Assignment 1 Bayesian Statistics

José María Martínez Marín - 100443343

Text classification exercise using the Naive Bayes Classifier.

You should find some data for which you can use a naive Bayes classifier (examples from the past are spam and ham emails, tweets by Donald Trump and Hilary Clinton, different types of car). For the data set you select, you should clean the data, produce a word cloud if appropriate, divide the data into a training set and a test set and run the classifier using both frequentist estimation and Laplacian smoothing (the Bayes method).

You should write a brief report, summarising the data and the results of the classifier and any possible problems and improvements that need to be used.

The dataset is a merge ot three datasets chosen from Kaggle ("https://www.kaggle.com/marklvl/sentiment-labelled-sentences-data-set") which contains information about ratings from users of three pages:

-**IMDb** (Internet Movie Database, as its name indicates, it is a website about cinema and TV series).

-**Amazon**.

-**Yelp** (reviews from various companies, rated with 1-5 stars).

It consists of 3000 instances, which are classified as either Positive (1) or Negative (0), and in the case of Yelp, a rating of 1 or 2 stars counts as Negative, while 4 or 5 stars accounts for a Positive review.

The first step is loading the data

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.6     v dplyr   1.0.4
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

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

```
library(gmodels)
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'
```

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

```
library(wordcloud)
```

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

```
library(e1071)

datapath <- "C:/Users/Jose Maria/Dropbox/Mi PC (DESKTOP-26FICGE)/Documents/5 MASTER BIG DATA ANALYTICS/1
csv.files <- list.files(path=datapath, full.names = T, recursive = T)

csv.files <- csv.files[str_detect(csv.files, "(amazon|imdb|yelp)")]
```

A function is created to create a dataset consisting on the three files from Amazon, IMDb and Yelp, and shuffling them in random order.

```
Loading <- function(file_list) {
    tables <- lapply(file_list, Reading)
    ratings <- do.call(rbind, tables)


    ratings$sentiment <- factor(ratings$sentiment, levels = c(0, 1),
                                labels = c("Negative","Positive"))

    return(ratings)
}

Reading <- function(file_name) {

    table <- read_delim(file_name,
                        delim = "\t",
                        col_names = c("text", "sentiment"),
                        quote = "")

    file.name <- str_split(file_name, "/", simplify = TRUE)[2]
    table['source']  <- str_split(file.name, "_", simplify = TRUE)[1]

    return(table)
}

ratings <- Loading(csv.files)


set.seed(1985)
ratings <- ratings[order(runif(n=3000)),]
```

The next step is to prepare the corpus, an element consisting of a collection of documents.

```
library(tm)
corpus <- Corpus(VectorSource(ratings$text))
inspect(corpus[1:10])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 10
##
##  [1] It has everything I need and I couldn't ask for more.
##  [2] This movie is terrible.
##  [3] It seems completely secure, both holding on to my belt, and keeping the iPhone inside.
##  [4] I love this cable - it allows me to connect any mini-USB device to my PC.
##  [5] I didn't realize how wonderful the short really is until the last two scenes.
##  [6] The commercials are the most misleading.
##  [7] Having trouble with volume.
##  [8] Seriously, it's not worth wasting your, or your kid's time on.
##  [9] This was my first time and I can't wait until the next.
## [10] My colleague & I now get great reception.A little expensive, but performance is great.
```

and cleaning it, removing numbers, punctuation signs, transforming the letters into lower case, and removing every non-content word, the so-called "stop words"

```
cleaning <- tm_map(corpus, tolower)
```

```
## Warning in tm_map.SimpleCorpus(corpus, tolower): transformation drops documents
```

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

```
## Warning in tm_map.SimpleCorpus(cleaning, removeNumbers): transformation drops
## documents
```

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

```
## Warning in tm_map.SimpleCorpus(cleaning, removePunctuation): transformation
## drops documents
```

```
cleaning <- tm_map(cleaning, removeWords, stopwords("en"))
```

```
## Warning in tm_map.SimpleCorpus(cleaning, removeWords, stopwords("en")):
## transformation drops documents
```

```
cleaning <- tm_map(cleaning, stripWhitespace)
```

```
## Warning in tm_map.SimpleCorpus(cleaning, stripWhitespace): transformation drops
## documents
```

```
inspect(cleaning[1:10])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 10
##
##  [1]  everything need couldnt ask
##  [2]  movie terrible
##  [3]  seems completely secure holding belt keeping iphone inside
##  [4]  love cable allows connect miniusb device pc
##  [5]  didnt realize wonderful short really last two scenes
##  [6]  commercials misleading
##  [7]  trouble volume
##  [8] seriously worth wasting kids time
##  [9]  first time cant wait next
## [10]  colleague now get great receptiona little expensive performance great
```

The word cloud is as it follows

```
Positive_indices <- which(ratings$sentiment == "Positive")
Positive_indices[1:10]
```

```
##  [1]  1  3  4  5  9 10 14 15 16 19
```

```
Negative_indices <- which(ratings$sentiment == "Negative")
Negative_indices[1:10]
```

```
##  [1]  2  6  7  8 11 12 13 17 18 23
```

```
library(wordcloud)
wordcloud(cleaning[Positive_indices], min.freq=40, scale=c(3,.5))
```


```
wordcloud(cleaning[Negative_indices], min.freq=40)
```

The most repeated words with regards to Positive ratings are:

-Good

-Great

Other words worth mentioned are “excellent”, “best” or “nice”. In general, they are good, positive adjectives showing satisfaction regarding the product.

On the contrary, the most repeated words with respect to the negative reviews are:

-Bad

-Don’t

-Movie

The first two ones clearly have a negative connotation, while the last one shows that the most reviewed products are movies, (in fact, one of the three datasets consists only on reviews about movies). One curious fact is that the word “good” is among the most repeated ones in this Negative reviews word cloud. This is not a good indicator for the classification, since it shows positiveness, and should not be included in that group.

Now the dataset is going to be splited into a train and a test partitions (75% and 25% of the dataset, respectively), in order to create a Negative/Positive filter

```
ratings_train <- ratings[1:2250,]
ratings_test <- ratings[2251:3000,]
corpus_train <- cleaning[1:2250]
corpus_test <- cleaning[2251:3000]
```

Shall the frequency of terms be computed

```
ratings_sparse <- DocumentTermMatrix(cleaning)
inspect(ratings_sparse[1:10, 40:50])
```

```
## <<DocumentTermMatrix (documents: 10, terms: 11)>>
## Non-/sparse entries: 10/100
## Sparsity           : 91%
## Maximal term length: 11
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs colleague expensive get great little next now performance receptiona wait
##   1          0         0   0     0      0    0   0           0          0    0
##   10         1         1   1     2      1    0   1           1          1    0
##   2          0         0   0     0      0    0   0           0          0    0
##   3          0         0   0     0      0    0   0           0          0    0
##   4          0         0   0     0      0    0   0           0          0    0
##   5          0         0   0     0      0    0   0           0          0    0
##   6          0         0   0     0      0    0   0           0          0    0
##   7          0         0   0     0      0    0   0           0          0    0
##   8          0         0   0     0      0    0   0           0          0    0
##   9          0         0   0     0      0    1   0           0          0    1
```

```
sparse_train <- ratings_sparse[1:2250,]
sparse_test <- ratings_sparse[2251:3000,]
```

It is highly advisable to delete the terms appearing only a few times, as they do not really contribute with new useful information

```
five_times <- findFreqTerms(sparse_train, 5)
length(five_times)
```

```
## [1] 569
```

```
five_times[1:10]
```

```
##  [1] "couldnt"    "everything" "need"       "movie"      "terrible"
##  [6] "belt"       "completely" "inside"     "seems"      "cable"
```

```
sparse_train <- DocumentTermMatrix(corpus_train, control=list(dictionary = five_times))
sparse_test <- DocumentTermMatrix(corpus_test, control=list(dictionary = five_times))
```

The count info has to be converted to “Yes” and “No” info

```
convert_count <- function(x){
  y <- ifelse(x > 0, 1,0)
  y <- factor(y, levels=c(0,1), labels=c("No", "Yes"))
  y
}
sparse_train <- apply(sparse_train, 2, convert_count)
sparse_train[1:10, 40:50]
```

```
##     Terms
## Docs will cooked flavor perfectly steak classic fantastic characters portrayal
##   1  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   2  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   3  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   4  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   5  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   6  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   7  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   8  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   9  "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##   10 "No" "No"   "No"   "No"      "No"  "No"    "No"      "No"       "No"
##     Terms
## Docs yet  comes
##   1  "No" "No"
##   2  "No" "No"
##   3  "No" "No"
##   4  "No" "No"
##   5  "No" "No"
##   6  "No" "No"
##   7  "No" "No"
##   8  "No" "No"
##   9  "No" "No"
##   10 "No" "No"
```

```
sparse_test <- apply(sparse_test, 2, convert_count)
sparse_test[1:10, 40:50]
```

```
##     Terms
## Docs yet  best little funny absolutely heart vegas liked cheap feels looks
##   1  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   2  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   3  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   4  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   5  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   6  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   7  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   8  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   9  "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
##   10 "No" "No" "No"   "No"  "No"       "No"  "No"  "No"  "No"  "No"  "No"
```

And finally, the data is ready for the application of a Naive Bayes Classifier

```
library(e1071)

#Using the training data
classifier <- naiveBayes(sparse_train, ratings_train$sentiment)
class(classifier)
```

```
## [1] "naiveBayes"
```

```
#Make predictions on the test data
predictions <- predict(classifier, newdata=sparse_test)
table(predictions, ratings_test$sentiment)
```

```
##
## predictions Negative Positive
##    Negative      299      103
##    Positive       78      270
```

There are 78 Negative instances that were predicted as Positive, whereas 103 Positive instances were predicted to be Negative. This corresponds to a 79.32% and a 72.39% rate of success, which are not bad results, but need to be improved.

One way to do so is with the **Laplacian Smoothing** method: from uniform **prior** distributions, obtaining the so-called *posterior** distributions, and computing the Bayesian Naive Bayes with theses uniform prior distributions.

```
LaplaceClass <- naiveBayes(sparse_train, ratings_train$sentiment,laplace = 1)
class(LaplaceClass)
```

```
## [1] "naiveBayes"
```

```
LaplacePred <- predict(LaplaceClass, newdata=sparse_test)
table(LaplacePred, ratings_test$sentiment)
```

```
##
## LaplacePred Negative Positive
##    Negative      301       98
##    Positive       76      275
```

Only 2 instances previously regarded as Positive were correctly classified as Negative, and 5 wrongly regarded as Negative instances were correctly predicted to be Positive. That involves improvements of 0.52% and 1.33%, to rates of success of 79.84% and 73.72% respectively, not a huge change, and still improvable, but slightly better than before.