Text Analysis – Module 18

Vijayakumar Ganapathy, Mayur Srinivasan, Anagh Pal

August 12, 2015

In the previous session, we looked at the foundations of text analytics - how to represent documents, *bag of words*, data structures to store the bag of words, *document term matrix* and tokenization. In this session, we will be looking at Term Frequency, Inverse document frequency and Latent Semantic Indexing. We will simulatenously be running you through two R scripts - nyt_stories.R and tm.examples.R - to illustrate some of the concepts discussed.

Please note that all the functions used in nyt_stories.R are sourced from textutils.R. The ones used in tm_examples.R are from tm library. For more reading on tm_library, please refer http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf

Each corpus (a set of documents) has a property called TF-IDF (product of term frequency and inverse document frequency) which will enable us label or classify or identify patterns in the corpus.

Term Frequency:

Number of times a term occurs in a document standardized by the number of words in that document.

Inverse document frequency:

The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. Most of the informative words, the ones that contain the central theme of the document, do not appear as often as say conjunctions or prepositions do in a document. Inverse document frequency tries to mitigate this bias by essentially upweighting the rare words (or downweighting the more frequent words) in the TF-IDF product.

TF-IDF is essentially a product of these two metrics. Let us look at these mathematically!

Let $\mathbf{X_{ij}}$ be the raw count of words in document i for word j.

And let $\mathbf{N_{i\cdot}} = \sum_{j=1}^{D} X_{ij}$ (read as N-i-dot). It represents the total word count in each document

Let $\mathbf{M_j}$ be the number of documents where $\mathbf{X_{ij} > 0}$

We transform the word count matrix as below:

$$Y_{ij} = TF(X_{ij}) . IDF(X_{ij})$$

where $TF(X_{ij})$ is the term frequency and $IDF(X_{ij})$ is the inverse document frequency.

Term frequency is given by the formula:

$$TF(X_{ij}) = \frac{X_j}{N_{i.}}$$

Inverse document frequency is given by the formula:

$$IDF(X_{ij}) = \log(\frac{N}{M_j})$$

where $\mathbf{N}$ is the number of documents in the corpus (not to be confused with $\mathbf{N_{i.}}$)

To summarize the ideas

- TF-IDF, short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.
- The TF-IDF 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 corpus, which helps to adjust for the fact that some words appear more frequently in general.

Let us now see how they are calculated in R. Please refer to the previous scribe to get introduced to nyt_stories data. We will start with the document term matrix.

```
# Turn these lists into a document-term matrix
art_stories_DTM = make.BoW.frame(art_stories_vec_std)

# First 10 stories x 15 words
# Think about stemming?
art_stories_DTM[1:10,1:15]
```

```
##        # #a #d #doug #elizabeth #feng #gerald #h #interstate #jenny #laura
##  [1,] 12  0  0     0          0     0       0  0           0      0      0
##  [2,]  4  0  0     0          0     0       0  0           0      0      0
##  [3,] 17  0  0     0          0     0       0  0           0      0      0
##  [4,] 31  0  0     0          0     0       0  0           0      0      0
##  [5,]  8  0  0     0          0     0       0  0           0      0      0
##  [6,]  2  0  1     0          0     0       0  0           0      0      0
```

```
##  [7,] 30  0  1     0          0     0       0  0           0      0      0
##  [8,]  5  0  0     0          0     0       0  0           0      0      0
##  [9,]  1  0  0     0          0     0       0  0           0      0      0
## [10,] 19  0  0     0          0     0       0  0           0      0      0
##       #nd #on #over #popular
##  [1,]   0   0     0        0
##  [2,]   0   0     0        0
##  [3,]   0   0     0        0
##  [4,]   0   0     0        0
##  [5,]   0   0     0        0
##  [6,]   0   0     0        0
##  [7,]   0   0     0        0
##  [8,]   0   0     0        0
##  [9,]   0   0     1        0
## [10,]   0   0     0        0
```

The above shows the first 10 rows (each representing a document) and first 15 columns (each representing a word from the super-set of words - i.e. the set of words contained in all of the 57 documents put together).

Now let us go ahead and calculate TF and IDF for this document term matrix.

```
#Calculating TF of the DTM
art_stories_DTM_TF = art_stories_DTM / rowSums(art_stories_DTM)
art_stories_DTM_TF[1:10,1:15]
```

```
##                 # #a           #d #doug #elizabeth #feng #gerald #h
##  [1,] 0.005873715  0 0.0000000000     0          0     0       0  0
##  [2,] 0.008492569  0 0.0000000000     0          0     0       0  0
##  [3,] 0.013877551  0 0.0000000000     0          0     0       0  0
##  [4,] 0.022254128  0 0.0000000000     0          0     0       0  0
##  [5,] 0.006968641  0 0.0000000000     0          0     0       0  0
##  [6,] 0.001796945  0 0.0008984726     0          0     0       0  0
##  [7,] 0.010162602  0 0.0003387534     0          0     0       0  0
##  [8,] 0.031645570  0 0.0000000000     0          0     0       0  0
##  [9,] 0.002136752  0 0.0000000000     0          0     0       0  0
## [10,] 0.016183986  0 0.0000000000     0          0     0       0  0
##       #interstate #jenny #laura #nd #on       #over #popular
##  [1,]           0      0      0   0   0 0.000000000        0
##  [2,]           0      0      0   0   0 0.000000000        0
##  [3,]           0      0      0   0   0 0.000000000        0
##  [4,]           0      0      0   0   0 0.000000000        0
##  [5,]           0      0      0   0   0 0.000000000        0
##  [6,]           0      0      0   0   0 0.000000000        0
##  [7,]           0      0      0   0   0 0.000000000        0
##  [8,]           0      0      0   0   0 0.000000000        0
##  [9,]           0      0      0   0   0 0.002136752        0
## [10,]           0      0      0   0   0 0.000000000        0
```

As you can see this is just DTM normalized by rows (number of occurences of a word in a document divided by total number of words in that document)

```
#Calculating IDF of the DTM
art_stories_DTM_TFIDF = idf.weight(art_stories_DTM_TF)
art_stories_DTM_TFIDF[1:10, 1:15]
```

```
##                  # #a           #d #doug #elizabeth #feng #gerald #h
##  [1,] 1.039623e-04  0 0.0000000000     0          0     0       0  0
##  [2,] 1.503149e-04  0 0.0000000000     0          0     0       0  0
##  [3,] 2.456268e-04  0 0.0000000000     0          0     0       0  0
##  [4,] 3.938887e-04  0 0.0000000000     0          0     0       0  0
##  [5,] 1.233420e-04  0 0.0000000000     0          0     0       0  0
##  [6,] 3.180517e-05  0 0.0021865349     0          0     0       0  0
##  [7,] 1.798738e-04  0 0.0008243948     0          0     0       0  0
##  [8,] 5.601132e-04  0 0.0000000000     0          0     0       0  0
##  [9,] 3.781961e-05  0 0.0000000000     0          0     0       0  0
## [10,] 2.864497e-04  0 0.0000000000     0          0     0       0  0
##       #interstate #jenny #laura #nd #on       #over #popular
##  [1,]           0      0      0   0   0 0.000000000        0
##  [2,]           0      0      0   0   0 0.000000000        0
##  [3,]           0      0      0   0   0 0.000000000        0
##  [4,]           0      0      0   0   0 0.000000000        0
##  [5,]           0      0      0   0   0 0.000000000        0
##  [6,]           0      0      0   0   0 0.000000000        0
##  [7,]           0      0      0   0   0 0.000000000        0
##  [8,]           0      0      0   0   0 0.000000000        0
##  [9,]           0      0      0   0   0 0.008638998        0
## [10,]           0      0      0   0   0 0.000000000        0
```

The function idf.weight takes the DTM's TF matrix as the input and gives the TF-IDF product matrix as the output.

Our original objective was to find a group of articles from a corpus that belong to the same genre. How do we go about that?

- We identify the most important words from each article - we have done this by calculating TF-IDF.
- now we try to identify the high-importance words from each of the document and compare them with those from others.

But is this robust? Should all articles, say, about romantic comedies contain the word romantic? How do we work around this? We need to find words semantically related to 'romantic' or 'comedies' and so on. This is where Latent Semantic Analysis/Indexing (LSA/LSI) kicks in.

- LSI is based on the principle that words that are used in the same contexts tend to have similar meanings. A key feature of LSI is its ability to extract the conceptual content of a body of text by establishing associations between those terms that occur in similar contexts.
- The method uncovers the underlying latent semantic structure in the usage of words in a body of text and how it can be used to extract the meaning of the text in response to user queries, commonly referred to as concept searches. Queries, or concept

searches, against a set of documents that have undergone LSI will return results that are conceptually similar in meaning to the search criteria even if the results don't share a specific word or words with the search criteria.

Now that we have calculated the TF-IDF matrix, let us do a LSI on this matrix.

```
# Run PCA on Term-frequency matrix
lsi_art = prcomp(art_stories_DTM_TFIDF, scale.=FALSE)
```

We are here doing a Principal component analysis to identify the major semantic indices. All articles in the same genre should contain a similar set of words though with varying frequencies. But these words will all have similar TF-IDF scores and hence the combination of these words will give a principal component with high variance across these documents. The PCA does just that. It will give us a set of words that are used in similar context across all documents by comparing words from different documents with TF-IDF scores in the similar range. Please note we are not scaling here since we use the TF-IDF scores as such to identify the semantic indices.

Let us have a look at some of the important words picked by the first principal component.

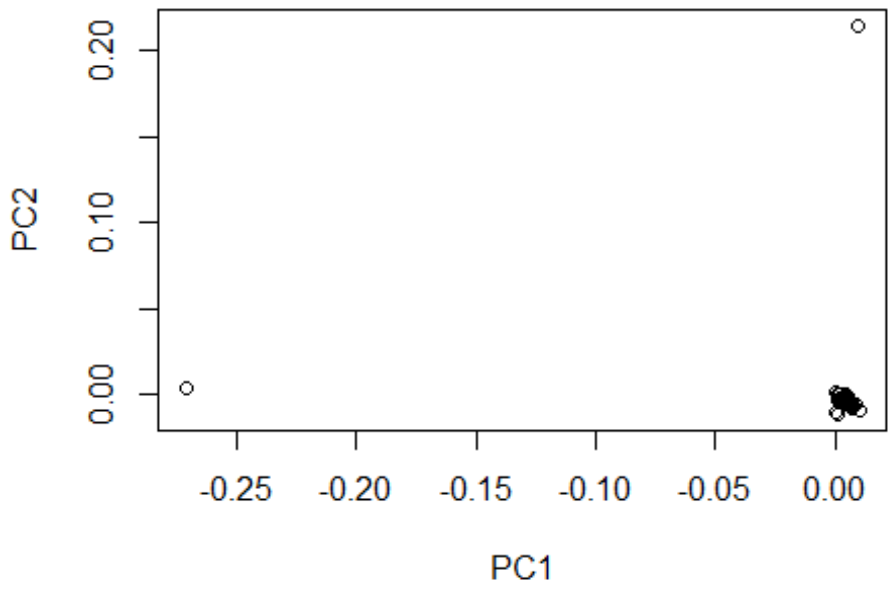
```
head(sort(lsi_art$rotation[,1], decreasing=FALSE),60)
```

```
##      memorial        statue        queens       service  metropolitan
## -3.738331e-01 -3.680079e-01 -3.095011e-01 -2.242164e-01 -1.933623e-01
##       dispute         rabin         slain      undraped       yitzhak
## -1.869166e-01 -1.869166e-01 -1.869166e-01 -1.869166e-01 -1.869166e-01
##          zeus  neighborhood     misstated       article         leave
## -1.869166e-01 -1.547418e-01 -1.547322e-01 -1.546395e-01 -1.542200e-01
##       israeli        report          lent        museum      decision
## -1.540540e-01 -1.359199e-01 -1.358381e-01 -1.276675e-01 -1.224634e-01
##      minister     displayed         prime         should           role
## -1.224415e-01 -1.224111e-01 -1.222455e-01 -1.120132e-01 -1.119711e-01
##         naked      involved          move       whether        during
## -1.107929e-01 -9.621554e-02 -9.594645e-02 -9.588870e-02 -9.580953e-02
##        sunday          part          last       another         about
## -8.307828e-02 -5.388397e-02 -4.367305e-02 -3.366376e-02 -2.961053e-02
##          over           was           not            be           art
## -1.886067e-02 -1.341755e-02 -1.178046e-02 -1.157894e-02 -5.221078e-03
##           but            at            an           for            to
## -4.967605e-03 -3.927583e-03 -3.825266e-03 -2.453394e-03 -1.215374e-03
##             a           and            in            of           the
## -7.693095e-17  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##       #gerald       aborted         agent      allusion     assertion
##  5.594385e-05  5.594385e-05  5.594385e-05  5.594385e-05  5.594385e-05
##     authority        ballet       ballets         baron        behest
##  5.594385e-05  5.594385e-05  5.594385e-05  5.594385e-05  5.594385e-05
```

By looking at the above words, it looks like these articles look like they are about art galleries.

Now let us plot the 57 documents by the first 2 principal components and see if we see a pattern.

```
# Scores on the first two PCs
plot(lsi_art$x[,1:2])
```



Oh yes we do! We can clearly see PC1 and PC2 has grouped all the documents except 2. Hence they should not be belonging to this genre. Let us verify that!

```
identify(lsi_art$x[,1:2], n=2)
```

```
head(art_stories[[16]],100)
```

```
##  [1] "an"           "article"      "in"           "the"
##  [5] "neighborhood" "report"       "last"         "sunday"
##  [9] "about"        "a"            "dispute"      "over"
## [13] "whether"      "a"            "naked"        "statue"
## [17] "of"           "zeus"         "at"           "the"
## [21] "queens"       "museum"       "of"           "art"
## [25] "should"       "be"           "displayed"    "during"
## [29] "a"            "memorial"     "service"      "for"
## [33] "the"          "slain"        "israeli"      "prime"
## [37] "minister"     "yitzhak"      "rabin"        "misstated"
## [41] "the"          "role"         "of"           "the"
## [45] "metropolitan" "museum"       "the"          "metropolitan"
## [49] "lent"         "the"          "statue"       "to"
## [53] "the"          "queens"       "museum"       "but"
## [57] "was"          "not"          "involved"     "in"
```

```
## [61] "the"          "decision"     "to"           "leave"
## [65] "the"          "statue"       "undraped"     "and"
## [69] "move"         "the"          "memorial"     "service"
## [73] "to"           "another"      "part"         "of"
## [77] "the"          "museum"
```

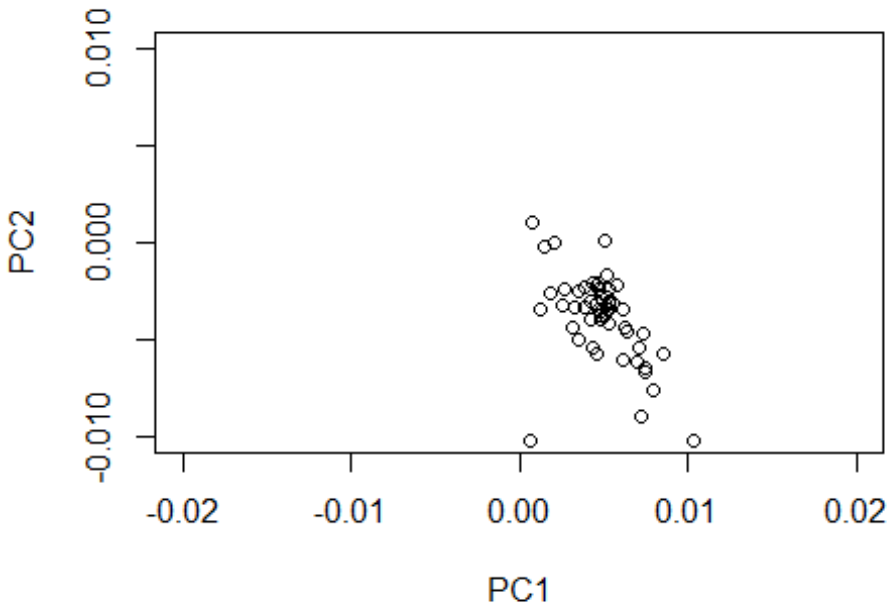
```
head(art_stories[[39]],100)
```

```
##   [1] "one"           "of"            "the"           "worlds"
##   [5] "most"          "successful"    "partnerships"  "won"
##   [9] "another"       "major"         "title"         "here"
##  [13] "on"            "sunday"        "night"         "at"
##  [17] "the"           "american"      "contract"      "bridge"
##  [21] "leagues"       "summer"        "national"      "championships"
##  [25] "the"           "life"          "master"        "pairs"
##  [29] "which"         "has"           "a"             "#"
##  [33] "year"          "history"       "was"           "won"
##  [37] "by"            "a"             "wide"          "margin"
##  [41] "by"            "robert"        "levin"         "of"
##  [45] "riverdale"     "n"             "y"             "and"
##  [49] "steve"         "weinstein"     "of"            "glen"
##  [53] "ridge"         "n"             "j"             "both"
##  [57] "have"          "won"           "the"           "event"
##  [61] "with"          "other"         "partners"      "and"
##  [65] "together"      "they"          "won"           "the"
##  [69] "#"             "cavendish"     "pairs"         "in"
##  [73] "las"           "vegas"         "these"         "were"
##  [77] "the"           "final"         "standings"     "first"
##  [81] "levin"         "and"           "weinstein"     "#"
##  [85] "#"             "#"             "match"         "points"
##  [89] "second"        "gary"          "cohler"        "of"
##  [93] "highland"      "park"          "ill"           "and"
##  [97] "ralph"         "katz"          "of"            "hinsdale"
```

The former is not about an art gallery, rather about a dispute at a museum. The latter is about a bridge championship.

Now let us look at two nearby stories

```
# Two nearby stories?
# Same plot but zoomed in on the cluster
plot(lsi_art$x[,1:2], xlim=c(-0.02, 0.02), ylim=c(-0.01, 0.01))
```




```
identify(lsi_art$x[,1:2], n=2)
```

```
head(art_stories[[12]],100)
```

```
##   [1] "elizabeth"    "murraypaula"  "cooper"       "gallery"
##   [5] "#"            "wooster"      "street"       "at"
##   [9] "houston"      "street"       "soho"         "through"
##  [13] "april"        "#elizabeth"   "murrays"      "faith"
##  [17] "in"           "the"          "complete"     "malleability"
##  [21] "of"           "paint"        "and"          "painting"
##  [25] "continues"    "unabated"     "in"           "this"
##  [29] "small"        "impressive"   "show"         "over"
##  [33] "the"          "course"       "of"           "only"
##  [37] "five"         "canvases"     "she"          "puts"
##  [41] "her"          "art"          "through"      "its"
##  [45] "paces"        "going"        "from"         "flat"
##  [49] "to"           "shaped"       "surfaces"     "and"
##  [53] "shifting"     "her"          "semi"         "abstract"
##  [57] "narratives"   "from"         "human"        "to"
##  [61] "animal"       "to"           "architecture" "all"
##  [65] "without"      "missing"      "a"            "beat"
##  [69] "it"           "helps"        "that"         "ms"
##  [73] "murray"       "has"          "downsized"    "things"
##  [77] "a"            "bit"          "giving"       "these"
##  [81] "works"        "a"            "compressed"   "clarity"
##  [85] "of"           "shape"        "color"        "and"
##  [89] "paint"        "handling"     "that"         "her"
```

```
##  [93] "larger"       "more"         "looming"      "efforts"
##  [97] "can"          "sometimes"    "lack"         "this"
```

```
head(art_stories[[17]],100)
```

```
##   [1] "laura"         "newman"        "tenri"         "cultural"
##   [5] "institute"     "#"             "broadway"      "at"
##   [9] "prince"        "street"        "soho"          "through"
##  [13] "july"          "#laura"        "newmans"       "new"
##  [17] "paintings"     "while"         "still"         "lacking"
##  [21] "in"            "originality"   "are"           "a"
##  [25] "big"           "improvement"   "over"          "her"
##  [29] "earlier"       "landscapelike" "abstractions"  "which"
##  [33] "often"         "seemed"        "forced"        "and"
##  [37] "heavy"         "handed"        "the"           "hints"
##  [41] "of"            "landscapes"    "remain"        "but"
##  [45] "the"           "pale"          "colors"        "and"
##  [49] "map"           "like"          "compositions"  "suggest"
##  [53] "a"             "realm"         "high"          "above"
##  [57] "the"           "earth"         "or"            "that"
##  [61] "of"            "the"           "imagination"   "a"
##  [65] "clear"         "source"        "is"            "#s"
##  [69] "new"           "image"         "painting"      "modified"
##  [73] "with"          "a"             "loose"         "childlike"
##  [77] "rendering"     "and"           "with"          "half"
##  [81] "buried"        "feminist"      "subject"       "matter"
##  [85] "that"          "seems"         "pure"          "#s"
##  [89] "one"           "can"           "imagine"       "ms"
##  [93] "newman"        "learning"      "from"          "such"
##  [97] "seemingly"     "unrelated"     "precedents"    "as"
```

These stories can be clearly identified as articles on art galleries.

Now let us look at another script tm_examples.R using tm library to understand how tm implements the same functionalities we used here.

The tm library and related plugins comprise R's most popular text-mining stack.

(See http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf)

```
## Loading required package: NLP
```

tm has many "reader" functions. Each one has arguments elem, language, id (see ?readPlain,?readPDF,etc)

The following is a function that wraps another function around readPlan to read plain text documents in English.

```
readerPlain = function(fname){
                readPlain(elem=list(content=readLines(fname)),
                          id=fname, language='en') }
```

We now apply this function to all of Simon Cowell's articles. We extract the names of all of his articles through the Sys.glob function

```
file_list = Sys.glob('../data/ReutersC50/C50train/SimonCowell/*.txt')
simon = lapply(file_list, readerPlain)
```

The names for these articles can be changed in many ways. Some examples are given below

```
names(simon) = file_list
names(simon) = substring(names(simon),first=41)
names(simon) = sub('.txt', '', names(simon))
```

We now create a vector of all of Simon Cowell's articles. This will be the 'corpus' on which we will do our text mining (We have to change the names of the Corpus object separately because 'tm' doesn't do it on its own)

```
my_documents = Corpus(VectorSource(simon))
names(my_documents) = names(simon)
```

tm has some in-built tokenization and pre-processing functions that can be used easily without creating any functions of our own

```
my_documents = tm_map(my_documents, content_transformer(tolower)) # make
everything lowercase
my_documents = tm_map(my_documents, content_transformer(removeNumbers)) #
remove numbers
my_documents = tm_map(my_documents, content_transformer(removePunctuation)) #
remove punctuation
my_documents = tm_map(my_documents, content_transformer(stripWhitespace))
```

We also have access to some 'stopwords' list. It's always advisable to know exactly what these lists contain as it may do more harm than good if not used judiciously.

To remove the stopwords from our Corpus:

```
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

We now create a Document-Term Matrix using the following function:

```
DTM_simon = DocumentTermMatrix(my_documents)
DTM_simon

## <<DocumentTermMatrix (documents: 50, terms: 2960)>>
## Non-/sparse entries: 9829/138171
## Sparsity           : 93%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

As an aside, we can see that we have access to a sparse matrix format of DTM as well, which is useful to save space and presumably faster handling.

```
class(DTM_simon)
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

We can find the terms that have frequencies greater than a certain threshold, using the following (threshold being 50 in the case below):

```
findFreqTerms(DTM_simon, 50)
```

```
##  [1] "abbey"         "also"          "billion"       "british"
##  [5] "business"      "companies"     "company"       "financial"
##  [9] "group"         "industry"      "insurance"     "life"
## [13] "market"        "may"           "million"       "new"
## [17] "offer"         "one"           "pence"         "percent"
## [21] "policyholders" "pound"         "pounds"        "profits"
## [25] "said"          "scotam"        "scottish"      "shares"
## [29] "two"           "will"          "year"          "years"
```

We can also find words that correlate very well with a given input word

```
findAssocs(DTM_simon, "market", .5)
```

```
## $market
##       exchange      delighted         regime      abandoned          adapt
##           0.62           0.56           0.56           0.54           0.54
##       andersen     antiquated        attempt           bang          began
##           0.54           0.54           0.54           0.54           0.54
##          begun        bourses         budget        century         citing
##           0.54           0.54           0.54           0.54           0.54
##      completes       computer     consulting          coped         costly
##           0.54           0.54           0.54           0.54           0.54
##          crest    culmination       delivery   disagreement     electronic
##           0.54           0.54           0.54           0.54           0.54
##        entered      exchanges    exclusively    experienced          feeds
##           0.54           0.54           0.54           0.54           0.54
##       fortunes      fourstage           ftse          happy       illfated
##           0.54           0.54           0.54           0.54           0.54
##    implemented   introduction    involvement       lawrence   marketmakers
##           0.54           0.54           0.54           0.54           0.54
##          marks        michael  modernisation        ongoing    operational
##           0.54           0.54           0.54           0.54           0.54
##    orderdriven    outsourcing      paperless   participants          paves
##           0.54           0.54           0.54           0.54           0.54
##       platform        posting        project    quotedriven        rawlins
##           0.54           0.54           0.54           0.54           0.54
##         relief    replacement        retains        sacking          seats
##           0.54           0.54           0.54           0.54           0.54
##       sequence          shake         struck    succesfully   successfully
##           0.54           0.54           0.54           0.54           0.54
```

```
##         summer    supervising       talisman         taurus telephonebased
##           0.54           0.54           0.54           0.54           0.54
##      temporary      threeyear     transition     turnaround      unrelated
##           0.54           0.54           0.54           0.54           0.54
##          users       weekends         system       trading
##           0.54           0.54           0.51           0.51
```

Finally, drop those terms that only occur in one or two documents. This is a common step: the noise of the "long tail" (rare terms) can be huge, and there is nothing to learn if a term occurred once.

Below removes those terms that have count 0 in >95% of docs

```
DTM_simon = removeSparseTerms(DTM_simon, 0.95)
```

We now run a PCA on this matrix. But first, we scale the X's row-wise for term-frequency

```
X = as.matrix(DTM_simon)
X = X/rowSums(X)  # term-frequency weighting

pca_simon = prcomp(X, scale=TRUE)
```

We can observe the loadings on PC1 and PC2 below:

```
pca_simon$rotation[order(abs(pca_simon$rotation[,1]),decreasing=TRUE),1][1:25
]
```

```
##         bidders          scotam         scotams         arising
##      0.10100451      0.09458365      0.09406492      0.09316081
##        arranged       comparing            cuts     differently
##      0.09316081      0.09316081      0.09316081      0.09316081
##         dispute           doors        evaluate        original
##      0.09316081      0.09316081      0.09316081      0.09316081
##    preservation         resolve      structured        tribunal
##      0.09316081      0.09316081      0.09316081      0.09316081
##          boards             tsb    bristolbased       consulted
##      0.09275756      0.09262520      0.09178428      0.09178428
##         deflect confidentiality        deadline           dozen
##      0.09178428      0.09178080      0.09142929      0.09142929
##       sealedbid
##      0.09142929
```

```
pca_simon$rotation[order(abs(pca_simon$rotation[,2]),decreasing=TRUE),2][1:25
]
```

```
##          abi        tests      genetic      testing  association
##  -0.10021028  -0.09945929  -0.09841106  -0.09547578  -0.09456509
##  detrimental      infancy    statement developments      account
##  -0.09456509  -0.09456509  -0.09039547  -0.09028649  -0.08870777
##         soon         take      started         able      experts
##  -0.08656192  -0.08449580  -0.08395307  -0.08264067  -0.08230988
##      certain          run        risks       favour      without
```

```
##  -0.08147333  -0.08138826  -0.08108180  -0.08064827  -0.07837465
##     genetics        extra       decide     policies       unable
##  -0.07832995  -0.07821757  -0.07744595  -0.07734492  -0.07673484
```

We now plot PC1 and PC2 and then inspect some outliers. The idea is to see if there's any relation between the articles that load similarly on this plot

```
plot(pca_simon$x[,1:2], col='grey', pch=19, xlab="PCA 1 direction", ylab="PCA
2 direction", bty="n")
identify(pca_simon$x[,1:2], n=4)
```