Text Processing

September 26, 2018

This part of the workshop is intended to acquaint you with the task of processing plain text documents. As always, this can be accomplished in several ways. Please note that these exercises have been transferred from the context of another course, there might be redundant instructions and formulations reflecting this fact.

Tokenization and Word Frequencies

Let us start by looking at the first preprocessing steps. Start up R Studio and check your working directory.

```
getwd()
```

You might get some output like
`[1] "/Users/mikaelgunnarsson/Box Sync/Statistics"`
below the command prompt, depending on your individual settings. You can always use the `setwd("path")` to change the working directory. We need to keep track of this in order for the IDE to find files residing somewhere else on your computer.

Let's assume you have downloaded all the stuff from Github (https://github.com/Mikael61/DLIR2) and unzipped it. Unzip now the archive `collection.zip` into your working directory — be sure that it is really uncompressed by looking in the file explorer. It should create two subdirectories in your working directory — `texts` and `text`. The former one

contains a corpus of texts and the latter one only one of the texts in the corpus. Set your working directory to the root of the newly created directory `collection`.

Let us start by loading the single text into R STUDIO in a variable named `txt`

```
txt = readChar("text/pbshelley.txt",
   file.info("text/pbshelley.txt")$size)
```

provided it reside in the subdirectory `text` and is named `pbshelley.txt`. Have a look at the contents in the View panel

```
View(txt)
```

We want to preprocess the text in order to generate a set of features (or variables, if you prefer) with values. For instance, we want to count the frequency with which each "word" occur in the text and regard the frequency of each word as a feature. The most crude way of doing this is simply just to split the sequence of characters into substrings defined by some kind of delimiters, such as whitespace — this process is called *tokenization*. Thereafter we will count the occurrence of each unique token (substring). Use the bult-in function in R for creating substrings

```
t = strsplit(txt, " ", fixed = T)
```

that will create and store the substrings in `t`. Look at it by

```
View(t)
```

and you see a table where each row contains a token (substring) from the text — almost 50 000 tokens.[1] You may note that, for instance, the first token identified is `"ADONAIS by SHELLEY edited With"` which is obviously five words. This depends on that we split the text based on only

[1]Note that in this view many characters are not shown, such as line-breaks, tabs and other paralinguistic characters that are no different from the letters and numbers from the computer's point of view. Please also note that different installations of R might produced different views.

one specific character, the space character, and this "phrase" contains other whitespace characters, such as line breaks. You'll also see that paralinguistic characters such as commas, full stops, single quotes, underscore etc remain as part of the tokens. *Are there more observations contradicting the general idea of what a word is?* We could proceed from here and find functions for cleaning up the text, counting the frequencies and so on, in order to get the features we want. However, *R* has libraries that make these things really simple for us, i.e. for instance the TM package.

Start by loading the library for "text mining". If needed install the package. (Outcommented below, by #)

```
# install.packages("tm")

library(tm)
```

By the following command you will load the one text in the directory `text` for the TM library to process it, which contains the full text of Shelley's work *Adonais* transcribed within the Project Gutenberg project. I have (unrightfully) deleted some Gutenberg attribution data from the text file, so please do not recycle the text(s). This has been done only for you to not having to do this deletion by yourself. If left there it would produce noisy data for our purposes.

```
doc <- Corpus(DirSource("text"))
```

You may inspect the document by

```
doc[[1]]
```

and the contents of it by

```
doc[[1]]$content
```

or

```
View(as.matrix(doc[[1]]$content))
```

The `[[1]]` is needed to grab the first document in the corpus — regardless of the fact that it only contains one text. Since it is loaded into memory by the TM package it is no longer a single text, but a compound object embedding the text itself. The `doc` data variable (not a variable in the statistical sense, but in a computational sense) now holds this abstract representation of all your data in a particular way that makes it susceptible to various operations. Despite this, the text is for the computer not much more than a long string of characters (we see the words as we are used to discern them, but the computer does not).

Any textual data for analytic purpose, be it for categorization or anything else, needs preprocessing – tokenization is not enough – otherwise our data will contain a lot of noise and errors. Luckily there are tools for this. The TM package provides several solutions.

First of all, we want to get rid of all the paralinguistic characters, the punctuation. We can either overwrite the existing text or not.

```
doc1 = tm_map(doc, removePunctuation)
```

will create a new variable `doc1`, while

```
doc = tm_map(doc, removePunctuation)
```

will overwrite the old contents of `doc1` in allocated memory and replace it with the new contents.

Inspect the contents.

```
View(as.matrix(doc1[[1]]$content))
```

Most often, you need not bother about capitals so you can convert any token into lower case only, remove numbers and the most common prepositions, pronouns etc (stop words), and remove redundant white space. Run them one after each other and be sure to refer to the correct variable.

```
doc2 = tm_map(doc1, content_transformer(tolower))

doc3 = tm_map(doc2, removeNumbers)

doc4 = tm_map(doc3, removeWords, stopwords("english"))
```

```
doc5 = tm_map(doc4, stripWhitespace)
```

and for some (to me quite unknown) reason you do best to issue the following command before proceeding. [2]

```
doc5 = tm_map(doc5, PlainTextDocument)
```

Be observant on how we have several intermediary instances of the text here, and that the final result resides in `doc5`. Hitherto, we have not much more than a cleansed copy of our source text, but we will do a lot with it. First of all, we will create a document-term matrix that gives us the frequency count for all unique strings (what we may recognize as words).

```
dtm = DocumentTermMatrix(doc5)
```

and a transpose of the same data, if we need it for some purpose.

```
tdm = TermDocumentMatrix(doc5)
```

Look at them both in the View panel. We now have the frequency for each unique string in the text. We can sort them in increasing or decreasing order by clicking at the table header in the View panel. **However, depending on our workstation's memory and processor, it can be extremely slow, if the number of columns is high.**

```
View(as.matrix(dtm))
```

We can also generate a "wordcloud" based on the frequencies.

```
library(wordcloud)
```

```
wordcloud(as.character(doc5), min.freq=30,
   scale=c(3, .1),
   colors=brewer.pal(6, "Dark2"))
```

[2] To be honest, it has shown out to be quite arbitrary if it is needed or not, and have sometimes corrupted the data. If something appears to go wrong, return and omit this step.

and print a list of all strings occurring at least, e.g., 50 times

```
findFreqTerms(dtm, lowfreq=50)
```

Working with Several Texts

Hitherto we have only been investigating one text at a time. This might be an interesting exploratory task in itself. However, we are mostly interesting in several instances of text in order to discern differences or fact to group them in several ways, based on e.g. similarity in word use.

We start by repeating some of the same commands as before, despite that we now tell R STUDIO to look at the subdirectory `texts` that contain 40 different texts downloaded from Project Gutenberg (20 from the psychology collection of Project Gutenberg and 20 texts from the history collection).

```
corp <- Corpus(DirSource("texts"))
```

Then take each of the following commands in turn and feed to R STUDIO. We will now consistently use the name `corp` as a variable name, and do not bother about intermediary representations, being satisified with changing the contents of the variable `corp`. Alas, the contents of the variable will change and if something goes wrong you need to repeat the loading of the corpus, as above.

```
corp = tm_map(corp, content_transformer(tolower))

corp = tm_map(corp, removePunctuation)

corp = tm_map(corp, removeNumbers)

corp = tm_map(corp, stripWhitespace)

corp = tm_map(corp, removeWords, stopwords(kind = "en"))

dtm = DocumentTermMatrix(corp)
```

```
tdm = TermDocumentMatrix(corp)
```

You can now take a look somewhere in the matrix for all 40 texts. Let us look for strings in positions 100 – 105 (they get their positions according to an alphabetical sorting).

```
inspect(dtm[,100:105])
```

You may, of course, generate a wordcloud for this corpus if you want, just as we did for one text previously. However, it will take a lot of time and you need to be patient, considering the many strings in the corpus.

The number of "features" generated is much too high to inspect in its entirety. If we just issue the expression `dtm` we get

```
<<DocumentTermMatrix (documents: 40, terms: 69688)>>
Non-/sparse entries: 247763/2539757
Sparsity           : 91%
Maximal term length: 55
Weighting          : term frequency (tf)
```

where the level 91% of sparsity is a measure of the portion of tokens that are more or less particular for one text. Removing some of this sparsity might help in exploring the differences between texts.

```
dtms = removeSparseTerms(dtm, 0.1)
View(as.matrix(dtms))
```

Try different values for the reduction of sparsity by changing 0.1 to other values (below 1.0) and look at the resulting number of "terms" by means of the command `dtms` and, if few enough, `View(as.matrix(dtms))`. Large matrices might take minutes to process, depending on your computer's performance.

The resulting document term matrix is now ready to be processed in many ways. However, there is one factor to consider before we move on. The application of a stopword list was due to the fact that many words bear no or less meaning since they are "function words" that mostly have syntactic

meaning in building sentences and are very frequent in every text. It does not compensate for the fact that there might be other words very frequent and some words occurring very non-frequent but still in more than one text. The term frequency counts may then be coupled with an inverse document frequency in stead.[3]

```
dtm <- DocumentTermMatrix(corp, control = list(weighting = weightTfIdf))
dtms <- removeSparseTerms(dtm, 0.1)
```

To sum up: having decomposed texts into smaller units for analysis in these ways opens up for many different ways of proceeding with the analysis. If you do the same with data you have, you can try to think about what becomes possible and try to find ways of doing this in R. I am quite sure it is not possible in SPSS, but there may be other environments that do facilitate it, such as *Python*.

Let us now proceed to machine learning tasks. If necessary you'll have to return to the text processing phase if results are not that successful. You might for instance redefine the stop word list or consider other ways to process the texts (e.g. as *n*-grams).

General remark for doing your own project
Whatever your analytic purpose is you should see to it that you have your data stored as one or more text files, preferrably in utf8 character encoding. Save them in one directory in files with discriminative and simple names – avoid special, non-english characters and white space in file names. Awareness of all the potential problems related to character encoding is crucial whenever you are dealing with non-English text

[3]Note that the = is almost equivalent to <-. We may ignore the difference.