Corpus analysis: the document-term matrix

Wouter van Atteveldt

The most important object in frequency-based text analysis is the *document term matrix*. This matrix contains the documents in the rows and terms (words) in the columns, and each cell is the frequency of that term in that document.

In R, these matrices are provided by the `tm` (text mining) package. Although this package provides many functions for loading and manipulating these matrices, using them directly is relatively complicated.

Fortunately, the `RTextTools` package provides an easy function to create a document-term matrix from a data frame. To create a term document matrix from a simple data frame with a 'text' column, use the `create_matrix` function (with removeStopwords=F to make sure all words are kept):

```
library(RTextTools)
input = data.frame(text=c("Chickens are birds", "The bird eats"))
m = create_matrix(input$text, removeStopwords=F)
```

We can inspect the resulting matrix m using the regular R functions to get e.g. the type of object and the dimensionality:

```
class(m)
```

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

```
dim(m)
```

```
## [1] 2 6
```

```
m
```

```
## <<DocumentTermMatrix (documents: 2, terms: 6)>>
## Non-/sparse entries: 6/6
## Sparsity           : 50%
## Maximal term length: 8
## Weighting          : term frequency (tf)
```

So, `m` is a `DocumentTermMatrix`, which is derived from a `simple_triplet_matrix` as provided by the `slam` package. Internally, document-term matrices are stored as a *sparse matrix*: if we do use real data, we can easily have hundreds of thousands of rows and columns, while the vast majority of cells will be zero (most words don't occur in most documents). Storing this as a regular matrix would waste a lot of memory. In a sparse matrix, only the non-zero entries are stored, as 'simple triplets' of (document, term, frequency).

As seen in the output of `dim`, Our matrix has only 2 rows (documents) and 6 columns (unqiue words). Since this is a fairly small matrix, we can visualize it using `as.matrix`, which converts the 'sparse' matrix into a regular matrix:

```
as.matrix(m)
```

```
##     Terms
## Docs are bird birds chickens eats the
##    1   1    0     1        1    0   0
##    2   0    1     0        0    1   1
```

Stemming and stop word removal

So, we can see that each word is kept as is. We can reduce the size of the matrix by dropping stop words and stemming (changing a word like 'chickens' to its base form or stem 'chicken'): (see the create_matrix documentation for the full range of options)

```
m = create_matrix(input$text, removeStopwords=T, stemWords=T, language='english')
dim(m)
```

```
## [1] 2 3
```

```
as.matrix(m)
```

```
##     Terms
## Docs bird chicken eat
##    1    1       1   0
##    2    1       0   1
```

As you can see, the stop words (*the* and *are*) are removed, while the two verb forms of *to eat* are joined together.

In RTextTools, the language for stemming and stop words can be given as a parameter, and the default is English. Note that stemming works relatively well for English, but is less useful for more highly inflected languages such as Dutch or German. An easy way to see the effects of the preprocessing is by looking at the colSums of a matrix, which gives the total frequency of each term:

```
colSums(as.matrix(m))
```

```
##    bird chicken     eat
##       2       1       1
```

For more richly inflected languages like Dutch, the result is less promising:

```
text = c("De kip eet", "De kippen hebben gegeten")
m = create_matrix(text, removeStopwords=T, stemWords=T, language="dutch")
colSums(as.matrix(m))
```

```
##   eet geget   kip  kipp
##     1     1     1     1
```

As you can see, *de* and *hebben* are correctly recognized as stop words, but *gegeten* (eaten) and *kippen* (chickens) have a different stem than *eet* (eat) and *kip* (chicken). German gets similarly bad results.

Loading and analysing a larger dataset from AmCAT

If we want to move beyond stemming, one option is to use AmCAT to parse articles. Before we can proceed, we need to save our AmCAT password (only needed once, please don't save the password in your file!) and log onto amcat:

```
library(amcatr)
amcat.save.password("https://amcat.nl", "username",  "password")
conn = amcat.connect("https://amcat.nl")
```

Now, we can upload articles from R using the `amcat.upload.articles` function, which we now demonstrate with a single article but which can also be used to upload many articles at once:

```
articles = data.frame(text = "John is a great fan of chickens, and so is Mary", date="2001-01-01T00:00"

aset = amcat.upload.articles(conn, project = 1, articleset="Test", medium="test",
                             text=articles$text, date=articles$date, headline=articles$headline)
```

```
## Created articleset 29490: Test in project 1

## Uploading 1 articles to set 29490
```

And we can then lemmatize this article and download the results directly to R using `amcat.gettokens`:

```
tokens = amcat.gettokens(conn, project=1, articleset = aset, module = "corenlp_lemmatize")
```

```
## GET https://amcat.nl/api/v4/projects/1/articlesets/29490/tokens/?module=corenlp_lemmatize&page_size=
```

```
head(tokens)
```

word	para	sentence	term_id	offset	pos	lemma	token_id	aid
test	NA	1	t1	0	N	test	w1	166707788
John	NA	1	t2	5	R	John	w2	166707788
is	NA	1	t3	10	V	be	w3	166707788
a	NA	1	t4	13	D	a	w4	166707788
great	NA	1	t5	15	G	great	w5	166707788
fan	NA	1	t6	21	N	fan	w6	166707788

And we can see that e.g. for "is" the lemma "be" is given. Note that the words are not in order, and the two occurrences of "is" are automatically summed. This can be switched off by giving `drop=NULL` as extra argument.

For a more serious application, we will use an existing article set: set 16017 in project 559, which contains the state of the Union speeches by Bush and Obama (each document is a single paragraph)

This data is available directly from the corpustools package:

```
library(corpustools)
data(sotu)
nrow(sotu.tokens)
```

```
## [1] 91473
```

```
head(sotu.tokens, n=20)
```

word	sentence	pos	lemma	offset	aid	id	pos1	freq
It	1	PRP	it	0	111541965	1	O	1
is	1	VBZ	be	3	111541965	2	V	1
our	1	PRP$	we	6	111541965	3	O	1
unfinished	1	JJ	unfinished	10	111541965	4	A	1
task	1	NN	task	21	111541965	5	N	1
to	1	TO	to	26	111541965	6	?	1
restore	1	VB	restore	29	111541965	7	V	1
the	1	DT	the	37	111541965	8	D	1
basic	1	JJ	basic	41	111541965	9	A	1
bargain	1	NN	bargain	47	111541965	10	N	1
that	1	WDT	that	55	111541965	11	D	1
built	1	VBD	build	60	111541965	12	V	1
this	1	DT	this	66	111541965	13	D	1
country	1	NN	country	71	111541965	14	N	1
:	1	:	:	78	111541965	15	.	1
the	1	DT	the	80	111541965	16	D	1
idea	1	NN	idea	84	111541965	17	N	1
that	1	IN	that	89	111541965	18	P	1
if	1	IN	if	94	111541965	19	P	1
you	1	PRP	you	97	111541965	20	O	1

As you can see, the result is similar to the ad-hoc lemmatized tokens, but now we have around 100 thousand tokens rather than 6. We can create a document-term matrix using the dtm.create command from `corpustools`:

```
dtm = dtm.create(documents=sotu.tokens$aid, terms=sotu.tokens$lemma, filter=sotu.tokens$pos1 %in% c("M"
```

```
## Ignoring words with frequency lower than 5

## Ignoring words with less than 3 characters

## Ignoring words that contain numbers of non-word characters
```

```
dtm
```

```
## <<DocumentTermMatrix (documents: 1089, terms: 907)>>
## Non-/sparse entries: 15896/971827
## Sparsity           : 98%
## Maximal term length: 14
## Weighting          : term frequency (tf)
```

So, we now have a "sparse" matrix of almost 7,000 documents by more than 70,000 terms. Sparse here means that only the non-zero entries are kept in memory, because otherwise it would have to keep all 70 million cells in memory (and this is a relatively small data set). Thus, it might not be a good idea to use functions like `as.matrix` or `colSums` on such a matrix, since these functions convert the sparse matrix into a regular matrix. The next section investigates a number of useful functions to deal with (sparse) document-term matrices.

Corpus analysis: word frequency

What are the most frequent words in the corpus? As shown above, we could use the built-in `colSums` function, but this requires first casting the sparse matrix to a regular matrix, which we want to avoid (even our relatively small dataset would have 400 million entries!). However, we can use the `col_sums` function from the `slam` package, which provides the same functionality for sparse matrices:

```
library(slam)
freq = col_sums(dtm)
# sort the list by reverse frequency using built-in order function:
freq = freq[order(-freq)]
head(freq, n=10)
```

America	409
year	385
people	327
job	256
country	228
world	198
tax	181
Americans	179
nation	171
Congress	168

As can be seen, the most frequent terms are America and recurring issues like jobs and taxes. It can be useful to compute different metrics per term, such as term frequency, document frequency (how many documents does it occur), and td.idf (term frequency * inverse document frequency, which removes both rare and overly frequent terms). The function `term.statistics` from the `corpus-tools` package provides this functionality:

```
terms = term.statistics(dtm)
terms = terms[order(-terms$termfreq), ]
head(terms, 10)
```

	term	characters	number	nonalpha	termfreq	docfreq	reldocfreq	tfidf
America	America	7	FALSE	FALSE	409	346	0.3177227	0.1275801
year	year	4	FALSE	FALSE	385	286	0.2626263	0.1540023
people	people	6	FALSE	FALSE	327	277	0.2543618	0.1547707
job	job	3	FALSE	FALSE	256	190	0.1744720	0.2102813
country	country	7	FALSE	FALSE	228	202	0.1854913	0.1677862
world	world	5	FALSE	FALSE	198	156	0.1432507	0.2127368
tax	tax	3	FALSE	FALSE	181	102	0.0936639	0.3308686
Americans	Americans	9	FALSE	FALSE	179	158	0.1450872	0.2033421
nation	nation	6	FALSE	FALSE	171	150	0.1377410	0.1985566
Congress	Congress	8	FALSE	FALSE	168	149	0.1368228	0.1917011

As you can see, for each word the total frequency and the relative document frequency is listed, as well as some basic information on the number of characters and the occurrence of numerals or non-alphanumeric characters. This allows us to create a 'common sense' filter to reduce the amount of terms, for example removing all words containing a letter or punctuation mark, and all short (`characters<=2`) infrequent (`termfreq<25`) and overly frequent (`reldocfreq>.5`) words:

```
subset = terms[!terms$number & !terms$nonalpha & terms$characters>2 & terms$termfreq>=25 & terms$reldoc
nrow(subset)
```

```
## [1] 192
```

```
head(subset, n=10)
```

	term	characters	number	nonalpha	termfreq	docfreq	reldocfreq	tfidf
job	job	3	FALSE	FALSE	256	190	0.1744720	0.2102813
country	country	7	FALSE	FALSE	228	202	0.1854913	0.1677862
world	world	5	FALSE	FALSE	198	156	0.1432507	0.2127368
tax	tax	3	FALSE	FALSE	181	102	0.0936639	0.3308686
Americans	Americans	9	FALSE	FALSE	179	158	0.1450872	0.2033421
nation	nation	6	FALSE	FALSE	171	150	0.1377410	0.1985566
Congress	Congress	8	FALSE	FALSE	168	149	0.1368228	0.1917011
time	time	4	FALSE	FALSE	166	137	0.1258035	0.2070660
child	child	5	FALSE	FALSE	153	112	0.1028466	0.2254874
economy	economy	7	FALSE	FALSE	150	122	0.1120294	0.2582953

This seems more to be a relatively useful set of words. We now have about 8 thousand terms left of the original 72 thousand. To create a new document-term matrix with only these terms, we can use normal matrix indexing on the columns (which contain the words):

```
dtm_filtered = dtm.filter(dtm, terms=subset$term)
dim(dtm_filtered)
```

```
## [1] 1082  192
```

Which yields a much more managable dtm. As a bonus, we can use the `dtm.wordcloud` function in corpustools (which is a thin wrapper around the `wordcloud` package) to visualize the top words as a word cloud:

```
dtm.wordcloud(dtm_filtered)
```