2ID26 Web Information Retrieval and Data Mining

Group project report

An Experiment on Election Prediction using Sentiment Analysis

Table 1: Group Members

Student	Email id
Ankur Tiwari	a.tiwari@student.tue.nl
Pham Duy	d.pham.duy@student.tue.nl
Saurabh Gawande	s.s.gawande@student.tue.nl
Shashank Srivastava	s.srivastava@student.tue.nl

Eindhoven University of Technology

Contents

Abstract

In this project, we perform an experiment of Sentiment Analysis to see if it applies to predict the election result. Our approach is to collect a set of Tweets about a candidate in the election, then compute the sentiment score for the whole set, to see how much the community support him. To achieve the goal, we develop some machine learning mechanisms to classify the tweets into positive or negative ones. It is crucial to evaluate the models carefully before using them. We address the problem by training the classifiers by a published labeled Twitter dataset, using two different methods. We would also explore Topic Modelling and Clustering to get deeper insights to figure out the relevant topics that are most discussed in public about a particular candidate.

Chapter 1

Introduction

Elections are very important events in any democratic country. Every election is always preceded by a large number of opinion polls, analysis, predictions incorporating several statistical methods (e.g. stratified sampling etc.) by political analysts, pollsters, media organizations.

In this project, we focus on The United Kingdom general election of 2015, which was held on 7 May 2015 to elect the 56^{th} Parliament of the United Kingdom. The major candidates were David Cameron from Conservative, and Ed Miliband from Labor party. It is a representative example of the wrong prediction from Opinion Polls, which has a significant error comparing to the actual election result.

"Sentiment Analysis", also known as the "Opinion Mining" , is a really powerful approach to analyze people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as political parties, leaders, products, services, organizations, individuals, issues, events, topics, and their attributes. Sentiment analysis can be considered in many areas like business, products etc. but here we are going to focus on the importance in the political elections of any country.

In order to perform Sentiment Analysis, we have to build robust text classifiers. In this project, we experiment 2 well-known classifiers: Naive Bayes and SVM.

Chapter 2

Overall Architecture

2.1 Architecture : Supervised Machine Learning

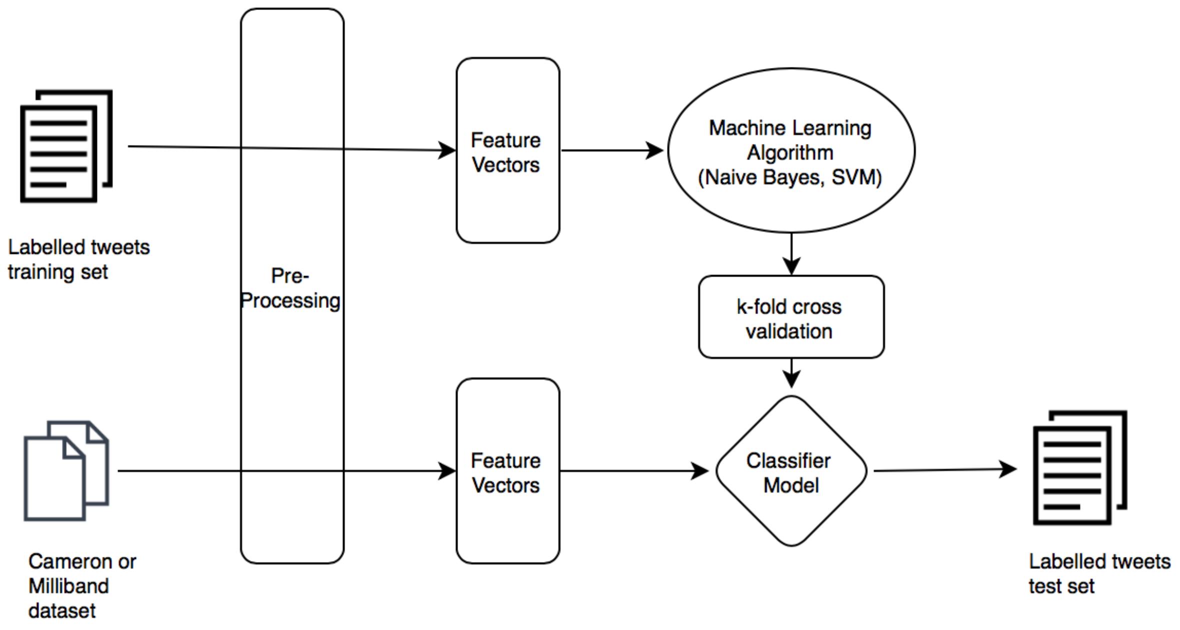


Figure 2.1: Architecture of Supervised Machine Learning

Figure 2.1 gives the architecture of supervised machine learning algorithm. The training set consists of *labelled* (positive or negative) tweets. Next step is to narmalize the tweets by using different pre-processing steps, and create feature vectors which can be used to create a *Classifier Model*. *k-fold cross validation* is done to verify the accuracy of the model.

Once we have obtained a good model, the model is used to label the tweets in the Cameron and Milliband dataset. This architecture is used for Naive Bayes and SVM (Support Vector Machine) supervised machine learning.

2.2 Architecture : Un-Supervised Learning

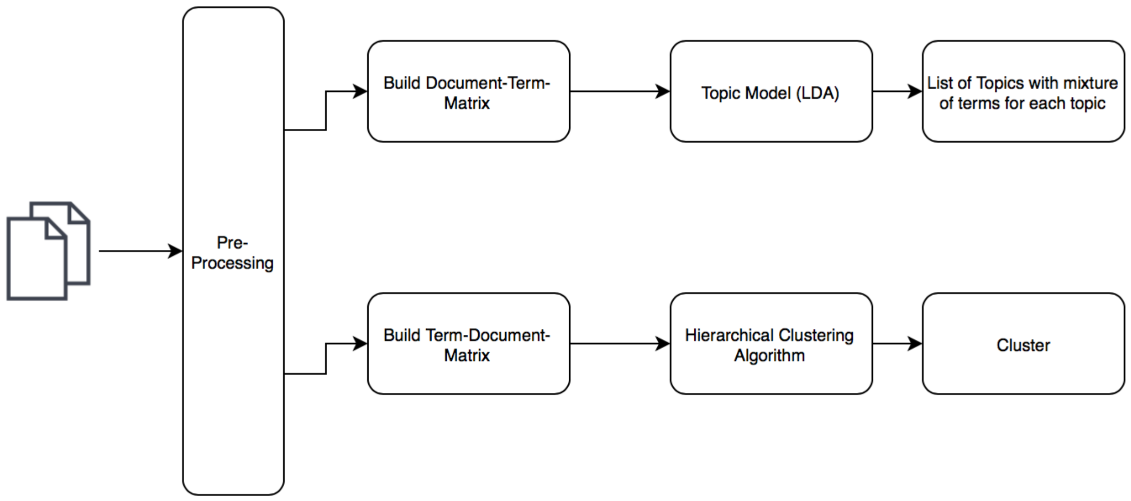

Pre-Processing
Build Document-Term-Matrix
Topic Model (LDA)
List of Topics with mixture of terms for each topic
Build Term-Document-Matrix
Hierarchical Clustering Algorithm
Cluster

Figure 2.2: Architecture of Un-Supervised Machine Learning

Figure 2.2 shows the overall architecture of clustering algorithm and topic modelling.

Chapter 3

Twitter Dataset and It's Pre-Processing

In this section, there will be a discussion on the data set used for the intended experimentation of sentiment-analysis. Further, the pre-processing methods and its necessities are discussed very briefly.

3.1 Twitter Dataset (Dataset of the Individual Candidates)

In this experimentation, the data set is created by extracting tweets from the Twitter website. The initially extracted data set contains a list of tweets, and each tweet has its meta-data like username, date of the tweet, the number of retweets, number of favorites. Tweets extraction was done using a web-crawler Java code, which extracts tweets within a time period and with specifies keywords. This process required the use of web-crawler because Twitter API does only allow tweets extraction for the past one week. The extracted tweets are stored in a CSV file which is used to in creating corpus for different learning algorithms.

Cameron Dataset

As explained above, the tweets are crawled from the Twitter website using keywords, and a time period for the tweets. For Cameron data set, the keywords used were 'Cameron','David_Cameron', etc..

Milliband Dataset

As explained above, the tweets are crawled from the Twitter website using keywords, and a time period for the tweets. For Milliband data set, the keywords used were 'Milliband', 'Ed Milliband', 'EdMilliband', etc..

3.2 Pre-Processing

User generated data present on the web is rarely present is a usable form for the sentiment analysis. Hence, Pre-processing techniques are important to normalize the text, and to decrease the size of the feature set which makes it suitable for the learning algorithms.

Removing URL

Tweets often contain hyperlinks shared by the twitter users. These links does not contribute to our sentiment-analysis. The links are removed from the tweets using regular expression.

Removing Hashtags '#'

Almost every tweet contains Hashtags that are used to relate a tweet to a subject, person, or/and topics etc.. For example, *#iPhone6*, *#FIFA*, *#TUEindhoven*. The function used in R to remove hashtags is:

```
removeHashTag <- function(x)
  gsub('#\\S+\\s*',' ', x)
```

The function uses regular expression to remove any hashtags.

Removing Handles '@'

Users can notify (or include) other users in their tweets by writing their usernames preceded by an '@' symbol. Often, the tweets have tagged users in it. The function used to remove handles is:

```
removeReference <- function(x)
  gsub('@\\S+', ' ', x)
```

Removing stop-words

Stop-words are the most commonly used words in a text in English language. There are no unique list of stop-words. In this experiment, the original list of stop words in *R's tm* library is removed from the data set to normalize the same.
Another observation is to filter the original list of stop-words, because their are R's library-defined stop-words which may prove to be useful in analyzing the sentiment of a tweet. For example, 'isn't' and 'not' are stop-words in the original list. But sentences like **he *isn't* a good person**, **he is *not* good in politics** are giving negative sentiments; whereas if we remove those stop-words, it might not be classified correctly. This analysis is done in the corresponding classification section of the report. R code used to remove stop words from the corpus is:

```
myCorpus <- tm_map(myCorpus, removeWords, stopwords("en"),
                                                lazy = T)
```

Removing Short Words

In this experiment, short words are considered as the words with length less than equal to 2. These words are mostly covered as *stop words*. Still there are few such words not removed as *stop words* and they do not contribute as positive or negative sentiment. Almost every word with *positive* or *negative* sentiment is at least three letters long in length. We wrote a function in R to remove such short words, which is:

```
removeShortWords <- function(x)
                         gsub("\\b[a-zA-Z0-9]{1,2}\\b", " ", x)
```

Removing Punctuation

Punctuation is heavily used in a language to make a written content meaningful and understandable. Tweets always contain number of punctuation marks which do not contribute to sentiment analysis because the classification (or clustering) algorithms run on words present in the tweets (documents). However, punctuation may change the sentiment of a text, but it is not trivial to implement. One such example is *tweet (sentence) with sarcasm*:
Tweet: "Really Cameron! You think you will be a great prime minister! LOL"
Above tweet would be classified to be a positive tweet, but if analyzed manually, it seems to be a negative sentiment because the tweet looks like a sarcasm. This report does the punctuation removal for any punctuation marks because the sarcasm detection is not included as part of this experiment. The R code used to remove punctuations is:

```
myCorpus <- tm_map(myCorpus, removePunctuation, lazy = T)
```

Strip White-spaces

Tweets are mostly written in a very poor way, and sometimes it contain extra spaces. It will be useful to remove excess white-spaces to get a better result from the algorithms used in sentiment analysis such that the algorithms can easily tokenize the words which is the very basic step in the algorithms.The R code used to remove extra whitespaces is:

```
myCorpus <- tm_map(myCorpus, stripWhitespace, lazy = T)
```

Convert to-lower

Once above pre-processing steps are executed on the data set (collection of tweets), and then the final step is to convert every word to lower. This will further normalize the collection. This processing step is important to have a uniform style of words in the tweets, and words like 'Great' and 'great' can be easily recognized to be the same words by the algorithms. This step is done using the R code:

```
myCorpus <- tm_map(myCorpus, content_transformer(tolower), lazy = T)
```

Chapter 4

IR Tasks

4.1 Dataset Retrieval (For Training Models) and Data Sampling

In order to perform the ML tasks, we need a good training set which contain reliable pre-labeled sentiment. Although self-labeling our own Twitter data is the best way to achieve such training set, the time limit does not allow us to do that. Instead, we look for public data sources that are verified by the community.

The Semeval dataset, which would be used to train the classification models, consists of three classes Positive, Negative and Neutral of total 13,500 tweets. There is a class imbalance in the dataset with 7017 tweets belonging to positive class, 3183 tweets belonging to the neutral class, and 3383 tweets belonging to the negative.

This class imbalance can have a detrimental effect on our classification model and can make it more biased towards a particular class (positive class in this case). To mitigate this potential problem ,we try various methods of sampling the data.

The R package caTools [11] is used to perform random sampling on the data.

We experimented with these sampling approaches:

a Random Distribution of 3 classes

b Equal distribution of 3 classes

c Random distribution of 2 classes.

d Equal distribution of 2 classes.

We found out that Random sampling with 2 classes gives positive results, and hence, the most of the classfication would be performed using this sampling technique.

4.2 Vector Space Construction

In order to apply the Machine Learning techniques, we have to convert the raw dataset to a vector space model. Depending on the later tasks, we convert the dataset either to a Term-Document Matrix or a Document-Term Matrix.

4.2.1 Term-Document Matrix

In this data structure, a term is represented by a vector of documents, where the weighting scheme is either tf or tf-idf. This data structure is helpful for doing the clustering and topic modeling tasks, as we would like to group the terms into the meaningful topics and relationships. In table 4.1, we show an example of the term-document matrix, built from a sub-sample of our dataset.

4.2.2 Document-Term Matrix

This data structure is very important to do the classification tasks. It is a form of the Bag-of-word model, where each document is modeled as a vector of terms, and the suitable values are assigned depending on the weighting scheme we use. In this project, we use 2 different weighting scheme: term-frequency (tf) and term-frequency.inversed-document-frequency (tf.idf).

Term-Frequency is simply the number of occurrences of a term in a document. We use the toy data set in table **??** to make an illustration of a Term-Document Matrix.

Table 4.1: Sample Term-Document Matrix of 6 terms and 6 documents. The cell values are the if.idf values of the term corresponding to the document.

school	0.0000000	0.0000000	0.0000000	0.0000000	0.2555329	0.0000000
season	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
see	0.2768273	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
senses	0.0000000	0.0000000	0.0000000	0.0000000	0.2555329	0.0000000
shane	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
similar	0.0000000	0.3321928	0.0000000	0.0000000	0.0000000	0.0000000

4.3 Clustering Terms

4.3.1 Description

Cluster analysis divides the data into meaningful and useful groups. While doing the sentiment analysis for predicting elections, clustering methods have been used to get more clarification. But this has not helped much in predicting the outcomes of the election but we can get some words which were trending during that time. Clustering has many algorithms but we have used hierarchical clustering method to make the dendrogram, this clustering refers to a group of closely related clustering techniques and produce a hierarchical clustering by starting with each point as a singleton cluster and simultaneously combine the two closest clusters until a single cluster will be formed. The distance between words has been calculated with different techniques, for our analysis we used *Euclidean, Manhattan and Maximum* distance with Ward's minimum variance method and UPGMA method (average method) of hierarchical clustering

4.3.2 Methodology

In the first step, preprocessing has been done for normalized the data. Then corpus has been created with the normalized data. Term document matrix has been created using unigram and bigram. `'hclust'` method in R has been used for doing hierarchical clustering. *'hclust'* provides many arguments to do the clustering with different methods. Some of the arguments are described below.

```
hclust(d, method = "ward")
```

Here, d = Distance matrix (In this project, has been calculated with Euclidean distance, Manhattan distance and Maximum distance)

method = methods to do the hierarchical clustering (In this report it has been done using Ward's minimum variance method and UPGMA method (average method))

The height of dendrogram represents the distance between two words, if it is more then the words have less similarities. In first scenario, cluster has been made using unigrams which didn't give some clear indication of the situation. Some random words appeared with which it was difficult to catch the trend of the tweets. Then bigram has been used for the analysis.

Bigrams have been used to make the dendrogram. Two methods of hierarchical clustering have been used those are Ward's minimum variance method and UPGMA method (average method) with different distance matrices.

Ward's minimum variance method is implemented recursively by a Lance–Williams algorithm [5]. The Lance–Williams algorithms are an infinite family of agglomerative hierarchical clustering algorithms which are represented by a recursive formula for updating cluster distances at each step (in each step a pair of clusters is combined). At each stage, it is necessary to optimize the objective function (find

Figure 4.1: Dendrogram : Using Maximum distance and UPGMA method

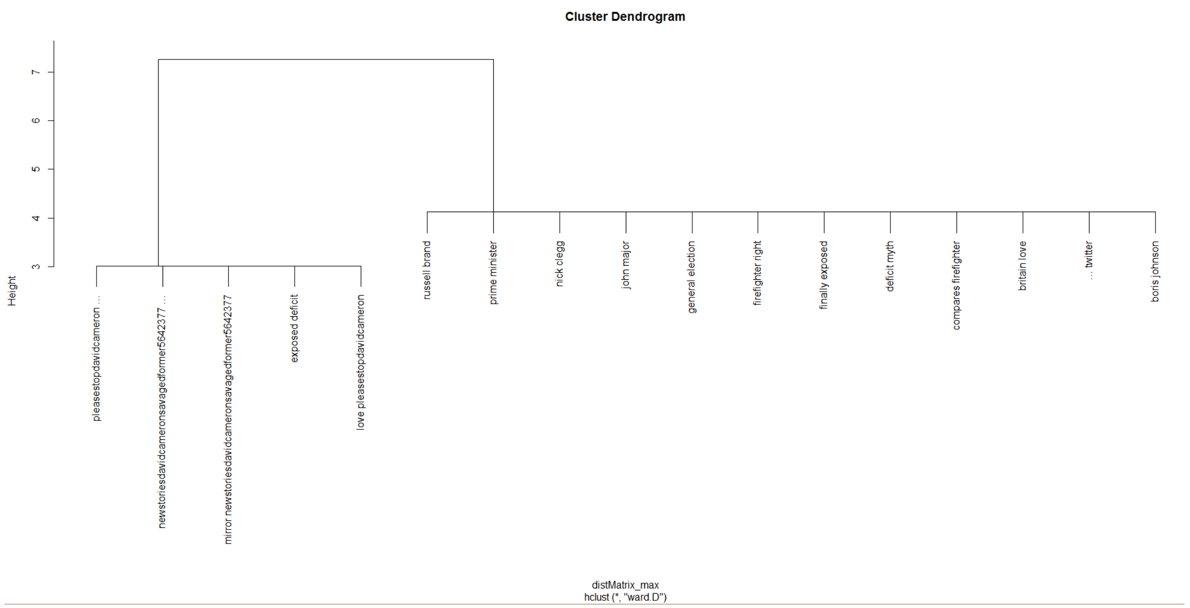
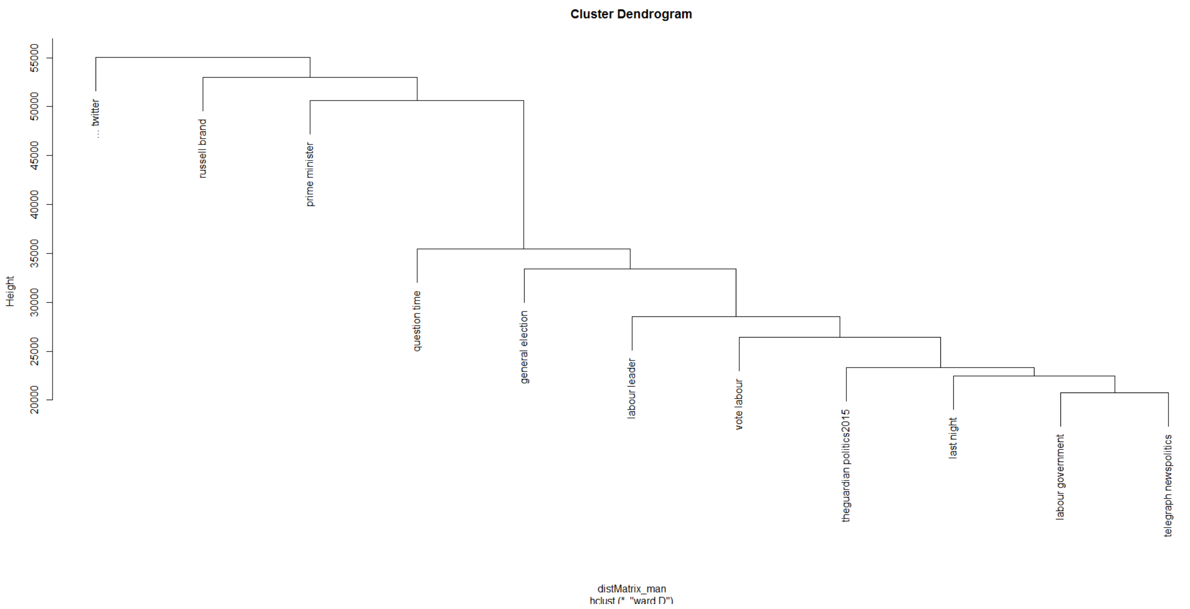
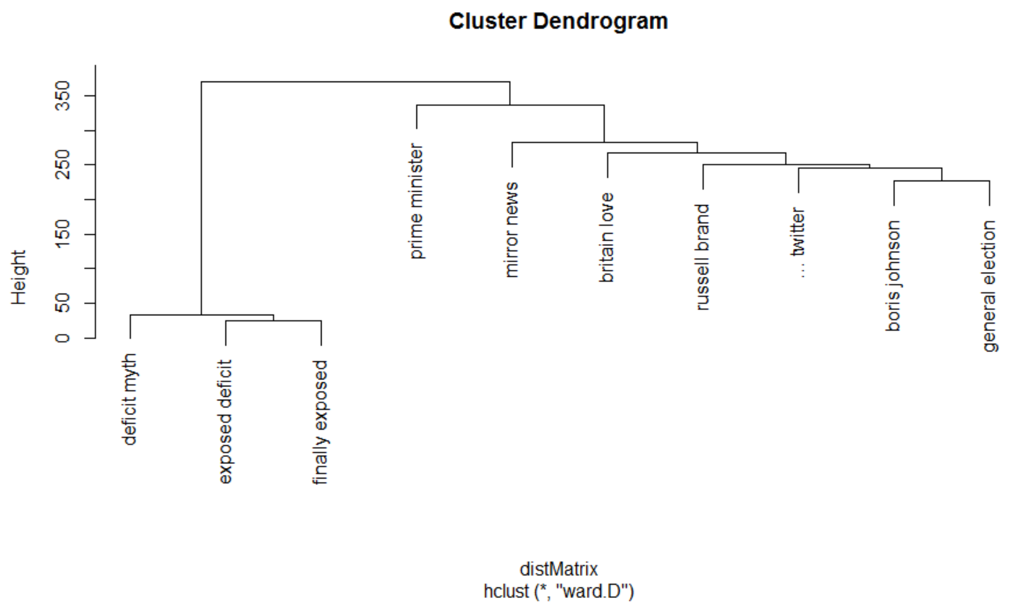


Figure 4.2: Dendrogram : Using Manhattan distance and Ward method



the optimal pair of clusters to merge). The recursive formula simplifies finding the optimal pair in cluster.

Figure 4.3: Dendrogram : Using Euclidean distance and Ward method



The UPGMA algorithm constructs a dendrogram. At each step, the nearest two clusters are combined into a higher-level cluster. The distance between any two clusters A and B is taken to be the mean of all distances between pairs, because it takes mean which is basically the average of the distances so this method is also called average method.

4.3.3 Analysis of clusters

Clusters are the groups of the data objects based on the information found in the data, these describe the objects and their relationship. If the similarity within a group is more and difference with other groups is also more then it will be good for getting more informative clusters. But the tweet dataset is too sparse because twitter users use different type of words to express the same sentiments. The terms, which are present in the dendrogram, have appeared after giving sparse threshold more than 0.99, it means these are only less than 1% of overall dataset.

Using above mentioned methods, some meaningful words appeared. We searched those word in the google to get more information. For example: the cluster in 4.2 is made by Manhattan distance, for the Miliband set we can see the *"russell brand"*, *"twitter"* and *"prime minister"* are very near in dendrogram, when these words have been searched on google as well as in the dataset, it has been found that during that time Russell Brand interviewed Ed Miliband and urged his fan to vote for

Miliband for prime minister.[10]

"Ed miliband and Russell brand vine is absolute genius"

In the Cameron data set we have again run the hierarchical clustering and prepared dendrogram using lesser sparse threshold 4.3, the terms *"deficit myth"* and *"finally exposed"* were near and these have enough distance with "prime minister". When these terms were being searched on google then we found that an old news article from the year 2012,[1] which was against Cameron had become trending during the 2015 elections.

When we make dendrogram using Manhattan distance and Euclidean distances, these have been appeared similar but dendrogram using Maximum distance is different that others, it kept most of the words in same cluster group. It has been happened because maximum distance has been calculated using the maximum co-ordinate. For example, in a 2-D space, if we calculate distance between the point (1,0) and the origin (0,0) is always 1 according to the usual method, but the distance between the point (1,1) and the origin (0,0) will be 2 under Manhattan distance, $\sqrt{2}$ under Euclidean distance, or 1 under maximum distance. It may be helpful if the sentences or words with similar meaning will be given some standard sentences or words. Eg. *"beautiful"* has synonyms *"attractive", "pretty", "handsome", "good-looking", "nice-looking", "pleasing", "alluring", "prepossessing"* , if these words appear in different sentences of a very big dataset then by the clustering but cluster can be more meaningful if these synonyms will be represented by one standard word then there is a chance to reduce the sparsity of the dataset.

4.4 Topic Modeling

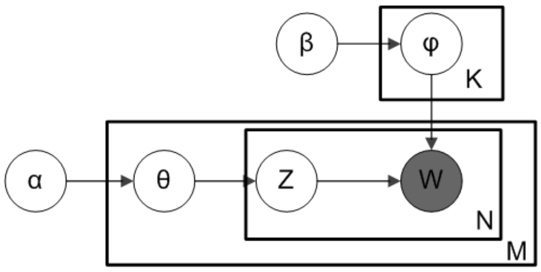
4.4.1 Description

Topic modeling [3] is done using a set of algorithms that extracts *set of topics* that best describes a collection of document(s). This helps in summarizing a large collection of documents which can be further used to categorize a collection of documents. Different analysis can be carried out using topic modeling; like, in this study, it is used to find *descriptive* and *discriminative* topics for political candidates 'Cameron' and 'Milliband'. This could help in finding out the views of the people about their political candidates. This will add more information to the sentiment analysis done by using the classification algorithms by giving us a list of topics which are most discussed among the people.

4.4.2 LDA : Topic Model

LDA (Latent Dirichlet Allocation) is a topic model used in topic modelling. It represents the documents as mixture of topics to get words with certain probabilities. Each document is assumed to be characterized by a mixture of topics. This is similar to bag-of-words model. From the figure 4.4:

Figure 4.4: LDA : Topic Model



- K is the number of topics
- N is the number of words in the document
- M is the number of documents
- α is the dirichlet-prior concentration parameter on the per-document topic distributions
- β is the dirichlet-prior concentration parameter on the per-topic word distribution
- Z(i,j) is the topic of j^th word in i^th document
- W(i,j) is the word to which above Z(i,j) topic is associated

Following is a brief discussion on LDA learning to get the idea behind its working. Suppose we have a collection of documents and k topics. Now, the LDA will learn topic distribution of each document, and words associated to the topics. This could be done using Gibb's Sampling, which is done in following steps:

- for each documents d:
  - randomly assign one of the k topics to each words in the document d.
- for each document d; and for each word w in d
  - for each topic t, calculate
    P(topic t | doc d) and
    P(word w | topic t)
    Here, we assume that all other words are assigned correct topic, and we refine the topic assignment for the current word. Now, the current word w is assigned a new topic t according to
    P(topic t | doc d) * P(word w | topic t)

- Above step is repeated a large number of times until a "near" stable topic assignment to the words is obtained. This assignment is used to find mixture of topics assigned to a document, and proportion of words assigned to each topic for a document.

4.4.3 Implementation and Evaluation

We build a term model using LDA algorithm as discussed in the previous section.The R package *topicmodels* was used to perform LDA for constructing a topic model. The model is built using different combination of features with weighting techniques for both the political candidates:

- Unigram with tf weighting scheme
- Bigram with tf weighting scheme

Effect of Unigram with tf weighting scheme

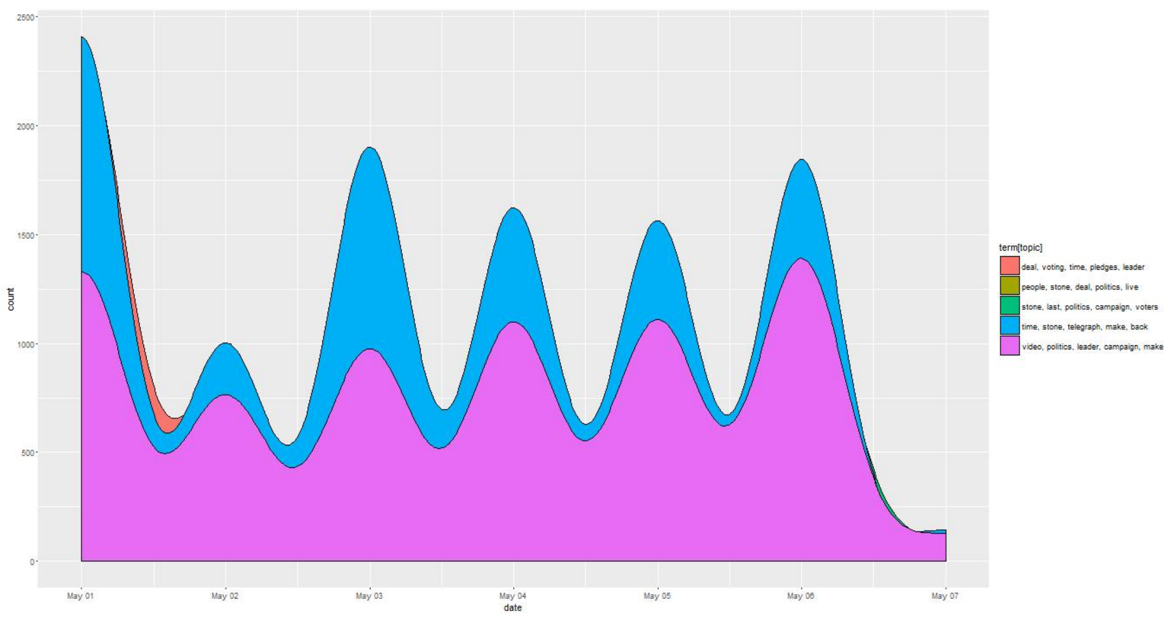


Figure 4.5: Topic Modelling on Cameron Data Set with Unigram feature and tf weighting scheme

Figure 7.2 and Figure 4.6 show the topic modelling result on Cameron and Miliband datasets respectively. As we can see from the topic modelling result that it is hard to decide a relevant topic for most the set of terms. It gives very little idea about the discussed topics at the time of elections. There are different possible reasons that the results are not very meaningful, and one of the main reason being the dataset is very *sparse*.

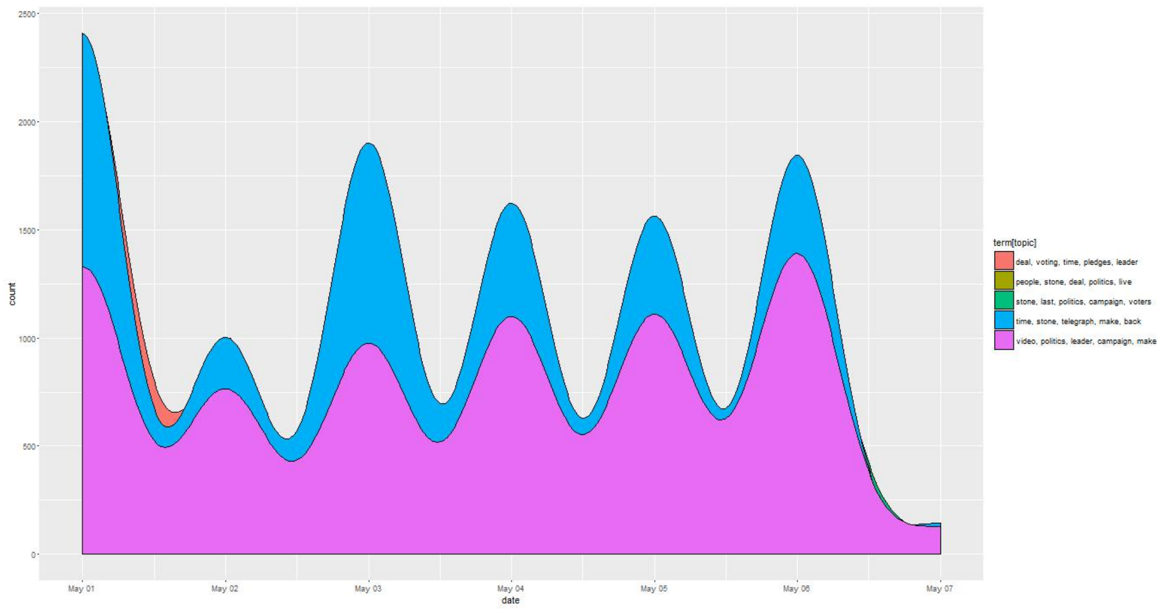


Figure 4.6: Topic Modelling on Milliband Data Set with Unigram feature and tf weighting scheme

Conventional topic models implicitly capture the document-level word co-occurrence patterns to reveal topics, and hence the results are often not meaningful because of severe data sparsity in short documents. The occurrences of words in short document play less discriminative role while in lengthy documents the model has enough word counts to find the relation between words. Thus, the results of topic modelling are often clear in case of lengthy document with less sparse data.

To overcome the problem due to sparse data in the original Cameron dataset, we included each tweet multiple number of times which is equal to the number of retweets. It will increase the number of times different words appear in the collection of tweets. And this may help in removing the severe data sparsity from the collection. Figure 4.7 shows the result of topic modelling on this extended dataset of original Cameron dataset. Still the result does not give all meaningful topics, but it seems to be better than the previous result. As we can see in the figure 4.7 that the first topic talks about *the birth of a baby by Duchess of Cambridge; and Duke of Cambridge*. This topics is discussed the most somewhere around 2nd May 2015, which is also the birth date of the baby; and we found a number of tweets at that time that were discussing Cameron's wishes for the same. There were a lot of people talking about Cameron and this topic at the time.
Another list of terms is the topic in color Blue. This topic mainly comes from the David Cameron's tweets one day before the election. It was to support their political party, and vote for them the next day. Figure 4.8 shows the tweet, and it was

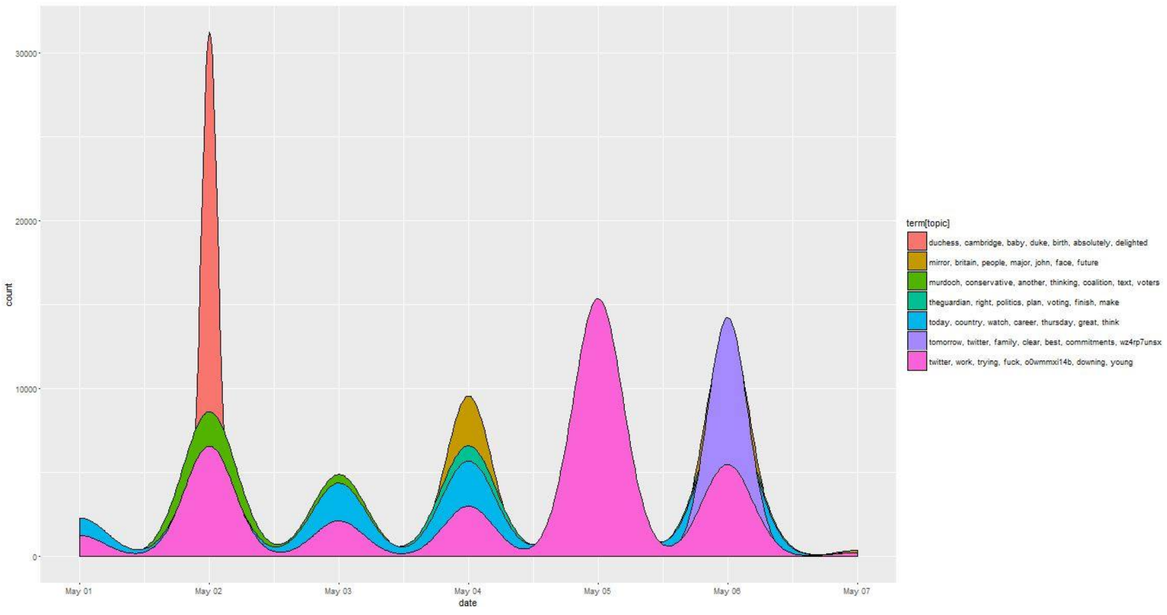


Figure 4.7: Topic Modelling on Cameron Dataset after including the tweets multiple number of time which is the number of retweets. The feature used is unigram with tf weighting scheme

(re)tweeted by number of users on the twitter. This topic shows the high support of people at around 6th May.

Another observation is that there are terms which points to the name of media organizations, like The Guardian, Mirror. These type of tweets also point to some news articles that discussed the politics related to Cameron at that time. It would have been very helpful to include the content of the news article in our topic modelling, but it was out of scope of the original plan of this experimentation.

Effect of Bigram with tf

We tried topic modelling with bigram feature and *tf* weighting scheme on both Cameron and Milliband dataset. But the results in Figure 4.9 and Figure 4.10 show very few distinct terms because of severe sparsity. It is known that the texts are very short, and word distribution is very sparse. This gives a less meaningful result when we used **tf** as the weighting scheme. The other weighting scheme *tf-idf* can be predicted to perform bad on short text documents because it further normalizes the word distribution. We tried the to get the results with *tf-idf* but, as expected, it was even less meaningful than the previous results with *tf*.

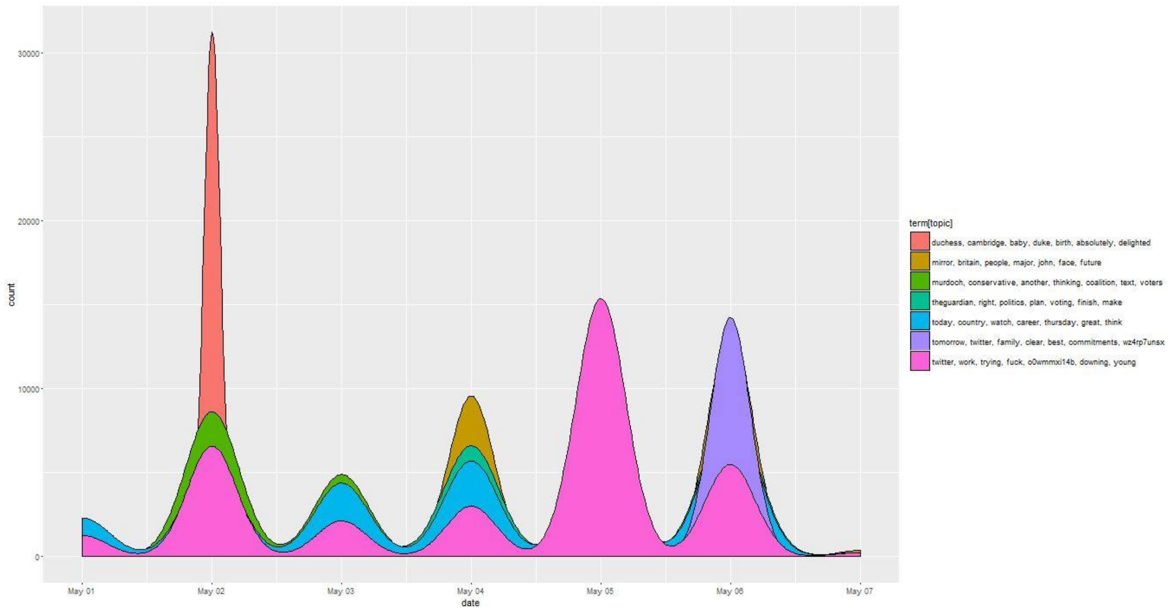


Figure 4.8: Topic Modelling on Cameron Dataset after including the tweets multiple number of time which is the number of retweets. The feature used is unigram with tf weighting scheme

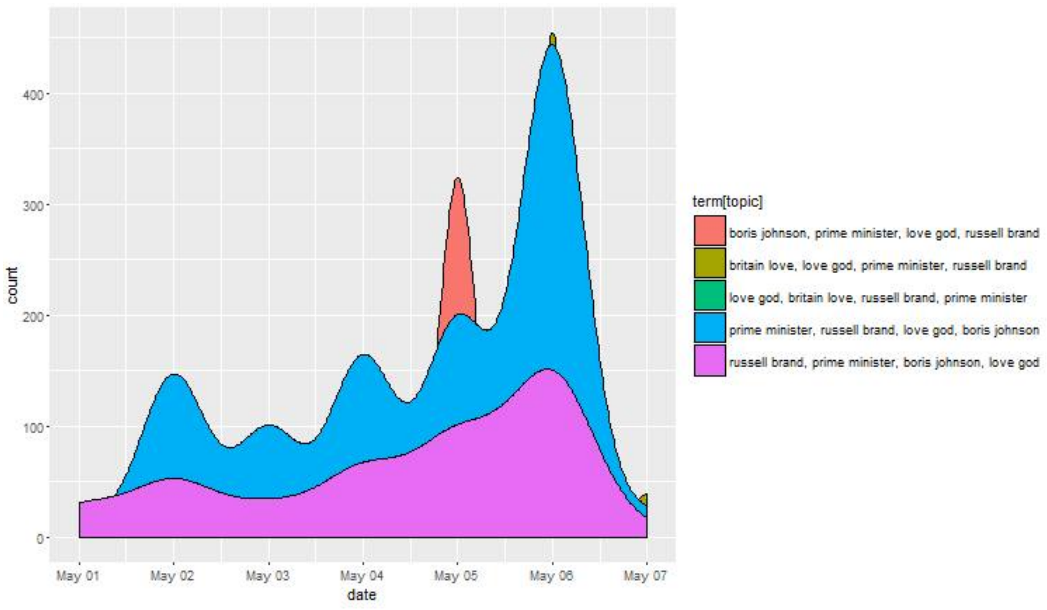


Figure 4.9: Topic Modelling on Cameron Dataset with bigram feature and tf weighting scheme

Conclusion on Topic Modelling

Short texts have been the main format of information exchange on social media for a long time now. Users try to convey their thoughts and sentiments through these

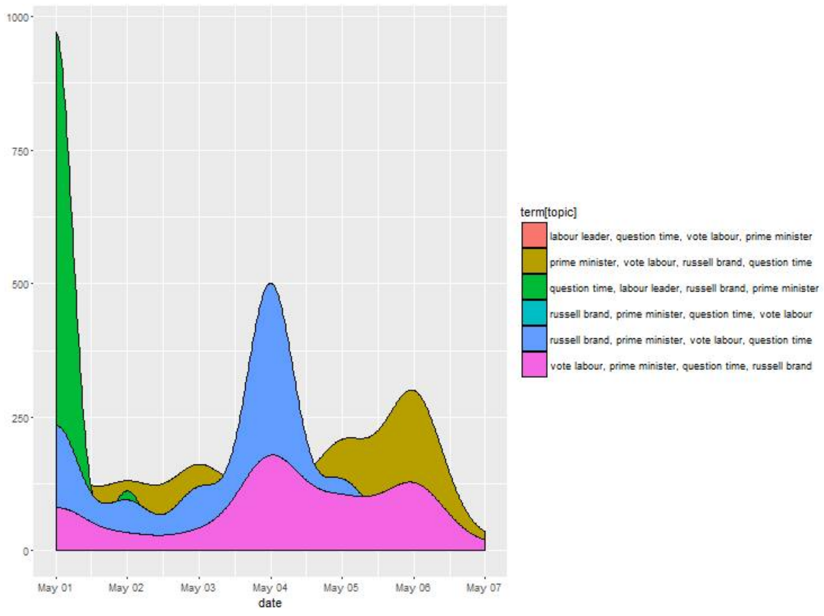


Figure 4.10: Topic Modelling on Milliband Dataset with bigram feature and tf weighting scheme

messages; hence, it becomes a useful source of topic modelling. But the challenge remain in tackling sparsity and imbalance of the short text while performing topic modelling. In this experiment, we tried different features and weighting techniques to get meaningful topics from the datasets, but the applied techniques are not able to tackle the main issues with the short text.

Chapter 5

DM Tasks

In order to correctly predict the sentiments of the crowd toward the 2 candidates, we have to construct robust learning models. In this section, we described various analysis approaches that we perform to build and evaluate the models we use. We focus mainly on 2 well-known classifiers: Naive Bayes and SVM.

5.1 Number of Classes

Our purpose is to classify the tweets into 3 classes: "positive", "neutral", and "negative". As described earlier, we use the dataset from SemEval 2014-subtask A-B [9], which contains 3383 labeled negative tweets, 3183 labeled neutral tweets, and 7017 labeled positive tweets.

Our first experiment comes with the Naive Bayes implementation on the whole dataset. We perform 10-fold cross-validation using unigram, tf.idf, stemming, filtering using Hu&Liu Corpus, and get the result as shown in table 5.3. We show only the results for positive and negative tweets as they are much more important than the neutral class.

We can see from table 5.3 that the accuracy achieved is very low. Especially, the result of the positive class is also not stable, represented by the high standard deviation. In order to explain this, we have to take a look at the actual dataset.

For instance, let's take a look at the following tweet.

> **Example 5.1 (*)**
> Tweet: "But now maybe Congress will do the right thing after all. Eagerly waiting for this weekends blockbuster. The interview in Wall Street Journal about zecco. Can't wait to follow my stats tools"

This one is defined as "positive", but the algorithm returns "neutral". After pre-

Table 5.1: Example vector

	blockbuster	eagerly	right
tf.idf	0.436	0.444	0.319

Table 5.2: Example Confusion Matrix

pred	negative	neutral	positive
negative	55	1	40
neutral	13	3	21
positive	280	25	562

processing and filtering, the vector of this tweet contains the terms in table 5.1, all other terms are 0.

The Naive Bayes model for those terms are:

- **blockbuster**: 0.01914589 negative, 0.02878913 neutral, 0.01718046 positive
- **eagerly**: 0.01945737 negative, 0.02925750 neutral, 0.01212602 positive
- **right**: 0.07030322 negative, 0.06795774 neutral, 0.05800842 positive

The neutral probabilities are very high for those term, especially for "blockbuster" and "eagerly". That's why the tweet is classified as neutral. Besides, taking a look at the confusion matrix in 5.2, we can see that it is difficult to classify the neutral class correctly, although this is the best result we achieve after tweaking a lot of arguments and data used.

Indeed, there are no clear indicator of neutral tweets, and the number of neutral words are much more than the number of clearly positive or negative words. Additionally, the neutral tweets don't contribute much to our prediction result because we just assign scores for the negative and positive ones. For this reason, we decide to remove the neutral class from our analysis in the later sections.

Table 5.3: Cross-Validation result of Naive Bayes Model. Setting: 3 classes, Unigram, tf.idf, stemming, filtering using Hu-Liu Corpus

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	42.34/1.76	46.76/33.16	53.67/33.17	49.77/33.28
Negative		75.47/2.24	66.18/5.94	70.40/3.71

5.2 Naive Bayes

5.2.1 Definition

Naive Bayes is a classical classifier, which we can find in most of statistics book nowadays. It is based on the Bayes theorem, stated in equation 5.1.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{5.1}$$

In our project, we would like to compute the probabilities that a tweet is positive and negative, then decide the suitable label, as being shown in equation 5.2.

$$\begin{aligned} P(class|tweet) &= \frac{P(tweet|class) \cdot P(class)}{P(tweet)} \\ &= P(tweet|class) \cdot P(class) \\ \text{where } P(tweet|class) &= \prod P(term \text{ in } tweet|class) \end{aligned} \tag{5.2}$$

$P(term \text{ in } tweet|class)$ and $P(class)$ are trained from a training set. In this project, we use 2 different weighting methods - tf and tf.idf - to contribute to the computation of those prior probabilities, instead of simply counting the number of occurrences. We also assume that our data contains tweets only, so $P(tweet) = 1$. Equation 5.2 shows that Naive Bayes assumes the terms are independent from each other, which is a weak assumption. However, Naive Bayes is still useful if the data is large enough. We will experiment it in the later sections.

A problem with Naive Bayes is if P(term | class) = 0 for a term, then the whole probability of the tweet containing that term will be 0. This problem can be solved by apply Laplace smoothing, which adds a small smoothing parameter to the prior probabilities.

5.2.2 Implementation

In this project, we mostly use R [8] and Python [7] to implement the experiment. For Naive Bayes, we use the package e1071 [4] to implement the trainer and experiment with different settings.

5.2.3 Evaluation

We train our models using different combination of features, as well as different sampling techniques, which are:

- With and without stemming
- With and without stop-word removal

- Tf or Tf.Idf weighting scheme
- Bigram, Unigram, or a mixture between the two
- Filtering the dimension using Hu&Liu [2] Corpus, or extracting the most-frequent terms

For all the analysis illustrated below, we use the training set with random distribution from 2 classes - "positive" and "negative".

Effect of Stemming

Our assumption is that stemming (as well as stop-word removal) can have effects only if we filter the terms using most-frequent-term method. If we use Hu-Liu corpus, then only the terms appearing in the corpus are selected, then there should be no differences between using stemming and not using stemming.

In this project, we use Porter's Stemmer [6]. Our experiments show that stemming provides a slightly better result. To illustrate that, we show a comparison of Naive Bayes models on stemmed and non-stemmed data in table 5.4 and table 5.5. We can see that the accuracy of Naive Bayes is more stable with stemming, indicated by the low standard deviation

Table 5.4: Naive Bayes statistics with unigram, if.idf, Most-Frequent-Terms, stop-word removal, without Stemming

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	69.61/31.98	80.11/3.76	42.82/3.81	63.99/1.74
Negative		53.30/1.16	72.80/2.31	53.79/2.91

Table 5.5: Naive Bayes statistics with unigram, if.idf, Most-Frequent-Terms, stop-word removal and Stemming

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	59.96/1.55	80.28/3.99	43.09/4.05	64.42/1.90
Negative		53.82/1.21	72.80/3.16	54.01/3.32

Effect of Stop-word Removal

As described earlier, stop words are the common words in a language, which are assumed not to have much benefit in sentiment classification. In this section, we

Table 5.6: Naive Bayes statistics with unigram, tf.idf, Most-Frequent-Terms, Stemming, and no stop-word removal

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	59.78/1.51	80.48/4.07	43.01/4.00	64.08/1.98
Negative		53.30/1.91	73.37/3.87	54.08/3.32

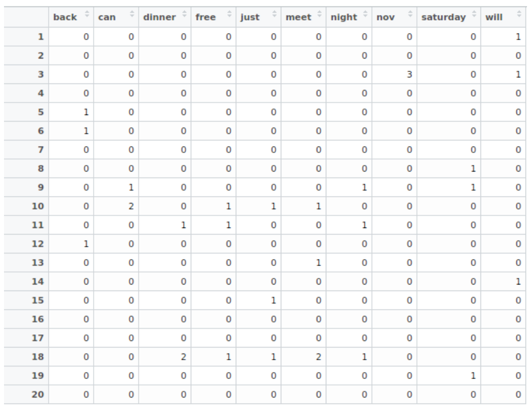
would like to verify this hypothesis by executing a cross-validation of Naive Bayes when there is no stop-word removal.

Similar to stemming, we experiment stop-word removal by applying the Naive Bayes trainer with most-frequent-term as the filtering technique. In table 5.6, we show the statistics of the cross-validation when apply Naive Bayes with unigram, Most-Frequent-Term, Stemming, and no stop-word removal. Comparing it with table 5.5, we see that the results are mostly the same, which is consistent with our assumption.

Different Weighting Techniques

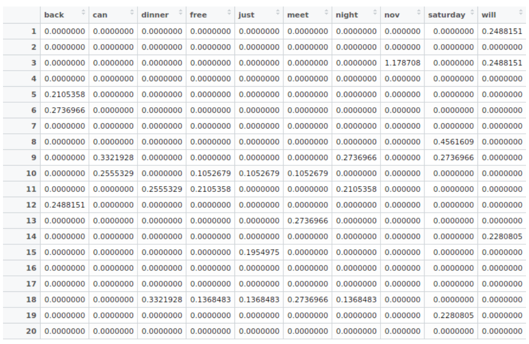
Our data consists of only short documents (tweets), so it is very likely that a term occurs only once in a documents. This results in the fact that most of tf.idf values are actually idf.

To see the differences between the 2 weighting methods, we sample 20 random tweets in our paper, filter out the terms that occur less than 3 times, then build the document-term matrix using tf and tf.idf as shown in figure 5.1.



	back	can	dinner	free	just	meet	night	nov	saturday	will
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	3	0	1
4	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	1	0	0	0	0	1	0	1	0
10	0	2	0	1	1	1	0	0	0	0
11	0	0	1	1	0	0	1	0	0	0
12	1	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	1
15	0	0	0	0	1	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0
18	0	0	2	1	1	2	1	0	0	0
19	0	0	0	0	0	0	0	0	1	0
20	0	0	0	0	0	0	0	0	0	0

(a) DTM by TF



	back	can	dinner	free	just	meet	night	nov	saturday	will
1	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.2488151
2	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
3	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.178708	0.0000000	0.2488151
4	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
5	0.2105358	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
6	0.2736966	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
7	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
8	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.4561609	0.0000000
9	0.0000000	0.3321928	0.0000000	0.0000000	0.0000000	0.0000000	0.2736966	0.000000	0.2736966	0.0000000
10	0.0000000	0.2555329	0.0000000	0.1052679	0.1052679	0.1052679	0.0000000	0.000000	0.0000000	0.0000000
11	0.0000000	0.0000000	0.2555329	0.2105358	0.0000000	0.0000000	0.2105358	0.000000	0.0000000	0.0000000
12	0.2488151	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
13	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.2736966	0.0000000	0.000000	0.0000000	0.0000000
14	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.2280805
15	0.0000000	0.0000000	0.0000000	0.0000000	0.1954975	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
16	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
17	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000
18	0.0000000	0.0000000	0.3321928	0.1368483	0.1368483	0.2736966	0.1368483	0.000000	0.0000000	0.0000000
19	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.2280805	0.0000000
20	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0.0000000	0.0000000

(b) DTM by TF.IDF

Figure 5.1: Example of Vector Space (DTM)

Indeed, this example is representative in our dataset. Because it is quite sparse, and each document is very short, most of tf values are 1. As a result, the value of

	hit	michael	sudden
tf.idf	0.432	0.381	0.597
positive prob	0.024	0.021	0.00027
negative prob	0.032	0.045	0.0

the cells in tf.idf matrix are actually idf. As an example, in document 11, the term "night" has the same tf with the term "dinner", but they have different idf values. In our context, these 2 terms may not be very different in the effects to the sentiments toward the 2 political candidates. Thus, we have an intuition that using tf is more suitable in our context. However, the comparison between table 5.5 and table 5.7 shows that the results with tf and tf.idf are quite similar. It is probably because our data is too sparse.

Effect of different features

The first language model that we use is unigram, that is, every document is converted into a vector of single words where each value is either the tf or tf.idf of the word in that document. As we can see in the cross-validation result in table 5.7, the precision of the negative class is low, which means there are some positive tweets which are detected as negative, such as the following example.

> **Example 5.2 (*)**
> Tweet: "Watching Michael Ball's performance of Empty Chairs At Empty Tables in the 10th anniv, and hit with SUDDEN EXTREME FEELS OGOD PERFECTION"

After preprocessing and most-frequent-term filtering, the unigram vector of this tweet contains 3 non-zero terms: "hit", "michael", and "sudden". According to the probabilities of the model, this tweet is classified as "negative".

Table 5.7: Naive Bayes statistics with unigram, tf, Most-Frequent-Terms, Stemming, and stop-word removal

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	61.81/1.44	82.92/4.03	44.85/3.35	65.73/2.53
Negative		54.47/1.94	77.11/2.81	56.61/2.50

The second language model that we use is bigram, which is a collection of word pairs that co-occur frequently in the dataset. The motivation is from the example that we show in the previous section, where a positive tweet is filtered out and only one negative term is left. Our intuition is that bigram can help avoid such situation.

Because of our hardware limitation, we cannot train the Naive Bayes model with bigram on the whole dataset (containing 10400 tweets). Instead, we have to filter the dataset to a smaller one with 6821 tweets, including 3383 negative tweets and 3438 positive tweets. In contrast to our expectation, the performance of Naive Bayes using bigram is low, as described by the cross-validation result in table 5.8.

In our data, there are only about 600 bigrams after we perform most-frequent-bigram extraction. The same situation as in the unigram example occurs. It seems that the number of bigrams is not enough to describe all the properties of our data. We come up with the final idea about feature extraction, a combination between unigrams and bigrams. That is, we select both the most frequent terms and the most frequent bigrams as the features. Then we build the Vector Space using the df values of them. In table 5.9, the cross validation result is good and quite stable. The ROC curve in 5.2 indicates that it is a good model.

Table 5.8: Naive Bayes statistics with bigram, tf, Most-Frequent-Terms, Stemming, and stop-word removal

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	56.13/4.98	65.22/7.02	53.52/7.44	38.82/2.71
Negative		27.74/2.20	84.82/3.43	65.41/6.12

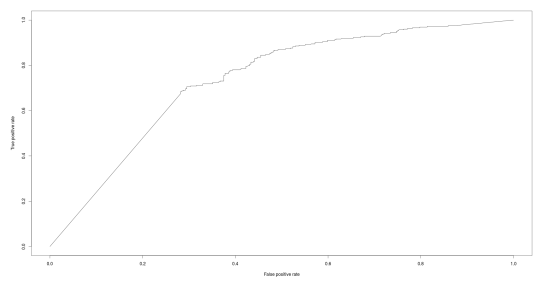


Figure 5.2: ROC Curve for Naive Bayes Model with mix feature.

Effect of different filters

Our original data consists of about 12000 terms, which can cause some problems. a) The matrix size can cause out-of-memory issues. b) There are many noises affecting the classification result. To solve this problem, we consider 2 approaches: extracting the most frequent terms, and filtering using a sentiment-labeled word list.

Table 5.9: Naive Bayes statistics with mixture of unigram and bigram, tf, Most-Frequent-Terms, Stemming, and stop-word removal

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	70.03/1.72	68.20/14.53	69.12/12.31	69.43/9.20
Negative		72.27/2.54	67.67/1.63	67.86/6.04

In the first approach, we count the number of occurrences of each term (unigram or bigram) in the whole collection, then extract the terms which the number of appearances is big enough. In this project, we use the threshold as 15 as it is the best balance between the number of features and our hardware capability. The drawback of this approach is that the filtered term list tends to contain arbitrary terms which do not have clear sentiment. Examples of Naive Bayes classifiers using most-frequent terms have been described in the the previous sections.

The second approach is to use a pre-defined dictionary of positive/negative words to indicate which words should be kept. We use the large Corpus developed by Hu&Liu [2], which is quite well-known. In this section, we will try to evaluate the hu-liu filtering method in the best model we have discovered so far: the mixture of unigram and bigram. We filter both the bigrams and the unigrams according to the Corpus. The result in table 5.10 is not as good as the most-frequent-term method shown in table 5.9.

The drawback of this approach is the limitation of the sentiment word list. Although it is a large dataset, it still lacks a lot of words, definitely. There are a lot of tweets, after pre-processing and feature extraction, contains only 1 or 2 non-zero terms, which causes the bias in assigning probabilities.

Table 5.10: Naive Bayes statistics with mixture of unigram and bigram, tf, Hu&Liu, Stemming, and stop-word removal

Mean / Standard Deviation	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Positive	60.67/7.78	57.19/15.16	68.10/13.22	67.03/11.95
Negative		84.56/1.85	36.31/3.58	46.94/46.94

5.3 SVM

5.3.1 Definition

Support Vector Machine (SVM) are large-margin, rather than probabilistic, classifiers, in contrast to Naive Bayes that samples hyperplanes which separate between two or multiple classes. Eventually, the hyperplane with the highest margin is retained, where "margin" is defined as the minimum distance from sample points to the hyperplane. The sample point(s) that form margin are called support vectors and establish the final SVM model.

The main motivation of using SVM for classification was to utilize its kernel trick to transform the data in a higher dimension to more efficiently perform classification between Positive and negative classes.

5.3.2 Implementation

Similar to Naive Bayes, we use the R package "e1071" [4] to implement and test SVM. The only difference is the dataset. SVM is a very asymptotically slow algorithm ($O(n^2)$), and our hardware power is limited. If we use the full SemEval 2014 training set, we always get "Out-of-memory" problem. As a result, we decide to train SVM on only half of the dataset, which is randomly sampled from the original data. We also use limited parameters and only the linear kernel.

The linear kernel is used as it is much faster and there are only a few parameters to optimise,the Cost C being one of them,which is responsible for the soft margin of the hyperplane.Smaller value of C corresponds to a greater soft margin.Limited Experimentation was carried out with different values of C (5,10,15,20) due to limited computing power and time constraints.

Assigning a cost of 10 to the SVM parameters gave better results and this setting was furthur used to build a model.

5.3.3 Evaluation

In order to evaluate the SVM performance on our training data, we perform a 10-fold cross validation with 2 different settings, which is shown in table 5.11 and table 5.12.

For all the analysis illustrated below, we use the training set with random distribution from 2 classes - "positive" and "negative".

Table 5.11: SVM statistics with unigram, tf, Most-Frequent-Terms, Stemming, and stop-word removal

Mean/Standard Deviation	Accuracy(%)	Precision(%)	Recall(%)	F1-Score(%)
Positive	68.72(2.26)	69.14/14.15	68.62/14.10	68.14/3.75
Negative		68.25/6.61	71.47/6.85	69.22/4.09

As illustrated in the table, stemming and stop-word removal do not have much influence on the classification accuracy. Comparing to table 5.4, the result for the negative class is greatly improved. Besides, SVM result is more stable, represented by the low standard deviation of the cross-validation statistics.

Table 5.12: SVM statistics with unigram, tf, Most-Frequent-Terms, and without Stemming, and stop-word removal

Mean/Standard Deviation	Accuracy(%)	Precision(%)	Recall(%)	F1-Score(%)
Positive	68.79(1.71)	80.50/11.60	62.91/4.82	70.63/3.43
Negative		58.57/9.48	77.49/8.50	66.72/2.58

5.4 Prediction

In this section, we try to use our best model so far, which is the Naive Bayes model with mixture between unigrams and bigrams, to predict the sentiments of the Cameron and Miliband dataset. In figure 5.3, it predicts that Miliband has more percentage of positive sentiments than Cameron. This result is not the same as the election outcome.

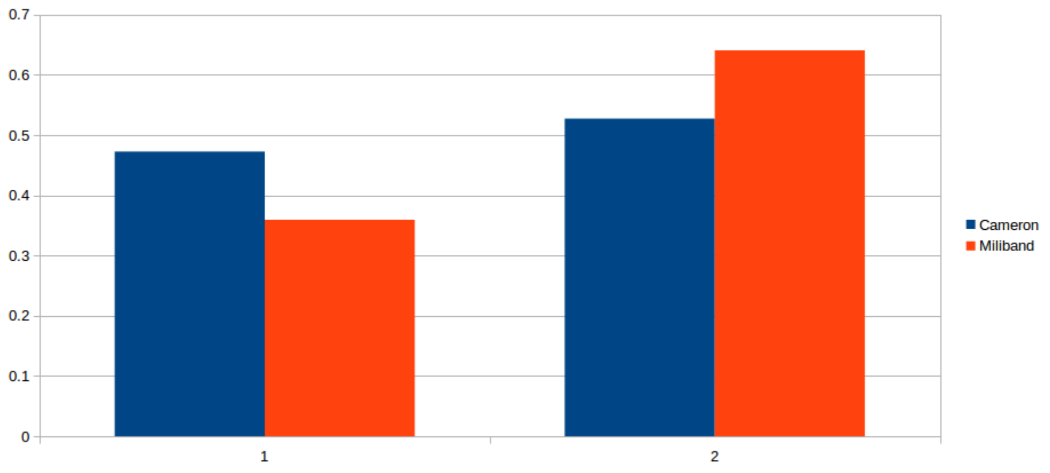


Figure 5.3: Miliband vs Cameron Sentiments

Chapter 6

Conclusion

In this project, we implemented and experimented several IR and ML techniques to evaluate Twitter sentiments for predicting election outcomes.

We attempted to build robust classifiers using feature engineering such as unigram, bigram, unigram + bigram, most-frequent-term extraction, and Hu&Liu Sentiment Corpus [2]. Throughout our experiments with each of these features, Naive Bayes is sensitive to the class imbalance, and tends to give prediction more leaning toward the positive class. However, Naive Bayes works best with the mixture between unigram and bigram, in conjunction with most-frequent-term extraction.

According to our analysis, since Naive Bayes often mis-classifies the negative class, we explored building a model using SVM to take advantage of its soft margin and kernel trick properties, which might improve the rate of classification of the negative class. The result is substancially better than most of the Naive Bayes experiments, especially in classifying the negative class. However, due to our limited hardware and time constraint, we could not experiment SVM with several different parameters (kernels,γ,Cost) and features.

Apart from classifying the tweets into just "positive" or "negative", we attempted to extract the hidden trends and opinions people might have about the two candidates. This was done by clustering and topic modeling. However, due to a high vocabulary of the dataset which consists of many distinct and sparse terms, not much definitive result were obtained. This might be because the topic models tends to capture the co-occurrences of words inside a documents, thus it is sensitive to data sparsity [12].

Chapter 7

Discussion

7.1 Classification

In section 5.1, we discussed about the number of classes and the effect of the neutral class. In this section, we show another experiment with the biased scoring function. Our strategy is described in the code snippet below. That is, we put a strict threshold for the neutral class (a tweet should be 100% neutral to be considered as neutral).

```
for (i in 1:len) {
    if (pred.df$neutral[i] == 1) {
      sent[i] <- "neutral"
    } else if (pred.df$positive[i] > 0.6) {
      sent[i] <- "positive"
    } else {
      sent[i] <- "negative"
    }
  }
```

Beside this scoring function, we also perform 2 additional steps. First, we modify the stop-word list, as we see several words (in the English stop-word dictionary) which can contain sentiment values. They also appear for 6978 times in the SemEval 2014 dataset. Those words, which are shown below, are removed from the stop-word list. Second, we combine the 2 methods of filtering features, Hu&Liu Corpus [2] and Most-Frequent-Term, so that the final features are both the most frequently appeared words and the words that are in the sentiment Corpus. While Hu&Liu word list helps keeping the terms which are clearly negative or positive, therefore increases the chance that the classifier can easily detect the right sentiment, the Most-Frequent terms help maintaining the representative structure of the dataset.

```
stopWords = ['against', "off", "out", "no" ,"nor" ,"not",
"isn't", "aren't", "wasn't", "weren't", "hasn't", "haven't",
"hadn't", "doesn't", "don't", "didn't", "won't", "wouldn't",
"shan't", "shouldn't", "can't", "cannot", "couldn't", "mustn't"]
```

With 3-class setting, even if we try to set a very high cost for the neutral class (as shown above), a large number of tweets (142 negative tweets and 195 positive tweets) are still labeled as neutral, as shown in the confusion matrix 7.1. That is why we decide to remove the neutral class completely from the dataset.

Table 7.1: A Confusion Matrix for Naive Bayes with 3 classes, unigram (mixture of filtering techniques), stemming, stopword removal, and a biased scoring penalty

Pred	negative	neutral	positive
negative	199	18	203
neutral	142	35	195
positive	115	16	435

After removing the neutral class, in order to overcome the imbalance in the training set. We also apply a scoring penalty to modify the posterior probabilities. There are about 6000 positive tweets and only 3000 negative tweets in the training set. Similar to the example above, we also apply a strict penalty scoring to the positive class. A tweet should be 100% positive to be considered as positive, otherwise it is negative. The cross-validation result is shown in table 7.2. Indeed, this result is quite impressive, which is comparable to the mixture of unigram and bigram shown in table 5.9. Its ROC Curve is shown in figure 7.1.

Table 7.2: Naive Bayes statistics with mixture Hu&Liu and MFT unigram, if.idf, Most-Frequent-Terms, stop-word removal, Stemming, and penalty scoring. And an example confusion matrix

Mean/ Standard Deviation	Accuracy	Precision	Recall	F1-Score
Positive	70.25/2.51	80.74/4.21	53.51/2.92	76.77/2.76
Negative		73.22/1.89	64.24/3.64	58.28/1.97

Pred	Negative	Positive
Negative	197	185
Positive	89	569

We also tried another approach for solving the imbalanced dataset. As the number of negative tweets is only a half of the number of positive tweets, we make a copy of the whole negative part of the training set before training the model. In this way, $P("negative")$ and $P("positive")$ are roughly 0.5. The cross-validation result is almost the same to the previous approach. A referenced confusion matrix can be found in table 7.3.

Table 7.3: A Confusion Matrix for Naive Bayes with 2 classes, unigram (mixture of filtering techniques), stemming, stopword removal

Pred	Negative	Positive
Negative	173	182
Positive	110	575

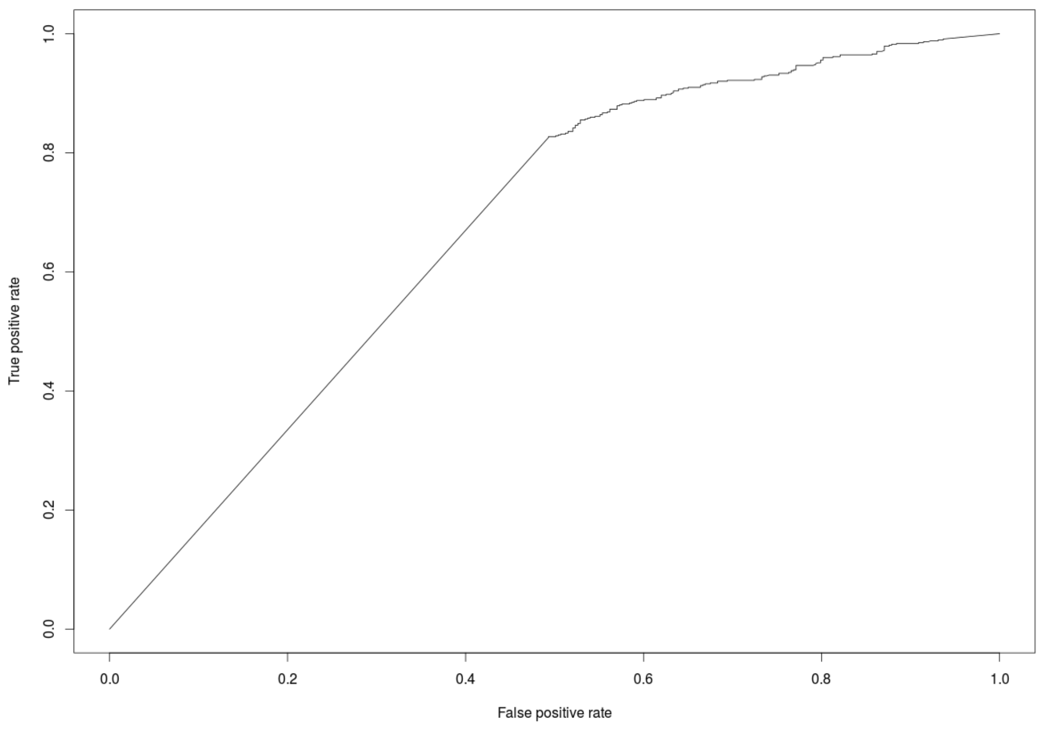


Figure 7.1: ROC Curve of The model built in this Chapter

Those results show that it is very potential to continue developing the classifier in this direction. Because of time limitation, we did not implement this approach (biased scoring penalties, mixture of filtering techniques) to the combination of unigram and bigram, which has been described to be very potential in the last chapter. In practice, it might also be very effective to combine even more features and language model, such as unigram + bigram + trigram, which can help us overcome the limitation of the assumption of term independence that Naive Bayes uses.

7.2 Topic Modeling

As per our discussion, We were trying to manually check the number of co-occurrences of pair of terms in a topic from the topic modelling result. For example: Figure 7.2 is the result of topic modelling on the Cameron dataset. In this result, there are five topics with five terms each.

Due to sparsity in the dataset, we filter our dataset to remove sparse terms using

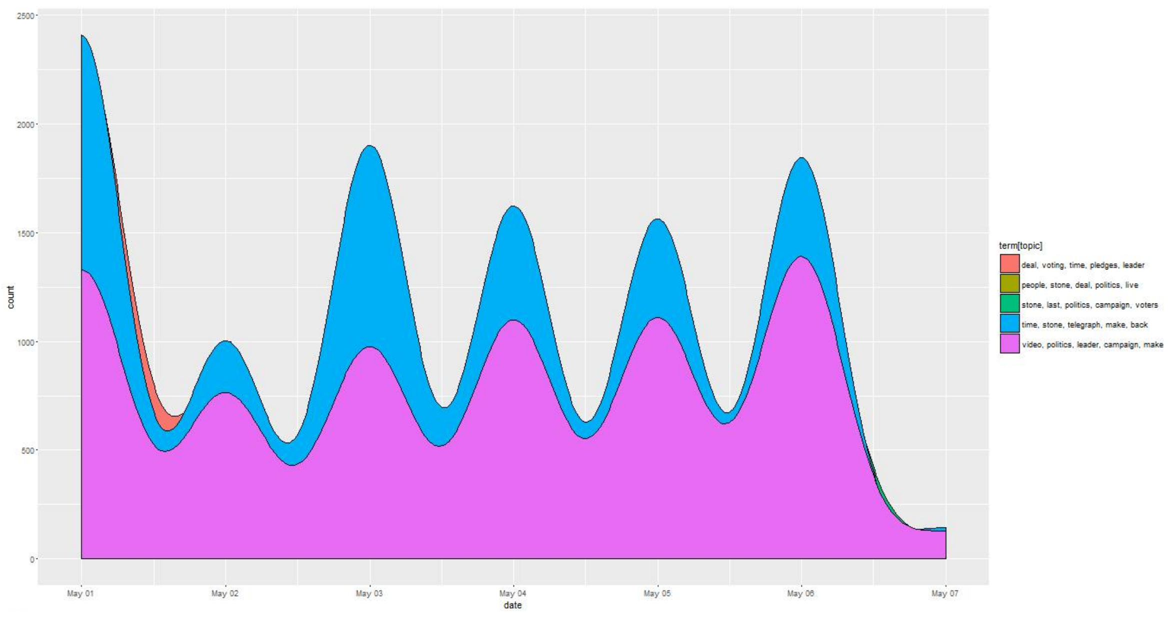


Figure 7.2: Topic Modelling on Cameron Data Set with Unigram feature and tf weighting scheme

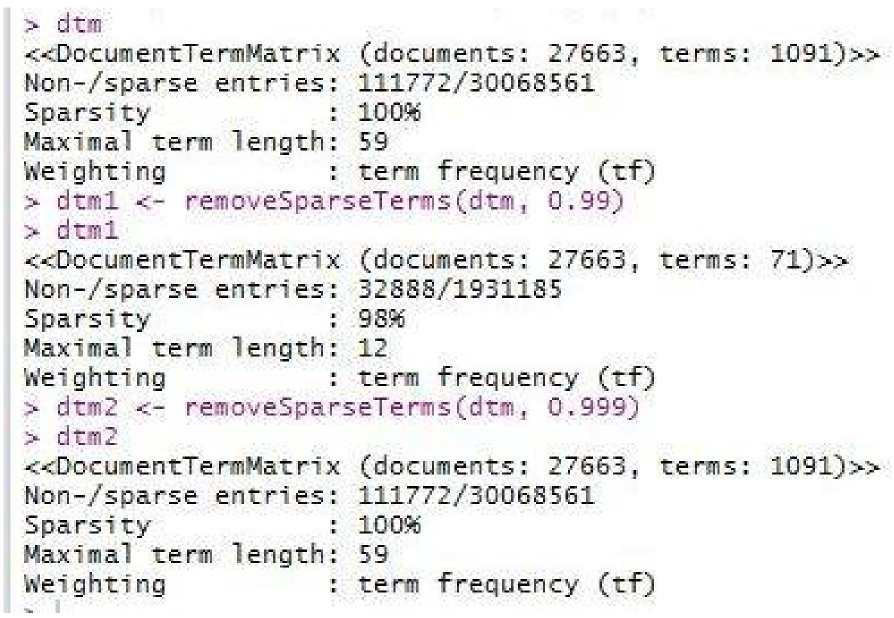
following R script:

```
dtm <- DocumentTermMatrix(myCorpus)
dtm <- removeSparseTerms(dtm, 0.99)
```

Above script removes the sparse terms that has document frequency $df_{term} \leq N \times (1 - 0.99)$, *i.e.* the terms appearing in less than 1% of the documents are removed from the *dtm* (Document Term Matrix). After removing sparse terms with argument 0.99, there are only 71 terms left in the *dtm* (Figure 7.3).

While, when we made a term frequency list (on whole tweets as a document) using a python script, there are more than 500 terms that have frequency greater than 100. This gives an intuition that the terms in above topics may not have large number of co-occurrences. The reason could be that many co-occurring terms may have been removed as a sparse term. The graph between pair of terms from a topic and number of co-occurring tweets is shown in 7.4:

This proves the intuition because many pairs of terms in the topic are not co-occurring at all. We tried the same for other topics as well, and the result is the



```
> dtm
<<DocumentTermMatrix (documents: 27663, terms: 1091)>>
Non-/sparse entries: 111772/30068561
Sparsity           : 100%
Maximal term length: 59
Weighting          : term frequency (tf)
> dtm1 <- removeSparseTerms(dtm, 0.99)
> dtm1
<<DocumentTermMatrix (documents: 27663, terms: 71)>>
Non-/sparse entries: 32888/1931185
Sparsity           : 98%
Maximal term length: 12
Weighting          : term frequency (tf)
> dtm2 <- removeSparseTerms(dtm, 0.999)
> dtm2
<<DocumentTermMatrix (documents: 27663, terms: 1091)>>
Non-/sparse entries: 111772/30068561
Sparsity           : 100%
Maximal term length: 59
Weighting          : term frequency (tf)
```

Figure 7.3: Number of terms left in *dtm* after removing sparse terms with different values of *sparse* in the method

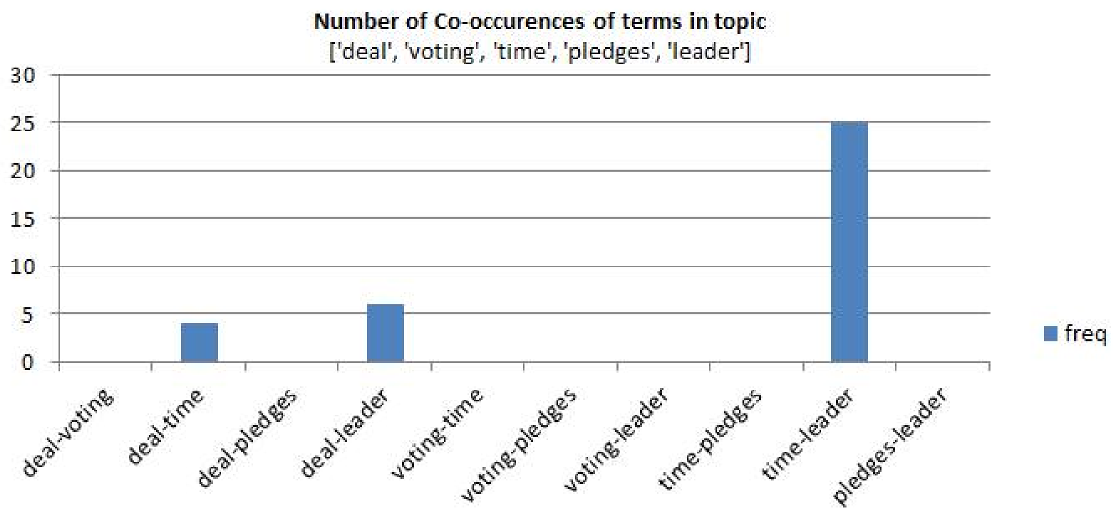


Figure 7.4: Pair-wise co-occurrences of terms in a topic at sparse=0.99 in topic modelling

same.

Thus, another option is to increase the value *sparse* to get at least 500 terms in the *dtm*. We run the topic modelling again with *sparse* = 0.998, and it gives around 600 terms in the *dtm*. Again, we create graphs for each topic (in total five topics in the topic modelling result) : number of tweets vs co-occurring terms from the topic, and the result is better than before (Figure 7.5 7.6 7.7). We get similar graphs for other two topics also. Graphs 7.5 7.6 7.7 show that the pair of terms in a topic

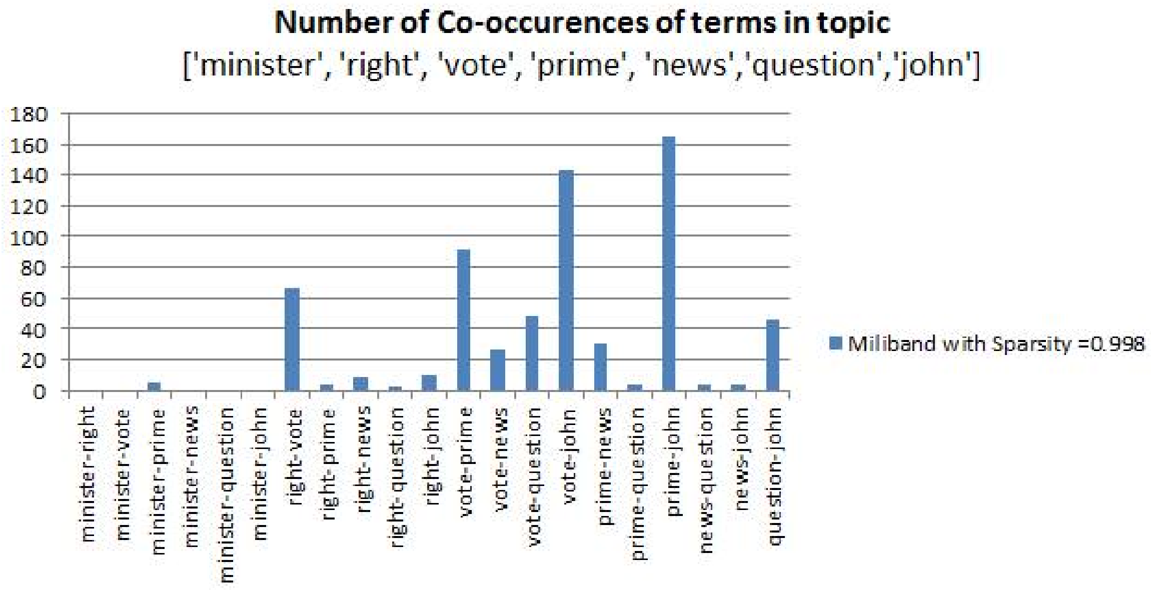


Figure 7.5: Pair-wise co-occurrences of terms in a topic at sparse=0.998 in topic modelling

actually co-occur very frequently. Still, the result is not good in terms of deducing a meaningful topic from the mixture of terms. But this gives an idea about the results produced by LDA algorithm, and we will like to further investigate into the core working of LDA to verify the argument given in this report.

7.3 Prediction

As we discussed in the presentation, grouping by users is a potential way of analyzing sentiments. Because the time is limited, we did not implement it in training the classifiers, as well as in building the topic models. In fact, our training data does not have the user data either. However, we try to group the tweets of a user into a bigger document. What we get is a new dataset of the 2 candidates, which contains 16111 documents for Miliband and 16820 documents for Cameron. Comparing to the original data which contains more than 27000 tweets for Cameron and 57000 tweets for Miliband, the new dataset seems to be more balanced and better to predict the election outcome.

As our new model described in this chapter seems to be reliable, we use it

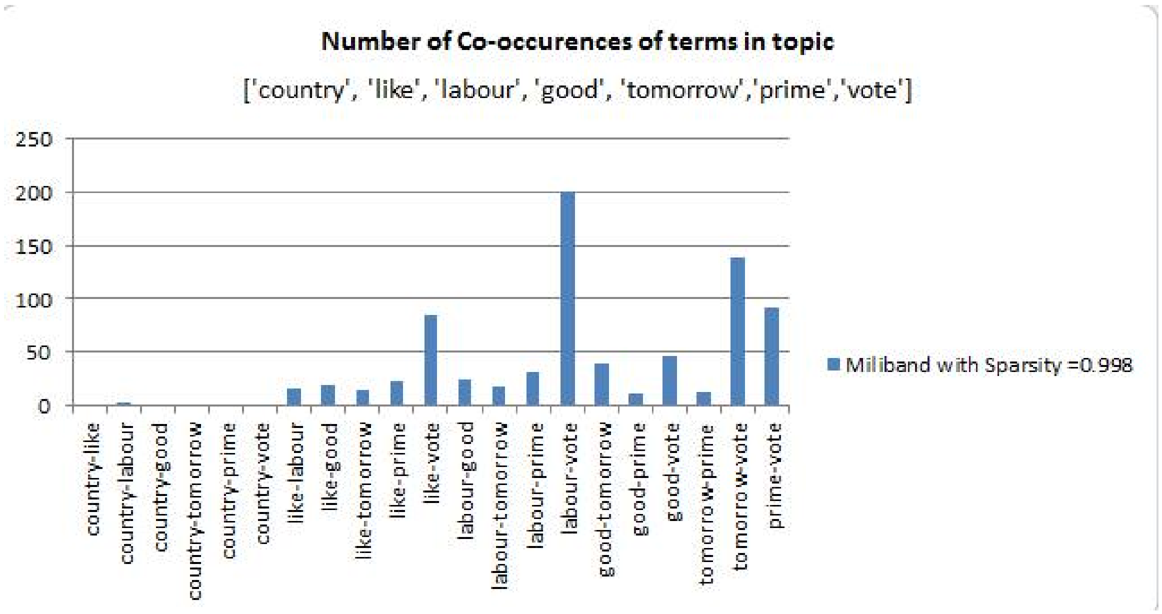


Figure 7.6: Pair-wise co-occurrences of terms in a topic at sparse=0.998 in topic modelling

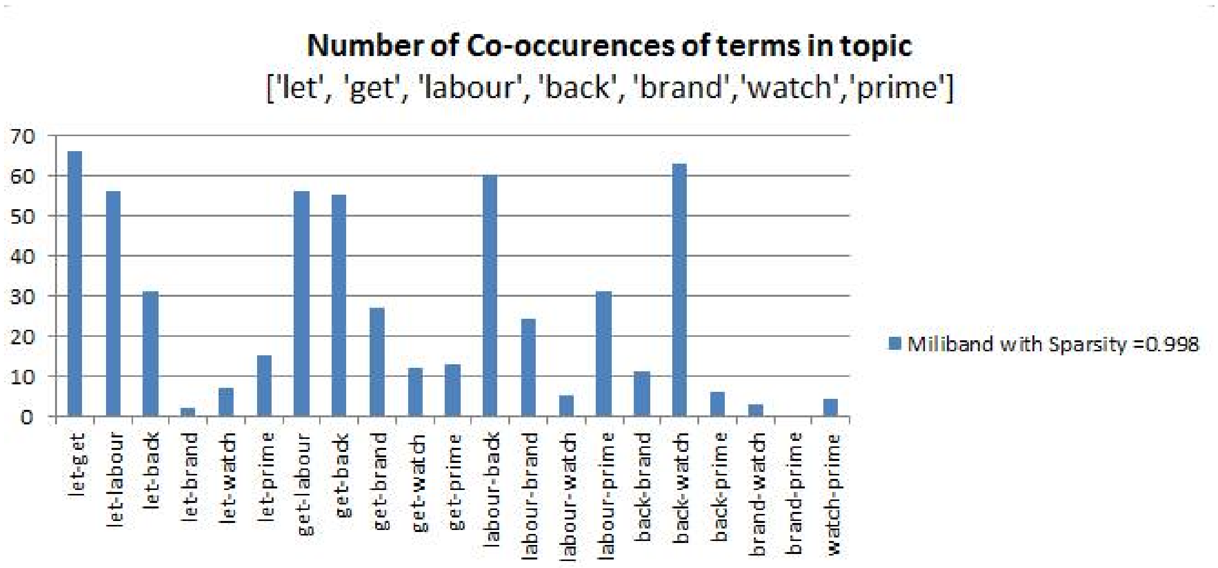


Figure 7.7: Pair-wise co-occurrences of terms in a topic at sparse=0.998 in topic modelling

to predict the election outcome using the new dataset. The sentiment labels now is the user opinions instead of the tweet sentiments, which makes more sense in predicting the election.

Because we trained the model using a dataset consisting of 2 classes only, we have to build a new scoring function to label the candidate dataset correctly. The idea is to assign the "neutral" class to the tweets which have small scores in both "positive" and "negative", and the "positive" class has stricter constraint (biased penalty) than the "negative" class. The detailed scoring function is described in the code snippet below.

```
if (pred.df$positive[i] == 1) {
    sent[i] <- "positive"
  } else if (pred.df$negative[i] > 0.5){
    sent[i] <- "negative"
  } else {
    sent[i] <- "neutral"
  }
```

Comparing to figure 5.3, where we sum the scores of the individual tweets using the mixture bigram and unigram model, this result is surprisingly suitable to the actual election result, where Cameron had more votes. In figure 7.8, we can see that Cameron has both more positive users, and less negative users.

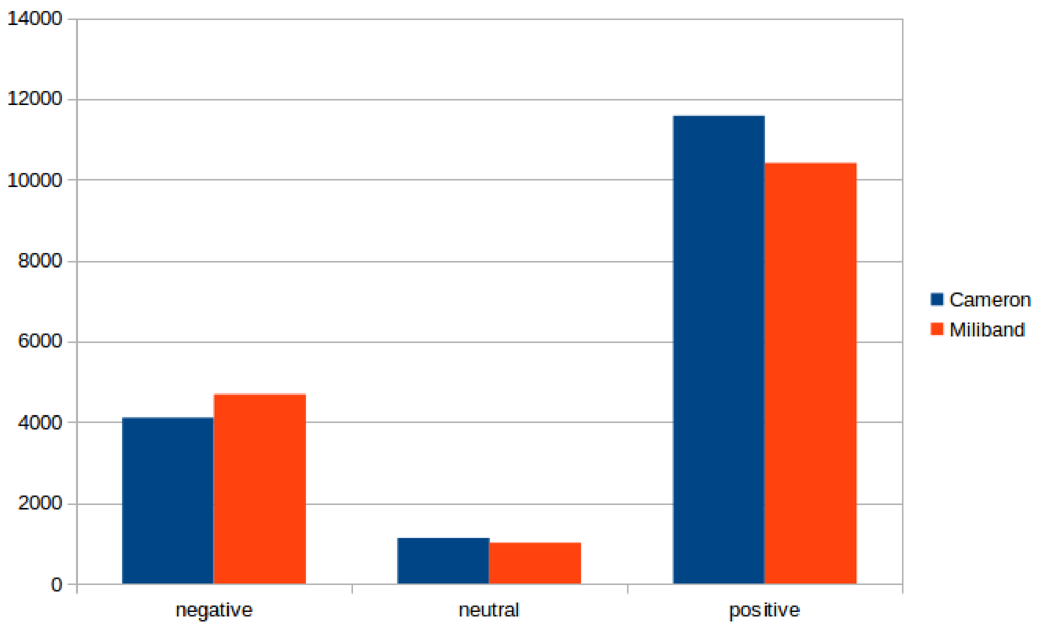


Figure 7.8: Miliband vs Cameron Sentiments, Grouped by Users

Bibliography

[1] *Finally! Exposed! The Deficit Myth! So, David Cameron When Are You Going to Apologise?, The Huffington Post [Online].* http://www.huffingtonpost.co.uk/ramesh-patel/growth-cameron-austerity_b_2007552.html.

[2] Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. Seattle, WA, USA: ACM, 2004, pp. 168–177. ISBN: 1-58113-888-1. DOI: 10.1145/1014052.1014073. URL: http://doi.acm.org/10.1145/1014052.1014073.

[3] *Intro to Topic Modelling*. URL: http://chdoig.github.io/pygotham-topic-modeling/#/.

[4] David Meyer et al. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.6-7. 2015. URL: http://CRAN.R-project.org/package=e1071.

[5] Fionn Murtagh and Pierre Legendre. "Ward's Hierarchical Clustering Method: Clustering Criterion and Agglomerative Algorithm". In: *CoRR* abs/1111.6285 (2011). URL: http://dblp.uni-trier.de/db/journals/corr/corr1111.html#abs-1111-6285.

[6] M.F. Porter. "An algorithm for suffix stripping". In: *Program* 14.3 (1980), pp. 130–137. DOI: 10.1108/eb046814. eprint: http://www.emeraldinsight.com/doi/pdf/10.1108/eb046814. URL: http://www.emeraldinsight.com/doi/abs/10.1108/eb046814.

[7] *Python Programming Language*. URL: https://www.python.org/.

[8] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2015. URL: https://www.R-project.org/.

[9] Sara Rosenthal et al. "SemEval-2014 Task 9: Sentiment Analysis in Twitter". In: *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Dublin, Ireland: Association for Computational Linguistics and Dublin

City University, 2014, pp. 73–80. URL: http://www.aclweb.org/anthology/S14-2009.

[10] *Russell Brand changes mind about voting and urges support for Labour, The Guardian [Online]*. http://www.theguardian.com/politics/2015/may/04/russell-brand-changes-mind-about-voting-and-urges-support-for-labour.

[11] Jarek Tuszynski. *caTools: Tools: moving window statistics, GIF, Base64, ROC AUC, etc.* R package version 1.17.1. 2014. URL: http://CRAN.R-project.org/package=caTools.

[12] Xiaohui Yan et al. “A Biterm Topic Model for Short Texts”. In: *Proceedings of the 22Nd International Conference on World Wide Web*. WWW ’13. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, 2013, pp. 1445–1456. ISBN: 978-1-4503-2035-1. URL: http://dl.acm.org/citation.cfm?id=2488388.2488514.