FACULTY OF SCIENCE, ENGINEERING AND COMPUTING

School of *Computer Science & Mathematics*

BSc DEGREE
IN
Computer Science

PROJECT REPORT

Name: Colin Roitt

ID Number: K1610762

Project Title: Real-World Oriented investigation Into the Application of Day-to-Day Machine Learning Based Sentiment Analysis

Project Type: Research

Date: 29/04/2019

Supervisor: James Orwell

Kingston University London

Plagiarism Declaration

The following declaration should be signed and dated and inserted directly after the title page of your report:

Declaration

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computer Science and Mathematics. All verbatim citations are indicated by double quotation marks ("..."). Neither in part nor in its entirety have I made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as their own work.

Date 29/4/2019

Signature [illegible]

Real-World Oriented investigation Into the Application of Day-to-Day Machine Learning Based Sentiment Analysis

Colin Roitt - Kingston University London

2018-2019

Abstract

Artificial intelligence is the forefront of technological innovation. This project investigates its use in a real-world style application, sentiment analysis for brand perception. Multiple methods of sentiment analysis will be implemented into an application that allows day-to-day tracking of sentiment of Tweets given a set of search terms.

Contents

1 Literature Review

1.1 Natural Language Processing

“I propose to consider the question, ‘Can machines think?’.”

This is the opening remark to Alan Turing’s publication Computing Machinery and Intelligence (Turing, 1950). It was with that remark Turing began the paradigmatic shift from an numerically intelligent machine to a linguistically intelligent machine. In the article Turing puts forward his idea of a trial. Dubbed The Imitation Game by Turing, he proposes a game in which a computer attempts to fool a person into believing they are the human of a human and computer paring. Turing proposes that this is the problem that should replace the question ‘Can machines think?’, as those are very abstract concepts that can mean so many different things.

Of course this would require a defined set of linguistic rules to be developed, ways in which we could understand language mathematically.

There exists, in the field of computer science, a vast array of sub-disciplines. One such field is the computation of language known as natural language processing (NLP). This is a branch of computer science that wishes to programmatically understand human language.

We see as far back as 1951 Claude Shannon, while working at Bell Laboratories, developed methodologies to calculate entropies of, and predict matters regarding the English language. Shannon (1951), posits that a speaker of a language possesses a natural and inherent understanding of the statistics within the language. He goes onto explain this is why one is able to fill in the blanks of missing letters and even entire words.

This concept of N-gram entropy offers valuable insight into the method of computation of language. Even predating Shannon, Zipf (1929) puts forward this notion of probabilistic interpretation of phonetics in language.

These are some early examples that put forward a good foundation for developing further this mathematical interpretation of language.

Up to this point NLP was a paper based rule set for examining how humans communicate. It wasn’t until the 80s and 90s when computing power was on the up (Moore, 1998) along with the the rise of machine learning techniques such as the implementation of backpropagation, where NLP came into it’s own as a field as prediction.

Indeed the concepts here can be taken further still. Now language can be understood as a series of probabilistic events, we arrive at this concept of machine learning applied to NLP. One of the many techniques within NLP is known as sentiment analysis.

1.2 Natural Language Processing, Beyond Prediction

With the advances in technology we are now suited to solve more complex problems. Language of course stretches beyond just it's content. As data analyst, Raval (2017) suggests, NLP as a field can now be expanded, we are now able to ask the question, *"Can computers understand human emotion?"*. Sentiment analysis is the field within NLP that focuses further on the meaning of language. Moving onwards from the 20$^{\text{th}}$ century we begin to see not just an understanding of the nature of language evolve, but a drive to understand it deeper than at a surface level. Sentiment analysis looks beyond the nature of language and focuses on meaning. This is the process of identifying a positive or a negative sentiment.

Human language is complex a nuanced. Machines talk in fully structured syntax and break if it's wrong. Conversely if you wrote *"I left it in there car."* rather than *"I left it in their car."*, the reader would simply understand the error you have made, correct it in their head, and move on - rather than throwing a run time error and crashing; this would become problematic for many reasons. Posing sentiment analysis as a problem to be solved allows us then to work on new ways to give computers knowledge of the human world (Raval, 2017). Sentiment analysis is not an easy problem to solve.

1.3 Use Of Sentiment Analysis

According to McCallum (2015) more than 30% of people, when asked if they would be more likely to complain to a company if it was over a social media such as Twitter or Facebook rather than more traditional private means of communication, said they would. This represents a sizeable portion of an active market that one can draw impressions from.

It's clear social media represents a very real and significant proportion of how a brand is perceived at any given time. Not only do they represent current perception but Hansen et al. (2018) also outline a specific event called a 'firestorm', a significant event of many users posting negatively about a brand. These are a key indicator of a significant perception shift of a brand and can offer key insights into the nature of the public's relationship with the brand.

1.4 Before Sentiment Analysis

Before wide spread computer usage and machine learning came along the only way to gather data this way was through questionnaires. The problem with this lies in that fact they questionnaires tend to be long, off-putting and of little obvious value to the participant. Guorui (2012) says that the values drawn from market questionnaires such as these are valuable based on mostly, the quality of data provided by the participant.

However despite these limitations, the practice of surveying customers is still widespread. One such example of this is You Tube's Brand Lift Feature. When watching a video on You Tube you may occasionally get a advertisement you must watch first. Now, however, you may also receive a single question survey. According to one paper from Google (2018) these surveys prompt users to pick a brand they have heard of, or one they most associate with a given concept or notion such as luxury or value. These are anonymously collated and sent back to the retailer.

Surveys and questionnaires have their pitfalls, thankfully a lot of these can be addressed with modern day sentiment analysis. Perhaps it is no longer is it strictly necessary to ask a customer, you can simply see what they are already posting.

1.5 The Current Approach

Of course one way to see some of the most state-of-the-art solutions to the problem of sentiment analysis is 'Competitive Research'. *"SemEval [...] is an ongoing series of evaluations of computational semantic analysis systems"* (SemEval, 2017). SemEval does just this, by setting a series of tasks around the field each year. The technologies implemented by the teams is therefore a good way to see what state of the art sentiment analysis looks like.

Each team publishes a paper with each challenge; in 2017 Task 4 was a Twitter based sentiment analysis. The team BB_twtr attempted the task with a Convolutional Neural Network (CNN) and a Long Short Term Memory Network (LSTM Network) (Cliche, 2017).

The team tried various different methods focusing on the training stages. They found that training yielded the most optimal results when using a combination of supervised and unsupervised methods, including the word2vec (supervised), fasttext (unsupervised), and glove (unsupervised) training algorithms. The final model they entered into the competition was a combination of 10 CNNs and 10 LSTMs with various hyper-perimeters and algorithms used during the training stages.

Another team, Amobee as stated in Rozental and Fleischer (2017), went a slightly more conventional route of using a Recursive Neural Network (RNN), but used it in a more novel way. They used the RNN to 'narrow down' on certain features then past this data through a Feed-Forward Neural Network in order to define a sentiment.

While the team placed high in the results ranking with this technique they didn't beat BB_twtr. This, none-the-less gives insight into how the conventional 'home-brew' systems tend to not perform as well as established and specialised tools as used by BB_twtr such as word2vec.

1.6 Measuring Success

The competition mentioned in section 1.5 shows how well teams can thrive in competition, however in order to make it competitive there must be some quantitative measure behind the effectiveness of each teams solution.

A classic approach to judging the effectiveness would be a simple percentage accuracy.

$$Percentage\ accuracy = \frac{Number\ of\ samples\ marked\ correctly}{Total\ number\ of\ samples} \times 100 \tag{1}$$

There are limitations to this; for example, in the case of sentiment analysis on Twitter, say we chose a large sample of tweets and 90% of them turned out to be labeled as positive. The machine learning algorithm could simple learn to label everything as positive and immediately earn a 90% accuracy score. This is far from ideal and not a totally unrealistic scenario. So what other options exist?

Rather than using a clear cut method like this we can use the quantitative nature of the raw data output from the systems. This is the logic behind mean squared error.

$$Mean\ Squared\ Error = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2 \tag{2}$$

Where N is the number of outputs and y is the output value and $\hat{y}$ is the known correct value. We can see the difference in the outputs normalized into a range, and weighted (or squared) to accentuate larger errors. Allowing adjustments to be made where the error is more pronounced.

For a more representative metric we can use the F1 score metric. F1 is designed to give a mean between precision and recall of a machine learning system.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{4}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{5}$$

This system allows us to achieve what is known as a harmonic mean. This results in a quantitative measure with a relation between the precision, correct answers, and recall, robustness or number of answers wrong.

1.7 Data and Feature Sets

A large part of data analysis is selecting data itself. There are many ways in which we can present, collect, and process data. Many of these methods are relevant to how we present data to machine learning solutions.

One such method cited as being used in multiple studies referenced in this document (Pang et al., 2002; Rozental and Fleischer, 2017; Cliche, 2017) is the bag-of-words technique. Let C be a vector of possible words that may appear in a document in a corpus. Each document can be represented as a series of true and false 'bits', $[1, 0, 1, ...]$, mapping to each possible word, $[C_1, C_2, C_3, ..., C_n]$. Giving a simple mathematical notation to pass into a machine learning algorithm.

We can take this bag-of-words technique further and 'trim it down' to only include words that appear in the corpus more than others, in order to remove words that won't reliably help classify the text. Further still we can eliminate words that occur too often such as *'the'*, *'then'*, and *'of'*; allowing us to make more reliable classifications and not have words, redundant to sentiment, affect the outcome. These words are here for grammar, for humans to understand, and are superfluous to actual sentiment.

1.8 A machine learning based approach to sentiment analysis

Pang et al. (2002) is often regarded as one of the seminal works of machine learning based approaches to sentiment analysis It tackles three distinct approaches to classification with a machine learning algorithm, Naive Bayes, Support Vector Machines, and Maximum Entropy. There are of course more options for tackling the problem of sentiment analysis.

1.8.1 Naive Bayes

The Naive Bayes algorithm is an example of a probabilistic classifier; given any input, it gives one of any pre-defined classifications. The Naive Bayes is based off the Bayes theorem for stating the relations between the probability of two events. It states the following;

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{6}$$

As described in Miller (2014) this equation put forward by Bayes in the eighteenth-century, states that the probability of an event A occurring given that event B has occurred.

An example of this correlation in practice is the link between a patients age and the likelihood of that patient contracting cancer.

Taking this further, a Naive Bayes classifier is one that makes use of the logic of conditional independence. That is to say that the set of events being classified are conditionally independent of one-another, the outcome of one does not shed light onto the probability of the outcome of another, regardless of other known information.

After altering the formula somewhat to fit a dataset and implement this conditional probability we see the following;

$$P_{NB}(c|d) \equiv \frac{P(c)(\prod_{i=1}^{m} P(f_i|c)^{n_i(d)})}{P(d)} \tag{7}$$

Let $\{f_1, f_2, ..., f_m\}$ be a feature set of m length. Let $n_i(d)$ be the number of times a feature f_i occurs in a document d.

The Pang et al. (2002) paper suggests that given its simplicity and the fact that conditional independence does not actually exist in language, that the model shouldn't work very well. Yet they acknowledge that it does indeed perform better than expected.

The Naive Bayes classifier has been shown to *"[be] simple but efficient"* as well as *"work in many domains"* as stated in Sánchez-Franco et al. (2018). It's also worth noting that the same study found that the classifier performed best with a feature set of 209 features, compared to significant performance drops under 100 features and any more seeming to add little value.

The Naive Bayes probabilistic classifier has also been shown effective in the classification of multi-drug resistants cancer cells (Hongmao, 2005), and in identifying white blood cells (Jaroonrut Prinyakupt, 2015).

When implementing this into a bag-of-words sentiment analysis system, it would be simple to pass into the algorithm the feature set vector and label. It can then use these probabilities to identify unseen feature sets. A drawback with this however is you need a very large corpus of training data in order to gather sufficiently substantial probability scores for the language.

1.8.2 Support Vector Machine - SVM

A Support Vector Machine is another algorithm used within machine learning. This method uses a hyperplane to simply separate a set of n training data, $(\vec{x}_1, y_1)...(\vec{x}_n, y_n)$, into two regions. This divide is described as a n-dimensional vector known as a hyperplane. The ideal solution to an SVM is one where the margin of $\vec{\omega}$ is maximised. This is the distance between the closest data points of each class to the hyperplane (Manning et al., 2008).

The support vector are the data points closest to decision hyperplane and are used to help calculate the hyperplane equation; these are the points that would be most difficult to classify correctly.

In this 2-dimensional example we can see 3 hyperplanes show, H_0, H_1 and H_2. H_0 being the line in the middle, the decision hyperplane, and the other two being the dashed lines delimiting the support vectors themselves.

We can define a generic equation for a hyperplane as follows;

$$w \cdot x_i + b \tag{8}$$

When the class delimited by H_1 is labeled -1, the result for that equation is given as ≥ -1. Therefore the class delimited by H_2 would be labeled 1, the

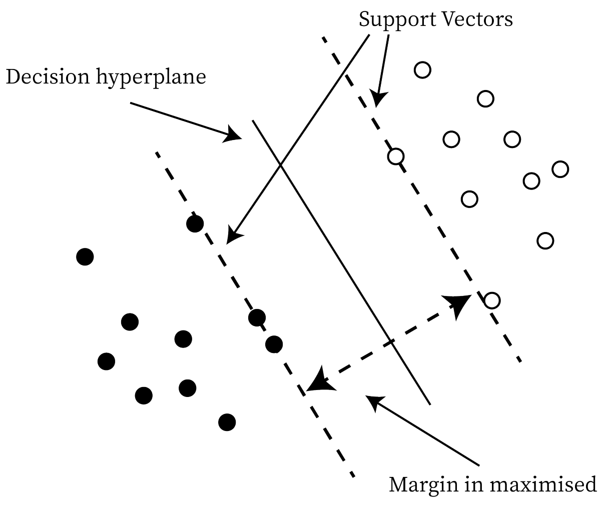


Figure 1: Visualisation of use of hyperplane and support vectors in SVM

hyperplane equation is given as ≤ 1. H_0 is defined as the median between these two points or where the equation becomes equal to 0.

This model produces a clear binary classification and can be extrapolated to many more dimensions, meaning features can be added as needed. This can be applied to sentiment analysis as this two class classification is exactly what is required. Each vector on the graph however is 2 dimensional, it can be plotted on a flat plain. Working in a higher dimensional space allows us to pass in more complex data, we can add more dimensions to the model and it will still achieve results.

1.8.3 Neural Network

The use of a fully connected neural network (NN) is also a possible solution to the problem of sentiment analysis. A neural network, being the classic concept of all the machine learning methodologies, has lots of possible implementations and applications.

The foundation of a neural network is the perceptron, often referred to as a node. It takes in multiple inputs and computes;

$$output = \sigma(inputs \cdot w + b) \tag{9}$$

Simple the node will take the matrix product of the inputs and it's corresponding weights. A Bias is then added on element-wise. Finally it's run through an activation function, allowing the output to be more complex and a simple linear solution.

A neural network is a combination of many layers of perceptrons, and can be used to model quite complex tasks, including semantic analysis. Activation functions can be swapped out at various points that allow for different outputs, these can be large integers, probability arrays, or just simply a range -1 to +1.

Stojanovski et al. (2018) found great success using a Convolutional style neural network along with other pre-trained deep neural networks to perform sentiment analysis. One benefit of using a NN over other machine learning techniques, Stojanovski says, is the reduction in feature extraction required. With a Convolutional Neural Network, there is even less feature extraction to do as the whole sentence can be fed in.

With regards to a bag-of-words implementation, thanks to the feature sets versatility it's trivial to pass this data into the machine learning algorithm. The only other thing that should be considered is the output activation function. The choice of activation function is crucial and should put the output in the correct range of the labels. A suitable choice for this would be something like a Sigmoid or a TanH function.

1.8.4 K-Nearest-Neighbor

A support vector machine calculates the distance between points and hyperplanes within the n-dimensional space; so too does this K-Nearest Neighbor (KNN) classifier. Once the data has been processed into a vector, we can easily compute the distance between the points. Once all the training data is processed, any new documents can be processed and have its nearest neighbor points found. The new data point can be assumed to have the same label as the majority of it's nearest points.

The K in KNN refers to the number of neighbors that must be found before the label is taken. The first class to reach K neighbors, closest to the new data point is the class that is assigned.

There are several ways to compute distance on an n-dimensional surface. One that is seen often is a Euclidean distance (Anton, 1994) given by the formula;

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2} \tag{10}$$

$$d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{11}$$

Where a is a set of cartesian coordinates $(a_1, a_2, \ldots, a_n)$ and b, also a set of cartesian coordinates, $(b_1, b_2, \ldots, b_n)$.

An alternative to the Euclidean metric is the Hamming distance. The Hamming distance was used primarily for error detection and bit correction...

Once again passing in our bag-of-words vector can be used in this method for sentiment analysis quite easily with 2 or 3 classes positive, negative, and neutral if necessary.

1.9 Dangerous Data

Data collection is very nuanced in and of itself. Data must be collected by humans, there's always someone behind it. Therefore it can only be expected that human biases may creep into data sets; much the same way humans learn from the data they can access, so too, machines only learn from the data we provide them.

Data can be biased, prejudice, or even just inaccurate. All this and more must be taken into consideration when reviewing and gathering it.

Perhaps one of the most stark examples of data misuse, in 2016, Microsoft launched the TayTweets Twitter account. The concept being an AI would tweet, read it's replies, and learn from them. Very quickly it started to learn from what users where tweeting at it, and it didn't take long before the AI had learned swear words, and even began reiterating racist ideology (Price, 2016). Granted TayTweets is a slightly more extreme example of bias in data, but it does illustrate that machine learning systems can only understand the world through the data we feed into them, and will only be as good as the data allows.

It was found in Caliskan et al. (2017) that given every bias tested, the machine learning algorithm picked it up. In this case the GLoVe algorithm was used.

"A bundle of names associated with being European American was found to be significantly more easily associated with pleasant than unpleasant terms, compared to a bundle of African American names" (Caliskan et al., 2017)

This highlights just how easily human biases can be transferred to machines. It can therefore be argued that data scientists must consider the ethical connotations more than ever within the realm of machine learning. Some (Wachter et al., 2017) even go as far as to suggest an AI watchdog, to regulate the decisions AI makes, especially as we progress into a world where more and more decisions being made autonomously.

But it's not just the data we feed into these systems, it's how we deal with the data we get out as we can see in the case study highlighted in Angwin et al. (2016). In Wisconsin, USA, the judges there use proprietary algorithm called COMPAS. This is part of a risk analysis system that is employed in parts all across the United States and other parts of the world. You get the prosecuted individual to answer a questionnaire and it calculates a risk score, mostly on the likelihood of re-offending. In 2013, Paul Zilly was charged with theft of a lawnmower. Him and his lawyer agreed that he would put forward for a plea deal as suggested by the prosecutor. He admitted the crime for a year or so sentence, and some supervision after the fact.

It was during the court case the judge was allowed to see Zilly's COMPAS score. "it is about as bad as it could be" Judge James Babler remarked. The judge overruled the deal and Zilly was sentenced to two years in prison and subsequent three years of supervision.

The COMPAS score has an accuracy of about 70%. So the rest of the time it was wrong. On top of that this system was in place to aid in deciding if someone deserved bail or not. Recidivism is a complicated topic and it's hard to say who will and wont re-offend, we certainly aren't able to predict with an algorithm yet 100% of the time.

Judge Babler had put more weight on the output of this algorithm than in the knowledge and background of everyone else in the judicial system, working on this case. We cant say for certain if this was the correct decision but if we idly put computers decision in front of our peers with such high stakes then we must be prepared to defend the systems in place. And besides 70% seems awfully low for such an important decision.

2 Methodology and Analysis

2.1 Introduction

We must now begin to develop a strategy to investigate the implementation of machine learning techniques to tackle this problem of social media sentiment analysis.

In order to properly investigate the techniques we will assume the role of a project developer. The project that must be made is a system to track the public perception of a brand or product via social media. Further on we will begin to identify the aspects of this project in more detail, developing functional and non-functional requirements along with epic and user stories.

The motivation behind this is based on understanding a demographic. As outlined in the Use Of Sentiment Analysis section, understanding your market is not only useful but key. The business insight that can be gained from knowing current public perception allows a business to react to, and even change, how they are being perceived.

2.2 Agile

In the fast paced sector of technology, it's important to maintain the pace of development with the rest of the market. New things come in constantly and changes need to be made as soon as possible if one wishes to beat their competitor. It's important to also respect the value of a coherent team to work with. A well-oiled machine of smart-working individuals will be far more likely to succeed than a group that works to a script.

This is the ethos surround the Agile project management style. While unable to adopt all aspects of an Agile framework workflow (of which there are many) due to team size, we can take several key aspects;

Open documentation - A key aspect of Agile systems is reacting to the changing environment. In order to reasonably respond to this we can allow stated goals and expectations to change. This is different to scope-creep. All changes should stay reasonably within scope of the project and changes should only be made that are reasonable. This is something that should be review case-by-case.

Project backlog - These are tasks that need to be complete in order to achieve all the goals of the project. They are defined at the start of the project, however like all documentation it is subject to change at any point during development.

Iterative development - One of the largest parts of agile is the sprint, sometimes called time boxes. Sprints are the time frames in which one can expect the team to implement one or more items from a project backlog. They can range from 2 to 5 weeks and are hyper-focused on specific tasks.

2.3 Stories

In order to understand the nature of the project and to identify requirements we can construct user stories. These are statements from the end users' point of view that state exactly what they want the product to do and why. This allows the perspective to be shared among the team of developers and also gives an insight into how the end user will use the product.

Epic Story:

> As a marketing manager I would like to be able to use a web application to track the social media perception of a certain product or brand using machine learning and sentiment analysis so that I can better understand the brand or products current market position.

User Stories:

> As a marketing manager I would like to be able to access a dashboard so that I can set various options for running sentiment analysis.
>
> As a marketing manager I would like to be able to edit the search terms, or selection of posts, that are used to crawl the social media website and find posts so that I can adjust exactly what sort of posts I want to find.
>
> As a marketing manager I would like to be able to set a specific social network, and have the application pull in posts from users on that.
>
> As a marketing manager I would like to be able to click a run button that will then collect social media posts and perform sentiment analysis.
>
> As a marketing manager I would like to have the results of the sentiment analysis presented to me in such a way that it allows me to easily understand it's results
>
> As a marketing manager I would like to be able to see a selection of the posts analyzed in order to better understand the results of the analysis.
>
> As a marketing manager I would like to be able to set up an alert to send a push notification when a significant event occurs.

2.4 Requirements

The following section outlines various requirements that should be taken into consideration during the design and development process of the application.

Each requirement is labeled with a priority using the MoSCoW system. MoSCoW stands for Must have, Should have, Could have, Won't have time for. It is in and of itself a very useful tool for making clear which parts of the scope are expected to be completed reasonably within the time frame, and which bits can be sacrificed should delays be incurred.

2.4.1 Functional

1. Create a sentiment profile [Must have]
   - 1.1. Set a new name [Must have]
   - 1.2. Save new profile [Must have]
2. Set some search parameters [Must have]
   - 2.1. Take in individual words or phrases [Must have]
   - 2.2. Take in social media profile [Should have]
     - 2.2.1. Twitter [Should have]
   - 2.3. Pick a date to back-date search from [Could have]
     - 2.3.1. Date must be in parameters [Could have]
     - 2.3.2. Must be within 7 days of current day [Could have]
     - 2.3.3. Format mmm/dd/yyyy [Could have]
   - 2.4. Pick a maximum number of tweets to search for each day [Must have]
     - 2.4.1. Between 1 and 100, inclusive[Must have]
3. See Sentiment report [Must have]
   - 3.1. Pull posts from social media [Must have]
     - 3.1.1. At regular intervals [Should have]
     - 3.1.2. The number of times a day specified [Should have]
     - 3.1.3. Assign each tweet a sentiment as it comes in [Must have]
       - 3.1.3.1. Use a pre-trained classifier [Must have]
   - 3.2. Display a selection of tweets and their respective sentiment [Should have]
   - 3.3. Show sentiment over time on graph [Must have]
     - 3.3.1. Hover over point to see the tweet that is that point [Could have]

3.4. Show positive and negative bar chart total [Should have]
  3.4.1. Separate into stacks with regards to confidence [Could have]
    3.4.1.1. $< 50\%$
    3.4.1.2. $> 50\%$
    3.4.1.3. $> 80\%$

4. Set alert [Should have]
  4.1. Add an alert [Should have]
    4.1.1. Set a name [Should have]
    4.1.2. Set Sentiment to look for [Should have]
      4.1.2.1. Positive [Should have]
      4.1.2.2. Negative [Should have]
    4.1.3. Set Bound to measure by [Should have]
      4.1.3.1. $<$ [Could have]
      4.1.3.2. $>$ [Could have]
    4.1.4. Set percentage as a threshold to look for [Should have]
  4.2. Delete an alert [Could have]
  4.3. Receive alert [Should have]
    4.3.1. Get an email with a message stating which alert has been triggered [Could have]

2.4.2 Non-Functional

1. The interface must support multiple configurations with regards to search terms
2. The system must compile tweets on a regular and up to date basis
3. The system should make reviewing how the tweets have been processed clear through the use of graphs
4. The database should store the data until a profile is removed
5. Twitter data should be collected via the twitter API
6. Alerts should be triggered as soon as the data is collected and sent to the users email
7. Machine learning processes should take place on a back-end Flask server through an API
8. Development of code should consider the separation of concerns. There should be a distinction between presentation code and business logic.

9. Front end should be hosted on a NodeJS server and built with React to be data driven
10. The machine learning algorithms that should be implemented are a K-Nearest-Neighbour classifier and a Neural Network classifier

2.5 Project Analysis

Now requirements have been defined we can look into what the output of the project may be and where the end product will sit within the wider world.

2.5.1 PEST Analysis

We can begin by conducting a PEST analysis. This gives an overview of the landscape in which the project will sit. It's designed to consider 4 areas of the anthropocene, political, economic, social, and technological, and allows these areas to be explored through the opportunities and threats they may lead to. There are multiple iterations of the PEST model that can be used also; these include the addition of other areas such as legal, environmental, and ethical. The one best suited to the project should be chosen, in this case the basic version has been used.

	Opportunities	Threats
Political	Politically divisive times that cast peoples opinions to social media	GDPR regulations must be considered with regards to storing personal data
Economic	With any risk of downturn economically, understanding brand perception gives focus areas for marketing	With any lack of public spending business are at risk
Social	People very likely to use social media to voice opinions of a product/brand Social media hosts large share of many markets	Social Media being viewed as negative and so receiving less use People more likely to talk about negative experiences introducing bias
Technological	More machine learning techniques are developed and refined everyday	It's difficult to go beyond binary positive or negative sentiment

2.5.2 SWOT Analysis

Now we can look specifically at the project and look at it's four attributes; strengths, weaknesses, opportunities, and threats. The strength behind a SWOT analysis is that it allows a wide overview of a project to be summarised into a small table, and even allows for an better risk analysis work flow.

Strengths	Weaknesses
Established techniques for sentiment analysis means plenty information with regards to how they work Online website allows constant searching of twitter even when user is offline The website can be accessed from any system without any installation	The Twitter API only allows free accounts access to a 30 day of tweets An online system needs to be able to communicate constantly with the backend database and as such will not be of any use should the user not have an internet connection Web applications are inherently slower than desktop ones, this is due to the need to transfer data of the internet connection – no information is stored locally
Opportunities	**Threats**
Due to the modular design of web systems and flask applications sections can be swapped out or extended upon. It would be simple to add access to the Facebook API for example, should that be off interest	An online system means anyone can access it and could exploit any known bugs to take control of the system Should either of the web services that run the system go down the system as a whole will fail For a constantly running application extra measures with regards to hosting should be considered for maximum uptime

3 Design

3.1 Introduction

We can now begin system design. This is a critical step in the development of an application. Should something be wrong here there is a chance it may cause delays further down the line.

3.2 Technology Stack

We can start with understanding what will be used to create each part of the application.

One of the primary language that will be used in the development of this application is Javascript. Developed for client side use, it allows basic HTML, markup, webpages to become interactive. Javascript has come along way since the ECMA standard was introduced. It has been ported to run as a server-side language with NodeJS and grew from there with thousands of libraries.

The user focused area will be a client side React app. React is a Javascript library that allows data driven user interfaces to be built. React focuses on breaking down each part of a web page into a component. From there, each component has a state; a Javascript object that stores information about that component, such as the data it should be showing. When using React HTML is never written directly. JSX is written, which is parsed through an interpretor called Babel that converts it to Javascript which in turn generates HTML in the client's browser. Within JSX one can embed variables and values from the components state. While this all sounds very complicated, the bonus of this is that when the state of the component is updated the view will automatically re-render whatever needs to update. As long as the code is written to get new data, the view will follow suit.

One step up the chain from React is the NodeJS server that will compile and serve the react to the user. Not a lot needs to be done in this case as React contains a publishing set of features that essentially contains all the necessary components for a NodeJS server.

Another step up the Tech Tree and we arrive at the Python driven API.

An API is an Application Programming Interface. It's a way to easily communicate with another program. In this case we will make API calls from the client side React code to the Python code. The API will contain the bushiness logic of the application. Functionality such as scrapping Twitter or running machine learning code.

The API itself will be written in the Python programming language. The justification behind doing this over writing that code in Javascript is that the machine learning suite in Python is more mature. There are a multitude of helpful libraries in Python such as Tensorflow for building models, and Numpy for doing complex matrix maths. Few of these exist for Javascript, it's meant for a client side API call, or a quick edit to a page. Putting so much weight on it to perform complex maths is not within it's scope. While not impossible, Python is certainly the better way to go.

How do we get Python to connect to the internet? We can use the Python library called Flask. Flask is a simple yet effective way to manage Python web code. After defining some routes (urls), functions can be assigned and HTML templates returned. And this of course, comes with the full functionality of Pythons collection of libraries and documentation.

There's one more link to put together however. That is where all the data is stored. We need a database to keep data while it's not being used. That will be in MariaDB server. MariaDB is fork of the MySql repository. For all intent and purpose it is the same. MySql is a relational database. Data is accessed, added, and modified through SQL queries; a powerful high level language that allows fine grain control of what data comes back and the structure the data will come back in.

This is the entire stack of technologies can be informally shown as the digram below;[1]

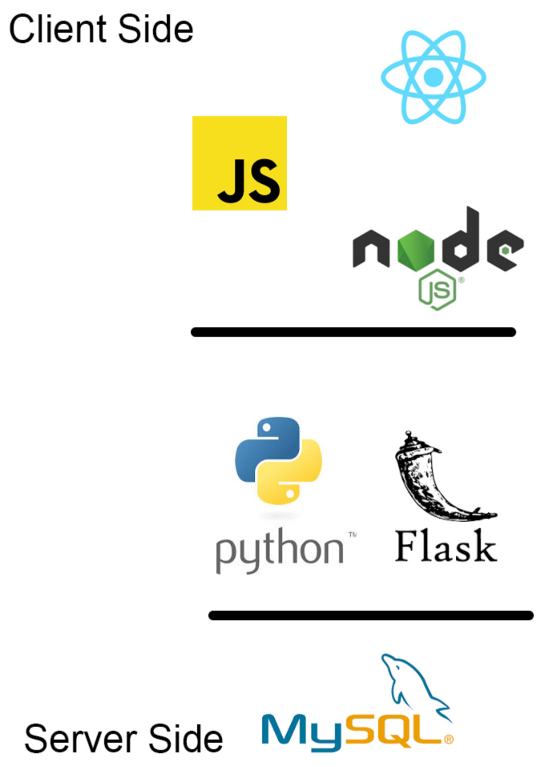


Figure 2: The stack of technologies used and their respective languages

3.3 UX and UI design

We can now begin to consider the user interface, and design with respect to both our user requirements and general UX guidelines. The requirements are broken down into sections based on their functionality and as such we are able to design to this specification.

Inline with our agile methodology this is an iterative process taken in 3 steps. We start with a low fidelity wireframe; a rough idea of layout and where certain aspects of the application should sit on the users side, in black and white. We

[1]The brand names and logos are trademarks and properties of their respective organisations

then carry on to our higher fidelity wireframe. This is an almost fully in colour perhaps with some place holder images, things are subject to change. It will be made in some sort of designer such as Adobe XD or InVision and made to be interactive. The final stage is the programmed prototype. This is the design near completion; written in code it's a fully interactive version of the front-end with all front-end functionality working.

3.3.1 Low fidelity wireframes

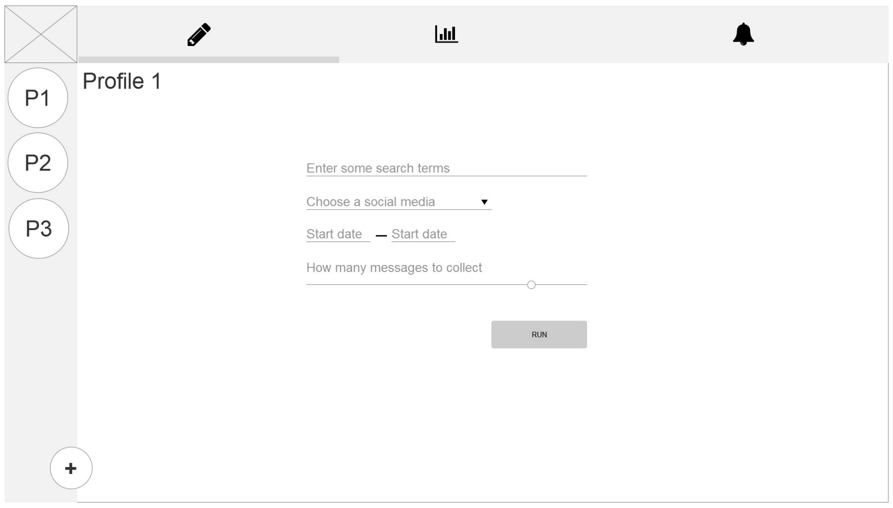


Figure 3: Low fidelity wireframe of profile set-up page

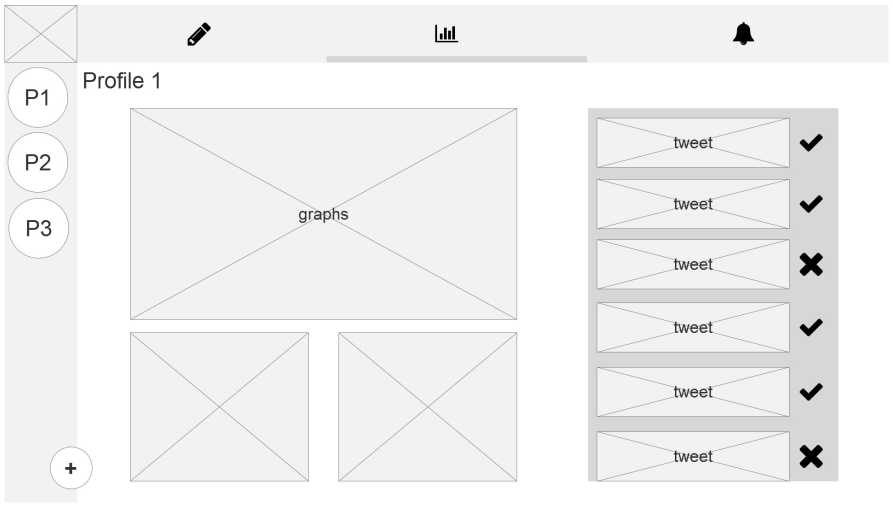


Figure 4: Low fidelity wireframe of the view profile results page

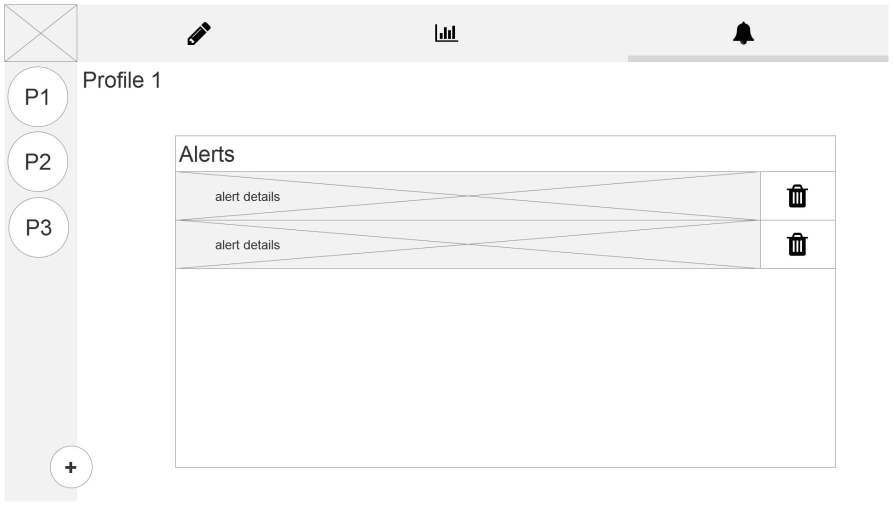


Figure 5: Low fidelity wireframe of the add alerts to a profile page

3.3.2 High fidelity wireframes

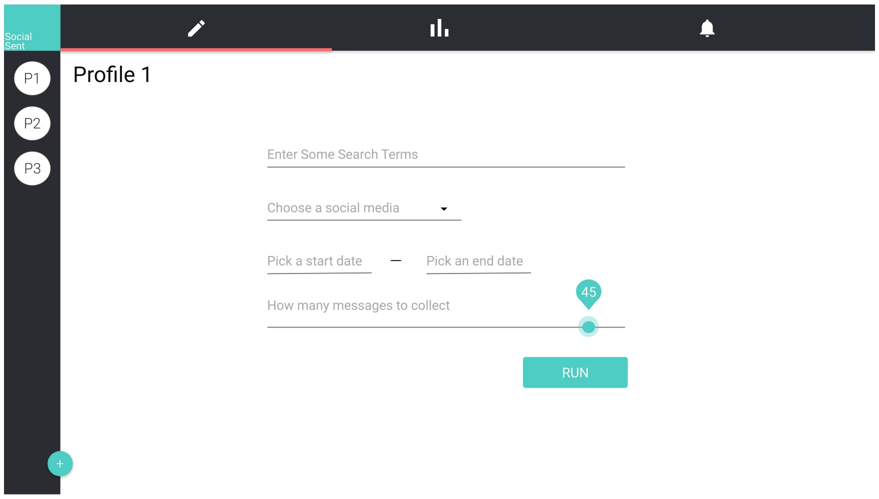


Figure 6: High fidelity wireframe of profile set-up page

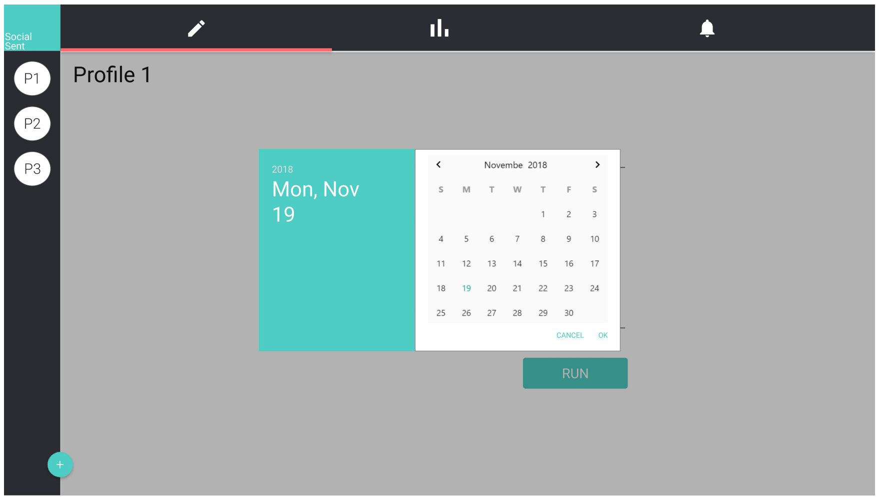
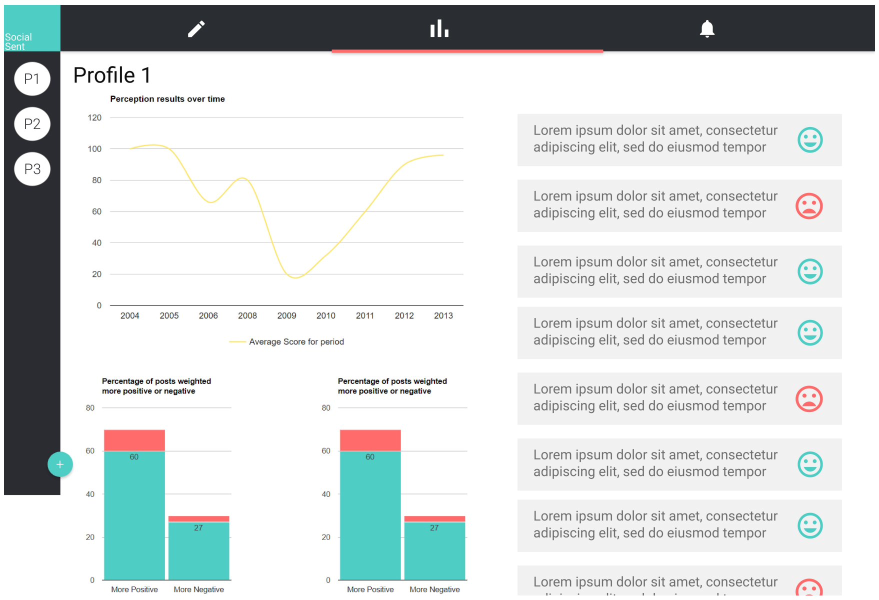


Figure 7: High fidelity wireframe of date selection UI of set-up page



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

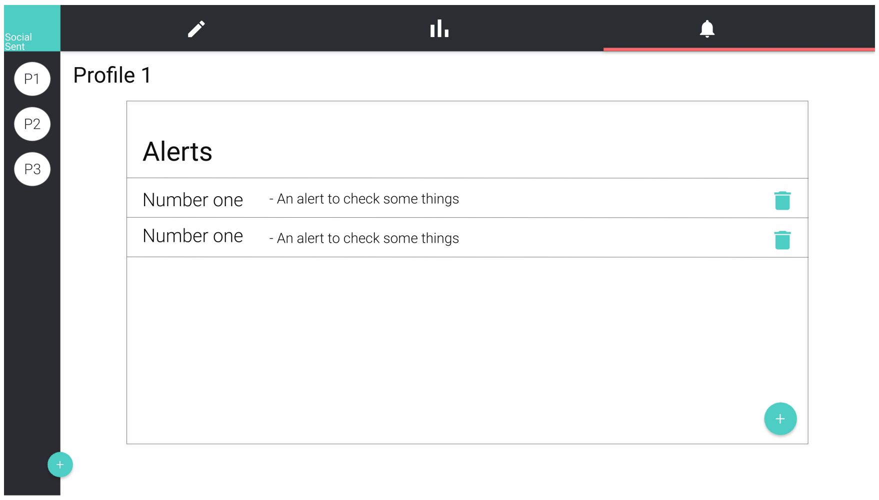
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur

Figure 8: High fidelity wireframe of the view profile results page



Alerts

Number one - An alert to check some things

Number one - An alert to check some things

Figure 9: High fidelity wireframe of the add alerts to a profile page

3.3.3 High fidelity, Interactive Prototype

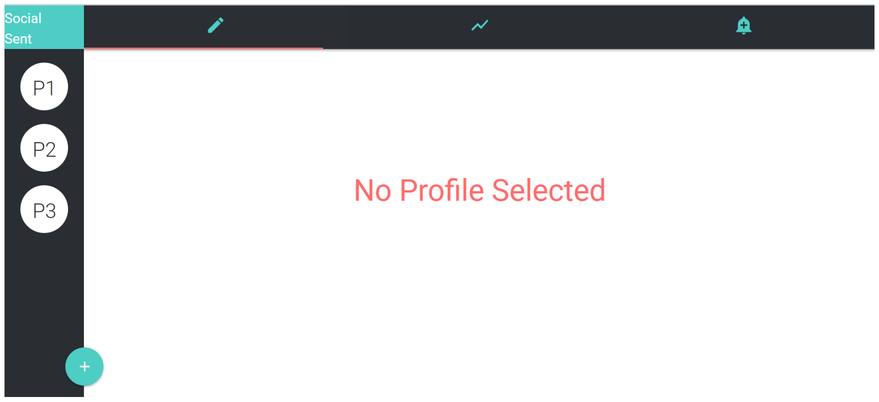


Figure 10: Interactive prototype, no profile selected page, landing page

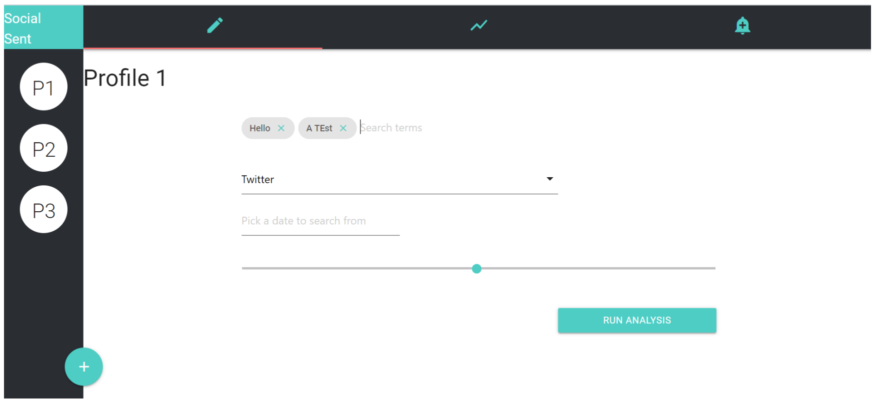


Figure 11: Interactive prototype, profile page

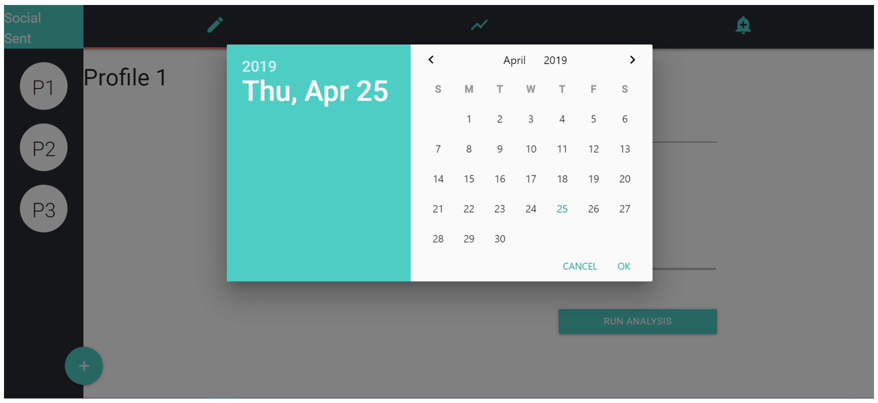
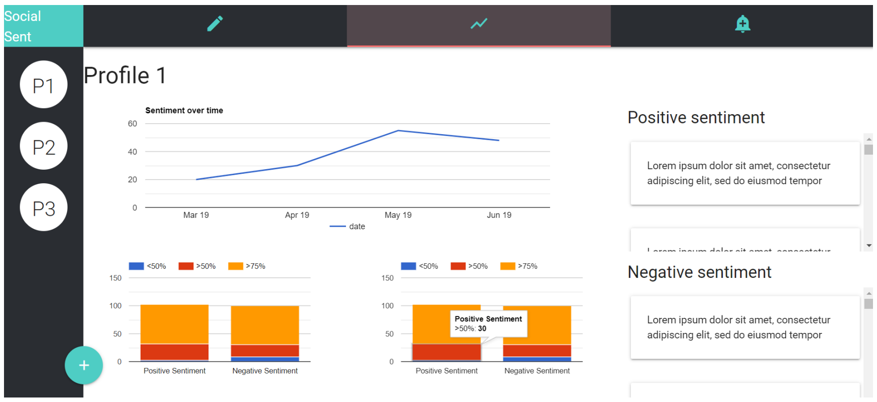


Figure 12: Interactive prototype, profile page, date selection



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Figure 13: Interactive prototype, view profile data page

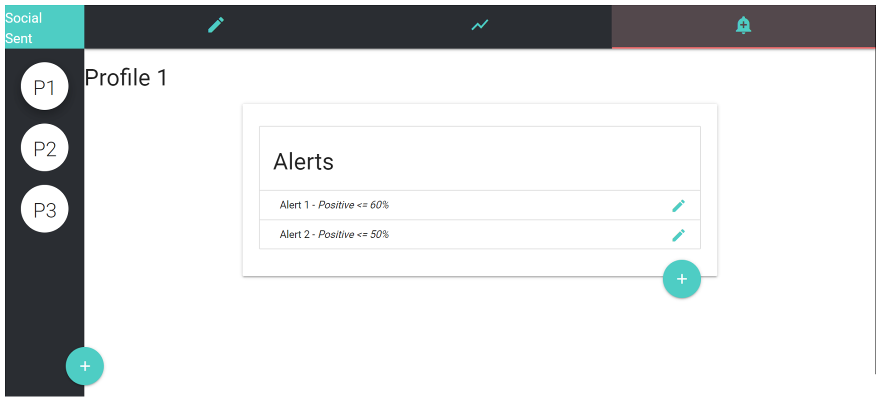


Figure 14: Interactive prototype, view profile alerts page

3.3.4 Design rational

The decision was made early on to implement a Material design style as this style of clean interface was both favoured by project stakeholders, and it leans well on existing UX paradigms that users are familiar with. This includes items such as Material Icons, floating action buttons, and paper regions (use of drop shadow) to indicate grouping.

One major part of the application is the necessity for separate profiles. This therefore becomes a major fixture in the UI, and it was placed aside as a permanent side bar. There is also a clear separation in the use of colour to denote the grouping of the profile selector buttons and the content.

Large tabs at the top and a clear set of symbols make for a recognisable experience offering some affordance to the user. The user will likely already be familiar with top nav bar paradigm, and will note highlight to denote an active tab.

The body of the profile takes centre stage and presents the data as clear as it can. Each input field, when blank, has it's purpose labeled and can be clearly seen. The whitespace around the edge also aides in to defining the region of the body of contents.

Colour is used sparingly in the application design in order to make use of it as a tool both for emphasising and denoting interactivity. For example the floating action button (FAB) sits above the plain of both the side bar and content, it's contrasting colour helps make this clear. It's a separate element and can be interacted with.

Drop shadows and transparency are also used in the design. They're effective at communicating depth and making the user aware of the currently active component. Drop shadows are used on both buttons visible in figure 6, these indicated the depth of the button and calls back to a somewhat more skeuomorphic design that has since fallen out of favour. This gives the user some orientation and reinforces that it is indeed a button. In figure 7 we see the date picker layered above the profile details. A transparency is used to show this layer and again makes the user focus on this component. The use of ok/cancel buttons reinforces the fact that this current tasks requires completion, and their bottom left hand positioning follows the left-to-right, top-to-bottom reading style english speaking users are familiar with.

The rational concerning the view screen is based on a data centric view. The main goal here is to present the data that is required in the simplest, clearest way. We don't want to hide information behind buttons and drop downs, and as such data is presented in clear style. This reduces the number of clicks needed to access information, making for a quicker more navigable experience.

To the right we see a list of the tweets gathered and separated by sentiment, this allows the user to access this information immediately. To represent the quantitative data we employ the use of a graph with contrasting separate colours to denote each section of the graph. A grid system is also used to clearly group each part of the UI.

Within the Alerts page we make use of a drop shadow again to show the panel on which the alerts sit. Each alert is broken up with a horizontal rule that allows a definition between each alert item. The use of a the rubbish bin icon is an example of affordance to let the user know that it's to delete the alert. The use of the colour here also carries with the theme of interactivity. The FAB to the bottom-right once again is within the group of the alerts panel; ensuring the user recognises the function of this button compared to the left most FAB.

3.3.5 Data Storage

Data shall be stored in a relational MySql/MariaDB Database. MariaDB is a fork of the MySql database backend and acts in the same way for our purposes. This decision was made as we already have a database server that runs this software and this therefore cuts down on overhead and cost for further data hosting.

We wish to store as little data as possible as there is a chance many megabytes or even some gigabytes of data could be stored should these profiles run for any extended period of time. It's also important to consider the nature of the data we are storing. The relational model of storing data allows for great optimisations against storing redundant data; this reduces both storage use and risk of inconsistent data.

There are three entities we wish to save, those are Tasks, Profiles, and Tweets. We can easily configure the cardinality and multiplicity of our relations. One Profile has none or many Tasks, one Profile also has none or Many Tweets. This, and the data that is saved about these objects, including their primary, composite, and foreign keys, can be seen in the following entity relationship diagram.

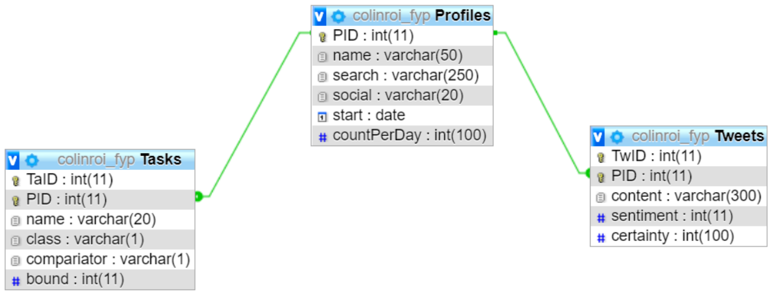


Figure 15: The Entity Relationship Diagram for the database in production as seen in the PHPMyAdmin window

3.3.6 Data processing

The following two sections will be an outline of the plan at this stage. Due to the agile nature of the project, this section will very likely change as the project progresses and the state of systems is analysed. It is also within the nature of machine learning that some experimentation will be necessary. Because of this it's hard to state unequivocally what the end product will look like at this stage. We can however look into how we will tackle the problem of optimising our algorithms.

We must acquire social media posts in two ways. Once as a labeled data set in the training phase, and again live during the running of the application. We shall first concern ourselves with the labeled data set. It is mostly trivial to access a labeled dataset. The two datasets that we will look at using for this project are the SemEval (SemEval, 2017) data sets and the NLTK (Bird et al., 2009) Twitter dataset. These data sets were both gathered around 2014 and 2015.

The Twitter API will certainly be used in order to pull data from Twitter in real-time. In order to gather data from a wide user sample, we will simply execute the data search through the twitter search API endpoint. This will return a sample of tweets that contain one or more of the search terms the user specifies. Once the data has been received it can be pre-processed.

Once the data has been collected it should be processed in such a way that it is normalised. In this case punctuation will be removed along with URLs. It may also be of interest to change any characters with repeating letters into a shortened version of this. For example we can convert "wayyyyy" into "wayy". Another option for normalising this textual data is a process known as stemming. This essential reverses verb conjugation, making "running" into "run".

In order for our mathematical based algorithms to understand our tweets they must be converted into some sort of numerical value set. This is the process of feature extraction. One such option as layed out in the Literature Review in section 1.7, the bag of words. Another option is to use a system we'll refer to as a meta vector. This is an input vector made up of values that represent characteristics about the tweet. These are things such as, number of positive and negative words, number of hashtags, number of punctuation marks, the number of likes and retweets, the total character count, and so on. Which method will work best remains to be seen during the implementation and experimentation phase.

3.3.7 Machine learning

The two methodologies that will be implemented are a neural network and a K-nearest neighbor classifier.

The neural network will be made up of of 3 to 4 layers. These will consist of one input and one output layer. There will be some amount of hidden layers in the middle; the number and shape of these layers still remains to be seen and will be evaluated during the implementation phase. At this time a small number of layers has been chosen, somewhat arbitrarily. This task should not require many more than 2 layers as it simply isn't that complex.

The input to the network will be dependent on the size of the vector the feature set extracting algorithm produces. The output however will be one node, and be activated with a sigmoid layer. The output values will range from 1 to -1 with 1 being positive sentiment and -1 being negative. Any values produced in the 0 range will be read as neutral in sentiment. The bounds for the classification is a not a simple rounding exercise.

In order to maintain an unbiased range of outputs, there should be an equal opportunity for all 3 sentiment classes to be chosen at random. If we simply round the output we have a strong bias towards neutral sentiment. Therefore we take the bounds of neutral between 0.3 and -0.3. This means of values between -1.0 and 1.0, to one decimal place, 7 are positive, 7 are neutral, and 7 are negative. This is just one way we can sway our machine learning algorithm against bias.

The other machine learning technique that will be employed is the K-nearest neighbor classifier. For this, the existing algorithm within the SciKitLearn Python library will suffice as we can set enough hyper-parameters for it to be useful for the testing phase, however we will implement our own KNN. We will use the Euclidean or the Manhattan distance as the metric for distance between our vectors as it is a simple and efficient algorithm to use and should not need to deal with high-dimensional space vectors, and a k of 5. We should not use a cosign distance algorithm as it is unable to deal well with negative numbers. Once again these hyper-parameters are subject to change during the development stage as some experimentation will be required. At this time k=5 is a generalised default value for these algorithms.

4 Implementation

It is now time to being programming. See full code on GitLab here;

https://gitlab.kingston.ac.uk/K1610762/FYPCode

While the implementation itself was achieved in Agile Sprints, items here are separated into their general product backlog category.

4.1 UI Structure and Setting the State

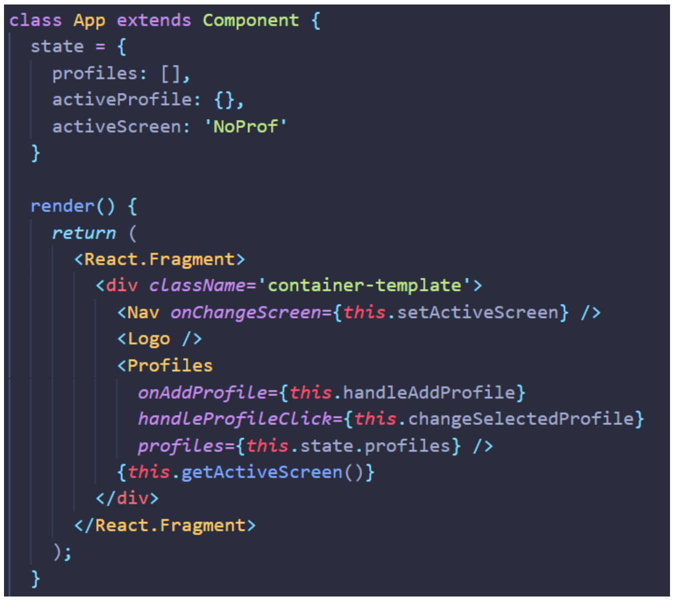
We can start by developing with React to create the user interface. React allows a data control model to sit external to the presentation code. As such data will be pulled in via the Python Flask API. At this time therefore we will design around default data.

React code is developed component by component. This means we can focus on developing individual and reusable parts of the UI. Because React is compiled server side within NodeJS, we have full access to NPM - Node Package Manager. As such we can use the Materializecss[2] library to aid in developing a Material

[2]https://materializecss.com/

Design style.

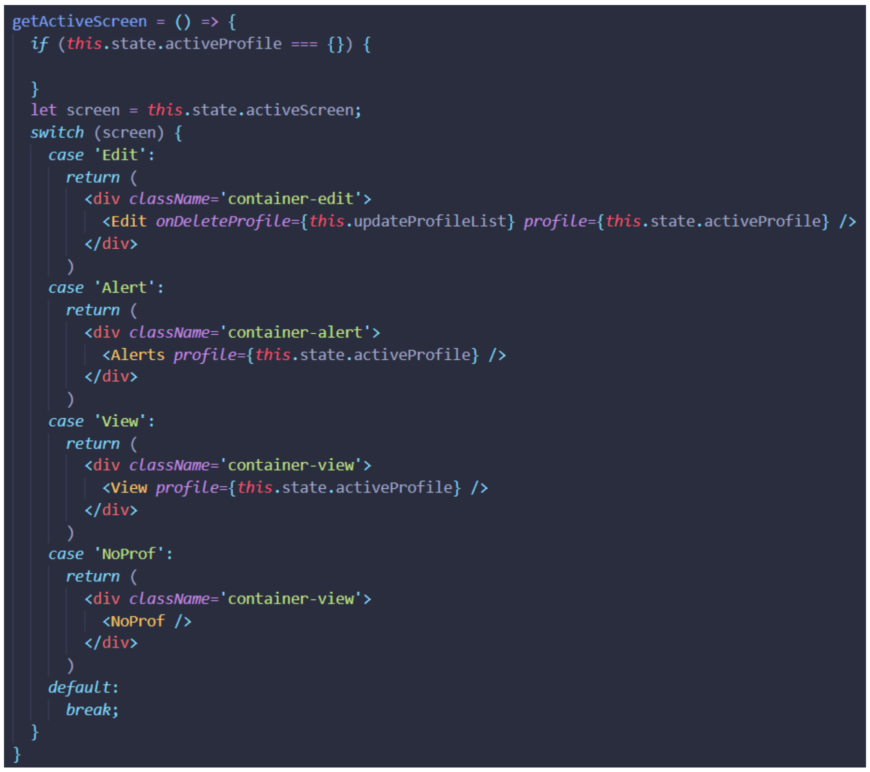
From there some simple data can be placed into the state object and it can then act as a functioning prototype.



```
class App extends Component {
  state = {
    profiles: [],
    activeProfile: {},
    activeScreen: 'NoProf'
  }

  render() {
    return (
      <React.Fragment>
        <div className='container-template'>
          <Nav onChangeScreen={this.setActiveScreen} />
          <Logo />
          <Profiles
            onAddProfile={this.handleAddProfile}
            handleProfileClick={this.changeSelectedProfile}
            profiles={this.state.profiles} />
          {this.getActiveScreen()}
        </div>
      </React.Fragment>
    );
  }
```

Figure 16: The basic component class for App, With some sate shown

When rendering the tab that should be open, we use this function *getActive-Screen()*.



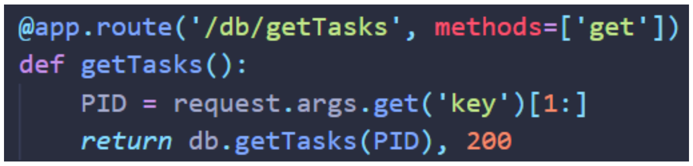
```
getActiveScreen = () => {
  if (this.state.activeProfile === {}) {

  }
  let screen = this.state.activeScreen;
  switch (screen) {
    case 'Edit':
      return (
        <div className='container-edit'>
          <Edit onDeleteProfile={this.updateProfileList} profile={this.state.activeProfile} />
        </div>
      )
    case 'Alert':
      return (
        <div className='container-alert'>
          <Alerts profile={this.state.activeProfile} />
        </div>
      )
    case 'View':
      return (
        <div className='container-view'>
          <View profile={this.state.activeProfile} />
        </div>
      )
    case 'NoProf':
      return (
        <div className='container-view'>
          <NoProf />
        </div>
      )
    default:
      break;
  }
}
```

Figure 17: Function used to render the appropriate component based on the active tab.

A simple case statement is used to find the active screen and then return, and in turn render, the active component. This design pattern of conditionally rendering components based on state, is used extensively within the React framework. This aids greatly in re-rending upon state change.

4.2 Flask API Endpoints

The core of any API is the multitude of endpoints that one creates. In the case of flask it is simple to set up an endpoint. Take for example the URL we call to get the tasks for a specific profile



```
@app.route('/db/getTasks', methods=['get'])
def getTasks():
    PID = request.args.get('key')[1:]
    return db.getTasks(PID), 200
```

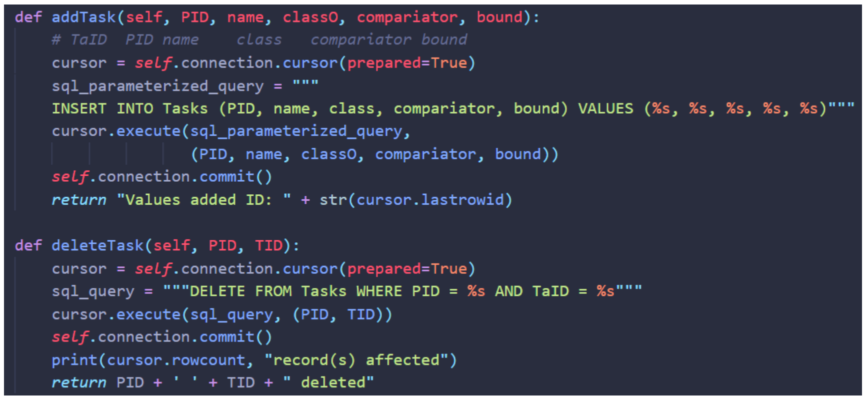
Figure 18: The Flask route for the get tasks endpoint within the API

We use the *app.route()* annotation to denote the URL that this function will run at. In this case when they visit the URL/db/getTasks. The function then gets the url parameters. In this case it is looking for key attribute which the frontend will send over in the form 'Psome Number'. This is because, unlike the backend, the frontend need not concern itself with the nuances of databases. Therefore for the backend to work with this value we must trip of the leading 'P'. Notice it is also referenced as a PID, this is database term and stands for the 'profile identification'.

To access the *getTasks()* function we simply call it via the DAO object and pass in the appropriate parameter. We then expect to return the result, as JSON (see next section), directly to the front end for use.

4.3 Flask API DAO

A simple data access object is used to gain access to the external database. The design pattern for inserting and deleting data can be seen bellow.



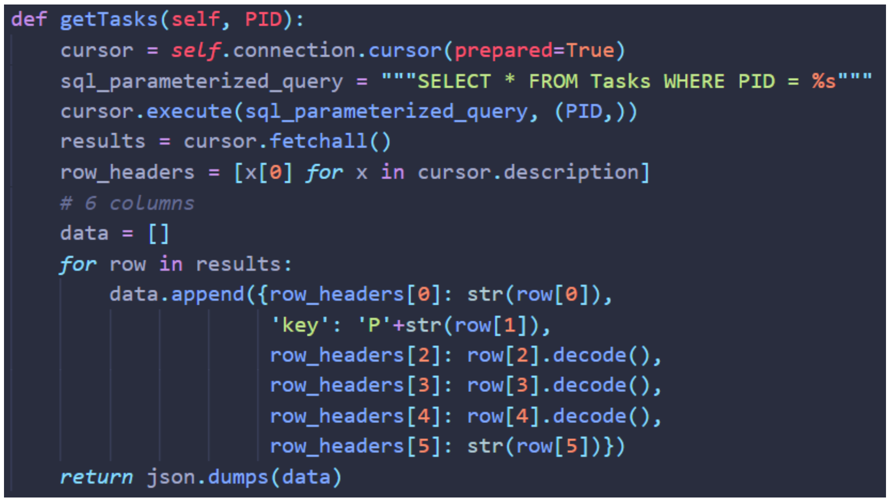
```
def addTask(self, PID, name, classO, compariator, bound):
    # TaID  PID name    class   compariator bound
    cursor = self.connection.cursor(prepared=True)
    sql_parameterized_query = """
    INSERT INTO Tasks (PID, name, class, compariator, bound) VALUES (%s, %s, %s, %s, %s)"""
    cursor.execute(sql_parameterized_query,
                   (PID, name, classO, compariator, bound))
    self.connection.commit()
    return "Values added ID: " + str(cursor.lastrowid)

def deleteTask(self, PID, TID):
    cursor = self.connection.cursor(prepared=True)
    sql_query = """DELETE FROM Tasks WHERE PID = %s AND TaID = %s"""
    cursor.execute(sql_query, (PID, TID))
    self.connection.commit()
    print(cursor.rowcount, "record(s) affected")
    return PID + ' ' + TID + " deleted"
```

Figure 19: Examples of add and delete methods, to add and delete a task

Each function is designed to perform a specific task, no queries are allowed to made from outside the DAO. Each query is carefully constructed from the parameters that are passed into it using prepared statements. This prevents against a lot of sql injection attacks be they deliberate or the product of misused data.

When data is modified the function will return the key of the affected item. Only one record will be affected at any one query.

When returning data we use a different set of rules.



```python
def getTasks(self, PID):
    cursor = self.connection.cursor(prepared=True)
    sql_parameterized_query = """SELECT * FROM Tasks WHERE PID = %s"""
    cursor.execute(sql_parameterized_query, (PID,))
    results = cursor.fetchall()
    row_headers = [x[0] for x in cursor.description]
    # 6 columns
    data = []
    for row in results:
        data.append({row_headers[0]: str(row[0]),
                     'key': 'P'+str(row[1]),
                     row_headers[2]: row[2].decode(),
                     row_headers[3]: row[3].decode(),
                     row_headers[4]: row[4].decode(),
                     row_headers[5]: str(row[5])})
    return json.dumps(data)
```

Figure 20: An example of a select query, to get information about a task

In this occasion do not consider individual bits of data for the most part. When the function will be being called from the front end we must make sure use server calls efficiently. Given this example of collecting tasks, we want to collect all the data being stored about this task as we will need it at some point if not immediately. This is because we have to make several communication requests from the user to the API and then to another server hosting the database. So where one would usually only select relevant columns, we simply acquire them all at once.

When sending the data back down from the API to the front end we construct a python array of objects. To make it simple to transfer and make use of the data on the Javascript side we convert this to a JSON string.

JSON has the wonderful advantage of being so ubiquitously adopted and simple that there are parsers for all commonly used programming languages. As such inter language data communication becomes incredible easy and convenient.

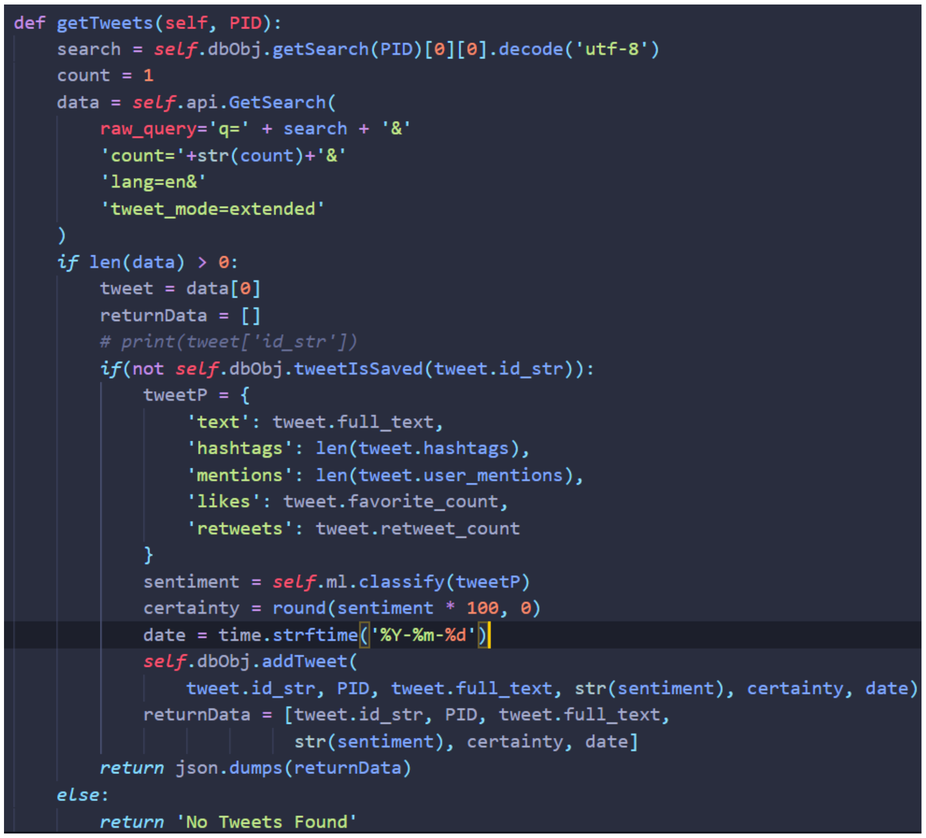
4.4 Flask API Social Class

The connection from this app to the social internet is the Twitter API. We maintain a simple function that is called from the Schedular class. This uses the twitter api python library and makes a search query with the terms from the databased pulled from the DAO.

This function runs as regularly in accordance with the users setting on the quantity of tweets collected per day.

The function will take in the query parameters and execute the query to find a single tweet. It then collates all the necessary data from the Twitter API response including, text, hashtag count, mention count, likes count, and retweet count.

This data is then passed through the machine learning algorithm.



```python
def getTweets(self, PID):
    search = self.dbObj.getSearch(PID)[0][0].decode('utf-8')
    count = 1
    data = self.api.GetSearch(
        raw_query='q=' + search + '&'
        'count='+str(count)+'&'
        'lang=en&'
        'tweet_mode=extended'
    )
    if len(data) > 0:
        tweet = data[0]
        returnData = []
        # print(tweet['id_str'])
        if(not self.dbObj.tweetIsSaved(tweet.id_str)):
            tweetP = {
                'text': tweet.full_text,
                'hashtags': len(tweet.hashtags),
                'mentions': len(tweet.user_mentions),
                'likes': tweet.favorite_count,
                'retweets': tweet.retweet_count
            }
            sentiment = self.ml.classify(tweetP)
            certainty = round(sentiment * 100, 0)
            date = time.strftime('%Y-%m-%d')
            self.dbObj.addTweet(
                tweet.id_str, PID, tweet.full_text, str(sentiment), certainty, date)
            returnData = [tweet.id_str, PID, tweet.full_text,
                          str(sentiment), certainty, date]
        return json.dumps(returnData)
    else:
        return 'No Tweets Found'
```

Figure 21: The *getTweets()* function within the social class in the API

Once the sentiment is received the certainty can be calculated. it simple computes the certainty based on the sentiment's distance from the true class value. The KNN however only produces 2 integers and as such will never have a certainty of less than 100%.

The result is stored as a Tweet entity in the database through the DAO.

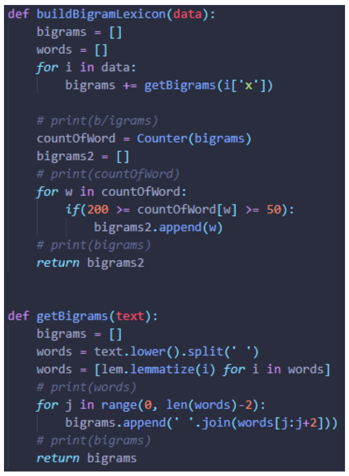
4.5 Flask API Machine Learning Class

4.5.1 Data pre-processing

Two methods were attempted to optimise the algorithms. In the first instance a Bag Of Words model was used both with unigrams and bigrams separately, with a threshold of frequency between 50 and 200 occurrences of a word in the corpus in order to limit rare words that are likely to carry no sentiment weight and common words that are too frequent to contribute to sentiment. Words were also stemmed in order to normalise the data and increase the weight of words regardless of tense.

Initially a lexicon was built of all the words the 'bag' should contain.

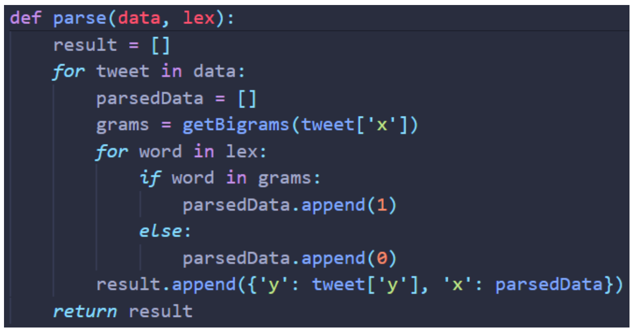
Then each document was converted into a vector based on which words from the lexicon it contains.



```python
def buildBigramLexicon(data):
    bigrams = []
    words = []
    for i in data:
        bigrams += getBigrams(i['x'])

    # print(b/igrams)
    countOfWord = Counter(bigrams)
    bigrams2 = []
    # print(countOfWord)
    for w in countOfWord:
        if(200 >= countOfWord[w] >= 50):
            bigrams2.append(w)
    # print(bigrams)
    return bigrams2


def getBigrams(text):
    bigrams = []
    words = text.lower().split(' ')
    words = [lem.lemmatize(i) for i in words]
    # print(words)
    for j in range(0, len(words)-2):
        bigrams.append(' '.join(words[j:j+2]))
    # print(bigrams)
    return bigrams
```

Figure 22: The *buildBigramLexicon()* function that constructs a lexicon of frequent bigrams within the corpus

The lexicon is then converted into a pickle using Python's Pickle library to save it and re-load it at a later date. Tweets can then be parsed through the following function in order to make them into usable vectors.



```python
def parse(data, lex):
    result = []
    for tweet in data:
        parsedData = []
        grams = getBigrams(tweet['x'])
        for word in lex:
            if word in grams:
                parsedData.append(1)
            else:
                parsedData.append(0)
        result.append({'y': tweet['y'], 'x': parsedData})
    return result
```

Figure 23: The algorithm that takes an array of tweets and turns them into vector representations

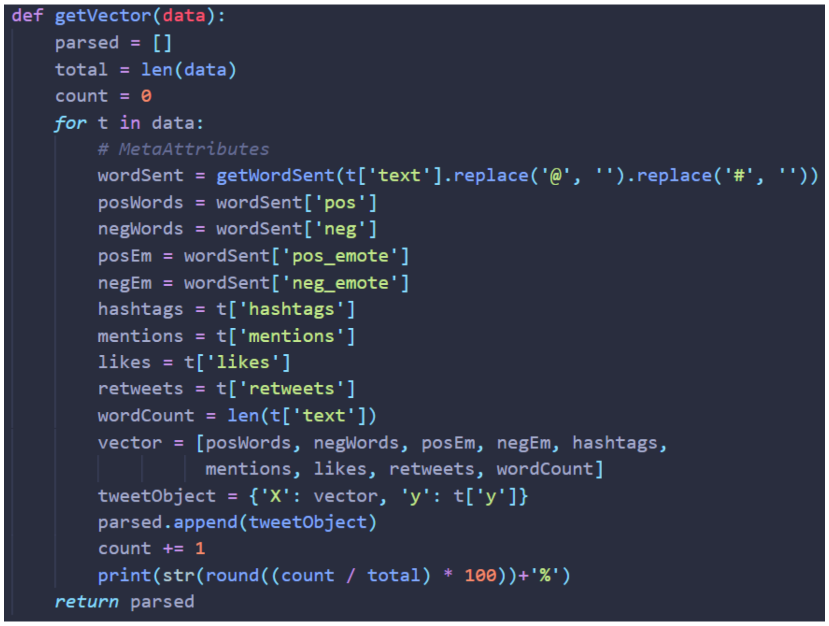
That is more of a passive method of feature extraction - we allow the machine learning algorithm to recognise the most important features. Unfortunately during all testing this failed to reach much further than just randomly guessing the sentiment. I hypothesise that this is down to the resolution of the method. Each vector can be several hundred features long, and due to the short nature of tweets they aren't going to contain many words that are also in our lexicon. A simple bag-of-words simply doesn't have the resolution to fit a tweet and as such, the error to adjust weights and the hyper-spacial distances between vectors simply isn't that great.

It was at this point that a more active method of feature extraction was implemented with a meta-vector was used.

The idea behind the meta vector is to take the tweet data and to extract features directly about the text but also about the tweet itself. We extract the following attributes:

- Number of positive words
- Number of negative words
- Number of positive emoticons :) :D :3 ;) ;3 :-) :-D
- Number of negative emoticons :/ :0 :o :O :(:-(
- Number of user mentions
- Number of likes
- Number of hashtags
- Number of characters used

We end up with 9 features in a vector that show certain aspects of the tweet. We maintain the classing of positive and negative words using a classification from the SentiWords dataset (Gatti et al., 2016) of over 100,000 positive and negative words. However we also include extra details about emoticons and how the tweet was received by other users. This will also help to see if there is a correlation between the length of a tweet and its likely sentiment.



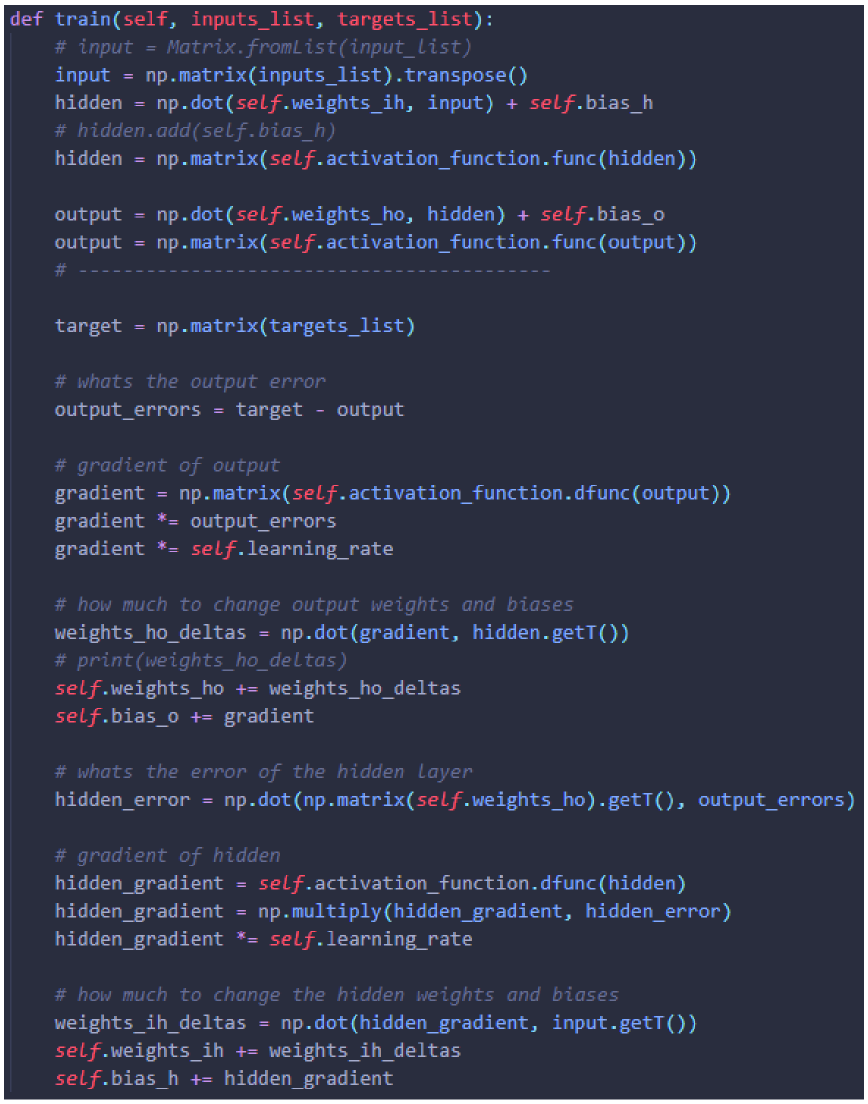
```python
def getVector(data):
    parsed = []
    total = len(data)
    count = 0
    for t in data:
        # MetaAttributes
        wordSent = getWordSent(t['text'].replace('@', '').replace('#', ''))
        posWords = wordSent['pos']
        negWords = wordSent['neg']
        posEm = wordSent['pos_emote']
        negEm = wordSent['neg_emote']
        hashtags = t['hashtags']
        mentions = t['mentions']
        likes = t['likes']
        retweets = t['retweets']
        wordCount = len(t['text'])
        vector = [posWords, negWords, posEm, negEm, hashtags,
                  mentions, likes, retweets, wordCount]
        tweetObject = {'X': vector, 'y': t['y']}
        parsed.append(tweetObject)
        count += 1
        print(str(round((count / total) * 100))+'%')
    return parsed
```

Figure 24: The function to convert a tweet and its meta data into a one-dimensional vector

4.5.2 Neural Network

The development process followed a somewhat unconventional route. Three separate networks were built at 3 different levels of abstraction, in order to test various levels of suitability.

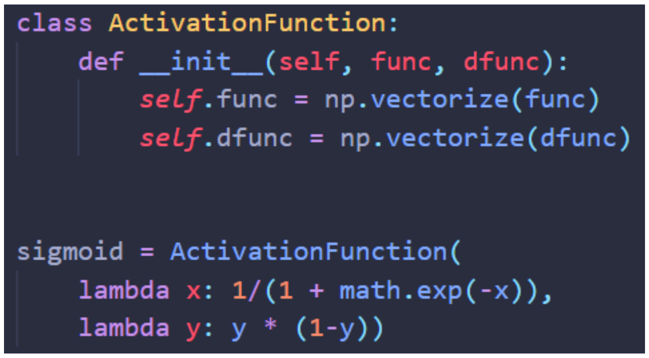
From scratch with NumPy - In the first instance a simple network with feedforward, backpropagation, and one hidden layer was created based off the work within Rashid (2016) using only the NumPy (Oliphant, 2006) library for Python. This was validated by solving a simply non-linear problem such as XOR classification.



```python
def train(self, inputs_list, targets_list):
    # input = Matrix.fromList(input_list)
    input = np.matrix(inputs_list).transpose()
    hidden = np.dot(self.weights_ih, input) + self.bias_h
    # hidden.add(self.bias_h)
    hidden = np.matrix(self.activation_function.func(hidden))

    output = np.dot(self.weights_ho, hidden) + self.bias_o
    output = np.matrix(self.activation_function.func(output))
    # -----------------------------------------

    target = np.matrix(targets_list)

    # whats the output error
    output_errors = target - output

    # gradient of output
    gradient = np.matrix(self.activation_function.dfunc(output))
    gradient *= output_errors
    gradient *= self.learning_rate

    # how much to change output weights and biases
    weights_ho_deltas = np.dot(gradient, hidden.getT())
    # print(weights_ho_deltas)
    self.weights_ho += weights_ho_deltas
    self.bias_o += gradient

    # whats the error of the hidden layer
    hidden_error = np.dot(np.matrix(self.weights_ho).getT(), output_errors)

    # gradient of hidden
    hidden_gradient = self.activation_function.dfunc(hidden)
    hidden_gradient = np.multiply(hidden_gradient, hidden_error)
    hidden_gradient *= self.learning_rate

    # how much to change the hidden weights and biases
    weights_ih_deltas = np.dot(hidden_gradient, input.getT())
    self.weights_ih += weights_ih_deltas
    self.bias_h += hidden_gradient
```

Figure 25: The backpropogation algorithm from the ‘from scratch’ version of the neural network

We start with constructing the feed forward process as we need to get the outputs of both the hidden layer and the output layer. With each layer we computer the product of the input matrix and the weights and add the bias element-wise. We then use the activation function to process the data. As the network was written as a general purpose class, each activation function can be written as a class that will interface with the network. They are made up of two parts, the function as a lambda function and the derivative of the function as a lambda function. It is a very compact solution to making ones own activation function that can be set within the neural network class.



```
class ActivationFunction:
    def __init__(self, func, dfunc):
        self.func = np.vectorize(func)
        self.dfunc = np.vectorize(dfunc)


sigmoid = ActivationFunction(
    lambda x: 1/(1 + math.exp(-x)),
    lambda y: y * (1-y))
```

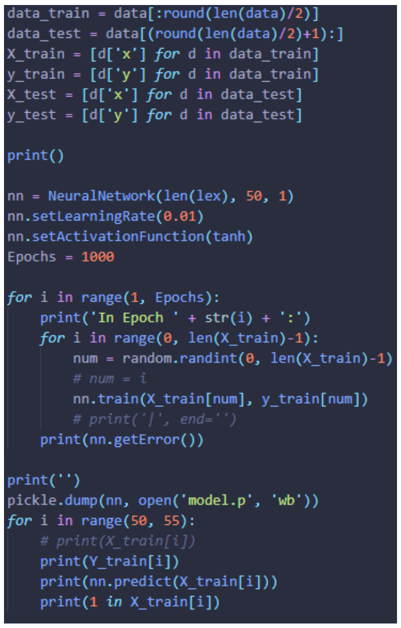
Figure 26: The activation function class and the sigmoid implementation of that. Note: the derivative is not the true derivative but works as the data is already processed through the root function

It's then time to being the actual process of backpropagation. We start by calculating the output errors which we can use to work out how to adjust the weights and biases of the output layer.

We need to calculate the gradient by passing the output through the derivative of the activation function we used and then multiplying this element-wise with the error. This gives us the value for weight that is inline with the effect that each weight has on the output value. These values are then multiplied element-wise once again, this time to the learning rate. We can then work out how far we wish to adjust the weights, we times them by the gradient we have calculated to get these deltas. The weights can then be adjusted by them. The bias can be adjust but the value of the gradient itself. This is the process of backpropagation for the output layer.

We now move to the hidden layer and we do exactly the same thing except to calculate the error we can not simply use target less the output as we do not know what the target should be. Therefore we use the output errors and find the matrix product with the weights of the hidden output. Calculating the gradient and deltas now follows the same pattern as before.

Running this function on all the data once is a single Epoch. To implement this class we use the following code.



```python
data_train = data[:round(len(data)/2)]
data_test = data[(round(len(data)/2)+1):]
X_train = [d['x'] for d in data_train]
y_train = [d['y'] for d in data_train]
X_test = [d['x'] for d in data_test]
y_test = [d['y'] for d in data_test]

print()

nn = NeuralNetwork(len(lex), 50, 1)
nn.setLearningRate(0.01)
nn.setActivationFunction(tanh)
Epochs = 1000

for i in range(1, Epochs):
    print('In Epoch ' + str(i) + ':')
    for i in range(0, len(X_train)-1):
        num = random.randint(0, len(X_train)-1)
        # num = i
        nn.train(X_train[num], y_train[num])
        # print('|', end='')
    print(nn.getError())

print('')
pickle.dump(nn, open('model.p', 'wb'))
for i in range(50, 55):
    # print(X_train[i])
    print(Y_train[i])
    print(nn.predict(X_train[i]))
    print(1 in X_train[i])
```

Figure 27: The training implementation for the previously described neural network class

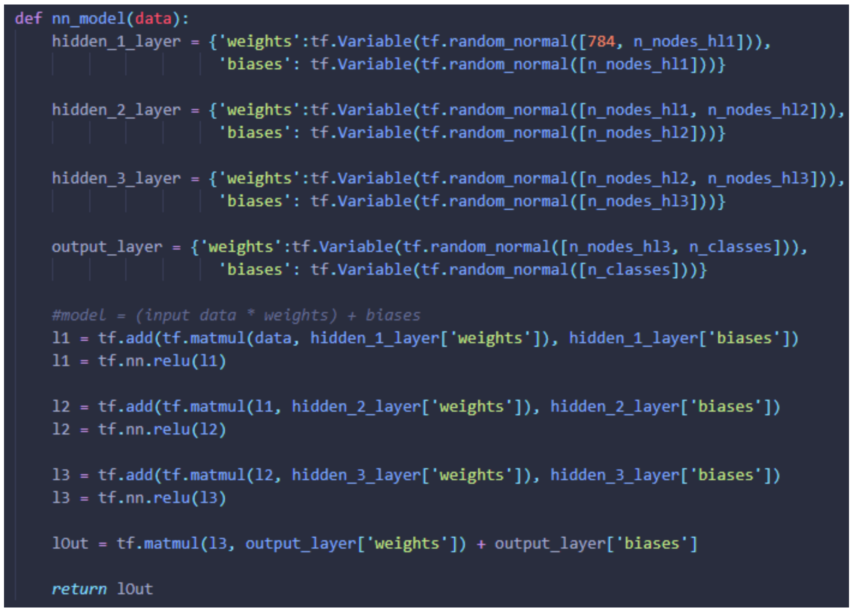
When training this network with the bag of words method the the data was broken into two sets, the training and testing data. This was broken into an array of input vectors and an array of output vectors.

Then going through each epoch a random vector was chosen to be trained against. This was repeated n number of times. n being the size of the training corpus. A testing set was then run through the prediction algorithm and compared to actual value.

Once the model has been trained the Python pickle module was used to export the model as a pickle file. This will save all the attributes and functions of an object, that be used later for deployment.

With Tensorflow - From there the Tensorflow library was taken advantage of for optimisation techniques such as GPU acceleration, and a simple 4 layer network was built within the Tensorflow API to solve the MNIST dataset (Lecun et al., 1998) handwriting classification problem a a proof of concept task.

The model is defined in terms of individual objects that hold the variable tensors that hold weights and biases, each being highly customisable. The input dimension to the network is 784 as each MNIST sample is a 24x24 bitmap image, this is flattened to a 784 feature vector.

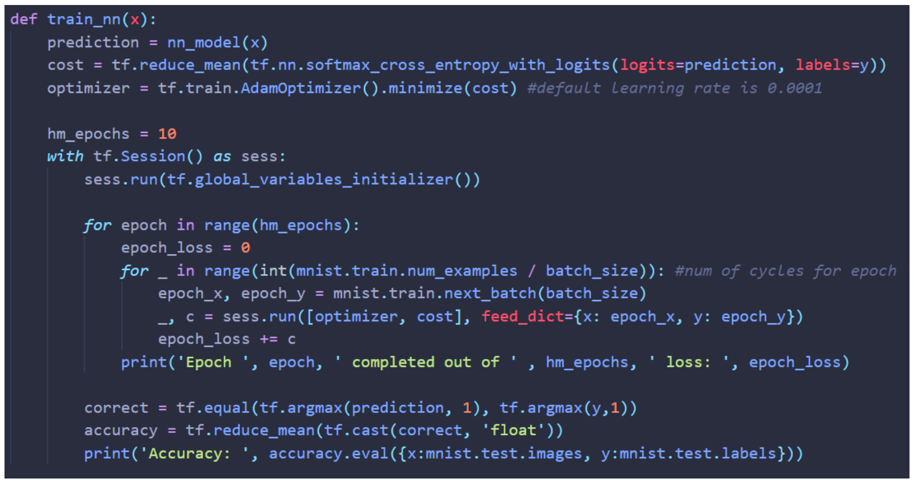


```
def nn_model(data):
    hidden_1_layer = {'weights':tf.Variable(tf.random_normal([784, n_nodes_hl1])),
                      'biases': tf.Variable(tf.random_normal([n_nodes_hl1]))}

    hidden_2_layer = {'weights':tf.Variable(tf.random_normal([n_nodes_hl1, n_nodes_hl2])),
                      'biases': tf.Variable(tf.random_normal([n_nodes_hl2]))}

    hidden_3_layer = {'weights':tf.Variable(tf.random_normal([n_nodes_hl2, n_nodes_hl3])),
                      'biases': tf.Variable(tf.random_normal([n_nodes_hl3]))}

    output_layer = {'weights':tf.Variable(tf.random_normal([n_nodes_hl3, n_classes])),
                      'biases': tf.Variable(tf.random_normal([n_classes]))}

    #model = (input data * weights) + biases
    l1 = tf.add(tf.matmul(data, hidden_1_layer['weights']), hidden_1_layer['biases'])
    l1 = tf.nn.relu(l1)

    l2 = tf.add(tf.matmul(l1, hidden_2_layer['weights']), hidden_2_layer['biases'])
    l2 = tf.nn.relu(l2)

    l3 = tf.add(tf.matmul(l2, hidden_3_layer['weights']), hidden_3_layer['biases'])
    l3 = tf.nn.relu(l3)

    lOut = tf.matmul(l3, output_layer['weights']) + output_layer['biases']

    return lOut
```

Figure 28: The model that will be the Tensorflow driven network

Once the variables are defined the connections are made with the simple matrix product and addition of the biases at each layer. This is then run through the activation function, in this case the Relu algorithm.

Using *tf.add()* we construct a model, this makes the training step much simpler.



```
def train_nn(x):
    prediction = nn_model(x)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=y))
    optimizer = tf.train.AdamOptimizer().minimize(cost) #default learning rate is 0.0001

    hm_epochs = 10
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        for epoch in range(hm_epochs):
            epoch_loss = 0
            for _ in range(int(mnist.train.num_examples / batch_size)): #num of cycles for epoch
                epoch_x, epoch_y = mnist.train.next_batch(batch_size)
                _, c = sess.run([optimizer, cost], feed_dict={x: epoch_x, y: epoch_y})
                epoch_loss += c
            print('Epoch ', epoch, ' completed out of ' , hm_epochs, ' loss: ', epoch_loss)

        correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y,1))
        accuracy = tf.reduce_mean(tf.cast(correct, 'float'))
        print('Accuracy: ', accuracy.eval({x:mnist.test.images, y:mnist.test.labels}))
```

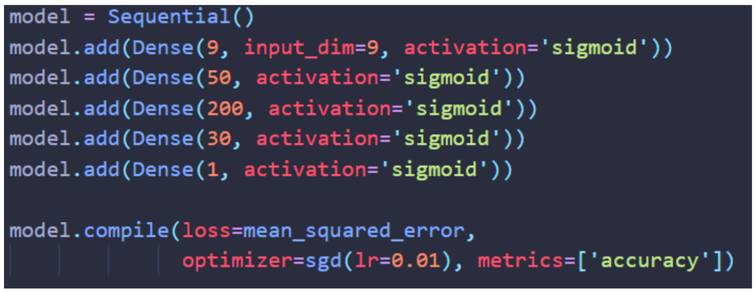
Figure 29: The training algorithm using the Tensorflow API

During training we specify the loss/cost function and the optimiser we used. The ‘from scratch’ version used stochastic gradient decent in this case the Adam optimiser was chosen. The loss function in this case was a variation on softmax. This is appropriate for the MNIST set as the labels are in the form of one-hot vectors. This means softmax will produce a set of probabilities for each label and we can use *argmax* to select the label that has the highest probability.

The epoch is run and batches of the dataset are sent into the session.The total loss is computed at the end of each epoch. After the 10 epochs run we can calculate the total accuracy.

This algorithm did quite well to label the MNIST dataset of handwritten digits but before we took it any further, we wished to see how Keras would perform.

With Keras - From there the use of the Keras (Chollet et al., 2015) library was implemented. Keras is an abstraction of the Tensorflow API and very good at building sequential models. We began running the MNIST set through a model with keras and it did very well, therefore we went on to build a model to run our dataset.

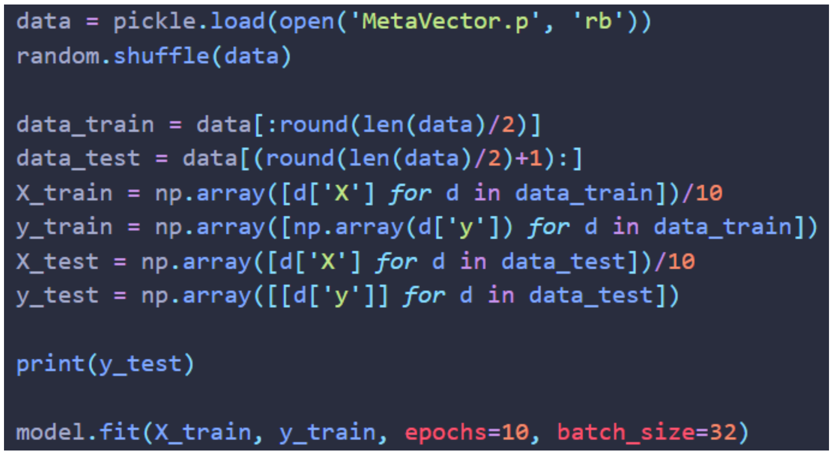


```
model = Sequential()
model.add(Dense(9, input_dim=9, activation='sigmoid'))
model.add(Dense(50, activation='sigmoid'))
model.add(Dense(200, activation='sigmoid'))
model.add(Dense(30, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss=mean_squared_error,
              optimizer=sgd(lr=0.01), metrics=['accuracy'])
```

Figure 30: The model build in Keras to handle meta vector based sentiment analysis

It's a network with four layers each with the sigmoid activation function. The loss function chosen is the mean squared error. Once again this network will be optimised with stochastic gradient decent.

Then we train the model on the data.



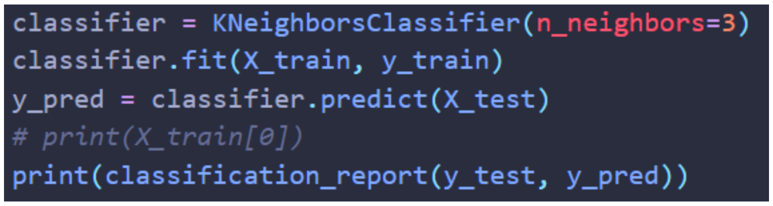
```
data = pickle.load(open('MetaVector.p', 'rb'))
random.shuffle(data)

data_train = data[:round(len(data)/2)]
data_test = data[(round(len(data)/2)+1):]
X_train = np.array([d['X'] for d in data_train])/10
y_train = np.array([np.array(d['y']) for d in data_train])
X_test = np.array([d['X'] for d in data_test])/10
y_test = np.array([[d['y']] for d in data_test])

print(y_test)

model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Figure 31: The training implementation for the Keras model

Unfortunately after much experimenting with hyperparameters, the model could at times only just scrape higher than random guessing. Sentiment analysis does not seem to be optimally solved with a standard neural network. We will move onto another method.

4.5.3 KNN Classifer

Implementing a KNN with a ready made model is a somewhat trivial task. Never the less it was useful to see how we could use a KNN and the meta-vector style feature sets to perform sentiment analysis.



```
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
# print(X_train[0])
print(classification_report(y_test, y_pred))
```

Figure 32: Fitting and predicting off of the SciKitLearn K-nearest neighbor model

In just four lines of code we can fit and predict how the KNN will react. Indeed it did very well. Note in the code that *n_neighbors* (this is the k value) is set ot 3. This is because it was found that 3 performed just as well as 4 or 5 neighbors, and reducing this number as much as one can will improve the speed of the classifier.

	precision	recall	f1-score	support
-1	0.90	0.93	0.91	2519
1	0.92	0.90	9.91	2480
average	0.91	0.91	0.91	4999

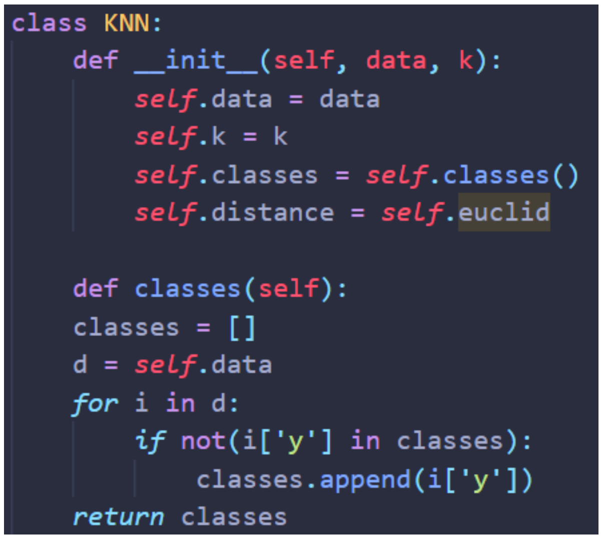
Table 1: F1 score and results for K = 5

	precision	recall	f1-score	support
-1	0.90	0.93	0.91	2519
1	0.92	0.90	9.91	2480
average	0.91	0.91	0.91	4999

Table 2: F1 score and results for K = 3

While these are good results, this is ideal circumstances with test data. We should expect a significant drop in quality during live classification.

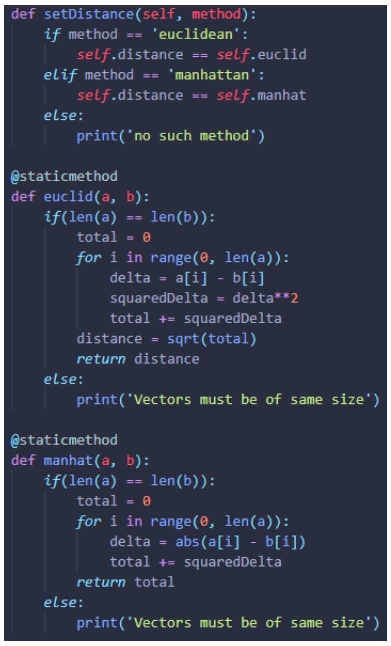
Now we know that this is a viable solution for this problem we can begin to implement our own algorithm. It is not a complicated algorithm to produce and we have a couple options at our disposal for building it. The KNN class is as follows.



```python
class KNN:
    def __init__(self, data, k):
        self.data = data
        self.k = k
        self.classes = self.classes()
        self.distance = self.euclid

    def classes(self):
    classes = []
    d = self.data
    for i in d:
        if not(i['y'] in classes):
            classes.append(i['y'])
    return classes
```

Figure 33: The constructor for the KNN class

This initialises the list of classes that the KNN will be trained on and the default distance algorithm, which is the Euclidean distance. No difference in performance was found between Euclidean and Manhattan distance but the two can be used interchangeably with the following code.



```python
def setDistance(self, method):
    if method == 'euclidean':
        self.distance == self.euclid
    elif method == 'manhattan':
        self.distance == self.manhat
    else:
        print('no such method')

@staticmethod
def euclid(a, b):
    if(len(a) == len(b)):
        total = 0
        for i in range(0, len(a)):
            delta = a[i] - b[i]
            squaredDelta = delta**2
            total += squaredDelta
        distance = sqrt(total)
        return distance
    else:
        print('Vectors must be of same size')

@staticmethod
def manhat(a, b):
    if(len(a) == len(b)):
        total = 0
        for i in range(0, len(a)):
            delta = abs(a[i] - b[i])
            total += squaredDelta
        return total
    else:
        print('Vectors must be of same size')
```

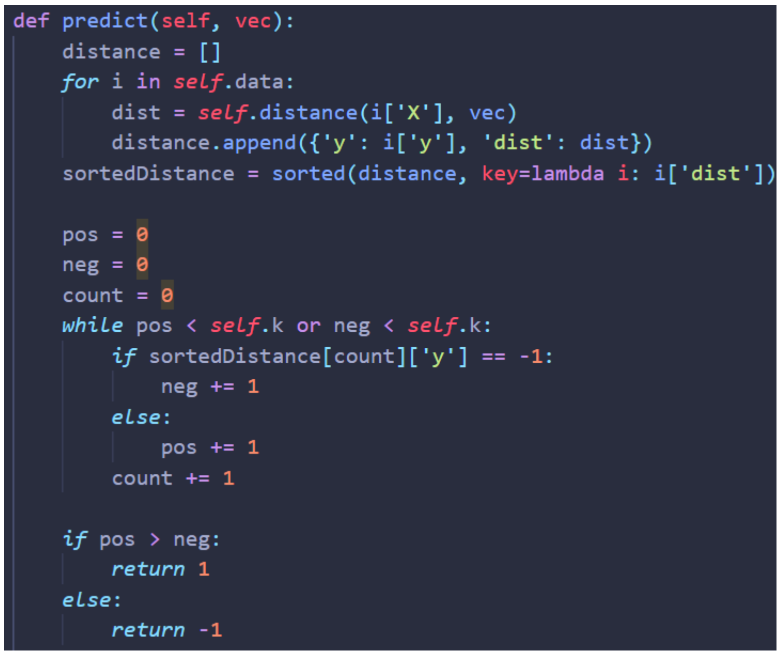
Figure 34: The two distance algorithms implemented and their setter function

The two distance algorithms are mathematical quite simple and can be described as follows;

$$ED(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \qquad (12) \qquad\qquad MD(a,b) = \sum_{i=1}^{n}(a_i - b_i) \qquad (13)$$

Figure 35: Euclidean distance formula left, Manhattan distance formula right

The training phase is now already complete, it's simply loading the labeled data, which is done with the user of a constructor parameter. So we must now look at the prediction algorithm.



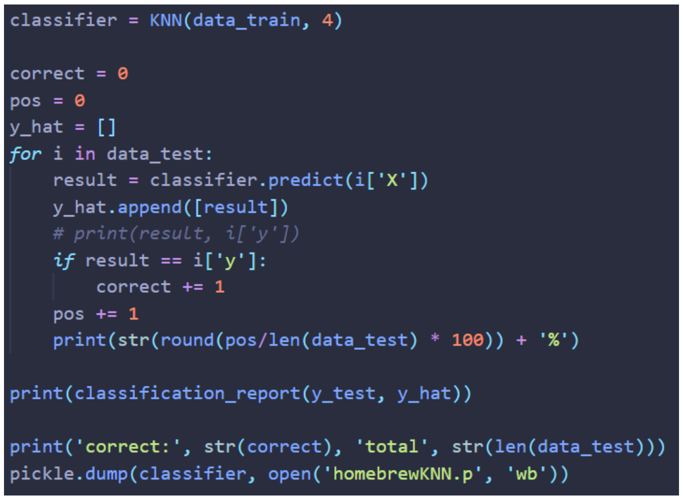
```python
def predict(self, vec):
    distance = []
    for i in self.data:
        dist = self.distance(i['X'], vec)
        distance.append({'y': i['y'], 'dist': dist})
    sortedDistance = sorted(distance, key=lambda i: i['dist'])

    pos = 0
    neg = 0
    count = 0
    while pos < self.k or neg < self.k:
        if sortedDistance[count]['y'] == -1:
            neg += 1
        else:
            pos += 1
        count += 1

    if pos > neg:
        return 1
    else:
        return -1
```

Figure 36: The prediction algorithm within the KNN class

We first take in the vector we wish to predict the label of. We then loop through our data and calculate the distance between our new vector and the existing data set vectors. These are saved as a dictionary with their label and distance.

We then make use of Python's sorting algorithm to sort them by distance with the smallest distances at the front. We can now find our new data's closest neighbors. We loop through all of these new distances, counting up the classes we find. Which ever class reaches our k value first is the label we return as the predicted sentiment.

The implementation of this class looks as follows.



```
classifier = KNN(data_train, 4)

correct = 0
pos = 0
y_hat = []
for i in data_test:
    result = classifier.predict(i['X'])
    y_hat.append([result])
    # print(result, i['y'])
    if result == i['y']:
        correct += 1
    pos += 1
    print(str(round(pos/len(data_test) * 100)) + '%')

print(classification_report(y_test, y_hat))

print('correct:', str(correct), 'total', str(len(data_test)))
pickle.dump(classifier, open('homebrewKNN.p', 'wb'))
```

Figure 37: The use case of the KNN class once training and testing data is loaded

This code is really only to evaluate the performance of the classifier. We simply produce a set of predicted values by looping through our test data and predicting the label. Once we have an array of predicted values we can use the convenient SciKitLearn function *classification_report()* which will generate an F1 score for us. Finally we create a pickle of the model so that it can be used later on.

That score is as follows:

	precision	recall	f1-score	support
-1	0.90	0.93	0.91	2519
1	0.92	0.90	9.91	2480
average	0.91	0.91	0.91	4999

Table 3: F1 score for the ‘from scratch’ KNN classifier

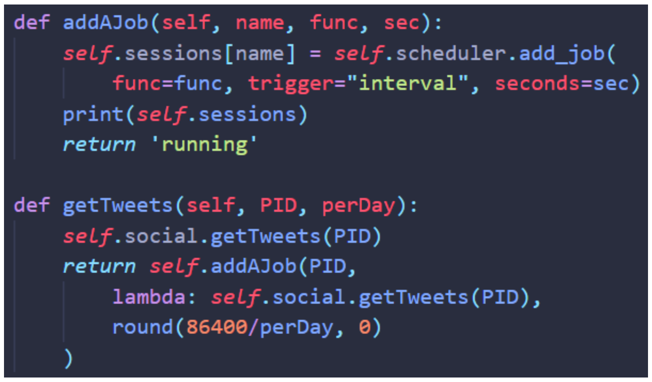
As you can see it’s done very well and as such will be the model pushed to the application.

4.6 Flask API Scheduling Class

They key feature of the app is to regularly collect tweets. In order to do this we divide the number of seconds in a day by the number of tweets gathered per day within a profile. For example say this was 50 tweets per day;

$$\frac{86400}{50} = 1728 \text{ seconds} \tag{14}$$

So every 1728 seconds we run the function to get a fresh tweet, if one is available.



```python
def addAJob(self, name, func, sec):
    self.sessions[name] = self.scheduler.add_job(
        func=func, trigger="interval", seconds=sec)
    print(self.sessions)
    return 'running'

def getTweets(self, PID, perDay):
    self.social.getTweets(PID)
    return self.addAJob(PID,
        lambda: self.social.getTweets(PID),
        round(86400/perDay, 0)
    )
```

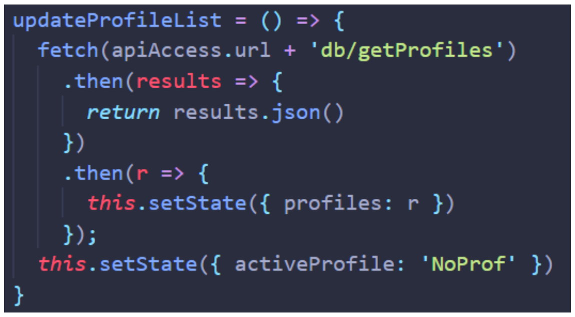
Figure 38: Function to add a job and its implementation for getting tweets regularly

The generic *addAJob()* function will take any function reference, name, and interval in seconds, and use the APSchedular for Python in order to call a function.

In the *getTweets()* function the one parameter for the social function and another parameter for the number of tweets to pull per day. We use the simply calculation shown above to work out how many seconds to wait between function calls. Into the *addAJob()* function we pass a lambda function that holds a call to the social class. So once this function is run we essentially add a regularly occurring call to our system that collections, classifies, and saves a tweet.

4.7 React-Flask Data connection

Once the back-end API was complete it was easy to connect the temporary functions within the presentation code, such as *getProfiles()*. Where these previously returned hard coded data, API calls were implemented with the fetch function within Javascript. This function is, as is the key feature of Javascript, asynchronous. This means that it will run this function but before its finished it will move onto the next line of code.



```
updateProfileList = () => {
  fetch(apiAccess.url + 'db/getProfiles')
    .then(results => {
      return results.json()
    })
    .then(r => {
      this.setState({ profiles: r })
    });
  this.setState({ activeProfile: 'NoProf' })
}
```

Figure 39: The function that makes the call to get the list of profiles from the database

The fetch function sets up what is known as a promise in JS. We can use the function *then()* with a callback function to tell the program what to do once that is finished; these can be chained together as shown. This is very necessary for dealing with async functions as without the use of then, the code would try to access the result variable but find that it was blank, as the server hasn't had a chance to respond yet and the fetch function is still running.

Within the callback function we convert the response from JSON into a JS object and then simply update the components state. Things that are not dependent on the function being complete can stay outside of the then clauses, this will speed up the running of the code as JS can now run these functions in parallel. With the example function however this will not result in much if any visible improvement due to the small amount of external code to the then callback function.

Notice that the *apiAccess* object contains the URL so that if this changes it can be updated in one file and all front end code will reflect that change.

5 Testing and Evaluation

5.1 Functional Requirements

We can now begin to look into how well the end product achieved the functional requirements. We shall break down each functional requirement and discuss whether or not it was achieved and if so how can we test this. Unless otherwise stated the expected outcome occurred when tested, relevant functional requirement numbers are shown in brackets. 1.1/2 references both 1.1 and 1.2.

5.1.1 Create a sentiment profile

Testing this is very simple. All we must do is select the add profile floating action button in the bottom left of the screen and enter a name for our profile (1.1). When we click 'OK' our profile will appear in the left hand menu. We can select this profile to see that there is indeed our title at the top. In order to test if the profile is saved we can close the tab and reopen it, or refresh without cache to see our profile remains (1.2).

5.1.2 Set some search parameters

To test 2.1 we can input some phrases testing the parameters to this. We should see that multiple words can be typed in, space separated, and it should treat this as one object, or material chip, and only the enter key will enter this chip and start a new one.

We should then be able to select a social media platform (2.2). While a drop down menu, and clicking on this reveals it is present that does allow the selection of a platform, at this time the project only support Twitter (2.2.1). There is scope for this to change in the future and the system, given a few alterations, would handle the addition of a new social media platform.

It is currently possible to back-date the search from date (2.3) in the range of seven days however it is possible to pick outside of the seven day range (2.3.2). The format produced will match that of (2.3.3) mmm/dd/yyyy. For example, you will see 'Apr, 15, 2019'.

For selecting the number of tweets to gather per day, the user makes use of a slider (2.4). When taken out to it's extremes the value will max out at 100 and bottom out at 1 (2.4.1). This is inline with the specified requirement.

5.1.3 See sentiment report

In order to begin the gathering of sentiment data we must click the the 'Run profile' button. We can then click on the middle of the three tabs to view the sentiment report page, after the program has been running for some time. We see that the page has now been populated with data. If we draw our attention over to the right hand side we can see that it is indeed pulling posts in from social media (3.1). If we look towards the graph on the left hand side we will see the appropriate number of entries as specified in the search parameters page (3.1.2). There is no real way to test 3.1.1 without some code, this is where the evaluation stage would benefit from some unit tests being written. We can also see that the tweet have been assigned sentiment (3.1.3) as it has been categorised in both the right hand side (3.2) and the wherever it may appear on the graph.

We can see that the sentiment has been tracked with each point on the graph representing a tweet, hovering on these points will display the body of that tweet (3.3, 3.3.1).

A bar chart to the bottom can be seen, this shows the percentage negative and the percentage positive of all the tweets that have come in (3.4). The final implementation is using a KNN classifier which means we don't strictly get confidence scores out of the algorithm. This means that we cannot separate the stacks into their confidence of their classification (3.4.1). This isn't a big problem though, should the algorithm be changed to another system such as a more complex and robust neural network, confidence scores could be extracted. For the time being however all results are marked as 100%, this is acceptable for this use case.

Let's look at how the classifier performs in real life. We will take 10 randomly selected Tweets, five positive, five negative, from the database and also pull the sentiment. Three individuals will also be asked to classify the tweet manually. We can then compare that to the machines results.

Tweet	H1	H2	H3
But did NASA's Twins Study analyze how playing Kerbal Space Program changes when done in zero gravity?	1	1	1
NASA spots mysterious galactic 'jellyfish' in space via the @FoxNews App	1	1	1
@BrexitBattalion @Conservatives I cannot/ will not even support Labour given their Brexit stance, so it looks like it's going to be an Independent/new party.	0	0	0
@OFOCBrexit When will Remain campaigns/ campaigners stop this kind of misreporting? PLEASE stop attacking 'Labour', and turn your attention to the PERSON who's refusing to back a #PeoplesVote: @jeremycorbyn! What's your core objective? To stop Brexit? Or to have a pop at Labour?	0	0	0
Tony Bennett is gutless. Don't make visit political.	0	0	0

Table 4: Tweets classified as positive

Tweet	H1	H2	H3
Former exec at International Space Station science lab indicted for allegedly 'expensing' prostitutes: report @ISS_CASIS @ISS_Research #ISS @NASA	0	0	0
WOW, we reversed global warming already ???? @realDonaldTrump got us out of the Paris deal just in time, that was close.	0	0	0
@Electroversenet: Here's yet another mainstream article linking a Grand Solar Minimum, increasing Cosmic Rays and a prolonged period of...	1	0	1
We're fine-tuning plans for the longer missions of @Astro_Christina & @AstroDrewMorgan. Researchers will use these extended expeditions to collect observations of the effects of long-duration spaceflight on both as part of their time on @Space_Station:	1	1	1
RT @HillReporter: Kellyanne Conway LOVES to talk about all of the documentation the White House provided to the Mueller Investigation while...	1	0	0

Table 5: Tweets classified as negative

While it is clear that there is some way to go in the development of the methods here, it's clear there is some promise and this acts a clear proof-of-concept.

5.1.4 Set alert

At this time it's simple to add an alert, you click the add alert floating action button (4.1). You are then guided through a series of popups to add in a name (4.1.1), the sentiment to look for in words either positive or negative (4.1.2, 4.1.2.1/2), less than or more than (4.1.3, 4.1.3.1/2), and the value of bound to look for (4.1.4). At this time there are no restrains on adding in these values outside of the database, therefore no errors are given and the values are not strictly limited. They should be set such that only a very specific amount of values can be entered, and this should be checked client-side.

Deleting an alert is again simple, clicking on the rubbish bin icon to the left of an alert will delete it (4.2).

At this time no alerts are actively listened for. While the infrastructure is there, some work still needs to be done to both regularly check for the alert and to send out an email confirmation of this (4.3, 4.3.1).

5.2 Usability testing

5.2.1 Introduction

User Experience is increasingly important and the best way to understand the current UX of a product is to conduct usability testing. A usability test is a series of steps written out that a moderator gets multiple users to complete one at a time, and in isolation. They are designed to get the cleanest results out of the user, who is encouraged to think out loud and voice opinions as they go.

5.2.2 Tasks

The usability tests were composed of the following tasks.

- Create a profile called 'test'
- Set some search terms for that profile
- Select the Twitter social media site
- Select the date, 3 days before the current day
- Set the number of tweets to pull to around 95

- Save and run the profile
- Find a classified tweet
- Pick a tweet on the graph and see what the contents of the tweet is
- Set up an alert with the following parameters and any name: Positive ¡ 60%
- Delete the profile

5.2.3 Results and Discussion

Of the 5 users tested all 5 of the the users completed all the tasks. There were some issues that arose however. We will discuss those now.

It took some users a moment to realise that the value of the slider would be revealed when they clicked on the UI, this lead some to look around for a moment before trying the slider. While not a big problem it could be resolved by having something on screen nearby that shows the value permanently visible.

One user also found that when they clicked away from the page without clicking save they lost the data they had inputted. This would be alleviated by some sort of mark to denote that the work is not saved and the user should save their work before continuing to another profile or screen.

While users easily recognised that the floating action button on the alerts page was to add an alert, they were somewhat confused by the learning curve of the alerts metric system. This could be solved by simply adding an example alert, perhaps with more descriptions to each stage of the alert creation process or the alerts list, just to give the user some hints about how the system works.

While navigating the tab systems, users also found that when changing profiles to view, the view reset to the profile details page, yet the tab highlight did not follow. This would be easily resolved with some Javascript and should be looked at in later versions of the application.

6 Critical Review

Upon reflection of the project my overarching opinion is that it went well. Regardless of the final product, there is a lot that can be learned from the process here.

6.1 Building on Existing work in the Field

The project itself built upon lots of work outlined in the literature review. We have seen how the concepts such as K-Nearest Neighbor classifiers can be used to solve the problem of sentiment analysis. The problem of sentiment analysis has been solved before in many different ways, both with and without machine learning. However the insight we gain from this is that we can end up with a viable product through a simple implementation of things like KNN classifiers, where the problem has been boxed in a the form of Twitter. Many sentiment analysis problems pre-date twitter as seen in the literature review. The dynamic application of these solutions has been seen in this project.

The Twitter API has been around for almost as long as Twitter. Many third party companies have been using it as a means of data collection for sometime. However once again we can see that it is possible to get quantitative data from tweets that aren't directly related to them. Most analytics available through Twitter exist in the realm of the likes of click-through rate and customer engagement. We can clearly see how now that it's possible to extract more information about a customer base through their tweets alone.

6.1.1 Ethical and Legal Considerations Of the Project

Artificial intelligence is a minefield of paradigmatic shifts. The way data is used is changing, even in the wider world of data science with Big Data. More data is collected from more and more people every day. Facebook has captured nearly a quarter of the population. More than 2 billion accounts exist on Facebook (Constine, 2017). This is a truly mind blowing figure. The amount of data we create collectively each day is in the petabytes (Marr, 2018).

There is much discussion that could be had therefore, about the ethics behind this vast scale data collection. Yet regardless of what we as individuals think, it is being done, and it's certainly legal. That being said the information contained within this project, while not ground breakingly revolutionary, should be treated with caution and respect as all new technology that interacts so deeply with the human centered anthropocene. The program doesn't store any information about the user who sent the tweet and as such that is one thing that need not be worried about. The option however to use this project to target an audience in general is very likely, this should be done with care and due diligence. Unfortunately I cannot assure that this sort of technology is not already being used with reckless abandon, nonetheless, we can hope.

6.1.2 Problems During the Project

The learning curve towards implementing a machine learning algorithm can be quite a steep one. And having little to no experience with machine learning this was certainly a a problem that needed to be dealt with as quickly as possible. In order to tackle this issue I started the project experimenting with various bits of machine learning and the theory behind them until I felt comfortable with implementing algorithms.

A large part of this learning curve was the mathematical theory and specifically the notation. The notation within each field of mathematics is it's own small system. These smaller systems come together to form the general world of mathematical notation. When information is presented in the form of an equation it can take some time, and research, to fully understand what is being presented. This was something that I had to deal with as and when it became a problem, most notedly during the literature review.

Another problem is hosting all the various elements online. In the end the final product did not run on the public internet. This isn't a problem to be too concerned about however. The goal of this project was never to make something that people can use openly. The application acts as a proof of concept, it's simply a *vector* for the transmission of machine learning concepts.

6.1.3 Future Work and Alterations

I believe the most valued part of this project is the construction of meta vectors with respect to tweets as these machine learning techniques have been studied extensively in many applications. One aspect I see this being taken is emoji support. For the time being the vector checks for the existence of several emoticons, however a cursory glance at twitter will leave one with the knowledge that this simply isn't as common as it once was. The now widely adopted emoji, originating from Japans mobile carriers and brought to prominence from their inclusion in Unicode (Pardes, 2018), is a staple of communication these days. Therefore where emoticons where used to understand sentiment, now emoji should join them. It would be worth giving some sentiment to each emoji where possible and using that to compile the meta vectors also.

With respect to the machine learning algorithms, I would wish to attempt a more complex neural network style approach, something in the vein of LSTMs or a convolutional network. Furthermore perhaps a better option for a neural network would be a softmax function on the output, to produce an array of confidences in the classification of each possible class.

The K Nearest Neighbor classifier that I implemented did end up being quite slow when running batches through the classification algorithm. In fact we could classify the time as somewhere in the region of $O(nd)$ (n being the amount of data, d being the dimensions of vector). This slowness comes from computing the distance between each point of training data and the new data. In the world of computer algorithms this isn't very good. It's certainly not the worst it could be; but it could be better.

While not a problem for classifying one tweet at a time, it would be beneficial to make the algorithm more efficient. More research would have to be done into this area before any implementations could be done. Solutions such as a K-D tree would be implemented to simplify the search for neighbors

With regards to project management style I would have liked to take advantage of services like Trello a little more in order to keep track of project backlog items during sprints.

6.2 The Final Word.

There is much to be learned from the field of machine learning. Much the world does not yet know and much that we simply haven't tried yet. With such a rapidly expanding and populous field it's important to ensure that the new technologies we breed are given a chance to be understood. We must take a step back think about the long term applications of the systems we produce; and more than ever we should treat new technologies with care. In her book Hello World, Dr Hannah Fry (2019) puts it well when she says;

> "In the age of the algorithm, humans have never been more important"

References

Angwin, Julia, Jeff Larson, Surya Mattu, and Lauren Kirchner (2016), "How much data do we create every day? the mind-blowing stats everyone should read." Available at: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing (Accessed :28/4/2019).

Anton, Howard (1994), *Elementary Linear Algebra, 7th edn.* New Jersey, USA: Wiley.

Bird, Steven, Ewan Klein, and Edward Loper (2009), *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

Caliskan, Aylin, Joanna Bryson, and Arvind Narayanan (2017), *Semantics derived automatically from language corpora contain human-like biases*, volume 356. Cornell University Library, arXiv.org, Ithaca.

Chang, G. and H. Huo (2018), *A METHOD OF FINE-GRAINED SHORT TEXT SENTIMENT ANALYSIS BASED ON MACHINE LEARNING*, volume 28.

Chollet, François et al. (2015), "Keras." Available at: https://keras.io (Accessed :28/4/2019).

Cliche, Mathieu (2017), *BB_twtr at SemEval-2017 Task 4: Twitter Sentiment Analysis with CNNs and LSTMs.*

Constine, Josh (2017), "Facebook now has 2 billion monthly users… and responsibility." Available at: https://techcrunch.com/2017/06/27/facebook-2-billion-users/?guccounter=1 (Accessed :28/4/2019).

Fry, Hannah (2019), *Hello world : how to be human in the age of the machine.*

Gatti, Lorenzo, Marco Guerini, and Marco Turchi (2016), *SentiWords: Deriving a High Precision and High Coverage Lexicon for Sentiment Analysis*, volume 7. IEEE.

Google (2018). *Think with Google: Brand Lift* Available at: https://www.thinkwithgoogle.com/products/brand-lift/ (Accessed: 18/2/19).

Guorui, Y. (2012), *The evaluating of the questionnaire of clothing brands and construction of dimensions based on consumers' perception.*

Hansen, Nele, Ann-Kristin Kupfer, and Thorsten Hennig-Thurau (2018), *Brand crises in the digital age: The short- and long-term effects of social media firestorms on consumers and brands*, volume 35. Elsevier B.V.

Hongmao, Sun (2005), *A naive bayes classifier for prediction of multidrug resistance reversal activity on the basis of atom typing.*, volume 48.

Jaroonrut Prinyakupt, Charnchai Pluempitiwiriyawej (2015), *Segmentation of white blood cells and comparison of cell morphology by linear and naïve Bayes classifiers*, volume 14.

Lecun, Y, L Bottou, Y Bengio, and P Haffner (1998), "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, 86, 2278–2324.

Manning, Christopher D, Prabhakar Raghavan, and Hinrich Schütze (2008), *Introduction to Information Retrieval*. Cambridge University Press - M.U.A.

Marr, Bernard (2018), "How much data do we create every day? the mind-blowing stats everyone should read." Available at: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/ (Accessed :28/4/2019).

McCallum, Layton (2015). To what extent do you agree or disagree with the following statement: "I'm more likely to let a company know I'm dissatisfied if I can contact them through Twitter or Facebook than if I went to their customer service"?. Statista. Available at:https://www.statista.com/statistics/476243/consumers-likelihood-of-reporting-dissatisfaction-on-social-media-in-the-uk/. (Accessed: January 24, 2019).

Miller, Michael B. (Michael Bernard) (2014), *Mathematics and statistics for financial risk management*, second edition.. edition. Wiley finance series.

Moore, G.E (1998), *Cramming More Components Onto Integrated Circuits*, volume 86. IEEE.

Oliphant, Travis (2006), *Guide to NumPy*.

Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan (2002), *Thumbs up? Sentiment Classification using Machine Learning Techniques*. Cornell University Library, arXiv.org, Ithaca.

Pardes, Arielle (2018), "The wired guide to emoji." Available at: https://www.wired.com/story/guide-emoji/ (Accessed :28/4/2019).

Price, Rob (2016), *Microsoft is deleting its AI chatbot's incredibly racist tweets*. Available at: https://www.businessinsider.com/microsoft-deletes-racist-genocidal-tweets-from-ai-chatbot-tay-2016-3 (Accessed: 18/2/2019).

Rashid, Tariq (2016), *Make your own neural network : a gentle journey through the mathematics of neural networks, and making your own using the Python computer language*. CreateSpace Independent Publishing Platform, United States.

Raval, Siraj (2017), *Natural Language Processing and Sentiment Analysis*. Medium.com. Available at: https://medium.com/udacity/natural-language-processing-and-sentiment-analysis-43111c33c27e (Accessed: 3/2/2019).

Rozental, Alon and Daniel Fleischer (2017), *Amobee at SemEval-2017 Task 4: Deep Learning System for Sentiment Detection on Twitter.* Cornell University Library, arXiv.org, Ithaca.

SemEval (2017), *SemEval-2017; International Workshop on Semantic Evaluation.* Available at: http://alt.qcri.org/semeval2017/ (Accessed: 10/2/2019).

Shannon, C. E. (1951), *Prediction and Entropy of Printed English*, volume 30. Blackwell Publishing Ltd, Oxford, UK.

Sánchez-Franco, Manuel J., Antonio Navarro-García, and Francisco Javier Rondán-Cataluña (2018), *A naive Bayes strategy for classifying customer satisfaction: A study based on online reviews of hospitality services.* Elsevier Inc.

Stojanovski, Dario, Gjorgji Strezoski, Gjorgji Madjarov, Ivica Dimitrovski, and Ivan Chorbev (2018), *Deep neural network architecture for sentiment analysis and emotion identification of Twitter messages*, volume 77.

Turing, A. M. (1950), *Computing Machinery and Intelligence*, volume 59. [Oxford University Press, Mind Association], URL `http://www.jstor.org/stable/2251299`.

Wachter, Sandra, Brent Mittelstadt, and Luciano Floridi (2017), *Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General Data Protection Regulation*, volume 7.

Zipf, George Kingsley (1929), *Relative Frequency as a Determinant of Phonetic Change*, volume 40.