# A short introduction to the `tm (text mining)` package in `R`: text processing

Iñaki Inza, Borja Calvo

`R` is a popular language and environment for statistical computing and graphics, `http://www.r-project.org/about.html`, specially suited for working with arrays and matrices. Hundreds of manuals can be found for different levels of users. To name a couple, Torfs and Brauer's introduction and a series of videotutorials by Google Developers. However, my experience tells that a good way to learn is to "just start playing", and pieces will start putting in place. A valuable source of help is the web, just searching for the desired concept or function, followed by the `R` term.

> `swirl` package provides an introduction to `R` by means of interactive courses. Type `?InstallCourses` at the `R` prompt for a list of functions that will help you to select a course fitted to your knowledge level and specific interests.

The language has a compact "core" and, by means of different packages programmed by the community, it is highly extensible. Thus, it nowadays offers tools (packages) to do specific computation and analysis in different domains such as bioinformatics, chemistry, computational physics, etc. Among those, the `tm` (text mining) [2, 1] package offers an interesting set of state-of-the-art functions to process text documents in an effective manner: creating a corpus, preprocessing-transformation operators, creating document-term matrices, etc. This tutorial is limited to these operations. Further, over these matrices, similar terms or documents can be clustered, and machine learning specialized packages (e.g. `caret` [4]) allow to train models to classify new documents in pre-defined classes-annotations.

> Function `library("packageName")` loads the functions of a package. Prior to these, it is needed to install it by `install.packages("packageName")`.

> The working directory can be consulted by `getwd()` and fixed by setwd(""C:/Users/User Name/Documents/FOLDER) in windows or setwd("/home/inza/Documentos/Software-Datasets/20 Newsgroups Data Set/20news-bydate-train/") in Linux.

The 20 NewsGroup dataset (link) is a benchmark collection of approximately 20,000 newsgroup documents, partitioned in 20 predefined thematic newsgroups. We work with the "20news-bydate.tar.gz" compressed file. After unzip the data, it first split it in training and testing folders. Each of them contains 20 folders, each containing the text documents belonging to one newsgroup. We focus on training "sci.electronics" and "talk.religion.misc" groups.

Help for a function in R is obtained by invoking `?VCorpus` or `help(VCorpus)`.
A good way to consult the different functions (and parameters) of each package is the `http://www.inside-r.org/` community. In our case, the functions of the `tm` package, grouped alphabetically, can be consulted in `http://www.inside-r.org/packages/cran/tm/docs`.
Searching in the web for the specific function, followed by the R term, usually produces good helping results.

We start by reading the documents of each subdirectory (annotated document class) and loading in a volatile corpora structure. Function `inspect` displays detailed information of the corpus.

```r
library(tm)
sci.elec = VCorpus(DirSource("sci.electronics"),readerControl=list(language="en"))
talk.religion = VCorpus(DirSource("talk.religion.misc"),readerControl = list(language="en"))
sci.elec # dimension of the corpus
inspect(sci.elec[1]) # first document of the corpus, or sci.elec.train[[1]]
inspect(sci.elec[1:3]) # first three documents of the corpus
```

The `tm` package allows the use of the `meta` function to access and modify metadata of documents, e.g. `meta(sci.elec[[1]],"language")` or `meta(sci.elec[[1]])`. Internal structure of an object in R can be consulted in the following form, e.g. `sci.elec.train[[1]]$meta$language`. After inserting the $ symbol, by means of the use of the tab key, the list of different slots-attributes of the object is displayed.

Transformations operators are applied via `tm_map` function, which applies (maps) a function to all elements of the corpus. As the same transformations will be applied to both "science" and "religion" newsgroups, both corpus are merged using base function `c`, which raises a collection of 968 documents. Function `tm_map` applies transformations to corpus objects. The list of available transformations can be obtained consulting the help of `?getTransformations`. Function `content_transformer` is used to apply customized transformations. We apply several transformations. Feel free to consult the help and parameters of each transformation's functions.

```r
sci.rel=c(sci.elec,talk.religion)
?getTransformations
sci.rel.trans=tm_map(sci.rel,removeNumbers)
sci.rel.trans=tm_map(sci.rel.trans,removePunctuation)
sci.rel.trans=tm_map(sci.rel.trans, content_transformer(tolower)) # convert to lowercase
stopwords("english") # list of english stopwords
sci.rel.trans=tm_map(sci.rel.trans,removeWords,stopwords("english"))
sci.rel.trans=tm_map(sci.rel.trans,stripWhitespace)
library(SnowballC) # to access Porter's word stemming algorithm
sci.rel.trans=tm_map(sci.rel.trans,stemDocument)
```

After corpus set transformation, a common approach in text mining is to create a *document-term matrix* from a corpus. Its transpose operator creates a term-document matrix. This document-term matrix is the starting point to apply machine-learning modelization techniques such as classification, clustering, etc. Different operations can be applied over this matrix. We can obtain the terms that occur at least, let say, 15 times; or consult the terms that associate with at least, for example, by a 0.7 correlation degree with the term "young". After all, it is easy no note the huge degree of sparsity of this matrix: a low amount of non-zero elements. Thus, one of the most important operations is to remove *sparse* terms, i.e., terms occurring in very few documents. The `sparse` parameter in the `removeSparseTerms` function refers to the maximum sparseness allowed: the smaller its proportion, fewer terms (but more common) will be retained. A "trial-an-error" approach will finally return a proper number of terms. This matrix will be the starting point for building further machine learning models. See next tutorial.

In case of an available dictionary of terms, the document-term matrix is created by tabulating against it, i.e., only terms from the dictionary appear in the matrix: `DocumentTermMatrix(sci.rel.trans,list=dictionary("student","trip","young"))`

```
sci.rel.dtm=DocumentTermMatrix(sci.rel.trans)
dim(sci.rel.dtm)
inspect(sci.rel.dtm[15:25,1040:1044]) # inspecting a subset of the matrix
findFreqTerms(sci.rel.dtm,15)
findAssocs(sci.rel.dtm,term="young",corlimit=0.7)
sci.rel.dtm.70=removeSparseTerms(sci.rel.dtm,sparse=0.7)
sci.rel.dtm.70 # or dim(sci.rel.dtm.70)
# note that the term-document matrix needs to be transformed (casted)
# to a matrix form in the following barplot command
barplot(as.matrix(sci.rel.dtm.70),xlab="terms",ylab="number of occurrences",
        main="Most frequent terms (sparseness=0.7)")
sci.rel.dtm.80=removeSparseTerms(sci.rel.dtm,sparse=0.8)
sci.rel.dtm.80
sci.rel.dtm.90=removeSparseTerms(sci.rel.dtm,sparse=0.9)
sci.rel.dtm.90
```

Different functionalities of `R` allow to convert these matrices to the file format demanded by other software tools. For example, the function `write.arff` of the `foreign` package converts a data matrix or data frame to the well-known arff ("attribute relation format file") of WEKA software. Note that a class vector is appended to the document-term matrix, labeling the type of each document. We know that the first 591 documents cover the "science-electronics" newgroup.

```
data=as.data.frame(inspect(sci.rel.dtm.90))
type=c(rep("science",591),rep("religion",377)) # create the type vector to be appended
write.arff(cbind(data,type),file="term-document-matrix-weka-format.arff")
```

Before starting learning the exposed machine learning models, let's build a *wordcloud* with the following package [3]. Its `wordcloud()` command needs the list of words and their frequencies as parameters. As the words appear in columns in the document-term matrix, the `colSums` command is used to calculate the word frequencies. In order to complete the needed calculations, note that the term-document matrix needs to be transformed (casted) to a matrix form with the `as.matrix` cast-operator. It is built for the "talk.religion" newsgroup: it covers the 591-968 range of samples-documents (rows) in the document-term matrix.

```
library(wordcloud)
# calculate the frequency of words and sort in descending order.
wordFreqs=sort(colSums(as.matrix(sci.rel.dtm.90)[591:968,]),decreasing=TRUE)
wordcloud(words=names(wordFreqs),freq=wordFreqs)
```

#### Other ways to retrieve-import text

It is easy to retrieve text in HTML format from the web by means of `R` functionalities. A simple example can be to retrieve a subset of the "Terms and Services" offered by Google, specifically those of the subsection "Using our services". The `readLines` command outputs a vector of character strings, each component storing a line. The text of interest exists from line 44 through 49. The HTML tags are removed. Now, by means of the `tm` package, the vector of character strings can be converted into a corpus of text using consecutively the `VectorSource` and the `VCorpus` functions.

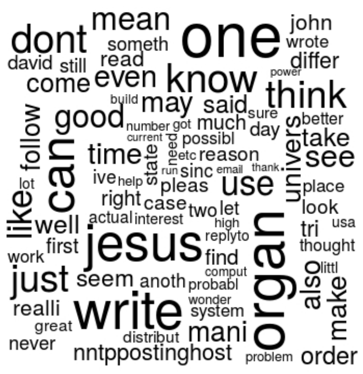

Figura 1: Wordcloud for the 0.90 sparseness-degree document-term matrix (“talk.religion newsgroup”)

```
policy.HTML.page = readLines("https://www.google.es/intl/en/policies/terms/regional.html")
length(policy.HTML.page)
text.data = policy.HTML.page[44:49] # 6 lines-paragraphs of desired text
text.data = gsub(pattern="<p>",replacement = "", x=text.data)
text.data = gsub(pattern="</p>",replacement = "", x=text.data)
text.data
text.data = VectorSource(text.data) # interpreting each element of the vector as a document
text.data.corpus = VCorpus(text.data) # from document list to corpus
```

Other common practice to retrieve text is by means of the examples stored by `R` packages. In this case, when installing the `tm` package a set of document collections are stored in hard disk. Among them, a subset of the popular “Reuters-21578” collection. The `system.file` function points the subdirectory-tree that has the documents of interest in our package. When reading the text, an specific XML reader developed by the community, known as “readReut21578XMLasPlain”, is needed.

```
reut21578 = system.file("texts","crude",package="tm")
reut21578
reuters = VCorpus(DirSource(reut21578),readerControl=list(reader = readReut21578XMLasPlain))
```

A fashion way to retrieve text is via Twitter posts. The `twitteR` library provides access to Twitter data. Twitter API requires identification-authentification: follow these instructions: pay attention, the current Twitter’s link to create Twitter applications is url. You need to be logged. It is needed to “create a new app”: this will provide you a set of 4 items related to the application called “consumerKey”, “consumerSecret”, “accessToken” and “accessSecret”. Both ‘accessToken” and “accessSecret” need to be activated after receiving the “consumerKey” and “consumerSecret”. Four items need to be used in the final authentification function call, `setup_twitter_oauth`.

The following link also explains the exposed Twitter’s identification-authentification process, which hangs from the general Twitter Developer Documentation.

At a first glance, this process can be felt as cumbersome-complex. Be patient, and after several trials sure that you will be able to authentificate. If the process fails, try installing the `httr` and `base64enc` packages; if the error continues, install the `twitteR` package from GitHub:

```
devtools::install_github("jrowen/twitteR", ref = "oauth_httr_1_0")
```

Once the authentification is done, tweets of any user or hashtag can be retrieved and converted to a corpus. The functions provided by the `twitteR` package evolve continuously and sure that you find interesting functions for your NLP analysis objectives.

```
# consult the different ways to retrieve tweets: from an user or from a hashtag
?userTimeline
?searchTwitteR
GreenPeaceTweets <- userTimeline(user="Greenpeace",n=100)
beerTweets = searchTwitteR(searchString = "#beer", n=100)
length(GreenPeaceTweets)
GreenPeaceTweets[1:4]
# convert each tweet to a data frame and combine them by rows
# a corpus can be created from GreenPeace or beer's hashtag tweets
GreenPeaceTweetsDataFrames <- do.call("rbind", lapply(GreenPeaceTweets, as.data.frame))
# consult the different attributes of this object using \£ symbol
GreenPeaceTweetsDataFrames$text
# interpreting each element of the vector as a document
GreenPeaceDocuments = VectorSource(GreenPeaceTweetsDataFrames$text)
# from document list to corpus
GreenPeaceCorpus = VCorpus(GreenPeaceDocuments)
```

**Proposed exercise**

We propose you to create your own corpus and document-term matrix of interest. The original documents can be in any of the exposed forms (files, html, twitter). Of course, not the documents used in this tutorial. Sure that during the master you have found interesting examples that can be processed with the exposed functions. In case you do not create your corpus starting from tweets, due to its novelty and hot-trend, we also encourage you to practice with the exposed Twitter API and `twitteR` package and create, at least, a brief corpus.

We ask you to edit a similar tutorial. Your document should mix explanations-descriptions of your decisions (as we do), together with the `R` code that implements them. In a colloquial way, "do your own tutorial".

This tutorial-document has been edited by means of the `knitr` package in the `RStudio` software (`Rnw` format). Great if you edit your tutorial with this technology, which is elegant and popular nowadays. However, feel free to edit your tutorial with other technologies.

## Bibliografía

[1] Ingo Feinerer. *tm: Text Mining Package*, 2012. R package version 0.5-7.1.

[2] Ingo Feinerer, Kurt Hornik, and David Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25(5):1–54, 3 2008.

[3] Ian Fellows. *wordcloud: Word Clouds*, 2014. R package version 2.5.

[4] M. Kuhn and K. Johnson. *Applied Predictive Modeling*. Springer, 2013.