Hands-On Data Science with R
Text Mining

Graham.Williams@togaware.com

10th January 2016

Visit http://HandsOnDataScience.com/ for more Chapters.

Text Mining (or Text Analytics) applies analytic tools to learn from collections of text data, like social media, books, newspapers, emails, etc. The goal can be considered to be similar to humans learning by reading such material. However, using automated algorithms we can learn from massive amounts of text, very much more than a human can. The material could consist of millions of newspaper articles to perhaps summarise the main themes and to identify those that are of most interest to particular people. Or we might be monitoring twitter feeds to identify emerging topics that we might need to act upon, as it emerges.

The required packages for this chapter include:

```
library(tm)                # Framework for text mining.
library(qdap)              # Quantitative discourse analysis of transcripts.
library(qdapDictionaries)
library(dplyr)             # Data wrangling, pipe operator %>%().
library(RColorBrewer)      # Generate palette of colours for plots.
library(ggplot2)           # Plot word frequencies.
library(scales)            # Include commas in numbers.
library(Rgraphviz)         # Correlation plots.
```

As we work through this chapter, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This chapter is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 Getting Started: The Corpus

The primary package for text mining, tm (Feinerer and Hornik, 2015), provides a framework within which we perform our text mining. A collection of other standard R packages add value to the data processing and visualizations for text mining.

The basic concept is that of a corpus. This is a collection of texts, usually stored electronically, and from which we perform our analysis. A corpus might be a collection of news articles from Reuters or the published works of Shakespeare. Within each corpus we will have separate documents, which might be articles, stories, or book volumes. Each document is treated as a separate entity or record.

Documents which we wish to analyse come in many different formats. Quite a few formats are supported by tm (Feinerer and Hornik, 2015), the package we will illustrate text mining with in this module. The supported formats include text, PDF, Microsoft Word, and XML.

A number of open source tools are also available to convert most document formats to text files. For our corpus used initially in this module, a collection of PDF documents were converted to text using pdftotext from the xpdf application which is available for GNU/Linux and MS/Windows and others. On GNU/Linux we can convert a folder of PDF documents to text with:

```
system("for f in *.pdf; do pdftotext -enc ASCII7 -nopgbrk $f; done")
```

The -enc ASCII7 ensures the text is converted to ASCII since otherwise we may end up with binary characters in our text documents.

We can also convert Word documents to text using anitword, which is another application available for GNU/Linux.

```
system("for f in *.doc; do antiword $f; done")
```

1.1 Corpus Sources and Readers

There are a variety of sources supported by tm. We can use getSources() to list them.

```
getSources()

## [1] "DataframeSource" "DirSource"       "URISource"       "VectorSource"
## [5] "XMLSource"       "ZipSource"
```

In addition to different kinds of sources of documents, our documents for text analysis will come in many different formats. A variety are supported by tm:

```
getReaders()

##  [1] "readDOC"                 "readPDF"
##  [3] "readPlain"               "readRCV1"
##  [5] "readRCV1asPlain"         "readReut21578XML"
##  [7] "readReut21578XMLasPlain" "readTabular"
##  [9] "readTagged"              "readXML"
```

1.2 Text Documents

We load a sample corpus of text documents. Our corpus consists of a collection of research papers all stored in the folder we identify below. To work along with us in this module, you can create your own folder called `corpus/txt` and place into that folder a collection of text documents. It does not need to be as many as we use here but a reasonable number makes it more interesting.

```
cname <- file.path(".", "corpus", "txt")
cname

## [1] "./corpus/txt"
```

We can list some of the file names.

```
length(dir(cname))

## [1] 46

dir(cname)

##  [1] "acnn96.txt"
##  [2] "adm02.txt"
##  [3] "ai02.txt"
##  [4] "ai03.txt"
##  [5] "ai97.txt"
##  [6] "atobmars.txt"
....
```

There are 46 documents in this particular corpus.

After loading the `tm` (Feinerer and Hornik, 2015) package into the R library we are ready to load the files from the directory as the source of the files making up the corpus, using `DirSource()`. The source object is passed on to `Corpus()` which loads the documents. We save the resulting collection of documents in memory, stored in a variable called `docs`.

```
library(tm)
docs <- Corpus(DirSource(cname))
docs

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 46

class(docs)

## [1] "VCorpus" "Corpus"

class(docs[[1]])

## [1] "PlainTextDocument" "TextDocument"

summary(docs)

##                                        Length Class             Mode
## acnn96.txt                             2      PlainTextDocument list
## adm02.txt                              2      PlainTextDocument list
## ai02.txt                          2      PlainTextDocument list
## ai03.txt                          2      PlainTextDocument list
## ai97.txt                          2      PlainTextDocument list
....
```

1.3 PDF Documents

If instead of text documents we have a corpus of PDF documents then we can use the `readPDF()` reader function to convert PDF into text and have that loaded as out Corpus.

```
docs <- Corpus(DirSource(cname), readerControl=list(reader=readPDF))
```

This will use, by default, the `pdftotext` command from `xpdf` to convert the PDF into text format. The `xpdf` application needs to be installed for `readPDF()` to work.

1.4 Word Documents

A simple open source tool to convert Microsoft Word documents into text is `antiword`. The separate `antiword` application needs to be installed, but once it is available it is used by `tm` to convert Word documents into text for loading into R.

To load a corpus of Word documents we use the `readDOC()` reader function:

```
docs <- Corpus(DirSource(cname), readerControl=list(reader=readDOC))
```

Once we have loaded our corpus the remainder of the processing of the corpus within R is then as follows.

The `antiword` program takes some useful command line arguments. We can pass these through to the program from `readDOC()` by specifying them as the character string argument:

```
docs <- Corpus(DirSource(cname), readerControl=list(reader=readDOC("-r -s")))
```

Here, `-r` requests that removed text be included in the output, and `-s` requests that text hidden by Word be included.

2 Exploring the Corpus

We can (and should) inspect the documents using `inspect()`. This will assure us that data has been loaded properly and as we expect.

```
inspect(docs[16])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 44776
```

```
viewDocs <- function(d, n) {d %>% extract2(n) %>% as.character() %>% writeLines()}
viewDocs(docs, 16)
```

```
## Hybrid weighted random forests for
## classifying very high-dimensional data
## Baoxun Xu1 , Joshua Zhexue Huang2 , Graham Williams2 and
## Yunming Ye1
## 1
##
....
```

3 Preparing the Corpus

We generally need to perform some pre-processing of the text data to prepare for the text analysis. Example transformations include converting the text to lower case, removing numbers and punctuation, removing stop words, stemming and identifying synonyms. The basic transforms are all available within tm.

```
getTransformations()

## [1] "removeNumbers"     "removePunctuation" "removeWords"
## [4] "stemDocument"      "stripWhitespace"
```

The function `tm_map()` is used to apply one of these transformations across all documents within a corpus. Other transformations can be implemented using R functions and wrapped within `content_transformer()` to create a function that can be passed through to `tm_map()`. We will see an example of that in the next section.

In the following sections we will apply each of the transformations, one-by-one, to remove unwanted characters from the text.

3.1 Simple Transforms

We start with some manual special transforms we may want to do. For example, we might want to replace "/", used sometimes to separate alternative words, with a space. This will avoid the two words being run into one string of characters through the transformations. We might also replace "@" and "|" with a space, for the same reason.

To create a custom transformation we make use of `content_transformer()` to create a function to achieve the transformation, and then apply it to the corpus using `tm_map()`.

```
toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
docs <- tm_map(docs, toSpace, "/")
docs <- tm_map(docs, toSpace, "@")
docs <- tm_map(docs, toSpace, "\\|")
```

This can be done with a single call:

```
docs <- tm_map(docs, toSpace, "/|@|\\|")
```

Check the email address in the following.

```
inspect(docs[16])

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 44776
```

3.2 Conversion to Lower Case

```
docs <- tm_map(docs, content_transformer(tolower))
```

```
inspect(docs[16])

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 44776
```

General character processing functions in R can be used to transform our corpus. A common requirement is to map the documents to lower case, using `tolower()`. As above, we need to wrap such functions with a `content_transformer()`:

3.3 Remove Numbers

```
docs <- tm_map(docs, removeNumbers)
```

```
viewDocs(docs, 16)

## hybrid weighted random forests for
## classifying very high-dimensional data
## baoxun xu , joshua zhexue huang , graham williams and
## yunming ye
##
##
## department of computer science, harbin institute of technology shenzhen gr...
## school, shenzhen , china
##
## shenzhen institutes of advanced technology, chinese academy of sciences, s...
## , china
## email: amusing gmail.com
## random forests are a popular classification method based on an ensemble of a
## single type of decision trees from subspaces of data. in the literature, t...
## are many different types of decision tree algorithms, including c., cart, and
## chaid. each type of decision tree algorithm may capture different information
## and structure. this paper proposes a hybrid weighted random forest algorithm,
## simultaneously using a feature weighting method and a hybrid forest method to
## classify very high dimensional data. the hybrid weighted random forest alg...
## can effectively reduce subspace size and improve classification performance
## without increasing the error bound. we conduct a series of experiments on ...
## high dimensional datasets to compare our method with traditional random fo...
## methods and other classification methods. the results show that our method
## consistently outperforms these traditional methods.
## keywords: random forests; hybrid weighted random forest; classification; d...
##
....
```

Numbers may or may not be relevant to our analyses. This transform can remove numbers simply.

3.4 Remove Punctuation

```
docs <- tm_map(docs, removePunctuation)
```

```
viewDocs(docs, 16)

## hybrid weighted random forests for
## classifying very highdimensional data
## baoxun xu  joshua zhexue huang  graham williams and
## yunming ye
##
##
## department of computer science harbin institute of technology shenzhen gra...
## school shenzhen  china
##
## shenzhen institutes of advanced technology chinese academy of sciences she...
##  china
## email amusing gmailcom
## random forests are a popular classification method based on an ensemble of a
## single type of decision trees from subspaces of data in the literature there
## are many different types of decision tree algorithms including c cart and
## chaid each type of decision tree algorithm may capture different information
## and structure this paper proposes a hybrid weighted random forest algorithm
## simultaneously using a feature weighting method and a hybrid forest method to
## classify very high dimensional data the hybrid weighted random forest algo...
## can effectively reduce subspace size and improve classification performance
## without increasing the error bound we conduct a series of experiments on e...
## high dimensional datasets to compare our method with traditional random fo...
## methods and other classification methods the results show that our method
## consistently outperforms these traditional methods
## keywords random forests hybrid weighted random forest classification decis...
##
....
```

Punctuation can provide gramatical context which supports understanding. Often for initial analyses we ignore the punctuation. Later we will use punctuation to support the extraction of meaning.

3.5 Remove English Stop Words

```
docs <- tm_map(docs, removeWords, stopwords("english"))
```

```
inspect(docs[16])

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 32234
```

Stop words are common words found in a language. Words like *for*, *very*, *and*, *of*, *are*, etc, are common stop words. Notice they have been removed from the above text.

We can list the stop words:

```
length(stopwords("english"))

## [1] 174

stopwords("english")

##   [1] "i"          "me"         "my"         "myself"     "we"
##   [6] "our"        "ours"       "ourselves"  "you"        "your"
##  [11] "yours"      "yourself"   "yourselves" "he"         "him"
##  [16] "his"        "himself"    "she"        "her"        "hers"
##  [21] "herself"    "it"         "its"        "itself"     "they"
##  [26] "them"       "their"      "theirs"     "themselves" "what"
....
```

3.6 Remove Own Stop Words

```
docs <- tm_map(docs, removeWords, c("department", "email"))
```

```
viewDocs(docs, 16)

## hybrid weighted random forests
## classifying  highdimensional data
## baoxun xu  joshua zhexue huang  graham williams
## yunming ye
##
##
##   computer science harbin institute  technology shenzhen graduate
## school shenzhen  china
##
## shenzhen institutes  advanced technology chinese academy  sciences shenzhen
##  china
##  amusing gmailcom
## random forests   popular classification method based   ensemble
## single type  decision trees  subspaces  data   literature
##  many different types  decision tree algorithms including c cart
## chaid  type  decision tree algorithm may capture different information
##  structure  paper proposes  hybrid weighted random forest algorithm
## simultaneously using  feature weighting method   hybrid forest method
## classify  high dimensional data  hybrid weighted random forest algorithm
## can effectively reduce subspace size  improve classification performance
## without increasing  error bound  conduct  series  experiments  eight
## high dimensional datasets  compare  method  traditional random forest
## methods   classification methods  results show   method
## consistently outperforms  traditional methods
## keywords random forests hybrid weighted random forest classification decis...
##
....
```

Previously we used the English stopwords provided by tm. We could instead or in addition remove our own stop words as we have done above. We have chosen here two words, simply for illustration. The choice might depend on the domain of discourse, and might not become apparent until we've done some analysis.

3.7 Strip Whitespace

```
docs <- tm_map(docs, stripWhitespace)
```

```
viewDocs(docs, 16)

## hybrid weighted random forests
## classifying highdimensional data
## baoxun xu joshua zhexue huang graham williams
## yunming ye
##
##
##  computer science harbin institute technology shenzhen graduate
## school shenzhen china
##
## shenzhen institutes advanced technology chinese academy sciences shenzhen
##  china
##  amusing gmailcom
## random forests popular classification method based ensemble
## single type decision trees subspaces data literature
##  many different types decision tree algorithms including c cart
## chaid type decision tree algorithm may capture different information
##  structure paper proposes hybrid weighted random forest algorithm
## simultaneously using feature weighting method hybrid forest method
## classify high dimensional data hybrid weighted random forest algorithm
## can effectively reduce subspace size improve classification performance
## without increasing error bound conduct series experiments eight
## high dimensional datasets compare method traditional random forest
## methods classification methods results show method
## consistently outperforms traditional methods
## keywords random forests hybrid weighted random forest classification decis...
##
....
```

3.8 Specific Transformations

We might also have some specific transformations we would like to perform. The examples here may or may not be useful, depending on how we want to analyse the documents. This is really for illustration using the part of the document we are looking at here, rather than suggesting this specific transform adds value.

```
toString <- content_transformer(function(x, from, to) gsub(from, to, x))
docs <- tm_map(docs, toString, "harbin institute technology", "HIT")
docs <- tm_map(docs, toString, "shenzhen institutes advanced technology", "SIAT")
docs <- tm_map(docs, toString, "chinese academy sciences", "CAS")
```

```
inspect(docs[16])

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 30117
```

3.9 Stemming

```
docs <- tm_map(docs, stemDocument)
```

```
viewDocs(docs, 16)

## hybrid weight random forest
## classifi highdimension data
## baoxun xu joshua zhexu huang graham william
## yunm ye
##
##
##  comput scienc HIT shenzhen graduat
## school shenzhen china
##
## SIAT CAS shenzhen
##  china
##  amus gmailcom
## random forest popular classif method base ensembl
## singl type decis tree subspac data literatur
##  mani differ type decis tree algorithm includ c cart
## chaid type decis tree algorithm may captur differ inform
##  structur paper propos hybrid weight random forest algorithm
## simultan use featur weight method hybrid forest method
## classifi high dimension data hybrid weight random forest algorithm
## can effect reduc subspac size improv classif perform
## without increas error bound conduct seri experi eight
## high dimension dataset compar method tradit random forest
## method classif method result show method
## consist outperform tradit method
## keyword random forest hybrid weight random forest classif decis tree
##
....
```

Stemming uses an algorithm that removes common word endings for English words, such as “es”, “ed” and “’s”.

4 Creating a Document Term Matrix

A document term matrix is simply a matrix with documents as the rows and terms as the columns and a count of the frequency of words as the cells of the matrix. We use `DocumentTermMatrix()` to create the matrix:

```
dtm <- DocumentTermMatrix(docs)

dtm

## <<DocumentTermMatrix (documents: 46, terms: 6508)>>
## Non-/sparse entries: 30061/269307
## Sparsity           : 90%
## Maximal term length: 56
## Weighting          : term frequency (tf)
```

We can inspect the document term matrix using `inspect()`. Here, to avoid too much output, we select a subset of inspect.

```
inspect(dtm[1:5, 1000:1005])

## <<DocumentTermMatrix (documents: 5, terms: 6)>>
## Non-/sparse entries: 7/23
## Sparsity           : 77%
## Maximal term length: 9
## Weighting          : term frequency (tf)
##
....
```

The document term matrix is in fact quite sparse (that is, mostly empty) and so it is actually stored in a much more compact representation internally. We can still get the row and column counts.

```
class(dtm)

## [1] "DocumentTermMatrix"    "simple_triplet_matrix"

dim(dtm)

## [1]   46 6508
```

The transpose is created using `TermDocumentMatrix()`:

```
tdm <- TermDocumentMatrix(docs)
tdm

## <<TermDocumentMatrix (terms: 6508, documents: 46)>>
## Non-/sparse entries: 30061/269307
## Sparsity           : 90%
## Maximal term length: 56
## Weighting          : term frequency (tf)
```

We will use the document term matrix for the remainder of the chapter.

5 Exploring the Document Term Matrix

We can obtain the term frequencies as a vector by converting the document term matrix into a matrix and summing the column counts:

```
freq <- colSums(as.matrix(dtm))
length(freq)

## [1] 6508
```

By ordering the frequencies we can list the most frequent terms and the least frequent terms:

```
ord <- order(freq)

# Least frequent terms.
freq[head(ord)]

## aaaaaaeaceeaeeieaeaeeiiaiaciaiicaiaeaeaoeneiacaeaaeooooo
##                                                        1
##                                                      aab
##                                                        1
##                                                 aadrbltn
##                                                        1
##                                               aadrhtmliv
##                                                        1
##                                                      aai
##                                                        1
....
```

Notice these terms appear just once and are probably not really terms that are of interest to us. Indeed they are likely to be spurious terms introduced through the translation of the original document from PDF to text.

```
# Most frequent terms.
freq[tail(ord)]

##     can dataset pattern     use    mine    data
##     709     776     887    1366    1446    3101
```

These terms are much more likely to be of interest to us. Not surprising, given the choice of documents in the corpus, the most frequent terms are: data, mine, use, pattern, dataset, can.

6 Distribution of Term Frequencies

```
# Frequency of frequencies.
head(table(freq), 15)

## freq
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2381 1030  503  311  210  188  134  130   82   83   65   61   54   52   51

tail(table(freq), 15)

## freq
##  483  544  547  555  578  609  611  616  703  709  776  887 1366 1446 3101
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
```

So we can see here that there are 2381 terms that occur just once.

7 Conversion to Matrix and Save to CSV

We can convert the document term matrix to a simple matrix for writing to a CSV file, for example, for loading the data into other software if we need to do so. To write to CSV we first convert the data structure into a simple matrix:

```
m <- as.matrix(dtm)
dim(m)

## [1]   46 6508
```

For very large corpus the size of the matrix can exceed R's calculation limits. This will manifest itself as a integer overflow error with a message like:

```
## Error in vector(typeof(x$v), nr * nc) : vector size cannot be NA
## In addition: Warning message:
## In nr * nc : NAs produced by integer overflow
```

If this occurs, then consider removing sparse terms from the document term matrix, as we discuss shortly.

Once converted into a standard matrix the usual `write.csv()` can be used to write the data to file.

```
write.csv(m, file="dtm.csv")
```

8 Removing Sparse Terms

We are often not interested in infrequent terms in our documents. Such “sparse” terms can be removed from the document term matrix quite easily using `removeSparseTerms()`:

```
dim(dtm)

## [1]   46 6508

dtms <- removeSparseTerms(dtm, 0.1)
dim(dtms)

## [1] 46  6
```

This has removed most terms!

```
inspect(dtms)

## <<DocumentTermMatrix (documents: 46, terms: 6)>>
## Non-/sparse entries: 257/19
## Sparsity           : 7%
## Maximal term length: 7
## Weighting          : term frequency (tf)
##
....
```

We can see the effect by looking at the terms we have left:

```
freq <- colSums(as.matrix(dtms))
freq

##    data  graham  inform    time     use william
##    3101     108     467     483    1366     236

table(freq)

## freq
##  108  236  467  483 1366 3101
##    1    1    1    1    1    1
```

9 Identifying Frequent Items and Associations

One thing we often to first do is to get an idea of the most frequent terms in the corpus. We use **findFreqTerms()** to do this. Here we limit the output to those terms that occur at least 1,000 times:

```
findFreqTerms(dtm, lowfreq=1000)

## [1] "data" "mine" "use"
```

So that only lists a few. We can get more of them by reducing the threshold:

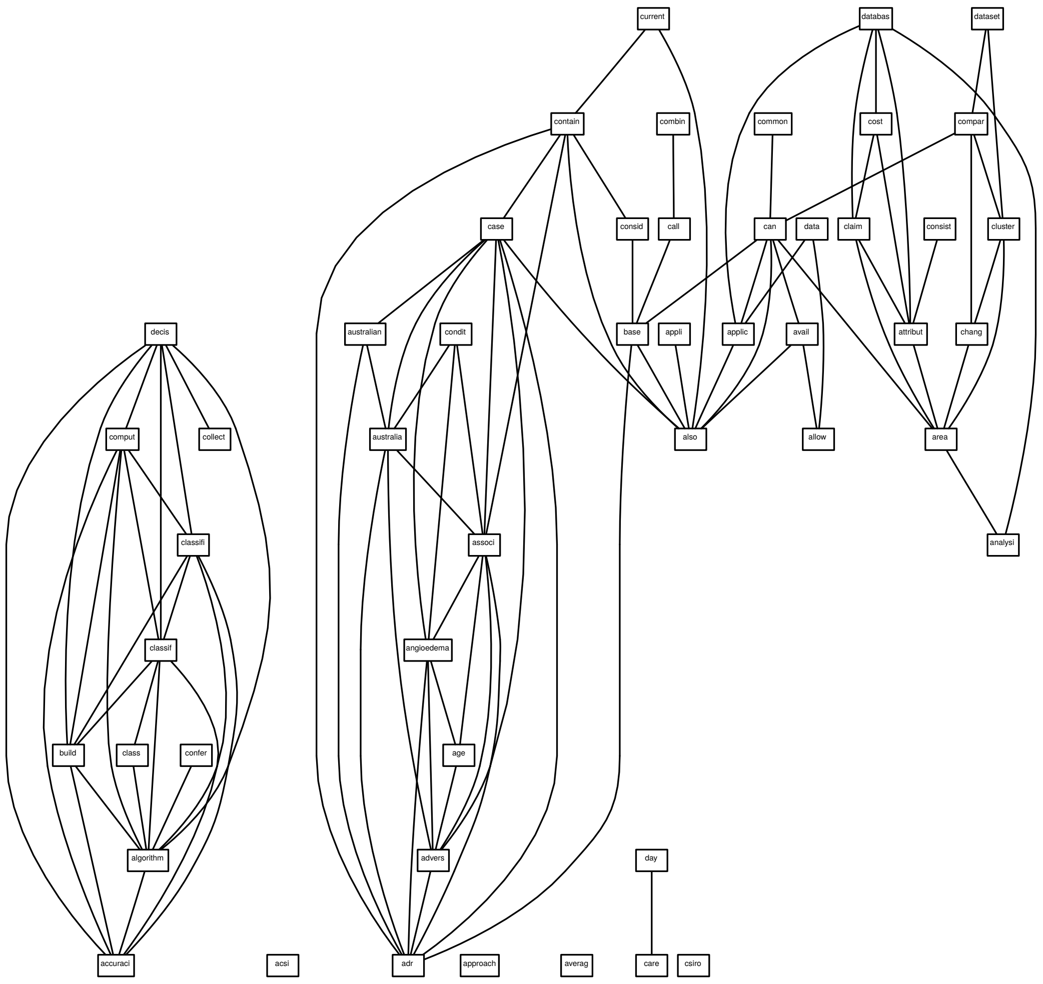
```
findFreqTerms(dtm, lowfreq=100)

##   [1] "accuraci"    "acsi"        "adr"         "advers"      "age"
##   [6] "algorithm"   "allow"       "also"        "analysi"     "angioedema"
##  [11] "appli"       "applic"      "approach"    "area"        "associ"
##  [16] "attribut"    "australia"   "australian"  "avail"       "averag"
##  [21] "base"        "build"       "call"        "can"         "care"
##  [26] "case"        "chang"       "claim"       "class"       "classif"
....
```

We can also find associations with a word, specifying a correlation limit.

```
findAssocs(dtm, "data", corlimit=0.6)

## $data
##         mine       induct     challeng         know       answer
##         0.90         0.72         0.70         0.65         0.64
##         need statistician      foundat      general        boost
##         0.63         0.63         0.62         0.62         0.61
##        major         mani         come
....
```

If two words always appear together then the correlation would be 1.0 and if they never appear together the correlation would be 0.0. Thus the correlation is a measure of how closely associated the words are in the corpus.

10 Correlations Plots

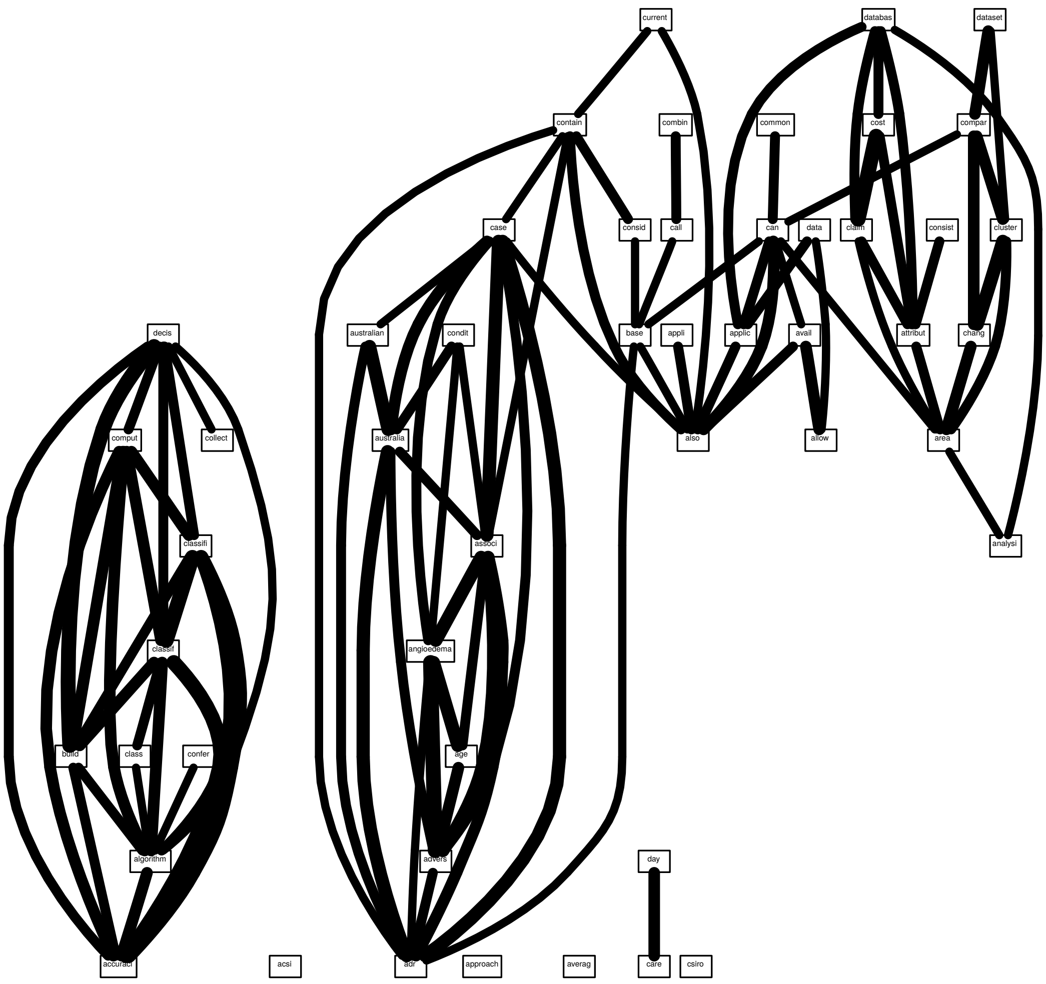


```
plot(dtm,
     terms=findFreqTerms(dtm, lowfreq=100)[1:50],
     corThreshold=0.5)
```

Rgraphviz (Hansen *et al.*, 2016) from the BioConductor repository for R (`bioconductor.org`) is used to plot the network graph that displays the correlation between chosen words in the corpus. Here we choose 50 of the more frequent words as the nodes and include links between words when they have at least a correlation of 0.5.

By default (without providing terms and a correlation threshold) the plot function chooses a random 20 terms with a threshold of 0.7.

11 Correlations Plot—Options



```
plot(dtm,
     terms=findFreqTerms(dtm, lowfreq=100)[1:50],
     corThreshold=0.5)
```

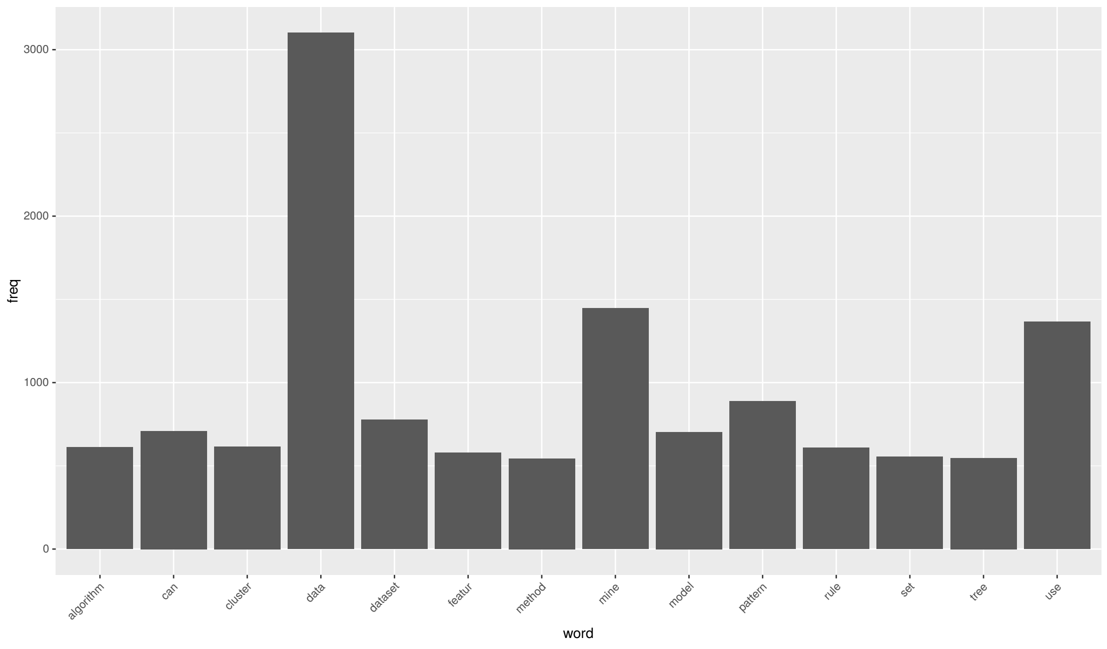
12 Plotting Word Frequencies

We can generate the frequency count of all words in a corpus:

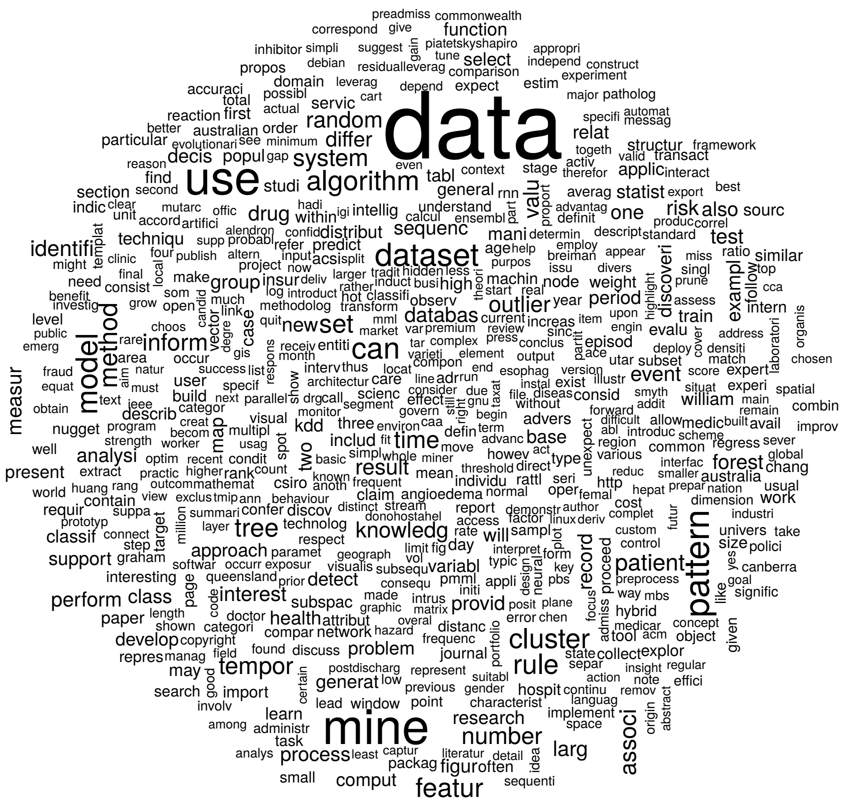
```
freq <- sort(colSums(as.matrix(dtm)), decreasing=TRUE)
head(freq, 14)

##      data      mine       use   pattern   dataset       can     model
##      3101      1446      1366       887       776       709       703
##   cluster algorithm      rule    featur       set      tree    method
##       616       611       609       578       555       547       544

wf   <- data.frame(word=names(freq), freq=freq)
head(wf)

##            word freq
## data       data 3101
## mine       mine 1446
## use         use 1366
## pattern pattern  887
## dataset dataset  776
....
```

We can then plot the frequency of those words that occur at least 500 times in the corpus:

```
library(ggplot2)
subset(wf, freq>500)                                            %>%
  ggplot(aes(word, freq))                                        +
  geom_bar(stat="identity")                                      +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```



13 Word Clouds



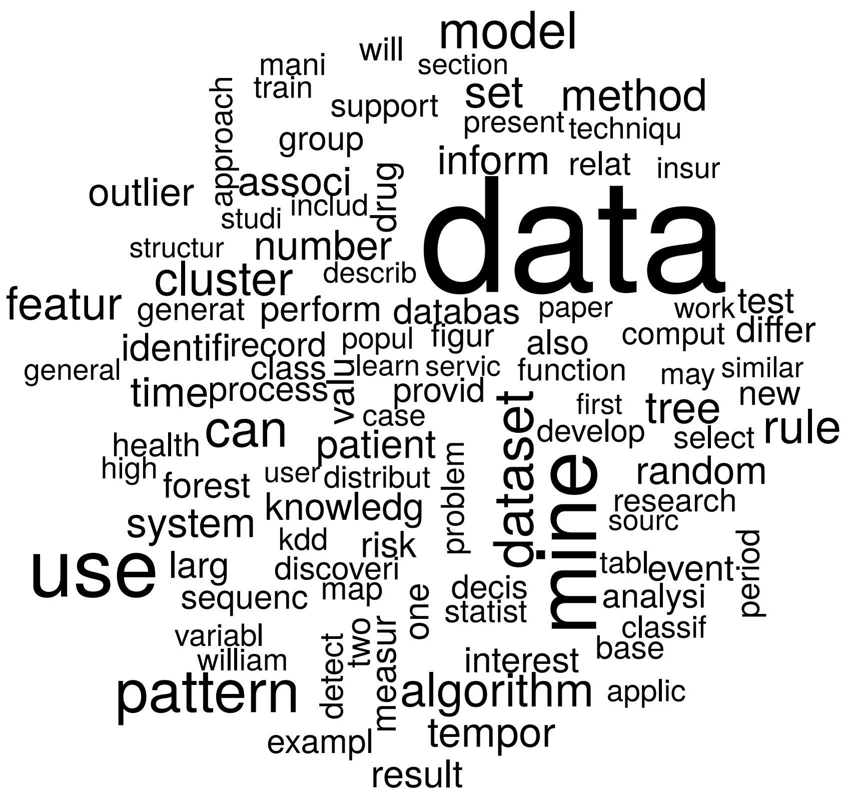
We can generate a word cloud as an effective alternative to providing a quick visual overview of the frequency of words in a corpus.

The wordcloud (?) package provides the required function.

```
library(wordcloud)
set.seed(123)
wordcloud(names(freq), freq, min.freq=40)
```

Notice the use of `set.seed()` only so that we can obtain the same layout each time—otherwise a random layout is chosen, which is not usually an issue.

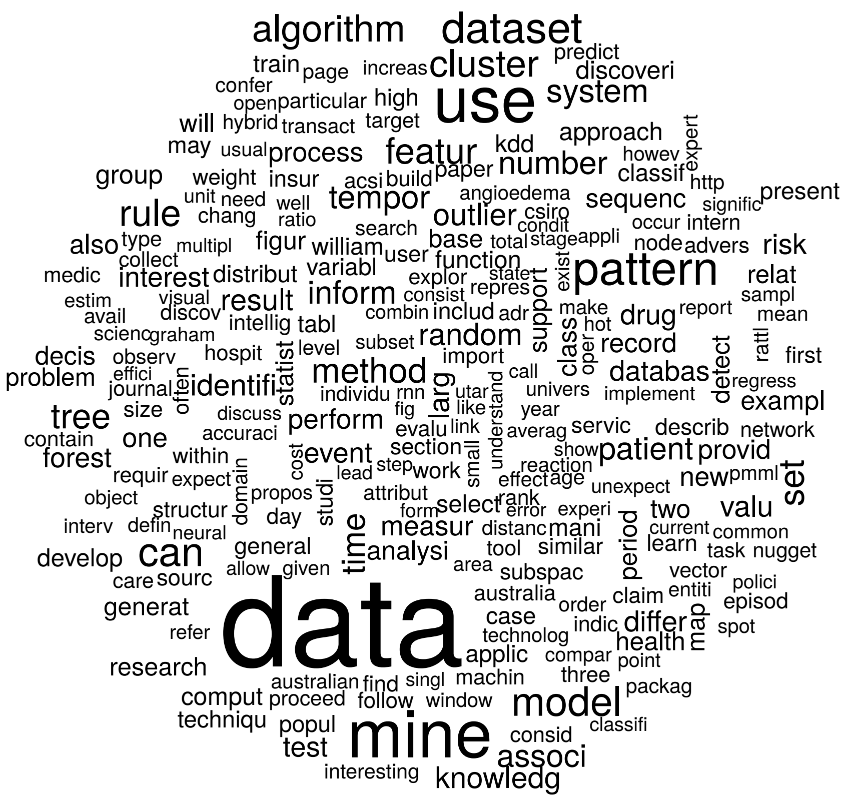
13.1 Reducing Clutter With Max Words



To increase or reduce the number of words displayed we can tune the value of `max.words=`. Here we have limited the display to the 100 most frequent words.

```
set.seed(142)
wordcloud(names(freq), freq, max.words=100)
```

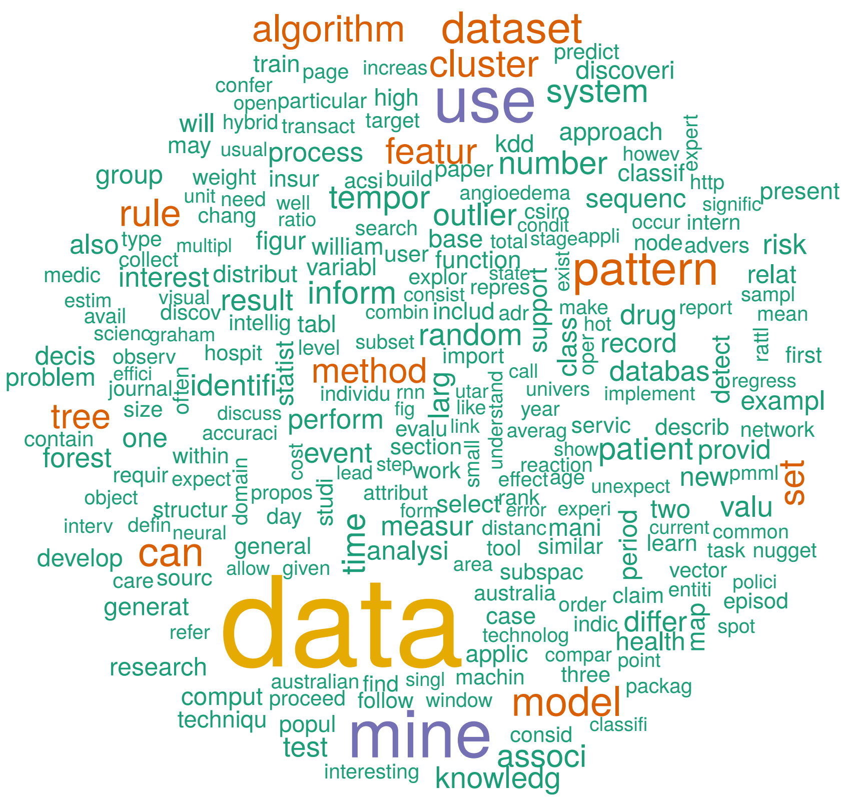
13.2 Reducing Clutter With Min Freq



A more common approach to increase or reduce the number of words displayed is by tuning the value of `min.freq=`. Here we have limited the display to those words that occur at least 100 times.

```
set.seed(142)
wordcloud(names(freq), freq, min.freq=100)
```

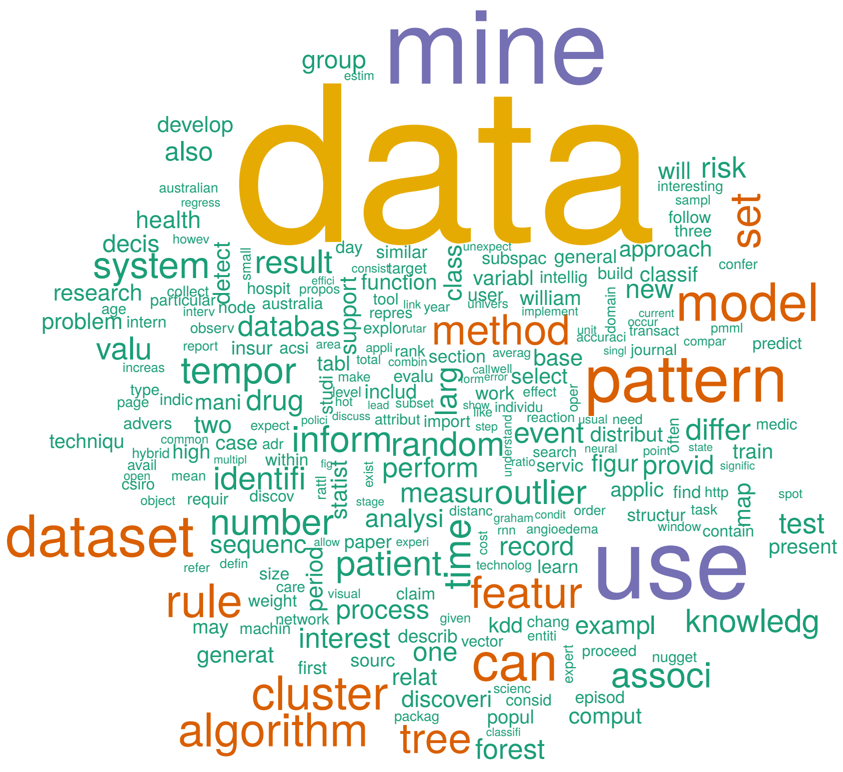
13.3 Adding Some Colour



We can also add some colour to the display. Here we make use of `brewer.pal()` from RColor-Brewer (Neuwirth, 2014) to generate a palette of colours to use.

```
set.seed(142)
wordcloud(names(freq), freq, min.freq=100, colors=brewer.pal(6, "Dark2"))
```

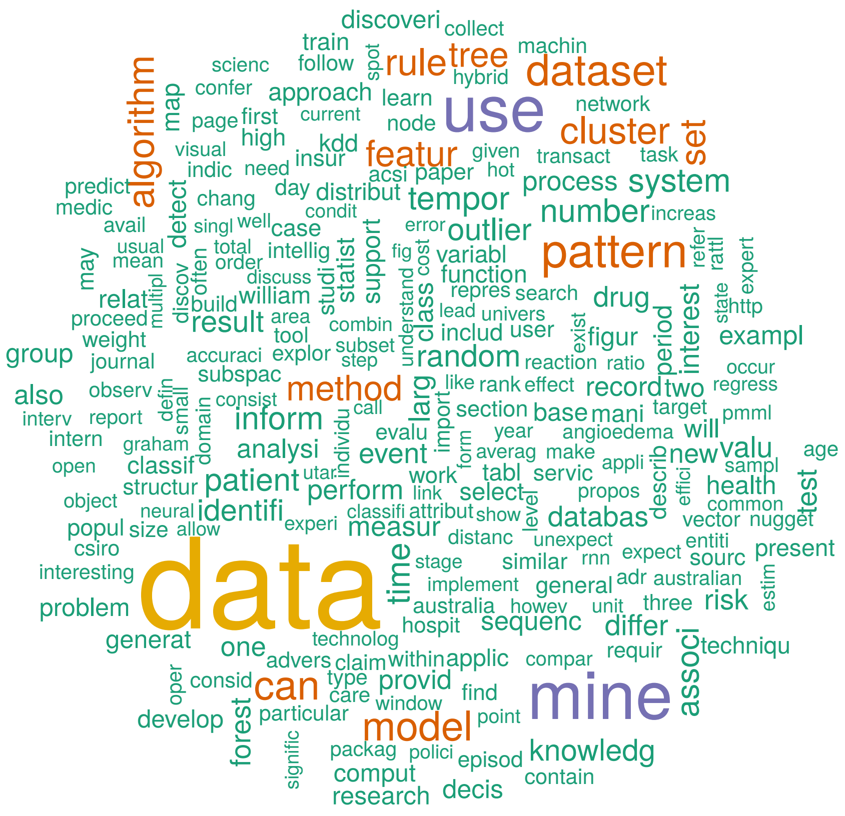
13.4 Varying the Scaling



We can change the range of font sizes used in the plot using the `scale=` option. By default the most frequent words have a scale of 4 and the least have a scale of 0.5. Here we illustrate the effect of increasing the scale range.

```
set.seed(142)
wordcloud(names(freq), freq, min.freq=100, scale=c(5, .1), colors=brewer.pal(6, "Dark2"))
```

13.5 Rotating Words



We can change the proportion of words that are rotated by 90 degrees from the default 10% to, say, 20% using `rot.per=0.2`.

```
set.seed(142)
dark2 <- brewer.pal(6, "Dark2")
wordcloud(names(freq), freq, min.freq=100, rot.per=0.2, colors=dark2)
```

14 Quantitative Analysis of Text

The qdap (Rinker, 2015) package provides an extensive suite of functions to support the quantitative analysis of text.

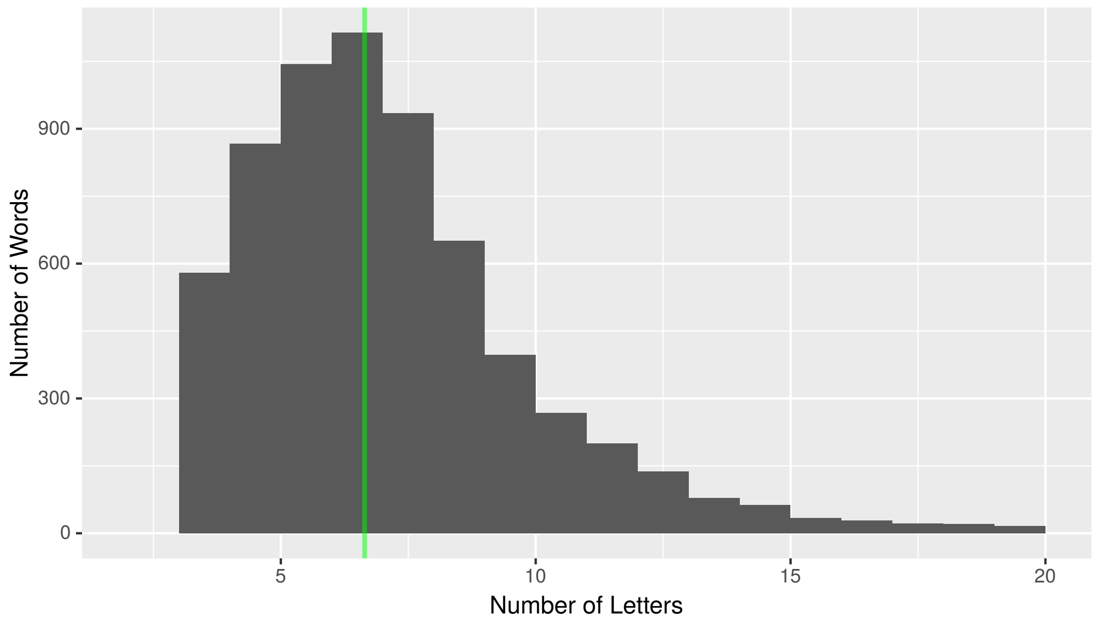
We can obtain simple summaries of a list of words, and to do so we will illustrate with the terms from our Term Document Matrix *tdm*. We first extract the shorter terms from each of our documents into one long word list. To do so we convert *tdm* into a matrix, extract the column names (the terms) and retain those shorter than 20 characters.

```
words <- dtm                                                    %>%
  as.matrix                                                     %>%
  colnames                                                      %>%
  (function(x) x[nchar(x) < 20])
```

We can then summarise the word list. Notice, in particular, the use of `dist_tab()` from qdap to generate frequencies and percentages.

```
length(words)

## [1] 6456

head(words, 15)

##  [1] "aaai"       "aab"        "aad"        "aadrbhtm"   "aadrbltn"
##  [6] "aadrhtmliv" "aai"        "aam"        "aba"        "abbrev"
## [11] "abbrevi"    "abc"        "abcd"       "abdul"      "abel"

summary(nchar(words))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   5.000   6.000   6.644   8.000  19.000

table(nchar(words))

##
##    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17
##  579  867 1044 1114  935  651  397  268  200  138   79   63   34   28   22
##   18   19
##   21   16

dist_tab(nchar(words))

##    interval freq cum.freq percent cum.percent
## 1         3  579      579    8.97        8.97
## 2         4  867     1446   13.43       22.40
## 3         5 1044     2490   16.17       38.57
## 4         6 1114     3604   17.26       55.82
## 5         7  935     4539   14.48       70.31
## 6         8  651     5190   10.08       80.39
## 7         9  397     5587    6.15       86.54
## 8        10  268     5855    4.15       90.69
## 9        11  200     6055    3.10       93.79
## 10       12  138     6193    2.14       95.93
....
```

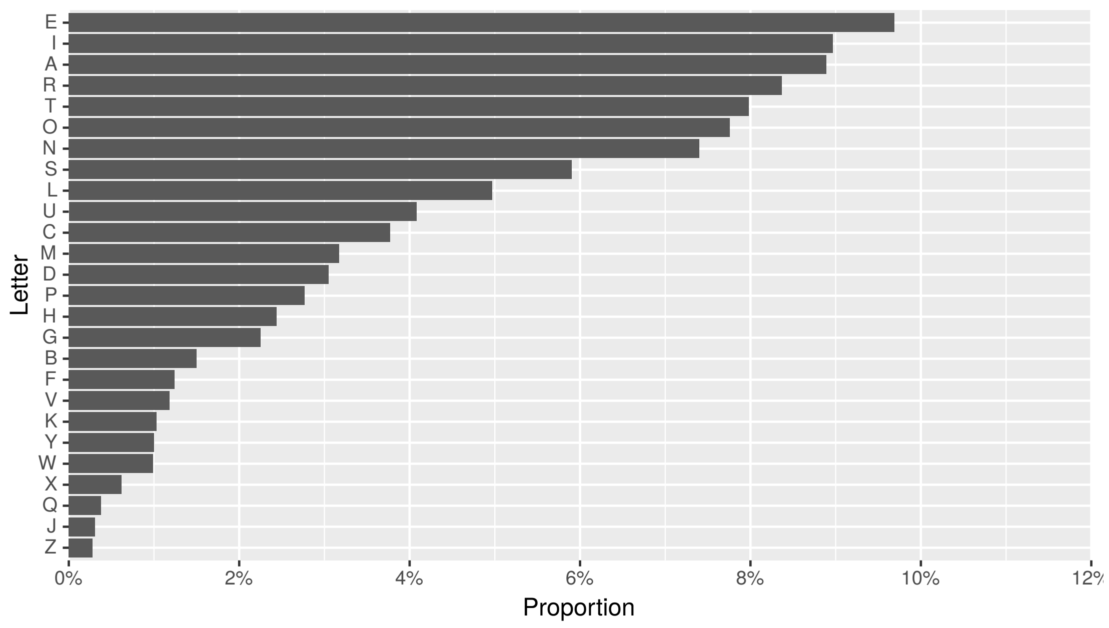
14.1 Word Length Counts



A simple plot is then effective in showing the distribution of the word lengths. Here we create a single column data frame that is passed on to `ggplot()` to generate a histogram, with a vertical line to show the mean length of words.

```
data.frame(nletters=nchar(words))                                  %>%
  ggplot(aes(x=nletters))                                           +
  geom_histogram(binwidth=1)                                        +
  geom_vline(xintercept=mean(nchar(words)),
             colour="green", size=1, alpha=.5)                      +
  labs(x="Number of Letters", y="Number of Words")
```

14.2 Letter Frequency

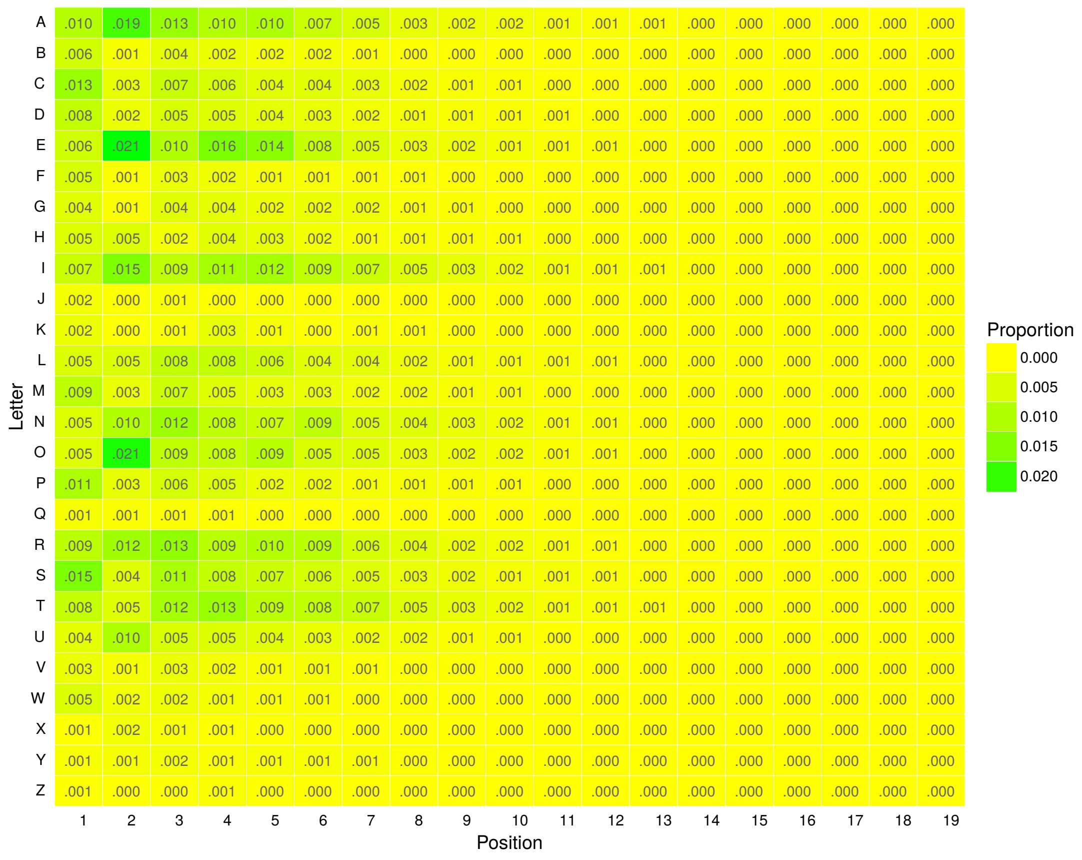


Next we want to review the frequency of letters across all of the words in the discourse. Some data preparation will transform the vector of words into a list of letters, which we then construct a frequency count for, and pass this on to be plotted.

We again use a pipeline to string together the operations on the data. Starting from the vector of words stored in *word* we split the words into characters using `str_split()` from stringr (Wickham, 2015), removing the first string (an empty string) from each of the results (using `sapply()`). Reducing the result into a simple vector, using `unlist()`, we then generate a data frame recording the letter frequencies, using `dist_tab()` from qdap. We can then plot the letter proportions.

```
library(dplyr)
library(stringr)

words                                                         %>%
  str_split("")                                                        %>%
  sapply(function(x) x[-1])                                            %>%
  unlist                                                               %>%
  dist_tab                                                             %>%
  mutate(Letter=factor(toupper(interval),
                       levels=toupper(interval[order(freq)])))         %>%
  ggplot(aes(Letter, weight=percent))                                   +
  geom_bar()                                                            +
  coord_flip()                                                          +
  labs(y="Proportion")                                                    +
  scale_y_continuous(breaks=seq(0, 12, 2),
                     label=function(x) paste0(x, "%"),
                     expand=c(0,0), limits=c(0,12))
```

14.3 Letter and Position Heatmap



The `qheat()` function from `qdap` provides an effective visualisation of tabular data. Here we transform the list of words into a position count of each letter, and constructing a table of the proportions that is passed on to `qheat()` to do the plotting.

```
words                                                                  %>%
  lapply(function(x) sapply(letters, gregexpr, x, fixed=TRUE))         %>%
  unlist                                                               %>%
  (function(x) x[x!=-1])                                               %>%
  (function(x) setNames(x, gsub("\\d", "", names(x))))                 %>%
  (function(x) apply(table(data.frame(letter=toupper(names(x)),
                                      position=unname(x))),
                     1, function(y) y/length(x)))                      %>%
  qheat(high="green", low="yellow", by.column=NULL,
        values=TRUE, digits=3, plot=FALSE)                              +
  labs(y="Letter", x="Position") +
  theme(axis.text.x=element_text(angle=0))                              +
  guides(fill=guide_legend(title="Proportion"))
```

14.4 Miscellaneous Functions

We can generate gender from a name list, using the genderdata (?) package

```
devtools::install_github("lmullen/gender-data-pkg")
```

```
name2sex(qcv(graham, frank, leslie, james, jacqui, jack, kerry, kerrie))

## The genderdata package needs to be installed.

## Error in install_genderdata_package():  Failed to install the genderdata package.
##  Please try installing the package for yourself using the following command:
##      install.packages("genderdata", repos = "http://packages.ropensci.org", type
= "source")
```

15 Word Distances

Continuous bag of words (CBOW). Word2Vec associates each word in a vocabulary with a unique vector of real numbers of length d. Words that have a similar syntactic context appear closer together within the vector space. The syntactic context is based on a set of words within a specific window size.

```
install.packages("tmcn.word2vec", repos="http://R-Forge.R-project.org")

## Installing package into '/home/gjw/R/x86_64-pc-linux-gnu-library/3.2'
## (as 'lib' is unspecified)

##
## The downloaded source packages are in
##  '/tmp/Rtmpt1u3GR/downloaded_packages'

library(tmcn.word2vec)
model <- word2vec(system.file("examples", "rfaq.txt", package = "tmcn.word2vec"))

## The model was generated in '/home/gjw/R/x86_64-pc-linux-gnu-library/3.2/tm...

distance(model$model_file, "the")

##        Word   CosDist
## 1         a 0.8694174
## 2        is 0.8063422
## 3       and 0.7908007
## 4        an 0.7738196
## 5    please 0.7595193
....
```

16 Review—Preparing the Corpus

Here in one sequence is collected the code to perform a text mining project. Notice that we would not necessarily do all of these steps so pick and choose as is appropriate to your situation.

```
# Required packages

library(tm)
library(wordcloud)

# Locate and load the Corpus.

cname <- file.path(".", "corpus", "txt")
docs <- Corpus(DirSource(cname))

docs
summary(docs)
inspect(docs[1])

# Transforms

toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
docs <- tm_map(docs, toSpace, "/|@|\\|")

docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeNumbers)
docs <- tm_map(docs, removePunctuation)
docs <- tm_map(docs, removeWords, stopwords("english"))
docs <- tm_map(docs, removeWords, c("own", "stop", "words"))
docs <- tm_map(docs, stripWhitespace)

toString <- content_transformer(function(x, from, to) gsub(from, to, x))
docs <- tm_map(docs, toString, "specific transform", "ST")
docs <- tm_map(docs, toString, "other specific transform", "OST")

docs <- tm_map(docs, stemDocument)
```

17 Review—Analysing the Corpus

```
# Document term matrix.

dtm <- DocumentTermMatrix(docs)
inspect(dtm[1:5, 1000:1005])

# Explore the corpus.

findFreqTerms(dtm, lowfreq=100)
findAssocs(dtm, "data", corlimit=0.6)

freq <- sort(colSums(as.matrix(dtm)), decreasing=TRUE)
wf   <- data.frame(word=names(freq), freq=freq)

library(ggplot2)

p <- ggplot(subset(wf, freq>500), aes(word, freq))
p <- p + geom_bar(stat="identity")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))

# Generate a word cloud

library(wordcloud)
wordcloud(names(freq), freq, min.freq=100, colors=brewer.pal(6, "Dark2"))
```

18 LDA

Topic Models such as Latent Dirichlet Allocation has been popular for text mining in last 15 years. Applied with varying degrees of success. Text is fed into LDA to extract the topics underlying the text document. Examples are the AP corpus and the Science Corpus 1880-2002 (Blei and Lafferty 2009). PERHAPS USEFUL IN BOOK?

When is LDA applicable - it will fail on some data and need to choose number of topics to find and how many documents are needed. HOw do we know the topics learned are correct topics.

Two fundemental papers - independelty discovered: Blei, Ng, Jordan - NIPS 2001 with 11k citations. Other paper is Pritchard, Stephens, and Donnelly in Genetics June 200 14K citations - models are exactly the same except for minor differences: except topics versus population structures.

No theoretic analysis as such. How to guarantee correct topics and how efficient is the learning procedure?

Observations:

LDA won't work on many short tweets or very few long documents.

We should not liberally over-fit the LDA with too many redundant topics...

Limiting factors:

We should use as many documents as we can and short documents less than 10 words won't work even if there are many of them. Need sufficiently long documents.

Small Dirichlet paramenter helps especially if we overfit. See Long Nguen's keynote at PAKDD 2015 in Vietnam.

number of documents the most important factor

document length plays a useful role too

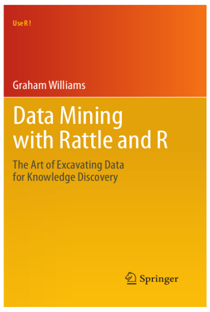
avoid overfitting as you get too many topics and don't really learn anything as the humn needs to cull the topics.

New work detects new topics as they emerge.

```
library(lda)

## Error in library(lda):  there is no package called 'lda'

# From demo(lda)
library("ggplot2")
library("reshape2")
data(cora.documents)

## Warning in data(cora.documents):  data set 'cora.documents' not found

data(cora.vocab)

## Warning in data(cora.vocab):  data set 'cora.vocab' not found
```

```
theme_set(theme_bw())
set.seed(8675309)
K <- 10 ## Num clusters
result <- lda.collapsed.gibbs.sampler(cora.documents,
                                      K,  ## Num clusters
                                      cora.vocab,
                                      25,  ## Num iterations
                                      0.1,
                                      0.1,
                                      compute.log.likelihood=TRUE)

## Error in eval(expr, envir, enclos):  could not find function "lda.collapsed.gibbs.sampler"

## Get the top words in the cluster
top.words <- top.topic.words(result$topics, 5, by.score=TRUE)

## Error in eval(expr, envir, enclos):  could not find function "top.topic.words"

## Number of documents to display
N <- 10

topic.proportions <- t(result$document_sums) / colSums(result$document_sums)

## Error in t(result$document_sums):  object 'result' not found

topic.proportions <-
   topic.proportions[sample(1:dim(topic.proportions)[1], N),]

## Error in eval(expr, envir, enclos):  object 'topic.proportions' not found

topic.proportions[is.na(topic.proportions)] <-  1 / K

## Error in topic.proportions[is.na(topic.proportions)] <- 1/K: object 'topic.proportions'
not found

colnames(topic.proportions) <- apply(top.words, 2, paste, collapse=" ")

## Error in apply(top.words, 2, paste, collapse = " "):  object 'top.words' not found

topic.proportions.df <- melt(cbind(data.frame(topic.proportions),
                                   document=factor(1:N)),
                             variable.name="topic",
                             id.vars = "document")

## Error in data.frame(topic.proportions):  object 'topic.proportions' not found

ggplot(topic.proportions.df, aes(x=topic, y=value, fill=topic)) +
    geom_bar(stat="identity") +
    theme(axis.text.x = element_text(angle=45, hjust=1, size=7),
          legend.position="none") +
    coord_flip() +
    facet_wrap(~ document, ncol=5)

## Error in ggplot(topic.proportions.df, aes(x = topic, y = value, fill = topic)):
object 'topic.proportions.df' not found
```

19 Further Reading and Acknowledgements


Use R!
Graham Williams
Data Mining with Rattle and R
The Art of Excavating Data for Knowledge Discovery
Springer

The Rattle Book, published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from http://datamining.togaware.com, including the Datamining Desktop Survival Guide.

This chapter is one of many chapters available from http://HandsOnDataScience.com. In particular follow the links on the website with a * which indicates the generally more developed chapters.

Other resources include:

- The Journal of Statistical Software article, *Text Mining Infrastructure in R* is a good start http://www.jstatsoft.org/v25/i05/paper
- Bilisoly (2008) presents methods and algorithms for text mining using Perl.

Thanks also to Tony Nolan for suggestions of some of the examples used in this chapter.

Some of the qdap examples were motivated by http://trinkerrstuff.wordpress.com/2014/10/31/exploration-of-letter-make-up-of-english-words/.

20 References

Bilisoly R (2008). *Practical Text Mining with Perl*. Wiley Series on Methods and Applications in Data Mining. Wiley. ISBN 9780470382851. URL http://books.google.com.au/books?id=YkMFVbsrdzkC.

Feinerer I, Hornik K (2015). *tm: Text Mining Package*. R package version 0.6-2, URL https://CRAN.R-project.org/package=tm.

Hansen KD, Gentry J, Long L, Gentleman R, Falcon S, Hahne F, Sarkar D (2016). *Rgraphviz: Provides plotting capabilities for R graph objects*. R package version 2.12.0.

Neuwirth E (2014). *RColorBrewer: ColorBrewer Palettes*. R package version 1.1-2, URL https://CRAN.R-project.org/package=RColorBrewer.

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

Rinker T (2015). *qdap: Bridging the Gap Between Qualitative Data and Quantitative Analysis*. R package version 2.2.4, URL https://CRAN.R-project.org/package=qdap.

Wickham H (2015). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.0.0, URL https://CRAN.R-project.org/package=stringr.

Williams GJ (2009). “Rattle: A Data Mining GUI for R.” *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York.

This document, sourced from TextMiningO.Rnw bitbucket revision 76, was processed by KnitR version 1.12 of 2016-01-06 and took 41.3 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.3 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 8 cores and 12.3GB of RAM. It completed the processing 2016-01-10 09:58:57.