Topic Modeling with R

Wouter van Atteveldt

June 3, 2016

Topic modelling techniques such as Latent Dirichlet Allocation (LDA) can be a usefull tool for social scientists to analyze large amounts of natural language data. Algorithms for LDA are available in R, for instance in the `topicmodels` package. In this howto we demonstrate several function in the `corpustools` package that facilitate the use of LDA using the `topicmodels` package.

As a starting point we use a Document Term Matrix (dtm) in the `DocumentTermMatrix` format offered in the `tm` package. Note that we also offer a howto for creating the dtm.

```
library(corpustools)
data(sotu) # state of the union speeches by Barack Obama and George H. Bush.
head(sotu.tokens)
```

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

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

```
## <<DocumentTermMatrix (documents: 1090, terms: 1133)>>
## Non-/sparse entries: 20342/1214628
## Sparsity           : 98%
## Maximal term length: 14
## Weighting          : term frequency (tf)
```

Not all terms are equally informative of the underlying semantic structures of texts, and some terms are rather useless for this purpose. For interpretation and computational purposes it is worthwhile to delete some of the less usefull words from the dtm before fitting the LDA model. As seen from the red message lines above, the dtm.create automatically uses some filtering of terms, but it can be good to customize this for your research.

Now we are ready to fit the model! We made a wrapper called `lda.fit` for the `LDA` function in the `topicmodels` package. This wrapper doesn't do anything interesting, except for deleting empty columns/rows from the dtm, which can occur after filtering out words.

The main input for `topmod.lda.fit` is: - the document term matrix - K: the number of topics (this has to be defined a priori) - Optionally, it can be usefull to increase the number of iterations. This takes more time, but increases performance.

```
set.seed(12345)
m = lda.fit(dtm, K=20, num.iterations=100)
terms(m, 10)[,1:5] # show first 5 topics, with ten top words per topic
```

```
##       Topic 1     Topic 2      Topic 3     Topic 4       Topic 5
##  [1,] "health"    "job"        "life"      "America"     "child"
##  [2,] "care"      "new"        "work"      "nation"      "school"
##  [3,] "reform"    "home"       "time"      "opportunity" "education"
##  [4,] "Americans" "company"    "american"  "action"      "college"
##  [5,] "cost"      "today"      "community" "course"      "student"
##  [6,] "insurance" "first"      "month"     "prosperity"  "high"
##  [7,] "Medicare"  "place"      "Americans" "Congress"    "better"
##  [8,] "coverage"  "investment" "public"    "side"        "higher"
##  [9,] "medical"   "industry"   "success"   "old"         "parent"
## [10,] "plan"      "innovation" "thousand"  "easy"        "training"
```

We now have a fitted lda model. The `terms` function shows the most prominent words for each topic (we only selected the first 5 topics for convenience).

Visualizing LDA models

The easiest way to visualize an LDA model is using the LDAvis package:

```
library(LDAvis)
serVis(ldavis_json(m, dtm))
```

(run interactively to see results)

Another way to visualize LDA results is to plot word use and metadata about the indivual topics: E.g., how much attention they get over time, how much they are used by different sources (e.g., people, newspapers, organizations). To do so, we first need to match the metadata to the documents in the model to make sure they are in the same order:

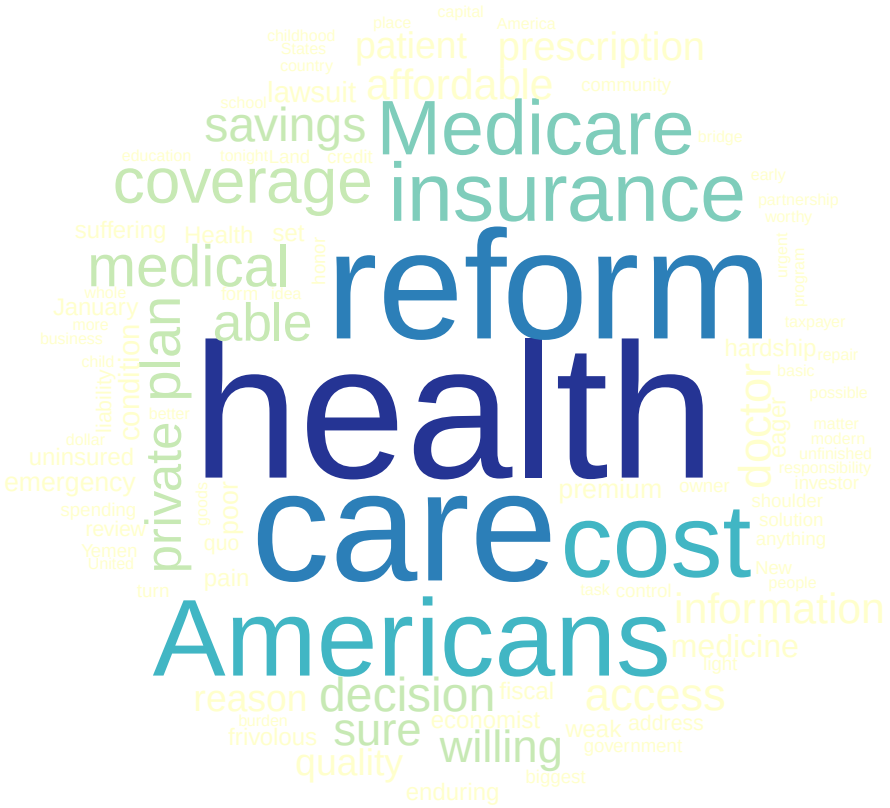
```
head(sotu.meta)
```

id	medium	headline	date
111541965	Speeches	Barack Obama	2013-02-12
111541995	Speeches	Barack Obama	2013-02-12
111542001	Speeches	Barack Obama	2013-02-12
111542006	Speeches	Barack Obama	2013-02-12
111542013	Speeches	Barack Obama	2013-02-12
111542018	Speeches	Barack Obama	2013-02-12

```
meta = sotu.meta[match(m@documents, sotu.meta$id),]
```

We can now do some plotting. First, we can make a wordcloud for a more fancy (and actually quite informative and intuitive) representation of the top words of a topic.

```
lda.plot.wordcloud(m, topic_nr=1)
```



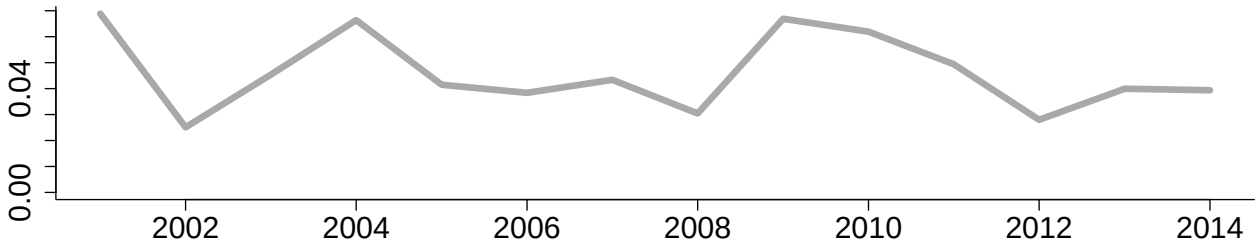

```
lda.plot.wordcloud(m, topic_nr=2)
```

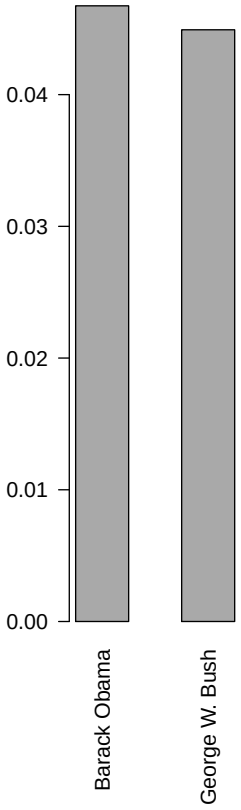
With `lda.plot.time` and `lda.plot.category`, we can plot the salience of the topic over time and for a given categorical variable.

```
lda.plot.time(m, 1, meta$date, date_interval='year')
lda.plot.category(m, 1, meta$headline)
```

It can be usefull to print all this information together. That is what the following function does.

```
lda.plot.topic(m, 1, meta$date, meta$headline, date_interval='year')
```






```
lda.plot.topic(m, 2, meta$date, meta$headline, date_interval='year')
```

For further substantive analysis, we can also create a data frame containing the topic proportion for each document:

```
docs = topics.per.document(m)
docs = merge(meta, docs)
head(docs)
```

id	medium	headline	date	X1	X2	X3	X4	X5	
111541965	Speeches	Barack Obama	2013-02-12	0.0431034	0.0603448	0.0431034	0.0431034	0.0431034	0.0775
111541995	Speeches	Barack Obama	2013-02-12	0.0312500	0.0312500	0.0312500	0.0562500	0.0812500	0.0312
111542001	Speeches	Barack Obama	2013-02-12	0.0301205	0.0421687	0.0542169	0.0421687	0.2108434	0.0421
111542006	Speeches	Barack Obama	2013-02-12	0.0367647	0.0367647	0.0514706	0.0367647	0.1397059	0.0661
111542013	Speeches	Barack Obama	2013-02-12	0.0320513	0.0448718	0.0448718	0.0448718	0.0576923	0.0448
111542018	Speeches	Barack Obama	2013-02-12	0.0337838	0.0743243	0.1418919	0.0337838	0.0472973	0.0337

Calculating perplexity

Although in the end the best guide to determining the amount and interpretation of topics is expert judgment, it can be useful to plot the 'perplexity' (model error) of various settings for K (and alpha).

The first step is to create separate data for fitting the model and validating it to avoid overfitting, basically using a split-half technique. Although it is easy to subset the dtm based on a sample of row names (ids), care must be taken that there are no zero-only rows and columns, and that the vocabulary (colnames) of the validation dtm matches that of the dtm used to fit the model:

```
ids = rownames(dtm)
fit_ids = sample(ids, length(ids) / 2)
dtm.subset <- function(dtm, rows, cols=colnames(dtm)) {
  dtm = dtm[rownames(dtm) %in% rows, colnames(dtm) %in% cols]
  dtm = dtm[row_sums(dtm) > 0, col_sums(dtm) > 0]
  weightTf(dtm)
}
dtm_fit = dtm.subset(dtm, fit_ids)
dtm_validate = dtm.subset(dtm, setdiff(ids,  fit_ids), colnames(dtm_fit))
```

Now, we can calculate the perplexity by fitting a number of models for each K, adding each to a data frame of perplexity scores:

```
perplex = NULL
for (k in seq(10, 50, by=10)) {
  for (i in 1:2) {
    m = lda.fit(dtm_fit, K=k, alpha=.1)
    p = perplexity(m, dtm_validate)
    perplex = rbind(perplex, c(k=k, i=i, p=p))
  }
}
head(perplex)
```

k	i	p
10	1	1866.089
10	2	1906.179
20	1	1946.521
20	2	1975.587
30	1	1809.673
30	2	1885.335

Now we can plot the average perplexity per K in a scree plot:

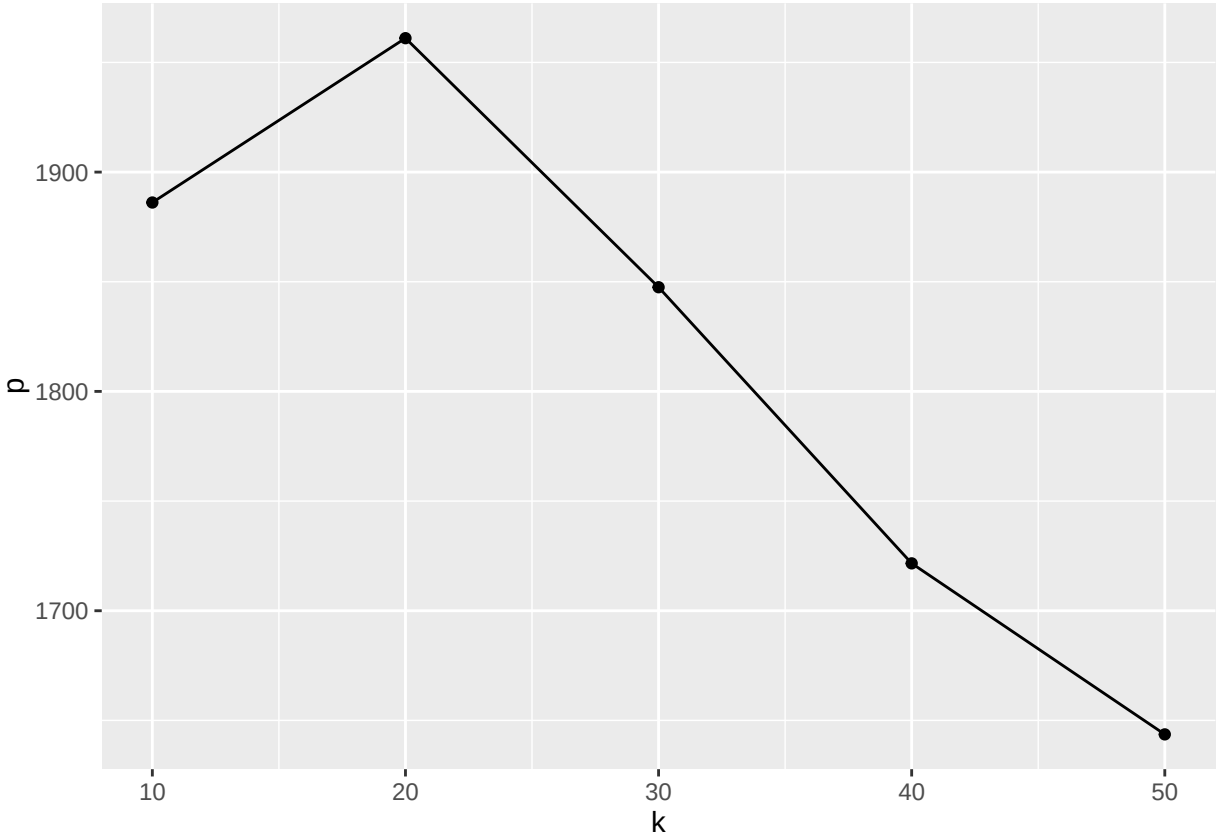
```
perplex = as.data.frame(perplex)
p = aggregate(perplex["p"], perplex["k"], mean)
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
```

```
##
##     annotate
```

```
ggplot(p, aes(x=k, y=p, )) + geom_line()  +geom_point()
```



This suggests that it could be interesting to inspect the region around k=10 better, as perplexity actually increased going to k=20. Perplexity is also still decreasing at k=50, but there seems to be an elbow point at k=40. Note that normally you should use more iterations per k and also test intermediate k values, increasing computational complexity especially for very large datasets.