DCM_ML2_Assignment

David Cabestany

2/7/2022

Call of the libraries that we will need to use to Train and test our models

In this part we will define our credentials to download Twitter data and make two corpora based in tweets from two rappers (Wiz Khalifa and Eminem), and in the other hand we have all the tweets from DeGeneres a journalist with a tv show called The Ellen Show on the Oprah's Show line.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidytext)
library(qdapRegex)
```

```
##
## Attaching package: 'qdapRegex'

## The following object is masked from 'package:dplyr':
##
##     explain

## The following object is masked from 'package:ggplot2':
##
##     %+%
```

```
library(stringr)
library(rtweet)
```

```
##
## Attaching package: 'rtweet'
```

```
## The following object is masked from 'package:purrr':
##
##     flatten
```

```
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
#Credentials for download and use Twitter Data

consumer_key <- "AcNvgbFPcmvEWsXlkTi6XfYDC"
consumer_secret <- "PHyQNe5bikame6YVpa11BnJYEWltPCqIPMXfLlmw3tk7orQlSE"
access_key <- "3407173097-LfJ82g1ylAA1vHw9uHEt9dX1nqcJKt1bKCwRk9R"
access_secret <- "z4dQ6FjLdjCm2YBMv8TV4Ja2DbOXBsOMqRSKRJH96OYPO"

token <- create_token(
  app = "mlclasstm",
  consumer_key = consumer_key,
  consumer_secret = consumer_secret,
  access_token = access_key,
  access_secret = access_secret)

token
```

In this chunk we download and make the two timelines and convert to DataFrames for make the future corpora for the model trainings and testing.

```
wizkhalT <- get_timeline("wizkhalifa", n=20000, include_rts = FALSE)
eminemT <- get_timeline("Eminem", n=20000, include_rts = FALSE)

rapperT <- rbind (eminemT, wizkhalT)

#rapperT

degeneresT <- get_timeline("TheEllenShow", n=20000, include_rts = FALSE)

#degeneresT
```

```
#eminemT

#degeneresT

rapperT <- rapperT[rapperT$is_retweet==FALSE, ]
rapperT <- subset(rapperT, is.na(rapperT$reply_to_status_id))


degeneresT <- degeneresT[degeneresT$is_retweet==FALSE, ]
degeneresT <- subset(degeneresT, is.na(degeneresT$reply_to_status_id))
```

In this chunk we combine the two corpora

```
Tweet_DF = rbind(rapperT, degeneresT)
nrow(Tweet_DF) #3709 tweets
ncol(Tweet_DF)
```

```
dim(rapperT)
dim(degeneresT)
```

```
tweets = VectorSource(Tweet_DF$text) #interpret annotated vector as a document
cp.tweets = VCorpus(tweets)# convert into a corpus
```

In the following chunks we are merging both corpora in one in order to work with it. After that we have applied some cleaning rules, we aim to remove numbers, puntuation, and the most of stopwords, the following step will remove the whitespaces, and in the last step we will do a stemming.

```
cp.tweets.trans <- cp.tweets #create a copy

toSpace <- content_transformer(function(x, pattern) gsub(pattern, "", x))
cp.tweets.trans<-tm_map(cp.tweets.trans,removeWords,stopwords("english"))


cp.tweets.trans <- tm_map(cp.tweets.trans,removePunctuation)
cp.tweets.trans <- tm_map(cp.tweets.trans, content_transformer(tolower))
cp.tweets.trans<-tm_map(cp.tweets.trans,stripWhitespace)
cp.tweets.trans<-tm_map(cp.tweets.trans,stemDocument)

rapperT$text[0:5]
degeneresT$text[0:5]
```

```
cleaningfunction <- function(corpus){
  remvURL <-function(x) gsub("http[^[:space:]]*", "", x)
  corpus <- tm_map(corpus, content_transformer(remvURL))

  remvusernames <- function(x) gsub("(^|[^@\\w])@(\\w{1,15})\\b", "", x)
  corpus <- tm_map(corpus, content_transformer(remvusernames))

  corpus <- tm_map(corpus, content_transformer(tolower))

  corpus <- tm_map(corpus, removePunctuation)
```

```
  corpus <- tm_map(corpus, removeNumbers)

  customstopwords <- c(stopwords("english"),"rt","retweet","amp","follow","\n")

  corpus <- tm_map(corpus, removeWords, customstopwords)

  corpus <- tm_map(corpus, stripWhitespace)
  return(corpus)
}


cp.tweet.clean <-cleaningfunction(cp.tweets.trans)
```

Here we apply the function for cleaning the corpora, and we aim to remove the sparse terms, but with two such different styles of writting, things get complicated and we need to get a very high parameter.

```
tweet.dtm = DocumentTermMatrix(cp.tweet.clean)
dim(tweet.dtm)

#tweetDTM have a dimension of 5537 7167

tweet.dtm.99 = removeSparseTerms(tweet.dtm, sparse=.99)
dim(tweet.dtm.99) # 5537   0
```

Here we plot the word cloud of the most frequent terms in the corpora.

```
findFreqTerms(tweet.dtm.99, lowfreq = 2, highfreq = Inf)
wordfreqsT=sort(colSums(as.matrix(tweet.dtm.99)[,]),decreasing=TRUE)

wordT<-names(wordfreqsT)

freqT<-wordfreqsT

palT <- RColorBrewer::brewer.pal(12, "Blues")

wordcloud(wordT,freqT, min.freq=1,
          max.words=Inf, random.order=FALSE, rot.per=.1, min_font_size=8,
          max_wordsnumber=800, colors=palT)
```

Here we print the matrixes of the two corpora by separated, where the rows are the instances and the columns are the particularities of every tweet as author, text, and so on. And we combine bpth to set the labels for further training.

```
rapperT = nrow(rapperT)
degeneresT = nrow(degeneresT)
dim(tweet.dtm.99) # 5537  117

type=c(rep("rappers",rapperT),rep("degeneres",degeneresT))
tweet.dtm.99=cbind(tweet.dtm.99,type)
dim(tweet.dtm.99) # 5537  118

tweet.dtm.col = ncol(tweet.dtm.99)
tweet.dtm.99.ML.matrix=as.data.frame(as.matrix(tweet.dtm.99))
colnames(tweet.dtm.99.ML.matrix)[tweet.dtm.col]="type"
dim(tweet.dtm.99.ML.matrix) # 5537  118

tweet.dtm.99.ML.matrix$type <- as.factor(tweet.dtm.99.ML.matrix$type)

tweet.dtm.99.ML.matrix <- as.data.frame(sapply(tweet.dtm.99.ML.matrix, as.numeric))
tweet.dtm.99.ML.matrix[is.na(tweet.dtm.99.ML.matrix)] <- 0
levels(tweet.dtm.99.ML.matrix$type) <- c("rappers", "degeneres")
```

Here we prepare the train and test by splitting the dataset in two in a proportion of 75% for train and 25% for test.

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
set.seed(0)
cp.train <- createDataPartition(y=tweet.dtm.99.ML.matrix$type,p=.75,list=FALSE)
#str(cp.train)

train <- tweet.dtm.99.ML.matrix[cp.train,]
test <- tweet.dtm.99.ML.matrix[-cp.train,]
nrow(train)
```

```
## [1] 4152
```

```
nrow(test)
```

```
## [1] 1384
```

In this chunk we made a train model with the naives bayes algorithm. For that the optimal value of folds for the cross-validation must be determined. The trainControl command is used for this. In the example, cross-validation has been set to 10 parts. The data is divided into 10 subsamples of the same size.

```
trctrl <- trainControl(method="repeatedcv", number = 10)
metric = "Accuracy"


train$type <- as.factor(train$type)
naive_model <- train(type ~ ., data=train,method="naive_bayes",
                     tuneLength=10, trControl=trctrl , metric=metric)
naive_model
```

```
## Naive Bayes
##
## 4152 samples
##  117 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 3736, 3738, 3736, 3736, 3736, 3738, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.6519955  0.2954752
##    TRUE      0.4879576  0.0000000
##
```

```
## Tuning parameter 'laplace' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = FALSE
##  and adjust = 1.
```

```
naive_modelClasses <- predict(naive_model, newdata = test, type = "raw")
confusionMatrix(data=naive_modelClasses,as.factor(test$type))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 640 469
##          2  51 224
##
##                Accuracy : 0.6243
##                  95% CI : (0.5982, 0.6499)
##     No Information Rate : 0.5007
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2492
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9262
##             Specificity : 0.3232
##          Pos Pred Value : 0.5771
##          Neg Pred Value : 0.8145
##              Prevalence : 0.4993
##          Detection Rate : 0.4624
##    Detection Prevalence : 0.8013
##       Balanced Accuracy : 0.6247
##
##        'Positive' Class : 1
##
```

We made the same for the Support Vector Machines algorithm but with a different seed.

```
#training SVM

#trctrl <- trainControl(method = "repeatedcv", number = 10)
set.seed(3)

svm_Linear <- train(type ~., data = train, method = "svmLinear",
                trControl=trctrl,
                tuneLength = 10, metric=metric)
```

```
svm_Linear

test_pred <- predict(svm_Linear, newdata = test, type= "raw")
```

```
#test_pred

confusionMatrix(data=test_pred,as.factor(test$type))
```

Here in the last chunk we made the resampling for contrast the two results of our models.

As we can see in the last two confusion Matrix the Bayesian model has an accuracy average of 0.62, while the SVM has an accuracy average of 0.744.

```
resampling <- resamples(list(svm_l=svm_Linear, nb=naive_model))
summary(resampling)

xyplot(resampling,what="BlandAltman")
```