Package 'tmT'

author

March 19, 2018

Contents

1 Introduction **2**

2 Data Preprocessing **2**
2.1 Read the Corpus - `textmeta`, `readWikinews` . . . 2
2.2 Remove Umlauts and XML/HTML Tags - `removeXML removeHTML removeUmlauts` . . . 4
2.3 Identifying Duplicates - `deleteAndRenameDuplicates`, `duplist` . . . 4
2.4 Clear Corpus - `makeClear` . . . 5
2.5 Generate Wordlist - `makeWordlist` . . . 6

3 Descriptive Analysis **6**
3.1 Generic Functions - `print`, `summary` . . . 6
3.2 Visualisation of Corpus over Time - `plotScot` . . . 7
3.3 Frequency Analysis - `plotFreq` . . . 8
3.4 Write CSV Files - `showTexts`, `showMeta` . . . 11

4 Generating Subcorpora **12**
4.1 Filter Corpus by Dates - `filterDate` . . . 12
4.2 Filter Corpus by Wordcount - `filterCount` . . . 12
4.3 Filter Corpus by Words - `filterWord` . . . 12

5 Latent Dirichlet Allocation **14**
5.1 Transform Corpus - `LDAprep` . . . 14
5.2 Performing LDA - `LDAgen` . . . 15
5.3 Validation of LDA Results - `intruderWords`, `intruderTopics` . . . 16
5.4 Clustering of Topics - `clusterTopics`, `mergeLDA` . . . 17
5.5 Visualisation of Topics over Time - `plotTopic` . . . 18
5.6 Visualisation of Topic Share over Time - `plotArea` . . . 19
5.7 Visualisation of Words in Topic over Time - `plotTopicWord, plotWordpt` . . . 20
5.8 Visualisation of Words in Articles allocated to Topics - `plotWordSub` . . . 22
5.9 Heatmap of Topics over Time including Clustering - `plotHeat` . . . 23
5.10 Individual Cases Contemplation - `topTexts`, `topicsInText` . . . 25

6 Conclusion **26**

1 Introduction

(TODO) how to install and so on, main target group… textmeta objekte sind grundlage des pakets; was bei eigenen texten; was wenn restriktionen nicht erfüllt von textmeta objekt, standard-Datum erklären; readTextmeta erklären; hilfeseiten anpreisen…

```
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`, `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 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 `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()
```

```
## Wikinews-20180305075910_politics.xml
## Wikinews-20180305080159_economy.xml
## number of observations in text: 7327
##
## NAs in text:
##  NA.abs NA.rel
##       0      0
## ---------------------------------------------------------------------
##
## meta: 7327 observations of 3 variables
##
## NAs in meta:
##       abs  rel
## id      0 0.00
## date  195 0.03
## title   0 0.00
## ---------------------------------------------------------------------
##
## range of date: 2004-11-13 till 2018-03-04
## NAs in date: 195 (0.03)
```

Another method to read both files is to read both files seperately and merge them with the function `mergeTextmeta`. This function should be used if we want to merge data from different sources using different read-functions.

```
politics <- readWikinews(file="Wikinews-20180305075910_politics.xml")
```

```
## Wikinews-20180305075910_politics.xml
## number of observations in text: 5000
##
## NAs in text:
##  NA.abs NA.rel
##       0      0
## ------------------------------------------------------------------------
##
## meta: 5000 observations of 3 variables
##
## NAs in meta:
##       abs  rel
## id      0 0.00
## date   67 0.01
## title   0 0.00
## ------------------------------------------------------------------------
##
## range of date: 2004-11-13 till 2009-11-19
## NAs in date: 67 (0.01)
```

```
economy <- readWikinews(file="Wikinews-20180305080159_economy.xml")
```

```
## Wikinews-20180305080159_economy.xml
## number of observations in text: 2327
##
## NAs in text:
##  NA.abs NA.rel
##       0      0
## ------------------------------------------------------------------------
##
## meta: 2327 observations of 3 variables
##
## NAs in meta:
##       abs  rel
## id      0 0.00
## date  128 0.06
## title   0 0.00
## ------------------------------------------------------------------------
##
## range of date: 2004-11-17 till 2018-03-04
## NAs in date: 128 (0.06)
```

```
corpus2 <- mergeTextmeta(list(politics, economy))
```

```
## NOTE: There are duplicates in the names of texts, could result in problems with unambiguity.
```

We get a note about duplicated texts (text 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 will be used for the merged corpora (`all = TRUE`, default) or only variables that appear in all corpora (`all = FALSE`).

After reading the raw data the text needs some preprocessing.

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 will be 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`: ø, `hex=TRUE`: ø or `entity=TRUE`: ø). 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 all of them.

```
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)
```

```
## delete remaining entities
```

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 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. TODO:[Genauer was umbenannt wird, tabelle mit beschreibung] 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.

```
corpus <- deleteAndRenameDuplicates(corpus)
```

```
## Delete "Complete Duplicates": 286  next Step
## Success
```

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

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

```
## ID-Fake-Dups... next Step
## ID-Real-Dups... next Step
## Unique (and Not-Duplicated) Texts... next Step
```

```
## Same Texts... Success

## Duplist, List of (Lists of) IDs with Names:
##  "uniqueTexts", "notDuplicatedTexts", "idFakeDups", "idRealDups",
##  "allTextDups", "textOnlyDups", "textMetaDups", "textOthersDups".
## Calculate Numbers of IDs and Texts...
##  Number of Unique Texts: 6375
##  Number of Not-Duplicated Texts: 6361
##  Number of Fake-Dup IDs: 0
##   Number of Texts with Fake-Dup IDs: 0
##  Number of Real-Dup IDs: 0
##   Number of Texts with Real-Dup IDs: 0
##  Number of different Text-Dups: 14
##   Number of all Text-Dups: 680
##    Number of different Text-Dups with identical Meta (except ID): 0
##     Number of all Text-Meta-Dups: 0
##    Number of Text-Dups which do not pass criteria above: 0
```

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

For further analysis, especially 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 small relicts like `"__NOTOC__"` or `"* * *"`

2.4 Clear Corpus - `makeClear`

For further preprocessing of text corpora tosca offers the function `makeClear`. 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 are some additional words 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 should represent paragraphs of a document. If you hand over a `textmeta` object instead of a `list` of texts you will receive one back. In this case you have to hand it over to the parameter `object` instead of `text`.

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"`).

```
corpusClear <- makeClear(object = corpus)
```

```
## Check Articles on UTF8: next Step
## Change to Lowercase: next Step
## Remove Punctuation: next Step
## Remove Numbers: next Step
## Remove Stopwords: next Step
## Remove redundant Whitespace: next Step
## Tokenize: next Step
## Remove empty Articles: 654 Success
```

The function `makeClear` deletes all `meta` entries which did 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.

```
textClear2 = makeClear(text = corpus$text)
```

```
## Check Articles on UTF8: next Step
## Change to Lowercase: next Step
## Remove Punctuation: next Step
## Remove Numbers: next Step
## Remove Stopwords: next Step
## Remove redundant Whitespace: next Step
## Tokenize: next Step
## Remove empty Articles: 654 Success
```

```
corpusClear2 = textmeta(text = textClear2, meta = corpus$meta)
```

2.5 Generate Wordlist - `makeWordlist`

After clearing the corpus with the function `makeClear` we are able to call the function `makeWordlist`, which creates a table of occuring words in a given corpus. THe function `table` needs much RAM. That's a problem for bis Corpora. In `makeWordlist` we use the parameter `k` (default: `100000L`) to reduce the number of texts which are processed at once. Large `k`s 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.

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

```
## Number of Articles: 6387

## Find out Vocabularies...
##  Done:

##   0

##   6387 next Step

## Calculate Counts...
##  Done:

##   0

##   6387 Success
```

3 Descriptive Analysis

After preprocessing the text data there is a typical workflow we recommend for looking at 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`.

```
print(corpus)
```

```
## Object of class "textmeta":
##  number of observations in text: 7041
##  meta: 7041 observations of 3 variables
##  range of date: 2004-11-13 till 2018-03-04
```

```
##  NAs in date: 191 (0.03)
```

The function provides the count of pages in the corpus (7041) and there are 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 will get more information, especially about counts of `NAs` 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.

```r
summary(corpus)
```

```
## number of observations in text: 7041
##
## NAs in text:
##  NA.abs NA.rel
##       0      0
## ----------------------------------------------------------------------
##
## meta: 7041 observations of 3 variables
##
## NAs in meta:
##       abs  rel
## id      0 0.00
## date  191 0.03
## title   0 0.00
## ----------------------------------------------------------------------
##
## range of date: 2004-11-13 till 2018-03-04
## NAs in date: 191 (0.03)
```

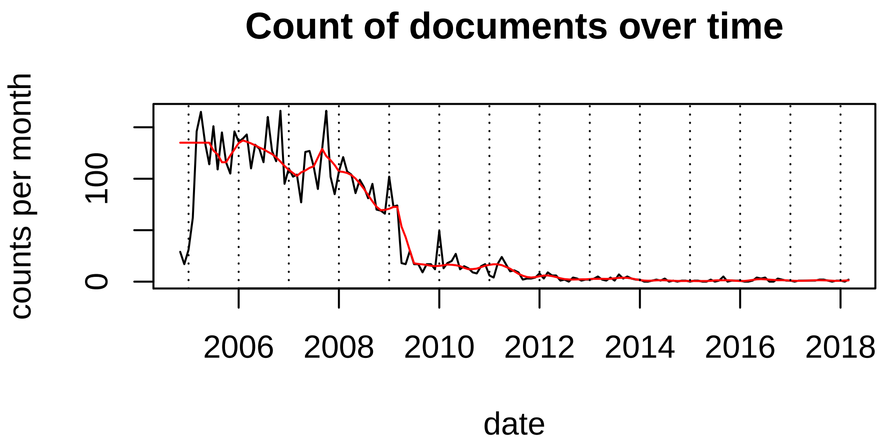
Apparently there are 191 `NAs` in the variable `date`.

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 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"`, the object which you handover should be a tokenized `textmeta` object. If `type = "docs"` (default) you can hand over untokenized `textmeta` object as well.

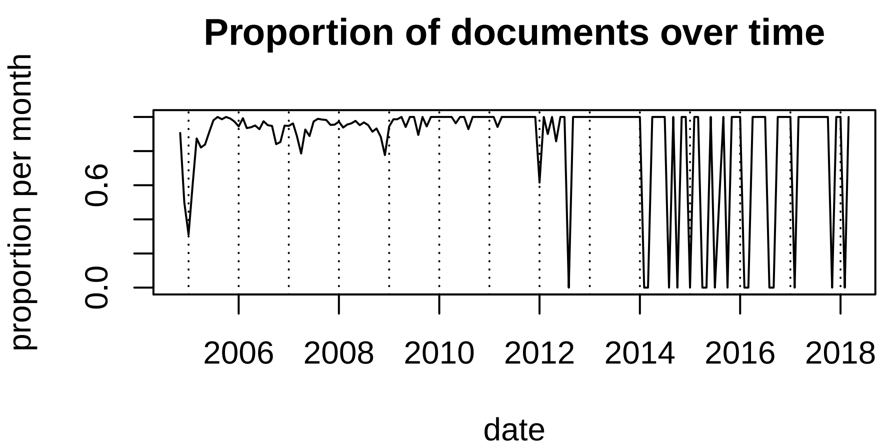
First of all the number of texts per month in the complete example corpus is plotted, as exact and smoothed curve.

```r
plotScot(corpusClear, 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 we want to identify the distribution of duplicates over the time we can use `plotScot` to plot the IDs of the not duplicated texts in the corpus.

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



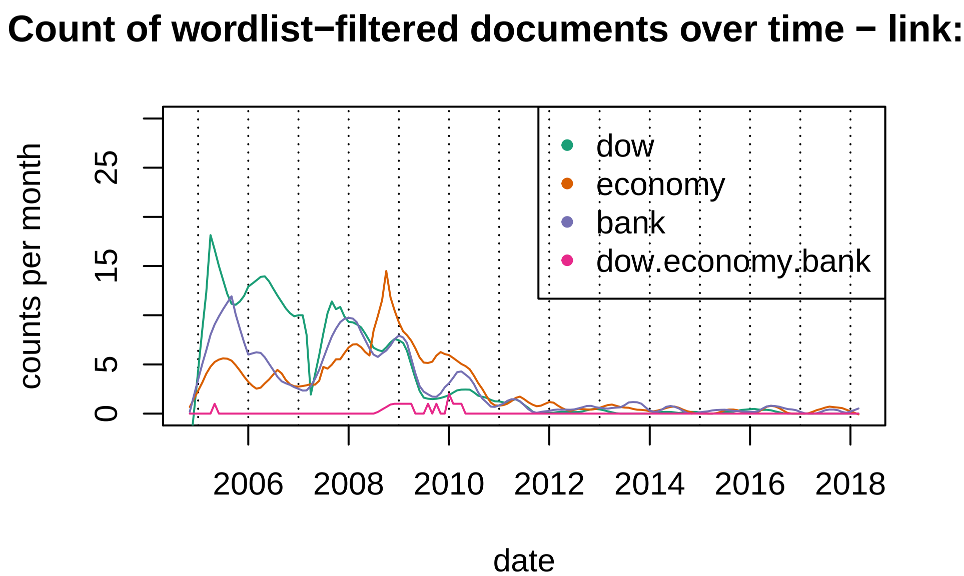
Most zeros in the plot result from no articles in the whole corpus during these time periods. It is possible to get these values as `NAs` by setting `natozero = FALSE` in `plotScot`. This option works if `rel = TRUE` and is offered by many other functions in the package. Usually all plot functions in 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.

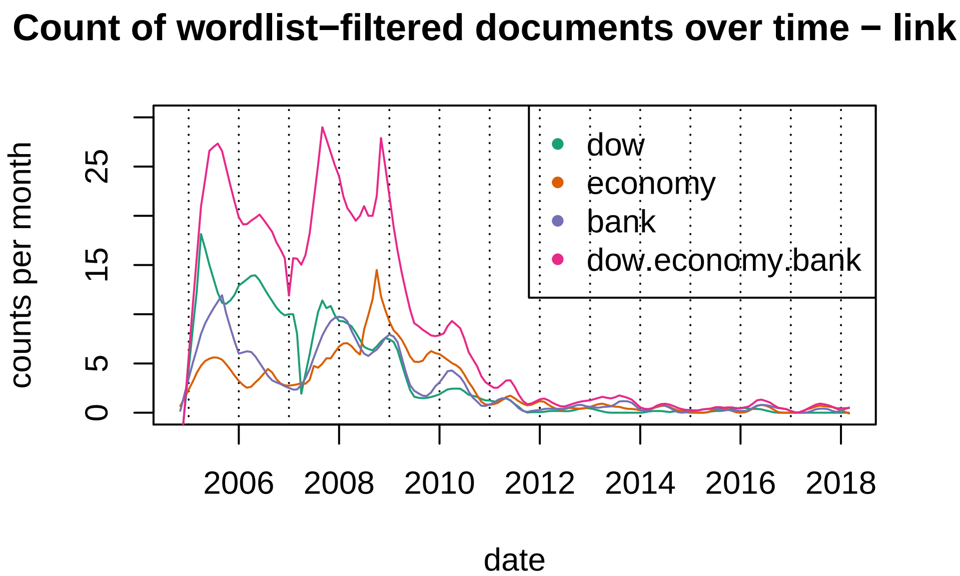
3.3 Frequency Analysis - `plotFreq`

The other descriptive plotting function is `plotFreq` which performs a frequency analysis. Most of the arguments 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 `filterWord` with `out = "count"`, which is explained later on, for counting.

The example corpus contains Wikinews articles concerning the categories *Politics_and_conflicts* and *Economy_and_business*. 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.

```
wordsEconomy <- list("dow", "economy", "bank", c("dow", "economy", "bank"))
plotFreq(corpusClear, wordlist = wordsEconomy, curves = "smooth",
  ylim = c(0, 30), legend = "topright")
plotFreq(corpusClear, wordlist = wordsEconomy, link = "or", curves = "smooth",
  ylim = c(0, 30), legend = "topright")
```

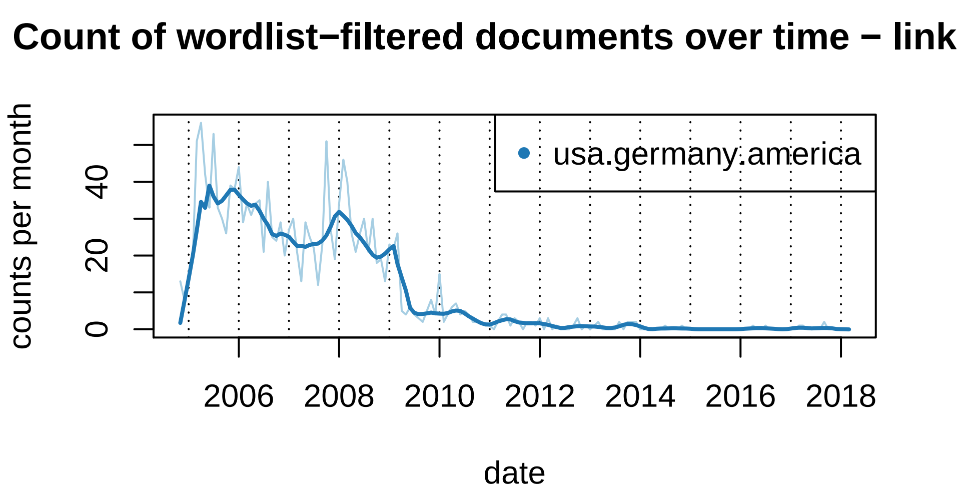




In the figures above you can see the difference between the *and* link and *or* link. The three curves indicating the single words. The fourth curve shows the number of texts in which all three words appear. For mostdates no texts meets this requirements. In the second figure the same single curves are 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 *usa*, *germany* and *america* 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.

```
plotFreq(corpusClear, wordlist = list(c("usa", "germany", "america")), link = "or",
  curves = "both", both.lwd = 2, legend = "topright")
```



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

There are two functions for writing csv files implemented in the package. Both needs an `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 handover a `character` matrix as well, so that each column will be represented in 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, whereas in the fourth column there will be the text itself.

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

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

```
##         id       date
## 1  ID45624 2006-07-19
## 2  ID45095 2006-07-12
## 3  ID61490 2007-03-01
## 4 ID116665 2008-11-14
## 5 ID191439 2010-06-15
## 6   ID7520 2005-04-06
##                                                                        title
## 1                                      Bush uses his first veto ever on stem cell bill
## 2                                                   EU fines Microsoft €280.5 million
## 3 Taliban leader: Osama bin Laden is still alive and Taliban ready to strike Americans
## 4                                                 Eurozone now officially in recession
## 5                             US report says Afghanistan has significant mineral wealth
## 6                               Iraq elects Kurdish president, Saddam said to be shaken
```

We now have 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 handover a matrix of IDs like in `showTexts` 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 `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.

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

```
## $`1`
##                                                title       date
## 2025 Bush uses his first veto ever on stem cell bill 2006-07-19
## 5646               EU fines Microsoft €280.5 million 2006-07-12
##
## $`2`
##                                                                                 title
## 2879 Taliban leader: Osama bin Laden is still alive and Taliban ready to strike Americans
## 6456                                                 Eurozone now officially in recession
##            date
## 2879 2007-03-01
## 6456 2008-11-14
##
```

```
## $`3`
##                                                       title       date
## 321    Iraq elects Kurdish president, Saddam said to be shaken 2005-04-06
## 6894 US report says Afghanistan has significant mineral wealth 2010-06-15
```

4 Generating Subcorpora

The preprocessing which was done before is mandatory. For further preparation the package offers functions for filtering the corpus by dates, wordcount or search terms to generate subcorpora.

4.1 Filter Corpus by Dates - `filterDate`

There are three implemented ways to filter your text corpora: One of these is the function `filterDate`, 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` - including texts from both dates. 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 first associated between 2006 and 2009.

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

```
## Object of class "textmeta":
##  number of observations in text: 6387
##  meta: 6387 observations of 3 variables
##  range of date: 2004-11-13 till 2018-03-04
##  NAs in date: 48 (0.01)
```

```
print(corpusDate)
```

```
## Object of class "textmeta":
##  number of observations in text: 4401
##  meta: 4401 observations of 3 variables
##  range of date: 2006-01-01 till 2009-12-31
```

The filtered corpus contains only the 3909 texts of the period 2006 till 2009.

4.2 Filter Corpus by Wordcount - `filterCount`

TODO:(folgt...)

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 some words handed over by `search`, which could be a simple `character` vector or a `list` of `data.frames`. In the case of a character vector handed over 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.

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.

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

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

```
## text1 text2
##  TRUE  TRUE
```

The returned values are `TRUE` twice. 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` handed over each `data.frame` is linked by an *or* and should contain columns `pattern`, `word` and `count`. The parameter `pattern` includs 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. You must set the argument `count` to an `integer`. As you can imagine this argument controls how often a word or pattern must appear in a document for it to be returned. Rows in each `data.frame` are linked by an *and*. An example is given by the following code.

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

```
## text1 text2
## 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 patterns appear as part of words in the two entries of `texts`. Therefore the function returns `FALSE` twice.

In another use case delete the word *anything* from the search terms.

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

```
## text1 text2
##  TRUE FALSE
```

If you omit the word *anything* from `words` 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 of `out = count` to receive a count for each text and search term combination is given by the following.

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

```
##       some text
## text1    0    1
## text2    3    2
```

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 searchs only for single words, not for strings containing these words. This is displayed by the following example.

```
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")
```

```
##       and and_w
## text1   3     2
## text2   4     3
```

The function returns counts `c(3, 4)` for the simple pattern search and `c(2, 3)` for the word search, cause 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, finally it is used for filtering the Wikipedia corpus. 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 as pattern. It is not necessary to set `ignore.case` because the Wikipedia corpus was cleared before. This step includes that all words are lower case now. TODO:[AND OR, OR Logik]

```
searchterm <- list(
  data.frame(pattern = "economy", word = FALSE, count = 1),
  data.frame(pattern = c("big", "data"), word = FALSE, count = 1),
  data.frame(pattern = "politics", word = FALSE, count = 1))
corpusFiltered = filterWord(corpusDate, search = searchterm)
print(corpusDate)
```

```
## Object of class "textmeta":
##  number of observations in text: 4401
##  meta: 4401 observations of 3 variables
##  range of date: 2006-01-01 till 2009-12-31
```

```
print(corpusFiltered)
```

```
## Object of class "textmeta":
##  number of observations in text: 451
##  meta: 451 observations of 3 variables
##  range of date: 2006-01-02 till 2009-12-17
```

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

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 `LDAgen` for performing the LDA, functions for validating the LDA results and various functions to visualize the results in different ways, especially over time. It is possible to analyse individual articles and its topic allocations as well. 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.

5.1 Transform Corpus - `LDAprep`

The 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 `LDAgen` from this package respectively. This is gained 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 options `ldacorrect`, `excludeNA` and `reduce` set all `TRUE` by default. The returned value is a `list` in which every entry symbolizes an article and contains a matrix with two rows. In the first row there is the index of each word in `vocab` minus one, in the second row there is the number of appearances of each 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. THis structure is needed by `lda.collapsed.gibbs.sampler`.

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

```
wordtableFiltered <- makeWordlist(corpusFiltered$text)
```

```
## Number of Articles: 451

## Find out Vocabularies...
##  Done:

##   0

##   451 next Step

## Calculate Counts...
##  Done:

##   0

##   451 Success
```

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

```
##     gt   said   will     lt people  party
##   1284   1229   1198   1165    831    823
```

```
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 handed over to the function which performs a Latent Dirichlet Allocation.

5.2 Performing LDA - `LDAgen`

The function which has to be applied first to the corpus manipulated by `LDAprep` is `LDAgen`. Therefore 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 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.

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

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

5.3 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 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 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`). TODO:[non-preprocessed corpus nach removexml, ..]

The LDA result of the example corpus is checked by `intruderWords` with a number of intruders of zero or one.

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

```
## counter 1
## 1 ’s
## 2 like
## 3 ds
## 4 think
## 5 cellspacingquotquot
```

As an 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 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 *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.

```
print(intWords)
```

```
## Parameters:
##  byScore numTopwords numIntruder numOutwords noTopic
##     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 many intruder are in the specific set, how many intruders were missed by the user and how many 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.

```
summary(intWords)
```

```
## Not interpretable Topics: 0
## Not evaluated Topics: 0
##
## Parameters:
##  byScore numTopwords numIntruder numOutwords noTopic
##     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 Number of missed Intruders: 0.1
## Table of missing Intruders:
## 0 1
## 9 1
##
## Mean Number of false Intruders: 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.

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

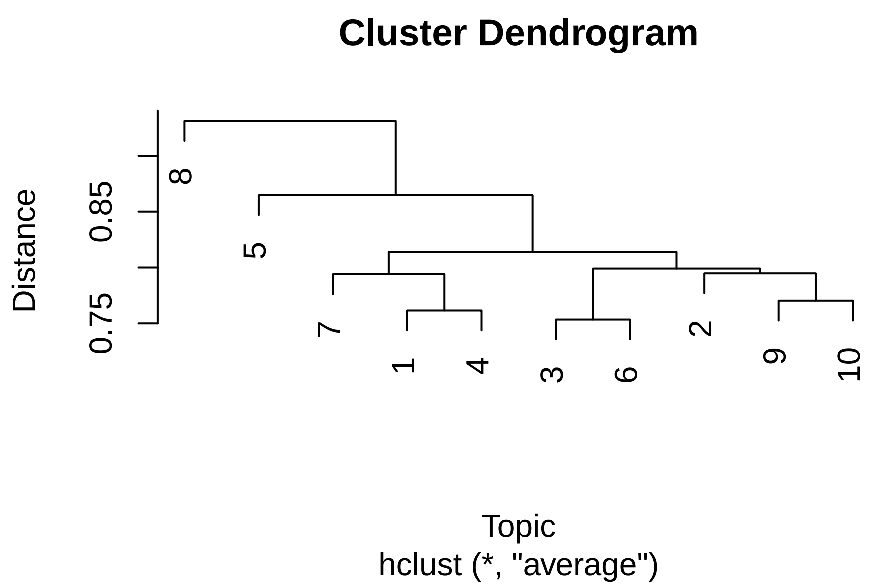
Analogously to `intruderWords` you can use `intruderTopics` for validation the other way around. This function is used for validation of topics associated to a specific document instead of validation of words associated to one topic. Therefore the document is displayed in another window and a sample 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 names the intruder(s). There are options for different numbers of topics and intruders like in `intruderWords` as well. The parameter `theta` should be set to `result$document_expects` given `result` is the LDA result. An example call is given below.

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

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 influences 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")
```




```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.

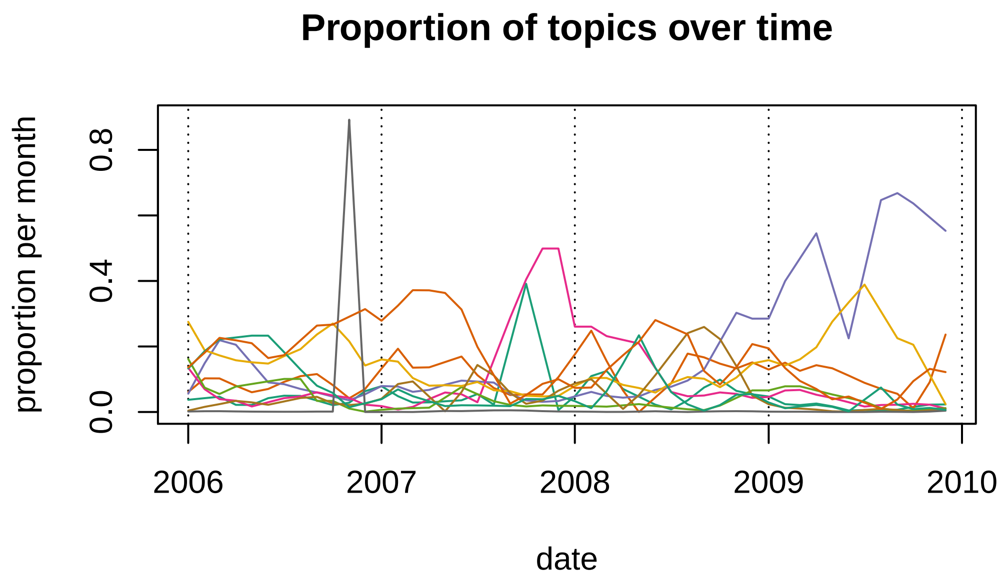
It is possible to merge different results of LDAs by calling `mergeLDA(list(result1, result2, ..., resultN))`. The function `mergeLDA` binds the `topics` elements of the results by row and only consider words which appears in all results. As result you receive the `topics` matrix including all topics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

5.5 Visualisation of Topics over Time - `plotTopic`

As extension of the highly flexible functions `plotScot` and `plotFreq` 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 `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`.

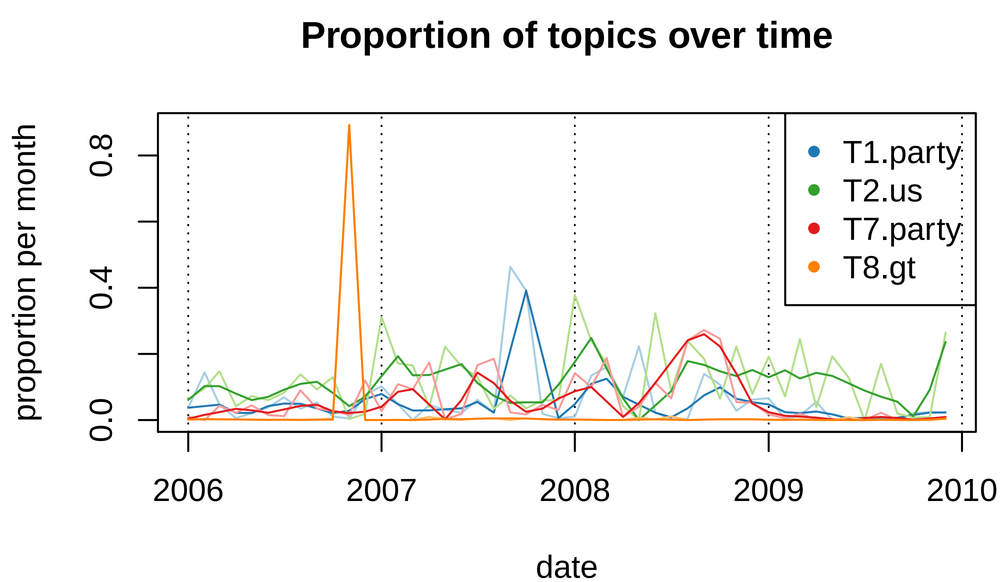
Often it is useful to choose `curves = "smooth"` if you do not select topics, because there is a massive fluctuation of exact curves. However, it is important to have a look at the exact curves, because the smoothed curves are someway manipulated by the statement `smooth`, so the user is tempted to optimise 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, 0.9))
```



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.

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

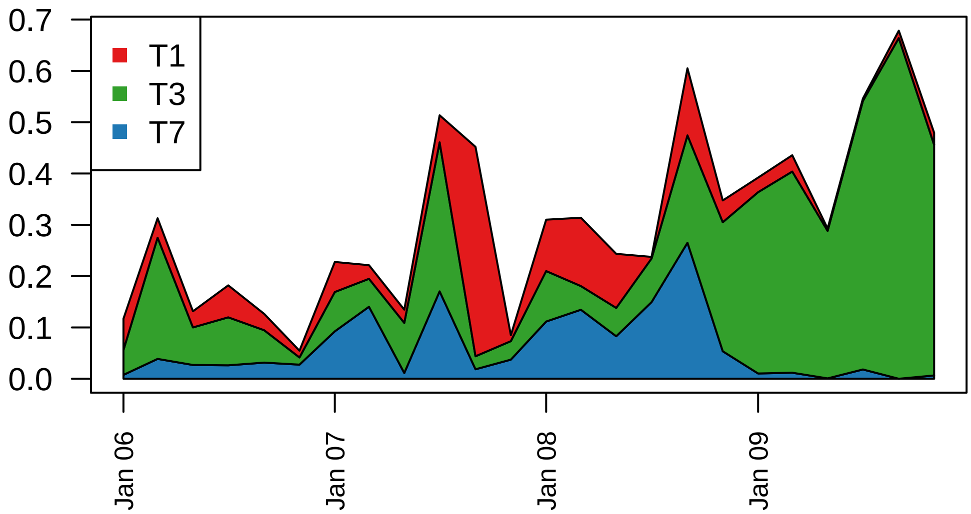


5.6 Visualisation of Topic Share over Time - `plotArea`

The function `plotArea` 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.

Because 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"` (default is `"quarter"`).

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



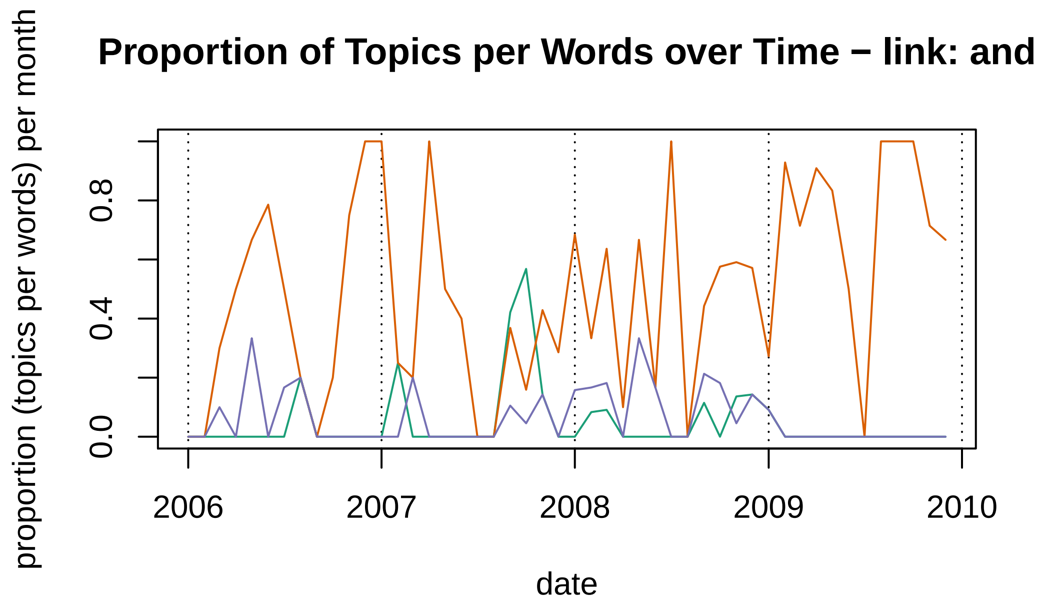
(TODO) Interpretation.

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

Another visualise 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 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. The certain object should be created while preparation for LDA 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 *syndrome* (lightgreen curve), *health* (orange) and *medicine* (purple).

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



The graphic shows that the word *economy* is associated to the topic *health* most often.

For interpretating 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.

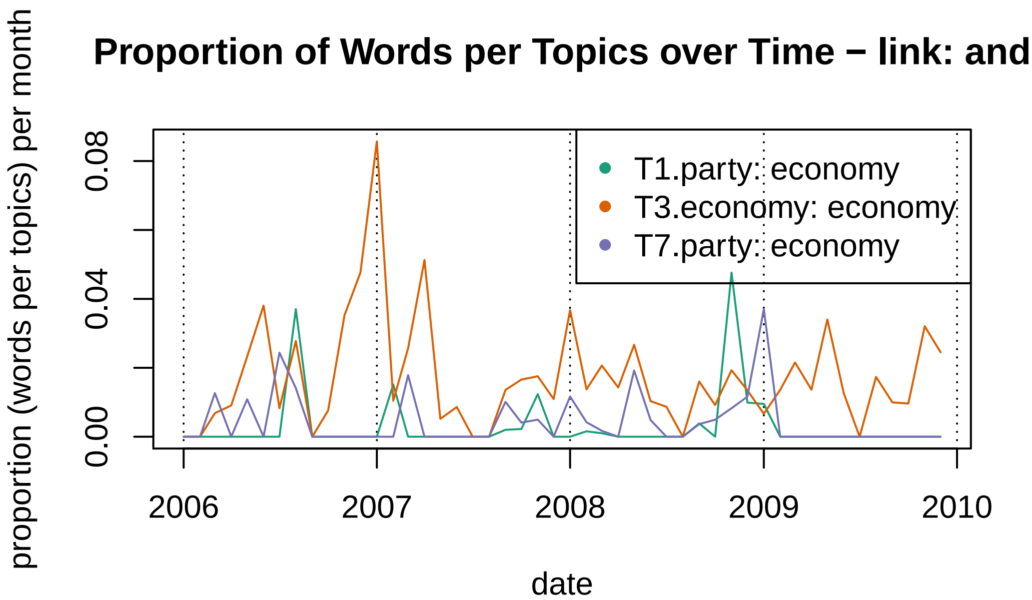
```
tab <- plotTopicWord(corpusFiltered, pagesLDA, result, ldaID, "economy", rel = TRUE)
all(round(rowSums(tab[, -1]), 10) == 1)
```

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

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.

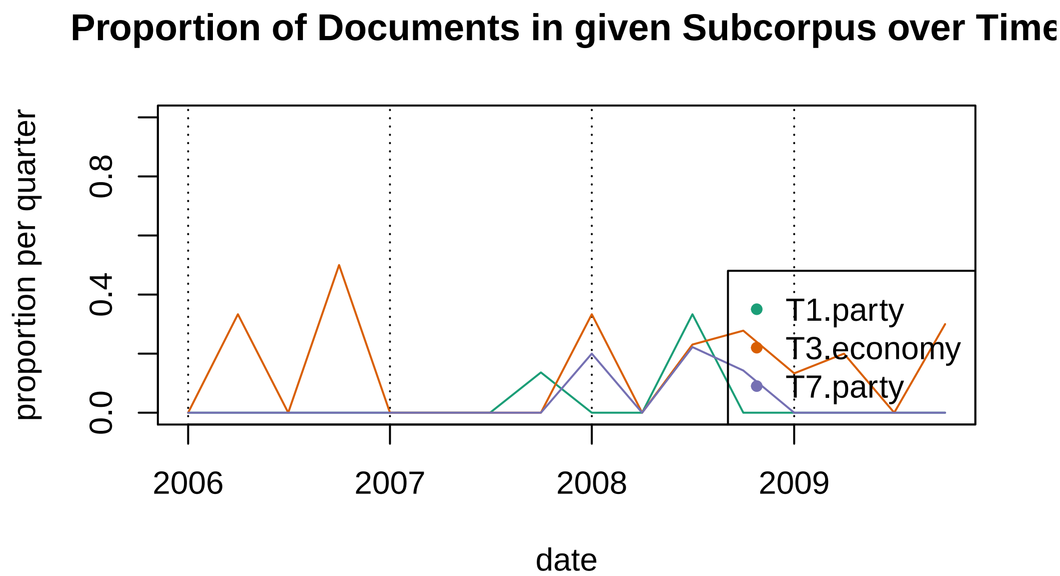
```
plotWordpt(object = corpusFiltered, docs = pagesLDA, ldaresult = result, ldaID = ldaID,
  wordlist = "economy", select = c(1, 3, 7), rel = TRUE)
```



5.8 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 `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.

```
searchmed <- data.frame(pattern = "economy", word = TRUE, count = 3)
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 *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`).

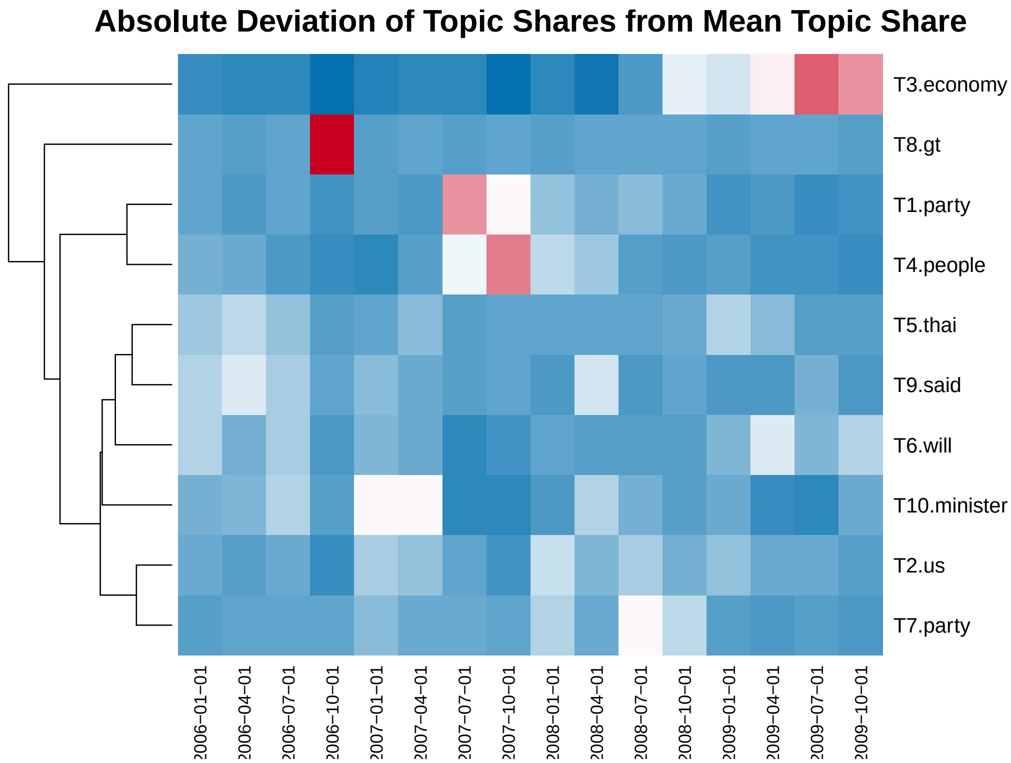
5.9 Heatmap of Topics over Time including Clustering - `plotHeat`

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`.

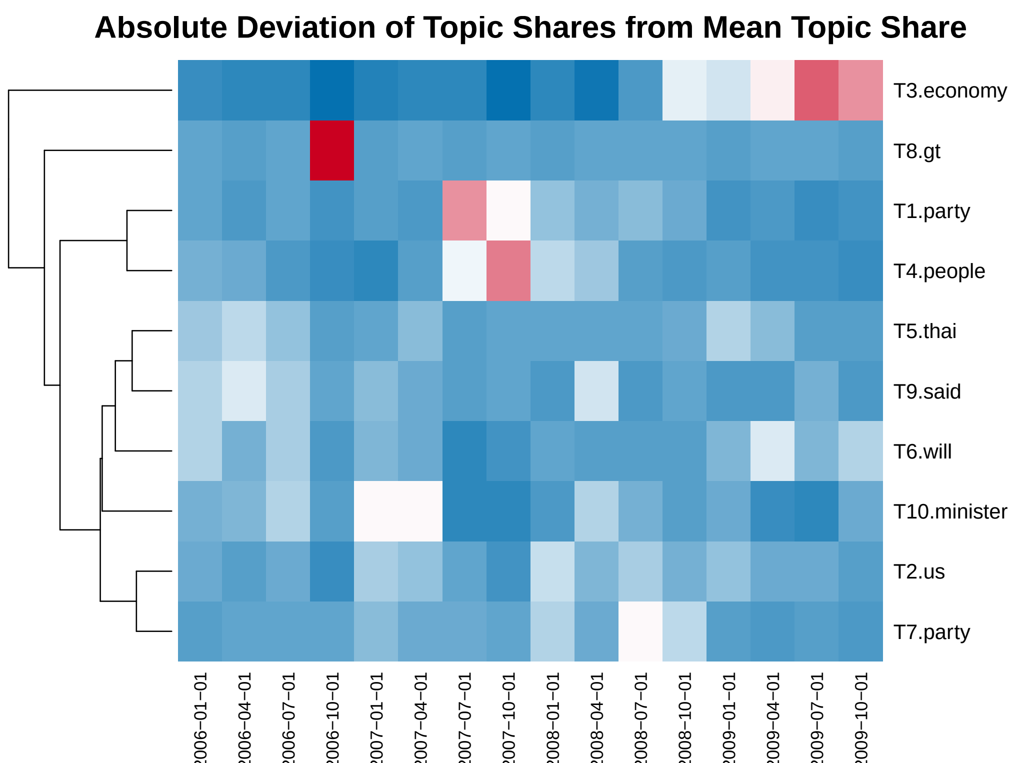
By default the proportions are calculated on the article lengths, but it is possible to force calculation on only the LDA vocabulary by setting `object` to a `textmeta` object only including meta information. Otherwise a strictly tokenized `textmeta` object is required. The parameters `ldaresult` and `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 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.

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




```
plotHeat(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`.

5.10 Individual Cases Contemplation - `topTexts`, `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 `topTexts`. 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 <- topTexts(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 `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.

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

```
## $T1.party
##        id
## 1 ID80831
## 2 ID80085
## 3 ID80627
## 4 ID81573
##                                                                                      title
## 1     Ontario Votes 2007: Interview with Libertarian candidate Mark Scott, Toronto-Danforth
## 2          Ontario Votes 2007: Interview with Green candidate Peter Ormond, Hamilton Centre
## 3        Ontario Votes 2007: Interview with Green candidate Andrew McAvoy, Windsor-Tecumseh
## 4 Ontario Votes 2007: Interview with Freedom Party candidate Wayne Simmons, Don Valley East
##         date
## 1 2007-09-27
## 2 2007-09-18
## 3 2007-09-24
## 4 2007-10-03
##
## $T3.economy
##         id
## 1 ID114565
## 2 ID114904
## 3 ID117712
## 4 ID140460
##                                                                    title
## 1                                      Worldwide markets fall precipitously
## 2 Dow Jones recovers hundreds of points, before losing them in minutes
```

```
## 3                             US November job losses reach 34-year high
## 4                                     US unemployment rate reaches 9.8%
##          date
## 1 2008-10-06
## 2 2008-10-10
## 3 2008-12-05
## 4 2009-10-02
##
## $T7.party
##         id
## 1 ID112754
## 2 ID112390
## 3 ID110264
## 4 ID111956
##                                                                                   title
## 1                        US presidential candidate John McCain now leads slightly in the polls
## 2 US presidential candidate Barack Obama's lead increases after Democratic National Convention
## 3                         US presidential candidate Barack Obama's lead in the polls increases
## 4                       US presidential race tied as the Democratic National Convention starts
##          date
## 1 2008-09-09
## 2 2008-09-01
## 3 2008-07-26
## 4 2008-08-26
```

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 `LDAprep` 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.

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

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. 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 . . .)