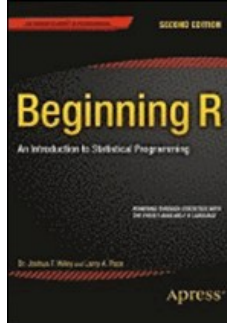


Chapters *To Go*



Beginning R: An Introduction to Statistical Programming, Second Edition

by Joshua F. Wiley and Larry A. Pace
Apress. (c) 2015. Copying Prohibited.

Reprinted for Sudheer K. Vetcha Vetcha, IBM

suvetcha@in.ibm.com

Reprinted with permission as a subscription benefit of **Skillport**,
<http://skillport.books24x7.com/>

All rights reserved. Reproduction and/or distribution in whole or in part in electronic, paper or other forms without written permission is prohibited.



Chapter 19: Text Mining

Overview

Our final topic is *text mining* or text data mining. It is really only something that can be done with access to comparatively decent computing power (at least historically speaking). The concept is simple enough. Read text data into R that can then be quantitatively analyzed. The benefits are easier to imagine than the mechanics. Put simply, imagine if one could determine the most common words in a chapter, or a textbook. What if the common words in one text could be compared to the common words in other texts? What might those comparisons teach us? Perhaps different authors have a set of go-to words they more frequently use. Perhaps there is a way to discover who wrote a historical text (or at least provide some likely suspects). Perhaps a model may be trained to sort "good" essays from "bad" essays (or to sort spam and ham in e-mail files). Full disclaimer: this is a beginning R text. There are many more things that would be brilliant to do to text data than what we will do in this chapter.

So what will we do in this chapter? For starters, we will discuss how to import text into R and get it ready for analysis. Text can come in many formats these days, and depending on the size and scope of your project, some ways may be more or less feasible. We will discuss some implementations that (currently) allow for importing a collection of common file(s). We'll also discuss pulling in some regularly updating files from the Internet.

Along the way, we will do some exploratory data analysis (EDA) with word clouds. Not only will this EDA let us confirm that we have successfully imported a document, but it will let us get to know what is in the text. An imported text file (e.g., `.txt`, `.pdf`, or `.docx`) is called a *corpus*. We'll discuss various transformations that are possible on the corpus, and we'll visualize just what those changes do.

From there, we will start to perform more in-depth inspections and analytics of a corpus: counting word frequency, finding correlations, visualizing correlations, topic models, and even some clever uses of our familiar `glm()` function to do some sorting.

Importing information from random file types can be difficult. UTF-8 `.txt` files are quite universal and thus are often comparatively easy. These days, most of the Microsoft Word files are `XML` files, and R tends to play well with `XML` files once you remove the wrappers that make them `.docx` files. Older Word formats (such as `.doc` pre-2006 or so) require more esoteric methods to use primarily just R. If you routinely deal in many different file types, we recommend learning a language other than R that is designed to process many formats into a consistent one to prepare a corpus of documents for analysis.

As you may well already know, but if not you will certainly discover, dealing with data usually requires data munging. For simplicity's sake, we will start off with basic `.txt` files, and build on additional formats from there as space allows. First, though, we need to install the right software.

19.1 Installing Needed Packages and Software

Several pieces of software are needed to run many of these examples. It may seem tedious to get them all working, and for that we do apologize. Text mining is fairly new, and the way we understand what may be done is changing very quickly. As with any effort that requires such an array of software to work, error messages or odd results may be more the norm than the exception. The web links we provide may prove to be inaccurate (although we have done our best to select more stable links). Our advice is to approach getting text mining functional as a series of steps in an adventure, after generous quantities of sleep, patience, and caffeine. As always with R, Google will prove a loyal friend. The R community is a clever bunch (although message board civility on occasion merits a "needs improvement" rating).

19.1.1 Java

It is important that you install Java on your computer. It is also important that you have installed the correct version(s) of R, RStudio, and Java. In this chapter we will provide some guidance for Windows users, as they make up a sizable cohort of users. For Windows 7, click the start button, right-click My Computer, and click properties. Next to system type it should say 64 bit, although it may say 32 bit. More detailed instructions may be found at <http://windows.microsoft.com/en-us/windows7/find-out-32-or-64-bit> for XP, Vista, or 7. Visit <https://support.microsoft.com/en-us/kb/827218> for Windows 8 or 10 or use the search option to find "Settings" and then select system. From there, select "about" and see next to system type that it says 64 bit, although it may say 32 bit. It is possible to try your luck with `Sys.info()` in the R command line. Ideally, under the machine output, you'd want to see something with a "64." This command returns information about the current system.

Once you've (ideally) confirmed you're on a 64-bit system, you may use the `R.Version()` command to see what version and type of R you installed. The information that follows is about the platform on which R was *built*, not the one on which it is running. There is quite a bit of output, but following are just a few key pieces to see the version of R you have and also whether it is 32 or 64 bit.

```
> R.Version()

$arch
[1] "x86_64"

$language
[1] "R"

$version.string
```

```
[1] "R version 3.2.1 (2015-06-18)"
```

```
$nickname
```

```
[1] "World-Famous Astronaut"
```

Provided your 64 bits (or 32 bits as the case may be) are all lined up, then all that remains is to make sure that you have the correct version of Java installed. Visit www.java.com/en/download/manual.jsp and select the 64-bit (or 32-bit) version to install. A restart or two of your system would be wise, and from there it should work seamlessly. Those are famous last words, and if they prove untrue, a quick Google of the error message(s) should prove helpful.

19.1.2 PDF Software

To allow for PDF files to be used, we need to have the `Xpdf` suite of tools available. It is perhaps simplest to download the zip file from www.foolabs.com/xpdf/download.html. To make the `Xpdf` suite work properly, we need to add it to the windows `PATH` variable so that it can be found from the command line and R. From the foregoing web site, download the `.zip` file. Extract the `zip` files to a convenient directory (e.g., `C:\usr\xpdfbin-win-3.04`). Right-click the Windows start menu, select "system," select "Advanced system settings," and in the system properties window that opens, select the "Advanced" tab. Then select the "Environment Variables," in the lower "System variables" box, scroll down until you see the "Path" line, select it to highlight, and click "Edit." Click the "Variable value:" box, being sure to only unhighlight the text. Go to the end (you can hit "End" on your keyboard to do this), and type `;%C:\usr\xpdfbin-win-3.04\bin64` and then select "OK." If you unzipped the files to another location, change the path appropriately. Also if you are using a 32-bit system and R, you should choose the 32-bit `Xpdf` suite by using the other directory. This will make it easier for R to find the correct file. Please be careful when doing this—it is not beyond the realm of possibility to cause heartache and grief.

19.1.3 R Packages

We also need a few R packages.

```
> install.packages("tm")
> install.packages("wordcloud")
> install.packages("tm.plugin.webmining")
> install.packages("topicmodels")
> install.packages("SnowballC")
> library(tm)
Loading required package: NLP
> library(wordcloud)
Loading required package: RColorBrewer
> library(tm.plugin.webmining)
```

```
Attaching package: 'tm.plugin.webmining'
```

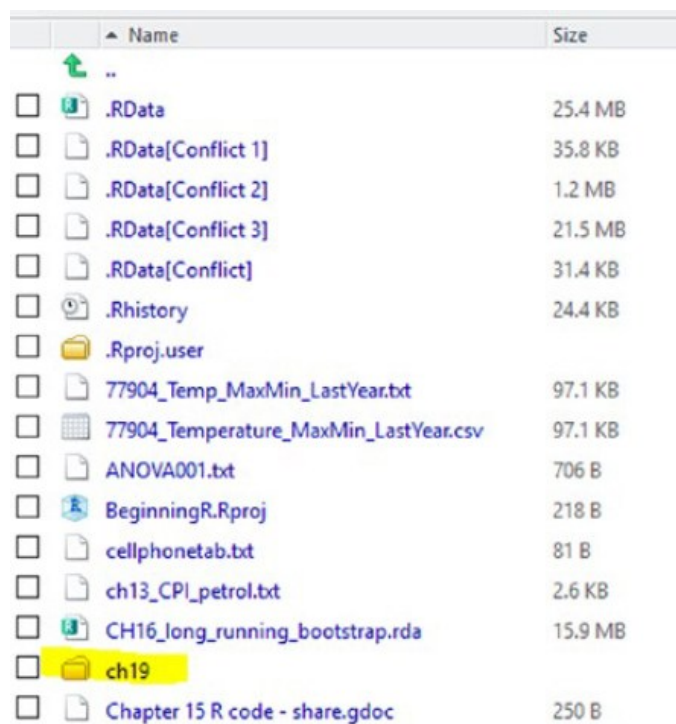
```
The following object is masked from 'package:base':
```

```
  parse
```

```
> library(topicmodels)
> library(SnowballC)
```

19.1.4 Some Needed Files

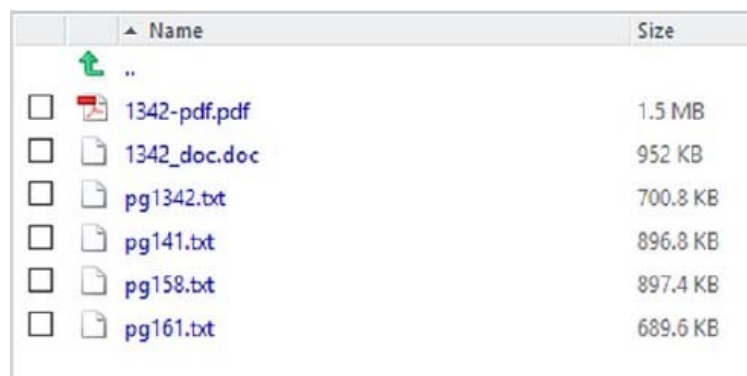
Finally, we turn our attention to having some local files for R to read into a *corpus*. First of all, in your working directory (really only for convenience and you may check with `getwd()`), please create a folder named `ch19` as shown in [Figure 19-1](#).



Name	Size
..	
.RData	25.4 MB
.RData[Conflict 1]	35.8 KB
.RData[Conflict 2]	1.2 MB
.RData[Conflict 3]	21.5 MB
.RData[Conflict]	31.4 KB
.Rhistory	24.4 KB
.Rproj.user	
77904_Temp_MaxMin_LastYear.txt	97.1 KB
77904_Temperature_MaxMin_LastYear.csv	97.1 KB
ANOVA001.txt	706 B
BeginningR.Rproj	218 B
cellphonetab.txt	81 B
ch13_CPI_petrol.txt	2.6 KB
CH16_long_running_bootstrap.rda	15.9 MB
ch19	
Chapter 15 R code - share.gdoc	250 B

Figure 19-1: Screenshot of folder in working directory

From there, please visit www.gutenberg.org/ebooks/author/68 and download the text files for *Pride and Prejudice*, *Emma*, *Sense and Sensibility*, and *Mansfield Park*. Also download the PDF for *Pride and Prejudice*.



Name	Size
..	
1342-pdf.pdf	1.5 MB
1342_doc.doc	952 KB
pg1342.txt	700.8 KB
pg141.txt	896.8 KB
pg158.txt	897.4 KB
pg161.txt	689.6 KB

Figure 19-2: Screenshot of ch19 folder contents

You are now ready to learn to do some text mining!

19.2 Text Mining

UTF-8 text files are the most common file type and the most readily managed in R. The first step is to get the textual information into R and in a format that our `tm` package can manipulate. There are several new function calls and arguments to discuss. We want to create a `Corpus()` and the self-titled function does just that. This command first takes an object (for us it will usually be a file), can take a type of "reader" if needed (e.g., to read PDF files), and can be given a language argument (the default is "en" for English). Our files are in a folder in our working directory called `ch19`. We use `DirSource(directory = ".", pattern = NULL, recursive = FALSE, ignore.case = FALSE, mode = "text")` to get files into R. A note on this function call is that the `directory` (which defaults to our working directory) can be given paths to folders outside the `getwd()`. The `pattern` and `ignore.case` variables may be used to set filename patterns so that only the files you wish are read into R. The `recursive` argument could be used to go deeper into the directory that you named. We show our first example that selects the *Pride and Prejudice* text file `pg1342.txt` as the sole source.

```
> austen <- Corpus (DirSource("ch19/", pattern="pg1342"))
> inspect(austen)
<<VCorpus>>
Metadata: corpus specific: 0, document level (indexed): 0
Content: documents: 1
```

```
[[1]]
```

```
<<PlainTextDocument>>
Metadata: 7
Content: chars: 690723

> summary(austen)
      Length Class      Mode 
pg1342.txt 2      PlainTextDocument list
```

As you can see, we have some information that indicates a successful read-in of our file. Another success indicator is that the global environment is now updated. If you're using RStudio, it would look like [Figure 19-3](#).

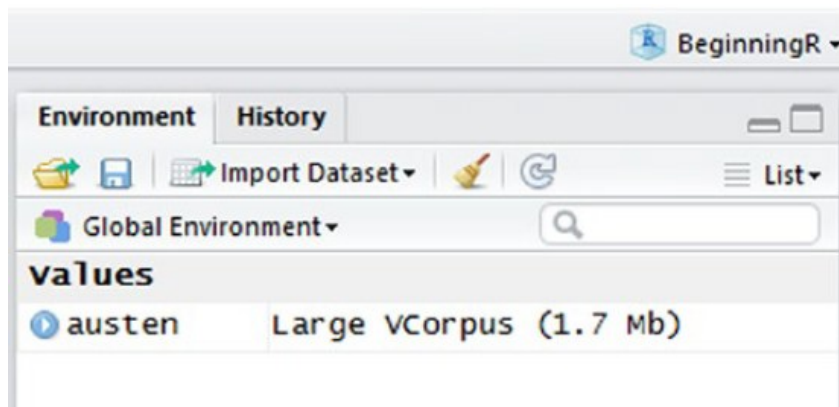


Figure 19-3: Screenshot of Pride and Prejudice VCorpus

19.2.1 Word Clouds and Transformations

Let's attempt to obtain a visual idea of what this corpus actually holds. In its raw form, the text file (and we recommend at least a quick glance through the raw text file to see what it contains) has many words. In real life, there would be some advantage to removing the "header" information and table of contents style information. We have not done that here. To get a sense of what is in our corpus, we turn to a *word cloud*. The `wordcloud` package has a function call named `wordcloud(words, freq, scale=c(4,.5), min.freq=3, max.words=Inf, random.order=TRUE, random.color=FALSE, rot.per=.1, colors="black", ordered.colors=FALSE, use.r.layout=FALSE, fixed.asp=TRUE, ...)` which takes several inputs. The first term, `words`, takes in a corpus. The `scale` argument assigns the range of word sizes (more common words are larger). The variable `max.words` sets the maximum number of words that show up in the cloud. Remember, there are many words in this corpus. Depending on screen real estate, you may well find efforts to show more words throw an error, thus our compromise of the top 100. The `0.35` of `rot.per` limits the vertical words to 35% of the total. Looking at the code that follows (which creates the output seen in [Figure 19-4](#)), we can see several of these arguments in action. Note, if you are getting warnings and a square output, then adjusting your viewing area to be larger may help. In R, this can be done simply by resizing the graphics window. In RStudio, this can be done by going full screen and using your mouse to increase the viewing area for the plots tab.

```
> wordcloud(austen, scale=c(5,0.5), max.words=100, random.order=FALSE,
+ rot.per=0.35, use.r.layout=FALSE, colors=brewer.pal(8, "Dark2"))
```

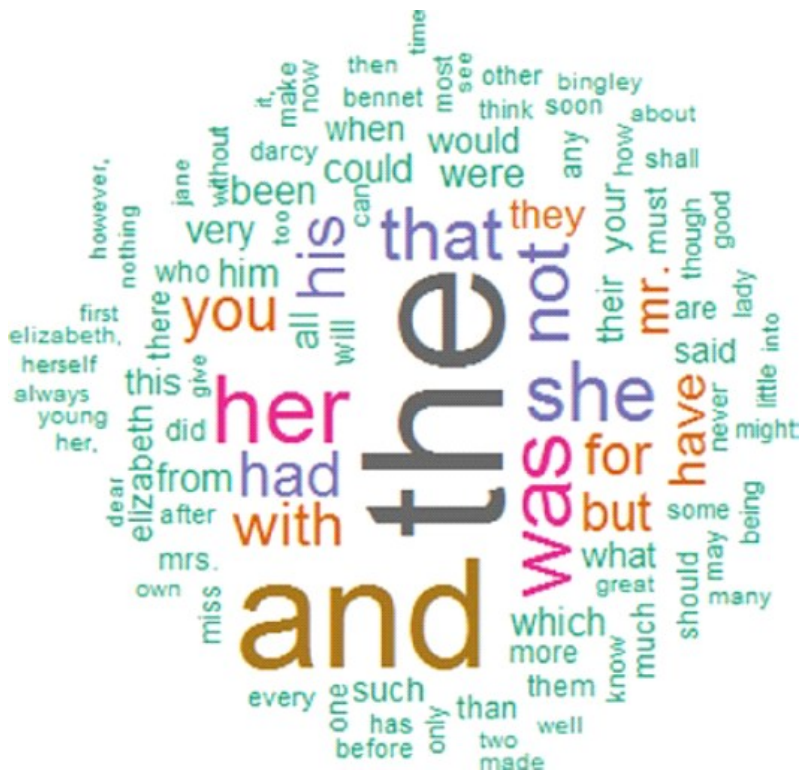
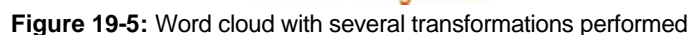


Figure 19-4: Word cloud of unprocessed corpus text

Notice that not only are there several words that are quite boring, but the word *elizabeth* shows up at least twice. Generally, we find it convenient to perform some transformations to the corpus to get the sorts of words we would expect from a text file. The function call `tm_map()` takes a corpus as its first input and then a function call to a transformation to apply to that corpus. We do not discuss the specifics of the various transformations simply because they are well named as shown in the following code. Note that `removeWords` can also take a custom word list. Of note, stop words are words to be removed prior to analysis. Canonically these are words such as "the" or "and," which do not add any real substance and simply serve to connect other words. It can be helpful when performing analytics on technical writing to remove words that may not be of interest (e.g., if analyzing R code, you may want to remove the assignment operator, `<-`). [Figure 19-5](#) shows our new word cloud after transformations and removing punctuation, whitespace, and stop words.

```
> austen <- tm_map(austen, content_transformer(tolower))
> austen <- tm_map(austen, removePunctuation)
> austen <- tm_map(austen, removeWords, stopwords("english"))
> austen <- tm_map(austen, content_transformer(stripWhitespace))

> wordcloud(austen, scale=c(5,0.5), max.words=100, random.order=FALSE,
+ rot.per=0.35, use.r.layout=FALSE, colors=brewer.pal(8, "Dark2"))
```

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

Page 7 / 15
Apress, Dr. Joshua F. Wiley and the estate of Larry A. Pace (c) 2015, Copying Prohibited

Word clouds are essentially EDA for text mining. Our readers will note that Ms Elizabeth seems to be the principal word and focus of *Pride and Prejudice*. Before we turn our attention to some other aspects of text mining, let us take our text transformation operations and package them into a single function. We have commented out a few additional possibilities, and leave it to the reader to use or not use pieces of this function as seems best for a particular document or set of documents.

```
> txttrans = function(text){
+   text = tm_map(text, content_transformer(tolower))
+   text = tm_map(text, removePunctuation)
+   ##text = tm_map(text, content_transformer(removeNumbers))
+   text = tm_map(text, removeWords, stopwords("english"))
+   text = tm_map(text, content_transformer(stripWhitespace))
+   ##text = tm_map(text, stemDocument)
+   text
+ }
```

While the word cloud creates a nice visual, it is not readily used to run additional code or analysis. Generally, we prefer to use `TermDocumentMatrix()` or `DocumentTermMatrix()` function calls. They create essentially the same structure (e.g., a matrix), and the difference is entirely whether the rows are terms or documents. Of course, for this specific example, we only have one document.

```
> austen = TermDocumentMatrix(austen)
> austen
<<TermDocumentMatrix (terms: 4365, documents: 1)>>
Non-/sparse entries: 4365/0
Sparsity           : 0%
Maximal term length: 26
Weighting          : term frequency (tf)
```

It can also be helpful not only to see a word cloud, where there is simply the top 100 words, but to see all the words above (or below for that matter) a certain frequency. The function call `findFreqTerms(x, lowfreq = 0, highfreq = Inf)` does just that on a Term Document or Document Term Matrix. Output is sorted alphabetically for all terms in *Pride and Prejudice* that show up at least 100 times. Notice these words have been stemmed, so words like *marry*, *marries*, and *married* would all be just *marri* as seen in row [49].

```
> findFreqTerms(austen, low = 100)

[1] "alway"      "answer"    "appear"    "attent"    "away"      "believ"
[7] "bennet"    "bingley"   "can"       "catherin"  "certain"   "collin"
[13] "come"      "darci"     "daughter"  "day"       "dear"      "elizabeth"
[19] "enough"    "even"      "ever"      "everi"     "expect"    "famili"
[25] "father"    "feel"      "felt"      "first"     "friend"    "gardin"
[31] "give"      "good"      "great"     "happi"     "hope"      "hous"
[37] "howev"     "jane"      "know"      "ladi"      "last"      "letter"
[43] "like"      "littl"     "long"      "look"      "love"      "lydia"
[49] "made"      "make"      "man"       "mani"      "manner"    "marri"
[55] "may"       "mean"      "might"     "miss"      "mother"    "mrs"
[61] "much"      "must"      "never"     "noth"      "now"       "one"
[67] "quit"      "receiv"    "repli"     "return"    "room"      "said"
[73] "saw"       "say"       "see"       "seem"      "shall"     "sister"
[79] "soon"      "speak"     "sure"      "take"      "talk"      "think"
[85] "though"    "thought"   "time"      "two"       "walk"      "way"
[91] "well"      "wickham"   "will"      "wish"      "without"   "work"
[97] "young"
```

So far we have accomplished a fair bit. Text has been input into a corpus, and we begin to have a sense of what makes that particular set of words perhaps unique. Still, that isn't the real power of what this was designed to do. We need more words. While we could simply get more text files (and indeed we have already), not all words come from text files. So first, we take a brief detour to the Portable Document Format (PDF).

19.2.2 PDF Text Input

We have our function, `txttrans()`, which can quickly process a corpus into something that may be readily analyzed. We have the correct software for PDFs, and our path is ready to help. With all that legwork done, it is quite simple to input a PDF file (or a few hundred). We do so with a PDF version of *Pride and Prejudice*, primarily to see that it is in fact equivalent to text. Note that if other PDF files were in this same directory, they would also be read in, so you may want to either specify the particular file or make sure the directory only has PDFs in it that you wish to read into R. To quickly demonstrate this, we show another word cloud in [Figure 19-7](#). Note that for this to work, it is important that the `pdftotext` program discussed earlier (in the section "PDF Software") can be executed directly from the command line. On Windows, this means ensuring it is correctly added to the `PATH` variable. On Linux or Mac, the program will need to be where the system usually searches for applications.

```
> austen2<-Corpus(DirSource("ch19/", pattern="pdf"), readerControl = list(reader=readPDF))
> austen2 <- txttrans(austen2)
> summary(austen2)
```



```

      Length Class           Mode
1342-pdf.pdf 2      PlainTextDocument list

> wordcloud(austen2, scale=c(5,0.5), max.words=100, random.order=FALSE,
+ rot.per=0.35, use.r.layout=FALSE, colors=brewer.pal(8, "Dark2"))

```



Figure 19-7: Word cloud of *Pride and Prejudice* from a PDF version

Once the information is in a corpus, it doesn't really matter from what type of file it came. Analytics may be readily performed on it. However, what is currently limiting us is that we only have a single document in these corpora. So, let's get 100 documents online in a relatively painless way.

19.2.3 Google News Input

Because we are getting these online, please be aware that your output may not match ours perfectly. The world of news involving Jane Austen is a happening place! The first thing to notice is that we have quite a few more documents than we had before. Still, all the same, we have what should now be a familiar process. [Figure 19-8](#) is not quite the same as our prior word clouds. Regardless, it seems clear our search was successful.

```

> austen4 = WebCorpus(GoogleNewsSource("Jane Austen"))
> austen4
<<WebCorpus>>
Metadata: corpus specific: 3, document level (indexed): 0
Content: documents: 100

> austen4 = txtttrans(austen4)
> wordcloud(austen4, scale=c(5,0.5), max.words=100, random.order=FALSE,
+ rot.per=0.35, use.r.layout=FALSE, colors=brewer.pal(8, "Dark2"))

```

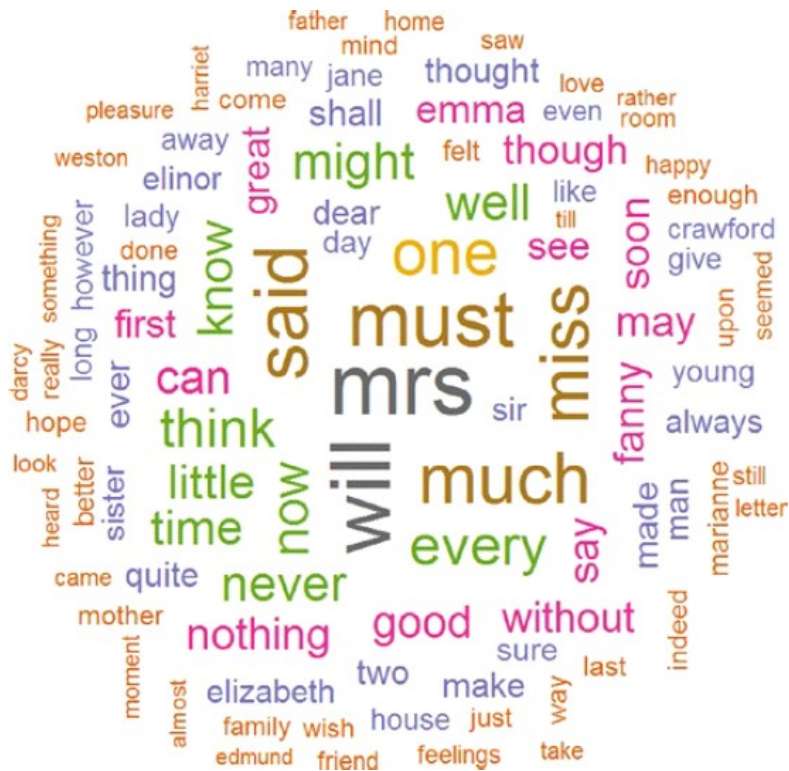



Figure 19-9: Word cloud of *Pride and Prejudice*, *Emma*, *Mansfield Park*, and *Sense and Sensibility*

Notice the difference in the word cloud. Across many novels, *mrs* becomes a much more common word. We begin to see that while in a single novel there may be an intense focus on Ms Elizabeth or Ms Woodhouse, there are certain recurring themes across the collection of novels. We could contrast this word cloud with one created by the works of Shakespeare and perhaps notice some defining differences. If you cared about spam vs. ham in electronic mail, you might do similar sorts of checks on e-mails to see if they looked more like the word cloud for spam or more like the word cloud for ham. You might even begin to perform some of the more familiar analytics we've already done to see if one could train some sort of model on ham or spam, and then assign a probability to a new message as to whether it should go into one folder or another.

That sort of activity quickly gets beyond the scope of this book, however. We shall delve into topic models, now that we have enough words and documents to make something sensible. First, while we have the word cloud, let us take a close look at the most frequent words. Since we have four novels instead of one, let's quadruple our low count to 400.

```
> austen_pesm_DTM = DocumentTermMatrix(austen_pesm)
```

```
> findFreqTerms(austen_pesm_DTM, low = 400)
```

[1]	"always"	"away"	"better"	"can"	"come"	"crawford"
[7]	"day"	"dear"	"done"	"elinor"	"elizabeth"	"emma"
[13]	"enough"	"even"	"ever"	"every"	"family"	"fanny"
[19]	"feelings"	"felt"	"first"	"friend"	"give"	"good"
[25]	"great"	"happy"	"hope"	"house"	"however"	"indeed"
[31]	"jane"	"just"	"know"	"lady"	"last"	"like"
[37]	"little"	"long"	"love"	"made"	"make"	"man"
[43]	"many"	"marianne"	"may"	"might"	"mind"	"miss"
[49]	"mother"	"mrs"	"much"	"must"	"never"	"nothing"
[55]	"now"	"one"	"quite"	"really"	"room"	"said"
[61]	"saw"	"say"	"see"	"seemed"	"shall"	"sir"
[67]	"sister"	"soon"	"still"	"sure"	"thing"	"think"
[73]	"though"	"thought"	"time"	"two"	"upon"	"way"
[79]	"well"	"will"	"wish"	"without"	"young"	

Since *elizabeth*, *emma*, and *miss* are frequent words, let's go ahead and see what sorts of associations can be made to those words that are both common and perhaps of interest. Before we do that, go back and rerun our text processing function, this time allowing stemming to occur.

```
> txttrans = function(text){
+   text = tm_map(text, content_transformer(tolower))
+   text = tm_map(text, removePunctuation)
+   text = tm_map(text, content_transformer(removeNumbers))
+   text = tm_map(text, removeWords, stopwords("english"))
+   text = tm_map(text, content_transformer(stripWhitespace))
+   text = tm_map(text, stemDocument)
+   text
}
```

```
+ }

> austen_pesm = txttrans(austen_pesm)
> austen_a = findAssocs(austen_pesm_DTM, terms = c("elizabeth", "emma", "miss"),
+ corlim = c(0.85, 0.90, 0.95))
```

Even with such a high set of correlations requested for each of these terms (and notice one can ask for different levels for different terms), the list is fairly long. For brevity's sake, we only show the correlations with *miss*. Even still, we only show some of the words in that one alone—there are many words. Perhaps more important, Jane Austen appears quite liberal in her use of that word. Notice these are sorted by correlation, and alphabetically inside a particular correlation.

```
austen_a$miss
```

agreed	appears	barely	bear	chances
1.00	1.00	1.00	1.00	1.00
communications	degree	equal	exactly	five
1.00	1.00	1.00	1.00	1.00
flutter	never	occupy	pay	peculiarly
1.00	1.00	1.00	1.00	1.00
pleasantly	putting	talked	understand	warmest
1.00	1.00	1.00	1.00	1.00
absolutely	anywhere	arrange	article	blessed
0.99	0.99	0.99	0.99	0.99
blue	came	can	certainly	cheerfully
0.99	0.99	0.99	0.99	0.99
clever	concurrence	decisive	encumbrance	excepting
0.99	0.99	0.99	0.99	0.99
fault	feature	feels	fourandtwenty	frightened
0.99	0.99	0.99	0.99	0.99
health	hear	hurry	included	irish
0.99	0.99	0.99	0.99	0.99
little	need	occupied	older	penance
0.99	0.99	0.99	0.99	0.99
prosperous	quite	sad	sanction	seized
0.99	0.99	0.99	0.99	0.99
shake	sort	south	spoken	substance
0.99	0.99	0.99	0.99	0.99
talents	visiting	will	worst	young
0.99	0.99	0.99	0.99	0.99
absenting	advise	agree	airy	amuses
0.98	0.98	0.98	0.98	0.98
apart	appealed	appropriated	approval	approved
0.98	0.98	0.98	0.98	0.98
arrangement	associates	augur	augusta	averted
0.98	0.98	0.98	0.98	0.98
basin	begin	biscuits	blended	blindness
0.98	0.98	0.98	0.98	0.98
breathe	commit	complains	conceived	conduce
0.98	0.98	0.98	0.98	0.98
convictions	council	dancer	dangers	dealings
0.98	0.98	0.98	0.98	0.98
decidedly	delighted	deplore	deserve	discipline
0.98	0.98	0.98	0.98	0.98
doubly	elegancies	english	enlivened	escorted
0.98	0.98	0.98	0.98	0.98
fasten	favouring	feasible	felicities	friendly
0.98	0.98	0.98	0.98	0.98
wainscot	watercolours	well	wholesome	writingdesk
0.98	0.98	0.98	0.98	0.98

Rather than look at such a chart, it might be better to connect each word to any related words above a certain threshold. Figure 19-10 shows the first attempt at such a graph. However, first note that to make this plot, you will need the Rgraphviz package available. This is not on CRAN, but instead is on another package repository, Bioconductor. We can install it relatively painlessly using the code that follows, and then make our graph.

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("Rgraphviz")

> plot(austen_pesm_DTM, terms = findFreqTerms(austen_pesm_DTM, lowfreq = 400),
+ corThreshold = 0.65)
```

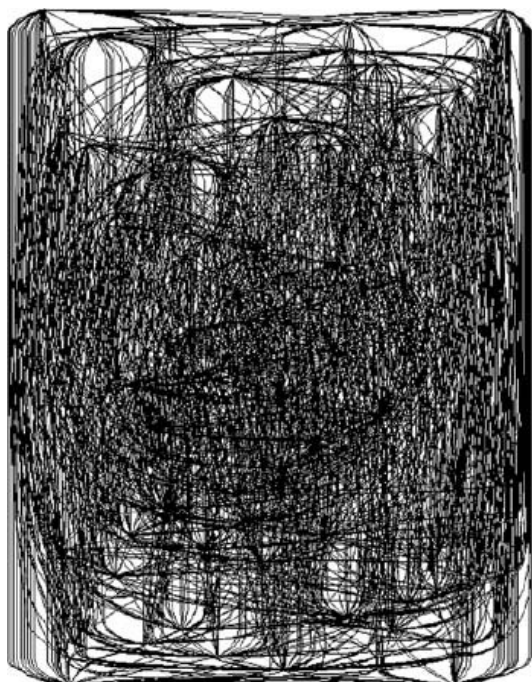



Figure 19-10: A first attempt at a plot of frequently associated terms

Clearly this is fairly useless. Well, that's not an entirely fair statement. What one sees is that there is a great deal of similarity between all these documents. Austen writes on fairly common theme(s), perhaps. We're going to have to be much more selective in our choice of both word frequency and correlation cut-off in order to have something readable. We take a second pass next and see that [Figure 19-11](#) is more legible. All the same, it is not quite there yet.

```
> plot(austen_pesm_DTM, terms = findFreqTerms(austen_pesm_DTM, lowfreq = 800),
+ corThreshold = 0.65)
```

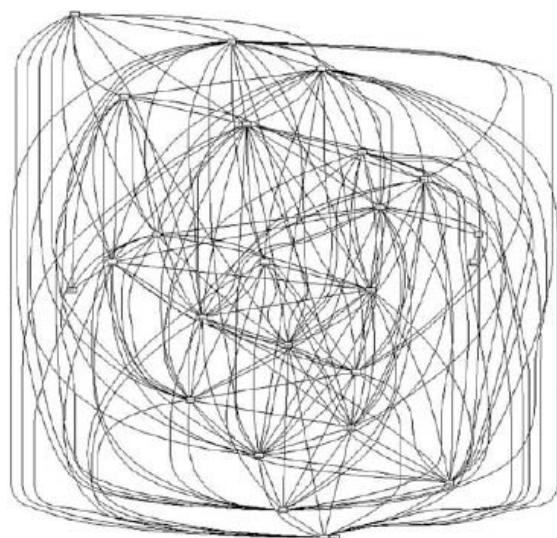


Figure 19-11: We've combed the hairball, and it gets tamer

One last pass, and we see that words actually live in the miniature boxes in [Figure 19-12](#).

```
> plot(austen_pesm_DTM, terms = findFreqTerms(austen_pesm_DTM, lowfreq = 850),
+ corThreshold = 0.95)
```

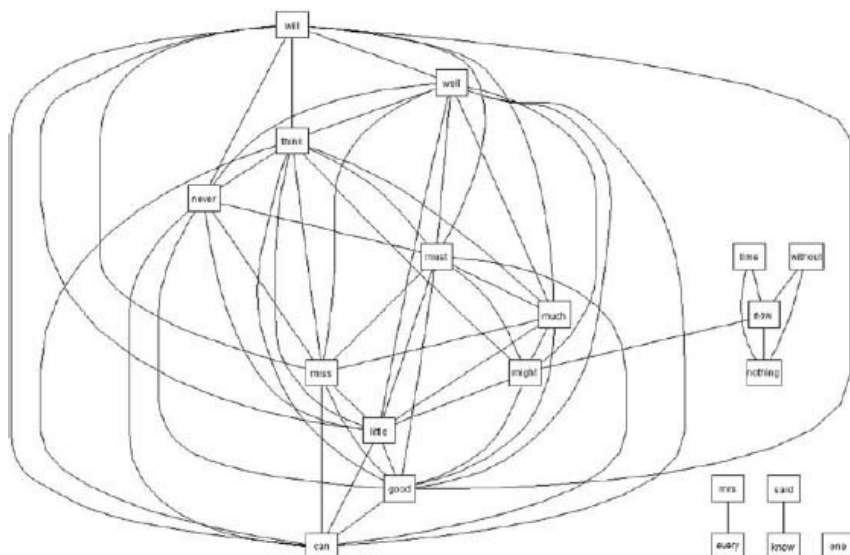



Figure 19-12: We can see words that are related in a nice graph

Our last bit of code is the actual topic model. Again, you may experiment with and on this to see what sorts of topics you may uncover. Also, again, there is a great deal more that you may do to determine how many topics might exist in a corpus under stable conditions. That is, it seems sensible that there are likely a certain number of distinct topics in any collection of texts. There ought to be a mathematical way to identify what the correct number of topics might be. One such way is using Latent Dirichlet Allocation (LDA, performed by the function with the same name).

Although we will not go into them in depth, there are many options to control these models and to assess quality. A common issue is that during optimization when R tries to figure out the "best" answer, the results you get may depend on where it started searching! One way around this is to have the model repeat itself many times from different random starts. In the example that follows, we use 100 different starting points, and R will pick the results that are most likely. You may get different results than we do based on a different random start set. If these were real analyses, we might keep adjusting the control parameters until we could get consistent results and be confident we had truly found the "best" solution.

To see all the options, you can (unintuitively) go to the documentation for the class of the object that the function expects to be used, which can be found by typing `?TopicModelcontrol-class` in the R console. This is where you can find out the names to use in the list to control how to estimate the LDA model, and how we knew to use the `alpha` and `nstart` in arguments. For now, we will simply note that even with 100 starting points, the model can take some time to finish running.

```
> austen_pesm_DTM
<<DocumentTermMatrix (documents: 4, terms: 16861)>>
Non-/sparse entries: 32951/34493
Sparsity           : 51%
Maximal term length: 32
Weighting          : term frequency (tf)

> rowTotals <- apply(austen_pesm_DTM, 1, sum)
> austen_pesm_DTM <- austen_pesm_DTM[rowTotals>0,]

> k <- 2
> austen_pesm_lda <- LDA(austen_pesm_DTM, control = list(alpha=0.2, nstart = 100), k)

> topics(austen_pesm_lda)
pg1342.txt pg141.txt pg158.txt pg161.txt
      1      2      2      1

> terms(austen_pesm_lda, 5)
      Topic 1 Topic 2
[1,] "mrs"    "must"
[2,] "said"    "mrs"
[3,] "will"    "will"
[4,] "much"    "miss"
[5,] "elinor"  "much"
```

In the foregoing code, `k` may be adjusted to various numbers, although from our plot, 2 or 3 looked about correct. With that, we are done with our brief introduction to text mining.

19.3 Final Thoughts

For text mining, our final thought is that this is a very exciting and really quite new field of research and study. Everything from movie reviews to research participant transcripts may be pulled into various corpora, and from there, you may perform many sorts of analytics. The generalized linear models we met in earlier chapters are helpful here, as are `k` nearest-neighbor cross-validation methods. One of the authors recalls reading an article, not too many years ago, that showed how various historical documents of unknown authorship could be fairly reliably matched (via comparison with known writings) to precise authors. More recently, anti-plagiarism methods can be similarly constructed from these sorts of inputs—or, as we mentioned, sorting spam and ham.

In the last few decades alone, the world of data analytics has undergone enormous changes. The way we think about data problems is changing. Nonparametric methods, bootstrapping, and text mining methods were not feasible in the past. Now, even for the large lakes of data we have access to, all of these methods are suddenly possible. It's a brave new world. Thank you for exploring it with us.

References

Annau, M. *tm.plugin.webmining: Retrieve Structured, Textual Data from Various Web Sources*. R package version 1.3, 2015. <http://CRAN.R-project.org/package=tm.plugin.webmining>.

Bouchet-Valat, M. *SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*. R package version 0.5.1, 2014. <http://CRAN.R-project.org/package=SnowballC>.

Feinerer, I., & Hornik, K. *tm: Text Mining Package*. R package version 0.6-2, 2015. <http://CRAN.R-project.org/package=tm>.

Fellows, I. *wordcloud: Word Clouds*. R package version 2.5, 2014. <http://CRAN.R-project.org/package=wordcloud>.

Gruen, B., & Hornik, K. "topicmodels: An R Package for Fitting Topic Models." *Journal of Statistical Software*, 40(13), 1-30 (2011).