

Assignment 2

Pavel Linder, Nikita Brancatisano

12/26/2019

0. Read input

```
train = read.table(file = 'train.tsv', sep = '\t', header = TRUE, stringsAsFactors = FALSE)
train.text = read.table(file = 'train.tsv', sep = '\t', header = TRUE, stringsAsFactors = FALSE)

test = read.table(file = 'test.tsv', sep = '\t', header = TRUE)
length(which(!complete.cases(train)))
```

```
## [1] 0
```

```
head(train$text_a)
```

```
## [1] "Xanax was her death blow. \xc2\xa0That stuff is totally dangerous because you
## [2] "you are both morons and that is never happening"
## [3] "you are just an idiot blabbermouth that is gonna get stopped HARD one day! You W
## [4] "how do the towers connect to the bottom pentagon? Since it's not flat..."
## [5] "I love Cam Newton's upside, and think he'll be an All-Pro caliber QB, but 21 TD'
## [6] "Eat shit and die Andrew"
```

1. Cleaning data

Remove punctuation and stopwords

```
train$text_a = as.character(train$text_a)
train$text_a = tm::removePunctuation(train$text_a)
train$text_a = tm::removeWords(x = train$text_a, stopwords(kind = "SMART"))
train$text_a = tm::stripWhitespace(train$text_a)
```

```
word_count <- lapply(train$text_a, wordcount)
length(which(word_count > 100))
```

```
## [1] 23
```

```
length(which(word_count < 100))
```

```
## [1] 802
```

```
length(which(word_count == 0))
```

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

```
length(train$text_a)
```

```
## [1] 825
```

```
train <- train[which(word_count < 100),]
word_count <- lapply(train$text_a, wordcount)
length(train$text_a)
```

```
## [1] 802
```

```
train <- train[which(word_count > 0),]
length(train$text_a)
```

```
## [1] 801
```

```
head(train$text_a)
```

```
## [1] "Xanax death blow xc2xa0That stuff totally dangerous build tolerance quickly stop abruptly xc2xa0"
## [2] " morons happening"
## [3] " idiot blabbermouth gonna stopped HARD day You WILL NOT saved"
## [4] " towers connect bottom pentagon Since flat"
## [5] "I love Cam Newtons upside hell AllPro caliber QB 21 TDs 17 Interceptions NFL Network 10th Heism"
## [6] "Eat shit die Andrew"
```

Anonymize proper nouns

```
n <- length(train$text_a)
word_ann <- Maxent_Word-Token-Annotator()
sent_ann <- Maxent_Sent-Token-Annotator()
pos_ann = Maxent_POS-Tag-Annotator()

for (i in 1:n) {
  while(1) {
    doc <- as.String(train$text_a[[i]])
    wordAnnotation <- annotate(doc, list(sent_ann, word_ann))
    POSAnnotation <- annotate(doc, pos_ann, wordAnnotation)
    POSWords <- subset(POSAnnotation, type == "word")
    POSTags <- vector()
    for (j in 1:length(POSWords$features))
      POSTags <- c(POSTags, POSWords$features[[j]]$POS)
    tokenPOS <- cbind(doc[POSWords], POSTags)
    ppn_idx <- which(tokenPOS[,2] == "NNP", 1)
    if (length(ppn_idx) == 0) {
      break;
    }
    words <- subset(wordAnnotation, type == "word")
    hashed <- digest(tokenPOS[ppn_idx, 1], "xxhash32")
    ppn <- words[ppn_idx]
    train$text_a[[i]] <- gsub(doc[ppn$start,ppn$end], hashed, doc)
  }
}

head(train$text_a)
```

```
## [1] "5dcac30f death blow xc2xa0That stuff totally dangerous build tolerance quickly stop abruptly xc2xa0"
## [2] " morons happening"
## [3] " idiot blabbermouth gonna stopped e8a1d6c8 day You WILL 91b0cb01 saved"
## [4] " towers connect bottom pentagon Since flat"
## [5] "I love a9350e16 cee1217a upside hell ea737b57 caliber 9c894fe8 21 1a620f48 17 Interceptions 914"
## [6] "d84ee5df shit die 2703f309"
```

```
n <- length(train$text_a)
```

Remove unknown symbols (non UTF-8 characters)

```
train$text_a <- iconv(train$text_a, to='UTF-8', sub='byte')
length(train$text_a)
```

```
## [1] 801
```

```
head(train$text_a)
```

```
## [1] "5dcac30f death blow xc2xa0That stuff totally dangerous build tolerance quickly stop abruptly xc"
## [2] " morons happening"
## [3] " idiot blabbermouth gonna stopped e8a1d6c8 day You WILL 91b0cb01 saved"
## [4] " towers connect bottom pentagon Since flat"
## [5] "I love a9350e16 cee1217a upside hell ea737b57 caliber 9c894fe8 21 1a620f48 17 Interceptions 914"
## [6] "d84ee5df shit die 2703f309"
```

2. Exploration

We are computing the TDM matrix from 2 corpuses: one with stemmization, one without. From the results we can see that the character of the corpus remains. Thus, we will use the stemmed corpus for the future evaluation.

I. Plot the frequency of words (without stemmization)

```
library(ggplot2)
```

```
##
```

```
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:NLP':
```

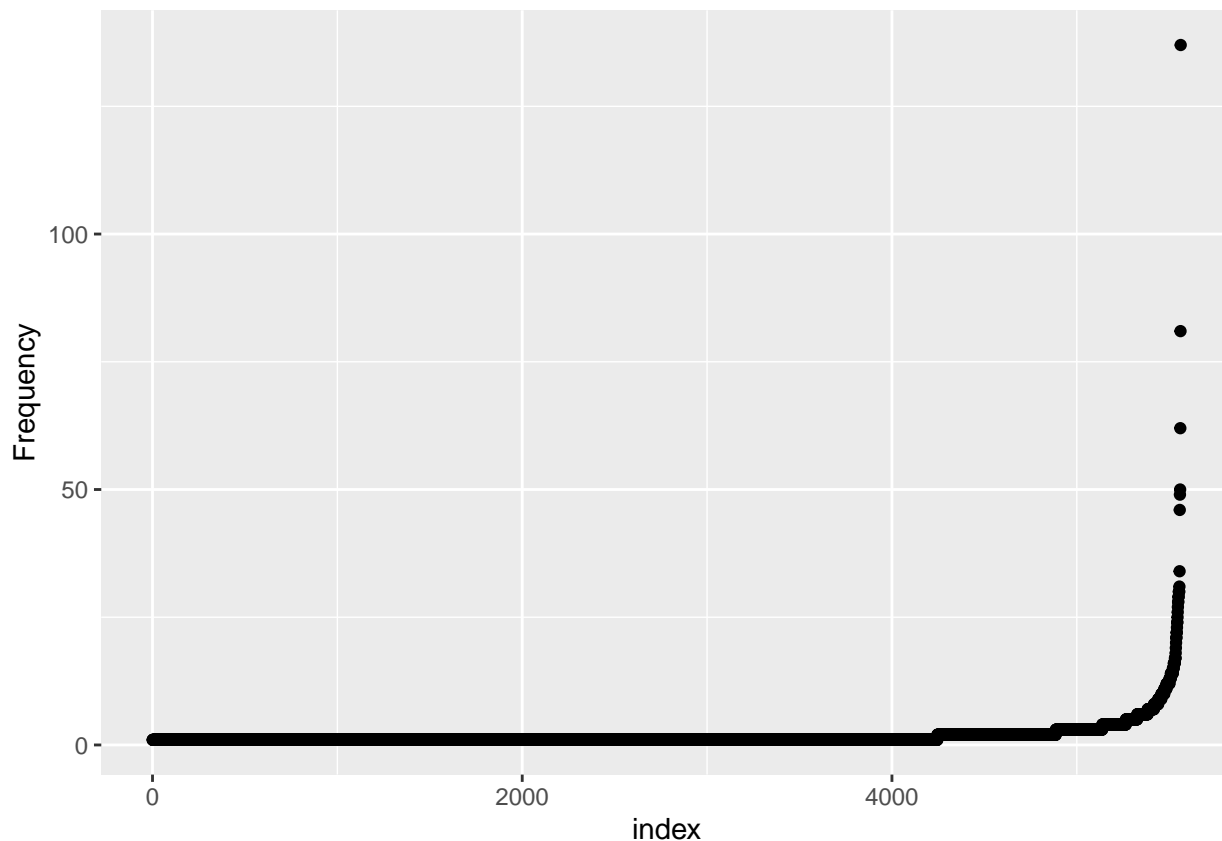
```
##
```

```
##      annotate
```

```
corpus <- Corpus(VectorSource(train$text_a)) # turn into corpus
tdm <- TermDocumentMatrix(corpus)
```

```
wordFreq <- sort(rowSums(as.matrix(tdm)), decreasing=TRUE)
```

```
qplot(seq(length(wordFreq)),sort(wordFreq), xlab = "index", ylab = "Frequency")
```



```
findFreqTerms(tdm, lowfreq=50)
```

```
## [1] "dont" "you" "the" "people"
```

```
mostFreq <- subset(wordFreq, wordFreq >= 50)
```

```
head(mostFreq, 10)
```

```
## you dont the people
```

```
## 137 81 62 50
```

```
length(wordFreq)
```

```
## [1] 5562
```

```
length(wordFreq[wordFreq<10])
```

```
## [1] 5457
```

```
length(wordFreq[wordFreq<5])
```

```
## [1] 5265
```

```
length(wordFreq[wordFreq==1])
```

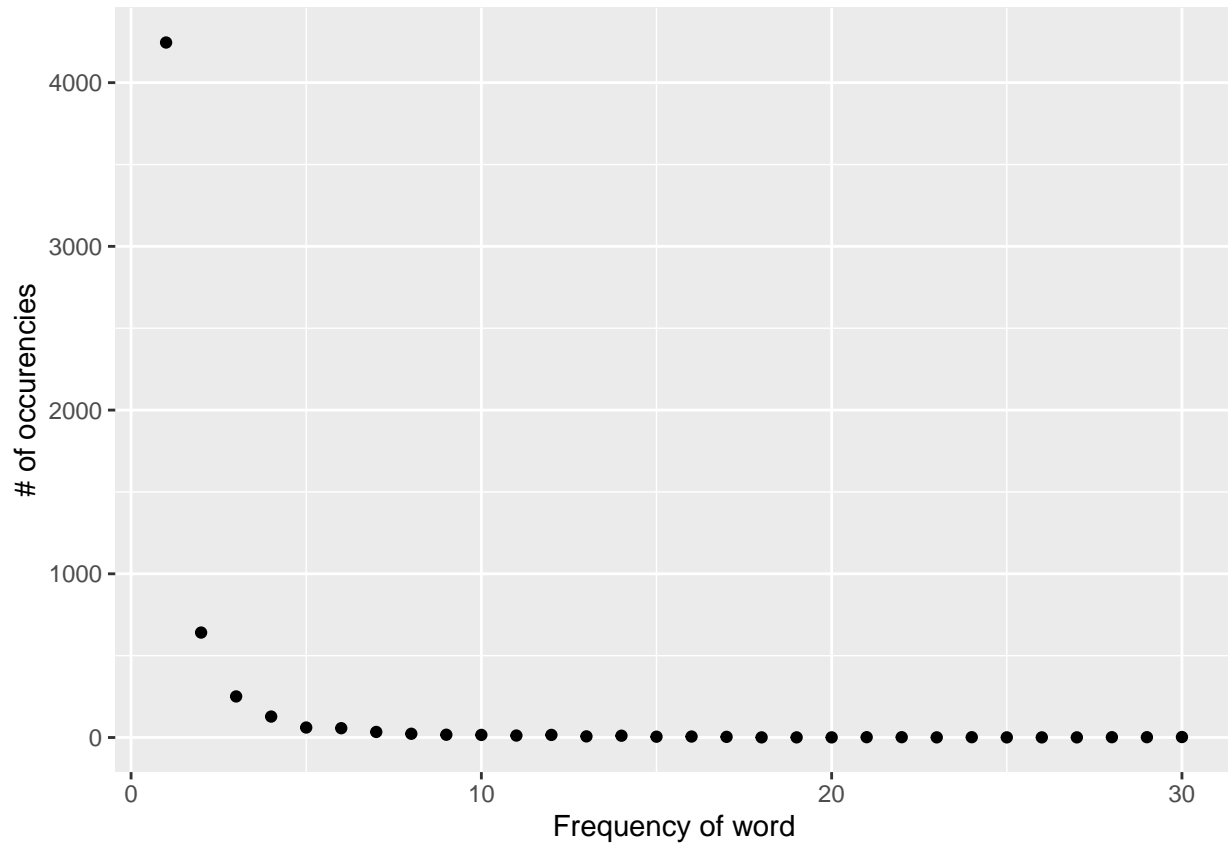
```
## [1] 4245
```

```
freq <- sort(unique(wordFreq), decreasing=FALSE)
```

```
occ <- vector()
```

```
for (i in 1:length(freq)) {
  occ[i] <- length(wordFreq[wordFreq == freq[i]])
}
```

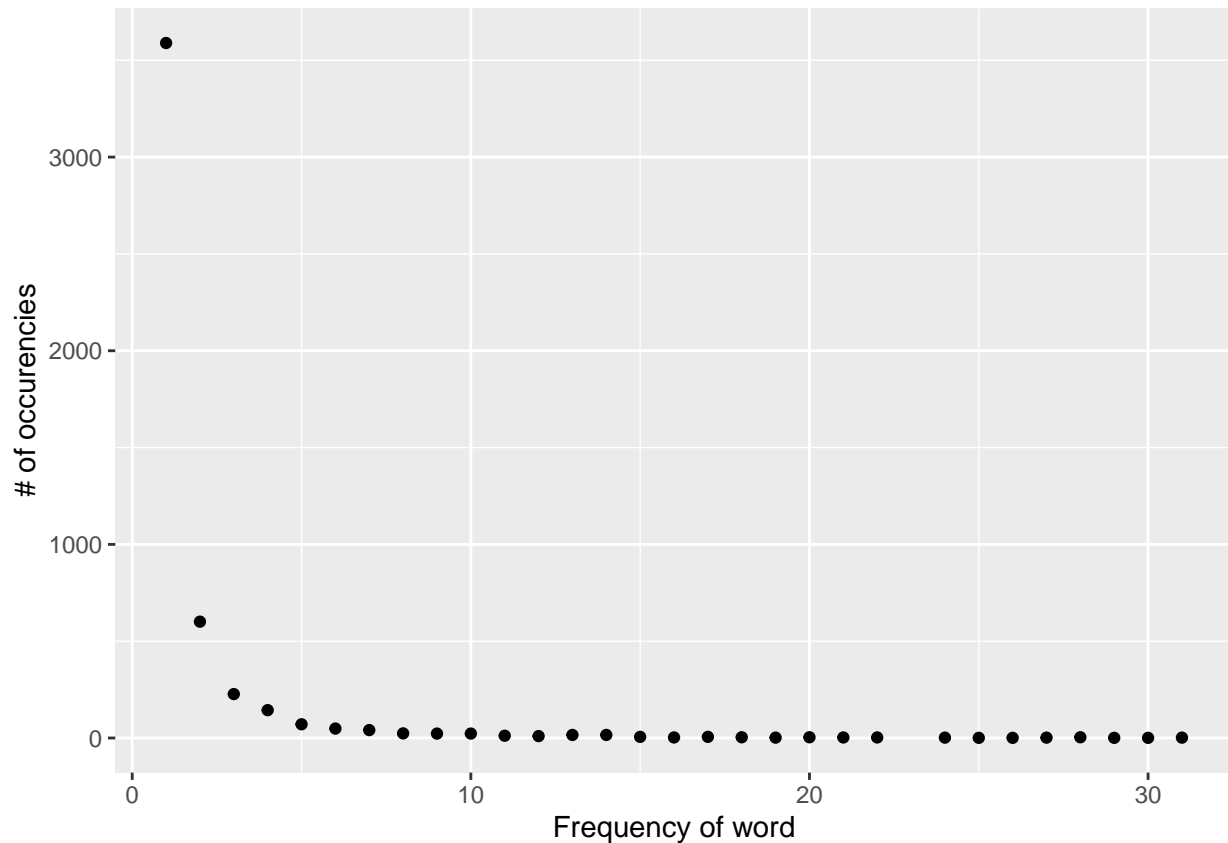
```
qplot(freq[1:30], occ[1:30], xlab = "Frequency of word", ylab = "# of occurencies")
```



II. Plot the frequency of words (with stemmization)

```
stemmed <- stemDocument(train$text_a, language = "english")
corpus2 <- Corpus(VectorSource(stemmed)) # turn into corpus
```

```
## you dont fuck the
## 137 81 79 62
## [1] 4904
## [1] 4769
## [1] 4561
## [1] 3589
```



II. Perform a clustering on the vectorized document space

We will use Weighted TF-IDF as a way to represent the document space:

```
tdm <- tm::DocumentTermMatrix(corpus2)
tdm.tfidf <- tm::weightTfIdf(tdm)
tdm.tfidf <- tm::removeSparseTerms(tdm.tfidf, 0.999) # sparsity being not well handled overall in R
tfidf.matrix <- as.matrix(tdm.tfidf)
```

Afterwards, we perform k-means algorithm to cluster in {2,4,8,16} classes.

```
cluster2 <- kmeans(tfidf.matrix, centers=2)
cluster4 <- kmeans(tfidf.matrix, centers=4)
cluster8 <- kmeans(tfidf.matrix, centers=8)
cluster16 <- kmeans(tfidf.matrix, centers=16)

cluster2.master <- cluster2$cluster
cluster4.master <- cluster4$cluster
cluster8.master <- cluster8$cluster
cluster16.master <- cluster16$cluster
```

We perform Classical multidimensional scaling (SMC) to map the data (distance matrix) into 2D dimension and then visualize it.

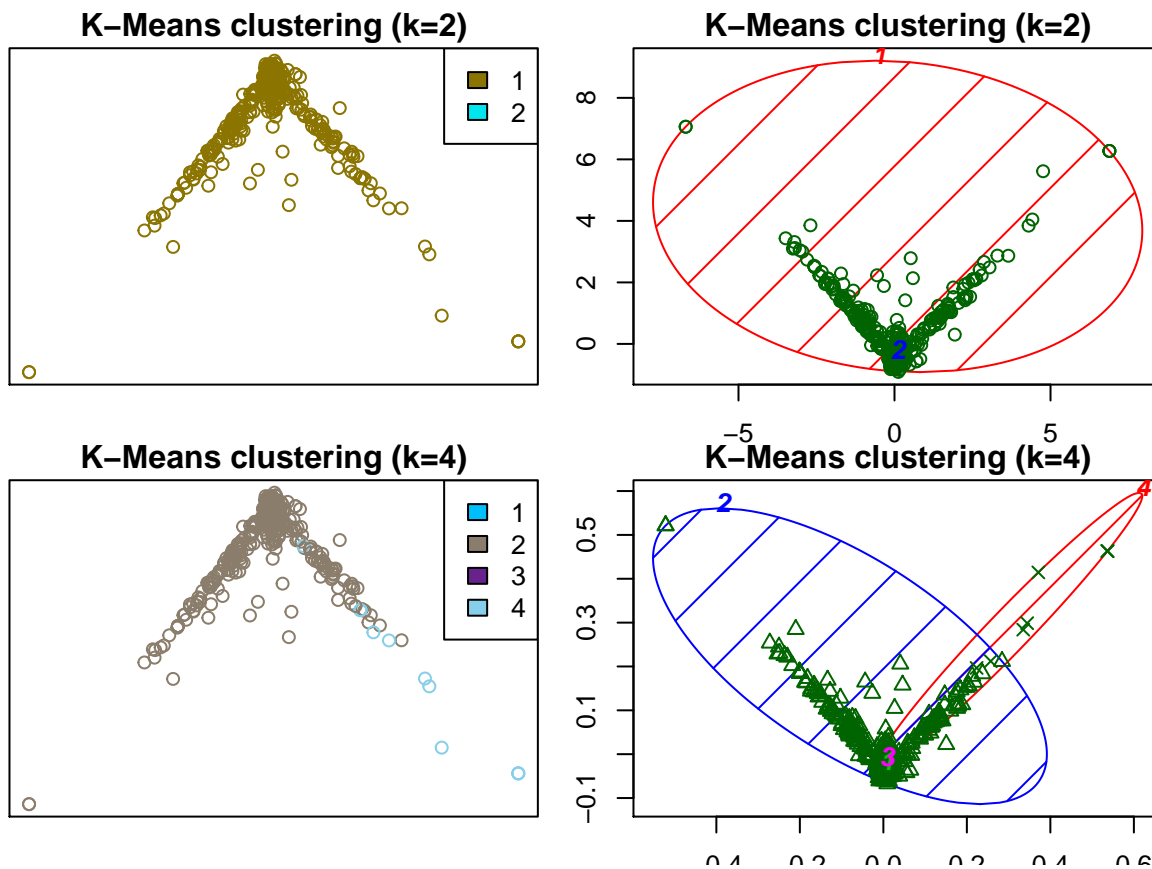
```
dist.matrix = proxy::dist(tfidf.matrix, method = "cosine")
points <- cmdscale(dist.matrix, k = 2)
previous.par <- par(mfrow=c(2,2), mar = rep(1.5, 4))
color <- grDevices::colors()[grep('gr(a|e)y', grDevices::colors(), invert = T)]
```

```

my_palette = sample(color, 2)
plot(points, main = 'K-Means clustering (k=2)', col = my_palette[as.factor(cluster2.master)],
     mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
     xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')
legend("topright", sprintf("%s",seq(1,2)), fill = my_palette[1:2])
clusplot(points, cluster2.master, main='K-Means clustering (k=2)', color=TRUE, shade=TRUE, labels=5, li

my_palette = sample(color, 4)
plot(points, main = 'K-Means clustering (k=4)', col = my_palette[as.factor(cluster4.master)],
     mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
     xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')
legend("topright", sprintf("%s",seq(1,4)), fill = my_palette[1:4])
clusplot(points, cluster4.master, main = 'K-Means clustering (k=4)', color=TRUE, shade=TRUE, labels=5, li

```



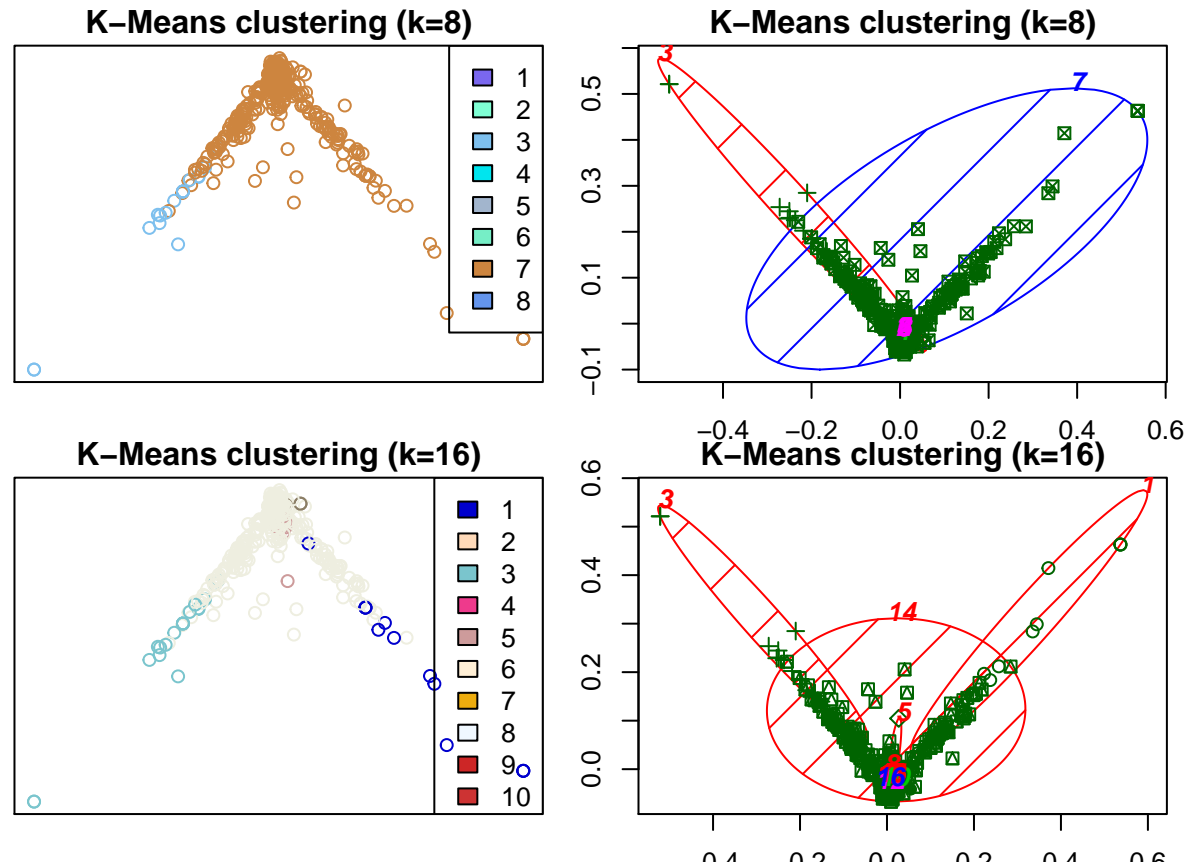
```

my_palette = sample(color, 8)
plot(points, main = 'K-Means clustering (k=8)', col = my_palette[as.factor(cluster8.master)],
     mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
     xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')
legend("topright", sprintf("%s",seq(1,8)), fill = my_palette[1:8])
clusplot(points, cluster8.master, main = 'K-Means clustering (k=8)', color=TRUE, shade=TRUE, labels=5, li

my_palette = sample(color, 16)
plot(points, main = 'K-Means clustering (k=16)', col = my_palette[as.factor(cluster16.master)],
     mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
     xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')
legend("topright", sprintf("%s",seq(1,16)), fill = my_palette[1:16])

```

```
clusplot(points, cluster16.master, main = 'K-Means clustering (k=16)', color=TRUE, shade=TRUE, labels=5
```

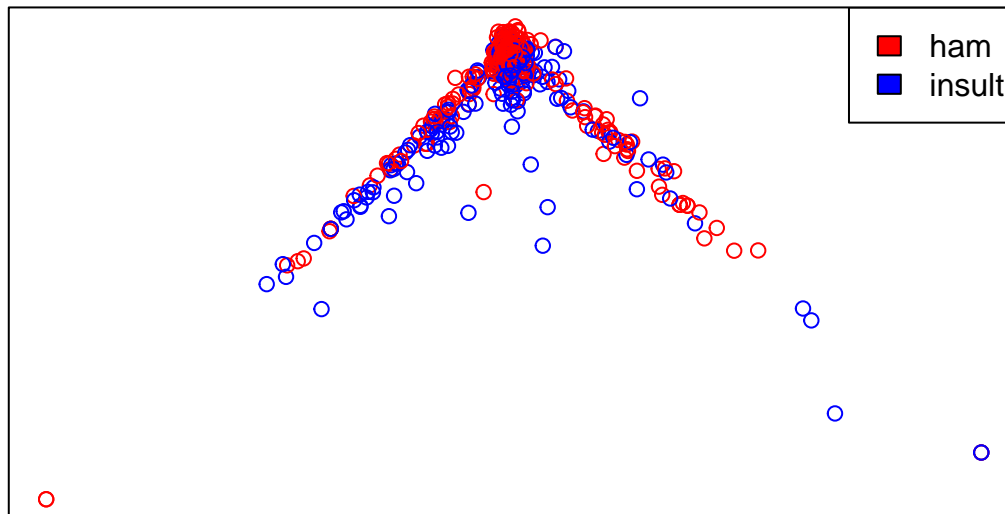


```
par(previous.par) # recovering the original plot space parameters
```

```
points <- cmdscale(dist.matrix, k = 2)
colors <- c('red', 'blue')
```

```
plot(points, main = 'Documents with class labels', col = colors[as.factor(train$label)],
     mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
     xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')
legend("topright", c('ham', 'insult'), fill = colors[1:2])
```


Documents with class labels



```
stemmed <- stemDocument(train$text_a, language = "english")
docs <- stemmed
n <- length(docs)
word_ann <- Maxent_Word-Token-Annotator()
sent_ann <- Maxent_Sent-Token-Annotator()
pos_ann = Maxent_POS-Tag-Annotator()
```

```
docsPOS <- list()
for (i in 1:n) {
  doc <- as.String(docs[[i]])
  wordAnnotation <- annotate(doc, list(sent_ann, word_ann))
  POSAnnotation <- annotate(doc, pos_ann, wordAnnotation)
  POSWords <- subset(POSAnnotation, type == "word")
  POSTags <- vector()
  for (j in 1:length(POSWords$features))
    POSTags <- c(POSTags, POSWords$features[[j]]$POS)
  docsPOS[[i]] <- list(POSTags)
}
```

```
library(purrr)
for(i in 1:length(docsPOS)){
  docsPOS[i] = flatten(docsPOS[i])
  docsPOS[[i]] = paste(docsPOS[[i]], collapse=" ")
}
```

```
head(docsPOS)
```

```
## [[1]]
## [1] "IN NN NN WP NN JJ NN VB RB JJ NN JJ NN RB VB VB FW NN VB PRP VBP JJ NN FW VB NN NNS VBP RP IN NN"
##
## [[2]]
## [1] "NN VB"
##
## [[3]]
## [1] "NN NN VBN NN VBG NN PRP MD VB IN"
```

```
##
## [[4]]
## [1] "NN VBP JJ JJ NNP NN"
##
## [[5]]
## [1] "PRP VBP DT JJ JJ NN VBD NN IN CD CD CD IN CD CD JJ CD JJ NN DT CD CD NN CD NN JJ NN JJ NN JJ NN"
##
## [[6]]
## [1] "JJ NN VBP CD"
```

```
head(stemmed)
```

```
## [1] "5dcac30f death blow xc2xa0That stuff total danger build toler quick stop abrupt xc2xa0It insidi
## [2] "moron happen"
## [3] "idiot blabbermouth gonna stop e8a1d6c8 day You WILL 91b0cb01 save"
## [4] "tower connect bottom pentagon Sinc flat"
## [5] "I love a9350e16 cee1217a upsid hell ea737b57 calib 9c894fe8 21 1a620f48 17 Intercept 91465074 4
## [6] "d84ee5df shit die 2703f309"
```

```
length(docsPOS)
```

```
## [1] 801
```

```
docsPOS <- matrix(docsPOS, 801, 1)
docsPOS <- unlist(docsPOS)
trainPOS <- train
textPOS <- trainPOS$text_a
textPOS <- paste(textPOS, docsPOS)
trainPOS$text_a <- textPOS
stemmedPOS <- stemDocument(trainPOS$text_a, language = "english")
```

```
head(stemmedPOS)
```

```
## [1] "5dcac30f death blow xc2xa0That stuff total danger build toler quick stop abrupt xc2xa0It insidi
## [2] "moron happen NN VB"
## [3] "idiot blabbermouth gonna stop e8a1d6c8 day You WILL 91b0cb01 save NN NN VBN NN VBG NN PRP MD VB
## [4] "tower connect bottom pentagon Sinc flat NN VBP JJ JJ NNP NN"
## [5] "I love a9350e16 cee1217a upsid hell ea737b57 calib 9c894fe8 21 1a620f48 17 Intercept 91465074 4
## [6] "d84ee5df shit die 2703f309 JJ NN VBP CD"
```

3. Modeling

```
test$text_a = as.character(test$text_a)
test$text_a = tm::removePunctuation(test$text_a)
test$text_a = tm::removeWords(x = test$text_a, stopwords(kind = "SMART"))
test$text_a = tm::stripWhitespace(test$text_a)
```

```
test <- test[which(lapply(test$text_a, wordcount) > 0),]
n <- length(test$text_a)
for (i in 1:n) {
  while(1) {
    doc <- as.String(test$text_a[[i]])
    wordAnnotation <- annotate(doc, list(sent_ann, word_ann))
    POSAnnotation <- annotate(doc, pos_ann, wordAnnotation)
    POSWords <- subset(POSAnnotation, type == "word")
```

```

POSTags <- vector()
for (j in 1:length(POSWords$features))
  POSTags <- c(POSTags, POSWords$features[[j]]$POS)
tokenPOS <- cbind(doc[POSWords], POSTags)
ppn_idx <- which(tokenPOS[,2] == "NNP", 1)
if (length(ppn_idx) == 0) {
  break;
}
words <- subset(wordAnnotation, type == "word")
hashed <- digest(tokenPOS[ppn_idx, 1], "xxhash32")
ppn <- words[ppn_idx]
test$text_a[[i]] <- gsub(doc[ppn$start,ppn$end], hashed, doc)
}
}
test$text_a <- iconv(test$text_a, to='UTF-8', sub='byte')
test$label=ifelse(test$label==0,"No","Yes")
test$label <- as.factor(test$label)
stemmedtest <- stemDocument(test$text_a, language = "english")
corpustest <- Corpus(VectorSource(stemmedtest)) # turn into corpus

tdmtest <- tm::DocumentTermMatrix(corpustest)
tdmtest.tfidf <- tm::weightTfIdf(tdmtest)

## Warning in tm::weightTfIdf(tdmtest): empty document(s): 457
tdmtest.tfidf <- tm::removeSparseTerms(tdmtest.tfidf, 0.999)
tfidftest.matrix <- as.matrix(tdmtest.tfidf)

```

#The train dataset is imbalanced with, where there are more than 2 times as much documents labeled as non-insults than insults. We performed oversampling to get rid of this imbalance:

```

train$text_a = stemmed
trainPOS$text_a = stemmedPOS

table(train$label)

##
##    0    1
## 572 229

trainSVM <- ovun.sample(label ~ ., data = train, method = "over")$data
trainRf <- train[sample(nrow(train), 100), ]
trainRf <- ovun.sample(label ~ ., data = trainRf, method = "over", 150)$data
trainPOS <- ovun.sample(label ~ ., data = trainPOS, method = "over")$data

print('SVM Model:')

## [1] "SVM Model:"
table(trainSVM$label)

##
##    0    1
## 572 553

print('RF Model:')

## [1] "RF Model:"

```

```

table(trainRf$label)

##
## 0 1
## 74 76

print('SVM with Tags Model:')

## [1] "SVM with Tags Model:"

table(trainPOS$label)

##
## 0 1
## 572 584

clean_test <- function(data) {
  test.matrix <- as.data.frame(tfidf.test.matrix)
  cols <- colnames(data)
  Missing <- setdiff(cols, names(test.matrix))
  test.matrix[Missing] <- 0
  test.matrix <- test.matrix[cols]
  return(test.matrix)
}

create_model_data <- function(matrix, avect) {
  result <- cbind(matrix, avect)
  result <- as.data.frame(result)
  result$lbl = ifelse(result$lbl == 0, "No", "Yes")

  # Adding a vector of labels to the tfidf matrix changing the 0s to No and 1s to Yes
  result$lbl <- as.factor(result$lbl)
  return(result)
}

corpus <- Corpus(VectorSource(trainRf$text_a)) # turn into corpus
tdm <- tm::DocumentTermMatrix(corpus)
tdm.tfidf <- tm::weightTfIdf(tdm)
tdm.tfidf <- tm::removeSparseTerms(tdm.tfidf, 0.999) # sparsity being not well handled overall in R
tfidf.matrixRf <- as.matrix(tdm.tfidf)

corpus <- Corpus(VectorSource(trainSVM$text_a)) # turn into corpus
tdm <- tm::DocumentTermMatrix(corpus)
tdm.tfidf <- tm::weightTfIdf(tdm)
tdm.tfidf <- tm::removeSparseTerms(tdm.tfidf, 0.999) # sparsity being not well handled overall in R
tfidf.matrixSVM <- as.matrix(tdm.tfidf)

corpus <- Corpus(VectorSource(trainPOS$text_a)) # turn into corpus
tdmPOS <- tm::DocumentTermMatrix(corpus)
tdmPOS.tfidf <- tm::weightTfIdf(tdmPOS)
tdmPOS.tfidf <- tm::removeSparseTerms(tdmPOS.tfidf, 0.999) # sparsity being not well handled overall in R
tfidf.matrixPOS <- as.matrix(tdmPOS.tfidf)

trainRf$lbl <- trainRf$label
trainSVM$lbl <- trainSVM$label
trainPOS$lbl <- trainPOS$label

```

```

avectorRf <- as.vector(trainRf['lbl'])
avectorSVM <- as.vector(trainSVM['lbl'])
avectorPOS <- as.vector(trainPOS['lbl'])

finalRf <- create_model_data(tfidf.matrixRf, avectorRf)
finalSVM <- create_model_data(tfidf.matrixSVM, avectorSVM)
finalPOS <- create_model_data(tfidf.matrixPOS, avectorPOS)

dat <- twoClassSim(200) #A custom f1 funtion for the metric
f1 <- function(data, lev = NULL, model = NULL) {
  f1_val <- F1_Score(y_pred = data$pred, y_true = data$obs, positive = lev[1])
  c(F1 = f1_val)
}
train.control <- trainControl(method = "repeatedcv",
                             number = 5,
                             repeats = 3,
                             classProbs = TRUE,
                             summaryFunction = f1,
                             search = "grid")

fitRf <- caret::train(lbl ~ ., data = finalRf, method = 'rf', tuneLength = 5, metric = "F1",
                     trControl = train.control)

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.

fitSvm <- caret::train(lbl ~ ., data = finalSVM, method = 'svmLinear', scale=F, tuneLength = 5, metric = "F1")
fitSvmPOS <- caret::train(lbl ~ ., data = finalPOS, method = 'svmLinear', scale=F, tuneLength = 5, metric = "F1")

print(fitRf$results)

##      mtry      F1      F1SD
## 1      2 0.2996417 0.30935246
## 2      8 0.9187013 0.06604385
## 3     39 0.9529806 0.04249923
## 4    175 0.9550076 0.03544502
## 5     779 0.9287678 0.03366483

print(fitSvm$results)

##      C      F1      F1SD
## 1 1 0.9018473 0.02432647

print(fitSvmPOS$results)

##      C      F1      F1SD
## 1 1 0.9097263 0.01309939

#We check what columns are missing in the test dataframe and we add them setting their values to 0
test.matrixRf <- clean_test(finalRf)
test.matrixSVM <- clean_test(finalSVM)
test.matrixPOS <- clean_test(finalPOS)

#We predict the two models on the test matrix
predRf = predict(fitRf, newdata=test.matrixRf)
predSvm = predict(fitSvm, newdata=test.matrixSVM)

```

```

predSvmPOS = predict(fitSvmPOS, newdata=test.matrixPOS)

#And then we factorize the labels for visualization
con.matrix.rf<-confusionMatrix(predRf, test$label)
con.matrix.svm<-confusionMatrix(predSvm, test$label)
con.matrix.svm.pos<-confusionMatrix(predSvmPOS, test$label)
print(con.matrix.rf)

```

```

## Confusion Matrix and Statistics
##
##              Reference
## Prediction  No  Yes
##      No  545 186
##      Yes   13  23
##
##              Accuracy : 0.7405
##              95% CI : (0.708, 0.7712)
##      No Information Rate : 0.7275
##      P-Value [Acc > NIR] : 0.2213
##
##              Kappa : 0.117
##
##  Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.9767
##              Specificity : 0.1100
##      Pos Pred Value : 0.7456
##      Neg Pred Value : 0.6389
##      Prevalence : 0.7275
##      Detection Rate : 0.7106
##      Detection Prevalence : 0.9531
##      Balanced Accuracy : 0.5434
##
##      'Positive' Class : No
##

```

```

print(con.matrix.svm)

```

```

## Confusion Matrix and Statistics
##
##              Reference
## Prediction  No  Yes
##      No  505 131
##      Yes   53  78
##
##              Accuracy : 0.7601
##              95% CI : (0.7283, 0.7899)
##      No Information Rate : 0.7275
##      P-Value [Acc > NIR] : 0.02241
##
##              Kappa : 0.315
##
##  Mcnemar's Test P-Value : 1.375e-08
##

```

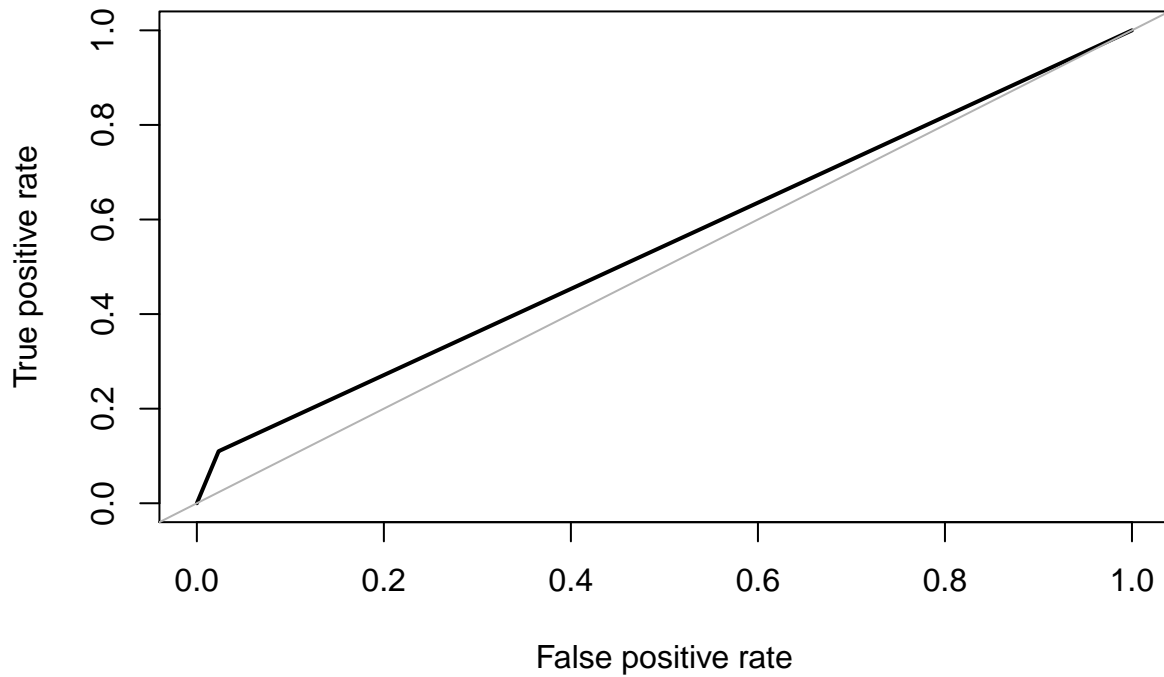
```
##           Sensitivity : 0.9050
##           Specificity : 0.3732
##           Pos Pred Value : 0.7940
##           Neg Pred Value : 0.5954
##           Prevalence : 0.7275
##           Detection Rate : 0.6584
##           Detection Prevalence : 0.8292
##           Balanced Accuracy : 0.6391
##
##           'Positive' Class : No
##
```

```
print(con.matrix.svm.pos)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##           No  499 122
##           Yes   59  87
##
##           Accuracy : 0.764
##           95% CI : (0.7323, 0.7937)
##           No Information Rate : 0.7275
##           P-Value [Acc > NIR] : 0.01201
##
##           Kappa : 0.3429
##
## Mcnemar's Test P-Value : 4.057e-06
##
##           Sensitivity : 0.8943
##           Specificity : 0.4163
##           Pos Pred Value : 0.8035
##           Neg Pred Value : 0.5959
##           Prevalence : 0.7275
##           Detection Rate : 0.6506
##           Detection Prevalence : 0.8096
##           Balanced Accuracy : 0.6553
##
##           'Positive' Class : No
##
```

```
roc.curve(test$label, predRf, main="RF - ROC curve")
```

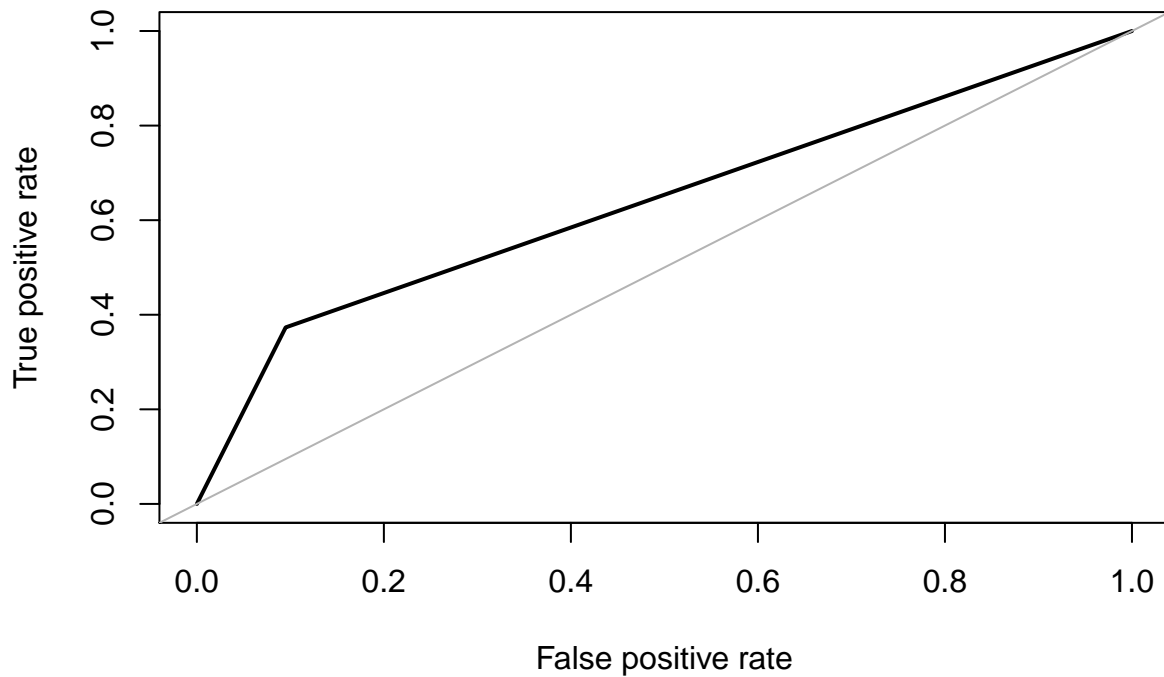
RF – ROC curve



Area under the curve (AUC): 0.543

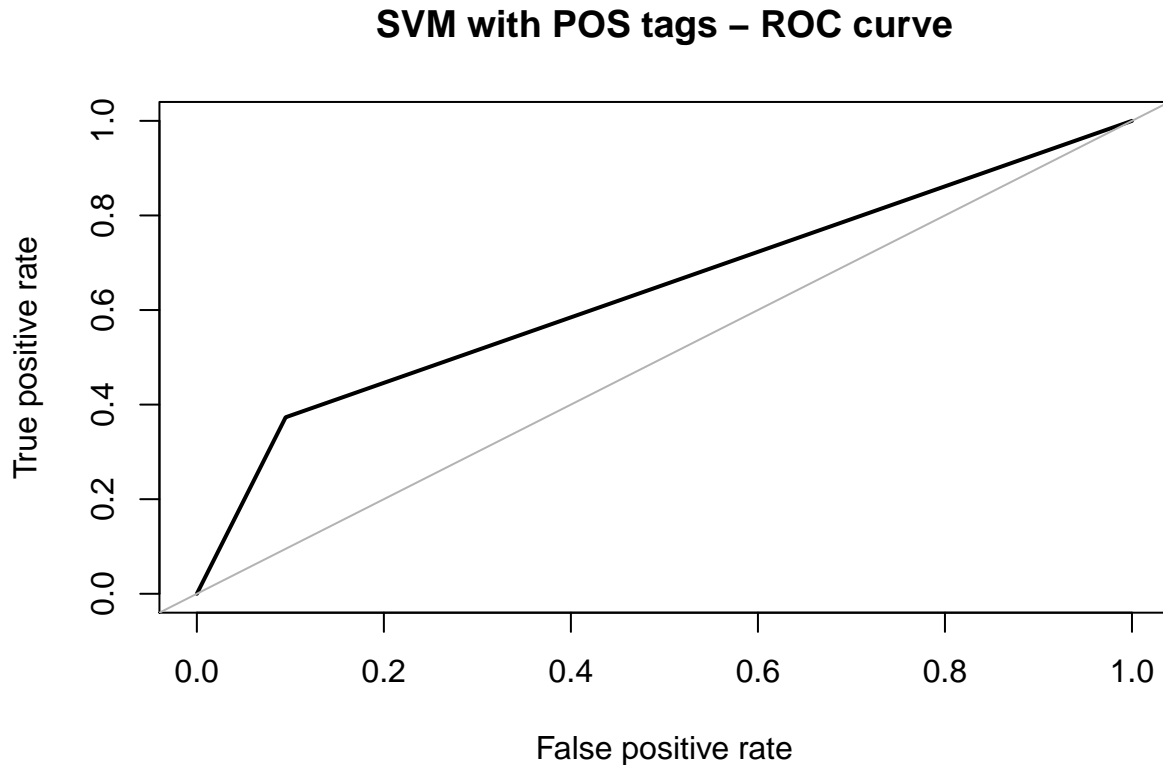
```
roc.curve(test$label, predSvm, main="SVM - ROC curve")
```

SVM – ROC curve



Area under the curve (AUC): 0.639


```
roc.curve(test$label, predSvm, main="SVM with POS tags - ROC curve")
```



```
## Area under the curve (AUC): 0.639
```

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

```
##
##  0  1
## 572 229
```

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

```
##
## No Yes
## 558 209
```

```
majority_classifier <- length(which(test$label == 'No')) / length(test$label)
```

```
majority_classifier
```

```
## [1] 0.7275098
```

4. Understanding

```
corpus <- Corpus(VectorSource(stemmed)) # turn into corpus
tdm <- tm::DocumentTermMatrix(corpus)
tdm.tfidf <- tm::weightTfIdf(tdm)
tdm.tfidf <- tm::removeSparseTerms(tdm.tfidf, 0.999) # sparsity being not well handled overall in R
tfidf.matrix <- as.matrix(tdm.tfidf)
```

We will use only terms which occur in more than one document:

```

# load the library
library(mlbench)
library(caret)

# get rid of words which are only in 1 document
dim(tfidf.matrix)

## [1] 801 4904

tfidf.matrix <- tfidf.matrix[,-which(rowSums(as.matrix(tdm2)) == 1)]
dim(tfidf.matrix)

## [1] 801 1315

```

Perform feature ranking

We will use the TF-IDF matrix to calculate information gain by filter method:

```

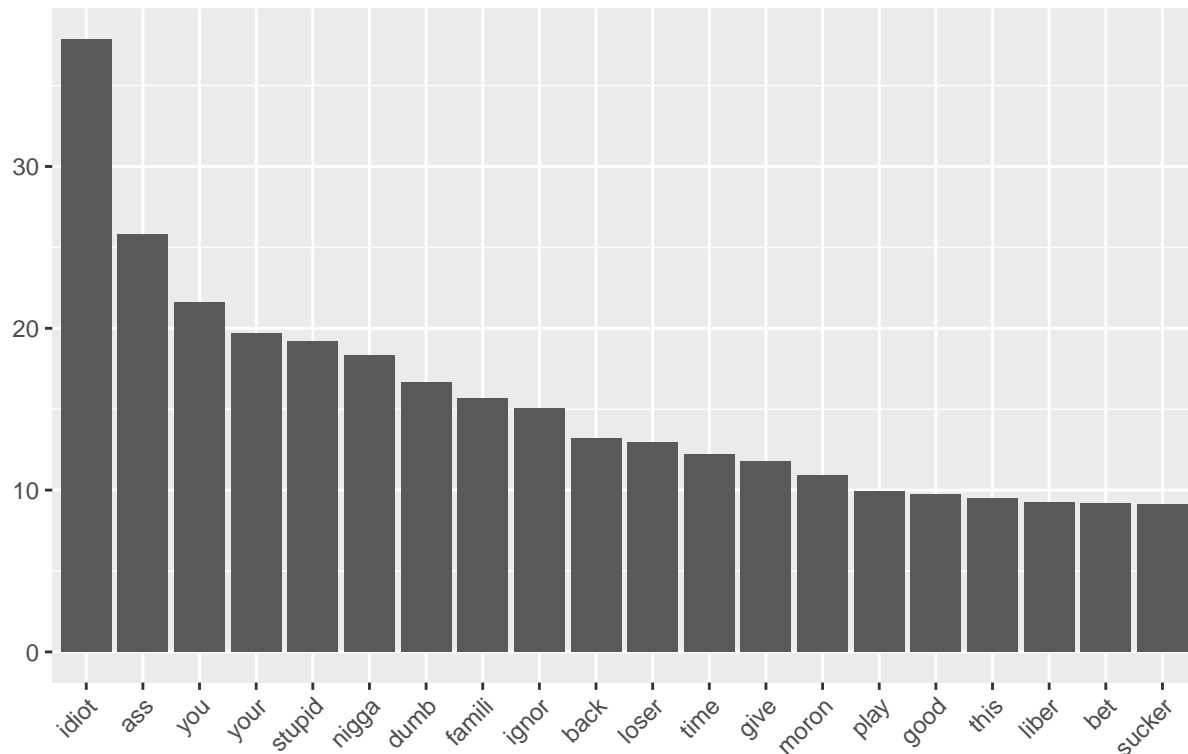
#install.packages('ggpubr')
library(mlr)
library(ggpubr)
finalRf <- create_model_data(tfidf.matrixRf, avectorRf)
final <- create_model_data(tfidf.matrixSVM, avectorSVM)
colnames(final) <- make.names(colnames(final),unique = T)
label.task <- makeClassifTask(data=final, target='lbl')

melt <- reshape2::melt
fv = generateFilterValuesData(label.task, method ="anova.test")

fv$data <- fv$data[order(-fv$data$value),]
fv.plot <- fv
fv.plot$data <- head(fv.plot$data, 20)
plotFilterValues(fv.plot)

```

final (1744 features), filter = anova.test



Re-evaluate the models performance for top n features

```
features <- fv$data$name
n_features <- c(head(features, 25), 'lbl')
train.data <- finalSVM[,n_features]

fitSvm <- caret::train(lbl ~ ., data = train.data, method = 'svmLinear',scale=F, tuneLength = 5, metric = 'F1',
  trControl = train.control)
fitRf <- caret::train(lbl ~ ., data = train.data, method = 'rf',tuneLength = 5, metric = "F1",
  trControl = train.control)
fitSvm$results

##      C      F1      F1SD
## 1 1 0.7802341 0.01646707

fitRf$results

##      mtry      F1      F1SD
## 1      2 0.7844055 0.02081507
## 2      7 0.7927247 0.02075628
## 3     13 0.7984545 0.01900419
## 4     19 0.8025227 0.01924396
## 5     25 0.8035573 0.02056539

test.matrix <- clean_test(train.data)

#We predict the two models on the test matrix
predRf = predict(fitRf, newdata=test.matrix)
predSvm = predict(fitSvm, newdata=test.matrix)
```

```

#And then we factorize the labels for visualization
con.matrix.rf2 <- confusionMatrix(predRf, test$label)
con.matrix.svm2 <- confusionMatrix(predSvm, test$label)
print(con.matrix.rf2$overall)

##      Accuracy      Kappa  AccuracyLower  AccuracyUpper  AccuracyNull
## 0.75619296 0.34487862 0.72420310 0.78619869 0.72750978
## AccuracyPValue  McNemarPValue
## 0.03945798 0.00344357
print(con.matrix.svm2$overall)

##      Accuracy      Kappa  AccuracyLower  AccuracyUpper  AccuracyNull
## 0.7848761408 0.4345704827 0.7540762346 0.8134671068 0.7275097784
## AccuracyPValue  McNemarPValue
## 0.0001545328 0.0429601460
print(con.matrix.rf$overall)

##      Accuracy      Kappa  AccuracyLower  AccuracyUpper  AccuracyNull
## 7.405476e-01 1.170495e-01 7.079872e-01 7.712460e-01 7.275098e-01
## AccuracyPValue  McNemarPValue
## 2.212727e-01 3.397311e-34
print(con.matrix.svm$overall)

##      Accuracy      Kappa  AccuracyLower  AccuracyUpper  AccuracyNull
## 7.601043e-01 3.149858e-01 7.282654e-01 7.899285e-01 7.275098e-01
## AccuracyPValue  McNemarPValue
## 2.240649e-02 1.374663e-08

```

Visualize model performance w.r.t. n by using the selected measure of performance.

```

library(ggplot2)
features <- fv$data$name
n <- c(seq(10, 50, 10), 100, 200, 500)
accs <- c()
for (i in 1:length(n)) {
  n_features <- c(head(features, n[i]), 'lbl')
  train.data <- final[,n_features]

  fitSvm <- caret::train(lbl ~ ., data = train.data, method = 'svmLinear', scale=F, tuneLength = 5, metric = 'acc',
    trControl = train.control)

  #We check what columns are missing in the test dataframe and we add them setting their values to 0
  test.matrix <- as.data.frame(tfidftest.matrix)
  cols <- colnames(final)
  Missing <- setdiff(cols, names(test.matrix))
  test.matrix[Missing] <- 0
  test.matrix <- test.matrix[cols]

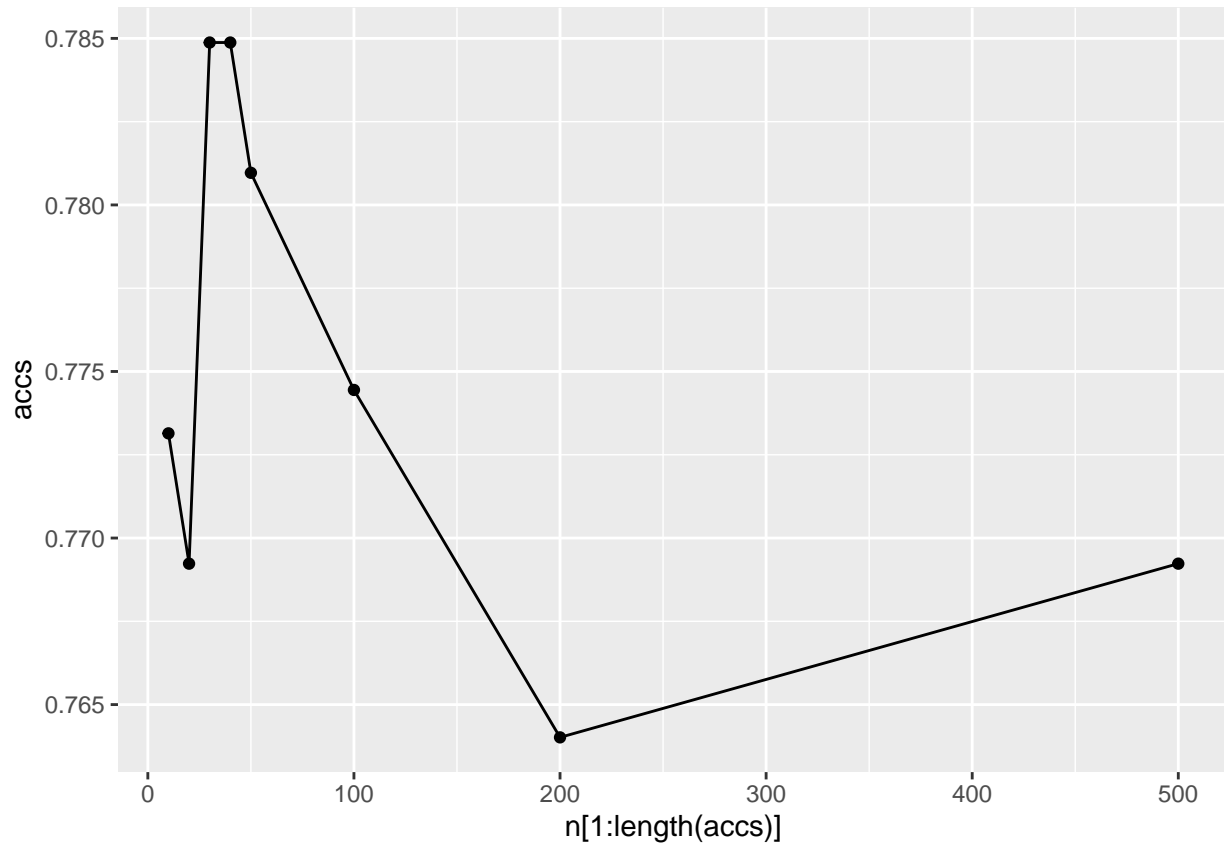
  #We predict the two models on the test matrix
  predSvm = predict(fitSvm, newdata=test.matrix)

```

```

#And then we factorize the labels for visualization
con.matrix.svm<-confusionMatrix(predSvm, test$label)
accs <- c(accs, con.matrix.svm$overall[[1]])
}
qplot(n[1:length(accs)], accs, geom=c("point", "line"))

```



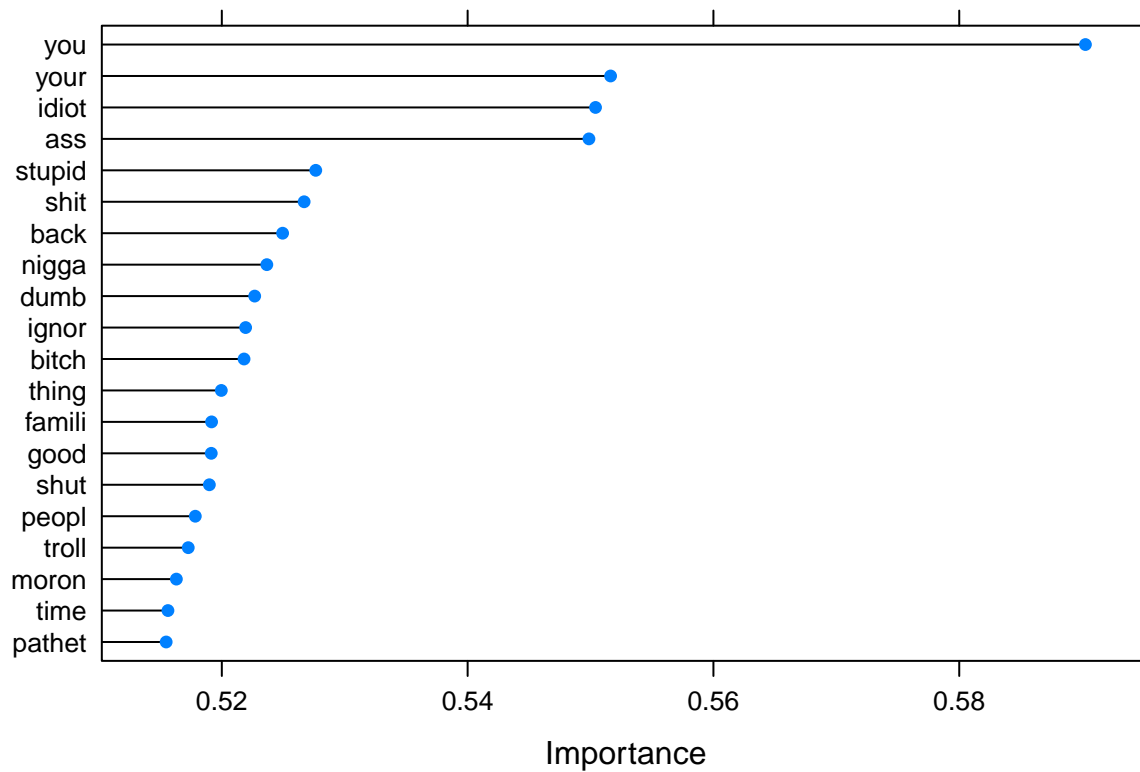
Extract feature importances from a wrapper method

```

# training the model
model <- caret::train(lbl ~ ., data = final, method = 'svmLinear', scale=F, tuneLength = 5, metric = "F
importance <- varImp(model, scale=FALSE)$importance

plot(varImp(model, scale=FALSE), top = 20)

```



Compare the two feature rankings

```
library(ggplot2)
get_jaccard <- function(A, B) {
  return(length(intersect(A,B)) / length(union(A,B)))
}
```

```
A <- fv$data$name
B <- rownames(importance[order(-importance$Yes),])
length(A) == length(B)
```

```
## [1] TRUE
```

```
Jaccard.score <- c()
for (i in 1:length(A)) {
  Jaccard.score[i] <- get_jaccard(A[1:i], B[1:i])
}
n <- seq(1, length(A), 1)
qplot(n, Jaccard.score)
```

