Data Mining with R

Text Mining

Hugh Murrell

reference books

These slides are based on a book by Yanchang Zhao:

R and Data Mining: Examples and Case Studies.

http://www.RDataMining.com

for further background try Andrew Moore's slides from:
http://www.autonlab.org/tutorials

and as always, **wikipedia** is a useful source of information.

text mining

This lecture presents examples of text mining with R.

We extract text from the BBC's webpages on Alastair Cook's letters from America. The extracted text is then transformed to build a term-document matrix.

Frequent words and associations are found from the matrix. A word cloud is used to present frequently occuring words in documents.

Words and transcripts are clustered to find groups of words and groups of transcripts.

In this lecture, *transcript* and *document* will be used interchangeably, so are *word* and *term*.

text mining packages

many new packages are introduced in this lecture:

- **tm**: [Feinerer, 2012] provides functions for text mining,
- **wordcloud** [Fellows, 2012] visualizes results.
- **fpc** [Christian Hennig, 2005] flexible procedures for clustering.
- **igraph** [Gabor Csardi , 2012] a library and R package for network analysis.

retrieving text from the BBC website

This work is part of the Rozanne Els PhD project

She has written a script to download transcripts direct from the website
http://www.bbc.co.uk/programmes/b00f6hbp/broadcasts/1947/01

The results are stored in a local directory, `ACdatedfiles`, on this apple mac.

loading the corpus from disc

Now we are in a position to load the transcripts directly from our hard drive and perform corpus cleaning using the `tm` package.

```
> library(tm)
> corpus <- Corpus(
+     DirSource("./ACdatedfiles",
+       encoding = "UTF-8"),
+       readerControl = list(language = "en")
+     )
```

cleaning the corpus

now we use regular expressions to remove at-tags and urls from the remaining documents

```
> # get rid of html tags, write and re-read the cleane
> pattern <- "</?\\w+((\\s+\\w+(\\s*=\\s*(?:\".*?\"|'
> rmHTML <- function(x)
+   gsub(pattern, "", x)
> corpus <- tm_map(corpus,
+                  content_transformer(rmHTML))
> writeCorpus(corpus,path="./ac",
+   filenames = paste("d", seq_along(corpus),
+                  ".txt", sep = ""))
> corpus <- Corpus(
+   DirSource("./ac", encoding = "UTF-8"),
+       readerControl = list(language = "en")
+     )
```

further cleaning

now we use text cleaning transformations:

```
> # make each letter lowercase, remove white space,
> # remove punctuation and remove generic and custom
> corpus <- tm_map(corpus,
+                  content_transformer(tolower))
> corpus <- tm_map(corpus,
+                  content_transformer(stripWhitespace))
> corpus <- tm_map(corpus,
+                  content_transformer(removePunctuation))
> my_stopwords <- c(stopwords('english'),
+   c('dont','didnt','arent','cant','one','two','three
> corpus <- tm_map(corpus,
+                  content_transformer(removeWords),
+                  my_stopwords)
```

stemming words

In many applications, words need to be stemmed to retrieve their radicals, so that various forms derived from a stem would be taken as the same when counting word frequency.

For instance, words update, updated and updating should all be stemmed to updat.

Sometimes stemming is counter productive so I chose not to do it here.

```
> # to carry out stemming
> # corpus <- tm_map(corpus, stemDocument,
> #                  language = "english")
```

building a term-document matrix

A term-document matrix represents the relationship between terms and documents, where each row stands for a term and each column for a document, and an entry is the number of occurrences of the term in the document.

```
> (tdm <- TermDocumentMatrix(corpus))
```

```
<<TermDocumentMatrix (terms: 54690, documents: 911)>>
Non-/sparse entries: 545261/49277329
Sparsity           : 99%
Maximal term length: 33
Weighting          : term frequency (tf)
```

frequent terms

Now we can have a look at the popular words in the term-document matrix,

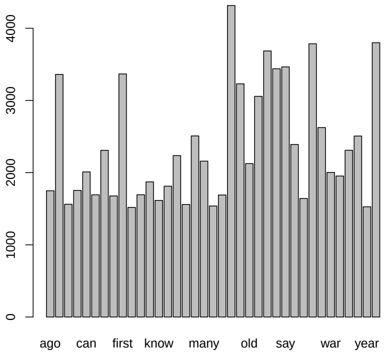
```
> (tt <- findFreqTerms(tdm, lowfreq=1500))
 [1] "ago"       "american" "called"
 [4] "came"      "can"      "country"
 [7] "day"       "every"    "first"
[10] "going"     "house"    "just"
[13] "know"      "last"     "like"
[16] "long"      "man"      "many"
[19] "much"      "never"    "new"
[22] "now"       "old"      "people"
[25] "president" "said"     "say"
[28] "states"    "think"    "time"
[31] "united"    "war"      "way"
[34] "well"      "will"     "year"
[37] "years"
```

frequent terms

Note that the frequent terms are ordered alphabetically, instead of by frequency or popularity. To show the top frequent words visually, we make a barplot of them.

```
> termFrequency <-
+   rowSums(as.matrix(tdm[tt,]))
> library(ggplot2)
> barplot(termFrequency)
> # qplot(names(termFrequency), termFrequency,
> #       geom="bar", stat="identity")  +
> #   coord_flip()
```

frequent term bar chart

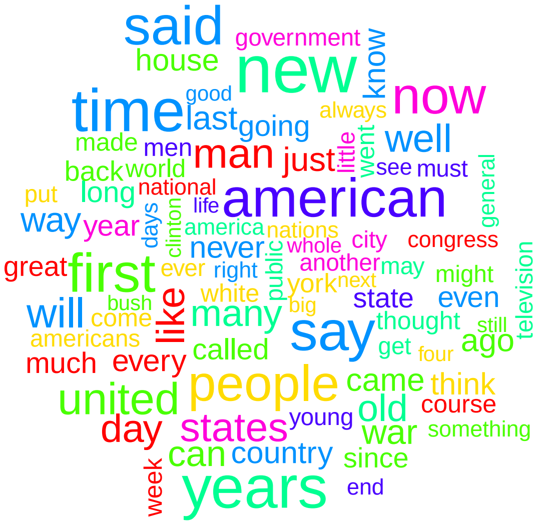


wordclouds

We can show the importance of words pictorally with a *wordcloud* [Fellows, 2012]. In the code below, we first convert the term-document matrix to a normal matrix, and then calculate word frequencies. After that we use `wordcloud` to make a pictorial.

```
> tdmat = as.matrix(tdm)
> # calculate the frequency of words
> v = sort(rowSums(tdmat), decreasing=TRUE)
> d = data.frame(word=names(v), freq=v)
> # generate the wordcloud
> library(wordcloud)
> wordcloud(d$word, d$freq, min.freq=900,
+           random.color=TRUE,colors=rainbow(7))
```

wordcloud pictorial



clustering the words

We now try to find clusters of words with hierarchical clustering.

Sparse terms are removed, so that the plot of clustering will not be crowded with words.

Then the distances between terms are calculated with `dist()` after scaling.

After that, the terms are clustered with `hclust()` and the dendrogram is cut into 10 clusters.

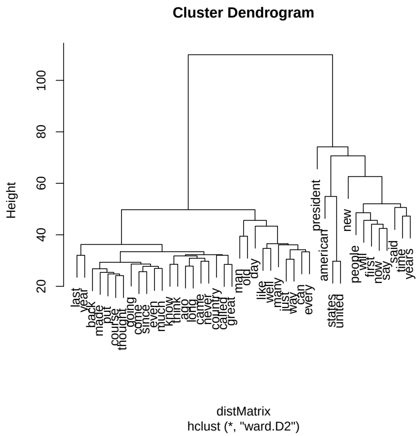
The agglomeration method is set to `ward`, which denotes the increase in variance when two clusters are merged.

Some other options are single linkage, complete linkage, average linkage, median and centroid.

word clustering code

```
> # remove sparse terms
> tdmat <- as.matrix(
+   removeSparseTerms(tdm, sparse=0.3)
+ )
> # compute distances
> distMatrix <- dist(scale(tdmat))
> fit <- hclust(distMatrix, method="ward.D2")
> plot(fit)
```

word clustering dendogram



clustering documents with k-medoids

We now try *k-medoids* clustering with the `Partitioning Around Medoids` algorithm.

In the following example, we use function `pamk()` from package `fpc` [Hennig, 2010], which calls the function `pam()` with the number of clusters estimated by optimum average silhouette.

Note that because we are now clustering documents rather than words we must first transpose the term-document matrix to a document-term matrix.

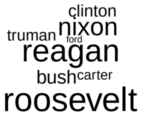
k-medoid clustering

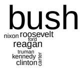
```
> # first select terms corresponding to presidents
> pn <- c("nixon","carter","reagan","clinton",
+         "roosevelt","kennedy",
+         "truman","bush","ford")
> # and transpose the reduced term-document matrix
> dtm <- t(as.matrix(tdm[pn,]))
> # find clusters
> library(fpc)
> library(cluster)
> pamResult <- pamk(dtm, krange=2:6,
+                   metric = "manhattan")
> # number of clusters identified
> (k <- pamResult$nc)

[1] 3

>
```

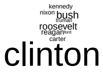
generate cluster wordclouds

```
> layout(matrix(c(1,2,3),1,3))
> for(k in 1:3){
+ cl <- which( pamResult$pamobject$clustering == k )
+ tdmk <- t(dtm[cl,])
+ v = sort(rowSums(tdmk), decreasing=TRUE)
+ d = data.frame(word=names(v), freq=v)
+ # generate the wordcloud
+ wordcloud(d$word, d$freq, min.freq=5,
+           random.color=TRUE,colors="black")
+ }
> layout(matrix(1))
```

cluster wordclouds







Natural Language Processing

These clustering techniques are only a small part of what is generally called *Natural Language Processing* which is a topic that would take another semester to get to grips with.

For a brief introduction to NLP and the problems it deals with sign up for the free Stanford *coursera* module on Natural Language Processing.

You can view the introductory lecture (without signing up) at: `https://class.coursera.org/nlp/lecture/124`

Sentiment Analysis

One of the sections in the NLP *coursera* module describes a field known as *sentiment analysis*. One can either, by hand, pre-label a set of training documents as being **positive** or **negative** in sentiment. Or one can pre-grade by hand documents on a scale of 1 to 5 say, on a *sentiment* scale.

Then by employing supervised learning techniques such as *decision trees* or the so called *Naive Bayes* algorithm, one can build a model to be used in the classification of new documents.

In these cases the document word matrix provides a *bag of words* vector associated with each document and the sentiment classifiactions provide a tartget variable which is used to train the classifier. See the `RTextTools` package on `CRAN` and read their paper.

Sentiment Lexicons

Another approach to predicting sentiment scores for a document is to acquire a *lexicon* of commonly used sentiment contributing words and then use the training set of documents to compute the *likelihood* of any word from the lexicon being in a particular class.

If $f(w, c)$ is the frequency of word w in class c then, the likelihood of word w being in class c is given by:

$$P(w|c) = \frac{f(w, c)}{\sum_{u \in c} f(u, c)}$$

and we can then use a *scaled likelihood* to compute the class of a document, (see next slide).

Sentiment Lexicons....

If C is the set of all possible classes then the class of a new document D can be computed using a *Naive Bayes* method as:

$$\text{class}(D) = \underset{c_j \in C}{\text{argmax}} P(c_j) \prod_{w_i \in D} \frac{P(w_i, c_j)}{P(w_i)}$$

This is just the start of the *Naive Bayes* story as you will find out if you listen to the Stanford lectures. A *smoothing* trick is required to deal with cases when $f(w, c) = 0$ and computations must be carried out in log space in order to avoid underflow.

Sentiment Lexicons....

Another approach is to obtain a vocabulary, such as the ANEW lexicon, where each of the words has had a *sentiment valence* on a scale of 0-1 assigned. These valences are assigned by human evaluations or learnt via machine learning from texts that have had valences assigned by humans.

To estimate the overall valence of a text then one calculates a weighted average of the valence of the ANEW study words appearing in the text.

$$v_{text} = \frac{\sum_{i=1}^{n} v_i f_i}{\sum_{i=1}^{n} f_i}$$

see the *Measuring Happiness* paper refered to in the last slide.

Sentiment Lexicons....

This simpler approach along with a time series technique called *Granger Causality* was used in a recent *arXiv* paper on predicting stock market trends from twitter sentiments.

exercises

Last lecture, no submissions this week. Study these slides and read the papers below in order to prepare for the final test.

`http://www.cs.ukzn.ac.za/~hughm/dm/docs/MeasuringHappiness.pdf`

`http://www.cs.ukzn.ac.za/~hughm/dm/docs/RTextToolsPaper.pdf`

`http://www.cs.ukzn.ac.za/~hughm/dm/docs/TwitterStockMarketPaper.pdf`

Good Luck