Petur Einarsson - Coursework 1

Part A: Anger Detection

1. parseTweet and preProcess

All data is loaded from the provided .CSV files line by line. Each line has an ID, date, username, language and text. The date and the text is extracted with 'preProcess' and stored, while empty lines are discarded. Date is stored as a datetime object which is easily read by Python. The text is then processed further by using 'parseTweet'. ParseTweet tokenizes the text using a Twitter tokenizer from NLTK, which splits the text string into its individual words and each word is then cleaned. The words made lowercase, unnecessary characters stripped away, urls replaced with a single identifiable string and the words stemmed using PorterStemmer.

2. toFeatureVector

Using the toFeatureVector function, a dictionary of each and every word is built whilst the data is loaded from the .CSVs. The function checks if a word is already in the dictionary. If it's not there, it gets added with a value of one, else it's value gets incremented by one.

3. TrainClassifier and cross validation

After all the data has been loaded, processed and cleaned, it can now be used to train a classifier to predict whether a tweet is happy or angry. However, before using the classifier on real data, it's good to test how accurate it is using cross validation.

Whilst the training tweets were loaded, they were given a label of either happy or angry, depending from which dataset they came from. This training dataset is then shuffled randomly, and split into 10 subsets, or folds. Each fold is then used as a testing set while the other nine are used to train the classifier. Since the true label of the testing subset is known, it can be used to test how accurately a classifier predicts the labels of the tweets. After each fold it's accuracy, precision, recall and f_score values are stored. The average scores from those 10 folds are used tell how accurate the classifier is. The initial classification gave an accuracy score of around 65% with other metrics hovering around 50%.

4. Improving classifier scores

The classifier was improved by cleaning the input data. Putting the words to lowercase seemed to improve the scores by around 5%. Lemmatizing the words didn't seem to improve the score by much, whilst stemming them improved the score by just under 10%. Removing stopwords was surprisingly detrimental and was therefore not included. Happy tweets were shown to be more likely to include URLs, so all URLs were then weighted by replacing them with a single unique string "__URL__". This improved scores slightly. Removing unnecessary characters such as '!?;:,.)(' improved scores by around 5%.

The tokenizer was later changed from a generic one, to a Twitter specific one, which improved the scores significantly The change meant that hashtags (#tubestrike) and

retweets(@TFL) were kept as they were instead of being split (#, tubestrike & @, TFL).

The final classifier shows to have about a 94.5% accuracy and is therefore successfully able to predict the labels of the tweets.

```
**************************
***** AVERAGE METRICS *****
**************************
 Accuracy:   0.9453
 Precision:  0.9454
 Recall:     0.9453
 Fscore:     0.9453
**************************
```

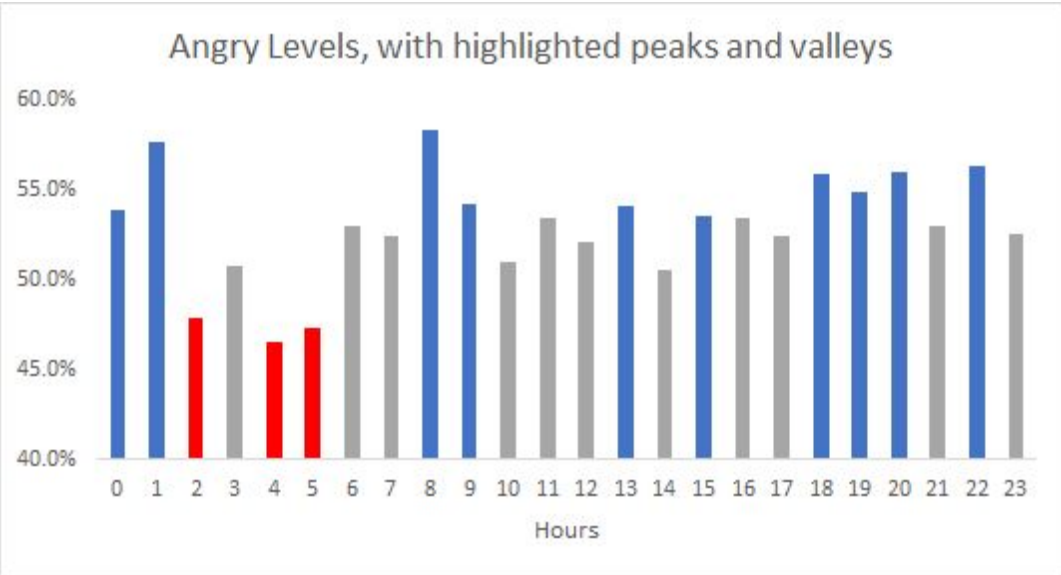
5. Angry levels

The classifier was trained on the entire training set of happy and angry tweets and then used to predict each and every tweet in London during the London tube strike on the 9 of January 2017. It then split the prediction by hour and found the angry levels of that hour, which was defined as the number of angry tweets, divided by the total number of tweets in a given hour. The top 10 angry level peaks were found by sorting the list by values and getting the first 10.

Top 10 -- Peaks	
Hour	**Percentage**
8	58.4%
1	57.7%
22	56.4%
20	56.1%
18	55.9%
19	54.9%
9	54.3%
13	54.2%
0	53.9%
15	53.6%

Most tweets were found to be angry on the day of the strike. The angry level was only below 50% on 3 hours of the day; between 02:00 - 02:59 and again between 04:00 - 05:59. The peaks were found to be around the same time as the busiest hours of the tube; when many commuters go to and from work, in the morning and again in the afternoon / night.

The below graph shows angry levels by hour, with valleys in red and peaks in blue.



Part B

1. getBigrams

As with Part A, the first thing is to load, process and clean the data. In addition to the same functions, a function to separate the dataset by dates was added. This allows us to get separate datasets for the 5th and 9th of January. GetBigrams was relatively simple to implement as it could use the text preprocessing from Part A. After the Tweets had been processed, getBigrams looped through each tweet, pairing up words with the one that came after it and appending it to a list of bigrams. The list is then simplified by using 'findUniqueBigrams', which creates a dictionary of unique bigrams. This makes it simpler and computationally less heavy to loop through later on.

2. Find probability of 'tube' followed by 'strike'

The full list of bigrams of 9th, the 5th and the entire dataset, is passed to the provided function conditionalProbDist, along with a probability distribution factory such as MLE. The function creates an object that can be used to calculate distributions over conditional bigram probabilities. Using the object it's possible to see how likely a word is to follow another word. Such as 'strike' coming after the word 'tube'. As shown below that particularly bigram is vastly more likely to occur on the day of the strike than any other day.

3. Ratio between two distributions

The next part is to see which bigrams are more likely to occur on the 5th and the 9th compared to the rest of the dataset. This is done by finding the probability of each and every bigram occurring on those days, and comparing its probability to occur in the entire dataset. Each bigram and their probability are stored as a separate dictionary for each dataset; the bigram is the key and the probability is the value. Then the bigrams that show the biggest variance are found and shown below. The expected results were to see some mention of the tube strike. The most likely cause is statistical, due to those particular tweets being tweeted only on that day. This might be fixed by using another probability distribution factory other than MLE.

From the 5th of January:

Index	Type	Size	Value
0	tuple	2	(('giroud', 'have'), 0.9966442953020134)
1	tuple	2	(('neo', 'fascist'), 0.9962121212121212)
2	tuple	2	(('hiphop', 'and'), 0.9961089494163424)
3	tuple	2	(('#fridayfeel', '#herecomestheweekend'), 0.99581589958159)
4	tuple	2	(('ukrain', 'it'), 0.9952380952380953)
5	tuple	2	(('@johnkerri', 'blame'), 0.9939759036144579)
6	tuple	2	(('#twglobe', '#countdown'), 0.9939024390243902)
7	tuple	2	(('starman', '('), 0.9939024390243902)
8	tuple	2	(('@potu', "putin'"), 0.9938650306748467)
9	tuple	2	(('@gilesmacdonogh', '@awedgewood'), 0.9933333333333333)

From the 9th of January:

Index	Type	Size	Value
0	tuple	2	(('👦', '🧭'), 0.9969788519637462)
1	tuple	2	(('giroud', 'close'), 0.9966442953020134)
2	tuple	2	(('#fridayfeel', '#giveaway'), 0.99581589958159)
3	tuple	2	(('payet', 'or'), 0.9957627118644068)
4	tuple	2	(('gospel', 'oak'), 0.9930555555555556)
5	tuple	2	(('💮', 'on'), 0.9928057553956835)
6	tuple	2	(('#tuesdaymotiv', '@akashagarni'), 0.9924812030075187)
7	tuple	2	(('farewel', 'in'), 0.9923076923076923)
8	tuple	2	(('0-0', '@flfc_offici'), 0.991869918699187)
9	tuple	2	(('@bullysspeedboat', 'what'), 0.9915254237288136)

4. Using Laplace Probability Distribution

The Laplace factory often tends to produce a more accurate result as it introduces the concept of smoothing to the way it calculates probabilities. This is shown in the results as they seem to be more accurate or more to what was to be expected, with many bigrams being related to the tube strike on the 9th of January.

From the 5th of January:

Index	Type	Size	Value
0	tuple	2	((None, '__URL__'), 0.0004505767175491419)
1	tuple	2	((None, '5'), 0.00028871768828802095)
2	tuple	2	(('wednesday', '4'), 0.00021948238894378393)
3	tuple	2	(('ask', 'for'), 0.00016280064202970718)
4	tuple	2	(('never', 'ask'), 0.00015866657687443162)
5	tuple	2	(('@fbi', 'never'), 0.00015709834372452 52)
6	tuple	2	(('comput', 'server'), 0.00015615796586796363)
7	tuple	2	(('server', '@whitehous'), 0.00014399821166261 63)
8	tuple	2	(('4', None), 0.00014142318651508 77)
9	tuple	2	(('are', 'you'), 0.00013576297585067202)

From the 9th of January:

Index	Type	Size	Value
0	tuple	2	(('tube', 'strike'), 0.000672343231457431)
1	tuple	2	((None, '#tubestrik'), 0.0004973441910144886)
2	tuple	2	(('to', 'work'), 0.00033767223435634137)
3	tuple	2	(('to', 'get'), 0.00029912961195200044)
4	tuple	2	(('strike', None), 0.0002985651850701845)
5	tuple	2	(('#tubestrik', '__URL__'), 0.00026610877669550525)
6	tuple	2	(('the', 'tube'), 0.00025996919573521 03)
7	tuple	2	((None, '9'), 0.00023847938748152442)
8	tuple	2	(('__URL__', '__URL__'), 0.00023570665955981 39)
9	tuple	2	(('meryl', 'streep'), 0.00022655887088373662)