# Report

## Assignment 2: Text classification for the Detection of the Opinion Spam

## Abstract

## Introduction

Text classification is the process of assigning tags or categories to text according to its content. It’s one of the fundamental tasks in Natural Language Processing with broad applications such as sentiment analysis, topic labelling, spam detection, and intent detection. Unstructured data in the form of text is everywhere: emails, chats, web pages, social media, support tickets, survey responses, and more. Text can be an extremely rich source of information but extracting insights from it can be hard and time-consuming due to its unstructured nature. Businesses are turning to text classification for structuring text in a fast and cost-efficient way to enhance decision-making and automate processes. Text classification is the task of assigning a set of predefined categories to free-text. Text classifiers can be used to organize, structure, and categorize pretty much anything. In this project, we have used the prowess of text classification in deciphering which of the reviews are fake or not. Generally, we define review manipulation as publishers, writers, authors or company people or any third-party those who writing bad comments or feedback on behalf of customer when needed, to maximize their sales of productivity. So by analysing and concluding writing behaviour of customer we can identify fake reviews.

## The data

The dataset of fake and genuine hotel reviews has been collected from several popular online review communities. The fake reviews have been obtained from Mechanical Turk. All these data have been collected by Myle Ott and others [1,2]. There are 400 reviews in each of the categories: positive truthful, positive deceptive, negative truthful, negative deceptive. We focused on the negative reviews and deciphered between truthful and deceptive reviews. Hence, the total number of reviews in our data set is 800. Each category is then subsplitted into 5 folds.

### Training set

We used folds 1-4 (640 reviews) for training and hyperparameter tuning.

### Test set

Fold 5 (160 reviews) is used to estimate the performance of the classifiers that were selected on the

## Setup of the experiments

All the experiments have been conducted in R. In order to do that, it’s necessary to install an environment which allows to run R code. We have used Rstudio. Some additional packages have been installed to exploit their functions. The following commands should be typed first.

install.packages("tm")

install.packages("randomForest")

install.packages("entropy")

install.packages("randomForest")

install.packages("rpart")

install.packages("rpart.plot")

install.packages("glmnet")

Data can be downloaded from the following site:

<https://myleott.com/op-spam.html>

Then the following variables have been created in the Rstudio workspace. The four variables represent respectively the corpus of fake reviews in the training set, the corpus of the fake reviews in the training set, the corpus of fake reviews in the test set, the corpus of real reviews in the test set.

training.corpusdec <- c("C: --address of the folder -- dop\_spam\_v1.4/negative\_polarity/deceptive\_from\_MTurk/fold1","C: --address of the folder -- op\_spam\_v1.4/negative\_polarity/deceptive\_from\_MTurk/fold2","C--address of the folder -- op\_spam\_v1.4/negative\_polarity/deceptive\_from\_MTurk/fold3","C--address of the folder -- op\_spam\_v1.4/negative\_polarity/deceptive\_from\_MTurk/fold4")

training.corpustrue<- c("C: --address of the folder --op\_spam\_v1.4/negative\_polarity/truthful\_from\_Web/fold2","C--address of the folder -- op\_spam\_v1.4/negative\_polarity/truthful\_from\_Web/fold2","C--address of the folder -- op\_spam\_v1.4/negative\_polarity/truthful\_from\_Web/fold3","C--address of the folder -- op\_spam\_v1.4/negative\_polarity/truthful\_from\_Web/fold4")

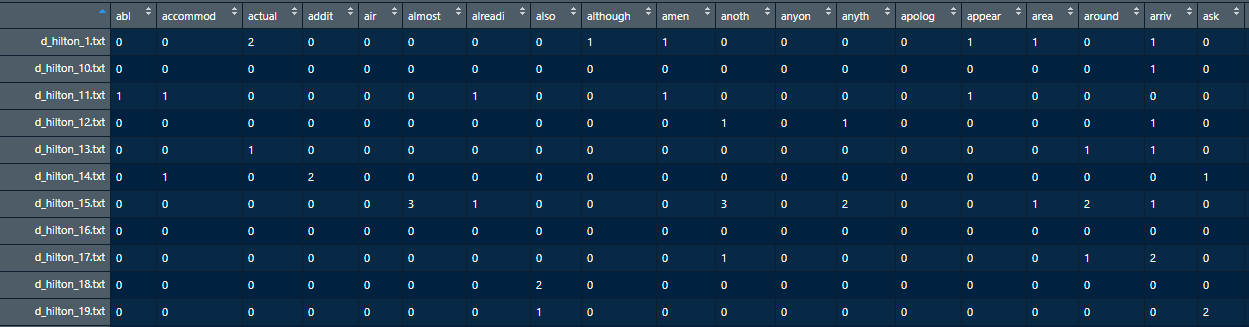
test.corpusdec <- "C--address of the folder -- op\_spam\_v1.4/negative\_polarity/deceptive\_from\_MTurk/fold5"

test.corpustrue <- C:/ --address of the folder -- op\_spam\_v1.4/negative\_polarity/truthful\_from\_Web/fold5"

First, with the use of Natural language processing, we cleaned the data. We used tm package, to

* Convert the alphabets to lower case
* Remove numbers
* Remove stopwords
* Remove whitespaces

We then converted the corpus to a sparse matrix, using DocumentTermMatrix, so that it looks like this:



After this, to limit the number of features, we removed words which were only in 5% of the total number of documents.

## Description of the experiments without Bigrams

### Naïve Bayes

The Naïve Bayes analysis has been conducted both with no kind of feature selection and with feature selection with MI. Still, the best results are obtained with no feature selection. As a comparison, in appendix 2 there is the confusion matrix selecting the best 50 features according to mutual information. In the code (Appendix 1) still there is the computation of mutual information, so that it is possible to modify it to select features imply changing the following line

predictions <- predict.mnb(train.mnb(training.dtm[, ], training.labels), test.dtm[, ])

with the following (given N the number of features we want to select)

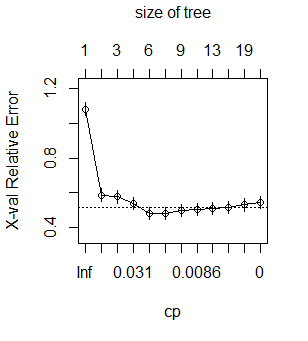
predictions <- predict.mnb(train.mnb(training.dtm[,training.mi.order[1:N ], training.labels), test.dtm[,training.mi.order[1:N]])

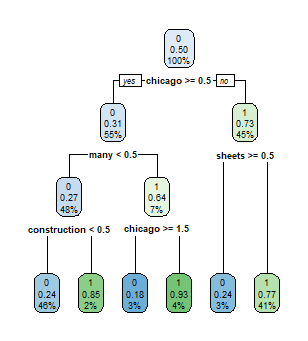
### Logistic Regression

The cv.glmnet function performs a cross validation on the lambda hyperparameter, but not on alpha. No other way of cross validation is performed. To reproduce the experiment, see Appendix 5.

### Classification Trees

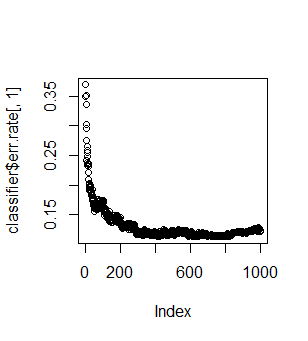
Some improvements have been made by using cross-validation to select the best value of the cost complexity pruning parameter: according to the following graph, the best value results to be 0.0171875. To reproduce the experiment, see Appendix 3.



This is the result tree. The most important features are Chicago, Many and Sheets. 

### Random Forests

An out of bag evaluation has been carried out to find the best hyperparameters. The function RFtune has been used to find the best number of randomly selected features per each split. The number is 5 (while the standard number for the Random Forest packages is 17). Then, an out of bag evaluation has been carried out to find the best number of trees (between 1 and 1000). Hereunder, there is the graph which represents the results. “Index” stays for the number of trees used, while “classifier$err.rate[,1]” stays for the OOB error. The best number of trees is 415. To reproduce the experiment, see Appendix 5.



## Numerical Results without Bigram features

### Analysis 1 (Naïve Bayes)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 65 | 14 |
| **the review** | Truthful | 15 | 66 |

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,81875 | 0,8228 | 0,8125 | 0,8176 |

### Analysis 2 (logistic regression)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 59 | 12 |
| **The review** | Truthful | 21 | 69 |

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,8000 | 0,8310 | 0,7375 | 0,7815 |

### Analysis 3 (classification trees)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 42 | 16 |
| **The review** | Truthful | 38 | 64 |

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,6625 | 0.7241 | 0,5250 | 0,6087 |

### Analysis 4 (random forests)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 69 | 24 |
| **The review** | Truthful | 11 | 56 |

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,7812 | 0,7420 | 0,8625 | 0,7977 |

## Experiments with Bigram features

## Numerical Results with Bigram features

## Discussion of the results

Random forests do not improve the performances of the linear classifiers, according to the data.

The main difference between the generative linear model (naïve Bayes) and the discriminative linear model (logistic regression) is the recall value: there is a significative number of deceptive reviews that the discriminative linear model is able to classify correctly, while the generative linear model is not able to do so.

## Appendix 1: Code for Naïve Bayes analysis

library(tm)

library(entropy)

library (SnowballC)

naive.bayes.function <- function (training.corpus.dec, training.corpus.true, testing.corpus.dec, testing.corpus.true){

#training set

training.dtm <- cleaning.function(training.corpus.dec,training.corpus.true)

training.dtm <- DocumentTermMatrix(training.dtm)

training.dtm <- removeSparseTerms(training.dtm,0.95)

training.dtm = as.matrix(training.dtm)

training.labels <- c(rep(0,320),rep(1,320))

#test set

test.dtm <- DocumentTermMatrix(cleaning.function (testing.corpus.dec,testing.corpus.true),list(dictionary=dimnames(training.dtm)[[2]]))

test.dtm = as.matrix(test.dtm)

test.labels <- c(rep(0,80),rep(1,80))

#feature selection

training.mi <- apply(training.dtm,2,function(x,y){

mi.plugin(table(x,y)/length(y))},training.labels)

training.mi.order <- order(training.mi,decreasing = T)

#predicting

predictions <- predict.mnb(train.mnb(training.dtm[, ], training.labels), test.dtm[, ])

table (predictions,test.labels)

}

#Training function for Naive Bayes

#labels = classes

train.mnb <- function (dtm,labels) {

call <- match.call()

V <- ncol(dtm) #vocabulary

N <- nrow(dtm) #number of documents

prior <- table(labels)/N

labelnames <- names(prior)

nclass <- length(prior)

cond.probs <- matrix(nrow=V,ncol=nclass)

dimnames(cond.probs)[[1]] <- dimnames(dtm)[[2]]

dimnames(cond.probs)[[2]] <- labelnames

index <- list(length=nclass)

for(j in 1:nclass){

index[[j]] <- c(1:N)[labels == labelnames[j]]

}

for(i in 1:V){

for(j in 1:nclass){

cond.probs[i,j] <- (sum(dtm[index[[j]],i])+1)/(sum(dtm[index[[j]],])+V)

#Laplace smoothing

}

}

x <- list(call=call,prior=prior,cond.probs=cond.probs)

return (x)

}

predict.mnb <- function (model,dtm) {

classlabels <- dimnames(model$cond.probs)[[2]]

logprobs <- dtm %\*% log(as.matrix(model$cond.probs))

N <- nrow(dtm) #number of documents to classify

nclass <- ncol(model$cond.probs) #number of classes

logprobs <- logprobs+matrix(nrow=N,ncol=nclass,log(model$prior),byrow=T)

x <- classlabels[max.col(logprobs)]

return (x)

}

cleaning.function <- function(corpus.dec, corpus.true){

reviews.dec <-VCorpus(DirSource(corpus.dec,encoding="UTF-8"))

reviews.true<-VCorpus(DirSource(corpus.true,encoding="UTF-8"))

review.all<-c(reviews.dec,reviews.true)

#clean the data

review.all <- tm\_map(review.all, content\_transformer(tolower))

review.all <- tm\_map(review.all, removeNumbers)

review.all <- tm\_map(review.all, removePunctuation)

review.all <- tm\_map(review.all, stripWhitespace)

review.all <- tm\_map(review.all,removeWords,stopwords("english"))

return (review.all)

}

## Appendix 2: Confusion matrix selecting the best 50 features according to MI

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 63 | 20 |
| **the review** | Truthful | 17 | 60 |

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,7687 | 0,7590 | 0,7875 | 0,7730 |

## Appendix 3: Code for classification trees analysis

library(tm)

library(entropy)

library (SnowballC)

library(rpart)

library(rpart.plot)

classification.function <- function (training.corpus.dec, training.corpus.true, testing.corpus.dec, testing.corpus.true){

#training set

training.dtm <- cleaning.function(training.corpus.dec,training.corpus.true)

training.dtm <- DocumentTermMatrix(training.dtm)

training.dtm <- removeSparseTerms(training.dtm,0.95)

training.dtm = as.matrix(training.dtm)

training.labels <- c(rep(0,320),rep(1,320))

#test set

test.dtm <- DocumentTermMatrix(cleaning.function (testing.corpus.dec,testing.corpus.true),list(dictionary=dimnames(training.dtm)[[2]]))

test.dtm = as.matrix(test.dtm)

test.labels <- c(rep(0,80),rep(1,80))

# grow the tree

reviews.rpart <- rpart(label~.,

data=data.frame(training.dtm,label = training.labels), cp=0,method="class")

# tree with lowest cv error

cp <- reviews. rpart$cptable[which.min(reviews.rpart$cptable[,"xerror"]),"CP"]

print(cp)

plotcp(reviews.rpart)

print(reviews.rpart$cptable)

reviews.rpart.pruned <- prune(reviews.rpart,cp = reviews.rpart$cptable[which.min(reviews.rpart$cptable[,"xerror"]),"CP"] )

rpart.plot(reviews.rpart.pruned)

# make predictions on the test set

reviews.rpart.pred <- predict(reviews.rpart.pruned,

newdata=data.frame(as.matrix(test.dtm)),type="class")

# show confusion matrix

table(reviews.rpart.pred,test.labels)

}

cleaning.function <- function(corpus.dec, corpus.true){

reviews.dec <-VCorpus(DirSource(corpus.dec,encoding="UTF-8"))

reviews.true<-VCorpus(DirSource(corpus.true,encoding="UTF-8"))

review.all<-c(reviews.dec,reviews.true)

#clean the data

review.all <- tm\_map(review.all, content\_transformer(tolower))

review.all <- tm\_map(review.all, removeNumbers)

review.all <- tm\_map(review.all, removePunctuation)

review.all <- tm\_map(review.all, stripWhitespace)

review.all <- tm\_map(review.all,removeWords,stopwords("english"))

return (review.all)

}

## Appendix 4: Code for random forests analysis

library(tm)

library(randomForest)

random.forest.function <- function (training.corpus.dec, training.corpus.true, testing.corpus.dec, testing.corpus.true){

#training set

training.dtm <- cleaning.function(training.corpus.dec,training.corpus.true)

training.dtm <- DocumentTermMatrix(training.dtm)

training.dtm <- removeSparseTerms(training.dtm,0.95)

training.dtm = as.matrix(training.dtm)

training.labels <- c(rep(0,320),rep(1,320))

#test set

test.dtm <- DocumentTermMatrix(cleaning.function (testing.corpus.dec,testing.corpus.true),list(dictionary=dimnames(training.dtm)[[2]]))

test.dtm <- as.matrix(test.dtm)

test.labels <- c(rep(0,80),rep(1,80))

training.dtm <- as.data.frame(training.dtm)

test.dtm <- as.data.frame(test.dtm)

training.dtm$label <- training.labels

training.dtm$label <- factor(training.dtm$label, levels = c(0, 1))

OOB.matrix <- tuneRF(x = training.dtm[-318],

y = training.dtm$label,

ntreeTry = 500, doBest = FALSE)

optimal.mtry <- OOB.matrix[which.min(OOB.matrix[,2]),1]

classifier <- randomForest(x = training.dtm[-318],

y = training.dtm$label, ntree = 1000,

mtry = optimal.mtry, type = "classification", err.rate = TRUE)

error\_rates <- classifier$err.rate[,1]

plot(classifier$err.rate[,1])

error\_rates <- cbind(error\_rates, c(1:1000))

optimal\_ntree <- error\_rates[which.min(error\_rates[,1]), 2]

classifier <- randomForest(x = training.dtm[-318],

y = training.dtm$label, mtry = optimal.mtry,

ntree = optimal\_ntree, type = "classification", err.rate = TRUE)

# Predicting the Test set results

test.dtm <- as.data.frame(test.dtm)

predictions <- predict (classifier, newdata = test.dtm)

table(predictions,test.labels)

}

cleaning.function <- function(corpus.dec, corpus.true){

reviews.dec <-VCorpus(DirSource(corpus.dec,encoding="UTF-8"))

reviews.true<-VCorpus(DirSource(corpus.true,encoding="UTF-8"))

review.all<-c(reviews.dec,reviews.true)

#clean the data

review.all <- tm\_map(review.all, content\_transformer(tolower))

review.all <- tm\_map(review.all, removeNumbers)

review.all <- tm\_map(review.all, removePunctuation)

review.all <- tm\_map(review.all, stripWhitespace)

review.all <- tm\_map(review.all,removeWords,stopwords("english"))

return (review.all)

}

## Appendix 5: Code for logistic regression analysis

library("glmnet")

logistic.function <- function (training.corpus.dec, training.corpus.true, testing.corpus.dec, testing.corpus.true){

#training set

training.dtm <- cleaning.function(training.corpus.dec,training.corpus.true)

training.dtm <- DocumentTermMatrix(training.dtm)

training.dtm <- removeSparseTerms(training.dtm,0.95)

training.dtm = as.matrix(training.dtm)

training.labels <- c(rep(0,320),rep(1,320))

#test set

test.dtm <- DocumentTermMatrix(cleaning.function (testing.corpus.dec,testing.corpus.true),list(dictionary=dimnames(training.dtm)[[2]]))

test.dtm = as.matrix(test.dtm)

test.labels <- c(rep(0,80),rep(1,80))

reviews.glmnet <- cv.glmnet(training.dtm,training.labels,

family="binomial",type.measure="class")

print (coef(reviews.glmnet,s="lambda.1se"))

reviews.logreg.pred <- predict(reviews.glmnet,

newx=test.dtm,s="lambda.1se",type="class")

table(reviews.logreg.pred,test.labels)

}

cleaning.function <- function(corpus.dec, corpus.true){

reviews.dec <-VCorpus(DirSource(corpus.dec,encoding="UTF-8"))

reviews.true<-VCorpus(DirSource(corpus.true,encoding="UTF-8"))

review.all<-c(reviews.dec,reviews.true)

#clean the data

review.all <- tm\_map(review.all, content\_transformer(tolower))

review.all <- tm\_map(review.all, removeNumbers)

review.all <- tm\_map(review.all, removePunctuation)

review.all <- tm\_map(review.all, stripWhitespace)

review.all <- tm\_map(review.all,removeWords,stopwords("english"))

return (review.all)

}

## References

[1] Myle Ott, Yejin Choi, Claire Cardie and Jerey T. Hancock, Finding deceptive opinion spam by any stretch of the imagination. Proceedings of the 49th meeting of the association for computational linguistics, pp. 309-319,2011.

[2] Myle Ott, Claire Cardie and Jerey T. Hancock, Negative deceptive opinion

spam. Proceedings of NAACL-HLT 2013, pp. 497-501, 2013.