Index

Title	Start	End
How the algorithm works and implementation	2	3
File Structure	3	3
File Explanation	3	10
BagOfWords directory	4	8
BagOfWordsTrainer directory	8	10
Collecting more data	11	11
Modifications to the Dataset	11	11
Re-training the model	11	11
Adding new words	11	12
Modifying the algorithm	12	12
Adding more social platforms	13	13
Adding new users	13	13
Supervised Learning/Re-Learning	14	14

How the algorithm works and implementation:

All the files related to the algorithm can be found inside the “bagOfWordsTrainer” directory. The algorithm itself is contained inside the “runAlgo.php” file. Given a string the algorithm will return an array which contains two elements, the first is the score of the tweet and the second is one of the letters 'g', 'b' or 'n' where they mean good, bad and neutral respectively. The algorithm is instantiated by making a call to the “returnSentiment” function. This function takes a string as the only argument and returns an array of length 2. Certain pre-processing is done to the string before being passed to the function. All @, # and numbers are stripped off. For example: “Had 2 pleasant coffees @virginTrain #goodCoffee” would be converted to “Had pleasant coffees virginTrain goodCoffee”. The returnSentiment fucntion will split this string and look at each non-stop words individually. Stop words are defined as words that regularly occur in the English Language such as “the”, “a” etc. We also include the train and service names such as “virginTain” in this stop-word list as we dont want any sentiment associated with these words. The complete stop-word list can be found in “atosrail/bagOfWords/bagOfWordsTrainer/stopWords.txt”. Additional stop words can be appended to this file and will result in the algorithm overlooking those words. The “wordArray.txt” is a serialised php array. Each element in the array is a word and it has three components 'g', 'b' and 'n' representing good, bad and neutral respectively. Each of these represents how many times this word appeared in the training data where the sentiment of the sentence in which it appeared in. For example: [thanks] => Array ([b] => 15 [g] => 359 [n] => 23) means that the word “thanks” appeared in 15 bad tweets and 359 good tweets and 23 neutral tweets. There is another file called “returnCountArray.txt”, this is also a serialised php array and contains the number of good, bad and neutral tweets in our training data. The sentiment of the word is determined by the “maximum participation percentage”. Lets assume we have 150 good, bad and neutral tweets, so 450 tweets in total. Then the “maximum participation percentage” is of good as 359/150 > 15/150 && 359/150 > 15/150. This is similar to evaluating how good or bad the word this. If the sentiment of the word is good then the overall score is incremented by 1 and if it is negative then decremented by 1 else score is unchanged. If the final score of the string is above the positive-threshold then the string is positive, if below negativeThreshold then negative else neutral. These thresholds are defined inside the “runAlgo.php” file and can be modified. If the word that is currently in focus is not found in the “wordArray.txt” then we look into the “positiveWordArray.txt”. If the word is found in this array then we classify the word as postive. If we do not find the word then we look into “negativeWordArray.txt” and if found the word is classified as negative else this is an unseen word and we have to skip this.

The description of the algorithm is quite verbose and its hard to clearly understand it from the text above, hopefully the algorithm given below will help clarify the concepts about this algorithm:

```
function returnSentiment(string):
    Pre-process the string to remove [#, @]
    score = 0
    foreach word in string:
        If word is in "stopWords.txt":
            break
        else if word in "wordArray.txt":
            get sentiment of word
            if sentiment == 'g':
                score += 1
            else if sentiment == 'b':
                score -= 1
        else if word in "positiveWordArray.txt":
            score += 1
        else if word in "negativeWordArray.txt":
```

score -= 1
compare score with threshold and return the appropriate sentiment and score

File Structure:

The application running on azure resides inside the atosrail/bagOfWords/ directory. Inside this we will find another directory called “bagOfWordsTrainer”. This “bagOfWordsTrainer” directory contains all the files that are needed for the algorithm to run. It is inside the “bagOfWordsTrainer” directory where we can modify the algorithm or extend the dataset and re-train the algorithm etc. The “bagOfWordsTrainer” and “bagOfWords” directory have these files in common: “negativeWordArray.txt”, “positiveWordArray.txt”, “returnCountArray.txt”, “runAlgo.php”, “stopWords.txt”, “wordArray.txt”. These are the files that are necessary for the algorithm to run, the other extra files inside “bagOfWordsTrainer” consists of the dataset and the code that trains the algorithm. If you want to modify anything about the algorithm you should do it inside the “bagOfWordsTrainer” directory, re-train the model and paste the following files: “negativeWordArray.txt”, “positiveWordArray.txt”, “returnCountArray.txt”, “runAlgo.php”, “stopWords.txt”, “wordArray.txt”, into the “bagOfWords” directory.

File Explanation:

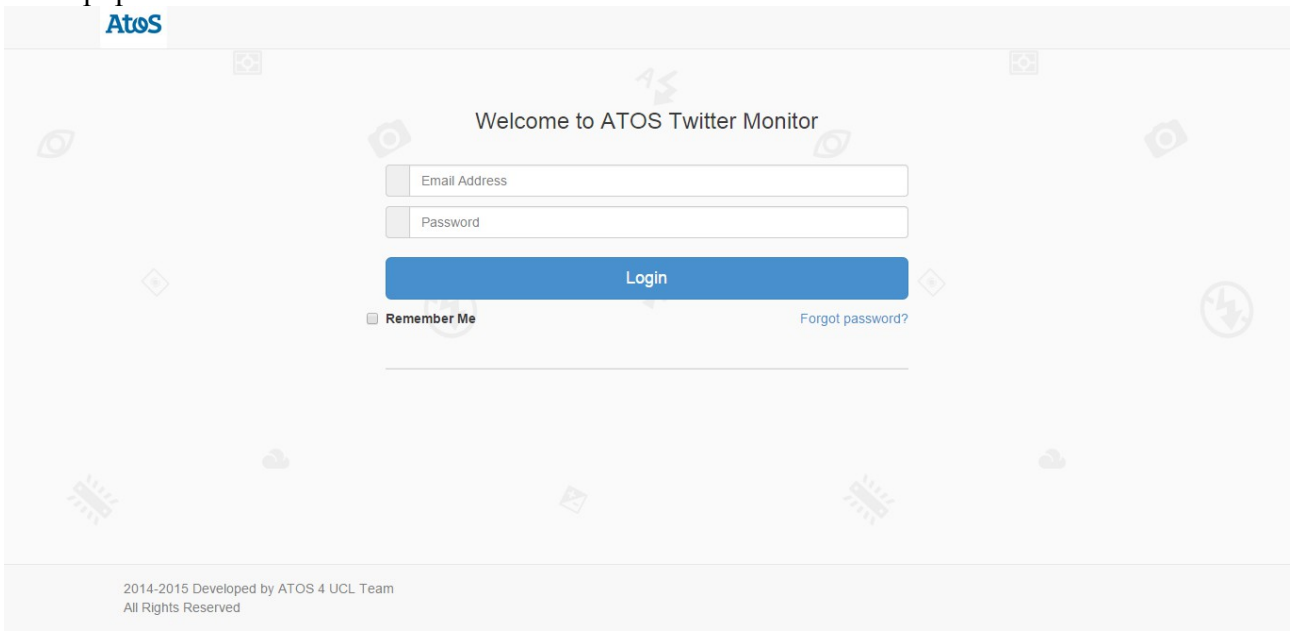
In this section we will go into detail and explain what each file contains, its role and how it fits into the application.

On the top-most level we have the following files and folders (folders are without extension)

bagOfWords
css
fonts
js
twemoji
index.php

Css, fonts, js, twemoji:
These folders contain the bootstrap code used to make the application dynamic and responsive, twemoji contains the libraries required for identifying and rendering emotions in the tweets. We are not concerned with this folder as there is virtually no code written by us, its all libraries and pre-packaged code.

index.php:



This is the welcome login screen.

This page will authenticate and allow entry to authorized personnel.

BagOfWords:

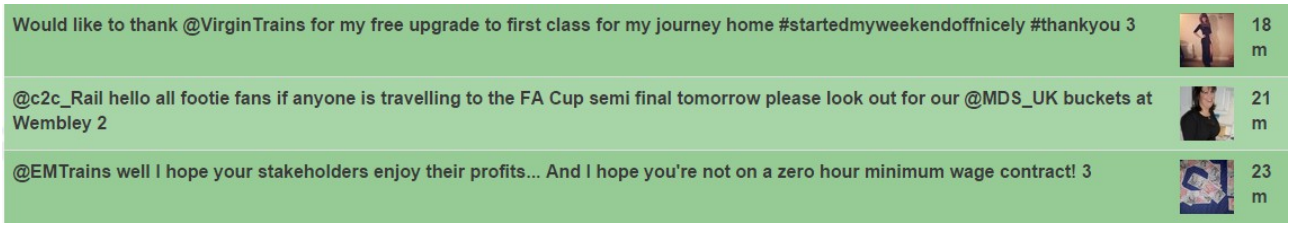
This is the directory that contains the core part of the project and all the code directly related to the project is inside this folder. The contents of this directory are as follows:
bagOfWordsTrainer
all.php
bad.php
chooseColorDepth.php
login.php
logo.png
menuconreal.png
negativeWordArray.txt
next.php
Oauth.php
photography.png
positiveWordArray.txt
reload.php
returnCountArray.txt
runAlgo.php
setings.php
stopWordArray.txt
success.php
testDate.php
testsentiment.php
twitteroauth.php
warning.php
wordArray.txt
all.php:

After the tweets have been fetched and processed by login.php, the tweets are stored in a session variable (passed_array) along with their sentiment. In this page the session variable is fetched and each element in the array is printed sequentially regardless of the sentiment attached with a specific tweet. The purpose of this file is to print the entire contents of the "passed_array".

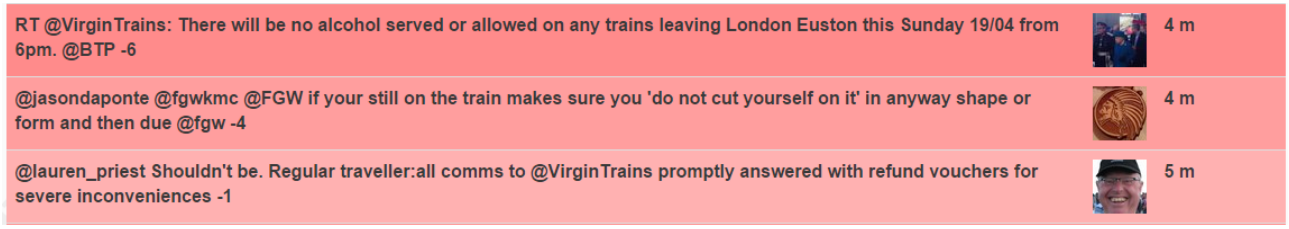
bad.php:
The session variable "passed_array" is loaded and looped over whilst printing only the tweets that have a negative sentiment. The purpose of this page is to print all the tweets that have a bad sentiment in the passed_array variable.

ChooseColorDepth.php:
Takes the sentiment score of each tweet and returns an appropriate good/bad color hexdecimal code. In this way, tweets with different sentiment scores get different shades of color.

login.php:
After the user enters their username and password they are taken to this page (if the authentication is successful). This page does most of the work in this application. First depending on the username a parameter called "query" is fetched from the database. This parameter is defined by the user in settings.php page and contains a list of space-delimited words which are the words to be searched for in the twitter data stream. For example if the query parameter is "virginTrain c2c_railway" then the twitter data stream will be searched for only these tags "#virginTrain @virginTrain # c2c_railway @c2c_railway". After the tweets are retrieved from twitter, we loop over each tweet calling the returnSentiment function to determine the sentiment of the tweet and display the tweet. The score of the tweet determines the color depth that the tweet is going to receive. A good tweet has three level of goodness, the more good the tweet is the greener the container containing the tweet becomes



and vice versa for bad tweets. The worse the tweet is the more red the container becomes:

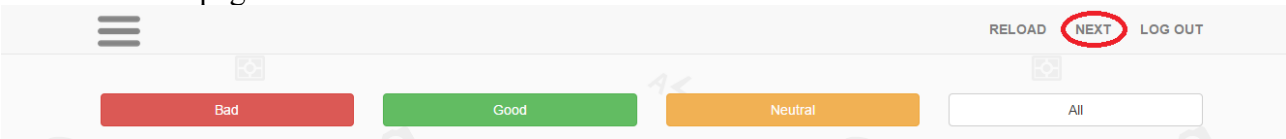


logo.jpg:
The logo used in almost all pages in the application.

menuconreal.png:
An image used to represent the settings button.

negativeWordArray.txt:
This is a serialised php array. It contains an array whose elements are negative words in the English dictionary. Example [0] => abnormal, [1] => abolish etc. This file is generated by the creayeNegativeArray.php file inside bagOfWordsTrainer directory.

next.php:
From any page if the user clicks on the next button on the top right of the current page they are directed to this page.



This page uses a session variable called “lastTweet” set by login.php page. This variable contains the id of the last tweet returned by twitter on the login.php page. The job of this page is to then fetch the next 100 tweets with id after the “lastTweet”. This is done by the following lines of code:

```
$max_id = $_SESSION["lastTweet"];
 $twitterQueryString = "https://api.twitter.com/1.1/search/tweets.json?q=".
$twitterQueryString."&count=100&max_id=".$max_id;
```

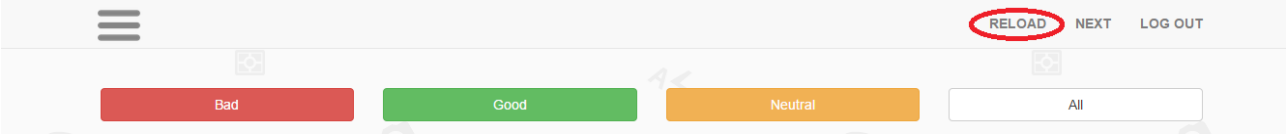
The max_id parameter of the url tells twitter to only return tweets older than the tweet with max_id as the twitter id. The rest of the code inside next.php is almost identical to the login.php as they both do the same processing on the tweets.

Oauth.php:
Used by the twitter api for authentication purposes. This file is provided by twitter so no modification should be made to this.

Photography.png:
The background image used on every page.

positiveWordArray.txt:
This is a serialised php array. It contains an array whose elements are positive words in the English dictionary. Example [0] => abundance, [1] => abundant etc. This file is generated by the createPositiveArray.php file inside bagOfWordsTrainer directory.

reload.php:
From any page if the user clicks on the reload button on the top right of the current page they are directed to this page.

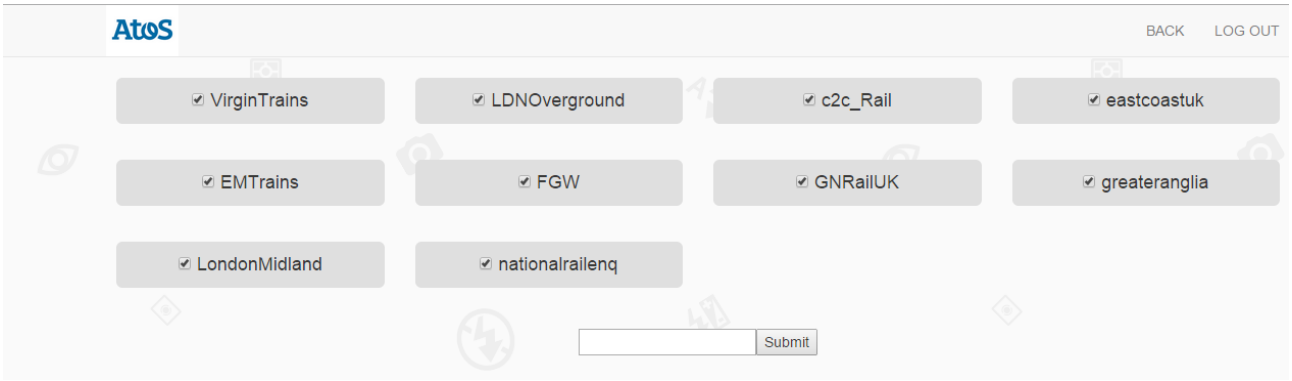


This is identical to a refresh button. This page requests twitter for 100 latest tweets and displays them accordingly. The code is again very similar to the login.php page with the exception that user authentication is not done as it is already taken care of at the login.php page.

returnCountArray.txt:
This is a serialised php array which contains the number of good, bad and neutral tweets in the dataset. The array is very simple and its structure is as follows:
[g] => 350, [b] => 604, [n] => 154. This means that we have 350, 604, 154 good, bad and neutral

tweets respectively. This file is generated by the returnCount.php file inside bagOfWordsTrainer directory.
runAlgo.php:
The returnSentiment function resides in this file. The primary job of this file is to provide the returnSentiment function which takes a string as an argument and returns the score and sentiment of the tweet. This file loads up most of the serialised arrays unserializes them and puts them to use. The working of the algoithm has been detailed previously and thus also includes a good description of the contents of the runAlgo.php. The runAlgo.php in the bagOfWords directory is the slave copy and the master copy is in the bagOfWordsTrainer directory, if one wants to make any change to the algorithm they should change the master copy and then replace the slave.

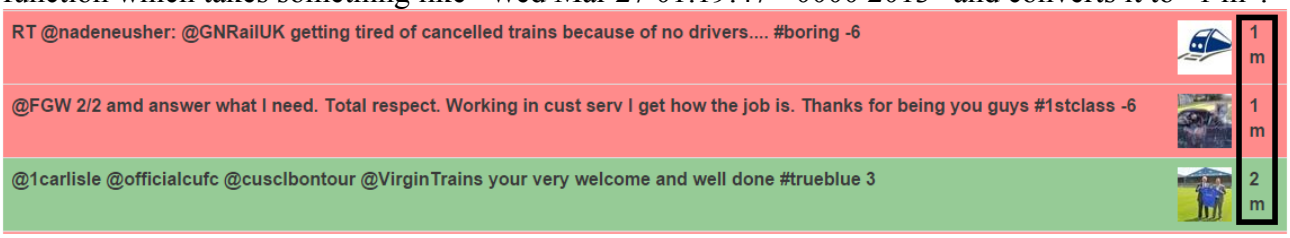
settings.php:
When the user logs into the system the username is saved in a session variable called "username". This file retrieves that variable to know which user is currently on the system and fetches the corresponding query field from the database and display the options. The user can add new parameters that they are interested in monitoring, delete current ones and the updates will be pushed to the database.



stopWordArray.txt:
This is a serialised php array. It contains an array whose elements are stop words in the English dictionary. Stop words are regularly occurring words and should not have any sentiment attached to them Example [0] => the, [1] => and etc. This file is generated by the createStopArray.php file inside bagOfWordsTrainer directory.

success.php:
The session variable "passed_array" is loaded and looped over whilst printing only the tweets that have a positive sentiment. The purpose of this page is to print all the tweets that have a positive sentiment in the passed_array variable.

testDate.php:
This page is used to render the date in a human readable format. The date and time returned by twitter is long and hard to process visually. The main job of this page is to provide the dateF function which takes something like "Wed Mar 27 01:19:47 +0000 2013" and converts it to "1 m".



testsentiment.php:
This is the shortest pages in the entire application in terms of lines. This allows the user to test the algorithm. The user inputs a string and the score and sentiment is returned. Its like a “sandbox” tool to get a feel for the output of the algorithm.

twitteroauth.php:
Used by the twitter api for authentication purposes. This file is provided by twitter so no modification should be made to this.

warning.php:
The session variable “passed_array” is loaded and looped over whilst printing only the tweets that have a neutral sentiment. The purpose of this page is to print all the tweets that have a neutral sentiment in the passed_array variable.

wordArray.txt:
This is a serialised php array. It is created by the trainModel.php file in bagOfWordsTrainer directory. This is perhaps one of the most essential files in the application as this contains the trained model that is used by the runAlgo.php. Each element in the array itself is an array with several parts so wordArray.txt is an array of arrays. An example entry in the array looks like [good] => Array ([b] => 15 [g] => 359 [n] => 23) meaning that the word “good” appeared in 15 bad tweets and 359 good tweets and 23 neutral tweets.

The following are the contents of the bagOfWordsTrainer directory

bagOfWordsTrainer:

1.txt
2b.txt
3cleaned.txt
createNegativeArray.php
createPositiveArray.php
createStopArray.php
negativeWordArray.txt
negativeWords.txt
positiveWordArray.txt
positiveWords.txt
returnCount.php
returnCountArray.txt
runAlgo.php
stopWordArray.txt
stopWords.txt
trainModel.php
wordArray.txt
TwitterFeedAtos.py

1.txt:
Contains a list of labelled training tweets for the model. By labelled we mean that after each tweet the expected sentiment of the tweet is separated by >!!!<. For example:
"@greateranglia morning guys. Is there a problem with the 7.26 from bic. Very slow and juddery. And ony just getting to sra. >!!!< b"
Each line in this file corresponds to a tweet and we have categorised each tweet manually by reading them and using our common sense to judge the sentiment. These tweets are then used as the training data for the model. Since there were three member of the team we split the initial 1000 unlabelled tweets into three files to be labelled. This is one of those three files.

2b.txt:
Exactly the same as 1.txt and contains the tweet labelled by the second member of the team.

3cleaned.txt:
Exactly the same as 1.txt and contains the tweet labelled by the third member of the team.

createNegativeArray.php:
This file reads the negativeWords.txt file, converts it into a serialised php file called negativeWordArray.txt.

createPositiveArray.php:
This file reads the positiveWords.txt file, converts it into a serialised php file called positiveWordArray.txt.

createStopArray.php:
This file reads the stopWords.txt file, converts it into a serialised php file called stopWordArray.txt.

NegativeWordArray.txt:
Created by createNegativeArray.php and is a serialised php array. This is the master copy and the slave copy is in the bagOfWords directory. It contains an array whose elements are negative words in the English dictionary. Example [0] => abnormal, [1] => abolish etc.

negativeWords.txt:
Is a normal text file where each line is a negative word in the English dictionary. Used by createNegativeArray.php to create NegativeWordArray.txt.

PositiveWordArray.txt:
Created by createPositiveArray.php and is a serialised php array. This is the master copy and the slave copy is in the bagOfWords directory. It contains an array whose elements are positive words in the English dictionary. Example [0] => accolade, [1] => admirer etc.

positiveWords.txt:
Is a normal text file where each line is a positive word in the English dictionary. Used by createPositiveArray.php to create positiveWordArray.txt.

returnCount.php:
This file creates the returnCountArray.txt. It loops over dataset i.e. 1.txt, 2b.txt and 3cleaned.txt and counts the total number of good, bad and neutral tweets and saves this information in returnCountArray.txt.

returnCountArray.txt:
Is a serialised array and contains the number of good, bad and neutral tweets in the dataset. Example: [g] => 200, [b] => 700, [n] => 100.

runAlgo.php:
This is the master copy and the returnSentiment function resides in this file. The primary job of this file is to provide the returnSentiment function which takes a string as an argument and returns the score and sentiment of the tweet. This file loads up most of the serialised arrays unserializes them and puts them to use. The working of the algorithm has been detailed previously and thus also includes a good description of the contents of the runAlgo.php. The runAlgo.php in the bagOfWords directory is the slave copy and this is the master copy, if one wants to make any change to the algorithm they should change the master copy and then replace the slave.
stopWordArray.txt:
Is a serialised array and contains a list of stop words in the English dictionary. Each element in this array corresponds to a stop word. Example: [0] => the, [1] => a etc.

stopWords.txt:
Contains a list of stop words in the english dictionary. This is a simple text file and each line corresponds to a stop word.

trainModel.php:
The wordArray is created by this file and thus it is called trainModel because the wordArray is the model used by the algorithm. This loops over each individual tweet and starts creating the wordArray, interestingly the stopWords are excluded here and not at the runAlgo.php page. The wordArray contains all the words in our dataset excluding the stop words and so we do not need to check to exclude the stop words every single time we call the returnSentiment function and thus this design pattern saves us a lot of computation.

wordArray.txt:
Finally this is th e master copy of the WordArray and if the algorithm is re-trained then this file should replace the one in bagOfWords directory. It is created by the trainModel.php file. This is perhaps one of the most essential files in the application as this contains the trained model that is used by the runAlgo.php. Each element in the array itself is an array with several parts so wordArray.txt is an array of arrays. An example entry in the array looks like [good] => Array ([b] => 15 [g] => 359 [n] => 23) meaning that the word “good” appeared in 15 bad tweets and 359 good tweets and 23 neutral tweets.

TwitterFeedAtos.py:
Description given in Collecting more data section

Collecting more data:

A single file called TwitterFeedAtos.py encapsulates all that is needed to fetch more data. This uses the twitters streaming api and thus will take a long time to get a reasonably good amount of tweets, maybe 4 to 5 hours for 1000 new tweets. Upon running this file another file will be created called testdata.csv. Since this is an open connection you must manually stop the running code to close the connection to twitters server.

Modifications to the Dataset:

Here we will show how to add more data to the dataset. All modification must be made in the bagOfWordsTrainer directory.
After having collected the tweets one needs to manually label each tweet. The strict requirements are that each tweet must be on a new line and at the end os each tweet put a “space” and then “>!!! <” followed by another “space” followed by the sentiment that you think this tweet should be. After you have labelled all the tweets that you collected simply copy-paste them to the end of “1.txt” file.

```
@SW_Trains awesome thanks >!!!< gCRLF
@SW_Trains Basingstoke. >!!!< nCRLF
@Se_Railway high speed from Stratford intl to Chatham! You too and thank you! >!!!< gCRLF
RT @DeanShulver: @Se_Raleway @Se_Railway @Garyw_ it should be \ >!!!< n CRLF
@fgw why is the 7.51 Westbury to Paddington stopped at the station? >!!!< bCRLF
@getsurrey is there money to refurb addlestone station? @SW_Trains @Rowtown @DontLetYourTea >!!!< nCRLF
@Se_Railway Adult Disability. >!!!< nCRLF
```

The CRLF represents a new line in windows machines however a single LF is fine as well. We leave it up to the user to decide how they want to achieve this.

Re-training the model:

Having collected the new data the next step is to re-train the wordArray to contain these new changes. Simply run the trainModel.php on an apcahe server and it will re-create the wordArray.txt file. This new wordArray contains everything that it add previously in addition to the new data introduced. Copy paste the newly generated wordArray.txt into the bagOfWords directory and replace the existing. The server should immediately reflect upon these changes and the strings will be classified accordingly to incorporate the new data.

Adding new words:

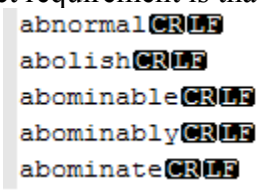
This section explains the process of adding new words to the positive, negative and stop word list.

Positive:
To add new words to the positive list simply add the new words at the end of positiveWords.txt, the user does not need to worry about duplication as while the serialised array is generated only unique occurrences are stored. The strict requirement is that each word must be on a new line.

```
aboundCRLF
aboundsCRLF
abundanceCRLF
abundantCRLF
accessableCRLF
```

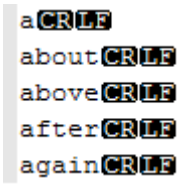
After adding the new positive words run the createPositiveArray.php file. This will create the serialised array positiveWordArray.txt. Copy paste this file into the bagOfWords directory.

Negative:
To add new words to the negative list simply add the new words at the end of negativeWords.txt, the user does not need to worry about duplication as while the serialised array is generated only unique occurrences are stored. The strict requirement is that each word must be on a new line.



After adding the new negative words run the createNegativeArray.php file. This will create the serialised array negativeWordArray.txt. Copy paste this file into the bagOfWords directory.

Stop-Words:
To add new words to the stop-word list simply add the new words at the end of stopWords.txt, the user does not need to worry about duplication as while the serialised array is generated only unique occurrences are stored. The strict requirement is that each word must be on a new line.



After adding the new stop words run the createStopArray.php file. This will create the serialised array stopWordArray.txt. Copy paste this file into the bagOfWords directory.

Modifying the algorithm:

The algorithm itself resides in the runAlgo.php file. If one wishes to make any changes to the algorithm this is where it should be done. The entire working of the algorithm can be changed here and more specifically certain thresholds can be changed which are used to determine the sentiment of the tweet. As discussed previously the core functionality of this file is to provide the returnSentiment function and changes made to this file will drastically affect the out put the entire application. One interesting thing to note about the application is the dataset. In any machine learning process the most important thing is the dataset which is going to be used to train the model. We recognised the importance of the training data set and thus decided to manually label 1000 tweets. This is a very boring and repetitive task but a necessary one indeed. If one reads through the dataset they would easily spot a pattern that is most of the tweets are negative. That is, the dataset is biased towards negativity and thus one might decide that every time we see a positive word we increase the overall score by 2 (currently positive words and negative words are treated equally that is +1 for positive and -1 for negative) to balance out the dataset bias. This is an interesting idea and we leave it for future collaborators to experiment on this or to add more positive tweet to make the dataset more “neutral”.

Adding more social platforms:

Currently the application is solely focused on twitter and collects all its information from twitters data stream. It is possible to extend the application to fetch information from Facebook, MySpace etc. The important thing to remember is that this application is text-based and the sentiment analysis is performed on the text only. In choosing new social platforms it is recommended to do so on text based platforms and not pictures or videos such as Instagram and youtube. However, if the developer can include algorithms for classifying pictures and videos and does so in a affordable computation cost then they can include whatever platform they wish to do so. The functional code resides in the following files:

1. login.php
2. all.php
3. success.php
4. warning.php
5. bad.php
6. settings.php
7. next.php

Changes made to these files would affect the front-end of the application and thus these are the files that will be modified to make changes to the appearance of the application.

Adding new user:

To add a new user that will be able to use the system first the admin must connect to the MySQL database using the following command from any terminal that supports MySQL:

mysql -heu-cdbr-azure-north-c.cloudapp.net -ubf5a119d46ef1d -p1b7bd7ec -DatosraiASL0S22pH

After connecting to the database the following code should be used add a new user

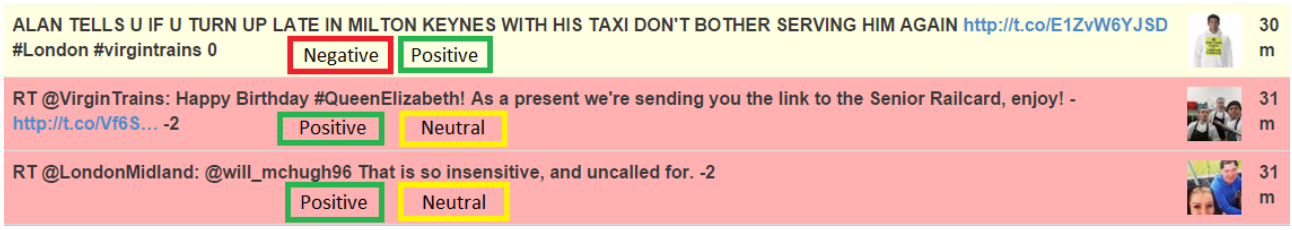
INSERT INTO users (username, password, query) VALUES ('<username>', '<password>', '<space delimited query>');

It is advised to enter at least 1 value in the query field and to not keep it empty.

Supervised Learning/Re-Learning:

In this section we brief how it might be possible to incorporate supervised learning to re-train the data-model over a period of time by user feedback from the application's usage.

The idea is to have opposing sentiment buttons at the end of each tweet i.e. if the sentiment of the tweet is positive then there should only be buttons for negative and neutral buttons available for that tweet.



If the current tweet has been labelled as neutral by the bag of words algorithm and the user thinks that the tweet should be negative then by pressing the negative button the user will re-train the data-model to more likely output negative in future instances of similar tweets. We will leave it up to the developer to come up with a suitable implementation of such a supervised learning algorithm to suit the data-model and the algorithm used in this application.

The Developer must take into account the fact that the data-model is already extensible and this is done by regenerating all the data from the tweets in “1.txt”, “2b.txt” etc. If the supervised changes are applied on the wordArray then they will be **over-written** if the data-model is re-generated thus extra caution must be taken and new code introduced in the core application to handle the supervised learning data.