2. Assignment - Mining insults

Melanija Kraljevska, Jana Štremfelj

1/7/2020

Reading data

```
options(warn=-1)
train <- read.table(file = 'insults/train.tsv', sep = '\t', header = TRUE)
test <- read.table(file = 'insults/test.tsv', sep = '\t', header = TRUE)
data <- rbind(train, test)
```

1. Cleaning

The function clean_data takes the given data as a parameter and returns a corpus which has punctuation, stopwords and unknown symbols removed. In this case, the parameter data is composed of both train and test files and then splited into training and testing set preserving the same documents as before.

```
library(ggplot2)
library(tm)
library(NLP)
library(openNLP)
clean_data <- function (data) {
  #Remove unknown symbols
  data$text_a <- sapply(data$text_a, function(x) gsub("x[[:alnum:]][[:digit:]]", "", as.character(x)))
  data$text_a <- sapply(data$text_a, function(x) gsub("\\n", "", as.character(x)))
  data$text_a <- sapply(data$text_a, function(x) gsub("\\r", "", as.character(x)))
  data$text_a <- sapply(data$text_a, function(x) gsub("\\t", "", as.character(x)))
  data$text_a <- sapply(data$text_a, function(x) gsub("\\u[[:digit:]][[:digit:]][[:digit:]][[:alnum:]]"

  corpus <- Corpus(VectorSource(data$text_a))

  #Remove punctuation and stopwords
  corpus <- tm_map(corpus, removeWords, stopwords('english'))
  conn = file("english.stop.txt", open="r")
  mystopwords = readLines(conn)
  close(conn)
  corpus <- tm_map(corpus, removeWords, mystopwords)
  corpus <- tm_map(corpus, removeNumbers)
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, stripWhitespace)

  #Anonymize proper nouns
  sent_ann <- Maxent_Sent_Token_Annotator()
  word_ann <- Maxent_Word_Token_Annotator()
  person_ann <- Maxent_Entity_Annotator(kind = "person")
  location_ann <- Maxent_Entity_Annotator(kind = "location")
  organization_ann <- Maxent_Entity_Annotator(kind = "organization")

  entities <- function(annots, kind)
  {
    k <- sapply(annots$features, `[[`, "kind")
```

```
    s[annots[k == kind]]
  }

  for (i in 1:length(corpus)){
    s <- as.String(content(corpus[[i]]))
    if(nchar(trimws(s)) == 0) next

    ann <- annotate(s, list(sent_ann, word_ann, person_ann, location_ann, organization_ann))

    corpus <- tm_map(corpus, removeWords, as.vector(entities(ann, "person")))
    corpus <- tm_map(corpus, removeWords, as.vector(entities(ann, "location")))
    corpus <- tm_map(corpus, removeWords, as.vector(entities(ann, "organization")))
    content(corpus[[i]])
  }
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, stemDocument)
  corpus <- tm_map(corpus, stripWhitespace)
  corpus
}
#get corpus for all data
corpus <- clean_data(data)
train_corpus <- corpus[1:dim(train)[1]]
#print first document of train corpus
content(train_corpus[[1]])
```

```
## [1] "xanax death blow stuff total danger build toler quick stop abrupt insidi take affect memori poi
```

```
test_corpus <- corpus[(dim(train)[1]+1):length(corpus)]
#print first document of test corpus
content(test_corpus[[1]])
```

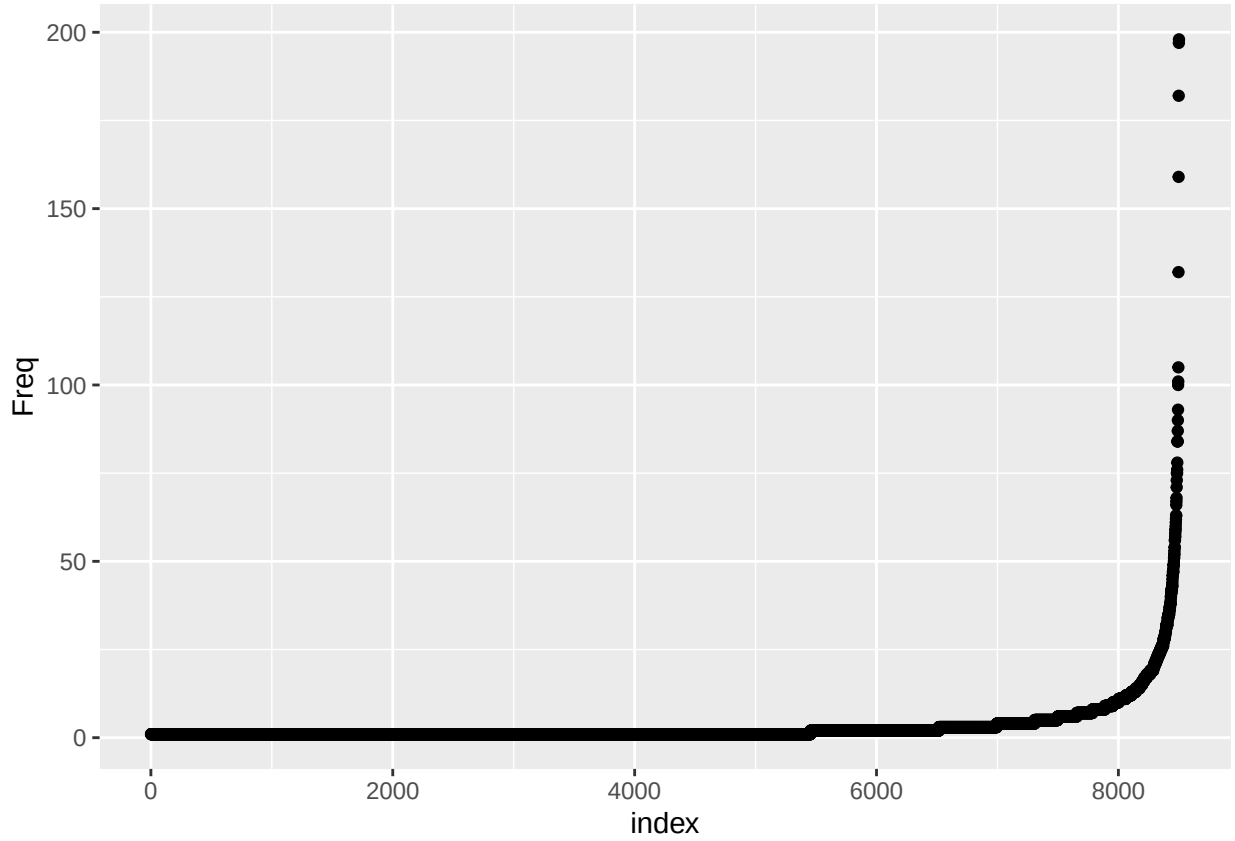
```
## [1] "wast breath paid shill entir propaganda program call megaphon"
```

2. Exploration

Frequency of words

In the graph below we can observe that a certain group of words is more frequently used. There are a lot of words that have a small occurence.The most frequently used words are shown in the data frame below.

```
tdm <- TermDocumentMatrix(train_corpus)
termFrequency <- rowSums(as.matrix(tdm))
#termFrequency <- subset(termFrequency, termFrequency >= 10)
v <- sort(rowSums(as.matrix(tdm)),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)
qplot(seq(length(termFrequency)),sort(termFrequency), xlab = "index", ylab = "Freq")
```




```
head(d, 10)
```

```
##        word freq
## fuck   fuck  198
## peopl peopl  197
## make   make  182
## dont   dont  159
## time   time  132
## year   year  105
## thing thing  101
## back   back  100
## good   good   93
## shit   shit   90
```

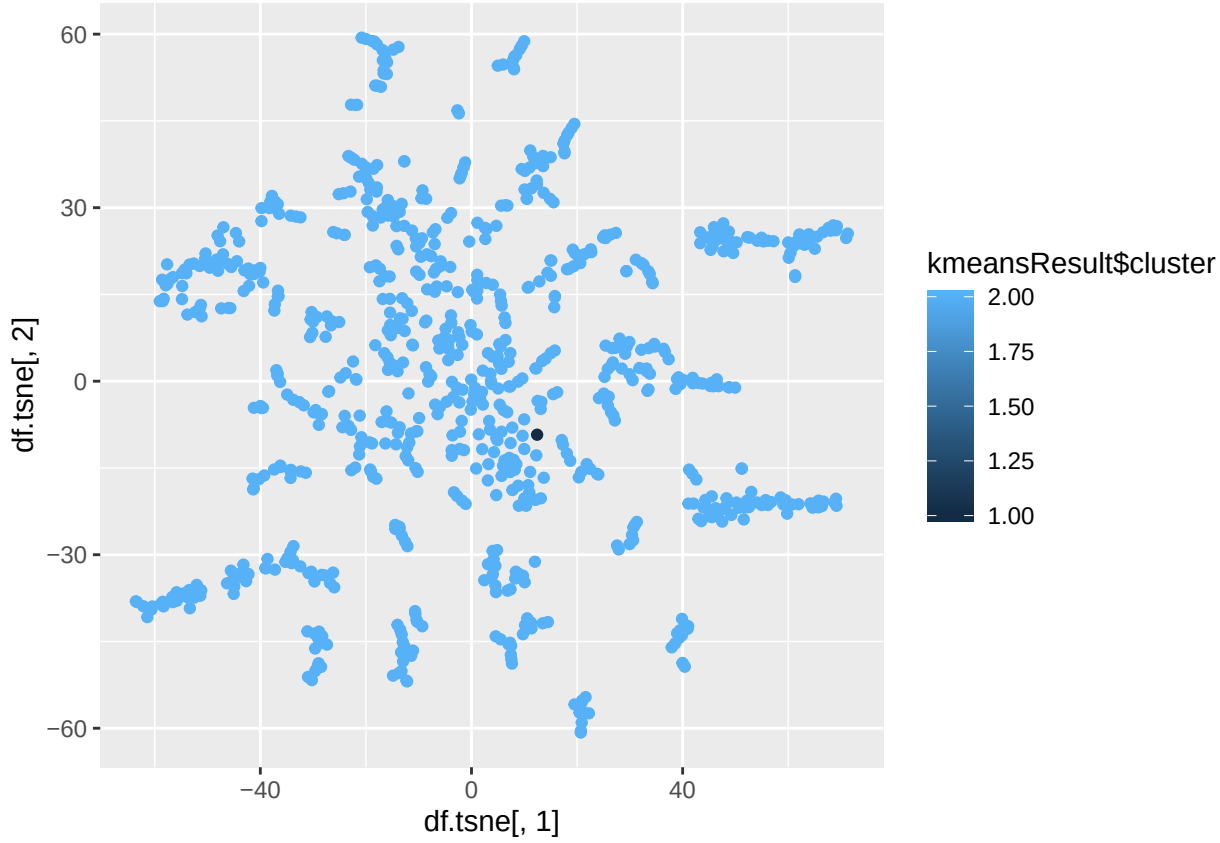
Clustering

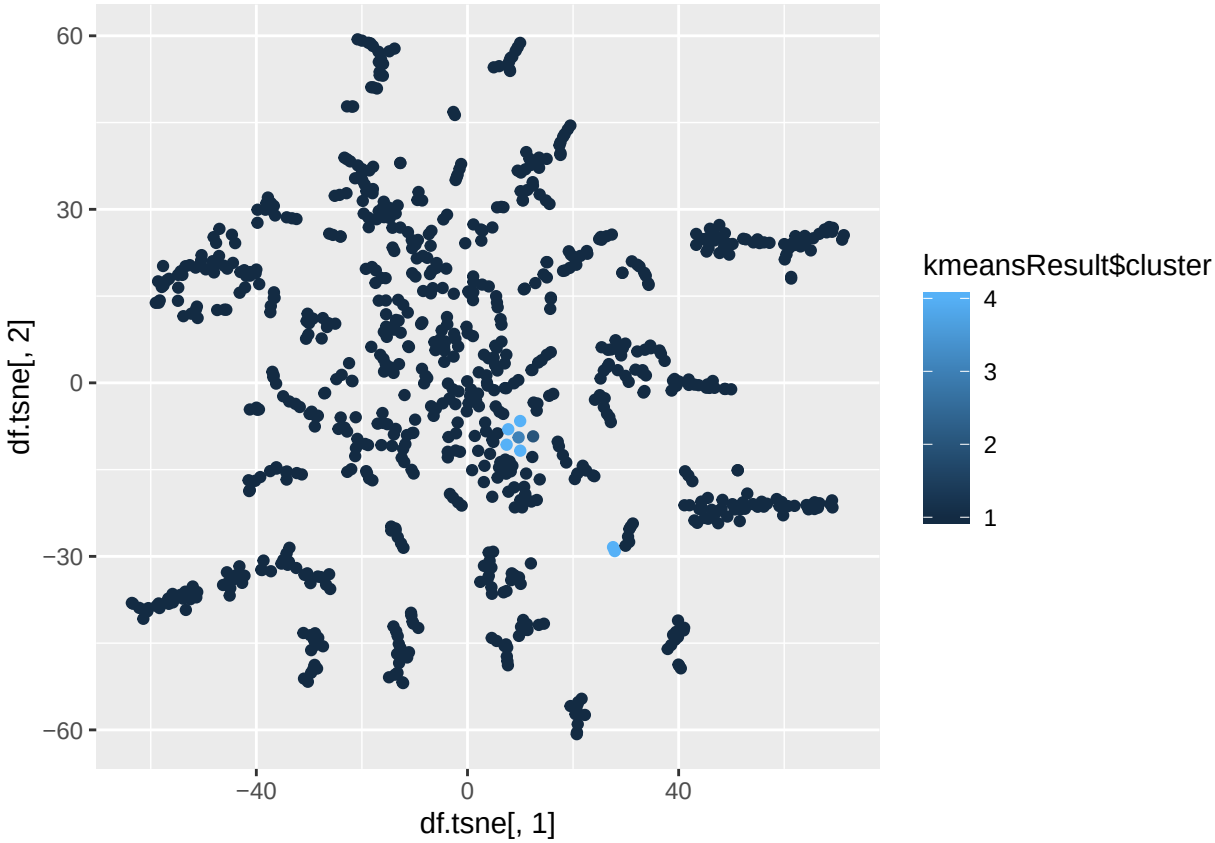
The first four graph represent the clustering for different value of the parameter k. The last graph shows the documents colored according to class labels. From the assigned clusters we can conclude that clustering works bad in this case.

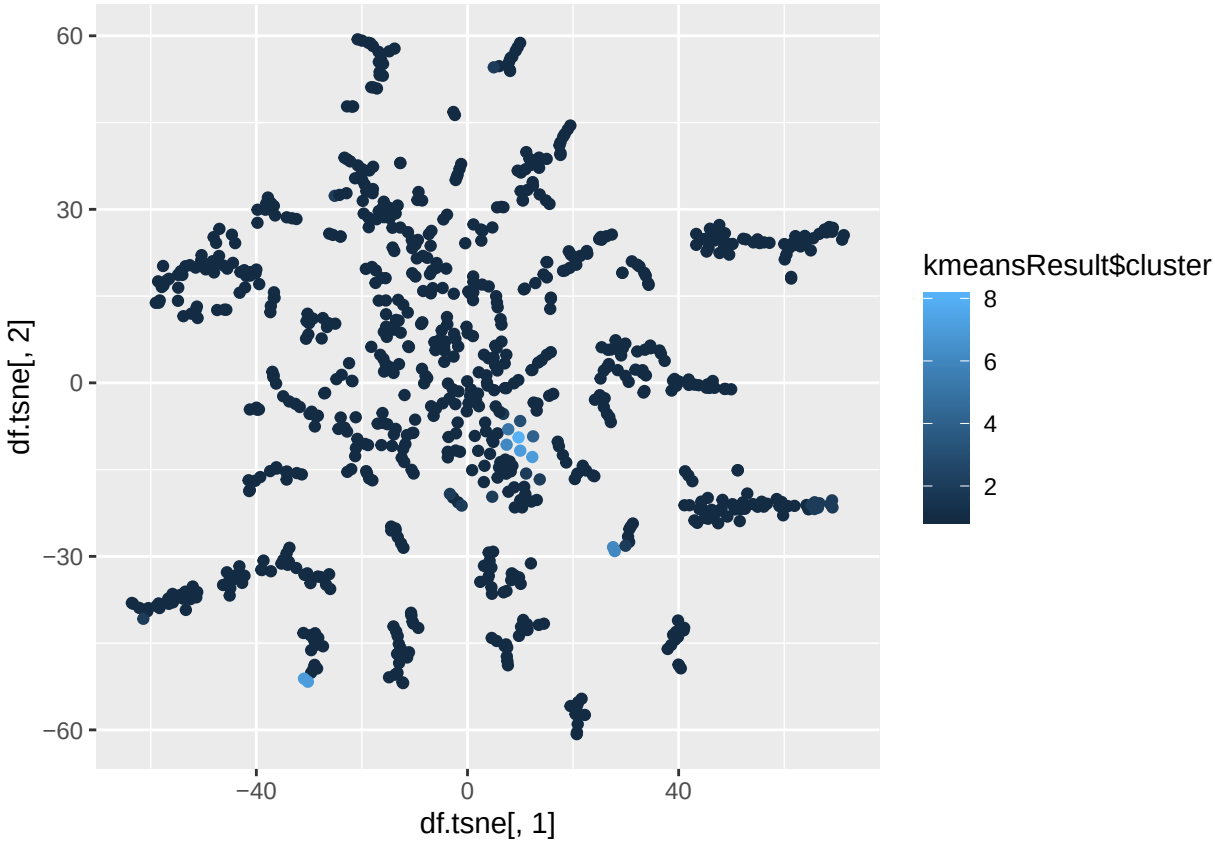
```
library(Rtsne)
dtm <- DocumentTermMatrix(train_corpus, control = list(weighting=weightTfIdf))
mat <- as.matrix(dtm)
tsne.proj <- Rtsne(mat, perplexity=10, theta=0.2, dims=2, check_duplicates = F)
df.tsne <-tsne.proj$Y
k <- c(2, 4, 8, 16)
for(i in 1:length(k)){
```

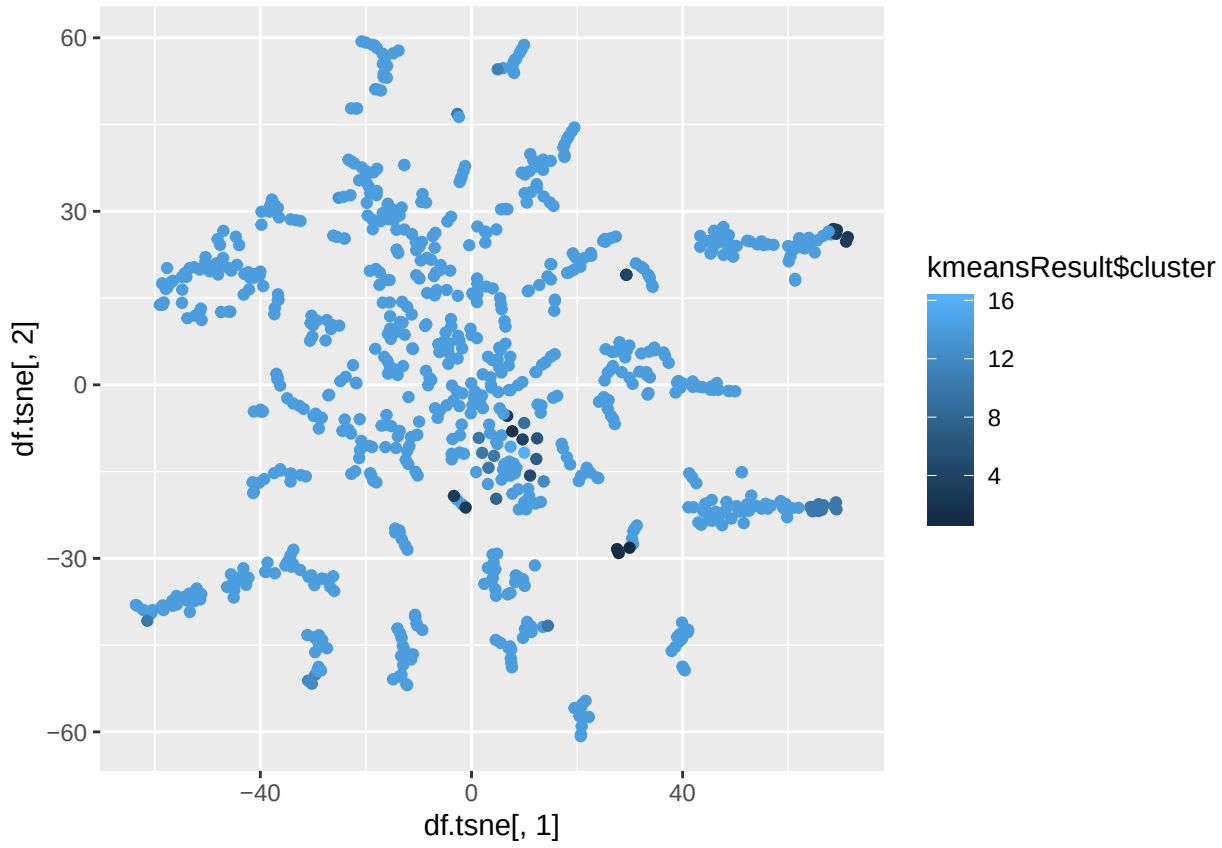
```
  kmeansResult <- kmeans(mat, k[i])
  #visualize the cluster assignments
  print(qplot(df.tsne[,1],df.tsne[,2], color = kmeansResult$cluster))

}
```

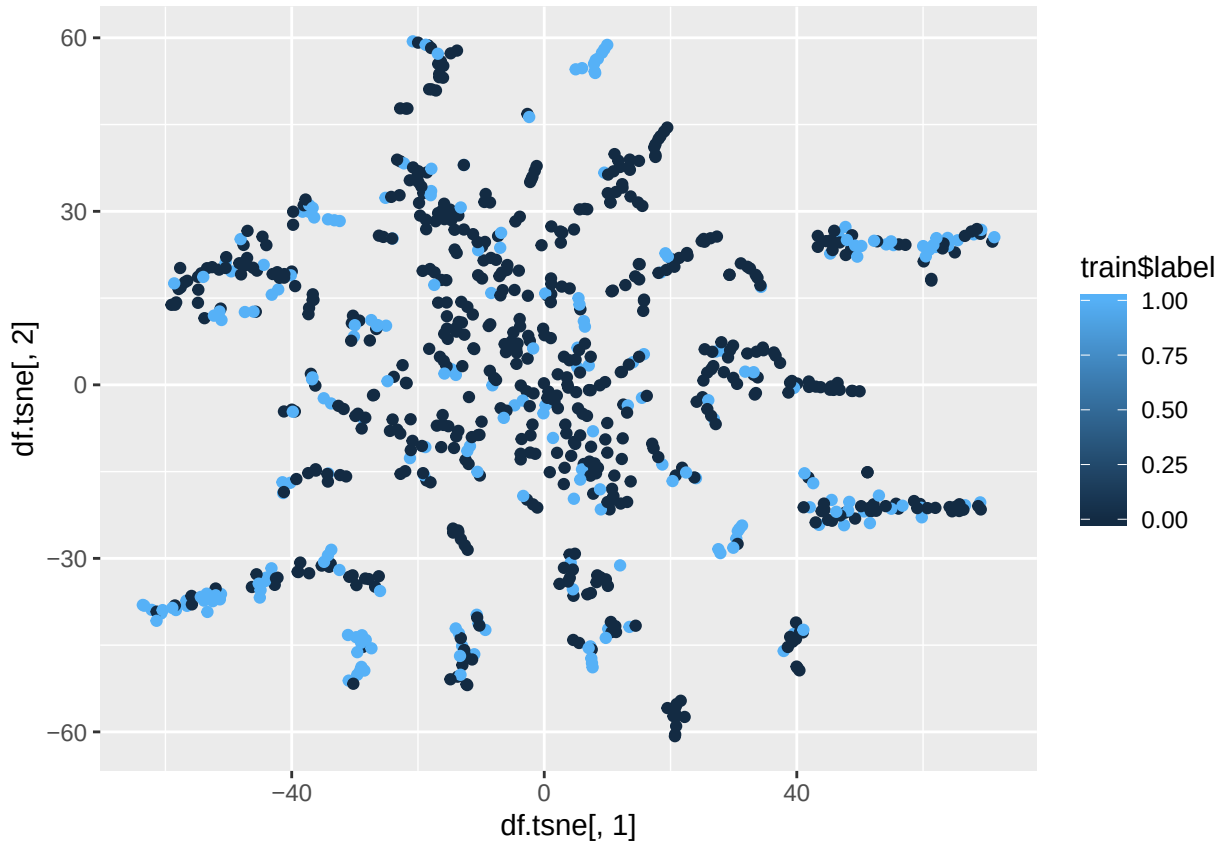



60
30
0
−30
−60
df.tsne[, 2]
−40
0
40
df.tsne[, 1]
kmeansResult$cluster
4
3
2
1


60
30
0
−30
−60
df.tsne[, 2]
−40
0
40
df.tsne[, 1]
kmeansResult$cluster
8
6
4
2




```
#plot document representations according to class labels
print(qplot(df.tsne[,1],df.tsne[,2], color = train$label))
```



Document representation with POS tags

```
#POS vector for each document
pos_ann <- Maxent_POS_Tag_Annotator()
word_ann <- Maxent_Word_Token_Annotator()
sent_ann <- Maxent_Sent_Token_Annotator()
posvectors <- list()
for(i in 1:length(corpus)){

  s <- as.String(content(corpus[[i]]))
  if(nchar(trimws(s)) == 0){
    posvectors[[i]] <- NULL
    next
  }
  a1 <- annotate(s, sent_ann)
  a2 <- annotate(s, word_ann, a1)
  a3 <- annotate(s, pos_ann, a2)
  a3w <- subset(a3, type == "word")
  tags <- sapply(a3w$features, `[[`, "POS")
  posvectors[[i]] <- tags
  #print(posvectors[[i]])
}
print(posvectors[[1]])
```

```
##  [1] "NN"  "NN"  "NN"  "NN"  "JJ"  "NN"  "VB"  "RB"  "JJ"  "NN"  "JJ"
## [12] "NN"  "VBP" "VB"  "FW"  "VBP" "VB"  "VBP" "JJ"  "NN"  "NNS" "VBP"
```

```
## [23] "NN"  "NNS" "VBP" "RP"  "IN"  "NN"  "NNS" "VBP" "JJ"  "JJ"  "NN"
## [34] "NN"  "NN"  "RB"  "VB"  "NN"  "JJ"  "NN"  "NN"  "JJ"  "NN"  "NN"
## [45] "NN"  "NN"  "NN"  "NN"
```

3. Model

Class label distribution

From the table we see that the ratio between the first and the second class is approximately 1:3, from which we can conclude that the dataset is imbalanced.

```
table(train$label)
```

```
##
##   0   1
## 592 233
```

```
table(test$label)
```

```
##
##   0   1
## 559 209
```

Classification models

Data representation

For the data representation, we have used document-term matrix composed of the corpus from both training and testing set and later splited the matrix to the two subsets as before. In that way the dtm matrix for both training and testing sets contains the same attributes - terms, which makes is appropriate for the modelling.

For perfomance metrics we have chosen accuracy and F1 score.

```
#Preparing data
dtm <- DocumentTermMatrix(corpus, control = list(weighting=weightTfIdf))
matrix <- cbind(as.matrix(dtm),data$label)
training_set <- matrix[1:dim(train)[1],-ncol(matrix)]
testing_set <- matrix[(dim(train)[1]+1):dim(matrix)[1],-ncol(matrix)]
train_data <- data.frame(matrix[1:dim(train)[1],])
names(train_data)[ncol(train_data)] <- "label_"
train_data$label_ <- make.names(train_data$label_)
```

KSVM - Support Vector Machines

SVM models perform well on text classification. They can handle high dimensional input space and sparse document vectors (the corresponding document vector contains only few entries which are not zero).

For this model we are using radial basis kernel, because linearly non-separable features often become linearly separable after they are mapped to a high dimensional feature space. However, we don't ever need to compute the feature mappings explicitly: we only need to work with their kernels, which are easier to compute.

```
library(mda)
library(modeltools)
library(mlr)

#Hyperparameter tuning
ksvm_task <- makeClassifTask(data = train_data, target = "label_")
discrete_ps <- makeParamSet(
  makeDiscreteParam("C", values = c(0.01, 0.05, 0.1,0.5, 1)),
```

```
  makeDiscreteParam("sigma", values = c(0.01, 0.05, 0.1,0.5, 0.6))
)
print(discrete_ps)
ctrl <- makeTuneControlGrid()
rdesc <- makeResampleDesc("CV", iters = 3L)
res <- tuneParams("classif.ksvm", ksvm_task , rdesc, measures=acc, par.set = discrete_ps, control = ctrl
```

```
library(kernlab)

# svm with a radial basis kernel
ksvm_model <- function(training_set, testing_set){
  print("Running model ksvm...")
  model.svm <- ksvm(training_set, make.names(as.factor(train$label)), kernel = "rbfdot")
  predicted <- predict(model.svm, testing_set, type = "response")
  t <- table(make.names(test$label), predicted)

  # Classification accuracy
  acc <- sum(diag(t))/sum(t)
  # Recall
  recall <- t[1,1]/sum(t[1,])
  # Precision
  precision <- t[1,1]/sum(t[,1])
  # F1 score
  f1 <- (2*recall*precision)/(precision+recall)
  cat("Accuracy: ",acc, "\n")
  cat("F1 score: ",f1, "\n")
  scores <- c(acc,f1)
  scores
}
scores_ksvm <- ksvm_model(training_set,testing_set)
```

```
## [1] "Running model ksvm..."
## Accuracy:  0.7486979
## F1 score:  0.8527841
```

GBM - Stochastic Gradient Boosting

Gradient boosting is one of the most powerful techniques for building predictive models, where decision trees are used as the weak learner. At each iteration a subsample of the training data is drawn at random (without replacement) from the full training dataset. The randomly selected subsample is then used, instead of the full sample, to fit the base learner.

```
library(caret)
library(gbm)
gbm_model <- function(training_set, testing_set){
  print("Running model gbm...")

  #Hyperparameter tuning
  man_grid <- expand.grid(n.trees = c(50, 100, 150),
                          interaction.depth = c(1, 3, 4),
                          shrinkage = 0.1,
                          n.minobsinnode = 10)

  control <- trainControl(method='cv',
                          number=3,
```

```
                          returnResamp='none',
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

  objModel <- caret::train(training_set,
                    make.names(train$label),
                    method='gbm',
                    trControl=control,
                    verbose = FALSE,
                    tuneGrid = man_grid)
  predictions <- predict(object=objModel,testing_set, type='raw')
  t <- table(make.names(test$label), predictions)
  # Classification accuracy
  acc <- sum(diag(t))/sum(t)
  # Recall
  recall <- t[1,1]/sum(t[1,])
  # Precision
  precision <- t[1,1]/sum(t[,1])
  # F1 score
  f1 <- (2*recall*precision)/(precision+recall)
  cat("Accuracy: ",acc, "\n")
  cat("F1 score: ",f1, "\n")
  scores <- c(acc,f1)
  scores
}
scores_gbm <- gbm_model(training_set,testing_set)
```

```
## [1] "Running model gbm..."
## Accuracy:  0.7695312
## F1 score:  0.8582866
```

The two models have approximately the same scores, both of them perfom well.

Adding POS tag-based representation

In order to consider the POS tags of the words in our data, each word in the data has been concatinated with its POS tag.

```
pos_concat <- list()
for(i in 1:2){
  words <- strsplit(content(corpus[[i]]), " ")
  pos_concat[[i]] <- paste(words[[1]],posvectors[[i]], sep = "", collapse = " ")
}
pos_corpus <- Corpus(VectorSource(pos_concat))
content(pos_corpus[[1]])
```

```
## [1] "xanaxNN deathNN blowNN stuffNN totalJJ dangerNN buildVB tolerRB quickJJ stopNN abruptJJ insidiNN
```

```
pos_dtm <- DocumentTermMatrix(corpus, control = list(weighting=weightTfIdf))
pos_matrix <- cbind(as.matrix(pos_dtm),data$label)
training_set_pos <- matrix[1:dim(train)[1],-ncol(matrix)]
testing_set_pos <- matrix[(dim(train)[1]+1):dim(matrix)[1],-ncol(matrix)]
```

The POS tagging did not have a significant improvement to the performance of the models. For the ksvm model almost no improvement, for the gbm model there is an improvement of 0.1 - 0.2.

```
# re-evaluating models with POS
scores_ksvm_pos <- ksvm_model(training_set_pos, testing_set_pos)
```

```
## [1] "Running model ksvm..."
## Accuracy:  0.7486979
## F1 score:  0.8527841
```

```
scores_gbm_pos <- gbm_model(training_set_pos, testing_set_pos)
```

```
## [1] "Running model gbm..."
## Accuracy:  0.7773438
## F1 score:  0.8594906
```

4. Understanding

Feature ranking

The method evaluates the quality of the features variables due to target variable. ##### filter method

```
N <- 20
estReliefF <- attrEval(label_ ~ ., train_data, estimator="InfGain", ReliefIterations=30)
 bestN <- head(sort(estReliefF, decreasing = TRUE), N)
write.table(bestN, "filter_stats.txt", append = FALSE, sep = " ",row.names = TRUE, col.names = TRUE)
```

The results of the function attrEval are stored in the file "filter_stats.txt", so we will read them from here.

```
library(dplyr)
library(COREelearn)
bestN <- read.table("filter_stats.txt", sep = " ")
training_set_bestN <- as.matrix(select(train_data,c(rownames(bestN))))
testing_set_bestN <- as.matrix(select(data.frame(testing_set),c(rownames(bestN))))
```

Re-evaluating models

```
scores_ksvm_bestN <- ksvm_model(training_set_bestN, testing_set_bestN)
```

```
## [1] "Running model ksvm..."
## Accuracy:  0.7669271
## F1 score:  0.8504595
```

```
scores_gbm_bestN <- gbm_model(training_set_bestN, testing_set_bestN)
```

```
## [1] "Running model gbm..."
## Accuracy:  0.7747396
## F1 score:  0.8569065
```

Wrapper model

For the wrapper method we used the Boruta library that selected important features vias random forest. The results from the Boruta method are stored in the file "wrapper_stats.txt".

```
library(randomForest)

train_data$label_ = replace(train_data$label_, train_data$label_=='X0', 0)
train_data$label_ = replace(train_data$label_, train_data$label_=='X1', 1)
train_data$label_ = factor(train_data$label_)

names(train_data)[names(train_data) == "shadowbeard"] <- "_shadowbeard"
```

```
names(train_data)[names(train_data) == "shadowshoss"] <- "_shadowshoss"

library("Boruta")
bor <- Boruta(label_~., data=train_data)


 plot(bor, cex.axis=.7, las=2, xlab="", main="Variable Importance")
 stats <- attStats(bor)
 write.table(stats[order(-stats$maxImp),], "wrapper_stats.txt", append = FALSE, sep = " ",row.names = TR
```

As we can see on the graph both models choose 5 to 15 similar most important features. They do not choose the same most important features if the number of chosen is less than five. If we extract more than 15 important features the models starts to choose different words because none of the them is later on very impornant.

```
N <- 20
stats <- read.table("wrapper_stats.txt", sep = " ")
bestWN <- head(stats[order(-stats$maxImp),], N)

filterNames <- rownames(bestN)
print("Most important terms from filter method")
```

```
## [1] "Most important terms from filter method"
```

```
print(filterNames)
```

```
##  [1] "idiot"  "dumb"   "moron"  "ass"    "bitch"  "stupid" "retard"
##  [8] "shut"   "loser"  "time"   "racist" "famili" "back"   "nigga"
## [15] "mouth"  "mother" "play"   "long"   "good"   "presid"
```

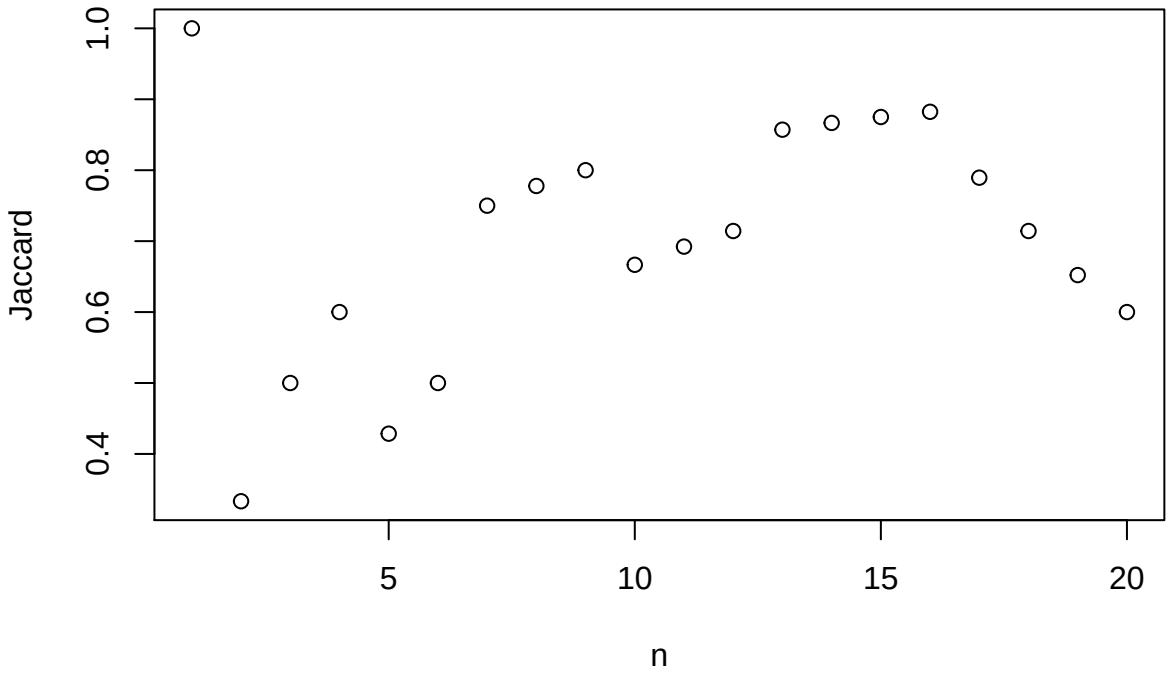
```
wrapperNames <- rownames(bestWN)
print("Most important terms from wrapper method")
```

```
## [1] "Most important terms from wrapper method"
```

```
print(wrapperNames)
```

```
##  [1] "idiot"  "moron"  "retard" "dumb"   "loser"  "stupid" "bitch"
##  [8] "shut"   "famili" "racist" "nigga"  "back"   "ass"    "mouth"
## [15] "mother" "piec"   "white"  "pathet" "exist"  "made"
```

```
# Jaccard
listJac <- c()
for (i in (1:N)){
  fil <- filterNames[1:i]
  wra <- wrapperNames[1:i]
  jac <- length(intersect(fil, wra)) / length(union(fil, wra))
  listJac <- c(listJac, jac)
}
plot(c(seq(1,N)), listJac, xlab="n", ylab="Jaccard")
```


Jaccard
0.4
0.6
0.8
1.0
5
10
15
20
n