**Text Mining en Social Media**

Master en Big Data Analytics

Universidad Politécnica de Valencia

7 de junio de 2021

**SERCOPA**

Sergio Campos Pérez

Adrián Cortijo Simarro

Pablo Pons Roger

En primer lugar para realizar el trabajo cargamos todas las librerías que van a ser necesarias para la realización del mismo, instalando previamente los paquetes que no venían en nuestra versión de R.



|  |
| --- |
| *#.libPaths("c://Local/R/libs") # Including needed libraries library(qdap) library(XML) library(tm) library(splitstackshape) library(caret) library(stringr) library(purrr)  start.time <- Sys.time()* |

Después, se indican los parámetros del número de palabras del vocabulario, el número de pliegues de la validación cruzada y el número de veces que la repite

|  |
| --- |
| n <- 1000 # Number of words in the vocabulary. Usually used 1000 or 10000 k <- 5 # Number of folds in cross-validation. Usually used 10 r <- 3 # Number of repeats in cross-validation. Usually used 3 |

A continuación, se asigna el nombre de los archivos del training y del test y el lenguaje en el que realizamos el intento.

|  |
| --- |
| path\_training <- "C:/Users/pabponro/Downloads/Text Mining en Social Media/pan21-author-profiling-training-2021-03-14/es" # Your training path path\_test <- "C:/Users/pabponro/Downloads/Text Mining en Social Media/pan21-author-profiling-test-without-gold-for-participants/es"  # Your test path lang <- "es" |

# Auxiliar functions

# \* GenerateVocabulary: Given a corpus (training set), obtains the n most frequent words

# \* GenerateBoW: Given a corpus (training or test), and a vocabulary, obtains the bow representation

# GenerateVocabulary: Given a corpus (training set), obtains the n most frequent words

GenerateVocabulary <- function(path, n = 1000, lowcase = TRUE, long2 = TRUE, url = TRUE, punctuations = TRUE, numbers = TRUE, whitespaces = TRUE, swlang = "", swlist = "", verbose = TRUE) {

setwd(path)

# Reading corpus list of files

files = list.files(pattern="\*.xml")

# Reading files contents and concatenating into the corpus.raw variable

corpus.raw <- NULL

i <- 0

for (file in files) {

xmlfile <- xmlTreeParse(file, useInternalNodes = TRUE)

corpus.raw <- c(corpus.raw, xpathApply(xmlfile, "//document", function(x) xmlValue(x)))

i <- i + 1

if (verbose) print(paste(i, " ", file))

}

# Preprocessing the corpus

corpus.preprocessed <- corpus.raw

if (lowcase) {

if (verbose) print("Tolower...")

corpus.preprocessed <- tolower(corpus.preprocessed)

}

A continuación, implementamos la siguiente función para eliminar las palabras que contienen menos de 2 caracteres.

|  |
| --- |
| if (long2) {  corpus.preprocessed <- keep(.x = corpus.preprocessed, .p = function(x){str\_length(x) > 1})  } |

Además, también eliminamos todas las URLS.

|  |
| --- |
| if (url) {  corpus.preprocessed <- str\_replace\_all(corpus.preprocessed,"http\\S\*","")  } |

if (punctuations) {

if (verbose) print("Removing punctuations...")

corpus.preprocessed <- removePunctuation(corpus.preprocessed)

}

if (numbers) {

if (verbose) print("Removing numbers...")

corpus.preprocessed <- removeNumbers(corpus.preprocessed)

}

if (whitespaces) {

if (verbose) print("Stripping whitestpaces...")

corpus.preprocessed <- stripWhitespace(corpus.preprocessed)

}

if (swlang!="") {

if (verbose) print(paste("Removing stopwords for language ", swlang , "..."))

corpus.preprocessed <- removeWords(corpus.preprocessed, stopwords(swlang))

}

if (swlist!="") {

if (verbose) print("Removing provided stopwords...")

corpus.preprocessed <- removeWords(corpus.preprocessed, swlist)

}

# Generating the vocabulary as the n most frequent terms

if (verbose) print("Generating frequency terms")

corpus.frequentterms <- freq\_terms(corpus.preprocessed, n)

if (verbose) plot(corpus.frequentterms)

return (corpus.frequentterms)

}

# GenerateBoW: Given a corpus (training or test), and a vocabulary, obtains the bow representation

GenerateBoW <- function(path, vocabulary, btruth = TRUE, n = 100000, lowcase = TRUE, long2 = TRUE, url = TRUE, punctuations = TRUE, numbers = TRUE, whitespaces = TRUE, swlang = "", swlist = "", verbose = TRUE) {

setwd(path)

Modificamos el código para cuando lanzamos el módelo contra el test, no vaya al truth porque la carpeta de test no existe.

|  |
| --- |
| # Reading the truth file  if (btruth) {  truth <- read.csv("truth.txt", sep=":", header=FALSE)  truth <- truth[,c(1,4)]  colnames(truth) <- c("author", "class")  } |

i <- 0

bow <- NULL

# Reading the list of files in the corpus

files = list.files(pattern="\*.xml")

for (file in files) {

|  |
| --- |
| # Obtaining truth information for the current author  author <- gsub(".xml", "", file)    if (btruth)  class <- truth[truth$author==author,"class"]  else  class <- 1    if (class==1) {  class = "faker"    } else {  class = "normal"  } |

# Reading contents for the current author

xmlfile <- xmlTreeParse(file, useInternalNodes = TRUE)

txtdata <- xpathApply(xmlfile, "//document", function(x) xmlValue(x))

# Preprocessing the text

if (lowcase) {

txtdata <- tolower(txtdata)

}

|  |
| --- |
| if (long2) {  txtdata <- keep(.x = txtdata, .p = function(x){str\_length(x) > 1})  }    if (url) {  txtdata <- str\_replace\_all(txtdata,"http\\S\*","") |

}

if (punctuations) {

txtdata <- removePunctuation(txtdata)

}

if (numbers) {

txtdata <- removeNumbers(txtdata)

}

if (whitespaces) {

txtdata <- stripWhitespace(txtdata)

}

# Building the vector space model. For each word in the vocabulary, it obtains the frequency of occurrence in the current author.

line <- author

freq <- freq\_terms(txtdata, n)

for (word in vocabulary$WORD) {

thefreq <- 0

if (length(freq[freq$WORD==word,"FREQ"])>0) {

thefreq <- freq[freq$WORD==word,"FREQ"]

}

line <- paste(line, ",", thefreq, sep="")

}

line <- paste(class, ",", line, sep="")

# New row in the vector space model matrix

bow <- rbind(bow, line)

i <- i + 1

if (verbose) {

print(paste(i, author, class))

}

}

return (bow)

}

# GENERATE VOCABULARY

vocabulary <- GenerateVocabulary(path\_training, n, swlang=lang)

vocabulary <- vocabulary[11:nrow(vocabulary),]

# GENERATING THE BOW FOR THE TRAINING SET

bow\_training <- GenerateBoW(path\_training, vocabulary)

# PREPARING THE VECTOR SPACE MODEL FOR THE TRAINING SET

training <- concat.split(bow\_training, "V1", ",")

training <- cbind(training[,2], training[,4:ncol(training)])

names(training)[1] <- "theclass"

Nuestro método ha sido separar el dataset de training en haters y no haters para ver cuales son las 5 palabras que usan con más frecuencia los haters.

|  |
| --- |
| library(dplyr) fakers <- training %>% filter(theclass=="faker") normal <- training %>% filter(theclass=="normal")  fakerswsum <- fakers %>% group\_by(theclass) %>% summarise\_each(funs(sum)) fakerswsum <- fakerswsum[2:91] fakerswsum <- sort(fakerswsum, decreasing = TRUE) fakersmfw <- fakerswsum[1:5]  Después, incluimos una característica más al dataset, que es cuantas veces ha utilizado cada autor las palabras hater.   fakecolumns <- training[,c('V1\_03','V1\_06','V1\_07','V1\_08','V1\_36')] fakecolsum <- rowSums (fakecolumns[ , 1:5]) training <- cbind(training,fakecolsum) |

* **PCA ( Principal Component Analysis)**

Se ha decidido hacer un PCA ( análisis de componentes principales) para intentar mejorar el modelo.

# trpca = prcomp(data.matrix(training))

Finalmente se descarta su utilización ya que hemos visto que no ayuda a reducir la dimensionalidad y por lo tanto no aporta valor en el proyecto.

Hemos probado un SVM (Support Vector Machine) con cross-validation y también un Gaussian-Mixture. Como vemos que el Gaussian-Mixture da peores resultados decidimos entrenar el SVM de nuevo con todo el training.

|  |
| --- |
| train\_control <- trainControl( method="repeatedcv", number = k , repeats = n) print("SVM...") model\_SVM <- train( theclass~., data= training, trControl = train\_control, method = "svmLinear") print(model\_SVM) #print("GLM...") #model\_GLM <- train( theclass~., data= training, trControl = train\_control, method = "gaussprLinear") #print(model\_GLM)  # Learning a SVM with the whole training set and without evaluating it train\_control <- trainControl(method="none") model\_SVM <- train( theclass~., data= training, trControl = train\_control, method = "svmLinear") |

A continuación, generamos el Bag of Words para los datos del test.

|  |
| --- |
| # GENERATING THE BOW FOR THE TEST SET bow\_test <- GenerateBoW(path\_test, vocabulary, btruth = FALSE) # # Preparing the vector space model and truth for the test set test <- concat.split(bow\_test, "V1", ",") truth <- unlist(test[,3]) test <- test[,4:ncol(test)] |

Añadimos la misma característica que habíamos añadido al training al test, añadiendo la columna de cuantas veces han utilizado las palabras más frecuentes de los haters.

|  |
| --- |
| fakecolumns <- test %>% select(contains(names(fakersmfw))) fakecolsum <- rowSums (fakecolumns[ , 1:5]) test <- cbind(test,fakecolsum) |

Utilizamos el modelo ya entrenado con el training para hacer una predicción sobre los datos de test y generamos el xml con la predicción para la entrega y evaluación.

|  |
| --- |
| # Predicting and evaluating the prediction pred\_SVM <- predict(model\_SVM, test) #confusionMatrix(pred\_SVM, truth)  clase <- NULL for (i in 1:length(pred\_SVM)){  if (pred\_SVM[i]=="normal"){  clase[i]<-0  } else {  clase[i]<-1  } } resultado<-data.frame(truth,clase, row.names = NULL) setwd("C:/Users/pabponro/Downloads/Text Mining en Social Media/resultados")  textoxml<-NULL for (i in 1:nrow(resultado)) {  textoxml[i] <- paste0("<author id=","\"",resultado$truth[i],"\"", "\n",  "lang=\"en\"", "\n",  "type=\"",resultado$clase[i], "\"", "\n",  "/",">")  name <- paste0(resultado$truth[i], ".xml")  fileConn<-file(name)  writeLines(textoxml[i], fileConn)  close(fileConn)   }   end.time <- Sys.time() time.taken <- end.time - start.time print(time.taken)  Comentar que los resultados en inglés son peores porque sus mensajes de odio son más explícitos y en cambio los españoles son mucho más implícitos. |