Assigment 2

Pavel Linder, Nikita Brancatisano

12/26/2019

0. Read input

```
train = 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's
## [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)

#train$text_a = tolower(train$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 Heisma
## [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 xc2
## [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"
```

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 xc2
## [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.

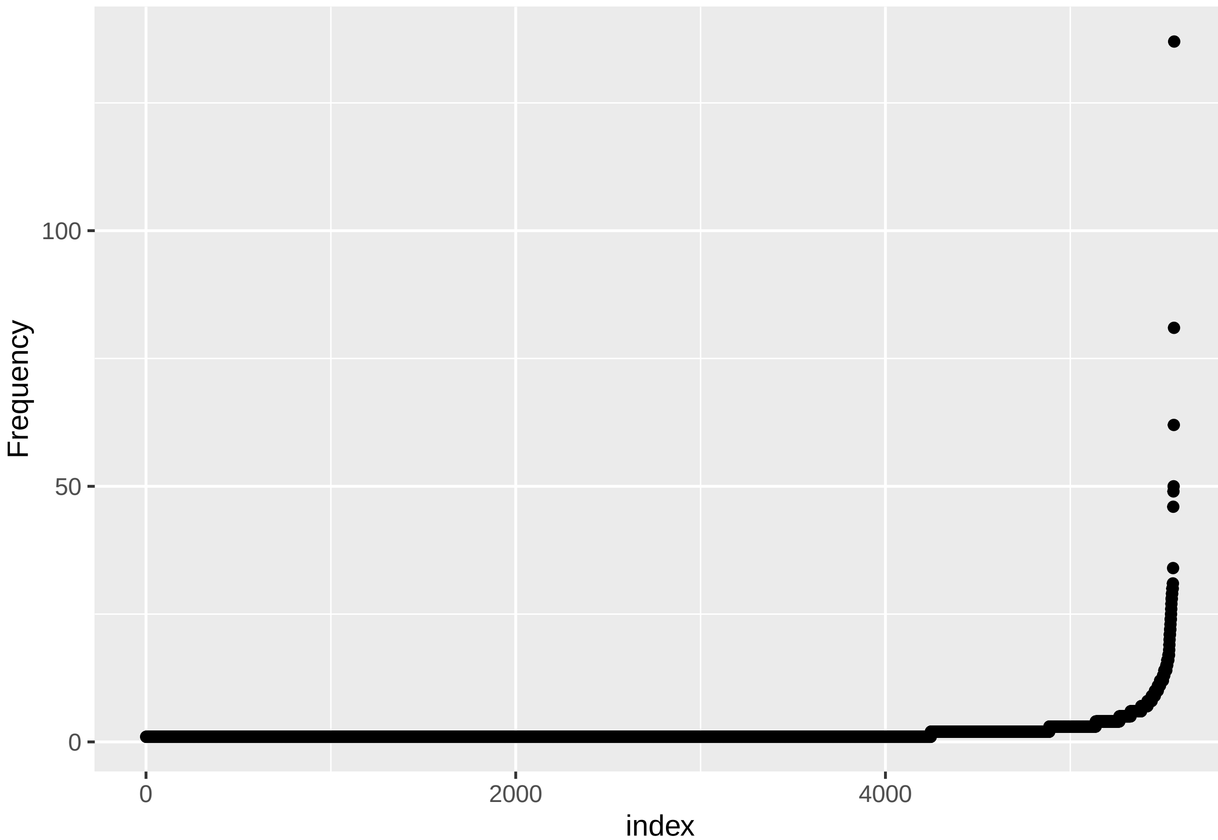
I. Plot the frequency of words (without stemmization)

```
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")
```




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

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

```r
mostFreq <- subset(wordFreq, wordFreq >= 50)
head(mostFreq, 10)
```

```
##    you   dont    the people
##    137     81     62     50
```

```r
length(wordFreq)
```

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

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

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

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

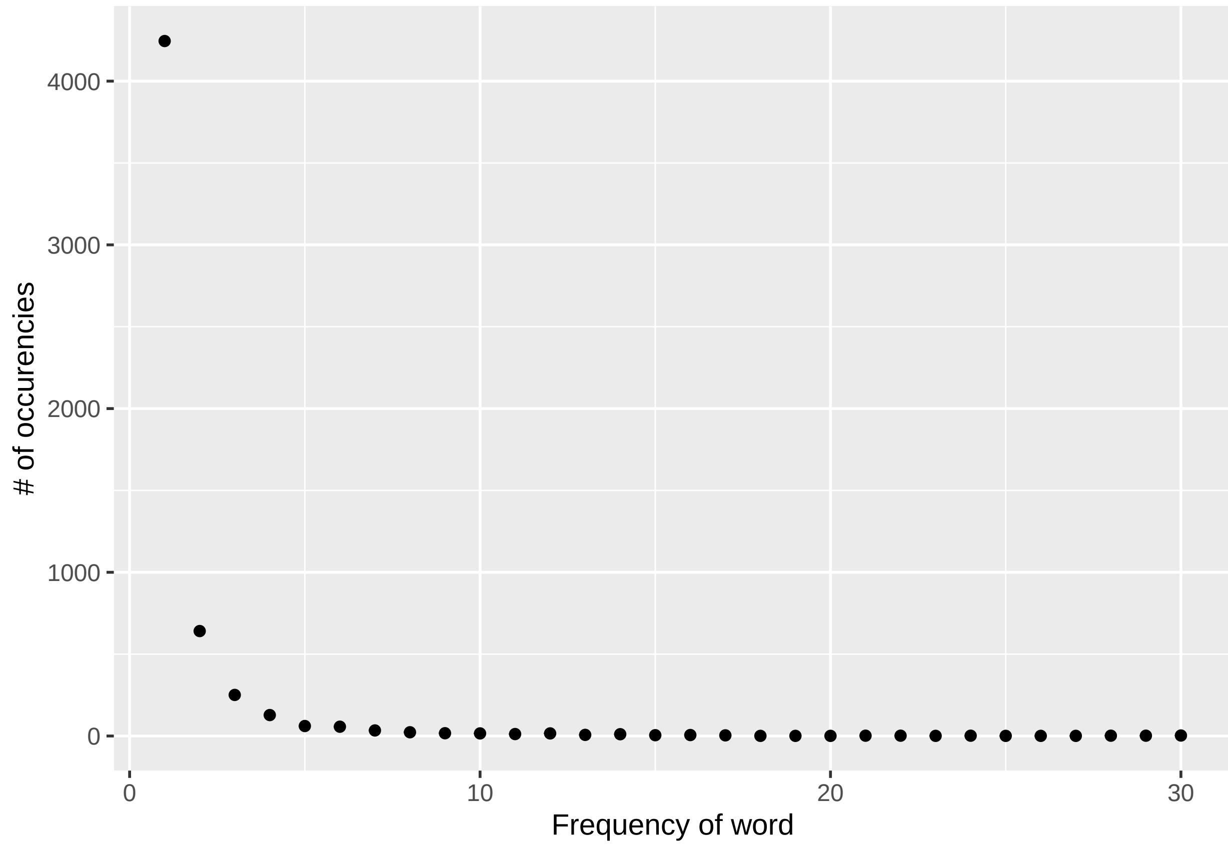
```
## [1] 5265
```

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

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

```r
freq <- sort(unique(wordFreq), decreasing=FALSE)
occ <- vector()
for (i in 1:length(freq)) {
  occ[i] <- length(wordFreq[wordFreq == freq[i]])
}
```

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



II. Plot the frequency of words (with stemmization)

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

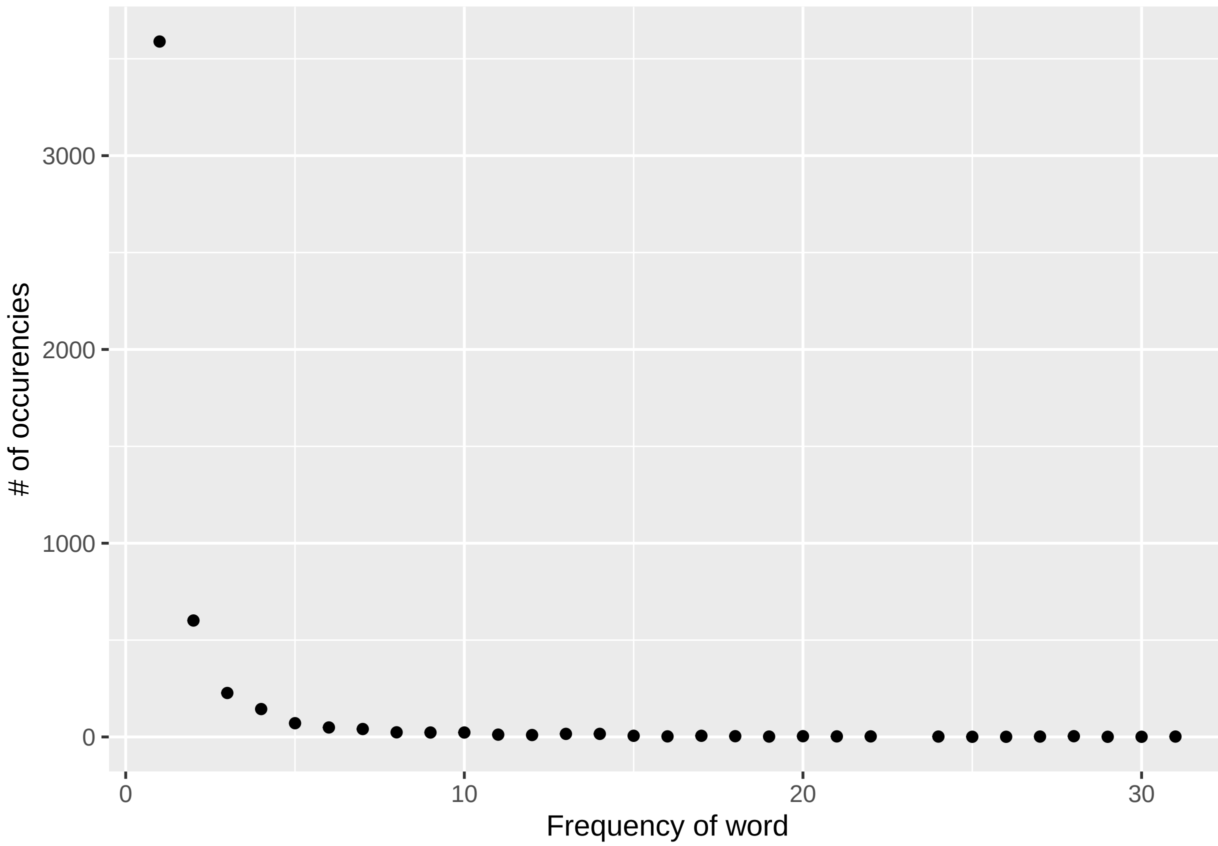
```
##  you dont fuck  the
##  137   81   79   62
```

```
## [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)
```

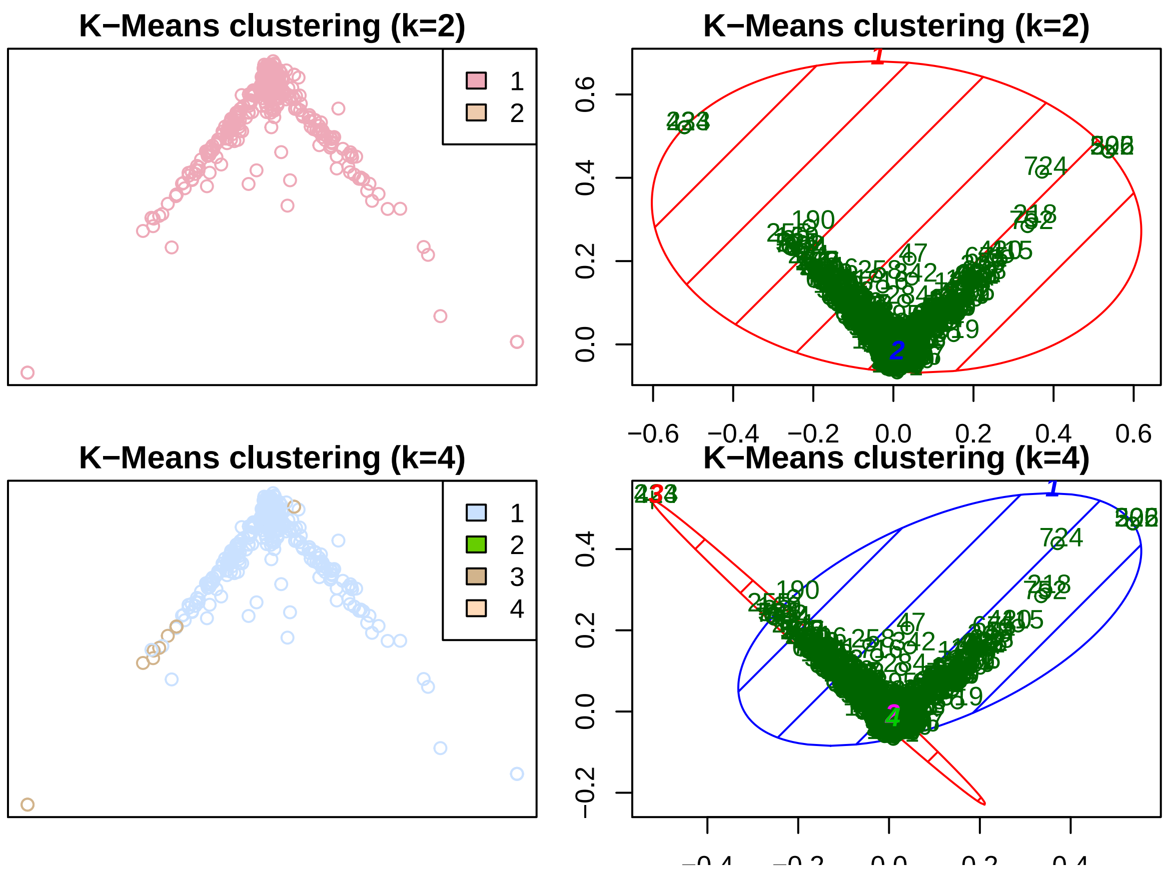
Afterwards, we perform k-means algorithm to cluster in {2,4,8,16} classes.

```
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.

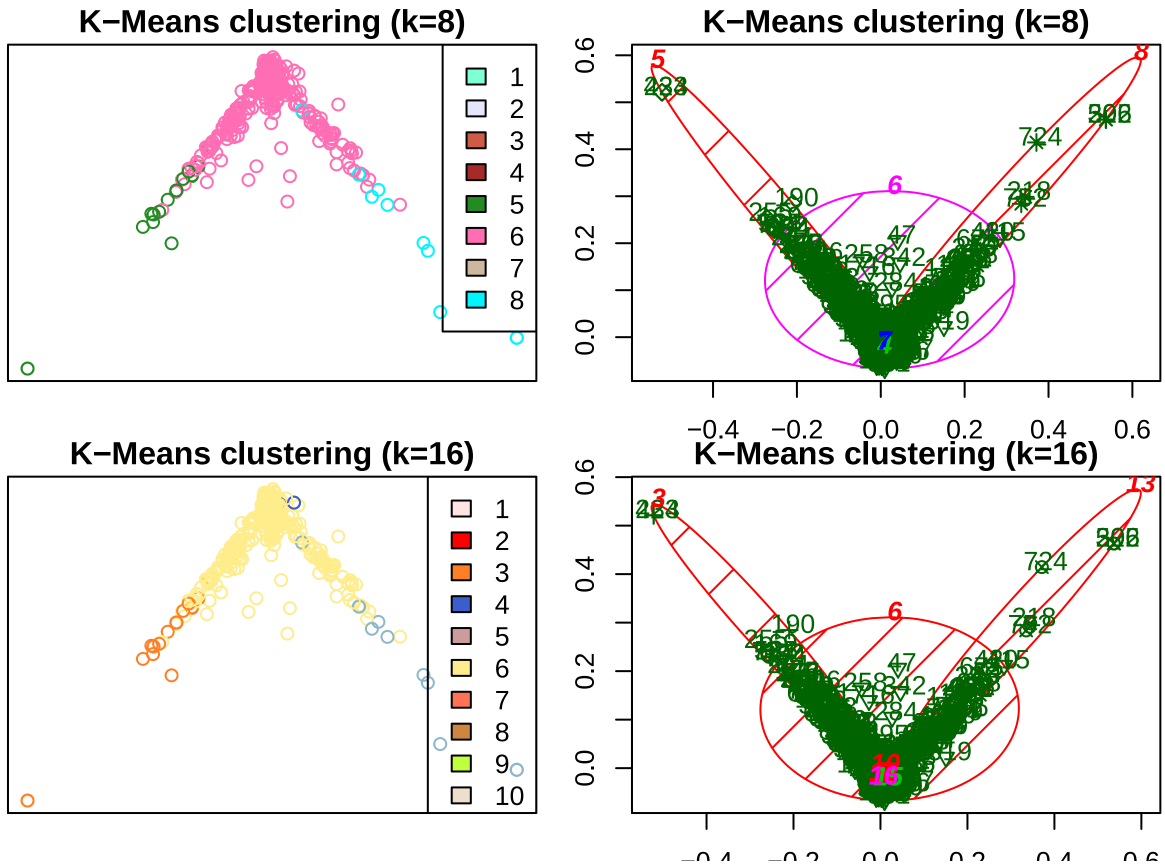
```
dist.matrix = proxy::dist(tfidf.matrix, method = "cosine")
points <- cmdscale(dist.matrix, k = 2)
previous.par <- par(mfrow=c(2,2), mar = rep(1.5, 4))
color <- grDevices::colors()[grep('gr(a|e)y', grDevices::colors(), invert = T)]
```

```
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=2, 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=2, 
```




```
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=2, 

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])
```

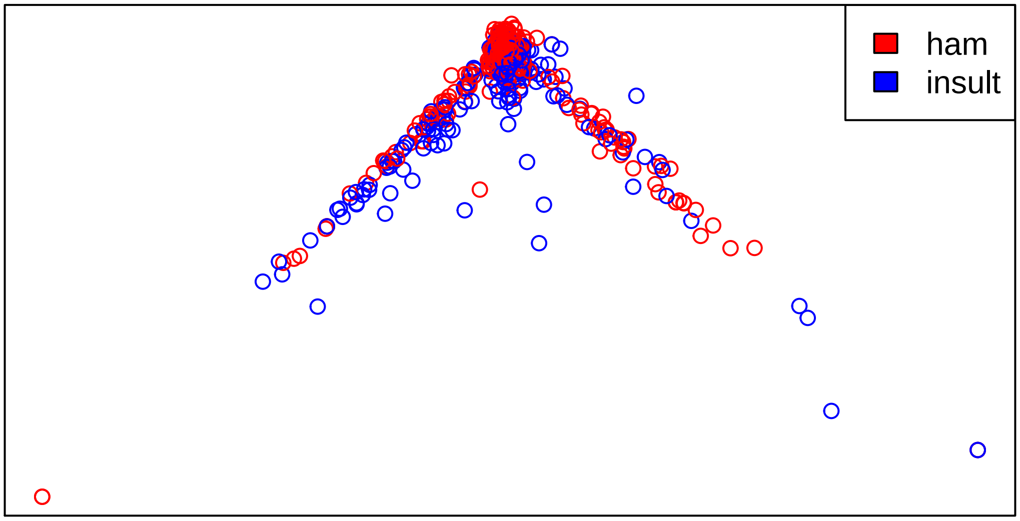
```
clusplot(points, cluster16.master, main = 'K-Means clustering (k=16)', color=TRUE, shade=TRUE, labels=2
```




```
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




```
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)
}

head(docsPOS)
```

```
## [[1]]
## [[1]][[1]]
##  [1] "IN"  "NN"  "NN"  "WP"  "NN"  "JJ"  "NN"  "VB"  "RB"  "JJ"  "NN"
## [12] "JJ"  "NN"  "RB"  "VB"  "VB"  "FW"  "NN"  "VB"  "PRP" "VBP" "JJ"
## [23] "NN"  "FW"  "VB"  "NN"  "NNS" "VBP" "RP"  "IN"  "NNP" "NNS" "NN"
## [34] "NN"  "JJ"  "JJ"  "NN"  "NN"  "IN"  "RB"  "VB"  "NN"  "JJ"  "NN"
## [45] "NN"  "JJ"  "NN"  "NN"  "NN"  "NN"  "NN"  "."
##
##
## [[2]]
## [[2]][[1]]
## [1] "NN" "VB"
##
##
## [[3]]
```

```
## [[3]][[1]]
##  [1] "NN"  "NN"  "VBN" "NN"  "VBG" "NN"  "PRP" "MD"  "VB"  "IN"
##
##
## [[4]]
## [[4]][[1]]
## [1] "NN"  "VBP" "JJ"  "JJ"  "NNP" "NN"
##
##
## [[5]]
## [[5]][[1]]
##  [1] "PRP" "VBP" "DT"  "JJ"  "JJ"  "NN"  "VBD" "NN"  "IN"  "CD"  "CD"
## [12] "CD"  "IN"  "CD"  "CD"  "JJ"  "CD"  "JJ"  "NN"  "DT"  "CD"  "CD"
## [23] "NN"  "CD"  "NN"  "JJ"  "NN"  "JJ"  "NN"  "JJ"  "NN"  "NN"  "NN"
## [34] "JJ"  "NN"  "JJ"  "NN"  "IN"  "CD"  "JJ"  "NN"  "IN"  "JJ"  "NN"
## [45] "NN"  "NN"  "NNS"
##
##
## [[6]]
## [[6]][[1]]
## [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 49
## [6] "d84ee5df shit die 2703f309"
```

```
length(docsPOS)
```

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

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 undersampling to get rid of this imbalance:

```
train$text_a = stemmed
table(train$label)
```

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

```
train2 <- ovun.sample(label ~ ., data = train, method = "over")$data
table(train2$label)
```

```
##
##   0   1
## 572 578
```

```
corpus <- Corpus(VectorSource(train2$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.matrix <- as.matrix(tdm.tfidf)
```

```
train2$lbl <- train2$label
avector <- as.vector(train2['lbl'])
final <- cbind(tfidf.matrix, avector)
final <- as.data.frame(final)
final$lbl=ifelse(final$lbl==0,"No","Yes")
final$lbl <- as.factor(final$lbl) #Adding a vector of labels to the tfidf matrix changing the Os to No
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 = final, method = 'rf',tuneLength = 5, metric = "F1",
 #              trControl = train.control
fitSvm <- caret::train(lbl ~ ., data = final, method = 'svmLinear', scale=F, tuneLength = 5, metric = "I

print(fitSvm$results$F1)
```

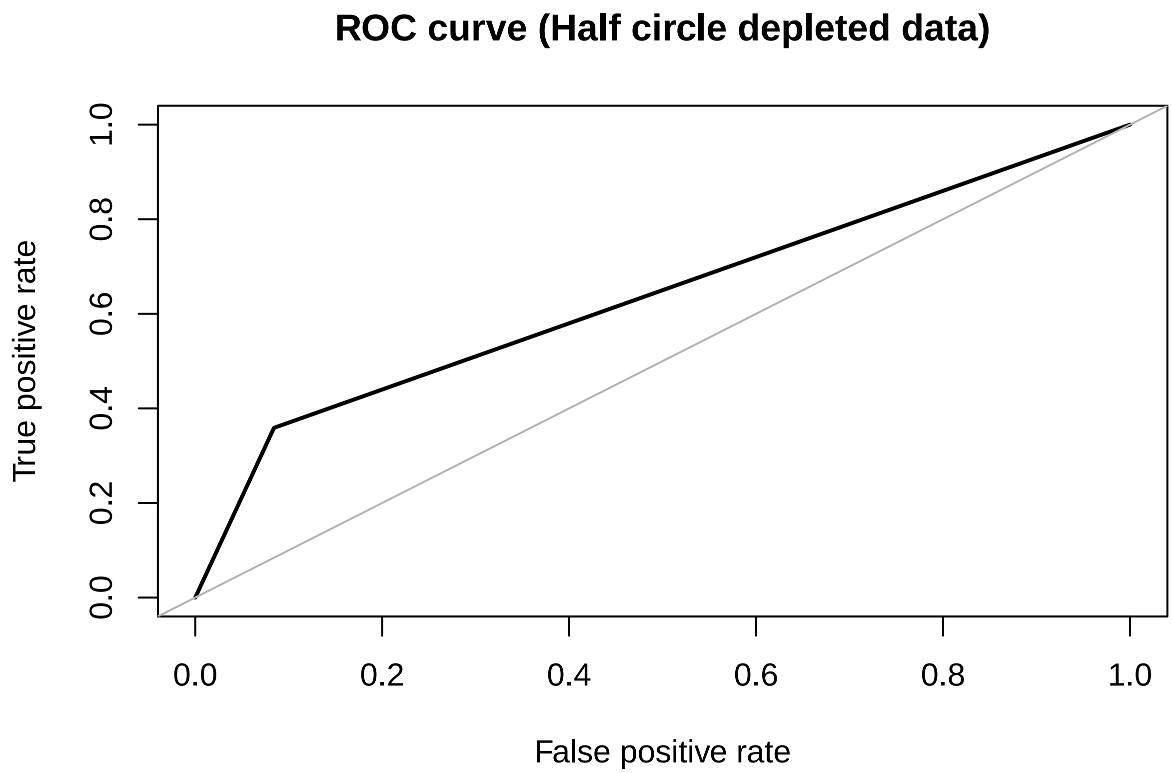
```
## [1] 0.9121491
```

```
#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
#predRf = predict(fitRf, newdata=test.matrix)
predSvm = predict(fitSvm, newdata=test.matrix)

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  511 134
##        Yes  47  75
##
##                Accuracy : 0.764
##                  95% CI : (0.7323, 0.7937)
##     No Information Rate : 0.7275
##     P-Value [Acc > NIR] : 0.01201
##
##                   Kappa : 0.3157
##
##  Mcnemar's Test P-Value : 1.634e-10
##
##             Sensitivity : 0.9158
##             Specificity : 0.3589
##          Pos Pred Value : 0.7922
##          Neg Pred Value : 0.6148
##              Prevalence : 0.7275
```

```
##          Detection Rate : 0.6662
##    Detection Prevalence : 0.8409
##       Balanced Accuracy : 0.6373
##
##        'Positive' Class : No
##
```

```
roc.curve(test$label, predSvm, main="ROC curve (Half circle depleted data)")
```




```
## Area under the curve (AUC): 0.637
```

```
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)
```

```r
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:

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

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

```
## [1]  801 4904
```

```r
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:

```r
#install.packages('ggpubr')
library(mlr)
```

```
## Loading required package: ParamHelpers
```

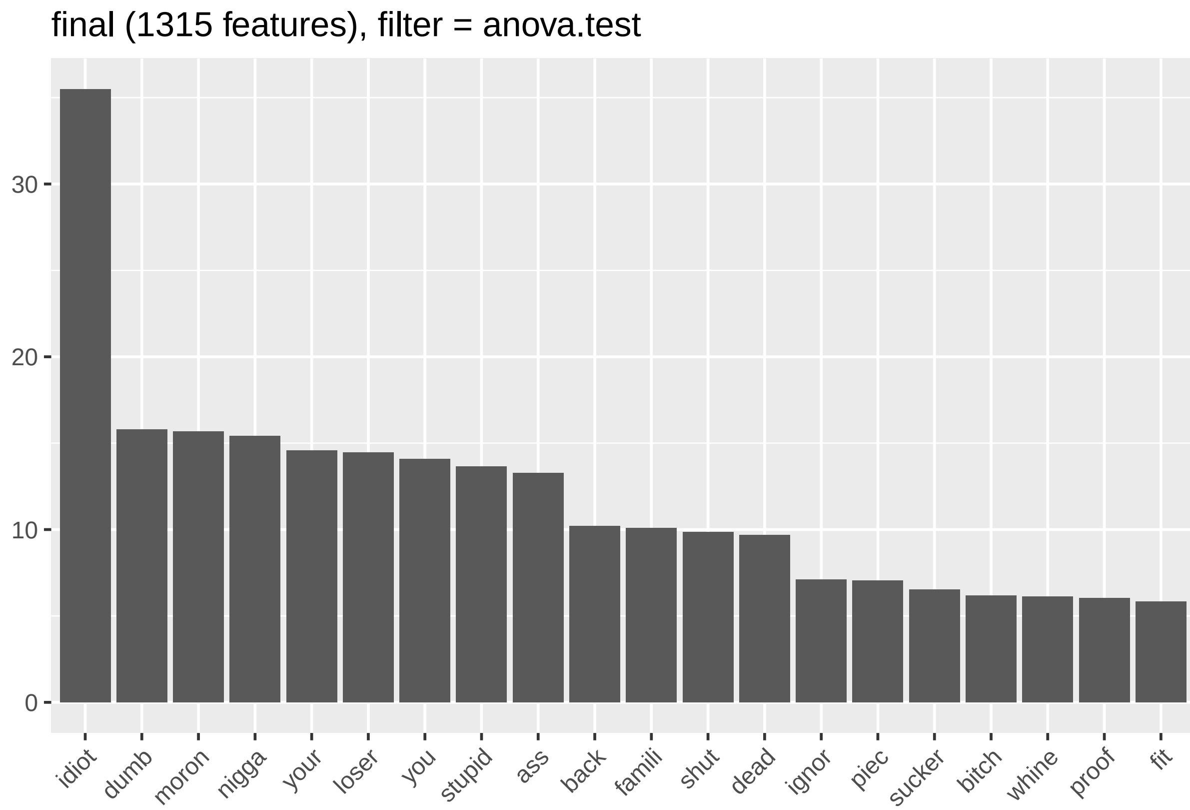
```
##
## Attaching package: 'mlr'
```

```
## The following object is masked from 'package:caret':
##
##     train
```

```r
library(ggpubr)
```

```
## Loading required package: magrittr
```

```r
train$lbl <- train$label
avector <- as.vector(train['lbl'])
final <- cbind(tfidf.matrix, avector)
final <- as.data.frame(final)
final$lbl=ifelse(final$lbl==0,"No","Yes")
final$lbl <- as.factor(final$lbl)
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)
```



Re-evaluate the models performance for top n features

```
features <- fv$data$name
n_features <- c(head(features, 20), 'lbl')
train.data <- final[,n_features]

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

```
## [1] 0.8637727
```

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

```
## Confusion Matrix and Statistics
##
##             Reference
```

```
## Prediction  No Yes
##        No  543 147
##        Yes  15  62
##
##                Accuracy : 0.7888
##                  95% CI : (0.7582, 0.8172)
##     No Information Rate : 0.7275
##     P-Value [Acc > NIR] : 5.575e-05
##
##                   Kappa : 0.3362
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9731
##             Specificity : 0.2967
##          Pos Pred Value : 0.7870
##          Neg Pred Value : 0.8052
##              Prevalence : 0.7275
##          Detection Rate : 0.7080
##    Detection Prevalence : 0.8996
##       Balanced Accuracy : 0.6349
##
##        'Positive' Class : No
##
```

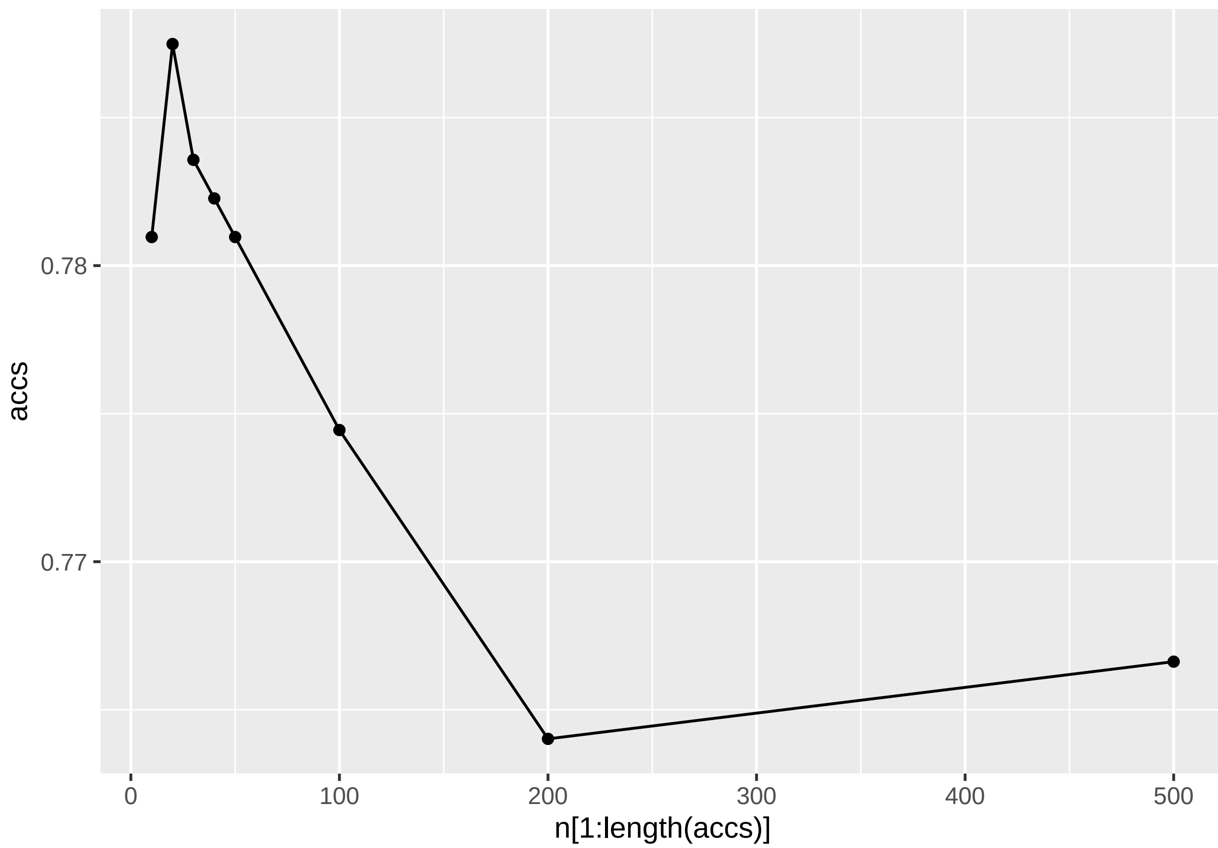
```
print(con.matrix.svm$overall[1])
```

```
##  Accuracy
## 0.7887875
```

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

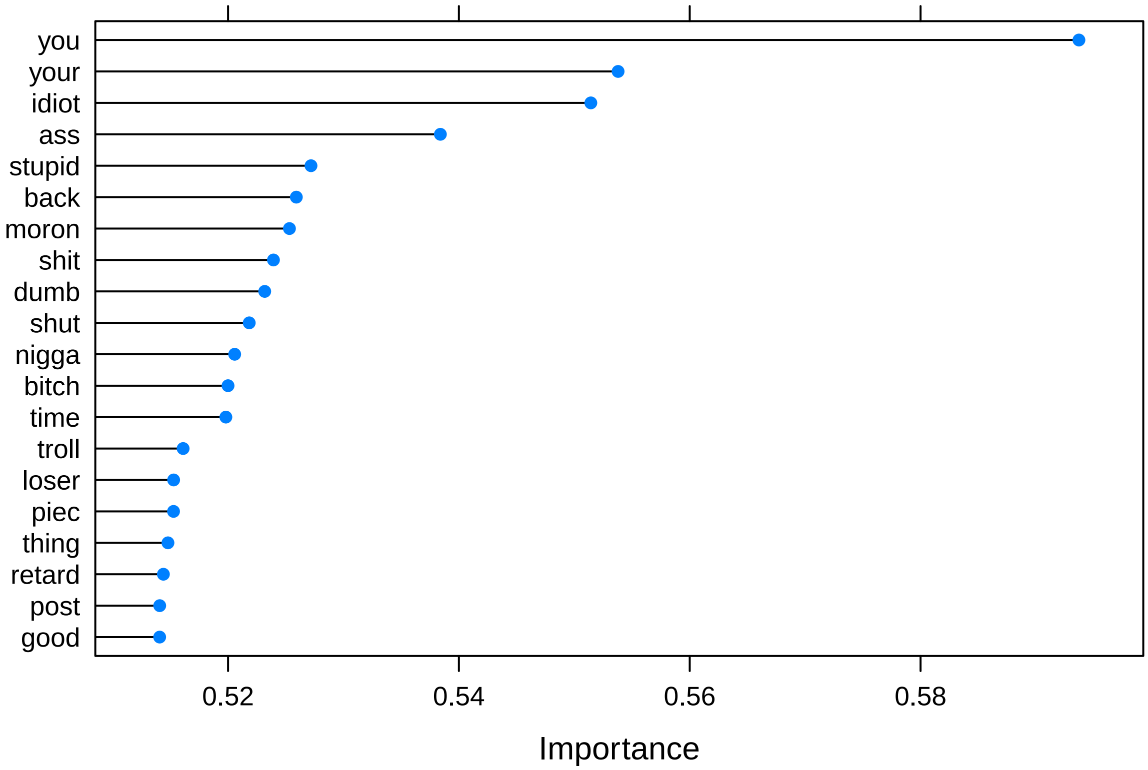
```
library(ggplot2)
features <- fv$data$name
n <- c(seq(10, 50, 10), 100, 200, 500)
accs <- c()
for (i in 1:length(n)) {
  n_features <- c(head(features, n[i]), 'lbl')
  train.data <- final[,n_features]

  fitSvm <- caret::train(lbl ~ ., data = train.data, method = 'svmLinear',scale=F, tuneLength = 5, metr
               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)
```

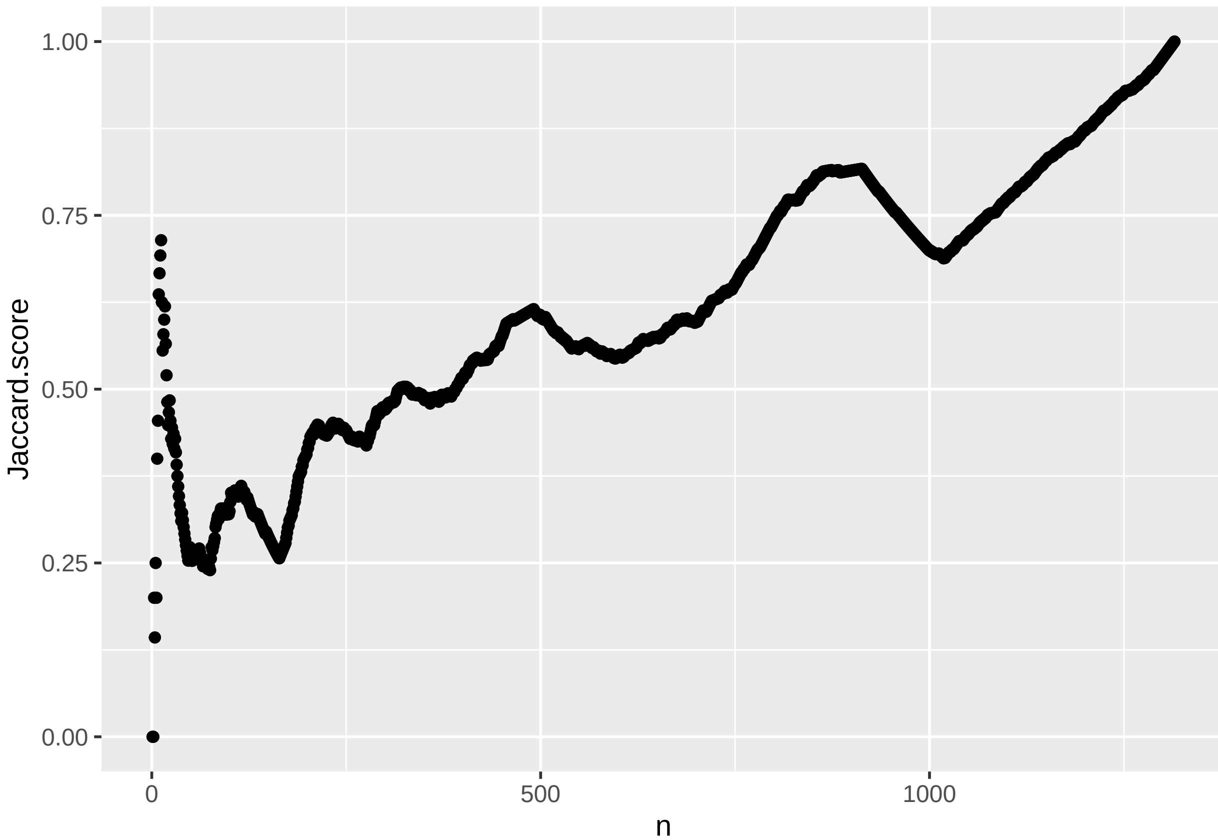


Compare the two feature rankings

```r
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
```

```r
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)
```



1.00
0.75
0.50
0.25
0.00
Jaccard.score
0
500
1000
n