Package 'tmT'

author

December 7, 2017

Contents

1 Introduction

(TODO) how to install and so on, main target group...

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

2 Data Preprocessing

A basic functionality of the package is data preprocessing. Therefore several functions are given 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`, `readWiki`

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` should be a `data.frame` and `metamult` is intended for mainly unstructured meta-information as a `list`. Furthermore `meta` should contain 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 `tmT` is the function `readWiki`, which is based on functionality given by the package `WikipediR`. You can handover a `category` where all pages which are associated to this category are downloaded. By default `subcategories = TRUE` are downloaded as well. The source could be specified by changing the defaults `language = "en"` and `project = "wikipedia"` analogously to `pages_in_category` in the `WikipediR` package. In addition to downloading the function creates a `textmeta` object. This object needs some preprocessing after reading in as well as all text data.

The function `readWiki` was used to generate the corpus, which will be used as an example corpus for further functions.

```
journ = readWiki("Journalism")
bigdata = readWiki("Big data")
med = readWiki("Medicine")
corpus = mergeTextmeta(list(journ, bigdata, med))
```

The pages were downloaded on June 6, 2017 and merged to one corpus with the function `mergeTextmeta`. There are no complications expected merging the `meta data.frames`, because you will get always the same `meta` columns using `readWiki`, e.g. `date` is interpreted as the date when the page was added to a specific (sub)category. If consistency is not given the function will fill columns with `NAs` by default(`all = TRUE`) and will only return the columns which appears in all `data.frames` setting `all = FALSE`.

2.2 Remove Umlauts and XML Tags - `removeXML`

You can use `removeXML` to delete or manipulate some characters in the `text` component of your `textmeta` object. The argument which you handover in `x` should be a `character` vector or a `list` of `character` vectors of length one. The value you receive back will be a `character` vector. If you wish to use `removeXML` on a `list` of documents with length greater than one, you should use `lapply(object, removeXML, ...)`. The function deletes XML tags if you set `xml = TRUE` (default) and it manipulates umlauts - and some other special characters - if you set `umlauts = TRUE` (default: `FALSE`) and `u.type = c("normal", "html", "all")`, which replaces umlauts through its normal forms.

As some kind of source code of Wikipedia pages the example corpus contains arrow brackets, which are typical for XML structured data. These are removed by the function `removeXML`.

```
corpus$text = lapply(corpus$text, removeXML)
```

After applying the function to the corpus, it is some kind of "clean". There should nothing be in the text which has not to do with the article or page respectively itself. 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 on your own. If you cannot ensure that, it is recommended to use the function `deleteAndRenameDuplicates`, which deletes complete duplicates - which means, that there are at least two entries with same ID and same information in `text` *and* in `meta` - and renames so called "real duplicates" - at least two entries with same ID and text, but diferrent information in meta - and renames also "fake duplicates" - 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 simultaneously associated to a category and to one of its subcategories. These duplicates are deleted. Duplicates of texts where `id` equals, but `date` differs are renamed.

```
corpus = deleteAndRenameDuplicates(corpus)
```

```
## Delete Duplicates: 86  next Step
## Rename Real-Duplicates: 227  next Step
## Success
```

The function `deleteAndRenameDuplicates` deleted 86 complete duplicates and renamed 227 "real" duplicates, so that `duplist` is applicable to the corpus.

```
dups = duplist(corpus)
```

```
## ID-Fake-Dups: next Step
## ID-Real-Dups: next Step
## Unique Texts: next Step
## Same Texts: Success
## Lengths:
##        uniqueTexts notDuplicatedTexts         idFakeDups
##               3340               3229                  0
##         idRealDups        allTextDups       textOnlyDups
##                110                111                111
##       textMetaDups     textOthersDups
##                  0                  0
```

For further analysis, especially performing the Latent Dirichlet Allocation, it is important that for each duplicate only one page is considered. In the concrete example for each case the `id` is chosen for which its page is associated first to any of the (sub)categories. These IDs including the IDs of fully unique texts are saved in the variable `myUniques`.

```
earliestOfDups =
  sapply(dups$allTextDups,
```

```r
    function(x) names(which.min(sapply(x,
      function(y) corpus$meta$date[corpus$meta$id == y]))))
myUniques = c(dups$notDuplicatedTexts, earliestOfDups)
```

2.4 Clear Corpus - makeClear

There is a function `makeClear` for some further preprocessing of your text corpora. It removes punctuation, numbers and stopwords. By default it removes german stopwords as a extension of the function `stopwords("german")` from the `tm` package. You can control which stopwords should be removed with the argument `sw`. In addition the function changes all words to lowercase and tokenize 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 should represent paragraphs of a document. If you hand over a `textmeta` object instead of a `list` of texts you will receive one back.

The examples corpus language is english, so that `sw` should be set to `stopwords()` from the `tm` package, which includes english stopwords by default (`kind = "en"`). To create a new `textmeta` object the corresponding function is used.

```r
texts = corpus$text[names(corpus$text) %in% myUniques]
textClear = makeClear(text = texts, sw = tm::stopwords(), paragraph = FALSE)
```

```
## to lower
## punctuation
## numbers
## stopwords
## whitespace
## tokenization
## remove empty article
## Empty Articles: 0
```

```r
corpusClear = textmeta(text = textClear, meta = corpus$meta)
```

2.5 Generate Wordlist - makeWordlist

When you cleared your corpus with the function `makeClear` you are also able to call the function `makeWordlist`, which creates a table of occuring words in a given corpus. For complexity reasons you can handover a parameter `k` to the function (default: `100000L`). This parameter controls how much documents should be processed at once. Large `ks` lead to faster calculation but more RAM usage.

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

```r
wordtable = makeWordlist(corpusClear$text)
```

```
## 3340
## wordlist
## 0
## table
## 0
```

3 Descriptive Analysis

After preprocessing the text data there is a typical workflow we recommend of looking at the corpus. These workflow contains the generic functions `print` and `summary` as well as the highly adaptable, and following from this, powerful functions `plotScot` and `plotWord`. These graphical function should be part of every start of an analysis of text data.

3.1 Generic Functions - `print`, `summary`

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

```
print(corpus)
```

```
## Object of class "textmeta":
##  number of observations in text: 3458
##  meta: 3458 observations of 5 variables
##  range of date: 2004-07-25 till 2017-06-06
```

The function provides that the count of pages in the corpus is 3458 and there are two additional columns in `meta` to the necessary ones `id`, `date` and `title`. The pages are dated from July 25, 2004 to June 6, 2017.

You will get 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 handover the argument `list.names` (default: `names(object)`) for specifying the components out of `text`, `meta` and `metamult` which should be analysed by the function.

```
summary(corpus)
```

```
## number of observations in text: 3458
##
## NAs in text:
##  NA.abs NA.rel
##       0      0
## ----------------------------------------------------------------------
##
## meta: 3458 observations of 5 variables
##
## NAs in meta:
##              abs rel
## id             0   0
## date           0   0
## title          0   0
## categoryCall   0   0
## touched        0   0
## ----------------------------------------------------------------------
##
## range of date: 2004-07-25 till 2017-06-06
```

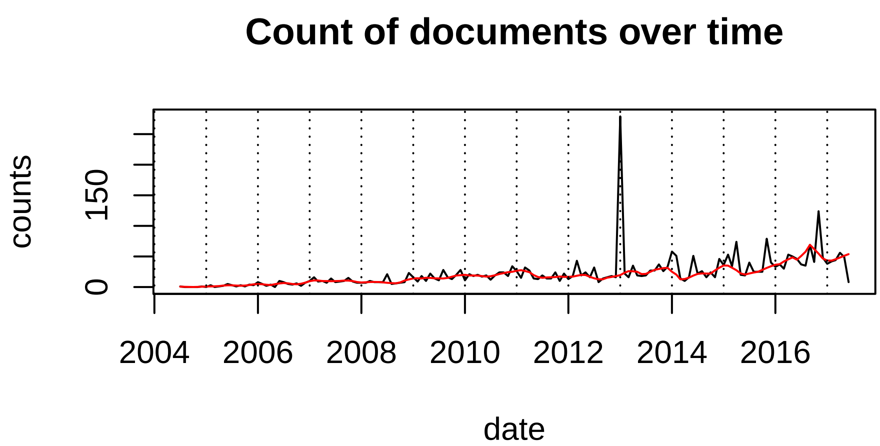
Apparently there are no `NA`s in the corpus as expected. The additional `meta` columns are `categoryCall` and `touched`, which are provided by `readWiki` always and contains the call of `category` in `readWiki` and the date of last revision.

3.2 Visualisation of Corpus over Time - `plotScot`

One of the descriptive plotting functions in the package is `ploScot` which creates a plot of counts or proportion 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"` (default: `type = "docs"`), the object which you handover should be a cleared `textmeta` object.

First of all the complete example corpus is be plotted, as exact and smoothed curve.

```
plotScot(corpus, curves = "both")
```



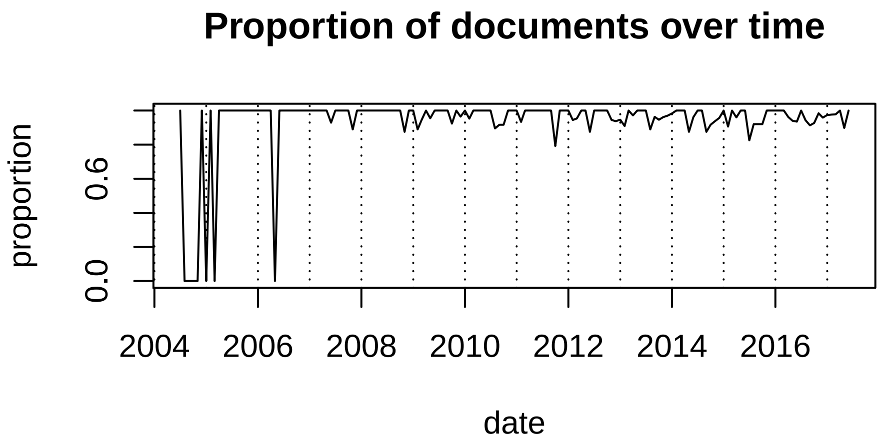
The black curve is the exact one and the red curve represents the smoothed values. As you can see there is a peak of assignments of pages to (sub)categories between December 2012 and January 2013. This is the result of 239 pages which were assigned on January 9, 2013 to one of the subcategories of the category "Medicine". The earliest pages of the subcorpus `bigdata` were assigned on September 9, 2013.

```
print(bigdata)
```

```
## Object of class "textmeta":
##  number of observations in text: 213
##  meta: 213 observations of 5 variables
##  range of date: 2013-09-09 till 2017-05-01
```

The following visualisation is one of the most important you should look at while analysing text data.

```
plotScot(corpus, id = myUniques, rel = TRUE)
```



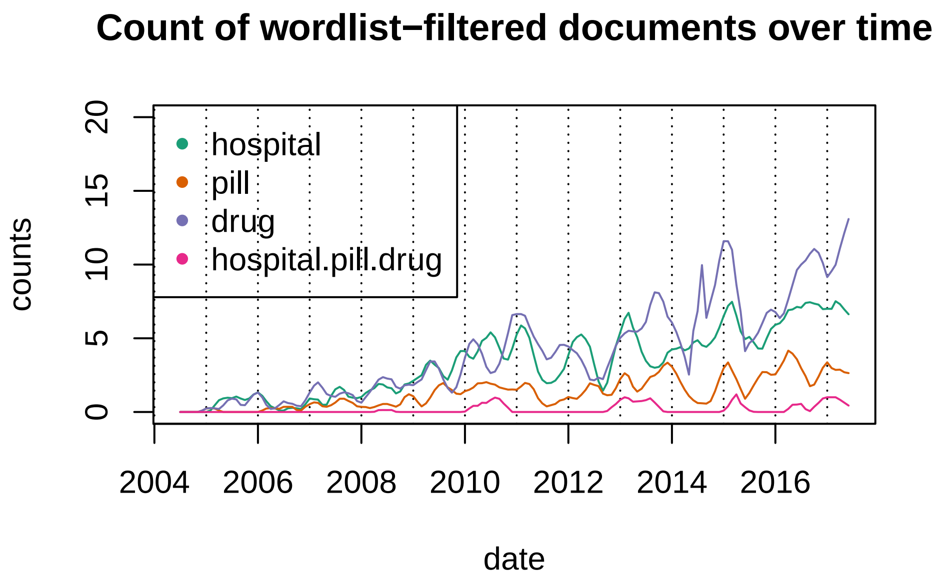
The proportion of the pages from these which IDs are in `myUnique` is very useful for identifying time effects concerning duplicates. In the wikipedia corpus there are some drops for example on October 2011 with 0.79 and on July 2015 with 0.83. The zeros from August to November of 2004, January and March of 2005 and May of 2006 results from no articles in the whole corpus during these time periods. It is possible to get these values as `NAs` by setting `natozero = FALSE`. This only has an effect if `rel = TRUE` and is offered by the functions `plotWord`, `plotTopic` and `plotTopicWord`, too.

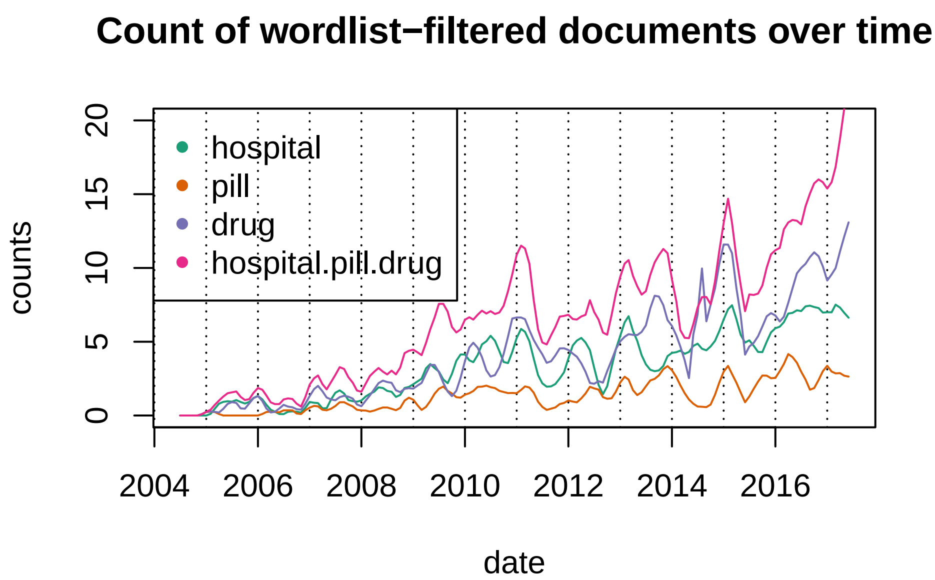
3.3 Frequency Analysis - `plotWord`

The other descriptive plotting function is `plotWord` which performs a frequency analysis. Most of the arguments does not differ from `plotScot`. But 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 vectors length. Each `list` entry represents a set of words which all (default `link = "and"`) or one of them (`link = "or"`) should appear in a article to be counted. The function uses `subcorpusWord` with `out = "count"`, which is explained later on, for counting.

The example corpus contains pages from Wikipedia concerning the categories (and their subcategories) "Journalism", "Medicine" and "Big data". Therefore some typical words out of these categories were taken to perform a frequency analysis. First of all the words *hospital*, *pill* and *drug* were taken.

```
wordsMed = list("hospital", "pill", "drug", c("hospital", "pill", "drug"))
plotWord(corpusClear, wordlist = wordsMed, curves = "smooth",
  ylim = c(0, 20), legend = "topleft")
plotWord(corpusClear, wordlist = wordsMed, link = "or", curves = "smooth",
  ylim = c(0, 20), legend = "topleft")
```

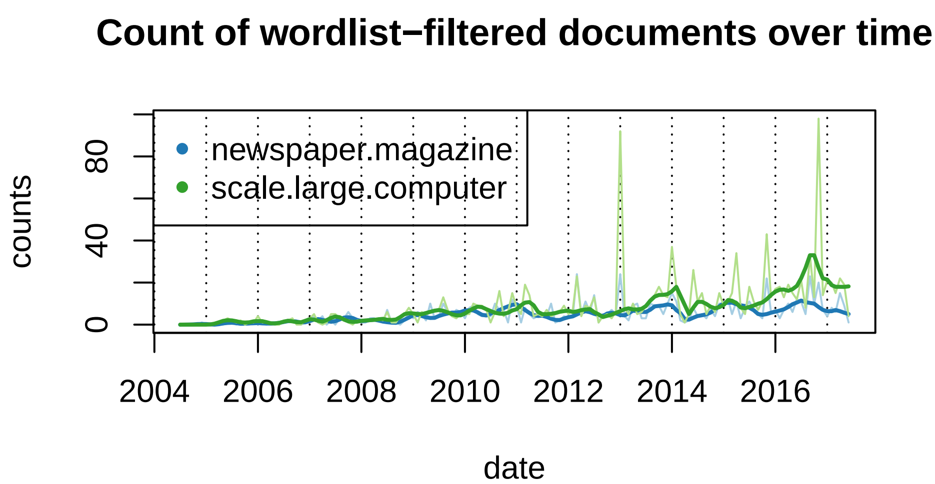




In the figures above you can see the difference between the *and* link and *or* link. The three curves indicating the single words rises to the end of observation time and seem proprtional to the observed pages over time. The word *pill* appears much less than the other two words and for this reason the curve of all three words together, which always has to lie beneath the lowest single curve if `link = "and"`, is zero on nearly every observation point. In the second figure the same single curves are shown. The fourth curve represents all three words again, but setting `link = "or"`. Off course the curve has to lie above all three single curves at every point. This is given except in one point which is caused by smoothing the curves. The curve lies above the three others in every point if you choose `curves = "exact"`. While the effect of the word *pill* is marginal for the cumulated curve it is remarkable that the intersect between pages, which contains the word *hospital* or *drug*, is not as high as one could expect.

In another figure the counts of pages in which the words *newspaper* and *magazine* or *scale*, *large* and *computer* respectively appears, are analysed.

```
wordsOthers = list(c("newspaper", "magazine"), c("scale", "large", "computer"))
plotWord(corpusClear, wordlist = wordsOthers, link = "or", curves = "both",
  both.lwd = 2, legend = "topleft")
```



With setting `curves = "both"` you can see the event on January 2013, like in the first figure of `plotScot`, in the light green curve. This shows that the event is not limited to pages from the category "Medicine". Therefore the event could have to do with some database structuring of Wikipedia at these dates or maybe the second curve which should specify words from the category "Big data" includes pages from the category "Medicine". The bigger and deeper colored curves represents the smoothed values. Especially the green curve increases for later dates. This fits to the expectation of "Big data" as a more modern category than the other ones, but nothing more. There is no chance of validation due to this figure.

The functions `plotScot` and `plotWord`, and usually all other plot functions in the package `tmT`, returns the table which is plotted as invisible output. Both functions offer a lot more functionality, which cannot be displayed in detail.

3.4 Write CSV Files - `showArticles`, `showMetadata`

There are two functions for writing csv files implemented in the `tmT` package. Any of both needs an any-formated `textmeta` object in `showArticles`, respectively the `meta` component of any-formated `textmeta` object in `showMetadata`, so you can even handover a `textmeta` object with tokenized documents to `showArticles`. The default of the parameter `id` in `showArticles` are all document IDs of the corpus as a `character` vector, but it is possible to handover a `character` matrix as well, so that each column will be represented in its own 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, whereas in the fourth column there will be the text itself.

Six IDs are sampled from the whole corpus. These pages are saved as `corpus1lesen.csv` and are returned as invisible to `temp`.

```
set.seed(123)
id = sample(corpus$meta$id, 6)
temp = showArticles(corpus, id = id)
temp[, c("id", "date", "title")]
```

```
##                  id       date                  title
## 1           51029165 2016-07-08       Imply Corporation
## 2 3310704_IDRealDup2 2013-01-09 Central venous pressure
## 3           19552284 2009-04-29   Immunization Alliance
## 4           19111093 2012-07-04       Chavutti Thirumal
## 5           54098843 2017-05-21       Caustic ingestion
## 6            4378291 2012-10-12             Kurt Schork
```

These six pages contain a "real" duplicate. You can identify them such fast looking at the column `id`. The six titles seems to lead to pages which are from different categories. Therefore you can have a look at the meta data.

The default of the parameter `id` in `showMetadata` are the IDs which are in the column `meta$id`. You can also handover a matrix of IDs like in `showArticles` and you can specify which columns of the `meta` component you want to be written in the csv by setting the argument `cols` (default: `colnames(meta)`).

Analogously to `showArticles` 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.

```
temp = showMetadata(corpus$meta, id = matrix(id, nrow = 2),
  cols = c("title", "categoryCall", "touched"))
temp
```

```
## $`1`
##                        title categoryCall    touched
## 1030       Imply Corporation     Big data 2017-05-15
## 2787 Central venous pressure     Medicine 2017-05-20
##
## $`2`
##                      title categoryCall    touched
## 1453 Immunization Alliance     Medicine 2017-05-30
## 3118     Chavutti Thirumal     Medicine 2017-03-23
##
## $`3`
##                  title categoryCall    touched
## 159        Kurt Schork   Journalism 2017-05-11
## 3321 Caustic ingestion     Medicine 2017-05-21
```

The function shows that the sampled pages are from all three category calls "Medicine", "Journalism" and "Big data". Five out of six pages were touched in the last month before downloading.

4 Further Preparation

The preprocessing which was done before is kind of mandatory. For further preparation the package offers functions for filtering the corpus by dates or some words to generate subcorpora. In addition a function for preparing your corpus for performing a Latent Dirichlet Allocation is given. This function creates a object which can be handed over to the function you could use for a LDA.

4.1 Filter Corpus by Dates - `subcorpusDate`

There are two implemented ways to filter your text corpora: One of these is the function `subcorpusDate`, which filters a given `textmeta` object by a time period. The function works on any formated objects of `class textmeta` and extracts documents out of the `text` component, from which the date column in the `meta` component is in between `s.date` and `e.date` - both including. The return value is the filtered `textmeta` object or a `list` which could be the `text` component of a `textmeta` object respectively, if you hand over the `text` and `meta` component not as a `textmeta` object.

The example corpus is filtered to pages which are associated first between 2011 and 2016.

```
corpusDate = subcorpusDate(corpusClear, s.date = "2011-01-01", e.date = "2016-12-31")
print(corpusDate)
```

```
## Object of class "textmeta":
##  number of observations in text: 2353
##  meta: 2353 observations of 5 variables
##  range of date: 2011-01-01 till 2016-12-29
```

The filtered corpus contains 2353 pages.

4.2 Filter Corpus by Words - `subcorpusWord`

The use of `subcorpusWord` works analogously. It filters the `text` component of a `textmeta` object by appearances of specific words. The function uses regular expressions and is very powerful by this. It filters the given documents in the `text` component by some words handed over by `search`, which could be a simple `character` vector or a `list` of `data.frames`. In the first case the entries of the vector are linked by an *or*, so *any* of the words must appear in one specific document for it to be returned. In the second case each `data.frame` is linked by an *or* and should contain columns `pattern` including the search terms, `word` and `count`. The column `word` is a logical variable which controls whether words (`TRUE`) or substrings are searched. Alternatively `word` can be a `character` string containing the keyword `left` or `right` for left- or right-truncated search. You must set the argument `count` to an `integer`. As you can imagine this argument controls how often a word or substring must appear in a document for it to be returned. Rows in each `data.frame` are linked by an *and*. Examples for the *or* or *and* link respectively are given by the next code example.

Maybe you are not interested in the texts of the documents itself. Therefore you can set `out` to control the output: By default (`out = text`) you get the filtered documents or if you hand over the argument `object` the corresponding `textmeta` respectively back. If you choose `out = bin` you get the corresponding logical vector of indices and if you choose `out = count` you get a matrix - with the number of documents rows and the `search`-length respectively vector-length columns - which indicates in row i and column j how often the j-th word of the `wordlist` appears in the according i-th document.

```
texts = list("schaafdung", "anything")
words = c("schaaf", "afd", "dung", "anything")
subcorpusWord(text = texts, search = words, out = "bin")
```

```
## [1] TRUE TRUE
```

The returned values are `TRUE` twice. There is at least one word in `words` which appears at least once in each of the strings *schaafdung* and *anything*.

```
wordframe = data.frame(pattern = words, word = FALSE, count = 1)
subcorpusWord(text = texts, search = wordframe, out = "bin")
```

```
## [1] FALSE FALSE
```

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

```
subcorpusWord(text = texts, search = wordframe[1:3,], out = "bin")
```

```
## [1]  TRUE FALSE
```

If you omit the word *anything* from `words` you receive a `TRUE` for *schaafdung* - all three words appears in it - and a `FALSE` for *anything*, because not all words appears in it, not even one of them.

An example of `out = count` is given by the following.

```r
subcorpusWord(text = list(c("i", "was", "here", "text"),
  c("some", "text", "about", "some", "text", "and", "something", "other")),
  search = c("some", "text"), out = "count")
```

```
##      some text
## [1,]    0    1
## [2,]    3    2
```

In the case of `out = count` it is useful, that `search` is a simple `character` vector. Another application of `subcorpusWord` is to apply the function with `word = TRUE`, so that the function searchs only for single words, not for strings containing these words. This is displayed by the following example.

```r
texts = list("land and and", c("and", "land", "and", "and"))
term = data.frame(pattern = "and", word = c(TRUE, FALSE), count = 1)
subcorpusWord(text = texts, search = split(term, term$word), out = "count")
```

```
##      and and_w
## [1,]   3     2
## [2,]   4     3
```

The function returns counts `c(3, 4)` for the simple search of substrings and `c(2, 3)` for the restricted word search, cause the word *and* appears once in every document of `texts` only as substring and not as own word.

The example corpus is filtered to those pages which fit to the chosen categories sofar, that the name of one of the catagories should appear on the page at least once, even in a substring.

```r
words = list(
  data.frame(pattern = "journalism", word = FALSE, count = 1),
  data.frame(pattern = c("big", "data"), word = FALSE, count = 1),
  data.frame(pattern = "medicine", word = FALSE, count = 1))
corpusFiltered = subcorpusWord(corpusDate, search = words)
print(corpusFiltered)
```

```
## Object of class "textmeta":
##  number of observations in text: 1460
##  meta: 1460 observations of 5 variables
##  range of date: 2011-01-01 till 2016-12-29
```

The date and word filtered corpus consists of 1460 pages.

4.3 Transform Corpus - `docLDA`

The next and last step before performing a Latent Dirichlet Allocation is to create corpus data, which could be handed over to the function `lda.collapsed.gibbs.sampler` from the `lda` package or the function `LDAstandard` from this package respectively. This is gained by using the function `docLDA` with its arguments `corpus` and `vocab` which expect a `text` component of a `textmeta` object and a `character` vector of vocabularies. These vocobularies are the words which are taken into account for LDA. The function offers options `ldacorrect`, `excludeNA` and `reduce` set all `TRUE` by default. The returned value is a `list` in which every entry symbolies a article and contains a matrix with two rows. In the first row there is the index of the word in `vocab` minus one, in the second row there is the number of appearances of the word in the article. The option `ldacorrect = TRUE` ensures the second row is always one and the number of the appearances of the word will be shown by the number of columns belonging to this word.

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

```
wordtableFiltered = makeWordlist(corpusFiltered$text)
```

```
## 1460
## wordlist
## 0
## table
## 0
```

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

```
##          -       edit retrieved   medicine    medical     health
##      15431      12706       9448       8825       8186       7713
```

The most often "word" which appears in the filtered corpora is the sign "-" with a count of 15431, so that it appears more than nine times on a page in mean. The second most often word is *edit*. Maybe this word and *retrieved* should be handled as a stopword, as well "-" as word is discussable. Nevertheless these words are also considered for analysis.

```
words5 = sort(names(sortedWords)[sortedWords > 5])
pagesLDA = docLDA(text = corpusFiltered$text, vocab = words5)
```

After receiving the words which appears at least six times in the whole filtered corpus, the function `docLDA` is applied to the example corpus with `vocab = words5`. The object `pagesLDA` will be handed over to the function which performs a Latent Dirichlet Allocation.

5 Latent Dirichlet Allocation

The main analytical functionality requested by text mining tools is to perform and analyse a Latent Dirichlet Allocation. In the package `tmT` this is ensured by the function `LDAstandard` for performing the LDA, functions for validating the LDA results and various functions for visualize the results in different ways, especially over time. It is possible to analyse individual articles and its topic allocations as well.

5.1 Performing LDA - `LDAstandard`

The function which has to be applied first to the corpus manipulated by `docLDA` is `LDAstandard`. Therefore the function offers the options `K` (`integer`, default: `K = 100`) to set the number of topics, `vocab` (`character` vector) for specifying the words which are considered in the manipulation 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 will be saved in a `R` workspace, the first part of the results name can be specified by setting the option `folder` (default: `folder = "lda-result"`).

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 consist of the `folder` argument followed by the options of `K`, `num.iterations`, `burnin` and the `seed` of the LDA.

```
LDAstandard(documents = pagesLDA, K = 10L, vocab = words5, seed = 123)
load("lda-result-k10i200b70s123.RData")
```

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

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

For validation of LDA results there are two functions in the package. These functions expect user input, the user works like a text labeller. The LDA result is handed over by setting `beta = result$topics`. During the function `intruderWords` the labeller gets a set of words. The number of words can be set by `numOutwords` (default: 5). These set represents one topic. It includes a number of intruders (default: `numIntruder = 1`), which can also be zero. In general, if the user identify the intruder(s) correctly this is a identifier for a good topic allocation. You can set options `numTopwords` (default: 30) to control which 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 a number of intruders of zero or one.

```r
set.seed(101)
intWords = intruderWords(beta = result$topics, numIntruder = 0:1)
```

```
## counter 1
## 1 computational
## 2 cell
## 3 cells
## 4 engineering
## 5 newspaper
```

By way of illustration the first set is shown. Obviously the word *newspaper* does not fit into the set with the words *computational*, *cell*, *cells* and *engineering*. Therefore the user would type `5` and press enter. If the user want to mark nonsense topics he would type `x` and `0` if he thinks there is no intruder word. Actually *newspaper* is the true intruder in the set above. As an example user input `c(5, 0, 0, 5, 1, 2, 0, 5, 3, 5)` is considered.

```r
print(intWords)
```

```
##   byScore numTopwords numIntruder numOutwords noTopic
## 1    TRUE          30         0 1           5    TRUE
##
##  Results:
##        numIntruder missIntruder falseIntruder
##  [1,]            0            0             0
##  [2,]            0            0             1
##  [3,]            0            0             0
##  [4,]            1            0             0
##  [5,]            0            0             1
##  [6,]            1            1             0
##  [7,]            1            0             0
##  [8,]            1            0             0
##  [9,]            1            0             0
## [10,]            1            0             0
```

By printing the object of `intruderWords` 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 much intruder are in the specific set, how much intruders were missed by the user and how much false intruders were named. Certainly the columns `missIntr` und `falseIntr` matchs if `numIntruder` is a scalar and the user names exactly this number of potential intruders for each set.

```r
summary(intWords)
```

```
## Non interpretable Topics: 0
## Non evaluated Topics: 0
##
##   byScore numTopwords numIntruder numOutwords noTopic
## 1    TRUE          30         0 1           5    TRUE
##
##  Number of meaningful topics: 10 out of 10 (100%)
##  Correct topics: 7 (70%)
##
##
##  Table of Intruders
## 0 1
## 4 6
##
##  Mean of number of missed Intruder: 0.1
##  Table of missing Intruders
## 0 1
## 9 1
##
##  Mean of number of false Intruder: 0.2
##  Table of false Intruders
## 0 1
## 8 2
```

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.

```r
intWords = intruderWords(oldResult = intWords)
```

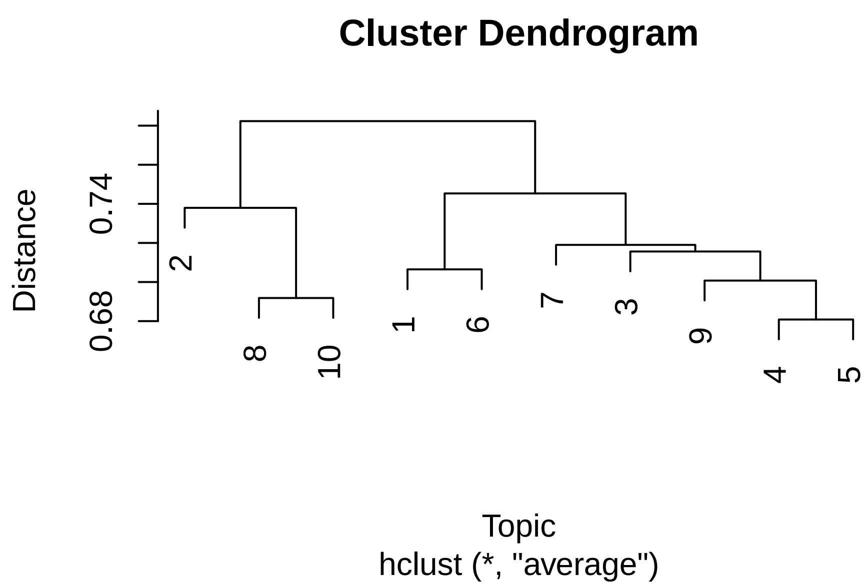
Analogously to `intruderWords` you can use `intruderTopics` for validation the other way around. This function is used for validation of topics associated to a specific document instead of validation of words associated to one topic. Therefore the document is displayed in another window and a sample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

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

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

For analysing topic similarities and looking for the right topic number for performing further LDA 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 your `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 influences the way the topics are clustered. The `method` statement is used for applying the distance matrix tothe 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")
```




```r
names(clustRes)
```

```
## [1] "dist"    "cluster"
```

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. Maybe it is of interest to look deeper at similar topics. In this example you could look at the topics 4 and 5.

```r
library(lda)
top.topic.words(result$topics[4:5, ], num.words = 6)
```

```
##      [,1]          [,2]
## [1,] "cells"       "may"
## [2,] "science"     "can"
## [3,] "engineering" "edit"
## [4,] "doi"         "s"
## [5,] "edit"        "mental"
## [6,] "research"    "also"
```

The top six representative words of the topics 4 and 5 do not look like they come from similar topics. This can be a side effect of the example corpus which is only a set of articles from three different categories in wikipedia, but ten topics were choosen for the LDA. There are also many words in the corpus which should be deleted in preprocessing like *edit* and one letter words.

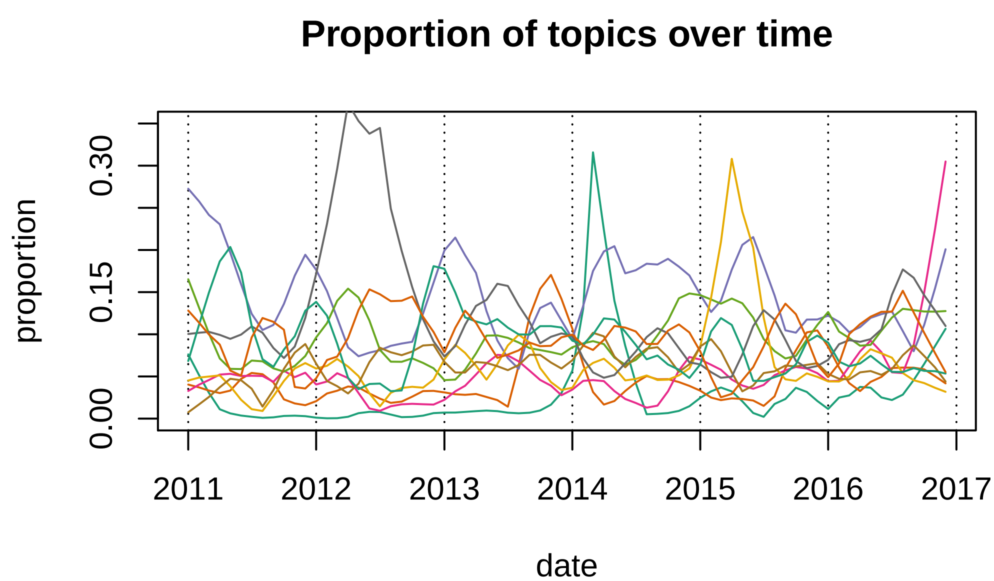
It is possible to merge different results of LDAs by calling `mergeLDA(list(result1, result2, ..., resultN))`. The function `mergeLDA` binds the `topics` elements of the results by row and only consider words which appears in all results. As result you receive the `topics` matrix including all topics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

5.4 Visualisation of Topics over Time - `plotTopic`

As extension of the highly flexible functions `plotScot` and `plotWord` the package `tmT` offers at least one more plotting function of the same type. The function `plotTopic` does something very similar to these two functions. It pictures the counts or proportion of words allocated to different topics of a LDA result over time. The result object is handed over in `ldaresult`, the belonging IDs of documents as a `character` vector in `ldaid`. In `object` the function expect a strictly tokenized `textmeta` object. You could set `select` for selecting topics by an `integer` vector. By default all topics are selected. Analoguesly to `wnames` in `plotWord` 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 like `plotScot` and `plotWord`.

Often it is useful to choose `curves = "smooth"` if you do not select topics, because there is a massive fluctuation of exact curves.

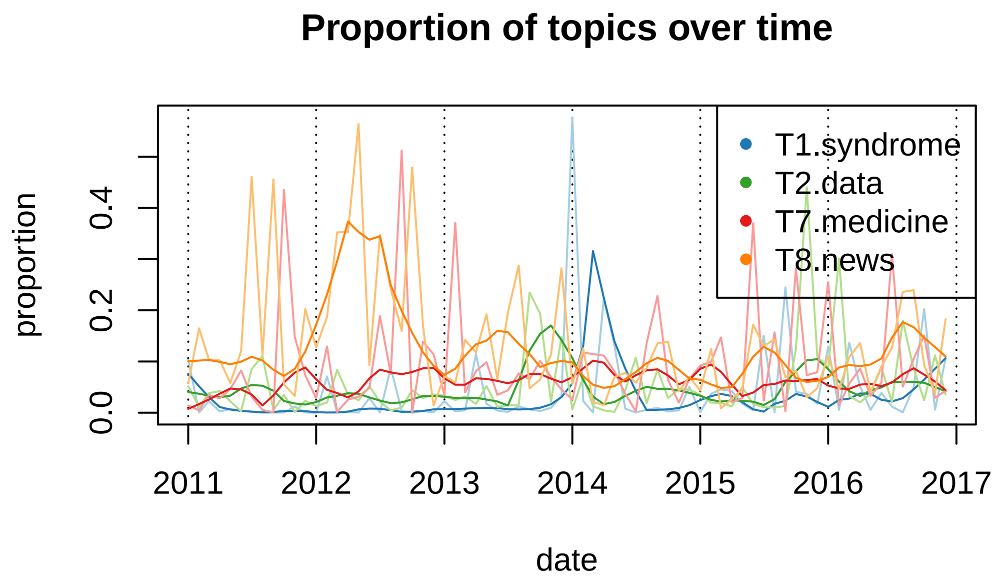
```
plotTopic(object = corpusFiltered, ldaresult = result, ldaID = ldaID,
  rel = TRUE, curves = "smooth", smooth = 0.1, legend = "none", ylim = c(0, 0.35))
```



There are not less than three topics with clear peaks at the first quarter of the years 2012, 2014 and 2015. At the beginning of year 2013 there is also a small peak of another topic. This is a good example for identifying periods of time where reporting could be less diversely. In the given example it is cumbersome trying to find reasons for the peaks.

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

```
plotTopic(object = corpusClear, ldaresult = result, ldaID = ldaID,
  select = c(1:2, 7:8), rel = TRUE, curves = "both", smooth = 0.1)
```



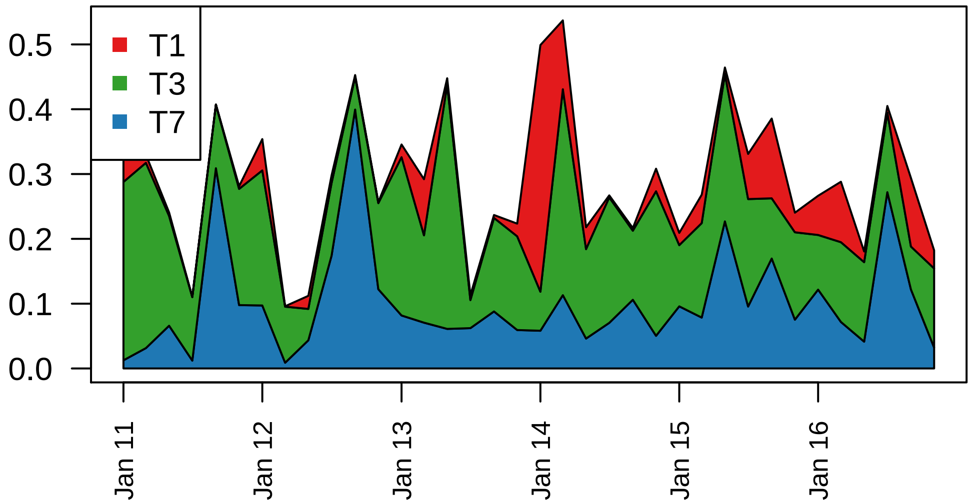
In the graphic above you could see, that the the topic *medicine* is nearly time independant. However the topic *news* has a peak on the first half of the year 2012, the topic *syndrome* on the first quarter of 2014. The topic *data* is less represented. Therefore the peak at the end of 2013 is considerable. The light colors displays the exact curves. Obvoiusly they alternate irrregular. 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.

5.5 Visualisation of Topic Share over Time - `sedimentPlot`

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

As this seems to be interesting topics *T1.syndrome* (red curve), *T3.health* (green) and *T7.medicine* (blue) are plotted in a sediment plot. The chosen `unit` is `"bimonth"`.

```
sedimentPlot(ldaresult = result, ldaID = ldaID, meta = corpusFiltered$meta,
  select = c(7, 3, 1), unit = "bimonth", sort = FALSE)
```



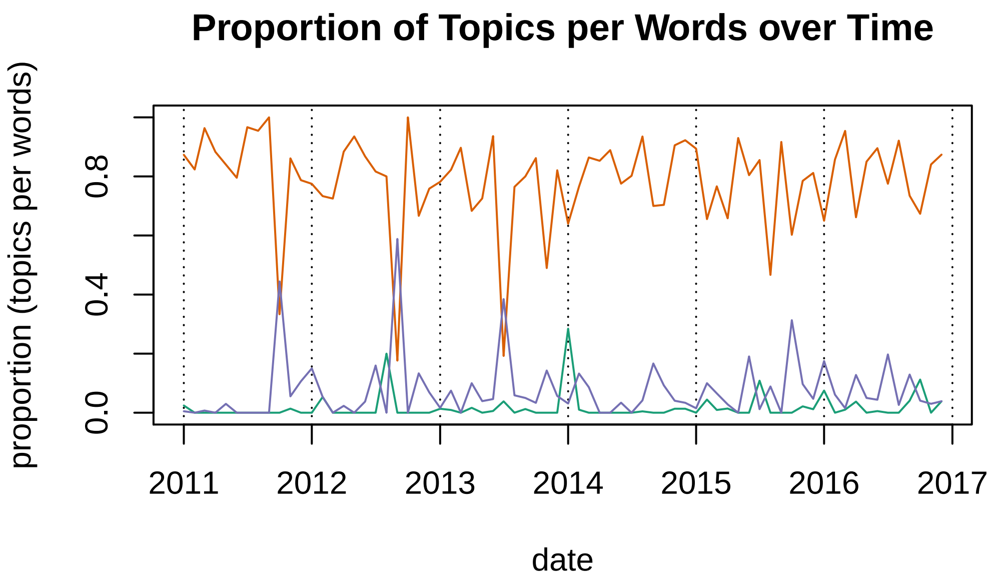
(TODO) Interpretation.

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

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

In the example corpus the proportion of the word *medical* in the topics one, three and seven is explored. The chosen word is the fifth most frequently word in the filtered corpus. The `top.topic.words` of the three chosen topics are *syndrome* (lightgreen curve), *health* (orange) and *medicine* (purple).

```
plotTopicWord(object = corpusFiltered, docs = pagesLDA, ldaresult = result, ldaID = ldaID,
  wordlist = "medical", select = c(1, 3, 7), rel = TRUE, legend = "none")
```



The graphic shows that the word *medical* is associated to the topic *health* most often. Drops of the orange curve goes with peaks of the purple curve (*medicine*). Aspects like this can be find easily with `plotTopicWord` and should lead to further analysis of the corresponding dates. The little peak of the topic allocations at January 2014 does not surprise because the topic itself has a massive peak at this date which you could see in the graphic before.

For interpretating it is important to keep in mind the baseline, the word counts of *medical*. To display this the sums of all topic-word proportions are calculated and are expected to be one for all dates.

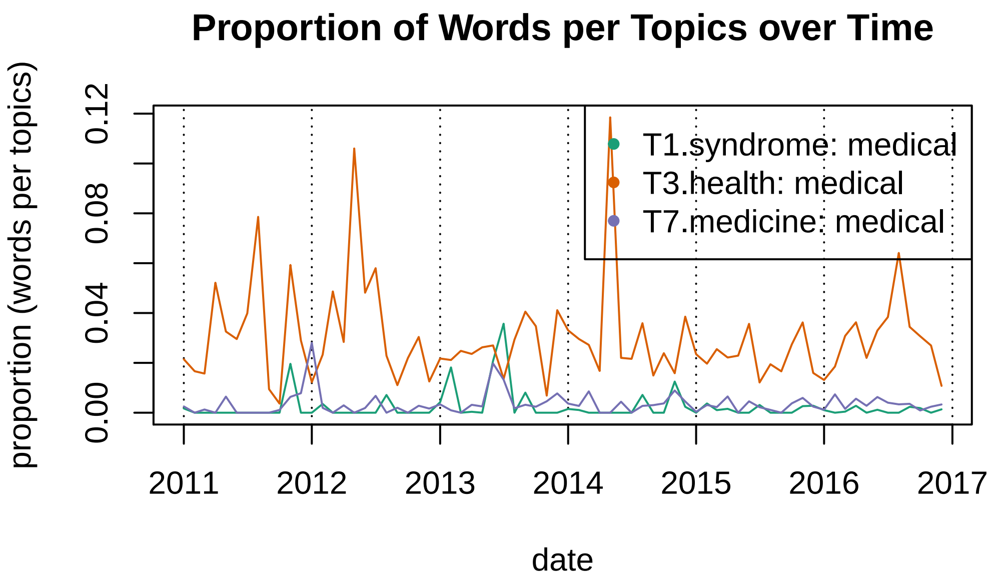
```
tab = plotTopicWord(corpusFiltered, pagesLDA, result, ldaID, "medical", rel = TRUE)
all(round(rowSums(tab[, -1]), 10) == 1)
```

```
## [1] TRUE
```

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. Therefore 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 both functions `plotWordpt` and `plotTopicWord` is given by the fact that `plotWordpt` considers topic peaks. You will get the relative counts of the selected word(s) in the selected topic(s). Obviously all curves sum up to one if you choose any topic and the whole vocabulary list as wordlist.

```
plotWordpt(object = corpusFiltered, docs = pagesLDA, ldaresult = result, ldaID = ldaID,
  wordlist = "medical", select = c(1, 3, 7), rel = TRUE)
```

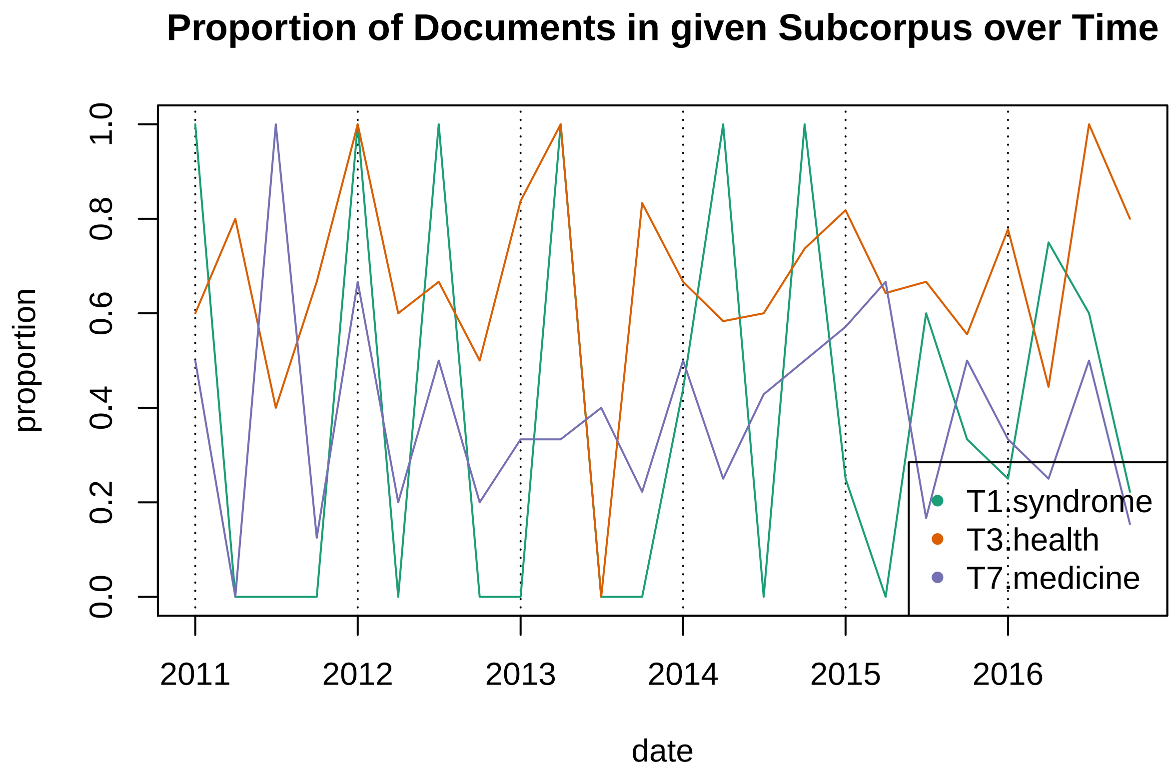


The plot indicates some peaks of the word *medical* in the topic *T3.health* especially in the middle of the years 2011, 2012 and 2014. The proportions of *medical* in the other two topics do not differ from zero often. In the end of 2011 and the middle of 2013 the topic proportion of the word increases in both topics, but do not reach five percent of the topic words in the given months.

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

Mention a problem where you want to identify words which are used frequently in articles allocated to a topic. The function which realizes a plot to the problem is called `plotWordSub`. The first problem is allocation of topics. Therefore you set a 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, maximum 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 `subcorpusWord`. 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.

```
searchmed = data.frame(pattern = "medical", word = TRUE, count = 4)
plotWordSub(object = corpusFiltered, ldaresult = result, ldaID = ldaID, limit = 1/3,
  select = c(1, 3, 7), search = searchmed, unit = "quarter", legend = "bottomright")
```



The plot shows subcorpora generated by the `search` argument above, which means articles must contain the word *medical* at least four 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`). The curves of the topics *T1.syndrome*, *T3.health* and *T7.medicine* differ lot from one to another quarter. This could be a problem of this example corpus, because the publication dates appears in groups, so that there are many quarters where no articles were published. This effect can be seen in the green curve best. The orange and purple curves seems to be interpretable, but time effects are difficult to determine from this plot. The function `subcorpusWord` could be used as a indicator how journalism reports in a given topic in dependence to the time.

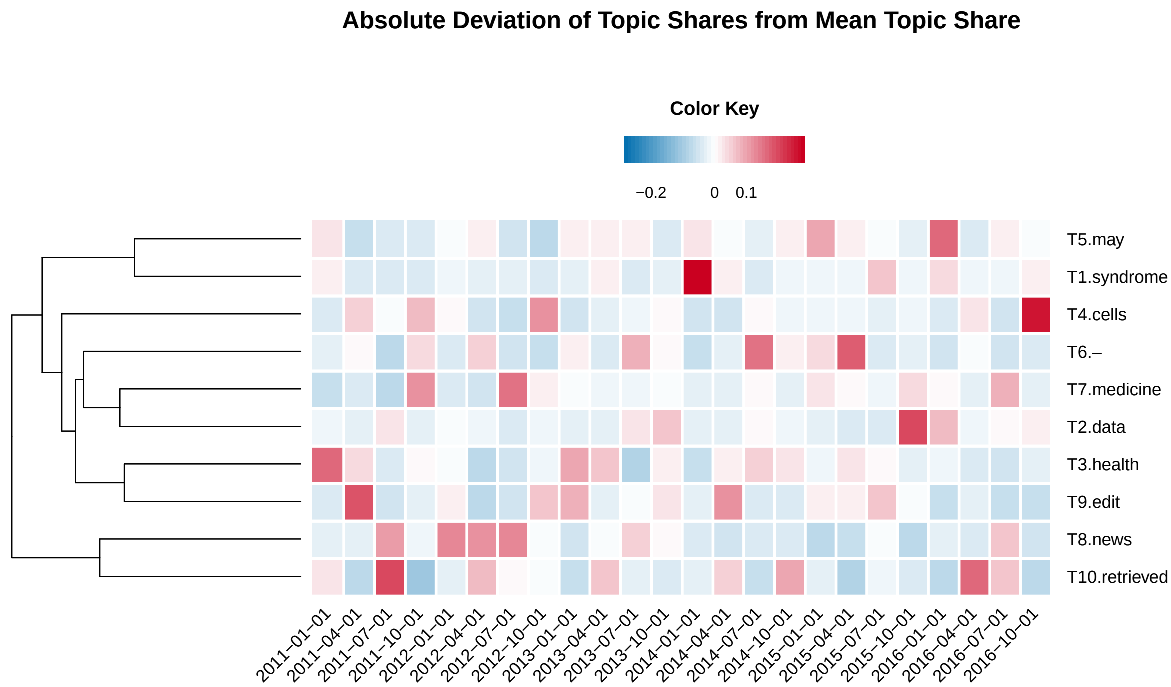
5.8 Heatmap of Topics over Time including Clustering - `totHeat`

Beside the plot functions using `Base R` there is one function using functionality from `gplots`. The function is called `totHeat`. Maybe there will be an opportunity to implement this function in `Base R` in later updates. The use case for `totHeat` 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 set 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 peak of the topic *T1.syndrome* in January 2014 was mentioned several times before. This should be visible in the following heatmap as well. As compromise between clarity and interpretability `unit = "quarter"` is chosen.

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



As expected the *T1.syndrome* topics peak is clearly identifiable. The according rectangle at the first quarter of 2014 is colored by the deepest red of this figure. On the other hand mostly all other quarters of years concerning this topic are colored lightblue. Other remarkable quarters are for example the fourth quarter of 2015 or 2016, where the topic *T2.data* or *T4.cells* respectively has noticeable peaks. The dendrogramm shows that none of the topics are similar to another concerning the absolute deviations of topic proportion from the mean topic proportion per quarter. This approves the findings of clustering the topics with `clusterTopics`.

5.9 Individual Cases Contemplation - `topArticles`, `topicsInText`

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

In the example the top four pages from the topics *T1.syndrome*, *T3.health* and *T7.medicine* are requested.

```
topID = topArticles(ldaresult = result, ldaID = ldaID, select = c(1, 3, 7), limit = 4)
dim(topID)
```

```
## [1] 4 3
```

Obviously the corresponding matrix has four rows and three columns.

After identifying the top pages it is possible to have a deeper look at these articles. Therefore the mentioned function `showArticles` 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.

```
topArt = showArticles(corpusFiltered, id = topID)
lapply(topArt, function(x) x[, 1:3])
```

```
## $`1`
##         id                                   title       date
## 1 29430760     Mild androgen insensitivity syndrome 2014-01-25
## 2 35934978      Inborn errors of steroid metabolism 2014-01-25
## 3 21631187                     Aromatase deficiency 2014-01-25
## 4   975417 Pseudovaginal perineoscrotal hypospadias 2016-10-05
##
## $`2`
##         id                             title       date
## 1 42515874 Clinical documentation improvement 2014-11-21
## 2 44327960  Conflict and Catastrophe Medicine 2014-11-09
## 3  5556168       European Practice Assessment 2013-01-09
## 4  1656748                    Family medicine 2011-03-08
##
## $`3`
##         id               title       date
## 1  9179093 List of kampo herbs 2012-06-02
## 2 38403582  Navajo ethnobotany 2013-02-02
## 3  4552570   Oriental medicine 2013-01-28
## 4 38464926    Zuni ethnobotany 2013-02-08
```

The top three pages from topic *T1.syndrome* were associated to their category on January 25th 2014. Once again this shows the peak of associations of this topic in January 2014.

At last the function `topicsInText` offers the possibilty to analyse a single documents 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 belonging `docLDA` object should be handed over in `text`, while 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 origing text by setting `originalText` to the belonging uncleared `text` component of your `textmeta` object. There are some more options - e.g. `wordOrder` - for modifying the output individually.

The article *Family medicine* with ID *1656748* from topic *T3.health* and category *Medicine* is analysed with the function `topicsInText` in more detail.

```
topicsInText(text = pagesLDA, ldaresult = result, ldaID = ldaID,
  id = topArt$`2`[4,1], vocab = words5, originaltext = corpus$text, wordOrder = "")
```

Document: 1656748

Topic 3:medical health medicine care patient clinical hospital healthcare surgery physicians education united public patients research emergency training insurance hospitals services
Topic 10:retrieved april show october original december news archived panama stewart said september daily november papers august january colbert february july
Topic 5:mental social can pp– interview vol m behavior psychiatry research often psychology imagery psychiatric questions example isbn disorder p pp
Topic 6:– death pmid doi mortality m disease j cell patients vaccine pmc al clinical diseases weekend antibiotics burial necrosis
Topic 9:blood imaging ultrasound tissue can surgery heart pain implant bone dental used medical implants device cardiac body pressure tube temperature

Family medicine (FM), formerly family practice (FP), is a specialty devoted to comprehensive health care for people of all ages; the specialist is named a family physician or family doctor . In Europe the discipline is often referred to as general practice and a practitioner as a General Practice Doctor or GP ; this name emphasises the holistic nature of this speciality, as well as its roots in the family. It is a division of primary care that provides continuing and comprehensive health care for the individual and family across all ages, genders, diseases , and parts of the body; [1] family physicians are often primary care physicians . It is based on knowledge of the patient in the context of the family and the community, emphasizing disease prevention and health promotion . [2] According to the World Organization of Family Doctors (WONCA), the aim of family medicine is to provide personal, comprehensive, and continuing care for the individual in the context of the family and the community. [3] The issues of values underlying this practice are usually known as primary care ethics . Contents 1 Scope of practices 2 Family medicine in Canada 3 Family medicine in the United States 3.1 History of Medical Family Practice 3.2 Education and training 3.3 Shortage of family physicians 3.4 Current practice 4 Family medicine in India 5 See also 6

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 clearing the corpus are colored black. The topic *T3.health* occurs very frequently. Only two words in the extract are not allocated to this topic. The word *named* belongs to the topic *T10.retrieved*, the word *underlying* to the topic *T5.may*. This way you are able to check plausibility of individual documents, so `topicsInText` can be seen as individual cases validation as well.

6 Conclusion

(TODO) Wichtigste Punkte des Workflows zusammenfassen, allgemeine Fallstricke (Duplikate . . .), Diskussion des Anwendungsbeispiels (viele stopwords nicht geloescht . . .), Ausblick? (weitere Funktionen . . .)