An introduction to the *polmineR* (v0.6.0)

Andreas Blaette (andreas.blaette@uni-due.de)

June 22, 2016

1 Purpose

The purpose of the package *polmineR* is to facilitate the interactive analysis of corpora using R. Core objectives for the development of the package are performance, usability, and a modular design.

There are quite a few R packages for text mining already, it is quite unnecessary to implement again what has already been implemented. Thus, this package is also meant to serve as an interface between the Corpus Workbench (CWB)[1], an efficient system for storing and querying large corpora, and existing packages for mining text with advanced statistical methods. There are several relevant packages on CRAN.[2]

Quantitative text analytics may bear the risk to get out of touch with the original text. The driller seeks to keep the actual text analysed accessible. This is a further reason why the CWB is used as a backend. Apart from the speed of text processing, the Corpus Query Processor (CQP) and the CQP syntax[3] provide a great machinery to query corpora. Queries found can be viewed with a concordancer.

Using a combination of R and the CWB implies a software architecture you will also find in the TXM project[4]. TXM, among other things, offers a rich functionality for importing corpora into the CWB. A very specific concern of the driller is to provide the means to examine and to compare subcorpora that are generated based on the metainformation stored as structural attributes to the CWB corpus. In line with the French tradition of discourse analysis (and for technical reasons), these subcorpora are called partitions here. The driller may be particularly useful for analyses of diachronic variation and synchronic change.

[1] `http://cwb.sourceforge.net/`
[2] `http://cran.r-project.org/web/views/NaturalLanguageProcessing.html`
[3] `http://cwb.sourceforge.net/files/CQP_Tutorial.pdf`
[4] `http://sourceforge.net/projects/txm/`

2 Corpora

The *driller* is an instrument to analyse corpora imported to the CWB. The CWB distinguishes structural attributes (s-attributes) that will contain the metainformation that can be used to generate subcorpora, and positional attributes (p-attributes). Typically, the p-attributes will be 'word', 'pos' (for part-of-speech) and 'lemma' (for the lemmatized word form).

The driller was developed for analysing corpora with a flat XML strucutre. Future versions of the driller will be able to process nested XML. Yet so far, it is generally speaking necessary that all metadata are attributes of one XML element. This may be unproblematic if you work with a corpus of newspaper articles, for instance (attributes might be 'date', 'author', 'newspaper', 'page' etc.). For the corpora of plenary debates that were the impetus to develop the package[5], a respective transformation generated the flat XML structure.

In the package, two (very small) sample corpora from the PolMine project are included. The corpus "PLPRBTTXT" is a set of five plenary protocols (out of several hundred) of the German Bundestag. The sample corpus can only be used if the respective registry files contain the path to the binary files of the indexed corpora. The setting is managed by a configure file that is called when the package is installed.

The sample corpora are encoded in latin-1, the traditional encoding used by the CWB. The package is not yet entirely generic as far as encodings are concerned. Problems may still arise when using a utf-8 encoded corpus. Future versions of the driller will be generic in this respect.

3 Installation

3.1 System requirements

For the time being, only the installation on Linux and Mac OS is supported.

For a Windows installation, the driller itself should not be the problem. The parallelization the package implements may only be used on a Mac/Linux system (due to forking), but multicore processing can be switched of. However, there may be problems with the rcqp package, as it uses several libraries potentially unavailable on Windows.[6]

3.2 Dependencies

The driller relies on a few packages (see DESCRIPTION): The tm and the rcqp package are crucial.

The tm package offers a wealth of functions for textmining. It is imported primarly because of the TermDocumentMatrix class and related methods. Once an object of that class has been generated, all tm functions can be used. There

[5] See `http://polmine.sowi.uni-due.de`

[6] A workaround may be an installation in a Linux virtual machine.

are quite a few packages that use the tm TermDocumentMatrix as an input (e.g. topicmodels or lsa). So in a sense, tm is not just a dependency but the link that pushes the R world of text mining open once you have extracted information from a CWB corpus using the driller.

Installation of the rcqp package can be tricky. The rcqp package is however core for the whole driller project. It provides the API for connecting to the CWB.

Apart from R packages rcqp requires (plyr, for instance) that can be installed easily with install.packages(), rcqp requires a few non-R libraries (pkg-config, libffi, gettext, glib). The installation files are easily found in the web. Then the ./configure, make, make install procedure is needed. A tricky hurdle is that pkg-config requires glib, and glib requires pkg-config. A workaround is to start the pkg-config installation with ./configure –without-internal-glib. To install glib on a Mac, I found it useful to use homebrew as an installer (brew install glib).

3.3 Loading the package

As mentioned, the rcqp package is the most important dependency of the driller. The CORPUS_REGISTRY environment variable *before* needs to be set before you load polmineR.

```
if ("CORPUS_REGISTRY" %in% names(Sys.getenv())){
  if (require(rcqp, quietly = T) && require(polmineR.sampleCorpus, quietly = T)
  ){
    execute <- TRUE
  } else {
    execute <- FALSE
  }
} else {
  execute <- FALSE
}
```

```
if (execute){
  library(polmineR)
  CQI <- CQI.rcqp$new()
  library(polmineR.sampleCorpus)
  use("polmineR.sampleCorpus")
}

##
## Attaching package:  'polmineR'
## The following objects are masked from 'package:rcqp':
##
##    corpus, size
```

```
## The following object is masked from 'package:plyr':
##
##     count

## [1] "/Users/blaette/Lab/cwb/registry"
```

4 Default settings

Default settings are stored in the general options settings.

```
if (execute){
  # to view all options defined for polmineR
  options()[grep("polmineR", names(options()))]

  # setting options
  options("polmineR.corpus" = "PLPRBTTXT")
  options("polmineR.left" = 15)
  options("polmineR.right" = 15)
  options("polmineR.mc" = FALSE)
}
```

Various methods will get default values from the options set. See the documentation for kwic, for instance.

5 Setting up a partition

Usually, any session using the driller will start with initializing a partition. The return of a call of the partition function is a S4 partition object, and almost every function of the driller package will require a partition object as an input.

In this example, I set up a partition with the speeches, not the interjections, deliverd by members of the CDU parliamentary group in the parliament of Northrhine-Westfalia. Attributes are handed over as a list. Setting up a partition sometimes consumes some time, so you will get messages about progress.

```
if (execute){
  bt <- partition("PLPRBTTXT", text_type="speech")
  cdu <- partition(
    "PLPRBTTXT",
    text_type="speech", text_party="CDU_CSU"
    )
}

## Setting up partition
```

```
## ...  get encoding:  latin1
## ...  get cpos and strucs
## ...  get partition size
## Setting up partition
## ...  get encoding:  latin1
## ...  get cpos and strucs
## ...  get partition size
```

To get some basic information about the partition that has been set up, the 'show'-method can be used. It is also called when you simply type the name of the partition object.

```
if (execute){
  cdu
}

## ** partition object **
## corpus:              PLPRBTTXT
## name:
## sAttributes:         text_type = speech
##                      text_party = CDU_CSU
## cpos:                570 pairs of corpus positions
## size:                56811 tokens
## count:               not available
```

Note that is possible to omit steps of the initialization of a partition object, thus speeding up the initialization significantly. Setting up a table with metadata and retrieving term frequencies can be switched off (metadata=FALSE, tf=FALSE). Other functions (context, distribution) however require respective information and do not work if the partition object lacks this information. Setting up a partition may be a bit slow, but generates information that allows further analytical steps based on a partition object to be much quicker.

There are two methods to set up a partition, 'grep' and 'in'. If the method is 'in', you can provide a character vector for every s-attribute. If the method is "grep", all s-Attribute values are kept that match a regex. As an example for the 'in'-procedure, if you want a partition comprising of CDU/CSU and FDP as parties, you might formulate:[7]

```
if (execute){
  coalition <- partition(
    "PLPRBTTXT",
    text_type="speech", text_party=c("CDU_CSU", "FDP")
    )
}
```

[7]Sometimes, you will want to be more specific about the start and end date of a partition. In this case, you can set a dateRange.

```
## Setting up partition
## ...  get encoding:  latin1
## ...  get cpos and strucs
## ...  get partition size
```

If you work with a flat XML structure, the order of the provided s-attributes may be relevant for speeding up the set up of the partition. For a nested XML, it is important that with the order, you move from ancestors to childs. For further information, see the documentation of the function.

6 Getting a tm TermDocumentMatrix

For many applications, term-document matrices are the point of departure. The tm class TermDocumentMatrix serves as an input to several R packages implementing advanced text mining techniques. Obtaining this input from a corpus imported to the CWB will usually involve setting up a partitionCluster and then applying a method to get the matrix.

```
if (execute){
  base <- partition("PLPRBTTXT", text_type="speech")
  parties <- partitionBundle(
    base, def=list(text_party=NULL),
    pAttribute="word", progress=TRUE, verbose=FALSE
    )
  tdm <- as.TermDocumentMatrix(parties, col="count")
  class(tdm) # to see what it is
  show(tdm)
  m <- as.matrix(tdm) # turn it into an ordinary matrix
  m[c("Integration", "Zuwanderung", "Migration"),]
}
```

7 Context analysis

The partition object has a method that will give you immediate access to view the concordances of a (single-word) word found in a partition by indexing the partition object accordingly (here: `min["Minderheitsregierung"]`). This a shortcut that may be useful at times. Usually, the context function will be called first, the summary method will give some information on the resulting context object.

```
if (execute){
  integration <- context(
     bt, "Integration", pAttribute="word",
```

```
    left=20, right=20
  )
  summary(integration)
}

## ...  count for pAttribute word not available
## ...  computing term frequencies (for p-attribute word)
## ...  getting cpos
## ...  generating contexts
## ...  counting tokens
## ...  statistical test:  ll

##
## ** Context object - general information: **
## CWB-Korpus:          PLPRBTTXT
## Partition:
## Node:                Integration
## P-Attribute:         word
## Node count:          23
## Stat table length:   392
```

Note that is is possible to provide a query that uses the full CQP syntax. The statistical analysis of collocations to the query can be accessed as the slot "stat" of the context object (here: `min@stat`).

To view some concordances, the context object can be indexed accordingly. If you put a query in double brackets, it is used as a filter, giving you those concordances containing this query (e.g. `min[["Scheitern"]]`). If you run R in a console, you may use xterm color highlighting. The colors can be set via drillingControls. This applies also for the metainformation you receive as output.

8 Distribution of queries

To understand the occurance of a phenomenon, the distribution of query results across one or two dimensions will often be interesing. This is done via the 'distribution' function. The query may use the CQP syntax. The function is a wrapper for three different functions. Which one is called will depend on the number of queries provided and whether one or two s-attributes are provided as dimensions. The output depends on the input and the respective function that is called.

```
if (execute){

  # one query / one dimension
  oneQuery <- dispersion(
```

```
    bt, query = '"Gerechtigkeit"',
    "text_party", progress = F
  )

  # # multiple queries / one dimension
  twoQueries <- dispersion(
    bt,
    c('"[eE]uro.*"', '"Br.ssel"'),
    "text_party", progress = F
  )

  # multiple queries / two dimensions
  twoDim <- dispersion(
    bt, query = '"Regierung"',
    c("text_date", "text_party"), progress = F
  )

}
```

9 Compare

To identify the specific vocabulary of a corpus of interest, a keyness test based on the chi square test can be performed. The following example also shows how a partitio can be used based on a grep procedue.

```
if (execute){
  coalition <- enrich(coalition, pAttribute="word")
  bt <- enrich(coalition, pAttribute="word")
  vocabulary <- compare(coalition, bt, included=TRUE)
}
```

10 Shiny

If the driller runs on a local installation, some shiny apps can be tested that are included in the package by calling polmineR().

The apps will search the global environment for partition objects and offer to choose one of these from a drop-down menu. Thus, for using the apps, relevant partition objects should be generated first during a command-line R session. Then you may start calling the apps.

The apps are still experimental and at times may throw out errors before you get a result. They are included to demonstrate how using the driller is meant to be made more convenient in future versions of the package.