Text Mining with R: Building a Text Classifier

Martin Schweinberger

July 28, 2016

This post[1] will exemplify how to create a text classifier with `R`, i.e. it will implement a machine-learning algorithm, which classifies texts as being either a speech by Barack Obama or Mitt Romney. The script is based on Timothy DAuria's YouTube tutorial "How to Build a Text Mining, Machine Learning Document Classification System in R!" (`https://www.youtube.com/watch?v=j1V2McKbkLo`).

As it has been suggested that it may be helpful to make the speeches available for download to render this example reproducible, the respective folders with the speeches are accessible at `http://martinschweinberger.de/docs/data/speeches.zip` and the code for downloading the speeches is available at `http://martinschweinberger.de/docs/scripts/DownloadingSpeechesTM.r`.

What we need is a folder containing the speeches of Barak Obama and Mitt Romney (in fact I download the speeches directly from two webpages which contain speeches but this would cause the post to be much, much longer). I hope that the annotation within the code is sufficient, otherwise feels free to contact me and I will elaborate and add more annotation...

So let's start with a short description of what this piece of code will do, cleaning the workspace, and activating the packages that are needed for creating a text classifier.

```
#########################################################
### Text Mining with R: Building a Text Classifier
#########################################################
# Title: Text Mining with R: Building a Text Classifier
```

[1] Please cite as:
Schweinberger, Martin. 2016. *Text Mining with R: Building a Text Classifier.* `http://www.martinschweinberger.de/blog/textclass/`, date.

```
# Author: Martin Schweinberger
# Date: 2016-07-28
# Description: This script uses  a sample of speeches by
# Barack Obama and Mitt Romney to train a text classifier
# based on the words the candidates use in order to classify
# unknown speeches of the two candidates.
#########################################################
# Remove all lists from the current workspace
rm(list=ls(all=T))
# load packages
library("plyr")
library("tm")
library("class")
# define options
options(stringsAsFactors = FALSE)
```

After initializing the R session, a vector with the names of the two candidates is created. Then, a function is written which cleans the texts by removes punctuation, strips superfluous white spaces, converts everything to lower case, and removes stop words, i.e. grammatical function words that do not carry lexical meaning such as *a*, *an*, *that*, *the*, *this* and so on.

```
# set parameters
candidates <- c("romney", "obama")
pathname <- "C:\\03-MyProjects\\TextMining\\speeches\\"
# clean texts
cleanCorpus <- function(corpus){
  corpus.tmp <- tm_map(corpus, removePunctuation)
  corpus.tmp <- tm_map(corpus.tmp, removePunctuation)
  corpus.tmp <- tm_map(corpus.tmp, stripWhitespace)
  corpus.tmp <- tm_map(corpus.tmp, content_transformer(
     tolower))
  corpus.tmp <- tm_map(corpus.tmp, removeWords, stopwords("
     english"))
  return(corpus.tmp)
  }
```

After writing a cleaning function the text document matrix is created and the cleaning function is applied to the texts.

```
# create text document matrix
generateTDM <- function(cand, path){
  s.dir <- sprintf("%s/%s", path, cand)
  s.cor <- Corpus(DirSource(directory = s.dir, encoding = "
     UTF-8"))
  s.cor.cl <- cleanCorpus(s.cor)
```

```
  s.tdm <- TermDocumentMatrix(s.cor.cl)
  s.tdm <- removeSparseTerms(s.tdm, 0.7)
  result <- list(name = cand, tdm = s.tdm)
  }
# execute function and create a Text Document Matrix
tdm <- lapply(candidates, generateTDM, path = pathname)
# inspect results
str(tdm)
```

The structure of the created object is displayed below.

```
List of 2
$ :List of 2
 ..$ name:  chr "romney"
 ..$ tdm :List of 6
 ..  ..$ i :  int [1:70033] 1 2 3 4 5 6 7 8 9 10 ...
 ..  ..$ j :  int [1:70033] 1 1 1 1 1 1 1 1 1 1 ...
 ..  ..$ v :  num [1:70033] 1 1 1 1 1 1 1 1 1 1 ...
 ..  ..$ nrow :  int 1179
 ..  ..$ ncol :  int 68
 ..  ..$ dimnames:List of 2
 ..  ..  ..$ Terms:  chr [1:1179] "011012" "011508" "012411" "013009" ...
 ..  ..  ..$ Docs :  chr [1:68] "romney001.txt" "romney002.txt" "romney003.txt" ...
 ..  ..- attr(*, "class")= chr [1:2] "TermDocumentMatrix" "simple_triplet_matrix"
 ..  ..- attr(*, "weighting")= chr [1:2] "term frequency" "tf"

 $ :List of 2
 ..$ name:  chr "obama"
 ..$ tdm :List of 6
 ..  ..$ i :  int [1:44000] 1 2 3 4 6 7 8 9 10 11 ...
 ..  ..$ j :  int [1:44000] 1 1 1 1 1 1 1 1 1 1 ...
 ..  ..$ v :  num [1:44000] 1 1 1 1 3 4 2 3 1 1 ...
 ..  ..$ nrow :  int 572
 ..  ..$ ncol :  int 102
 ..  ..$ dimnames:List of 2
 .. .. .. $ Terms: chr [1:572] ""call""| __truncated__ "2002" "2004" "2005" ...
 ..  ..  ..$ Docs :  chr [1:102] "obama001.txt" "obama002.txt" "obama003.txt" ...
 ..  ..- attr(*, "class")= chr [1:2] "TermDocumentMatrix" "simple_triplet_matrix"
 ..  ..- attr(*, "weighting")= chr [1:2] "term frequency" "tf"
```

Now, a function is written which creates a data frame of the list objects which combines the TDM and the name of the respective candidate.

```
# attach names of candidates
bindCandidatetoTDM <- function(tdm){
  s.mat <- t(data.matrix(tdm[["tdm"]]))
  s.df <- as.data.frame(s.mat, stringsAsFactors = FALSE)
```

```
  s.df <- cbind(s.df, rep(tdm[["name"]], nrow(s.df)))
  colnames(s.df)[ncol(s.df)] <- "targetcandidate"
  return(s.df)
  }
# apply function
candTDM <- lapply(tdm, bindCandidatetoTDM)
# inspect data
str(candTDM)
```

The structure of the created object is displayed below.

```
List of 2
 $ :'data.frame':  68 obs.  of 1180 variables:
 ..$ 011012 :  num [1:68] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 011508 :  num [1:68] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 012411 :  num [1:68] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 013009 :  num [1:68] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 013112 :  num [1:68] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 020312 :  num [1:68] 1 1 1 1 1 1 1 1 1 1 ...
 ..  [list output truncated]

 $ :'data.frame':  102 obs.  of 573 variables:
 ..$ \call :  num [1:102] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 2002 :  num [1:102] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 2004 :  num [1:102] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 2005 :  num [1:102] 1 1 1 1 1 1 1 1 1 1 ...
 ..$ 2006 :  num [1:102] 0 0 0 0 0 0 0 0 0 0 ...
 ..$ 2007 :  num [1:102] 3 3 3 3 3 3 3 3 3 3 ...
 ..$ 2008 :  num [1:102] 4 6 7 6 5 5 5 5 5 5 ...
 ..  [list output truncated]
```

Next, the two list objects are combined into a single data frame and rows containing `NA` (non available values) are removed.

```
# stack texts
tdm.stack <- do.call(rbind.fill, candTDM)
tdm.stack[is.na(tdm.stack)] <- 0
# inspect data
head(tdm.stack)
```

We are now in a position to separate the data frame into a training and a test data set. The training data is used to train our classifier that is then applied to the test data.

```
# create hold-out
```

```
train.idx <- sample(nrow(tdm.stack), ceiling(nrow(tdm.stack)
    * 0.7))
test.idx <- (1:nrow(tdm.stack))[-train.idx]
```

Now, we use K-Nearest-Neighbor Clustering to classify the texts.

```
# create model - knn clustering
tdm.cand <- tdm.stack[, "targetcandidate"]
tdm.stack.nl <- tdm.stack[, !colnames(tdm.stack) %in% "
    targetcandidate"]
# set up model
knn.pred <- knn(tdm.stack.nl[train.idx,], tdm.stack.nl[test.
    idx,], tdm.cand[train.idx])
```

In a final step, we determine the accuracy of or classifier.

```
# determine accuracy
conf.mat <- table("Predictions" = knn.pred, Actual = tdm.cand
    [test.idx])
# calculate accuracy
accuracy <- sum(diag(conf.mat)) / length(test.idx) * 100
# inspect accuracy
accuracy
```

The number being evoked by `accuracy` represents the percentage of correctly classified documents. In our case, the number is...

```
100
```

This means that our text classifier works very well as it classifies all documents, i.e. 100% of the documents, correctly. In other words, it assigns each text to its actual author (Obama or Romney) based on the words they use.