tosca: Tools for Statistical Content Analysis

Lars Koppers, Jonas Rieger, Karin Boczek, Gerret von Nordheim

April 22, 2025

Contents

1 Introduction

This package provides different functions to explore text corpora with topic models. The package focuses on the visualisation and validation of content analysis. Therefore it provides some filters for preprocessing and a wrapper for the latent Dirichlet allocation (lda) from the lda-package to include a topic model. Most visualisations aim at the presentation of measures for corpora, subcorpora or topics from lda over time. To use this functionality every document needs a date specification as metadata. To harmonize different text sources we provide the S3 object `textmeta`.

The following table shows an overview over the functions in the package.

Function	Description
Preprocessing	
cleanTexts	tokenization, stopwords, removal of numbers, punctuation
deleteAndRenameDuplicates	deletion of duplicates, correction of not unique id's
duplist	list of different duplication types
filterCount	filter texts with few words
filterDate	filter texts depending on date
filterWord	filter texts depending on word lists
is.duplist	generic function for S3 object duplist
is.textmeta	generic function for S3 object textmeta
makeWordlist	wordlists and wordtables
mergeTextmeta	combining corpora
readTextmeta	create textmeta objects from csv files
readWiki	create textmeta objects from Wikipedia
readWikinews	create textmeta objects from Wikinews
removeHTML	converts html entities in UTF-8
removeUmlauts	converts german umlauts
removeXML	remove xml (html) tags
textmeta	S3 object textmeta
Topic Models	
LDAgen	wrapper for the lda in the lda-package
LDAprep	converts a tokenized textmeta object for the lda-package
clusterTopics	cluster analysis for topics
mergeLDA	combining lda-topics (for cluster analysis)
Descriptive Analysis: Corpus	
plotFreq	plotting wordcounts over time
showMeta	export of meta data in csv
showTexts	export of text data in csv
Descriptive Analysis: Topics	
plotArea	area plot for topics over time
plotHeat	heatmaps for topics over time
plotScot	plotting document counts of subcorpora over time
plotTopic	plotting topic counts over time
plotTopicWord	plotting wordcounts/proportion in topics over time
plotWordSub	plotting wordcounts/proportion in supcorpora relative to the original corpus
plotWordpt	plotting wordcounts/proportion relative to the topics
topTexts	filter representative texts for topics
topicsInText	visualisation of topics in a text

Function	Description
Validation	
intruderTopics	intruder topics for topic validation
intruderWords	intruder words for topic validation

The current version of the package can be installed with the `devtools` package. The data for this vignatte can be found in the toscaData package on gitHub.

```
devtools::install_github("DoCMA-TU/tosca")
devtools::install_github("DoCMA-TU/toscaData")
library(tosca)
```

The actual version on CRAN can be installed with `install.packages`.

```
install.packages("tosca")
library(tosca)
```

This vignette gives an overview over the functionality of the package. For a detailed description of the functions see the help pages.

2 Data Preprocessing

A basic functionality of the package is data preprocessing. Therefore several functions are provided for reading text data, creating text objects, manipulating these objects and especially handling duplicates of different forms in the text data.

2.1 Read the Corpus - `textmeta`, `readWikinews`

Read the corpus data through one of your self-implemented read-functions and create a `textmeta` object with the function of the same name and the arguments `text`, `meta` and `metamult`. The `text` component should be a `list` of `character` vectors or a `list` of `lists` of `character` vectors, whereas `meta` is a `data.frame` and `metamult` is intended for mainly unstructured meta-information as a `list`. Furthermore `meta` must contain the columns `id`, `date` and `title`. You can test whether your object meets the requirements of a `textmeta` object with the function `is.textmeta`.

A read-function which is part of the package `tosca` is the function `readWikinews`. `readWikinews` reads XML-files created by the wikinews export page: https://en.wikinews.org/wiki/Special:Export. By default `readWikinews` reads all XML-files in the working directory. The function creates a `textmeta` object. For this vignette we used two categories: *Politics_and_conflicts* and *Economy_and_business*. The pages were downloaded on 2018-03-05 in a file for each category. We can use `readWikinews` for reading both files, if they are in the same folder.

```
corpus <- readWikinews()
```

Another method to read both files is to read them seperately and merge them with the function `mergeTextmeta`. This function should be used if you want to merge data from different sources using different read-functions. We use the two example datasets from the package.

```
data(politics)
data(economy)
corpus <- mergeTextmeta(list(politics, economy))
```

You obtain a note about duplicated texts (texts that appear in both categories). We have to handle this issue later. If we merge corpora with different meta-variables we can decide if all variables are used for the merged corpora (`all = TRUE`, default) or only variables that appear in all corpora (`all = FALSE`).

After reading the raw data the texts need to be preprocessed.

2.2 Remove Umlauts and XML/HTML Tags - `removeXML` `removeHTML` `removeUmlauts`

You can use `removeXML` to delete XML-tags (`<...>`) in character strings or a `list` of `character` vectors. The value you receive back are either a `character` vector or a list, if the input was a list.
If your texts contain html entities use `removeHTML`. If you want to transform the entities in UTF-8 characters you can choose between the entity-type (`dec=TRUE`: &#248;, `hex=TRUE`: &#xf8; or `entity=TRUE`: &oslash;). If you are unsure which type was used, we recommend to enable all entity-type (disadvantage: longer run time). To choose which character should be replaced you can choose from all 16 ISO-8859 lists, e.g. `symbolList=c(1,15)` for ISO-8859-1 (latin1) and ISO-8859-15 (latin9). If `delete=TRUE` all remaining entities will be deleted. To replace german umlauts (ä ö ü ß -> ae oe ue ss) use `removeUmlauts`.

We remove XML-tags and HTML-entities from our Wikinews corpus. Since we have only punctuation as HTML-entities in the Corpus we remove it completely.

```
corpus$text <- removeXML(corpus$text)
corpus$text <- removeHTML(corpus$text, dec=FALSE, hex=FALSE, entity=FALSE)
```

It is possible to apply the function to the `meta` component of a `textmeta` object as well, for example to remove XML tags or umlauts from the title of the Wikipedia pages.

```
corpus$meta$title <- removeXML(corpus$meta$title)
corpus$meta$title <- removeHTML(corpus$meta$title, dec=FALSE, hex=FALSE, entity=FALSE)
```

After applying the function to the text component, we have removed all database relicts like XML-tags. At this point you should deal with identifying different types of duplicates in your text data.

2.3 Identifying Duplicates - `deleteAndRenameDuplicates`, `duplist`

You should ensure unique IDs in all three components of your `textmeta` object. If you cannot ensure that, it is recommended to use the function `deleteAndRenameDuplicates`. This function performs three actions. It deletes "complete duplicates", i.e. at least two entries with same ID and same information in `text` *and* in `meta`. It renames so called "real duplicates" i.e. at least two entries with same ID and text, but diferrent information in meta, and it and renames also "fake duplicates" i.e. at least two entries with same ID but different `text` components. It is important to know that for technical reasons - expecting duplicates in the names of the `lists` - this is the only function, which works with classic indexing, so that it assumes the same order of articles in all three components.

Additionally you can identify `text` component duplicates in your corpus with the function `duplist`, which creates a `list` of different types of duplicates. Non-unique IDs are not supported by the function, which implies that `deleteAndRenameDuplicates` should be executed before.

In the given example corpus complete duplicates are only expected if pages were associated to both categories. These duplicates are deleted.

```
any(duplicated(corpus$meta$id))
sum(duplicated(names(corpus$text)))
length(corpus$text) - nrow(corpus$meta)
corpus$meta <- corpus$meta[match(names(corpus$text), corpus$meta$id),]
corpus <- deleteAndRenameDuplicates(corpus)
```

The function `deleteAndRenameDuplicates` deleted 286 complete duplicates, so that `duplist` is applicable to the corpus.

```
dups <- duplist(corpus)
```

There is a possibility to visualize duplicates over time by the function `plotScot` which is explained in section 3.2.

For further analysis, especially for performing the latent Dirichlet allocation, it is important that for each duplicate only one page is considered. Therefore it is the aim to reduce the corpus, so that it contains all pages which appear only once and a represantative page for all pages which appear twice or more frequent. In our example we have only duplicated texts containing the empty string `""` or short relicts like `"__NOTOC__"` or `"* * *"`

2.4 Clean Corpus - `cleanTexts`

For further preprocessing of text corpora tosca offers the function `cleanTexts`. It removes punctuation, numbers and stopwords. By default it removes english stopwords. It uses the stopword list of the function `stopwords` from the `tm` package. For the german stopword list some additional word (different spelling) are implemented (e.g. "dass" and "fuer"). You can control which stopwords should be removed with the argument `sw`. In addition the function changes all words to lowercase and tokenizes the documents. The result is a `list` of `character` vectors, or if `paragraph` is set `TRUE` (default) a `list` of `lists` of `character` vectors. The sublists represent additional text structure like paragraphs of a document. If you commit a `textmeta` object instead of a `list` of texts you will also receive a textmeta object back. In this case you have to commit it to the parameter `object` instead of `text`.

The language of the example corpora is english, so that `sw` should be set to `stopwords()` from the `tm` package, which includes english stopwords by default (`kind = "en"`).

```
corpusClean <- cleanTexts(object = corpus)
```

The function `cleanTexts` deletes all `meta` entries which do not belong to one of the texts (e.g. deleted empty texts). To create a `textmeta` object including this data the corresponding function is used.

```
textClean2 <- cleanTexts(text = corpus$text)
corpusClean2 <- textmeta(text = textClean2, meta = corpus$meta)
```

2.5 Generate Wordlist - `makeWordlist`

After cleaning the corpus with the function `cleanTexts` we are able to call the function `makeWordlist`, which creates a table of all words that occur in a given corpus. The function `table` needs a high amount of RAM. That's a problem for very large Corpora. In `makeWordlist` we use the parameter `k` (default: `100000L`) to reduce the number of texts which are processed at once. Large values of `k` lead to faster calculations but require more RAM usage.

For calculating wordlists a tokenized corpus must be used. In the given example `corpusClean$text` is committed to the function accordingly.

```
wordtable <- makeWordlist(corpusClean$text)
```

3 Descriptive Analysis

After preprocessing the text data there is a typical workflow we highly reccomend as initial descriptive data analysis of the corpus. This workflow contains the generic functions `print` and `summary` as well as the highly adaptable functions `plotScot` and `plotFreq`. These graphical functions should be part of any initial analysis of text data.

3.1 Generic Functions - `print`, `summary`

Some information about the (one to) three components of the `textmeta` object is obtained by calling the generic function `print`.

```r
print(corpus)
```

The function provides the number of pages in the corpus (7041) and adds two additional columns in `meta` to the mandatory ones `id`, `date` and `title`. The pages are dated from 2004-11-13 till 2018-03-04.

You obtain more information, especially about counts of `NA`s and tables of some `candidates` (default: `resource` and `downloadDate`) with the generic function `summary`. In addition to `candidates` you can commit the argument `list.names` (default: `names(object)`) for specifying which components out of `text`, `meta` and `metamult` should be analysed by the function.

```r
summary(corpus)
```

Apparently there are 191 `NA`s in the variable `date`.

3.2 Visualisation of Corpus over Time - `plotScot`

One of the descriptive plotting functions in the package is `plotScot` (**S**ub**C**orpus**O**ver**T**ime) which creates a plot of counts or proportions of either documents or words in a (sub)corpus over time. The subcorpus is specified by `id` and it is possible to set the `unit` to which the dates should be floored (default: `"month"`). The argument `curves = c("exact", "smooth", "both")` determine which curve(s) should be plotted. If you select `type = "words"`, the object which you commit should be a tokenized `textmeta` object. If `type = "docs"` (default) you can commit untokenized `textmeta` object as well.

First of all the number of texts per month in the complete example corpus is plotted, as exact and smoothed curve.

```r
plotScot(corpusClean, curves = "both")
```

The black curve is the exact one and the red curve represents the smoothed values. The grafic gives a first impression about the distribution of the texts over time. Most of the news articles where written between 2005 and 2009. If you want to identify the distribution of duplicates over the time you can use `plotScot` to plot the IDs of the not duplicated texts in the corpus.

```r
plotScot(corpus, id = dups$notDuplicatedTexts, rel = TRUE)
```

The plot shows that between 2006 and 2011 around 80 per cent of the corpus are not duplicated texts. Most zeros in the plot result from no articles in the whole corpus during these time periods. It is possible to set these values to `NA` by setting `natozero = FALSE` in `plotScot`. This option works if `rel = TRUE` and is offered by many other functions in the package. Usually all plot functions in the package return the data belonging to the plot as invisible output. These plot functions offer a lot more functionality, which is described in the corresponding help functions.

3.3 Frequency Analysis - `plotFreq`

The other descriptive plotting function is `plotFreq` which performs a frequency analysis. Most of the arguments are the same as in `plotScot`. The options `wordlist` and `link = c("and", "or")` are added for specifying the words of the frequency analysis and their link within one vector. In detail `wordlist` could either be a `list` of `character` vectors or a single `character` vector, which will be coerced to a `list` of the vector's length. Each `list` entry represents a set of words which all (default `link = "and"`) or one of them (`link = "or"`) should appear in an article to be counted. The function uses `filterWord` with `out = "count"` for counting, which is explained later on.

The example corpus contains Wikinews articles concerning the categories *Politics_and_conflicts* and *Economy_and_business*. Therefore some typical words out of these categories are selected to perform a frequency analysis. As a first example the words *unemployment*, *growth* and *trade* were used. The function identifies patterns.

```r
wordsEconomy <- list("unemployment", "growth", "trade",
                     c("unemployment", "growth", "trade"))
plotFreq(corpusClean, wordlist = wordsEconomy, curves = "smooth",
  ylim = c(0, 25), legend = "topright",
  main = "Wordlist-filtered texts over time. link: and")
plotFreq(corpusClean, wordlist = wordsEconomy, link = "or", curves = "smooth",
  ylim = c(0, 25), legend = "topright",
  main = "Wordlist-filtered texts over time. link: or")
```

In the figures above you can see the difference between the *and* link and the *or* link. In the first figure three curves indicate the single words. The fourth curve shows the number of texts in which all three words appear. For most dates no texts meets this requirement. In the second figure the same three curves representing single words are shown. The fourth curve represents all three words again, but setting `link = "or"`. The curve lies above the three others in every point. Due to smoothing it is possible that the line falls under one of the single word lines. This can be avoided by choosing `curves = "exact"`.

In another figure the counts of pages in which the words *crisis*, *war* and *conflict* appear, are analysed. You can see that it is often useful to compare smoothed and exact curves to visualize the variance and a trend in the data.

```r
plotFreq(corpusClean, wordlist = list(c("crisis", "war", "conflict")), link = "or",
  curves = "both", both.lwd = 2, legend = "topright",
  main = "Wordlist-filtered texts over time. link: or")
```

3.4 Write CSV Files - `showTexts`, `showMeta`

There are two functions for writing information from a `textmeta` object in csv files implemented in the package. Both need a `textmeta` object in `showTexts`, respectively the `meta` component of any-formated `textmeta` object in `showMeta`. The default of the parameter `id` in `showTexts` are all document IDs of the corpus as a `character` vector, but it is possible to commit a `character` matrix as well, so that each column will be represented in a seperated csv file. In the first column of the csv file there will be the ID of each document, in the second and third the title and the date and the fourthe column contains the text itself.

Six IDs are sampled from the whole corpus with a given seed. Since we don't use the `file` parameter, the dataset is only returned as invisible to `temp`. To generate a csv file `file` must be specified.

```r
set.seed(123)
ids.selected <- sample(corpus$meta$id, 6)
temp <- showTexts(corpus, id = ids.selected)
temp[, c("id", "date", "title")]
```

We now take a look at the meta data. The default of the parameter `id` in `showMeta` are the IDs which are in the column `meta$id`. You can also commit a matrix of IDs like in `showTexts` and you can specify which columns of the `meta` component to write in the csv file by setting the argument `cols` (default: `colnames(meta)`).

Analogously to `showTexts` the following code example will create three files named `corpus<i>meta.csv`, where $i = 1, 2, 3$ stands for the i-th column of the matrix of IDs.

```r
temp <- showMeta(corpus$meta, id = matrix(ids.selected, nrow = 2),
  cols = c("title", "date"))
temp
```

4 Generating Subcorpora

The preprocessing presented above is mandatory. For further preparation the package offers functions for filtering the corpus by dates, wordcount or search terms to generate subcorpora. There are three implemented ways to filter your corpora: filterDate for date filter, filterCount for wordcounts, and filterWord for word and pattern filter.

4.1 Filter Corpus by Dates - filterDate

filterDate, filters a given textmeta object by a time period. The function works on any formated object of class textmeta and extracts documents out of the text component, from which the date column in the meta component is between s.date and e.date - including documents from both exact dates. The return value is either the filtered textmeta object or a list, e.g. the text component of a textmeta object, if you commit the text and the meta component not as a textmeta object.

The example corpus is filtered to articles dated between 2006 and 2009.

```
corpusDate <- filterDate(corpusClean, s.date = "2006-01-01", e.date = "2009-12-31")
print(corpusClean)
print(corpusDate)
```

The filtered corpus contains only the 3909 documents from the period 2006 till 2009.

4.2 Filter Corpus by Wordcount - filterCount

After cleaning the example corpus and restricting it to the given dates it consists of documents with the distribution of wordcounts (including symbols) given below.

```
textCounts <- lengths(corpusDate$text)
quantile(textCounts, probs = c(0, 0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 0.95, 1))
```

To exclude very short documents from your corpus. You can use the function filterCount. The function considers only words which only consist of letters and which are seperated by any word seperating symbol like whitespace or punctuation. Tokenized documents can also be processed. The function call filterCount(corpus, count = 5) for example deletes all documents from corpus that consist of less than five words.

```
mean(textCounts != filterCount(corpusDate, out = "count"))
```

The counts returned by filterCount need not to match the lengths of the tokenized documents exactly, because based on the way of preprocessing the corpus may contain symbols - which leads to smaller counts - or multiple words in one token. These multiple words could be unaffected by tokenization in cleanTexts if they are seperated by remaining symbols instead of whitespace and then lead to higher counts.

4.3 Filter Corpus by Words - filterWord

The use of filterWord works analogously. It filters the text component of a textmeta object by appearances of specific words. The function uses regular expressions. It filters the given documents in the text component by words committed by search, which could be a simple character vector or a list of data.frames. In the case of a character vector committed the entries of the vector are linked by an *or*, so if *any* of the words appears in one specific document, it is returned.

If you are not interested in the texts of the documents itself you can set out to control the output: By default (out = text) you receive the filtered documents. If you commit the argument object you receive the corresponding textmeta object. If you choose out = bin you get the corresponding logical vector of indices, and if you choose out = count you get a matrix that contains in row i and column j how often the j-th word of the wordlist appears in the i-th document.

Now examples are given for understanding the functionality of the function filterWord. An example for the *or*-link is given by the next code example.

```r
toyCorpus <- list(text1 = "dataset", text2 = "anything")
searchterm <- c("data", "as", "set", "anything")
filterWord(text = toyCorpus, search = searchterm, out = "bin")
```

The returned values are both `TRUE`. There is at least one pattern in the `searchterm` vector which appears at least once in each of the strings *dataset* and *anything*.

In the case of a `list` of `data.frames` committed each `data.frame` is linked by an *or* and should contain columns `pattern`, `word`, and `count`. The parameter `pattern` includes the search terms, the column `word` is a logical variable which controls whether words (`TRUE`) or patterns are searched. Alternatively `word` can be a `character` string containing the keyword `left` or `right` for left- or right-truncated search, i.e. word = right searches for the exact pattern on the left of the word and all possible endings of the pattern. You must set the argument `count` to an `integer`. This argument controls how often a word or pattern must appear in a document to be returned. Rows in each `data.frame` are linked by an *and*. An example is given by the following code.

```r
searchframe <- data.frame(pattern = searchterm, word = FALSE, count = 1)
filterWord(text = toyCorpus, search = searchframe, out = "bin")
```

In the case that `search` is committed as `data.frame`, the *and* link is active. The function checks whether all of the patterns appear as part of words in the two entries of `texts`. Therefore the function returns `FALSE` twice.

For another emeplary case, we will delete the word *anything* from the search terms.

```r
filterWord(text = toyCorpus, search = searchframe[1:3,], out = "bin")
```

By omitting the word *anything* from `searchframe` you receive a `TRUE` for text1 (*dataset*) - all three patterns appear in it - and a `FALSE` for text2 (*anything*), because not all patterns appear in it, not even one of them.

An example with `out = count` to receive a count for each document and search term combination is the following.

```r
filterWord(text = list(text1 = c("i", "was", "here", "text"),
  text2 = c("some", "text", "about", "some", "text", "and", "something", "else")),
  search = c("some", "text"), out = "count")
```

In the case of `out = count` it is useful, that `search` is a simple `character` vector.

Another application of `filterWord` is to apply the function with `word = TRUE`, so that the function searches only for single words, not for strings containing these words. This is displayed by the following example.

```r
searchterm <- list(text1 = "land and and", text2 = c("and", "land", "and", "and"))
searchframe <- list(
  data.frame(pattern = "and", word = FALSE, count = 1),
  data.frame(pattern = "and", word = TRUE, count = 1))
filterWord(text = searchterm, search = searchframe, out = "count")
```

The function returns counts `c(3, 4)` for the simple pattern search and `c(2, 3)` for the word search, because the word *and* appears once in every document of `searchterm` only as pattern and not as single word.

After understanding the functionality of the function, now it is used for filtering the Wikipedia corpus. The example corpus is filtered to those pages which include the names of the categories as a pattern at least once. It is not necessary to set `ignore.case` because the Wikipedia corpus was cleaned before. This step includes that all words are lowercase now.

```r
searchterm <- list(
  data.frame(pattern = "economy", word = FALSE, count = 1),
  data.frame(pattern = c("world", "economy"), word = FALSE, count = 1),
  data.frame(pattern = "politics", word = FALSE, count = 1))
```

```r
corpusFiltered <- filterWord(corpusDate, search = searchterm)
print(corpusDate)
print(corpusFiltered)
```

The date and word filtered corpus consists of 451 documents compared to 3909 documents in the original `corpusDate` corpus.

5 Latent Dirichlet Allocation

The central analytical functionality in this package is to perform and analyse a latent Dirichlet allocation. The package provides the function `LDAgen` for performing the LDA, functions for validating the LDA results and various functions for visualizing the results in different ways, especially over time. It is possible to analyse individual articles as well as their topic allocations. In tosca in addition a function for preparing your corpus for performing a latent Dirichlet allocation is given. This function creates a object which can be committed to the function you could use for a LDA.

5.1 Transform Corpus - `LDAprep`

The last step before performing a latent Dirichlet allocation is to create corpus data, which can be committed to the function `lda.collapsed.gibbs.sampler` from the `lda` package or the function `LDAgen` from this package, respectively. This is done by using the function `LDAprep` with its arguments `text` (`text` component of a `textmeta` object) and `vocab` (`character` vector of vocabularies). These vocabularies are the words which are taken into account for LDA.

You can have a look at the documentation of the `lda.collapsed.gibbs.sampler` for further information about lda. The function `LDAprep` offers the option `reduce` all set to `TRUE` by default. The returned value is a `list` in which every entry represents an article and contains a matrix with two rows. In the first row there is the index of each word in `vocab` minus one (The index starts at 0), in the second row is always one and the number of the appearances of the word is indicated by the number of columns belonging to this word. This structure is needed by `lda.collapsed.gibbs.sampler`.

For the example corpus first a new wordlist must be generated based on the filtered corpus.

```r
wordtableFiltered <- makeWordlist(corpusFiltered$text, method = "radix")
```

```r
head(sort(wordtableFiltered$wordtable, decreasing = TRUE))
```

```r
words5 <- wordtableFiltered$words[wordtableFiltered$wordtable > 5]
pagesLDA <- LDAprep(text = corpusFiltered$text, vocab = words5)
```

After receiving the words which appear at least six times in the whole filtered corpus, the function `LDAprep` is applied to the example corpus with `vocab = words5`. The object `pagesLDA` will be committed to the function which performs a latent Dirichlet allocation.

5.2 Performing LDA - `LDAgen`

The function that has to be applied first to the corpus prepared by `LDAprep` is `LDAgen`. The function offers the options `K` (`integer`, default: `K = 100L`) to set the number of topics, `vocab` (`character` vector) for specifying the words which are considered in the preparation of the corpus and several more e.g. number of iterations for the burnin (default: `burnin = 70`) and the number of iterations for the Gibbs sampler (default: `num.iterations = 200`). The result is saved in a `R` workspace, the first part of the results name can be specified by setting the option `folder` (default: `folder = file.path(tempdir(),"lda-result")`). If you want to save your data permanent, you have to change the path in an non temporary one.

In the concrete example corpus the manipulated corpus `pagesLDA` is used for `documents`, the topic number is set to `K = 10` and for reproducibility a seed is set to `seed = 123`. The filename consists of the `folder` argument followed by the options of `K`, `num.iterations`, `burnin` and the `seed` of the LDA. The hyperparameter `alpha` and `eta` are set to $1/K$ by default.

```
result <- LDAgen(documents = pagesLDA, K = 10L, vocab = words5, seed = 123)
load(file.path(tempdir(),"lda-result-k10i200b70s123alpha0.1eta0.1.RData"))
```

For validation of the LDA result and further analysis, the result is loaded back to the workspace.

5.3 Validation of LDA Results - `intruderWords`, `intruderTopics`

For validation of LDA results there are two functions in the package. `intruderWords` and `intruderTopics` are extended version of the method Chang et al. (2009) present in their paper "Reading Tea Leaves: How Humans Interpret Topic Models". These functions expect user input, the user works like a document labeler. The LDA result is committed by setting `beta = result$topics`. From the function `intruderWords` the labeler gets a set of words. The number of words can be set by `numOutwords` (default: 5). This set represents one topic. It includes a number of intruders (default: `numIntruder = 1`), which can also be zero. In general, if the user identifies the intruder(s) correctly this is an identifier for a good topic allocation. You can set options `numTopwords` (default: 30) to control how many top words of each topic are considered for this validation. In addition it is possible to enable or disable the possibility for the user to mark nonsense topics. By default this option is enabled (`noTopic = TRUE`). The true intruder can be printed to the console after each step with `printSolution = TRUE` (default: `FALSE`).

The LDA result of the example corpus is checked by `intruderWords` with the number of intruders being either 0 or 1.

```
set.seed(155)
intWords <- intruderWords(beta = result$topics, numIntruder = 0:1)
```

As an illustration the first set is shown. The word *parliament* does not fit into the set with the words *obama*, *bush*, *presidential* and *mccain*. Therefore the user would type `1` and press enter. If the user wants to mark nonsense topics he would type an `x` (in the summary the number of meaningful topics is shown) and `0` if he thinks there is no intruder word. Actually *will* is the true intruder in the set above. As an example user input `c(1, 2, 5, 1, 4, 0, 0, 5, 1, 2)` is considered.

```
print(intWords)
```

By printing the object `intruderWords` generated by to the console, you get information about options for the validation strategy and a results matrix with ten rows an three columns. The rows indicate the different sets of potential intruders. For each set the matrix contains information how many intruders are in the specific set, how many intruders were missed by the user and how many false intruders were named. Of course the columns `missIntr` und `falseIntr` match if `numIntruder` is a scalar and the user names exactly this number of potential intruders for each set.

```
summary(intWords)
```

Applying `summary` to an object of type `intruderWords` will result in an ouput of some measures concerning the validation. Each function call contains ten sets. You are able to continue labelling by calling `intruderWords` with `oldResult = intWords` if your set was not finished.

```
intWords <- intruderWords(oldResult = intWords)
```

Analogously to `intruderWords` you can use `intruderTopics` for validation the other way around. This function is used for validation of topics associated to a specific document instead of validation of words associated to one topic. Therefore the document is displayed in another window and a sample of topics - represented by the ten `top.topic.words` - is shown in the console. You should commit in `text` the text component of the original untokenized corpus before manipulation by `cleanTexts`, so that the document is readable. The user then names the intruder(s). There are options for different numbers of topics and intruders as in `intruderWords` as well. The parameter `theta` should be set to `result$document_expects` where `result` is the LDA result. An example call is given below.

```r
intruderTopics(text = corpus$text, id = ldaID,
  beta = result$topics, theta = result$document_sums)
```

5.4 Clustering of Topics - `clusterTopics`, `mergeLDA`

For analysing topic similarities it is useful to cluster the topics. The function `clusterTopics` implements this. The main argument is `topics` and should be set to the `topics` element of the `result` object. You could specify `file`, `width` and `height` (both `integers`) to write the resulting plot to a pdf. Other options are `topicnames` for labelling the topics in the plot and `method` (default: `"average"`), which determines the way the topics are clustered. The `method` statement is used for applying the distance matrix to the function `hclust`. The distance matrix is computed based on the Hellinger distance and is returned in a list together with the value of the `hclust` call as invisible by `clusterTopics`.

```r
clustRes <- clusterTopics(ldaresult = result, xlab = "Topic", ylab = "Distance")
names(clustRes)
```

The same plot as above can be recreated by calling `plot(clustRes$cluster)`. In the plot you can see the similarities concerning the hellinger distance of the topics. For example the plot hints at a similarity between topic 6, 3 and 5 which all contain economic terms with topic 6 focussing more on economic policy and topic 5 concentrating on international politics and trade. Topic 2 and 8 both include words on national politics with topic 2 mentioning Australia, Ireland and the UK and topic 8 concentrating on U.S. politics. Topic 4 on war and conflict is close to topics 2 and 8.

It is possible to merge different results of LDAs by calling `mergeLDA(list(result1, result2, ..., resultN))`. The function `mergeLDA` binds the `topics` elements of the results by row and only considers words which appear in all results. As result you receive the `topics` matrix including all topics from the results.

5.5 Visualisation of Topics over Time - `plotTopic`

As extension of the highly flexible functions `plotScot` and `plotFreq` the package `tosca` offers another plotting function of the same type. The function `plotTopic` does something very similar to these two functions. It plots the counts or proportion of words allocated to different topics of a LDA result over time. The result object is committed in `ldaresult`, and the corresponding IDs of documents as a `character` vector in `ldaid`. In `object` the function expects a strictly tokenized `textmeta` object. You could set `select` for selecting topics by an `integer` vector. By default all topics are selected. Analoguously to `wnames` in `plotFreq` it is possible to set topic names with `tnames`. By default the index and the most representative word (`top.topic.words`) per topic are chosen as names. For further individualisation the function offers mostly the same options as `plotScot` and `plotFreq`.

Often it is useful to choose `curves = "smooth"` if you do not select topics, because there is a massive fluctuation of exact curves. However, it is important to have a look at the exact curves, because the smoothed curves are someway manipulated by the statement `smooth`, so the user is tempted to optimise the smoothing parameter for getting the curves he or she wants.

```r
plotTopic(object = corpusFiltered, ldaresult = result, ldaID = ldaID,
  rel = TRUE, curves = "smooth", smooth = 0.1, legend = "none", ylim = c(0, 0.7))
```

There is no difference of handing over an inflated corpus with documents which were not used for LDA. But the corpus must contain all documents of the LDA.

```r
plotTopic(object = corpusClean, ldaresult = result, ldaID = ldaID,
  select = c(3:4, 6, 8), rel = TRUE, curves = "both", smooth = 0.1, legend = "topleft")
```

5.6 Visualisation of Topic Share over Time - `plotArea`

The function `plotArea` offers possibilities to create so called area visualisation of topics over time. It requires the arguments `ldaresult`, `ldaid` and `meta` as introduced before. There are options `select`, `tnames`, `unit` and others. Additionally you can set `threshold` to a `numeric` value between 0 and 1, as a limit that a topics proportion has to surpass at least once to be plotted.

Because this seems to be interesting topics *T3.economy* (blue curve), *T6.economic policy* (green) and *T8.US politics* (red) are plotted in a sediment plot. The chosen `unit` is `"bimonth"` (default is `"quarter"`).

```r
plotArea(ldaresult = result, ldaID = ldaID, meta = corpusFiltered$meta,
  select = c(3, 6, 8), unit = "bimonth", sort = FALSE)
```

Examplary interpretation: The topic *T3.economy* increases over time, especially after the bank Lehman Brothers declared bankruptcy in September 2008, the starting point of the global financial crisis. *T8.US politics* is considerably larger during the 2008 presidential election campaign.

5.7 Visualisation of Words in Topic over Time - `plotTopicWord, plotWordpt`

Another visualisation of topics over time is given by `plotTopicword`. It displays the counts or proportions of given topic-word combinations. If `rel = TRUE` the baseline for normalisation are the words counts, not the counts of topics. Arguments which have to specified are `object` (corpus, `textmeta` object), `docs` (corpus manipulated by `LDAprep`, the input for `LDAgen`) and the `ldaresult` with its `ldaid` (IDs of documents in `docs` or `ldaresult` respectively). The function asks for `docs` for complexity reasons. This object was created by `LDAprep` anyway. The options `wordlist` and `select` are known from other plot functions and offer a lot of different topic word combinations which should be plotted by `plotTopicword`.

In the example corpus the proportion of the word *economy* in the topics one, three and seven is explored. The `top.topic.words` of the three chosen topics are *economy* (T3.economy, lightgreen curve), *million* (T6.economic policy, orange) and *obama* (T8.US politics, purple).

```r
plotTopicWord(object = corpusFiltered, docs = pagesLDA, ldaresult = result, ldaID = ldaID,
  wordlist = "economy", select = c(3, 6, 8), rel = TRUE, legend = "topleft")
```

The graphic shows that the word *economy* is associated to the topic *T3.economy, lightgreen curve* more often than with *T6.economic policy, orange curve* and *T8.US politics, purple curve.*

For interpretation it is important to keep in mind the baseline, the word counts of *economy.* To display this the sums of all topic-word proportions are calculated and are expected to be one for all dates which appear at least once, otherwise zero.

```r
tab <- plotTopicWord(corpusFiltered, pagesLDA, result, ldaID, "economy", rel = TRUE)
all(round(rowSums(tab[, -1]), 10) %in% c(1, 0))
```

This is confirmed by the call above. For some analysis maybe it could be interesting to take the other possible baseline, the topic counts, into account. For such tasks there is an additional function called `plotWordpt`.

The function `plotWordpt` works analogously like its pendant `plotTopicWord`, but with baseline topic sums instead of word sums. The difference between the functions `plotWordpt` and `plotTopicWord` is that `plotWordpt` considers topic peaks. You will get the relative counts of the selected word(s) in the selected topic(s). All curves sum up to one if you choose any topic and the whole vocabulary list as wordlist.

```r
plotWordpt(object = corpusFiltered, docs = pagesLDA, ldaresult = result, ldaID = ldaID,
  wordlist = "economy", select = c(3, 6, 8), rel = TRUE)
```

5.8 Visualisation of Words in Articles allocated to Topics - `plotWordSub`

To identify words which are used frequently in articles allocated to a topic one can use the function `plotWordSub`. The first problem is allocation of topics. Therefore you set an absolute or relative limit how often words of a given article are allocated to one topic. Additionally you have to specify whether one article is allocated exactly once, maximally once or multiple times depending on the `limit` argument. The default is `limit = 10` and `alloc = "multi"`, so an article is allocated to a topic if it contains at least 11 words which are allocated to the given topic. Multiple or no allocations are possible. After allocating the articles to the topics the function creates subcorpora using `filterWord`. To control the filter you have to set the `search` argument. The counts of the subcorpora (normalized to their whole corpora) are plotted. There are many options to personalize your plot like in the other plot functions.

```
searcheco <- data.frame(pattern = "economy", word = TRUE, count = 3)
plotWordSub(object = corpusFiltered, ldaresult = result, ldaID = ldaID, limit = 1/3,
  select = c(3, 6, 8), search = searcheco, unit = "quarter", legend = "topright")
```

The plot shows subcorpora generated by the `search` argument above, which means articles must contain the word *economy* at least three times. The corpora from which these subcorpora are generated have to contain one third of words which are allocated to the corresponding topic (`limit = 1/3`).

5.9 Heatmap of Topics over Time including Clustering - `plotHeat`

The use case for `plotHeat` is given by searching for explicit peaks of coverage of some topics. Therefore the resulting heatmap shows the deviation of the proportion of a given topic at this current time from its mean proportion. In addition a dendrogramm is plotted on the left side of the heatmap showing similarities of topics. The clustering is performed with `hclust` on the dissimilarities computed by `dist`.

By default the proportions are calculated on the article lengths, but it is possible to force calculation on only the LDA vocabulary by setting `object` to a `textmeta` object only including meta information. Otherwise a strictly tokenized `textmeta` object is required. The parameters `ldaresult` and `ldaID` expect a LDA result and according IDs like in functions mentioned before. Options `tnames` (topic label), `file` (if you want to save the plot in a pdf) and `unit` (default: round dates to `"year"`) are given as well. Additionally it is possible to specify whether the deviations should be normalised to take different topic sizes into account (default: `norm = FALSE`). You can change the intervals of labeling on the x-axis by setting `date_breaks`. By default (`date_breaks = 1`) every label is drawn. If you choose `date_breaks = 5` every fifth label will be drawn.

The increase of the topic *T3.economy* after September 2008 was mentioned before. This should be visible in the following heatmap as well. As compromise between clarity and interpretability `unit = "quarter"` is chosen.

```
plotHeat(object = corpusFiltered, ldaresult = result, ldaID = ldaID, unit = "quarter")
```

In this figure, as expected the *T3.economy* topic increase is clearly identifiable. The according rectangles are colored more and more red, starting from the first quarter of 2009. Almost all other quarters of years concerning this topic are colored lightblue. Other remarkable quarters are for example the third and fourth quarter of 2007, where the topic *T10.Canadian politics* and topic *T1.stopwords* have noticeable peaks. The dendrogramm shows that the topics are not very similar to another concerning the absolute deviations of topic proportion from the mean topic proportion per quarter. This supports the findings of clustering the topics with `clusterTopics`.

5.10 Individual Cases Contemplation - `topTexts`, `topicsInText`

Sometimes it is useful to look at individual cases. Especially the documents with the highest counts or proportion of words belonging to one topic are of interest. These documents can be extracted by `topTexts`. By default (`rel = TRUE`) the proportion is considered. The function requires a `ldaresult` and the according object `ldaid`. It offers options `select`, `limit` and `minlength`, which control how much articles per topic (default: all topics) are returned (default: `limit = 20`) and articles of which minimum length (default: `minlength = 30`) are taken into account. The output value is a matrix of the corresponding IDs.

In the example the top four pages from the topics *T8.US politics*, *T3.economy* and *T6.economic policy* are requested.

```r
topID <- topTexts(ldaresult = result, ldaID = ldaID, select = c(8, 3, 6), limit = 4)
dim(topID)
```

Obviously the corresponding matrix has four rows and three columns.

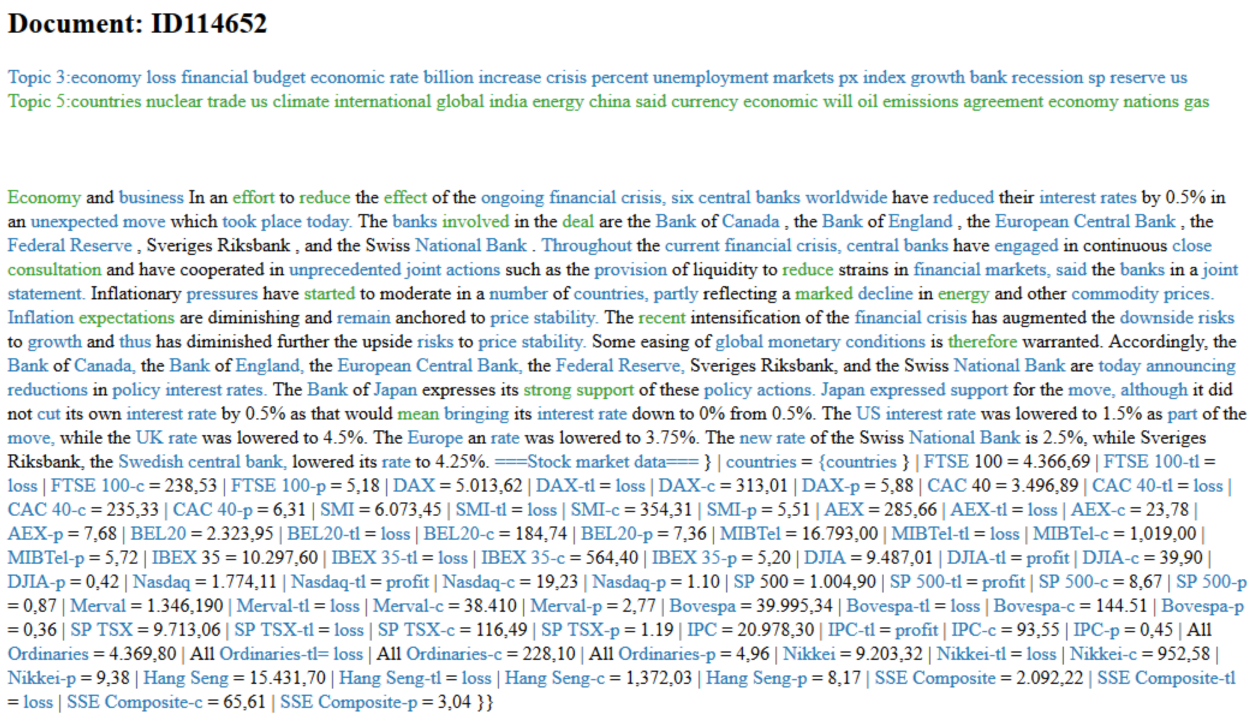
After identifying the top pages it is possible to have a closer look at these articles. Therefore the mentioned function `showTexts` can be used. The returned value is a list with three entries with `data.frames` of four rows - the different pages - and four columns each - *id*, *title*, *date* and *text*. For displaying, the fourth column of each `data.frame` containing the pages content itself is removed.

```r
topArt <- showTexts(corpusFiltered, id = topID)
lapply(topArt, function(x) x[, 1:3])
```

At last the function `topicsInText` offers the possibilty to analyse a single document's topic allocations. The function creates a HTML document with its words colored depending on the topic allocations of each word. It requires arguments `ldaresult` and `ldaID` as usual. The corresponding `LDAprep` object should be committed in `text`, and the vocabulary set as `character` vector in `words`. You will set `id` to the documents ID you are interested in. It is possible to show the original text by setting `originalText` to the corresponding uncleaned `text` component of your `textmeta` object. There are some more options - e.g. `wordOrder` - for modifying the output individually.

The article *Central banks worldwide cut interest rates* with ID *ID114652* from the top article list of topic *T3.economy* is analysed with the function `topicsInText` in more detail.

```r
topicsInText(text = pagesLDA, ldaresult = result, ldaID = ldaID,
  id = topArt$T3.economy[4,1], vocab = words5, originaltext = corpus$text, wordOrder = "")
```



Document: ID114652

Topic 3:economy loss financial budget economic rate billion increase crisis percent unemployment markets px index growth bank recession sp reserve us
Topic 5:countries nuclear trade us climate international global india energy china said currency economic will oil emissions agreement economy nations gas

Economy and business In an effort to reduce the effect of the ongoing financial crisis, six central banks worldwide have reduced their interest rates by 0.5% in an unexpected move which took place today. The banks involved in the deal are the Bank of Canada , the Bank of England , the European Central Bank , the Federal Reserve , Sveriges Riksbank , and the Swiss National Bank . Throughout the current financial crisis, central banks have engaged in continuous close consultation and have cooperated in unprecedented joint actions such as the provision of liquidity to reduce strains in financial markets, said the banks in a joint statement. Inflationary pressures have started to moderate in a number of countries, partly reflecting a marked decline in energy and other commodity prices. Inflation expectations are diminishing and remain anchored to price stability. The recent intensification of the financial crisis has augmented the downside risks to growth and thus has diminished further the upside risks to price stability. Some easing of global monetary conditions is therefore warranted. Accordingly, the Bank of Canada, the Bank of England, the European Central Bank, the Federal Reserve, Sveriges Riksbank, and the Swiss National Bank are today announcing reductions in policy interest rates. The Bank of Japan expresses its strong support of these policy actions. Japan expressed support for the move, although it did not cut its own interest rate by 0.5% as that would mean bringing its interest rate down to 0% from 0.5%. The US interest rate was lowered to 1.5% as part of the move, while the UK rate was lowered to 4.5%. The Europe an rate was lowered to 3.75%. The new rate of the Swiss National Bank is 2.5%, while Sveriges Riksbank, the Swedish central bank, lowered its rate to 4.25%. ===Stock market data=== } | countries = {countries } | FTSE 100 = 4.366,69 | FTSE 100-tl = loss | FTSE 100-c = 238,53 | FTSE 100-p = 5,18 | DAX = 5.013,62 | DAX-tl = loss | DAX-c = 313,01 | DAX-p = 5,88 | CAC 40 = 3.496,89 | CAC 40-tl = loss | CAC 40-c = 235,33 | CAC 40-p = 6,31 | SMI = 6.073,45 | SMI-tl = loss | SMI-c = 354,31 | SMI-p = 5,51 | AEX = 285,66 | AEX-tl = loss | AEX-c = 23,78 | AEX-p = 7,68 | BEL20 = 2.323,95 | BEL20-tl = loss | BEL20-c = 184,74 | BEL20-p = 7,36 | MIBTel = 16.793,00 | MIBTel-tl = loss | MIBTel-c = 1,019,00 | MIBTel-p = 5,72 | IBEX 35 = 10.297,60 | IBEX 35-tl = loss | IBEX 35-c = 564,40 | IBEX 35-p = 5,20 | DJIA = 9.487,01 | DJIA-tl = profit | DJIA-c = 39,90 | DJIA-p = 0,42 | Nasdaq = 1.774,11 | Nasdaq-tl = profit | Nasdaq-c = 19,23 | Nasdaq-p = 1.10 | SP 500 = 1.004,90 | SP 500-tl = profit | SP 500-c = 8,67 | SP 500-p = 0,87 | Merval = 1.346,190 | Merval-tl = loss | Merval-c = 38.410 | Merval-p = 2,77 | Bovespa = 39.995,34 | Bovespa-tl = loss | Bovespa-c = 144.51 | Bovespa-p = 0,36 | SP TSX = 9.713,06 | SP TSX-tl = loss | SP TSX-c = 116,49 | SP TSX-p = 1.19 | IPC = 20.978,30 | IPC-tl = profit | IPC-c = 93,55 | IPC-p = 0,45 | All Ordinaries = 4.369,80 | All Ordinaries-tl= loss | All Ordinaries-c = 228,10 | All Ordinaries-p = 4,96 | Nikkei = 9.203,32 | Nikkei-tl = loss | Nikkei-c = 952,58 | Nikkei-p = 9,38 | Hang Seng = 15.431,70 | Hang Seng-tl = loss | Hang Seng-c = 1,372,03 | Hang Seng-p = 8,17 | SSE Composite = 2.092,22 | SSE Composite-tl = loss | SSE Composite-c = 65,61 | SSE Composite-p = 3,04 }}

In the part of the HTML output above at first the different topics in the order of its absolute appearences in the given document are displayed. The topics are represented by its 20 `top.topic.words` each and are colored each in its own color. Words which were deleted by cleaning the corpus are colored black. This way you are able to check plausibility of individual documents, so `topicsInText` can also be seen as individual case validation.

6 Example pipeline

In this section we summarize the presented functions to a standard pipelines for datasets.

```r
library(tosca)
## load data
data(politics)
data(economy)
corpus <- mergeTextmeta(list(politics, economy))

## Remove XML-tags and HTML-entities in title and text
corpus$text <- removeXML(corpus$text)
corpus$text <- removeHTML(corpus$text, dec=FALSE, hex=FALSE, entity=FALSE)
corpus$meta$title <- removeXML(corpus$meta$title)
corpus$meta$title <- removeHTML(corpus$meta$title, dec=FALSE, hex=FALSE, entity=FALSE)

##looking for duplicates and first summaries
corpus <- deleteAndRenameDuplicates(corpus)
dups <- duplist(corpus)
plotScot(corpus, id = dups$notDuplicatedTexts, rel = TRUE)
print(corpus)
summary(corpus)
plotScot(corpus, curves = "both")

## corpus preprocessing / wordlists
corpusClean <- cleanTexts(object = corpus)
wordtable <- makeWordlist(corpusClean$text)
corpusDate <- filterDate(corpusClean, s.date = "2006-01-01", e.date = "2009-12-31")
searchterm <- list(
  data.frame(pattern = "economy", word = FALSE, count = 1),
  data.frame(pattern = c("world", "economy"), word = FALSE, count = 1),
  data.frame(pattern = "politics", word = FALSE, count = 1))
corpusFiltered <- filterWord(corpusDate, search = searchterm)

## prepare for LDA
wordtableFiltered <- makeWordlist(corpusFiltered$text, method = "radix")
words5 <- wordtableFiltered$words[wordtableFiltered$wordtable > 5]
pagesLDA <- LDAprep(text = corpusFiltered$text, vocab = words5)
LDAresult <- LDAgen(documents = pagesLDA, K = 10L, vocab = words5)
```

After generating the lda model further analysis depends on the specific aims of the project.

7 Conclusion

Our package tosca is an addition to the existing textmining packages on CRAN. It contains functions for a typical pipeline used for content analysis and uses the implementation of standard preprocessing of existing packages. Additionaly tosca provides functionality for visual exploration of corpora and topics resulting from the latent Dirichlet allocation. tosca focusses on analysis over time, so it needs texts with a date as meta data. The actual version of the package offers an implementation of intruder topics and intruder words (Chang et al., 2009). For future versions a framework for effective sampling in (sub-) corpora is under preparation. There are plans for a better connection to the frameworks of the tm and the quanteda package.