# Report

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

## Abstract

[write here the results]

## Introduction

The habit of reviewing products and services on online platforms ( Tripadvisor, Yelp, Rate Your Music, …) is increasing very quickly. In parallel, according to (Cone, 2011) users are relying more and more on others’ reviews when it comes to make a decision. Due to that, not all the reviews are truthful, but some of them are deliberately manufactured to sound authentic by malicious people who want to gain more customers against competitors. We call these fictitious reviews DECEPTIVE OPINION SPAM. Furthermore, we distinguish between them depending on the general *sentiment* a review expresses: positive reviews express satisfaction about the product or the service, while negative reviews express disappointment. We are aware that this distinction excludes all the possible intermediate evaluations, so it’s a strong limit for our analysis ( following the normal categorization of web reviews, we are considering only 5-star and 1-star or 2-star reviews).

Previous works have already tried to build automatic classifiers which help to find out deceptive reviews and showed that they perform better than chance and human judge. In particular, Ott et al. (2011) focused on positive deceptive reviews, while Ott et al. (2013) focused on negative deceptive reviews. An important contribution of these researches is the creation of two datasets, respectively of gold-standard deceptive positive reviews and gold standard deceptive negative reviews. Indeed, the biggest obstacle to analysis before these two papers was the absence of sample data (reviews in our case) which can represent in an unbiased manner the deceptive reviews, since it’s complicated even to identify (at a certainty level of 100%) deceptive reviews, for obvious reasons. Thanks to that, further analysis is now possible exploiting those data.

We are trying now to improve the performance of the linear classifiers (naïve Bayes and Support Vector Machine) of the two previous works exploring different possibilities via more flexible classifiers and trying to build the best models via hyperparameters tuning. Due to simplicity reasons, we will focus only on negative reviews, leaving the positive ones to future work.

Our experiment is then divided into four parts, and each part takes into account a model. In details, we have:

1. Naïve Bayes
2. Regularized logistic regression
3. Classification tree
4. Random forests

For each model, we consider two sets generated by the reviews:

1. Only the unigrams
2. Unigrams and bigrams (set of all couples of two consecutive words)

## The data

As introduced before, we are exploiting the work of Ott et al. (2013) and using their data. Truthful reviews were collected from the following reviewing websites: Expedia, Hotels.com, Orbitz, Priceline, Tripadvisor and Yelp. Although they can’t be considered gold standard data, Mayzlin et al. (2012) and Ott et al. (2012) suggest that deception rate among those data is acceptably small. On the other side, deceptive opinion spam has been generated using Amazon’s Mechanical Turk service. The quality of these reviews has been proved by showing the actual difficulty of human testers in identifying them.

In details, the dataset is composed only of negative reviews (800). Data is equally divided into deceptive opinion spam (400) and truthful reviews (400). Each of these two sets is then divided into 5 folders. We use folders 1:4 of both deceptive and positive reviews (640 reviews in total) as training set, while we use folder 5 (160) as test set.

The reviews regard 20 popular hotels of Chicago, so that each hotel has 20 truthful reviews and 20 deceptive reviews. The truthful reviews were sampled according to a log normal distribution fit to the lengths of the deceptive reviews, since truthful reviews are on average longer than deceptive reviews. For further information about the collecting procedure and about Mechanical Tusk, refer to (Ott et al., 2013), the original paper.

## Setup of the experiments

All the experiments have been conducted in R language. 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 in the analysis. The following commands should then be typed first in the command line.

install.packages("tm")

install.packages("entropy")

install.packages("randomForest")

install.packages("randomForest")

install.packages("rpart")

install.packages("rpart.plot")

install.packages("glmnet")

Data can be downloaded from the following website:

<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 deceptive reviews in the training set, the corpus of the truthful reviews in the training set, the corpus of deceptive reviews in the test set, the corpus of truthful reviews in the test set. Be careful to substitute in the commands the real address of the folder which contains the data (usually, the Download folder).

training.corpus.dec <- c("C:/--address of the folder --op\_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.corpus.true<- 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")

testing.corpus.dec <- "C:/--address of the folder -- op\_spam\_v1.4/negative\_polarity/deceptive\_from\_MTurk/fold5"

testing.corpus.true <- “C:/ --address of the folder -- op\_spam\_v1.4/negative\_polarity/truthful\_from\_Web/fold5"

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

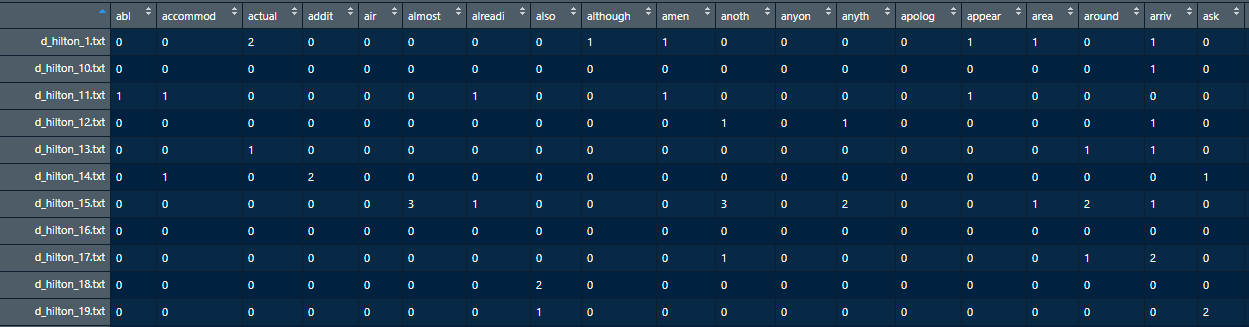
* Convert the entire text to lower case
* Remove numbers
* Remove whitespaces
* Remove punctuation
* Remove stopwords

In order to do that, we create a new Rscript with a function we call cleaning.function, to reuse it more times. The code of this function can be found in Appendix1. It unites the two matrices but does not mix deceptive and truthful reviews and returns a list of documents of words.

## Description of the experiments

### Naïve Bayes

For order purposes, we create a Rscript called Naïve Bayes and put all the code inside the function naïve.bayes.function. We clean the data using the cleaning function already mentioned, we extract all the unigrams, we remove all the words which appear in no more than 5% of the documents (“sparse terms”), we create a matrix. Here there is a part of it, just to show how data are represented in the dataset we are manipulating. We have the words as columns and the reviews as rows: each value (i,j) is 1 if the j-th word appears in the i-th review.



We do the same with bigrams. We have 307 unigrams and 12 bigrams. The bigrams are

"called front" "chicago hotel" "customer service" "even though"

"front desk" "got room" "hard rock" "hotel chicago"

"never stay" "room service" "stay hotel" "will never"

We train three models:

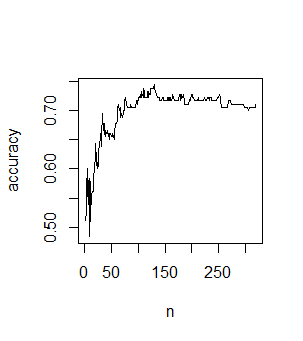
1. the first is built with all and only unigrams
2. the second is built with both bigrams and unigrams;
3. for the third, we compute the mutual information (MI) every feature (unigram or bigram) has and order the features according to the MI value. We then build 318 different models all with a different number of features, selecting each time the n best features according to the ordering, n ∈ [2,319]. We select the one with the highest accuracy, corresponding to n = 130. For example, these are the best 10 and their values:

chicago location smell luxury hotel chicago

0.09752866 0.03632829 0.03275744 0.03246057 0.03018794

chicago hotel decided recently finally millennium

0.02797741 0.02588616 0.02519993 0.02244856 0.02215038

The following graph shows the value of the accuracies depending on n. 

The results are reported with the following contingency tables

1. Only unigrams

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,7222 | 0,8289 | 0,7875 | 0,8077 |

1. Both unigrams and digrams

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 63 | 13 |
| **the review** | Truthful | 17 | 67 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 61 | 13 |
| **the review** | Truthful | 19 | 67 |

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,7111 | 0,8243 | 0,7625 | 0,7922 |

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | F1 score |
| 0,6555 | 0,7500 | 0,7125 |  |

1. Using only the most “useful” features

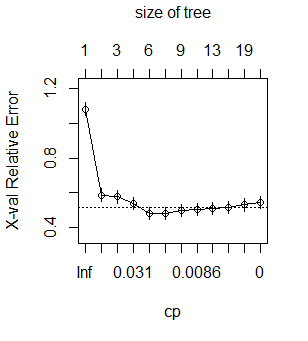
|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Review is actually** | |
|  |  | Deceptive | Truthful |
| **Model predicts** | Deceptive | 57 | 19 |
| **the review** | Truthful | 23 | 61 |

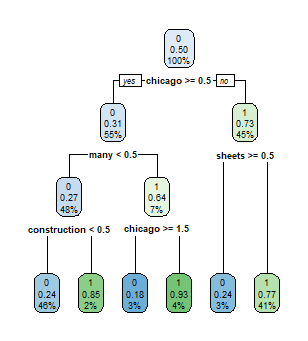
### Logistic Regression

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

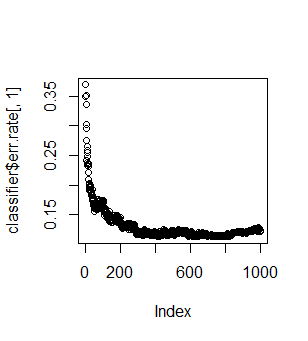
Some improvements have been made by using cross-validation to select the best value of the cost complexity pruning parameter: according to the following graph, the best value results to be 0.0171875. To reproduce the experiment, see Appendix 3.



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)

### 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 |

## 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: Cleaning function

library(tm)

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.

[3] Cone. 2011. 2011 Online Influence Trend Tracker.