CS909 Week 10: Text classification, clustering and topic models

Samuel McDermott u1466355

1 Introduction

This report demonstrates the use of text classification, clustering and topic models for the Reuters-21578 dataset [1]. This dataset consists of 21578 documents, extracted from the Reuters newswire in 1987, each with multiple or no labels. The aim of this work is to test a variety of features (Topic Models, n-grams), as well as text classifiers (naive Bayes, Support Vector Machine, Maximum Entropy, Neural Networks, and Random Forests) and clustering (K-means, Hierarchical Agglomerative, Expectation Maximisation).

The work in this project was done using R and several packages (cited as used). The code associated with this report can be found at `http://git.io/vvNci`.

2 Preprocessing and Data Cleaning

2.1 Text preprocessing

Associated R code: `TextPreprocessing.R`

The Reuters-21578[1] dataset is a `.csv` file which consists of binary classified manually added labels for document, the title of the article and the text in the article.

Some articles have several labels, and some have none. The first step is to take this information apart, so that in the final dataset, each document has only one label. This means that the same document may appear several times in the corpus, once for each label. This was done to ensure that each label accurately contained each document associated with it.

The next stage is to select the 10 most popular labels in the dataset. These were provided to us and are: *earn, acquisitions, money-fx, grain, crude, trade, interest, ship, wheat, corn.* This reduces the dataset size and provides a more concentrated selection of documents to label. The documents are then randomly ordered, so that k-fold evaluation can be carried out later.

At this stage, we now have 9612 documents, with the 10 most popular labels, their titles, and their manually added labels. This array can be passed into the other functions, `lda, featureClassification, clustering`

2.2 Document Term Matrix

Associated R code: `convertToDtm.R`

A document term matrix (DTM) is a matrix that describes the frequency of terms that occur in a collection of documents. The rows correspond to the documents in the collection, and the columns correspond to the terms.

In the first stage of each text analytical function, a DTM is calculated for the inputted array. This makes the code more reusable, as a user only has to enter the label, title and text for the documents into each of the functions. This is achieved using the `tm` package [2].

2.2.1 Corpus

Firstly, the documents to be converted into a DTM are changed into a Corpus. This is just a character vector for each document, with a few attributes used by the package. Some more preprocessing is achieved with the function `tm_map`:

- `tolower`: This turns all the data into lowercase, so that uppercase letters (for example at the beginning of sentences) are ignored.
- `removeWords, stopwords("english")`: Stopwords are words that are filtered out to improve the performance of natural language processing. These tend to be the most common words in a language, which do not provide much information gain. These stopwords are provided from the Snowball stemmer project[3] in English.
- `removePunctutation`: This removes non alpha-numeric characters from the documents as they are are unneeded and provide little information gain.
- `stemDocument`: To stem a word is to reduce it to its word stem, base or root form. For example: *argue*, *argued*, *argues*, *arguing* all reduce to the stem *argu*. This is done for all the words in the document using Porter's stemming algorithm[4] and Snowball[3]. Stemming is done as it is useful to make connections between derivationally related words as well as reduce sparseness.
- `removeNumbers`: This removes numeric characters from the documents as they are unneeded and provide little information gain.
- `stripWhitespace`: There may be extra whitespace in the documents, either from the input, or from the preceding transformations. Multiple whitespace characters are therefore collapsed into a single blank.
- `PlainTextDocument`: Mainly for correcting formatting errors.

The documents are now in an informationally useful format and can be turned into a DTM. The next stage is to create the word features for each document.

2.2.2 *n*-grams

I am using *n*-grams (also known as *Shingles*) for my document features. A *n*-gram is a contiguous sequence of *n* items (in this case words) from the document. This is an extension of a *bag-of-words* technique, which can be represented as a uni-gram. Intuitively, these *n*-length phrases can reduce the uncertainty of the meaning of single words and provide more context. In this report, I will test uni, bi and tri-grams. This is achieved using the `RWeka` package[5].

The resulting corpus can now be convert into a DTM, ready for topic models, classification and clustering. Only words that are longer than three characters are included to reduce redundant information. In addition, I found it useful to reduce the sparsity of the DTM to 0.98, to reduce the size of the resulting DTM for processing whilst not losing too much information. This resulting DTM had 1047 terms and 9612 documents for unigram features. `tf-idf` or *term frequency-inverse document frequency* was used as the weighting. This value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the whole corpus. This helps to adjust for words appearing more frequently in general.

3 Topic models

Associated R code: `lda.R`

Topic models are algorithms that find hidden thematic structure in documnet collections. Given that a document is about a particular topic, we should expect to see certain words appearing in it more frequently. This algorithm finds these 'most relevant' words for each unsupervised topic.

Latent Dirichlet Allocation (LDA)[6] is a type of topic model where each topic is assumed to be characterised by a particular set of terms, similar to the standard *bag-of-words* model. Having created a DTM as described previously, I applied an LDA model using the `topicmodels` package[7].

Using *uni-grams*, and finding 10 topics, the following table shows the top 10 words associated with each topic:

Topic 1	**Topic 2**	**Topic 3**	**Topic 4**	**Topic 5**	**Topic 6**	**Topic 7**	**Topic 8**	**Topic 9**	**Topic 10**
said	said	said	trade	said	dlrs	mln	cts	cts	billion
oil	bank	company	tonnes	will	share	reuter	net	april	pct
prices	rate	inc	wheat	offer	quarter	total	loss	record	year
will	rates	shares	said	agreement	year	interest	mln	reuter	last
government	dollar	reuter	exports	reuter	per	end	shr	pay	said
production	market	corp	grain	new	earnings	tax	profit	prior	rose
reuter	pct	pct	imports	board	sales	assets	revs	dividend	february
also	banks	stock	japan	per	first	year	reuter	march	january
industry	exchange	group	reuter	may	company	compared	note	may	expected
last	interest	share	export	spokesman	operations	four	avg	div	compared

It is possible to see how these terms might be connected to each other and the original categories:

- Topic 1 appears to be connected to *crude*.
- Topic 2 appears to be connected to *money-fx*.
- Topic 3 does not immediately appear to have a connection.
- Topic 4 appears to be connected to *grain* or *wheat*.
- Topic 5 may be connected to *acquisitions*.
- Topic 6 does not immediately appear to have a connection.
- Topic 7 does not immediately appear to have a connection.
- Topic 8 does not immediately appear to have a connection.
- Topic 9 does not immediately appear to have a connection.
- Topic 10 may be connected to *earn*.

However, it is not always immediately obvious, especially as some of the topics are very similar.

We can repeat this for *bi-grams*:

Topic 1	**Topic 2**	**Topic 3**	**Topic 4**	**Topic 5**
mln stg	company said	mln tonnes	united states	cts vs
vs mln	inc said	department said	interest rates	cts prior
vs billion	per share	us agriculture	central bank	qtly div
billion vs	corp said	soviet union	analysts said	vs cts
mln vs	crude oil	said reuter	billion dlrs	div cts
money market	dlrs per	agriculture department	officials said	record april
bank england	common stock	securities exchange	dealers said	prior pay
billion stg	also said	exchange commission	last year	record march
profit mln	dlrs share	last month	sources said	april reuter
england said	oil prices	sources said	foreign exchange	march reuter

Topic 6	**Topic 7**	**Topic 8**	**Topic 9**	**Topic 10**
mln vs	mln dlrs	oper net	mln vs	vs loss
vs profit	billion dlrs	oper shr	vs mln	cts vs
vs mln	first quarter	cts share	cts vs	net profit
net loss	last year	dlrs cts	net vs	cts net
vs loss	dlrs mln	cts per	vs cts	revs vs
shr loss	dlrs reuter	per share	shr cts	loss revs
revs mln	mln dlr	cts oper	cts net	shr profit
loss dlrs	year ago	gain dlrs	revs mln	profit vs
loss cts	year earlier	vs dlrs	vs revs	loss cts
loss mln	dlrs billion	net excludes	avg shrs	avg shrs

This does not appear to make the identities of the topics any clearer. **Topic 2** appears to be related to oil; **Topic 3** appears to be related to grain and agriculture and **Topic 4** appears to be related to money-fx. The other topics are very vague.

Working this time with *tri-grams*:

Topic 1	**Topic 2**	**Topic 3**	**Topic 4**	**Topic 5**
vs mln avg	cts per share	mln vs mln	mln vs mln	cts vs cts
mln avg shrs	mln dlrs mln	vs mln note	net mln vs	net vs sales
avg shrs vs	st qtr net	revs mln vs	dlrs vs dlrs	sales mln vs
mln vs mln	mln dlrs reuter	vs mln year	shr dlrs vs	div cts vs
shrs mln vs	dlrs mln dlrs	mln year shr	billion vs billion	vs cts prior
avg shrs mln	billion dlrs billion	mln note net	vs dlrs net	vs sales mln
vs avg shrs	sales mln dlrs	mln revs mln	vs mln revs	cts prior pay
shrs vs note	mln dlrs year	barrels per day	vs mln nine	qtly div cts
shrs vs reuter	net profit mln	note net includes	mln nine mths	pay april record
revs vs avg	mln dlrs cash	vs mln reuter	cts net mln	record march reuter

Topic 6	**Topic 7**	**Topic 8**	**Topic 9**	**Topic 10**
cts vs profit	net profit vs	dlrs per share	shr cts vs	net loss vs
vs profit revs	profit vs loss	securities exchange commission	cts net vs	loss vs loss
net loss vs	cts oper net	us agriculture department	cts vs cts	cts vs loss
loss vs profit	cts net profit	dlrs cts share	vs cts net	cts net loss
vs profit cts	cts vs loss	mln dlrs cts	net vs revs	shr loss cts
cts net loss	vs loss revs	bank england said	vs revs mln	loss cts vs
profit cts net	profit cts vs	uk money market	revs mln vs	vs loss revs
profit revs mln	shr profit cts	mln dlrs dlrs	mln vs mln	vs loss cts
shr loss cts	oper net vs	agriculture department said	vs mln reuter	vs loss dlrs
loss cts vs	vs loss cts	filing securities exchange	vs revs vs	loss cts net

Again, this doesn't seem to make the topics any more obvious. We can assume that **Topic 3** is related to oil, and **Topic 8** is related to agriculture, but the rest of the topics are too similar to draw any conclusions.

From this we can see the uni-grams are the best text feature for topic models, although as many of the topics in the documents are similar, the use of topic models is not particularly useful in this case.

4 Text Classification

Associated R code: `featureClassification.R`, `foldAnalytics.R`, `microOverall.R`, `macroOverall.R`

Using uni-grams, bi-grams and tri-grams as my features, I then carried out classification using 5 classifiers: Naive Bayes, Support Vector Machine, Maximum Entropy, Neural Networks, and Random Forests.

I created code that could apply as many different classification algorithms for comparison, allowed n-grams for any n, used k-fold classification and produced aggregated analysis.

After creating the DTM as described earlier, using both the title and document texts as documents to create the best dataset, and a user defined n-gram, I then used k-fold selection to maximise the data available for testing. Having already randomised the documents, I slid a testing window across the data and trained the models on the rest of the documents.

4.1 Classifiers

The user is free to select which model to use. A very useful package for this is `RTextTools`[8]. This encaspulates many other packages and classification algorithms, providing a uniform interface. It does not, however, incorporate Naive Bayes, and so I added this myself. I have tested 5 classifiers for comparison:

- **Naive Bayes:** This classifier applies Bayes' theorem with strong independence between the features. In this project I use the package `e1071`[9]. This was used as it is a useful benchmark for comparing other classifiers. Call with `"NB"`.

- **Support Vector Machine:** This classifier constructs a hyperplane for use in classification. As a low memory, but previously successful algorithm in the area of text, it should provide good performance. I used a linear kernel as in general text documents are linearly separable[10]. The package used for this is `e1071`[9]. Call with `"SVM"`.

- **Maximum Entropy:** This is a tool for low memory multinomial logistic regression, implemented by `maxent`[11]. It is probabilistic like Naive Bayes, although does not assume conditional independence between the features. This is particularly true for text classification, where words in a document are obviously not independent. Call with `"MAXENT"`.

- **Neural Networks:** A neural network text classifier is a network of units, where the input units represent terms, the output units represent the labels and the weights on the edges between the units represent dependence relations. This is a higher memory algorithm, and subsequently takes more time than the previous classifiers. However, they have been shown to demonstrate statistically comparable performance to that of other on-line linear classifiers[12]. This implementation uses the package `nnet`[13] and is a feed-forward neural network with a single hidden layer. Call with `"NNET"`.

- **Random Forests:** This algorithm constructs a number of decision trees at training time and outputs the class that is the mode of the classes at testing time. Although this is therefore a high memory algorithm, it shares advantages with *bagging* and corrects for decision trees overfitting tendencies. The package `randomForest`[14] is used. Call with `"RF"`.

4.2 Results and Analysis

The following sections will demonstrate the relative successes of these algorithms. The precision, accuracy and recall are calculated both at the macro and micro level for 10-fold classification. Micro-averaging gives weight to every document classification decision, whereas macro-averaging gives equal weight to each topic. As some of the topics (earn $\approx$ 3900 documents; acquisitions $\approx$ 1830 documents) have considerable more documents than others (wheat $\approx$ 280 documents, corn $\approx$ 240 documents) the macro average is a more realistic indicator of effectiveness.

4.3 Uni-gram results

4.3.1 Naive Bayes

avg-type	precision	accuracy	recall
macro	0.246	0.811	0.128
micro	0.057	0.811	0.895

4.3.2 Support Vector Machine

avg-type	precision	accuracy	recall
macro	0.528	0.952	0.508
micro	0.760	0.952	0.973

4.3.3 Maximum Entropy

avg-type	precision	accuracy	recall
macro	0.453	0.936	0.447
micro	0.680	0.936	0.964

4.3.4 Neural Networks

avg-type	precision	accuracy	recall
macro	0.226	0.928	0.287
micro	0.638	0.928	0.960

4.3.5 Random Forests

avg-type	precision	accuracy	recall
macro	0.522	0.948	0.486
micro	0.739	0.948	0.971

4.4 Uni-gram analysis

It can be seen that the SVM is the most effective algorithm. It has the highest accuracy (0.953), precision (0.528) and recall (0.508) compared to the others. All the algorithms perform better than Naive Bayes, which shows they are all at least as good as the baseline.

4.5 Bi-gram results

4.5.1 Naive Bayes

avg-type	precision	accuracy	recall
macro	0.094	0.813	0.106
micro	0.066	0.813	0.896

4.5.2 Support Vector Machine

avg-type	precision	accuracy	recall
macro	0.484	0.896	0.204
micro	0.481	0.896	0.942

4.5.3 Maximum Entropy

avg-type	precision	accuracy	recall
macro	0.439	0.862	0.182
micro	0.312	0.862	0.924

4.5.4 Neural Networks

avg-type	precision	accuracy	recall
macro	0.161	0.892	0.171
micro	0.461	0.892	0.940

4.5.5 Random Forests

avg-type	precision	accuracy	recall
macro	0.463	0.896	0.200
micro	0.478	0.896	0.942

4.6 Bi-gram analysis

Once again, the SVM algorithm performed best. It has the same accuracy (0.896) as Random Forests, but is more precise (0.484) and has a better recall (0.204). The worst performing algorithm is Naive Bayes, as expected. Bi-grams appear to be less accurate the Uni-grams.

4.7 Tri-gram results

4.7.1 Naive Bayes

avg-type	precision	accuracy	recall
macro	0.072	0.814	0.143
micro	0.070	0.814	0.897

4.7.2 Support Vector Machine

avg-type	precision	accuracy	recall
macro	0.164	0.875	0.163
micro	0.373	0.875	0.930

4.7.3 Maximum Entropy

avg-type	precision	accuracy	recall
macro	0.302	0.889	0.184
micro	0.447	0.889	0.939

4.7.4 Neural Networks

avg-type	precision	accuracy	recall
macro	0.038	0.845	0.097
micro	0.224	0.845	0.914

4.7.5 Random Forests

avg-type	precision	accuracy	recall
macro	0.042	0.845	0.100
micro	0.224	0.845	0.914

4.8 Tri-gram analysis

We can see that the Maximum Entropy algorithm performed best with tri-grams. It also has a considerably higher precision than a SVM. Once again, as expected, the Naive Bayes algorithm performed the worst. Tri-grams are less accurate then bi-grams or uni-grams and so we can conclude the uni-grams are the most effective word feature, and the SVM the best classifier for these text documents.

5 Clustering

Associated R code: `clustering.R`

This section focusses on clustering algorithms. The same dataset is used, with 10 topics, and I will test the clustering algorithms ability to cluster these documents into 10 clusters, and then compare the results. As it has been previously demonstrated to be the most effective feature, both for topic models and classification, I shall use uni-gram features. The data is also normalised. The three clustering algorithms I shall test are Hierarchical Agglomerative, k-means and Expectation Maximisation.

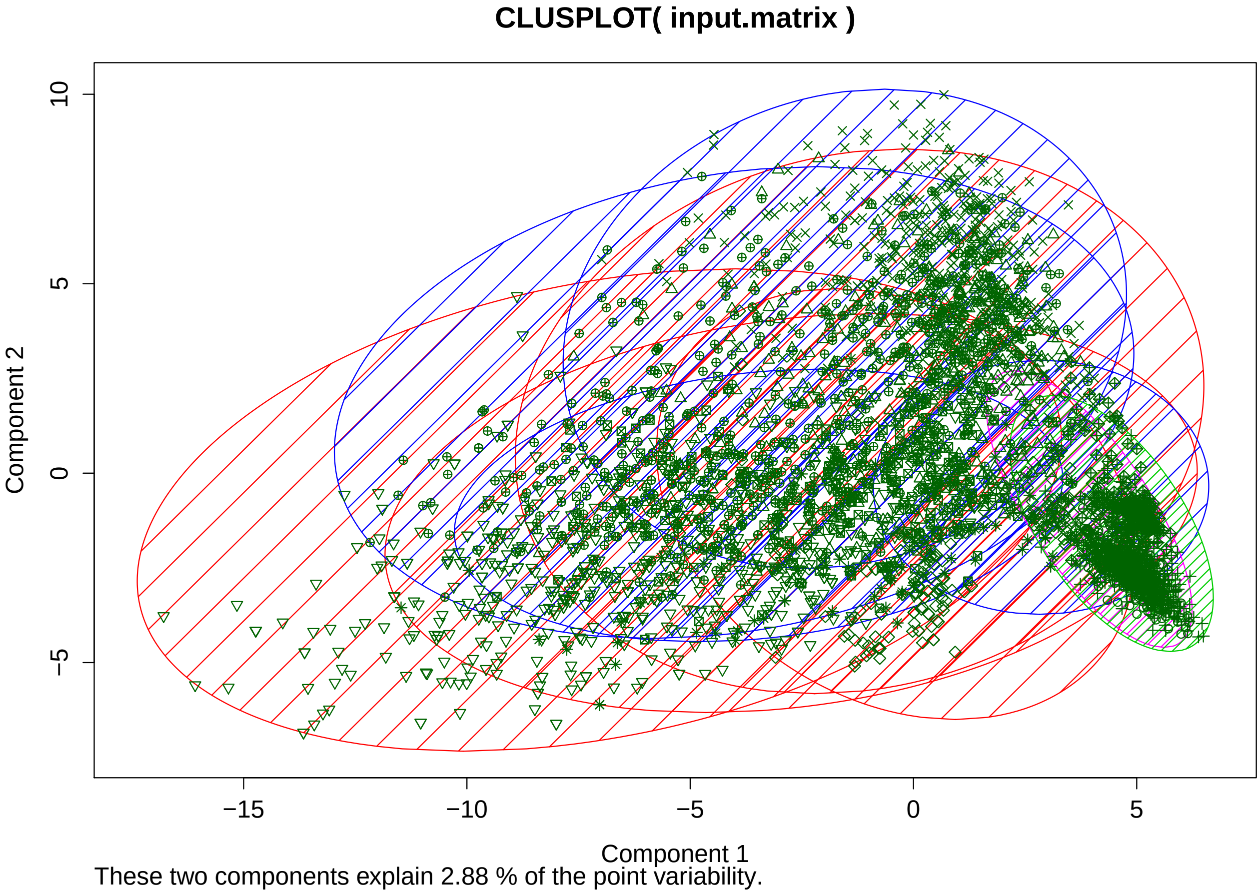
5.1 Clustering Algorithms

The three algorithms have been chosen to represent different clustering techniques.

- **k-means:** This algorithm aims to partition n observations into k clusters, such that each observation belongs to the cluster with the nearest mean. As the data has 10 labels, I shall split the data into 10 clusters. This implementation uses the built in `kmeans` function and plotted using the package `cluster`.
- **Hierarchical Clustering:** This algorithm builds a hierarchy of clusters. Each document is assigned to its own cluster and then the algorithm proceeds iteratively, at each stage joining the two most similar clusters, until there is a single cluster. Then, we place a cut in the resulting dendrogram at the position where there are 10 clusters. This algorithm is very slow ($O(n^2)$). The implementation used is the standard function `hclust`.
- **Expectation Maximisation:** The EM algorithm iteratively tries to find the parameters of the probability distribution that has the maximum likelihood of its attributes. This algorithm is even slower, as it works until convergence is reached. This implementation uses the package `mclust`[15].

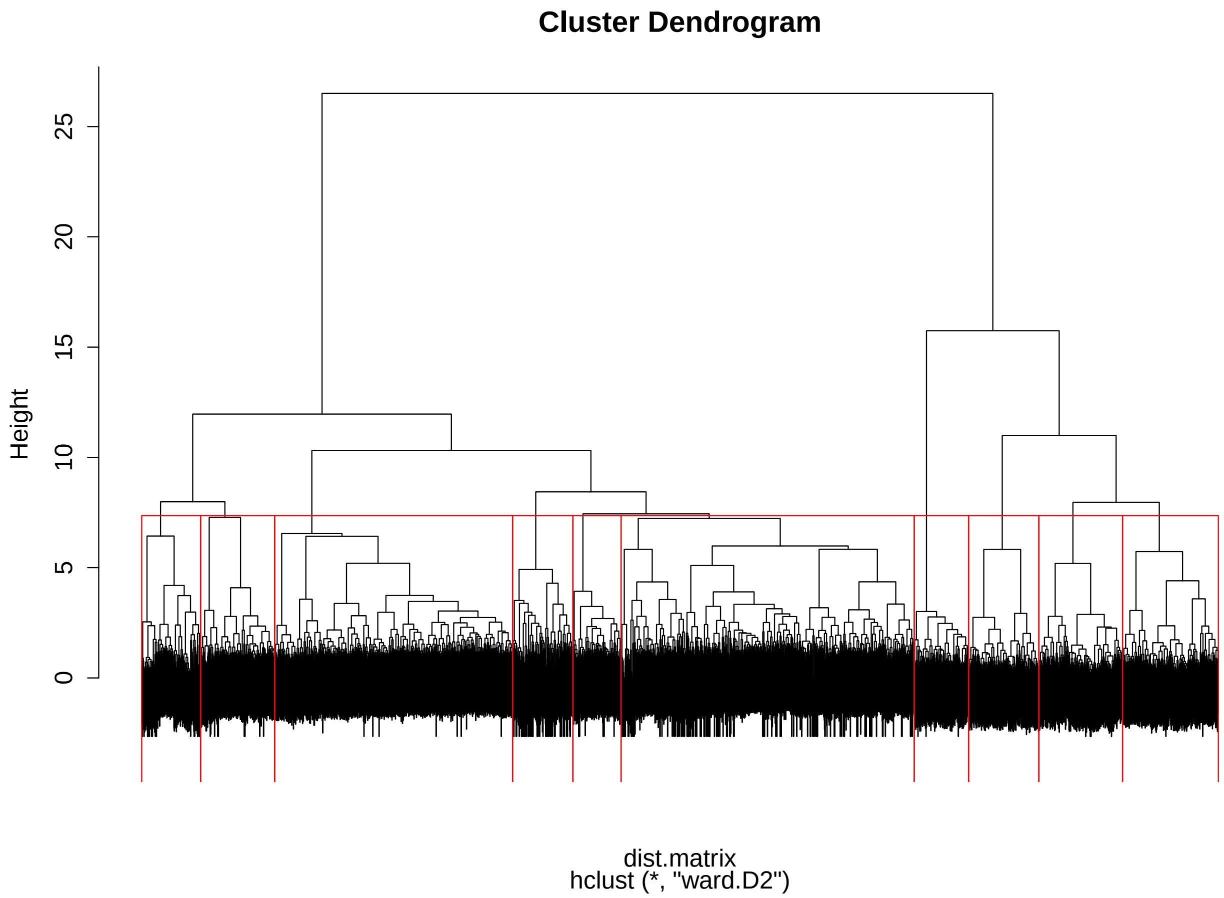
5.2 Results and Analysis

5.2.1 k-means



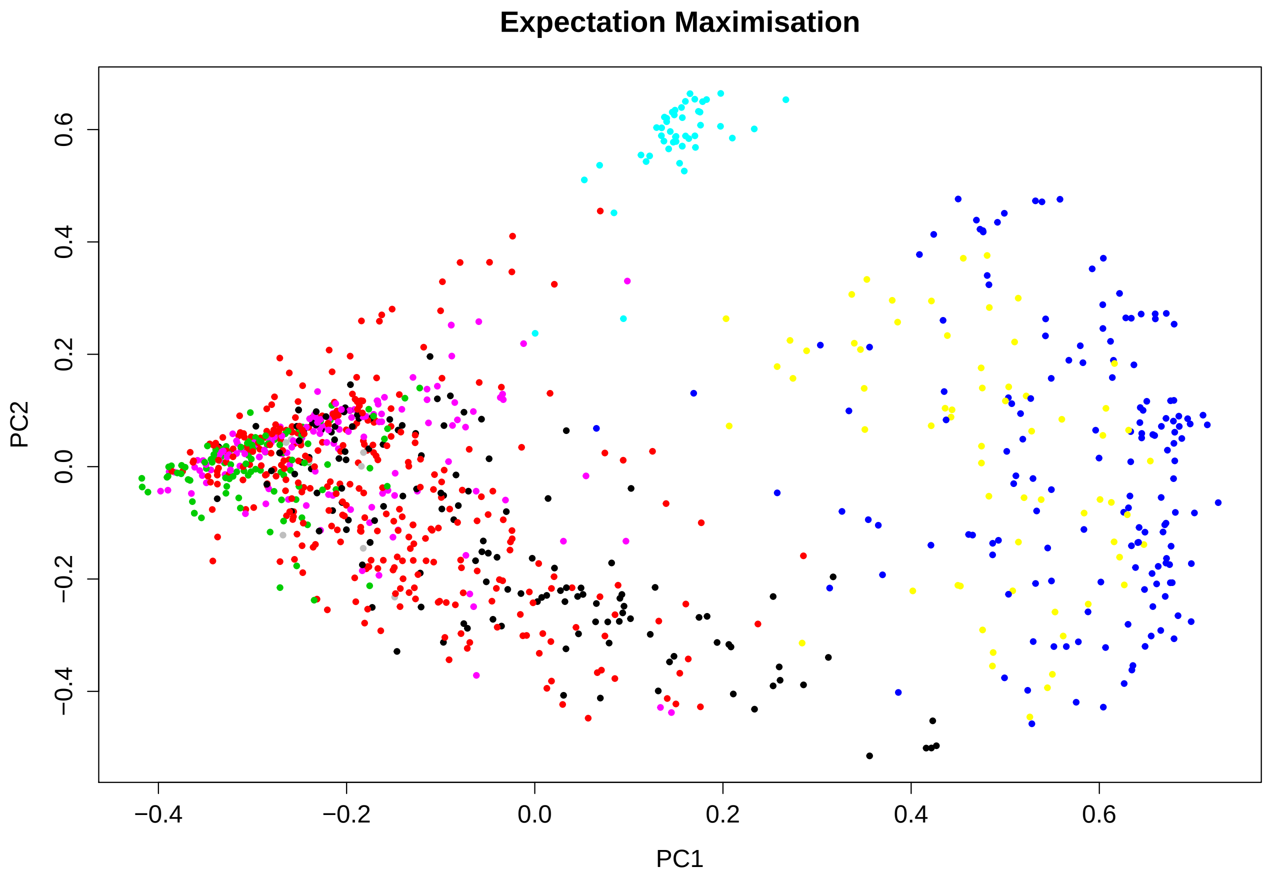
This graph is plotted when using the k-means clustering algorithm with 10 means, showing the distribution of points and the generated clusters. It is displayed using the two principle components, although they only explain 2.88% of the variability. This demonstrates that documents require a lot of features for clustering. It can also be seen that there is considerable overlap between many of clusters. This may represent that the feature representation is not very effective, or reflect the fact that many of the documents have many labels, and are very similar to each other.

5.2.2 Hierarchical Clustering



This dendrogram is created when using Hierarchical Agglomerative clustering. The red boxes indicate the 10 cut off classes. It can been seen that the classes created are of very different size, which accurately recreates thee uneven sizes of the manually labelled classes. With this clustering, there appears to be fairly separate branches.

5.2.3 Expectation Maximisation



This graph shows how the expectation maximisation algorithm performs, when the points are distributed over the two principle components. There appears to be three major clusters: left, right and up. The 10 clusters overlap considerably.

5.2.4 Rand index

For each clustering algorithm, the Rand index is calculated compared to the manual labels. This is implemented by the `fpc` package[16]. It is a measure of similarity between two data clusterings and has a value between 0 and 1. A 0 indicates that the two data clusters do not agree on any pair of points and 1 indicates the data clusters are exactly the same.

Algorithm	Rand Index
k-means	0.350
Hierarchical Agglomerative	0.293
Expectation Maximisation	0.306

The k-means algorithm is the most effective, indicating that the clusters that it creates overlaps the most with the manually labelled clusters. This may be because, as discussed earlier, textual features in general are lineally separable. However, only 35% of points overlap, indicating that this has not worked well. This appears to be because the documents have overlap over several topics, and are not clearly separable from one another.

6 Conclusions

This report has demonstrated various analytical procedures for use with text documents. It has described suitable preprocessing, feature representation, topic models, classification and clustering techniques.

Unigrams (as a bag-of-words) were found to be the best feature representation, and a Support Vector machine with linear kernel the best classifier. Using uni-grams for clustering indicated that the k-means algorithm worked most effectively, although not to a high degree of accuracy

References

[1] D. Lewis. (2013) Reuters-21578, distribution 1.0. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection

[2] I. Feinerer and K. Hornik, *tm: Text Mining Package*, 2014, r package version 0.6. [Online]. Available: http://CRAN.R-project.org/package=tm

[3] (2015) Snowball stemmer project. [Online]. Available: http://svn.tartarus.org/snowball/trunk/website/algorithms/english/stop.txt

[4] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[5] K. Hornik, C. Buchta, and A. Zeileis, "Open-source machine learning: R meets Weka," *Computational Statistics*, vol. 24, no. 2, pp. 225–232, 2009.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[7] B. Grün and K. Hornik, "topicmodels: An R package for fitting topic models," *Journal of Statistical Software*, vol. 40, no. 13, pp. 1–30, 2011. [Online]. Available: http://www.jstatsoft.org/v40/i13/

[8] T. P. Jurka, L. Collingwood, A. E. Boydstun, E. Grossman, and W. van Atteveldt, "Rtexttools: Automatic text classification via supervised learning." [Online]. Available: http://CRAN.R-project.org/package=RTextTools

[9] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, C.-C. Chang, and C.-C. Lin, "Misc functions of the department of statistics (e1071), tu wien." [Online]. Available: http://cran.r-project.org/web/packages/e1071/index.html

[10] T. Joachims, *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[11] T. P. Jurka and Y. Tsuruoka, "maxent: Low-memory multinomial logistic regression with support for text classification." [Online]. Available: http://cran.r-project.org/web/packages/maxent/index.html

[12] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[13] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, 4th ed. New York: Springer, 2002, iSBN 0-387-95457-0. [Online]. Available: http://www.stats.ox.ac.uk/pub/MASS4

[14] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: http://CRAN.R-project.org/doc/Rnews/

[15] C. Fraley, A. E. Raftery, T. B. Murphy, and L. Scrucca, *mclust Version 4 for R: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation*, 2012.

[16] C. Hennig, *fpc: Flexible procedures for clustering*, 2014. [Online]. Available: http://cran.r-project.org/web/packages/fpc/

A Code used:

This is the code used in the project. It can also be found on github at: `http://git.io/vvNci`.

A.1 TextPreprocessing.R

```r
preprocess<-function(input.filename){

  options(stringsAsFactors = FALSE) #a default option that we need to change

  print("Reading in data")
  #The 10 most populus classes, and the ones we'll use for evaluation
  populus = c("topic.earn","topic.acq", "topic.money.fx", "topic.grain", "topic.crude", "topic.trade", "topic.interest", "topic.ship", "topic.wheat", "topic.corn") #make sure we don't use factors for strings as default

  #get in the data
  input.raw <- read.csv(file=input.filename, header=T, sep=",")

  #this will hold everything we're outputting
  output.df <- NULL

  #find the columns that identify the topics
  topicColumns <-grep("topic",attributes(input.raw)$names,ignore.case = TRUE, value = FALSE)

  for(i in 1:nrow(input.raw)){
    #Find the number of topics associated with this document
    numTopics <- sum(input.raw[i,topicColumns])
    #if this document has topics associated and contains text (otherwise this document will get dropped)
    if (numTopics > 0 && input.raw$doc.text[i]!= ""){
      for(j in topicColumns){
        if(input.raw[i,j] == 1){
          #take each row and create a new document for each topic, and use the actual name of the topic
          oldrow<-input.raw[i,]
          newrow <- data.frame(attributes(oldrow[j])$names,oldrow$doc.title,oldrow$doc.text)
          #add this row
          output.df <-rbind(output.df,newrow)
        }
      }
    }
  }

  names(output.df)<- list("topic","title","text")

  #choose the documents that have the 10 most popular topics
  output.df <- subset(output.df, subset = topic %in% populus)
  #shuffle up the instances for bias free k fold
  output.df <- output.df[sample(1:nrow(output.df),size=nrow(output.df),replace=FALSE),]
  #we want the topic to be a factor
  output.df$topic <- as.factor(output.df$topic)
  #return a dataframe with the topic, title and text for the correct documents
  return(output.df)
}
```

A.2 convertToDtm.R

```r
convertToDtm <-function(input,n){
  require(tm)
  require(RWeka)
  #make the input a corpus (dataframesource is so we can use multiple columns)
  corpus <- Corpus(DataframeSource(input))

  corpus <- tm_map(corpus, tolower) #all lowercase
  corpus <- tm_map(corpus,removeWords,stopwords("english")) #remove stopwords
  corpus <- tm_map(corpus, removePunctuation) #remove punctuation
  corpus <- tm_map(corpus, stemDocument) #stem the document
  corpus <- tm_map(corpus, removeNumbers) #remove numbers
  corpus <- tm_map(corpus, stripWhitespace) #remove extra whitespace
  corpus <- tm_map(corpus,PlainTextDocument) #fix formatting cos it breaks somewhere above

  #make the ngram function, where n is inputted by the user
  ngramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = n, max = n))
  #make corpus into document term matrix, with ngrams
  dtm <-DocumentTermMatrix(corpus,control = list(tokenize = ngramTokenizer,weighting=weightTfIdf))
  #remove the sparse terms, best set between 0.95 and 0.99
  dtm <- removeSparseTerms(dtm,0.98)
  #return this document term matrix
  return(dtm)
}
```

A.3 lda.R

```
lda<-function(input,n,k){
  require(topicmodels)

  #put everything in right format (& make the topic a numeric factor)
  input$topic <- factor(input$topic)
  input$title <- as.character(input$title)
  input$text <- as.character(input$text)
  #get the document term matrix
  dtm<- convertToDtm(cbind(input$text,input$title),n)

  rowTotals <- apply(dtm , 1, sum) #find the sum of words in each Document
  dtm <- dtm[rowTotals> 0, ]#remove all docs without words
  #create a topic model with k topics
  lda <-LDA(dtm,k)
  #return this topic model
  return(lda)
}
```

A.4 featureClassification.R

```r
featureClassification<-function(input,n,k,classifier,...){
  require(RTextTools)
  require(e1071)
  require(RWeka)
  #just incase user inputted incorrectly
  classifier <- toupper(classifier)

  #put everything in right format (& make the topic a numeric factor)
  input$topic <- as.numeric(factor(input$topic))
  input$title <- as.character(input$title)
  input$text <- as.character(input$text)
  #create a dtm
  input.matrix <- convertToDtm(cbind(input$title,input$text),n)


  print(paste("Creating ",n,"-gram features"))
  #these will store the analytical information during and after folding
  analytics <-NULL
  allAnalytics <- NULL

  # lets fold this up
  for(fold in 1:k){
    print(paste("Fold ", fold))
    #train using (k-1)n/k instances and test using n/k, see documentation on what this
        is doing
    sizeOfTest <- floor(nrow(input)/k)
    testLower <-  ((fold-1)*sizeOfTest)+1 #position in corpus
    testUpper<- testLower + sizeOfTest #position in corpus
    if(testUpper >= nrow(input)){testUpper <- nrow(input)-1} #correction for when we
        reach the top


    if(classifier=="NB"){
      #Naive Bayes
      print("Using Naive bayes classifier")
      NB.matrix <-as.matrix(input.matrix) #for Naive Bayes we need it as an actual
          matrix
      #train using the training data
      NB.model <- naiveBayes(NB.matrix[c(1:testLower,testUpper:(nrow(input))),],
          as.factor(input[c(1:testLower,testUpper:(nrow(input))),c("topic")]))
      #predict using the testing data
      NB.predicted <- predict(NB.model,NB.matrix[(testLower+1):(testUpper-1),])
      #get the analytics for this fold and append to previous folds
      analytics <-
          append(analytics,foldAnalytics(cbind(NB.predicted,input[(testLower+1):(testUpper-1),c("t
    }else{
      #create a container to use with RTextTools
      input.corpus <-create_container(input.matrix,as.factor(input$topic),trainSize =
          c(1:testLower,testUpper:(nrow(input))), testSize
          =c((testLower+1):(testUpper-1)), virgin = FALSE)
      #which classifier from RTextTools
      print(paste("Using ",classifier," classifier"))
      #train the classifier, carrying any arguments that user inputted for classifier
      model <- train_model(input.corpus,classifier,list(...))
      #predict using the test data
      result <- classify_model(input.corpus, model)
      #get the analytics for this fold and append to previous folds
      analytics<-
          append(analytics,foldAnalytics(cbind(result[,1],input$topic[(testLower+1):(testUpper-1)])

    }
  }

  print("Calculating Micro and Macro averages")
```

```
  #get the micro and macro analytics
  micro <- unname(microOverall(analytics))
  macro <- unname(macroOverall(analytics))
  micro <- cbind("micro",micro)
  macro <- cbind("macro",macro)
  #for rbind-ing
  names(micro) <- c("avg-type","precision","accuracy","recall")
  names(macro) <- c("avg-type","precision","accuracy","recall")

  allAnalytics <- rbind(allAnalytics,macro,micro)
  names(allAnalytics) <- c("avg-type","precision","accuracy","recall")
  print(allAnalytics)
  #write to disk
  write.csv(allAnalytics,paste0(n,"gram_",k,"fold_",classifier,".csv"))

  return(allAnalytics)
}
```

A.5 foldAnalytics.R

```
foldAnalytics <-function(predictedAndActual, classes){
  classesList <-list() #stores the list of classes analytics
  for(c in 1:length(classes)){
    #get the predictions and actual classes for the actual classes that are this class
        (true set)
    trueset <- predictedAndActual[predictedAndActual[,2]==classes[c],]
    #get the predictions and actual classes for the actual classes that aren't this
        class (false set)
    falseset <- predictedAndActual[predictedAndActual[,2]!=classes[c],]
    #get the performance measures for the model, fold and class
    falsePositive <- length(which(falseset[,1]==classes[c]))
    trueNegative <- length(which(falseset[,1]!=classes[c]))
    falseNegative <- length(which(trueset[,1]!=classes[c]))
    truePositive <- length(which(trueset[,1]==classes[c]))
    precision <- truePositive/(truePositive+falsePositive)
    accuracy <- (truePositive+trueNegative)/(length(predictedAndActual[,1]))
    recall<- truePositive/(truePositive+falseNegative)
    #make a list of these measures
    measureList <- list(falsePositive=falsePositive,trueNegative = trueNegative,
        truePositive = truePositive, falseNegative = falseNegative,precision=precision,
        accuracy = accuracy, recall=recall)
    #if any of the results are NaN, they should be 0
    measureList <- rapply( measureList, f=function(x) ifelse(is.nan(x),0,x),
        how="replace" )
    #add this measure list to the class list
    classesList <- append(classesList,list(measureList))
  }
  return(list(classesList))
}
```

A.6 `macroOverall.R`

```
macroOverall <- function(res){
  macroOverallAverage <- data.frame() #output frame
  numClasses <- length(res[[1]]) #number of classes
  numFolds <- length(res) #number of folds

  for(measure in 1:3){
    sum <- 0 #the sum for this measure across all classes and folds
    for(c in 1:numClasses){
      for(i in 1:numFolds){
        sum <- sum+res[[i]][[c]][[measure+4]] #add this measure for this class and fold
      }
    }
    macroOverallAverage[1,measure] <- sum/(numFolds*numClasses) #get the overall macro
        average for this measure
  }
  return(macroOverallAverage)
}
```

A.7 `microOverall.R`

```r
microOverall <- function(res){
  microAverage <- data.frame() #output frame
  numClasses <- length(res[[1]]) #number of classes
  numFolds <- length(res) #number of folds
  sumFP <- 0 #sum of false positives across all classes, across all folds
  sumTN <- 0 #sum of true negatives across all classes, across all folds
  sumTP <- 0 #sum of true positives across all classes, across all folds
  sumFN <- 0 #sum of false negatives across all classes, across all folds
  for(c in 1:numClasses){
    for(i in 1:numFolds){
      sumFP <- sumFP + res[[i]][[c]]$falsePositive #add the measure for this fold and class
      sumTN <- sumTN + res[[i]][[c]]$trueNegative
      sumTP <- sumTP + res[[i]][[c]]$truePositive
      sumFN <- sumFN + res[[i]][[c]]$falseNegative
      }
  }
  #calculate micro average across all classes for this measure:
  microAverage[1,1] <- sumTP/(sumTP+sumFP) #precision
  microAverage[1,2] <- (sumTP +sumTN)/(sumTP + sumFP+sumFN+sumTN) #accuracy
  microAverage[1,3] <- sumTN/(sumFP+sumTN) #recall

return(microAverage)
}
```

A.8 clustering.R

```r
clustering <-function(input){
  require(cluster)
  require(mclust)
  require(fpc)

  #get everything in the right format
  input$topic <- as.numeric(factor(input$topic))
  input$title <- as.character(input$title)
  input$text <- as.character(input$text)
  #create a dtm (use unigrams)
  input.matrix <- convertToDtm(cbind(input$title,input$text),1)
  #turn it into a matrix
  input.matrix <- as.matrix(input.matrix,stringsAsFactors = FALSE)
  rownames(input.matrix) <- 1:nrow(input.matrix)

  #normalise matrix
  norm_eucl <- function(m) m/apply(m, MARGIN=1, FUN=function(x) sum(x^2)^.5)
  input.matrix <- norm_eucl(input.matrix)
  dist.matrix <- dist(input.matrix, method = "euclidean") # distance matrix

  # K-means
  kmean <- kmeans(input.matrix, 10) #do kmeans clustering for 10 clusters
  clusplot(input.matrix, kmean$cluster, color=TRUE, shade=TRUE, labels=FALSE, lines=0)
      #plot the kmeans over principle components

  # Hierarchical Agglomerative
  HA<- hclust(dist.matrix, method="ward.D2") #do hierarchical clustering
  plot(HA, labels = FALSE) # display dendogram
  HAcut <- cutree(HA, k=10) # cut tree into 10 clusters
  # draw dendogram with red borders around the 10 clusters
  rect.hclust(HA, k=10, border="red")

  #Expectation maximisation
  EM <- Mclust(input.matrix,G=10) # do expectation maximisation and plot
  plot(prcomp(input.matrix)$x, col=EM$cl,pch=3,main ="Expectation Maximisation" )

  #calculate analytics
  kmeanRand <- cluster.stats(dist.matrix, kmean$cluster, input$topic, compareonly =
      TRUE)
  print(kmeanRand)
  HARand <- cluster.stats(dist.matrix, HAcut, input$topic, compareonly = TRUE)
  print(HARand)
  EMRand <- cluster.stats(dist.matrix, EM$classification, input$topic, compareonly =
      TRUE)
  print(EMRand)
}
```