated characters removed, all words stemmed). | Cleaned dataset |
| 2 | Cleaned dataset | tf-idf matrix function | 1. Tf-idf matrix of training data 2. Word vector |
| 3 | Tf-idf matrix of training data | Learning algorithm (gradient descent) | Hypothesis vector of coefficients for each column in training data |
| 4 | User inputs (social media, analysis metrics, search term) | Text scraping algorithms | File of scraped text |
| 5 | File of scraped text | Cleaning algorithms (all lowercase, no punctuation, no hashtags, no @s, consecutive repeated characters removed, all words stemmed). | Cleaned text |
| 6 | Cleaned text, word vector | Tf-idf matrix function | Tf-idf matrix of text |
| 7 | Tf-idf matrix of text, hypothesis vector of coefficients for each column in training data | Matrix-vector multiplication | Predictions |
| 8 | Predictions, File of scraped text | Graphing/averaging | Charts, text and number to output to user |

**KEY CLASSES, VARIABLES, FILES**

Corresponding to the numbers in the above tables, these are the key classes, variables and files required for each stage.

**Stage 1**

The main input file for this stage is the sentiment140.csv file. There are about 1.6 million records in this file. There are 6 fields which are ‘sentiment’, ‘id’, ‘date’, ‘query’, ‘user’ and ‘text’. I will store the data in the file in a variable called df. It is necessary to store this data in a variable so that the pandas library can remove fields from it and allow me to select training examples/cut out records in the file as necessary.

I will reduce the size of the file by removing unnecessary fields, and only selecting 20000 each of positive and negative records, giving a new dataframe which will be stored in a variable called dataset. This new variable is necessary as the descriptive name dataset will be carried throughout the cleaning functions and storing this select data in a new variable to df means the original data will be preserved in case I need more training examples.

The variable dataset will be passed into the cleaning functions and returned, so now the data stored in the variable dataset is the new cleaned dataset, and therefore I won’t require another variable. This is the main output variable for this stage.

**Stage 2**

The main input variable here is the cleaned dataset from the previous stage. It will go through functions which transform it into a matrix as well as the function which creates a word vector of all words in the dataset. Therefore, the main output variables are called tfidfM, which is the matrix, and wordvector, which is the word vector.

I will create a class in this stage as follows.

|  |
| --- |
| Analyser |
| -trainingfile : string  -wordvector : list |
| +setTrainingFile  +setTrainingWordVector |

The class encapsulates the file containing the training examples and the word vector formed during training. It is necessary as no other operations other than class methods should be able to affect the training data and the word vector produced must be directly derived from the training file and not tampered with. The setTrainingWordVector method uses functions established outside the class/in other Python files to clean the dataset and return a word vector.

**Stage 3**

The main input variable for this stage is tfidfM from the previous stage. This feature matrix will undergo gradient descent for logistic regression to arrive at the hypothesis vector. Other input variables required here are the original prediction vector, which I will store in a variable predictionvector. It can be initialised to a vector of all 1s with length equal to the number of features (columns) in the feature matrix. Another variable is actualsentimentvector which will contain the actual sentiment of each row in the feature matrix. These variables are all required as they are inputs for gradient descent. Some more minor variables are required, like alpha (the learning rate) and iters (the number of iterations) but their role will be fully apparent when the code is written.

The main output file for this stage is theta.csv. This will store the hypothesis vector which is the output of gradient descent. A file is necessary as this vector must be stored even after the training program finishes running because the vector will be reused in the main program to classify the user inputs, and if it were instead stored in a variable, the results would be lost after the program finishes running.

A class will be created in this stage as follows. It inherits from Analyser.

|  |
| --- |
| Trainer (Analyser) |
| -outputfile : string  -iterations : integer  -learningrate : float  -datasetlength : integer |
| +setOutputFile  +setIterations  +setLearningRate  +setDatasetLength  +saveHypothesis |

The final class method, saveHypothesis, runs the classifier functions which will be imported from a separate Python file which I will create and saves the output vector in theta.csv.

**Stage 4**

The main input variable for this stage is the users inputs to the answer to the questions and prompts given by the UI, which I will call ‘searches’. I will make this variable a 2D list, as it is easy to index and retrieve the values within it using Python indexing. It is easier to comprehend this variable as a 2D list as the answers to the questions can each be represented as their own list within the main list.

An example of this variable is [[1,0,0],[1,0,0,0,1],[”ice cream recipe”]]. In the first list, 1 represents that the user would like text from the social media represented by the index and 0 represents that the user has not ticked that option. Index 0 represents Youtube, index 1 represents Facebook and index 2 represents Twitter. Similarly in the second list, 1 represents the user would like that specific analysis metric to be carried out, and 0 means they have not ticked that option. Each element in this list represents: sentiment, emotion, strength of sentiment, subjectivity and keywords. The final list contains the search term the user inputted as a string value.

The main output for this stage is a file called extractedtext.csv which contains the text extracted from the social media specified by the user in ‘searches’. The csv has only one field: text. I chose a csv file as the data can easily be stored in variables using the csv library.

I will create the following class for this stage:

|  |
| --- |
| TextScraper |
| -socials : list  -keywords : string  -filename : string |
| +setSocials  +setKeywords  +seFilename  +extractText  +refreshCSV  +readCSV |

The refreshCSV method is necessary as it removes all records in the csv file, ready for a new extraction to take place if the user chooses different keywords/social media. The readCSV method is needed as it turns the csv file into a list with each item a line in the list.

The main output of this stage is the list outputted by the readCSV method, stored in a variable named ‘newdataset’.

**Stage 5**

The input variable is the output of the previous step and it will be passed in directly to cleaning functions to turn it into a cleaned dataset, which is the output of this stage. This cleaned text will be stored in a variable (a list) during runtime which will subsequently be passed directly into the tf-idf matrix function.

**Stage 6**

The wordvector variable from Stage 2 and the cleaned data from Stage 5 will be passed into the tf-idf function to output the tf-idf matrix of the extracted text. This will be stored in a variable named Matrix.

**Stage 7**

The hypothesis vector stored in theta.csv will be multiplied by the Matrix variable from the previous stage to give the predictions, and therefore, for further processing of these predictions, they must be stored in a variable, which I will name ‘predictions’.

**Stage 8**

This stage involves outputting results of processing on the ‘predictions’ from the previous stage to the user. I am using Tkinter for my GUI and to make the task of coding a program where the user can switch between two windows (the main window and the results window) easy, I will use an object-oriented paradigm.

This is the first of two classes:

|  |
| --- |
| MainWindow (tk.Tk) |
| -title : string  -searches : list  -predictions: list |
| +minsize  +runanalysis  +opensecondarywindow |

The title attribute stores the title of the window. ‘searches’ stores the 2D array derived from the users checkbox and typing inputs (which will become the input to Stage 4). ‘predictions’ stores the result of classification which is the output of Stage 7. The minsize method will be included in the initialisation function and simply sets the size of the window. ‘runanalysis’ will validate the user inputs to the main window and then store these inputs in ‘searches’. ‘opensecondarywindow’ will scrape the required text, run the classifier and open the output window.

This is the second of two classes:

|  |
| --- |
| SecondaryWindow (MainWindow) |
| -alive : Boolean |
| +destroy |

‘alive’ holds a Boolean value representing whether or not the secondary window is open, which will be needed by the ‘opensecondarywindow’ method in the MainWindow class. ‘destroy’ closes this window. In the initialisation of the SecondaryWindow will be included the visual structure of the GUI and the functions of the buttons. I may define further functions in the initialisation (to be attached to the role of different buttons) for outputting results to the user. For example, for the user to be able to click a keyword/emotion and receive an example of text which matched that keyword/emotion I will create a function which interacts with the file extractedtext.csv and searches it based on the classification results.

The table below summarises the key variables in this program.

|  |  |  |
| --- | --- | --- |
| Name | Datatype | Justification |
| df | pandas DataFrame (similar to 2D array) | Stores the data in the sentiment140.csv file so that I can select records/process the data using Python. |
| dataset | pandas DataFrame (similar to 2D array) | Stores the smaller number of training examples that I have chosen so that they can be processed using Python. |
| tfidfM | 2D list | Stores a mathematical form of text in the training examples so that gradient descent (machine learning) can be run. |
| wordvector | list | Stores all words in the original training data so that they can be used to label columns in the matrix of the extracted text the user specifies during program runtime. The same word vector must be used as it has a one-to-one correspondence with the hypothesis vector. |
| predictionvector | list | Initialises a hypothesis vector for the first iteration of gradient descent. |
| actualsentimentvector | list | Contains the actual sentiments/emotions of the training data; necessary for gradient descent in calculating cost. |
| alpha | float | Learning rate (determines how quickly machine learning converges on the final hypothesis vector); necessary for gradient descent. |
| iters | integer | Stores the number of iterations needed for gradient descent. |
| searches | 2D list | Stores the user’s choices of social media, analysis metrics and search terms inputted. Required for the text scraping functions. |
| theta | list | During runtime of main program, values in theta.csv (the hypothesis vector) are copied into this variable for classification purposes (multiplication by the ‘Matrix’ variable). |
| newdataset | pandas DataFrame (similar to 2D array) | Stores the data in the extractedtext.csv file so that I can process the data using Python. |
| Matrix | 2D list | Stores a mathematical form of text in extractedtext.csv for multiplication by the hypothesis vector to make predictions. |
| predictions | list | Stores the sentiments/emotions outputs from classification for further processing (eg: turning into graphs etc.) |

**VALIDATION**

User inputs must all be validated to ensure the data being inputted is correct and won’t cause the program to crash, and also won’t cause logical errors with the program, such as an empty results window.

The user inputs are all managed by the Tkinter GUI. The first two inputs are ticking boxes in a checklist as the answer to two questions. Validation for this will require a function which checks that at least one box has been checked in answer to both of the questions. It can do this by checking the state of variable.get() which will be 1 if checked and 0 if unchecked, where variable represents the name I gave to that checkbox.

The third input is the search terms the user types into the search box. The only validation I should do for this is to make sure that the search box was not left empty (as even if the user’s search query is absurd and no results can be found on the internet for it, that would be expected in a program like this by the user, and they shouldn’t be restricted in what they search for fear of no results). This can be carried out by checking that the length of the searched term, once again using the .get() method, is not equal to 0.

If both validation checks described above are passed, the output window should open. Otherwise, the relevant error message will be displayed.

The other user inputs are all button clicks; Tkinter manages these, and no further validation is needed.

**TEST DATA DURING ITERATIVE DEVELOPMENT**

As I develop my solution function-by-function, I will test each of them with the relevant data. For each function (only as necessary) I will include normal data (which should give the correct output), boundary data (which should also give correct output, although the data may fall at the extremities of allowed inputs), absent data (no data is inputted) and erroneous data (the incorrect datatype; the functions should be able to manage this using try, except blocks).

However, it is important to realise that most of my functions will not, during the runtime of the program, normally have to deal with absent or erroneous data at all as they receive correct inputs due to user-end validation. Furthermore, by the nature of my program, the major testing necessary will be to see if the mathematics of my functions are correct. For example, to test the tf-idf matrix function which turns a list of text examples into a matrix, I will have to choose some rows of text as an input, print the matrix to the terminal and use my calculator and the actual formula for term frequency and inverse document frequency to verify that the values in the matrix are actually correct.

Below are my test data for each stage. I will justify why I have chosen this test data in underlined text.

**Stage 1**

|  |  |  |
| --- | --- | --- |
| Function/module being tested | Input data | Expected output |
| shortenDataset | (not data, as no input required, but still a test) dataset.head(100)  dataset.tail(100) | Should print to screen the first 100 positive text examples followed by the last 100 negative text examples.  I should easily be able to check whether the correct text examples are being displayed by browsing sentiment140.csv in Excel. It is not necessary that specifically the first 25000 of each of the positive and negative training examples have been selected; it is only necessary that the data which has been selected matches the correct sentiment, so no further tests are required. In reality, I am likely to use far fewer than all 50000 of the pieces of text in the shortened dataset. Borderline/erroneous data is not applicable here. |
| URLremover | “hi https://www.bbc.co.uk This is great”  “no urls in this”  “many urls in this <https://www.bbc.co.uk> and again https://www.bbc.co.uk” | “hi” “no urls in this” “many urls in this and again”  This tests a range of data (examples without url and examples with several urls). There is no erroneous data as in my project due to the nature of all input files, there will only be string datatypes to work with. |
| numremover | “12hiokbyr241”  “nonumstosee here”  “” | “hiokbyr”  “nonumstosee here”  The first test is normal data and the second two can be considered an absent data test as they have no digits. |
| atHashRemover | ds = shortenDataset()  ds.head(10) | Prints first ten examples without any phrases beginning with # or @  Ten text examples is plenty to check this function works. Besides, even if there are minor caveats in the workings of this function, it won’t ruin the logistic regression gradient descent algorithm. The convergence may be less efficient, but since practical machine learning involves a trial-and-error approach, more similar to engineering than computer science, this is no issue. |
| punctuationRemover | ds = shortenDataset()  ds.head(10) | Prints first ten examples with no punctuation  Ten text examples is plenty to check this function works. |
| repetitionRemover | ds = shortenDataset()  ds.head(10) | Prints first ten examples with no consecutive repeated characters  Ten text examples is plenty to check this function works. |
| stopwordRemover | ds = shortenDataset()  ds.head(10) | Prints first ten examples with all stopwords removed  Ten text examples is plenty to check this function works. |
| stemmer | ds = shortenDataset()  ds.head(10) | Prints first ten examples with all words stemmed  Ten text examples is plenty to check this function works. |
| cleanDataset | ds = shortenDataset()  ds.head(10) | Prints fully cleaned dataset  Ten text examples is plenty to check this function works. |

**Stage 2**

|  |  |  |
| --- | --- | --- |
| Function/module being tested | Input data | Expected output |
| createTfidfMatrix | dataset = [“you were born with potential”, “you were born with goodness and trust”, “you were born with ideals and dreams”, “you were born with greatness”, “you were born with wings”, “you are not meant for crawling, so don't”, “you have powerful powerful wings”, “learn to use them and fly”,””] | [[0.0218, 0.0511, 0.0511, 0.0511, 0.1908, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0156, 0.0365, 0.0365, 0.0365, 0.0, 0.1363, 0.0682, 0.1363, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0156, 0.0365, 0.0365, 0.0365, 0.0, 0.0, 0.0682, 0.0, 0.1363, 0.1363, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0218, 0.0511, 0.0511, 0.0511, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1908, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0218, 0.0511, 0.0511, 0.0511, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1306, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0136, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1193, 0.1193, 0.1193, 0.1193, 0.1193, 0.1193, 0.1193, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0218, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1306, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1908, 0.3817, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0795, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.159, 0.159, 0.159, 0.159, 0.159, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9542]]  I have painstakingly calculated the tf-idf values for this short dataset. This dataset includes many repeated words eg: “you” and a range of sentence lengths, including one with no text, which makes the algorithm fail-safe for absent data (if there are comments which after being cleaned have no words left) and ensures a range of tf-idf scores are displayed for verification purposes. |
| createWordVector | dataset = [“you were born with potential”, “you were born with goodness and trust”, “you were born with ideals and dreams”, “you were born with greatness”, “you were born with wings”, “you are not meant for crawling, so don't”, “you have powerful powerful wings”, “learn to use them and fly”,””] | ['you', 'were', 'born', 'with', 'potential', 'goodness', 'and', 'trust', 'ideals', 'dreams', 'greatness', 'wings', 'are', 'not', 'meant', 'for', 'crawling,', 'so', "don't", 'have', 'powerful', 'learn', 'to', 'use', 'them', 'fly']  This dataset contains many repeated words and an empty string, which tests whether the algorithms logic is correct (that it only outputs each word once) and that it can handle empty inputs. |

**Stage 3**

|  |  |  |
| --- | --- | --- |
| Function/module being tested | Input data | Expected output |
| sparseToCSR | [[0, 0, 1, 0, 0, 4, 0, 0, 0, 0], [0, 0, 2, 2, 0, 2, 0, 0, 0, 0], [6, 0, 0, 0, 0, 0, 0, 5, 0, 0], [1, 0, 0, 0, 0, 0, 0,  0, 0, 0], [0, 0, 0, 0, 0, 3, 0, 0, 0, 0], [0, 0, 6, 0, 0, 0, 0, 0, 9, 0], [0, 0, 0, 1, 0, 0, 3, 0, 0, 0], [0, 4, 0, 0,  0, 0, 0, 0, 0, 0], [0, 0, 5, 3, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]  [[0,0,0]] | [[1, 4, 2, 2, 2, 6, 5, 1, 3, 6, 9, 1, 3, 4, 5, 3], [2, 5, 2, 3, 5, 0, 7, 0, 5, 2, 8, 3, 6, 1, 2, 3], [0, 0, 1, 1, 1, 2, 2, 3, 4, 5, 5, 6, 6, 7, 8, 8]]  [[],[],[]]  The first matrix is a typical (albeit very scaled down) example of a sparse matrix, containing some rows with several non-zero values and some rows with only zeroes, testing the function’s logic. The second matrix is borderline test data as it is a single row of zeroes. However, the algorithm is still expected to output a 2D list with 3 empty lists. |
| vTimesCSRM | sparsematrix = [[0, 0, 1, 0, 0, 4, 0, 0, 0, 0], [0, 0, 2, 2, 0, 2, 0, 0, 0, 0], [6, 0, 0, 0, 0, 0, 0, 5, 0, 0], [1, 0, 0, 0, 0, 0, 0,  0, 0, 0], [0, 0, 0, 0, 0, 3, 0, 0, 0, 0], [0, 0, 6, 0, 0, 0, 0, 0, 9, 0], [0, 0, 0, 1, 0, 0, 3, 0, 0, 0], [0, 4, 0, 0,  0, 0, 0, 0, 0, 0], [0, 0, 5, 3, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]  CSRmatrix = sparseToCSR(sparsematrix)  vector = [0,1,2,3,4,5,6,7,8,9] | [22, 20, 35, 0, 15, 84, 21, 4, 19, 0]  This example is easy to verify mathematically. There is no need for other tests (such as vectors with a length not equal to the number of columns in the matrix) as by the design of the program and nature of the other algorithms, there will be no instances of non-multiplicably-conformable matrices and vectors. |
| runGD | (using standardisation) matrix = [[0, 0, 1, 0, 0, 4, 0, 0, 0, 0], [0, 0, 2, 2, 0, 2, 0, 0, 0, 0], [6, 0, 0, 0, 0, 0, 0, 5, 0, 0], [1, 0, 0, 0, 0, 0, 0,  0, 0, 0], [0, 0, 0, 0, 0, 3, 0, 0, 0, 0], [0, 0, 6, 0, 0, 0, 0, 0, 9, 0], [0, 0, 0, 1, 0, 0, 3, 0, 0, 0], [0, 4, 0, 0,  0, 0, 0, 0, 0, 0], [0, 0, 5, 3, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]  yvector = [0,0,1,0,0,1,0,0,0,0]  thetavector = [1,1,1,1,1,1,1,1,1,1]  iters = 3  alpha = 1 | [1.92, 0.667, 1.04, 2.08, 1, 1.33, 1.10, 1.19, 1.16, 1]  The calculation of this parameter vector (the expected output) for only 3 iterations took me 20 minutes. It is not feasible to test for more than this many iterations with a pre-calculated expected output. Therefore, during iterative development, I will run gradient descent for many iterations and multiply the parameter vector by other feature matrices to ensure the predictions are accurate. Furthermore, in the runGD function I will include print statements to print the cost at each iteration; if the cost is decreasing, gradient descent is certainly working. |
| standardise | matrix = [[0, 0, 1, 0, 0, 4, 0, 0, 0, 0], [0, 0, 2, 2, 0, 2, 0, 0, 0, 0], [6, 0, 0, 0, 0, 0, 0, 5, 0, 0], [1, 0, 0, 0, 0, 0, 0,  0, 0, 0], [0, 0, 0, 0, 0, 3, 0, 0, 0, 0], [0, 0, 6, 0, 0, 0, 0, 0, 9, 0], [0, 0, 0, 1, 0, 0, 3, 0, 0, 0], [0, 4, 0, 0,  0, 0, 0, 0, 0, 0], [0, 0, 5, 3, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]  CSRm = sparseToCSR(matrix)  matrix=[[0, 0, 1, 0, 0, 4, 0, 0, 0, 0], [0, 0, 2, 2, 0, 2, 0, 0, 0, 0], [6, 0, 0, 0, 0, 0, 0, 5, 0, 0]]  CSRm = sparseToCSR(matrix) | [-1.19, 0.204, -0.727, -0.727, -0.727, 1.13, 0.669, -1.19, -0.262, 1.13, 2.53, -1.19, -0.262, 0.204, 0.669, -0.262]  [-1.24, 0.497, -0.662, -0.662, -0.662, 1.66, 1.08]  These are two tests with different dimensions to ensure the algorithm works with different size matrices. |
| cost | actual = [0,0,0,1,1,1]  preds = [0,0.1,0.5,0.5,0.9,1] | [10^-7, 0.105, 0.693, 0.693, 0.105, 10^-7]  This test checks the cost function works for both 0 and 1 sentiment values. Erroneous data here is unnecessary as it won’t occur. |
| sigmoid | vals = [-100,-10,0,10,100] [3.72x10^-44, 4.54x10^-05, 0.5, 0.9999546, 1.0] | [3.72x10^-44, 4.54x10^-05, 0.5, 0.9999546, 1.0]  A range of numbers is being tested. |

**Stage 4**

|  |  |  |
| --- | --- | --- |
| Function/module being tested | Input data | Expected output |
| scrapeTwitter  scrapeYoutube  scrapeFacebook | “ice cream cake” | Comments/tweets/posts related to this topic in extractedtext.csv  One piece of input data is plenty. Due to front-end verification, empty inputs won’t occur. |
| refreshCSV | (None; just run function) | Extractedtext.csv file is blank  The function has a simple purpose; only one test is necessary. |
| readCSV | (contents of extractedtext.csv)  one, two  three  hi; people | ['one, two', 'three', 'hi; people']  The first line has a comma to ensure the readCSV understands that although there is a comma, it doesn’t mean end of line. There is other punctuation in the last line too. |

**Stage 5 + 6**

Since the same functions that have been used previously in stages 1 and 2 are re-used here, there is no additional formal testing required.

**Stage 7**

|  |  |  |
| --- | --- | --- |
| Function/module being tested | Input data | Expected output |
| matrixVectorMultiplication | matrix = [[0, 0, 1, 0, 0, 4, 0, 0, 0, 0], [0, 0, 2, 2, 0, 2, 0, 0, 0, 0], [6, 0, 0, 0, 0, 0, 0, 5, 0, 0], [1, 0, 0, 0, 0, 0, 0,  0, 0, 0], [0, 0, 0, 0, 0, 3, 0, 0, 0, 0], [0, 0, 6, 0, 0, 0, 0, 0, 9, 0]]  vector = [1,2,3,4,5,6,7,8,9,10]  matrix = [[2],[1],[4]]  vector = [3] | [27, 26, 46, 1, 18, 99]  [6,3,12]  The first test is standard. The second is borderline, as the matrix is hardly a matrix and the vector only has one item. Nevertheless, the algorithm should work for even these unconventional inputs. |

**Stage 8**

|  |  |  |
| --- | --- | --- |
| Function/module being tested | Input data | Expected output |
| sentimentAnalysis | [1,1,0,0,1]  [0,0,0] | Pie chart with 60% red and 40% green outputted.  Pie chart with 100% red outputted.  The first test is standard; the second is extreme, as there are no positive examples. |
| strengthAnalysis | [0.4,0.5,0.6,0.7]  [0.5]  [1,1,1] | Returns the number 20  Returns the number 0  Returns the number 100  The first test is standard. The second is extreme (as it checks the condition, if i>0.5) and the third is also extreme, as 1 is the highest prediction after the sigmoid function possible. |
| emotionAnalysis | [1,1,2,4,1,0,1,2,4,3]  [0,0] | Bar chart with 5 classes (happiness, sadness, anger, fear, surprise). Heights of each bar respectively is: 1,3,2,1,2  Bar chart with 1 class; happiness, height 2  The first example is standard and contains all the emotions. The second is extreme; ensuring the algorithm can output a bar chart with only one class. The classifier will not output any numbers over 4, so there is no point checking for erroneous data. |
| subjectivityAnalysis | [1,1,1,0,0,1,1,1,1,0,1,0] | Returns the number 0.667  This function doesn’t require much testing. |
| keywordAnalysis | Different Csv text files | Returns a word cloud of the most common non-trivial words in the file  This can be verified easily by reading the csv or even inputting its contents into an online word-cloud generator. |

**TEST DATA FOR POST-DEVELOPMENT PHASE**

For post-development, I will try several inputs into the search term box and look at the analysis in the results window. To test whether the classifier makes accurate judgements on the whole, I can use my own judgement and also read the extracted text from the extractedtext.csv file to determine the accuracy of my program. Also, I will try to test the validation by clicking ‘Run Analysis’ without checking at least one box for the questions.

Some of the combinations of inputs I will use before showing the program to stakeholders are shown in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| 1st question | 2nd question | Search term | Expected output |
| No box checked | >1 box checked | Arbitrary term | Error message “Please tick at least one option for Question 1”  Ensures 1st error message works. |
| >1 box checked | No box checked | Arbitrary term | Error message “Please tick at least one option for Question 2”  Ensures second error message worls. |
| >1 box checked | >1 box checked | Nothing entered | Error message “Please enter a search query in the search box”  Ensures third error message works. |
| Youtube checked | Sentiment, Emotion checked | “Rishi Sunak” | On clicking ‘Run Analysis’, results window displays only the buttons labelled ‘sentiment’ and ‘emotion’. extractedtext.csv contains only comments from Youtube on the topic “Rishi Sunak”. When ‘sentiment’ is clicked a pie chart pops up in the window with two classes: positive and negative. When ‘emotion’ is clicked a correct bar chart pops up in the window with 5 classes labelled by emotion.  Ensures correct buttons labelled in secondary window appear and the first 2 analyses work. |
| Twitter, Facebook checked | Strength, Subjectivity and keyword checked | “Ice cream recipe” | On clicking ‘Run Analysis’, Results window displays only the buttons labelled ‘strength’, ‘subjectivity’ and ‘keyword’. extractedtext.csv contains only comments from Facebook and Twitter on the topic “Ice cream recipe”. When ‘strength’ is clicked a number shows in the window with an explanation of how to interpret that number. When ‘subjectivity’ is clicked a number shows in the window with an explanation of how to interpret that number. When ‘keyword’ is clicked a word cloud pops up on the window.  Ensures correct buttons labelled in secondary window appear and the last 3 analyses work. |
| Click ‘Back to Search’ | | | Results window, all popups close and only original window with inputs still preserved remains.  Ensures user can easily switch between windows and use program as much as they wish. |

**MILESTONES FOR DEVELOPMENT**

1. Create cleaned dataset
2. Turn dataset into tf-idf matrix
3. Create gradient descent function
4. Store the parameter vector (output of training the classifier) in theta.csv
5. Create a text extractor
6. Create the user interface
7. Arrive at predictions from the extracted text and parameter vector
8. Output the processed predictions to the user

(These do not have a one-to-one correspondence to the stages that I identified above).

**DEVELOPMENT**

**MILESTONE 1: CREATE CLEANED DATASET**

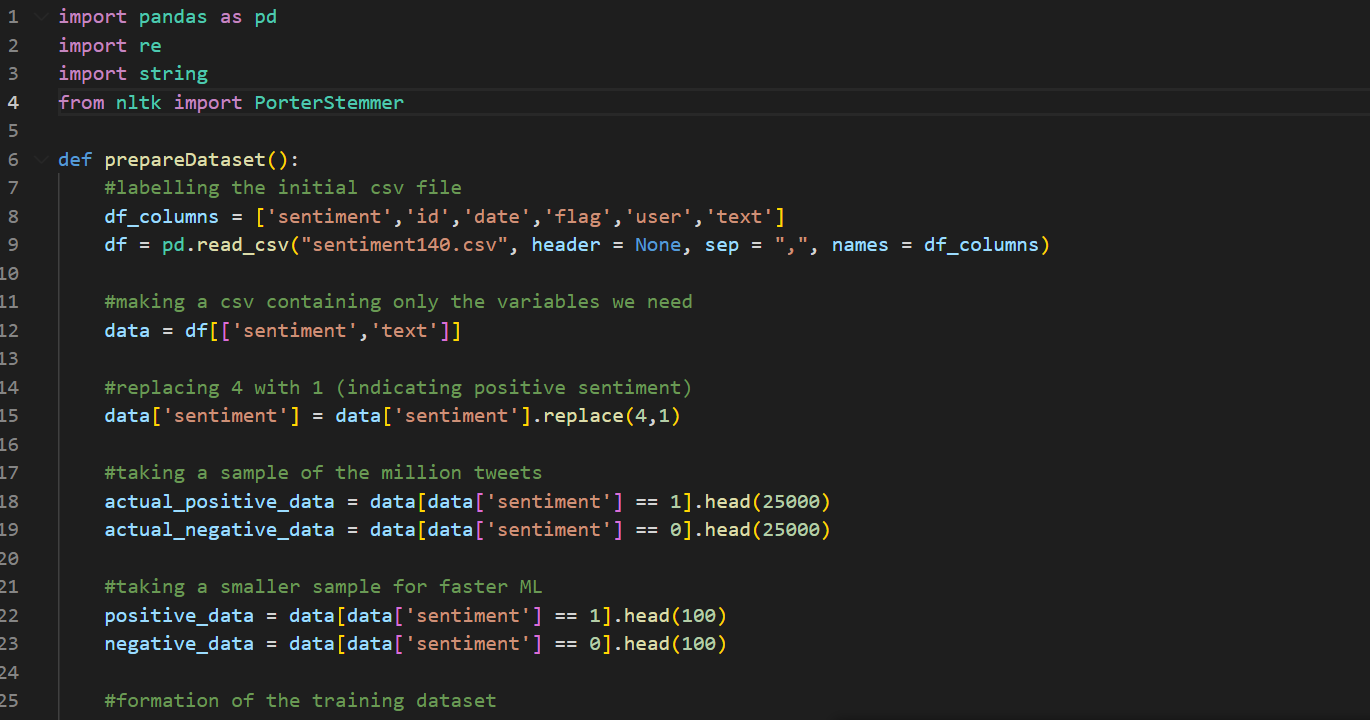
**Aim**

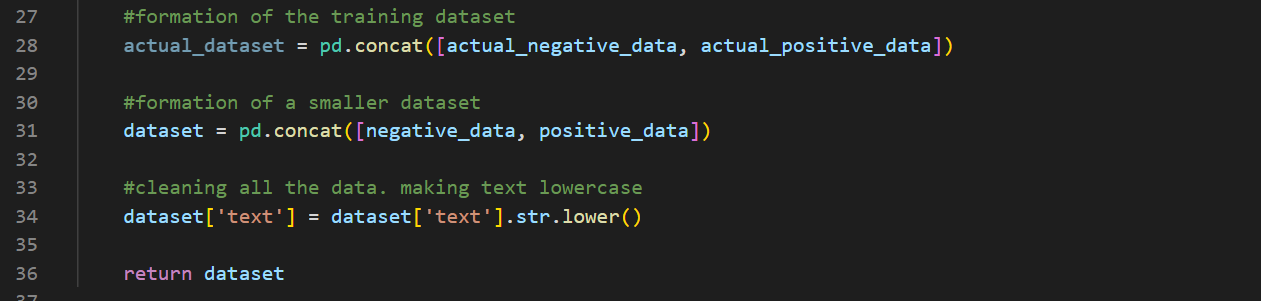
The aim is to download the sentiment140.csv file which contains sentiment data and Tweets, shorten this dataset, clean all the text and stem every word, thence arriving at the cleaned dataset, which should be stored in a variable holding the ‘pandas dataframe’ datatype.

**Code and Explanation**

The prepare dataset function reads the sentiment140.csv file and labels the 6 fields. On line 12 I create new pandas dataframe containing only the 2 necessary fields for my project; ‘sentiment’ and ‘text’.

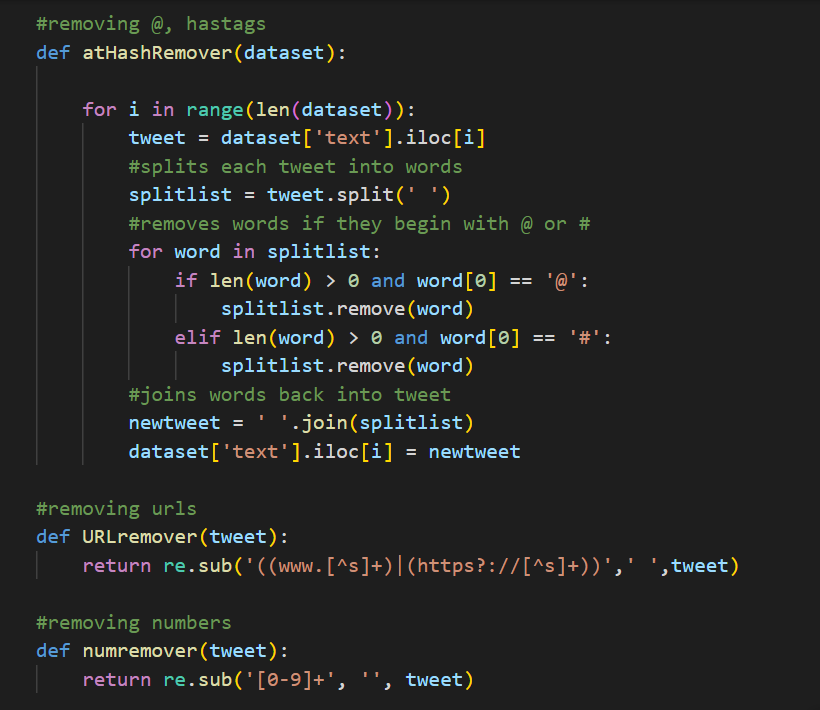
The original sentiment140.csv file labelled positive examples with 4, so on line 15 I replace these 4s with a 1, which is the convention for binary classification logistic regression. Since a dataset of 1.6 million examples is too large for my project, I take snippets of the positive and negative sections on lines 18 and 19, and combine these to create a smaller dataset. Thinking again, I realise that this dataset may be too large and time-consuming to process for iterative testing, so in a similar fashion, I create an even smaller dataset on line 31 and choose to return this 200-training-example-long dataset for the time being. Line 34 converts all text to lowercase.





The role of the code below is removing any tags in the tweets beginning with @ and any hashtags. This is because tags are usually a version of proper nouns, which don’t provide sentiment information and take up unnecessary space in the feature matrix.

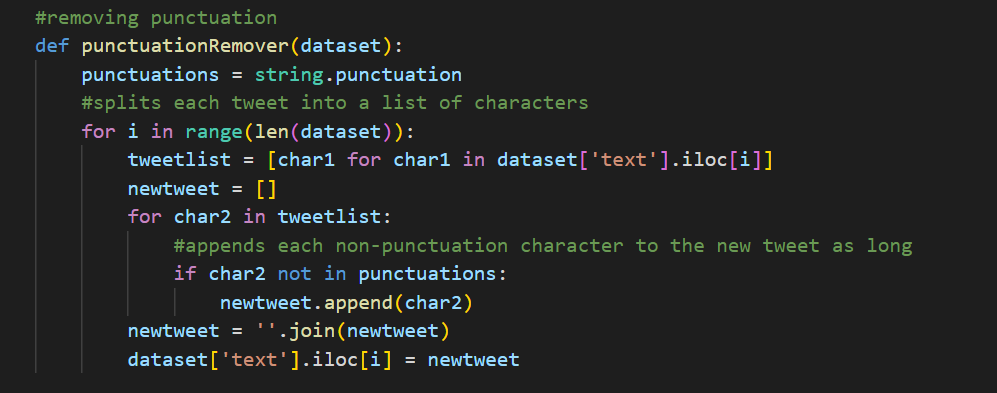
This function takes any dataset (by value) as a parameter; it is expected to be of the pandas dataframe datatype. Each tweet in the dataset is split into words (by the .split(“ “) method) and stored in a list called splitlist. Each word in the splitlist is checked to ensure it is longer than 0 characters (otherwise the index 0 will not exist) and then if the character at index 0 is and @ or #, in which case they are removed from the splitlist. The .join method combines the splitlist into a tweet and the text at the correct location in the dataset is set to the corrected text.



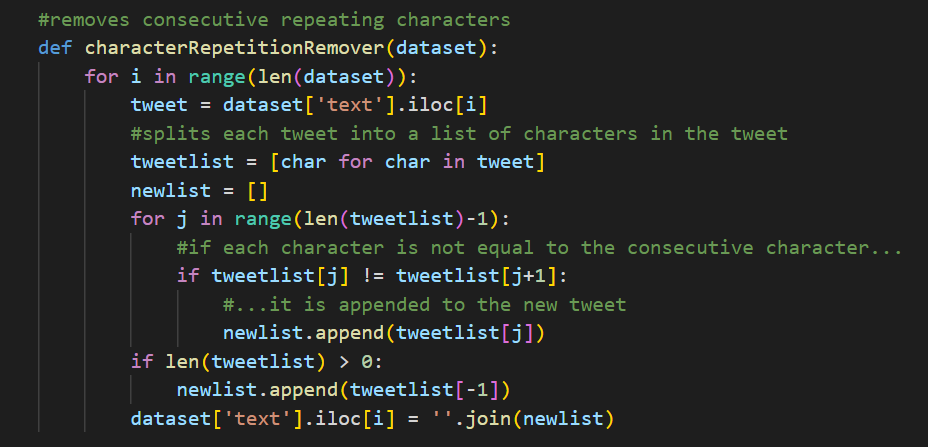
The following code removes URLs and digits from the inputted ‘tweet’ which is a string datatype. I have chosen for these functions to act on each tweet individually as these functions can be applied to the entire database easily using the .apply and lambda functionalities built into Python. I was having trouble removing URLs using the method I used to remove tags in the above function, so I resorted to the regex (regular expression) library for these tasks, which uses far fewer lines and being a library is more time and memory efficient.

The first regex expression replaces any group of characters beginning with either ‘www.’ or ‘https://’ and ending with whitespace with a single space and the second regex expression simply removes any digits.

The following function removes all punctuation in a similar way to the atHashRemover function, but in this function, rather than splitting each text into words, each text (which is a string) is split into a list of characters, stored in tweetlist. I iterate through each character in tweetlist, and if it is not is punctuations (which is a string of all punctuation, which I obtained using the string library and stored in the variable names punctuations) I append it to the newlist (which is initialised for each text). Finally, each list of non-punctuation characters is combined into a string again in the penultimate line and the corrected text replaces the input text in the last line. The .iloc[i] is the method of indexing for a pandas dataframe data structure. Due to Python’s inbuilt list comprehension methods (used on the 6th line) and .join methods, this was the first method which occurred to me.



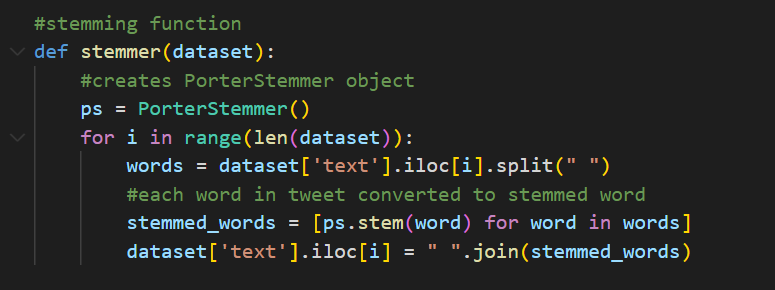
In the same way as the above code, I split each tweet in the input parameter dataset into characters on the 6th line. I then iterate through each of the characters (up to the penultimate character) and check if they are equal to the character in the next index. If not, the first character is added to the newlist variable, which is initialised for each tweet. The final character is appended to the newlist either way, as it is bound to be unique to the other characters in newlist which have been checked against their adjacent characters already. Finally, newlist is combined into a string which replaces the previous string in the dataset.



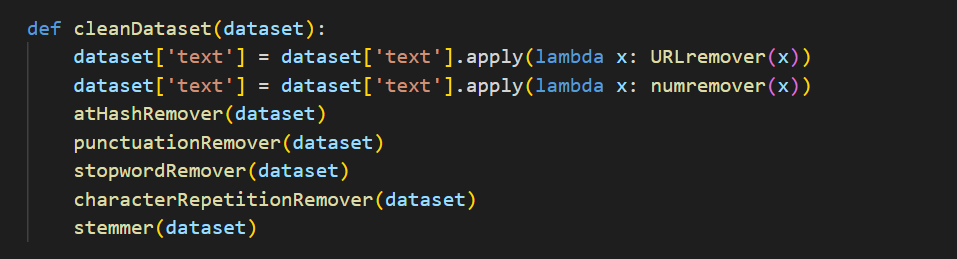
In the same way as atHashRemover, this function splits all the tweets in the database into words and removes them if they are in the stopwords list.



Similarly, this function stems all words in the dataset by iterating through each tweet. The PorterStemmer() object, ps, has a method stem() which converts a word into its stemmed word, so that all words (eg: rain, raining, rained) which contribute the same meaning towards sentiment analysis are reduced to the same form. I imported the PorterStemmer class from the library nltk.



This final function contains the other functions and applies them to the input dataset in the correct order. The correct order is vital and I will justify why I have chosen this order as follows. Firstly, URLs are removed, as if they were removed after punctuation, the regex library wouldn’t be able to identify occurences of www. or https://. Secondly, numbers are removed. Then, all words starting with @s and #s are removed; if punctuation were removed before this, they would become unidentifiable and these tags should also not be stemmed as they are useless to sentiment analysis. Then punctuation is removed, followed by stopwords. If stopwords were removed after consecutive repeated characters, they would become unidentifiable (as their spellings would now be incorrect and not match the words in the stopword list). Finally repeated characters are removed and words are stemmed. Stemming is applied and removal of repeated characters to give standardisation to the cleaning process. If repeated characters were removed after stemming, the stems would be different for each dataset and also irreversible.

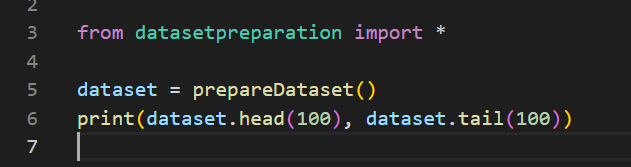


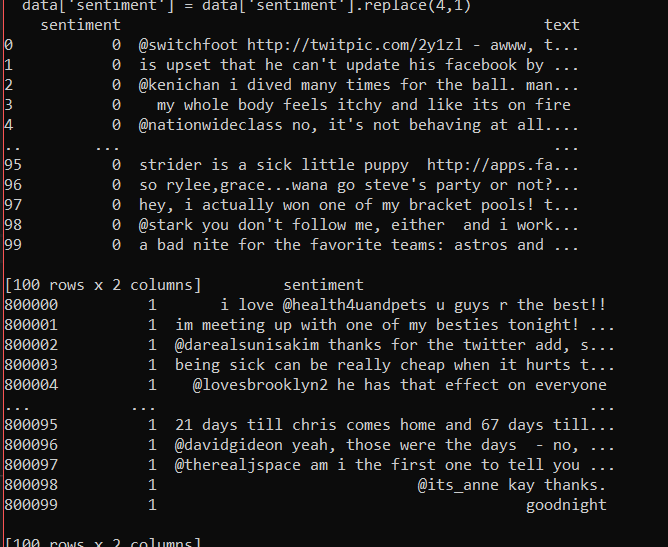
**How this relates to the structure of the problem**

Running the prepare dataset function then the clean dataset function will complete the ‘collation of dataset’ phase and also the ‘splitting into test and training data’, as the prepare dataset function could be modified slightly to select a different 100 or so examples for testing. For now, I will leave it be.

**Testing**

The following screenshots provide evidence of testing the functions with the test data specified in the table in Analysis. I have slightly changed the names of my functions in the code to the names in the pseudocode and test tables.

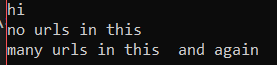
*shortenDataset*



As you can see, the first 100 negative examples (which were at iloc[0:100] in the original sentiment140.csv file) and the first 100 positive examples (which were at iloc[800000:800100] in sentiment140.csv) have been selected as the first and last 100 items of the shortened dataset respectively.

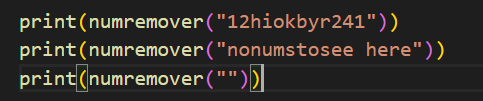
*URLremover*





The output matches the expected output given in the test table, so this function has succeeded.

*numremover*

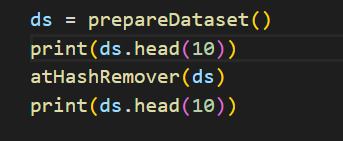


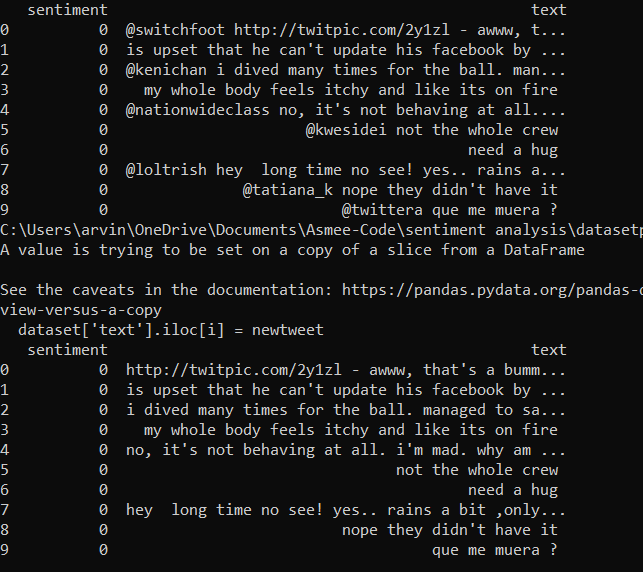




Expected output equals the actual output, so the function is successful.

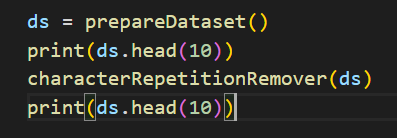
*atHashRemover*

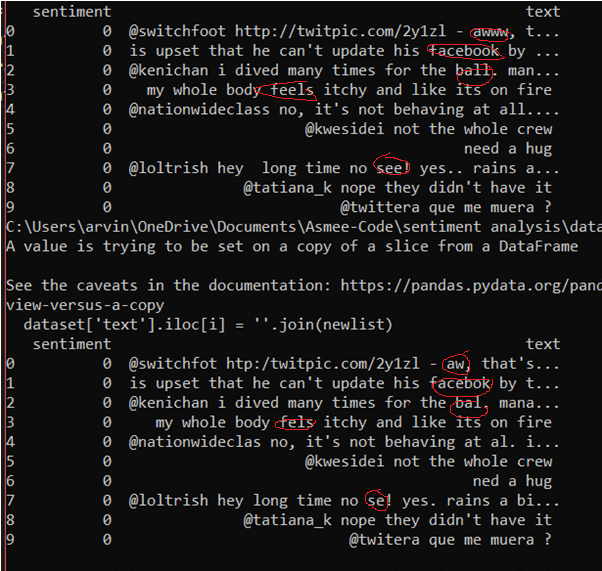




As you can see, the @s have been removed, comparing the dataset before and after the function. As there were no hashtags in the original data being printed, I tested the function on a different slice of the dataset. However, this slice also contained no hashtags. Since there are no #s in the dataset, this won’t be an issue for sentiment analysis, and even if the function was somewhat faulty for #s, it wouldn’t be detrimental to machine learning. Besides, #s are often followed by more sentimentally meaningful words than @s.

*repetitionremover*

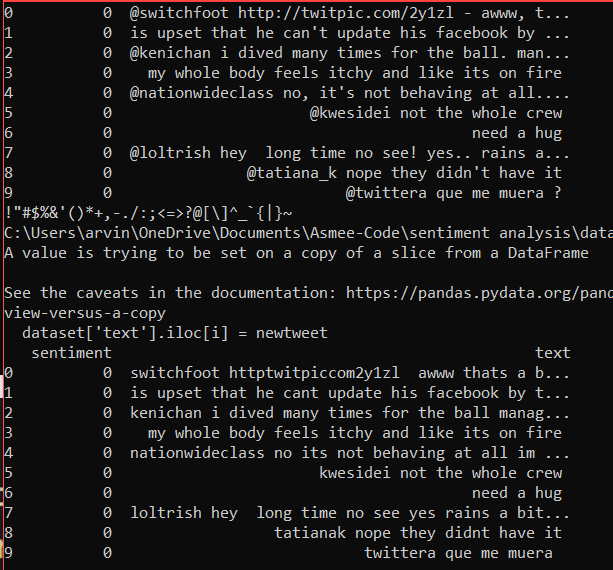




This function works as repeated characters have been removed, as shown by the red circles on the screenshot (not comprehensive, only for demonstration purposes).

*Punctuationremover*

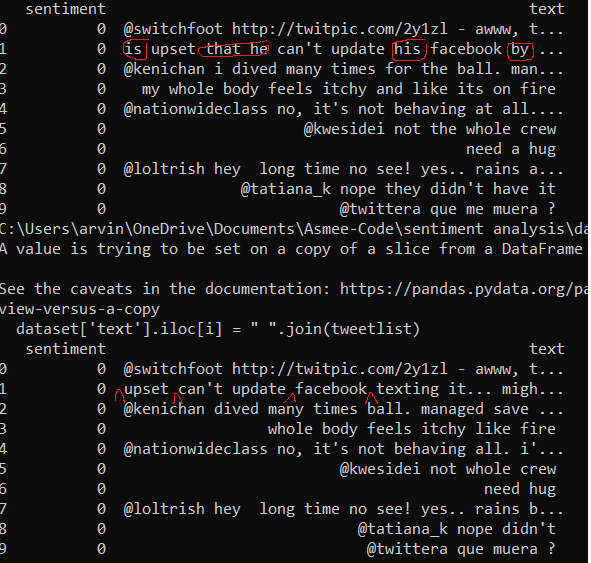




This function works as all punctuation has visibly been removed.

*Stopwordremover*



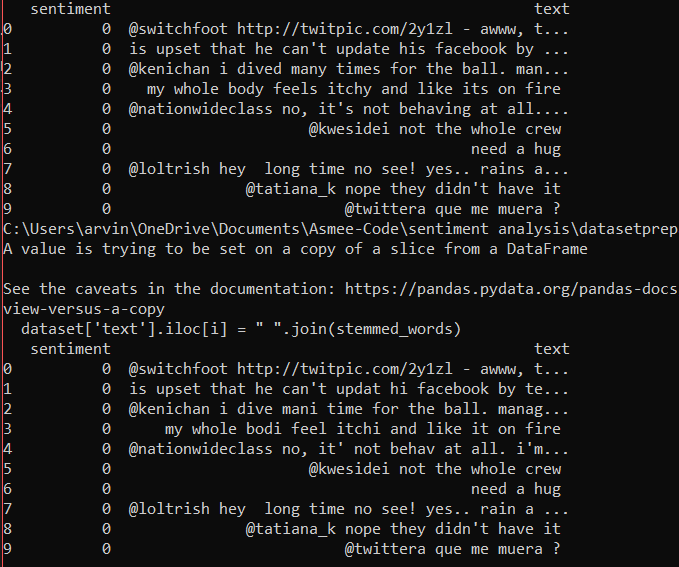


This function works as all stopwords have been removed (some examples circled).

*Stemmer*

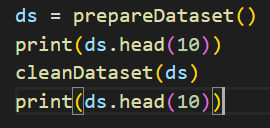
This function works as all words have visibly been stemmed (or are already in their stemmed forms).

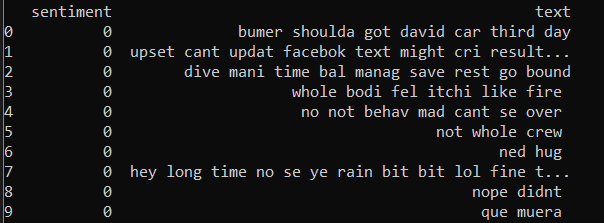




*cleanDataset*

This function works as everything (removing ats, #s, punctuation, stopwords, repeated consecutive characters and stemming) has been applied.





The original data is clear from the screenshots for the previous functions.

**Issues during development**

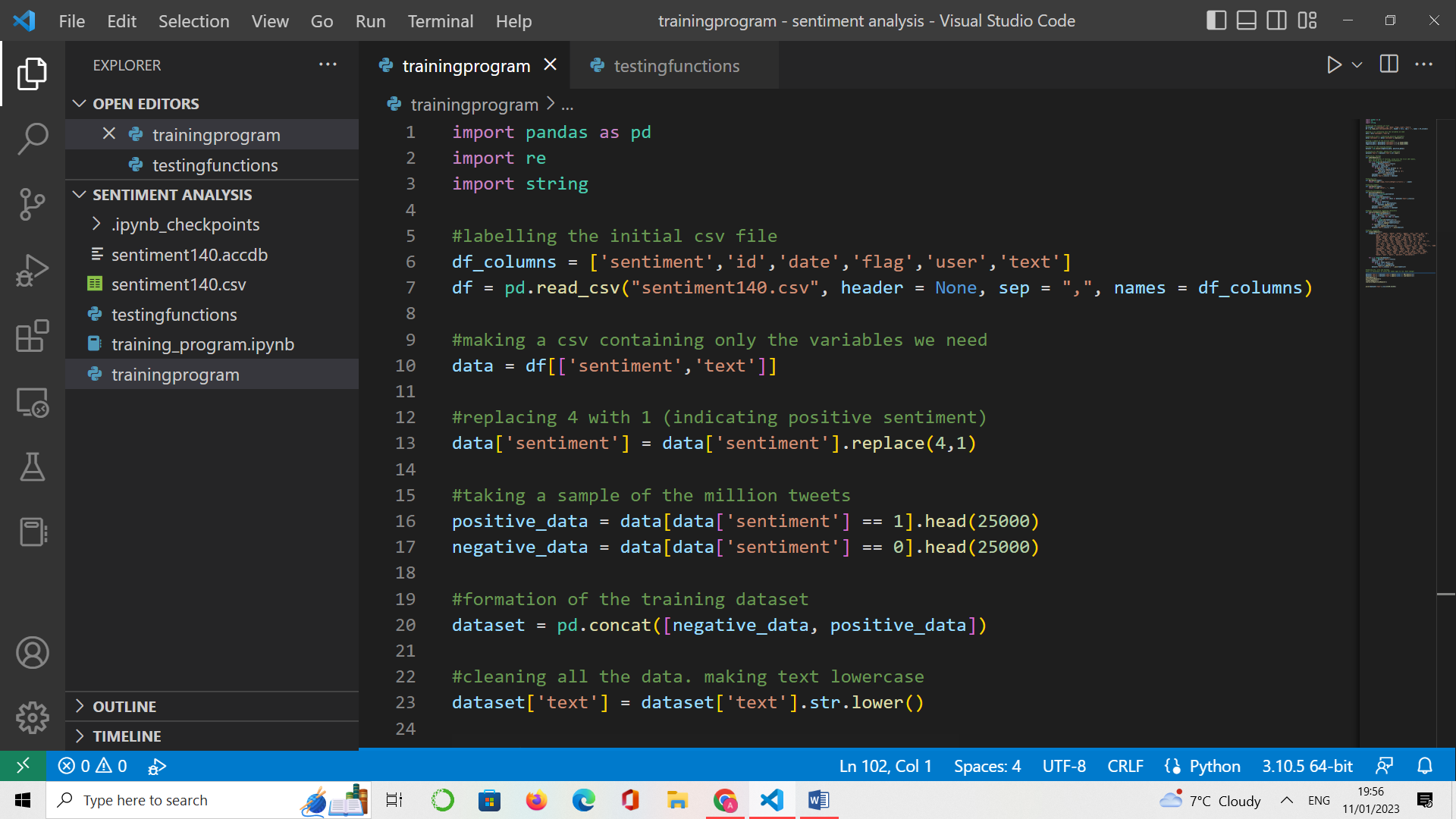
There were no significant issues during development for this module.

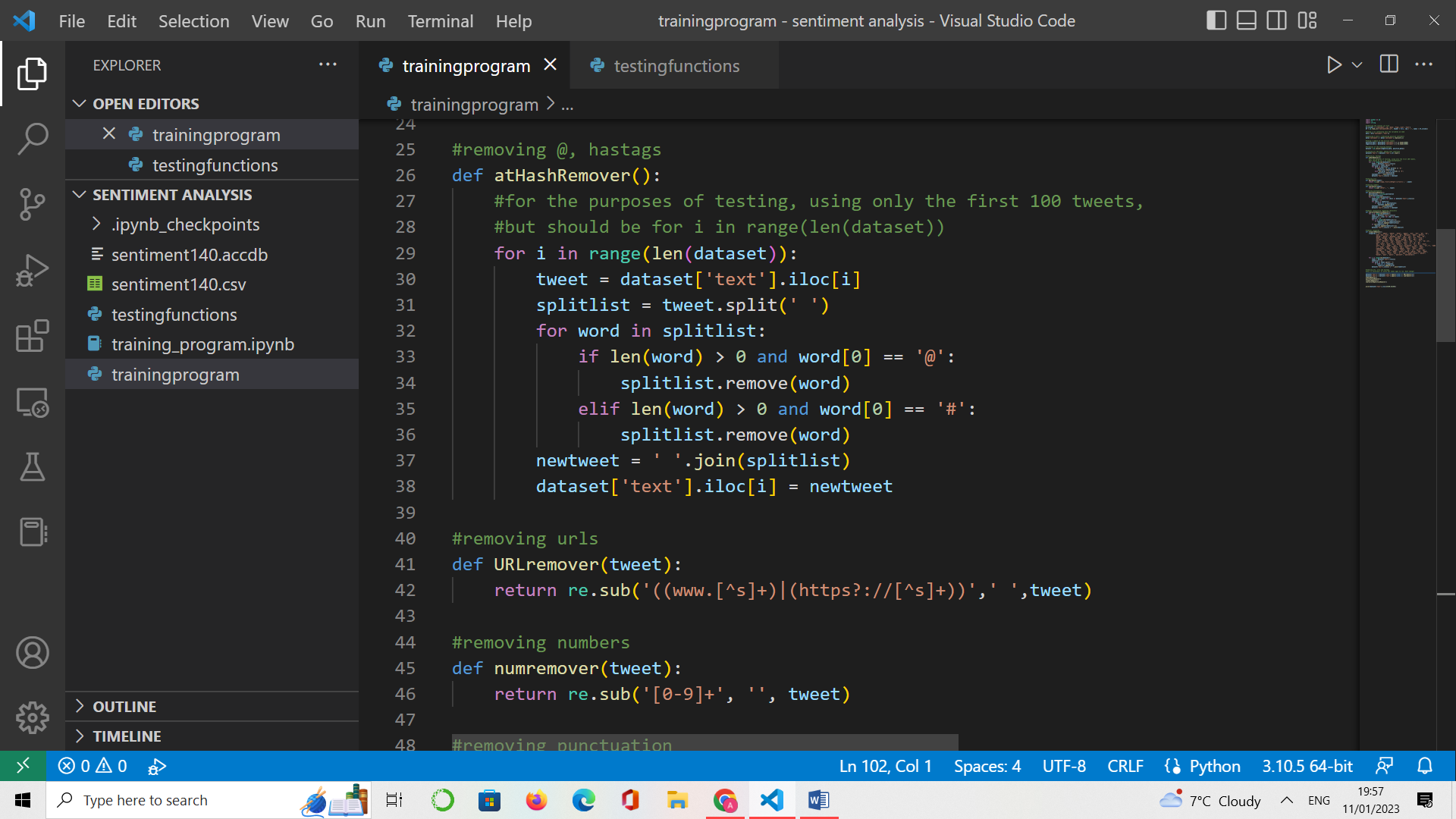
**Prototype program**

Since I have not yet made a user interface for the program and there is no machine learning going on yet, there is no prototype version of the program at this stage to show.

**Reflections**

The following screenshots will demonstrate my initial approach to cleaning the database. Essentially, I stored the entire dataframe of 50000 items to a global variable named ‘dataset’ and all the cleaning functions took no input parameters, instead directly altering the items in the ‘dataset’ variable (which being a global variable, can be accessed from within functions). This was incredibly time consuming and I had to wait 5 minutes during each run of the program for an input.





Therefore, I changed my approach and removed the existence of a global variable entirely. Furthermore, I decided to output a dataframe of 200 items only (rather than 50000) from the prepareDataset() function so that the program ran faster. At a later stage, I can easily alter the function to include more training examples, but for now, if the cleanDataset function works on 200 items, it is good enough proof that it works on more items.