Weekly Assignment 5

Loïc Roldán Waals & Kimberly Cheng

Wednesday 25 April, 2018

Question 1

Using a database with at least 200 selected tweets from previous weekly assignments, clean up your data set and build a term frequency matrix.

In this question we just need the text part of last weeks assignment. Before we answer the sub-questions, we first import the data and make a *corpus* and *document term matrix* (DTM). these will then be used for the next questions.

Amongst the words that were removed were the search terms *artificial, intelligence* and *artificialintelligence*, and several usernames.

```
#import the text + the sentiment
library(readr)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(jsonlite)
library(tm)
```

```
## Loading required package: NLP
```

```
library(SnowballC)

AITweets <- read_delim("C:/Users/Loic RW/Google Drive/Big Data and Business Analytics/Assignments/Assig
                        ">", escape_double = FALSE, trim_ws = TRUE)
```

```
## Parsed with column specification:
## cols(
##   text = col_character(),
##   retweetCount = col_integer(),
##   count = col_integer()
## )
```

```
#remove emoji's
AITweets$text <- sapply(AITweets$text, function(row) iconv(row, "latin1", "ASCII", sub = ""))

#create corpus
vecdata <- VectorSource(AITweets$text)
myCorpus <- VCorpus(vecdata)


#remove the links
removeLinks <- function (x) {
  gsub("http [^[:blank:]]+","",x)
}
myCorpus <- tm_map (myCorpus, content_transformer(removeLinks))

#only lower case
tweets_corpus_clean <- tm_map(myCorpus, content_transformer(tolower))

#remove punctuation
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removePunctuation)

#remove numbers
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removeNumbers)

#remove stopwords
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removeWords, stopwords("english"))
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removeWords, c("userexperienceu", "artificial", "inte

#remove unnecessary white space
tweets_corpus_clean <- tm_map(tweets_corpus_clean, stripWhitespace)

#stemming
tweets_corpus_clean <- tm_map(tweets_corpus_clean, stemDocument, language = "english")

myCorpus <- tweets_corpus_clean


#make term frequency matrix
dtm <- TermDocumentMatrix(tweets_corpus_clean)
dtm
```

```
## <<TermDocumentMatrix (terms: 2467, documents: 1021)>>
## Non-/sparse entries: 9078/2509729
## Sparsity           : 100%
## Maximal term length: 42
## Weighting          : term frequency (tf)
```

Part (a)

(a)Cluster the resulting documents using the k-means algorithm with a suitable distance metric. Explain in plain text the information potentially encoded in each cluster. Tip: you can use decision trees to help you with this task.

-> explain why this metric is used!

We first remove the spare items to allow for a more easily understandable analysis. Then we plot a cluster dendogram to see if we can see any natural clusters forming. The euclidean distance metric was used. It appears that 3 groups have formed, these were labeled with the red boxes.

The first cluster appears to discuss machine learning and the new capabilities of robots (thanks to artificial intelligence). The second cluster is solely defined by the word *human*. This cluster could be encoding the comparison between humans and AI (e.g. when they become smarter than us, if they can act like us). The final cluster on the right seems to be about the (potential) behaviour of AI, but this is highly speculative.

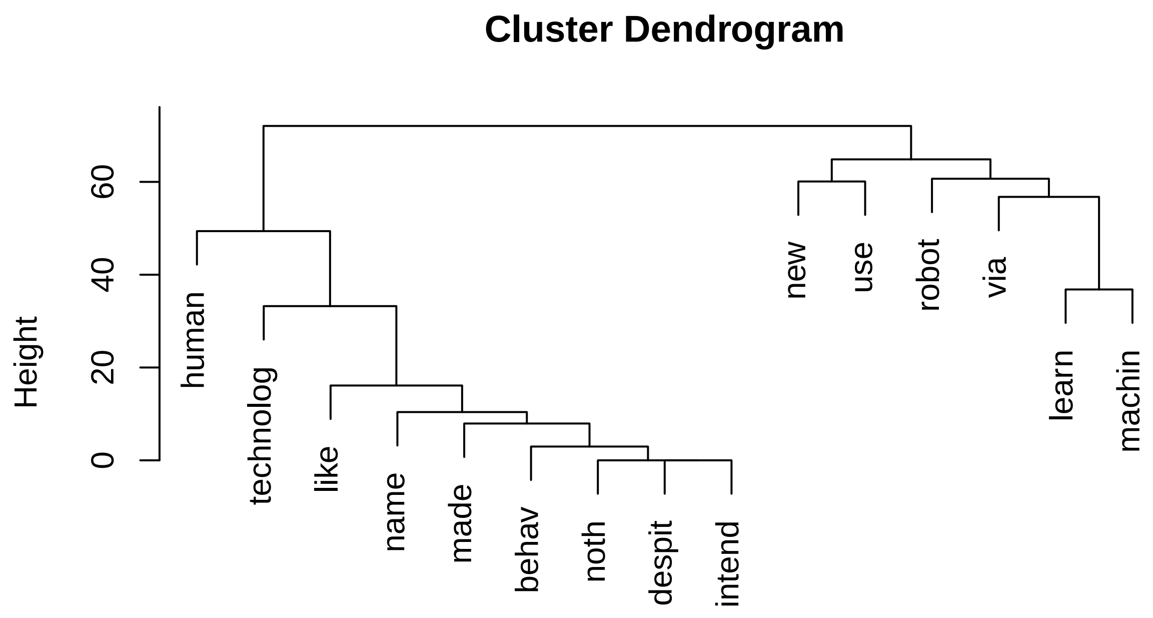
Finally the elbow method is used to determine whether our choice of 3 clusters seems reasonable. Plotting the within groups sum of squares against the number of clusters shows different values every time, but the large drop is always between 2 and 5 clusters. This means our choice of k=3 is reasonable.

We finally finish with a clusterplot that shows that two components explain 67.9% of the point variability. Additionally, there is not much overlap between the clusters. A decision tree will also be created to show how each cluster can be attained.

```
set.seed(156)

#load libraries
library(cluster)

#remove sparce words
dtm2 <- removeSparseTerms(dtm, sparse = 0.95)
dtm2
```

```
## <<TermDocumentMatrix (terms: 15, documents: 1021)>>
## Non-/sparse entries: 1193/14122
## Sparsity           : 92%
## Maximal term length: 9
## Weighting          : term frequency (tf)
```

```
dtm2 <- as.matrix(dtm2)
distmatrix <- dist(scale(dtm2))

fit <- hclust(distmatrix)

plot(fit)
```

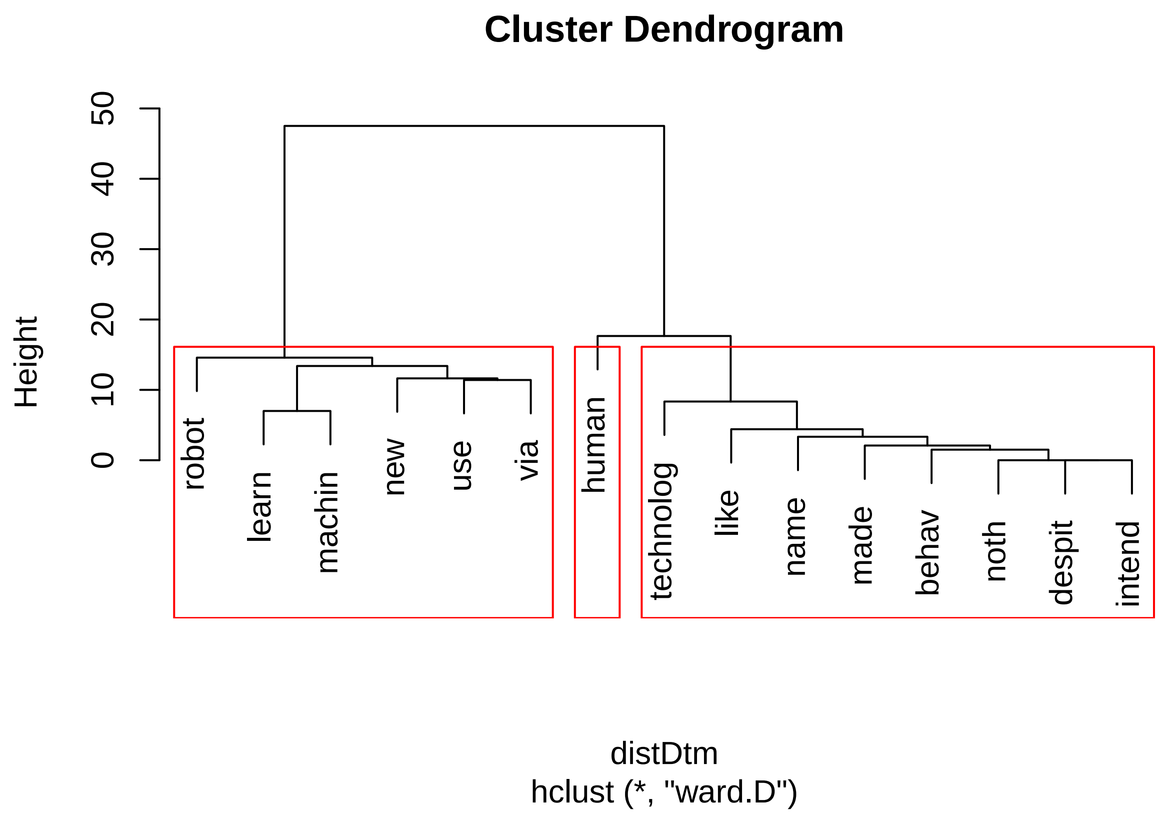


distmatrix
hclust (*, "complete")

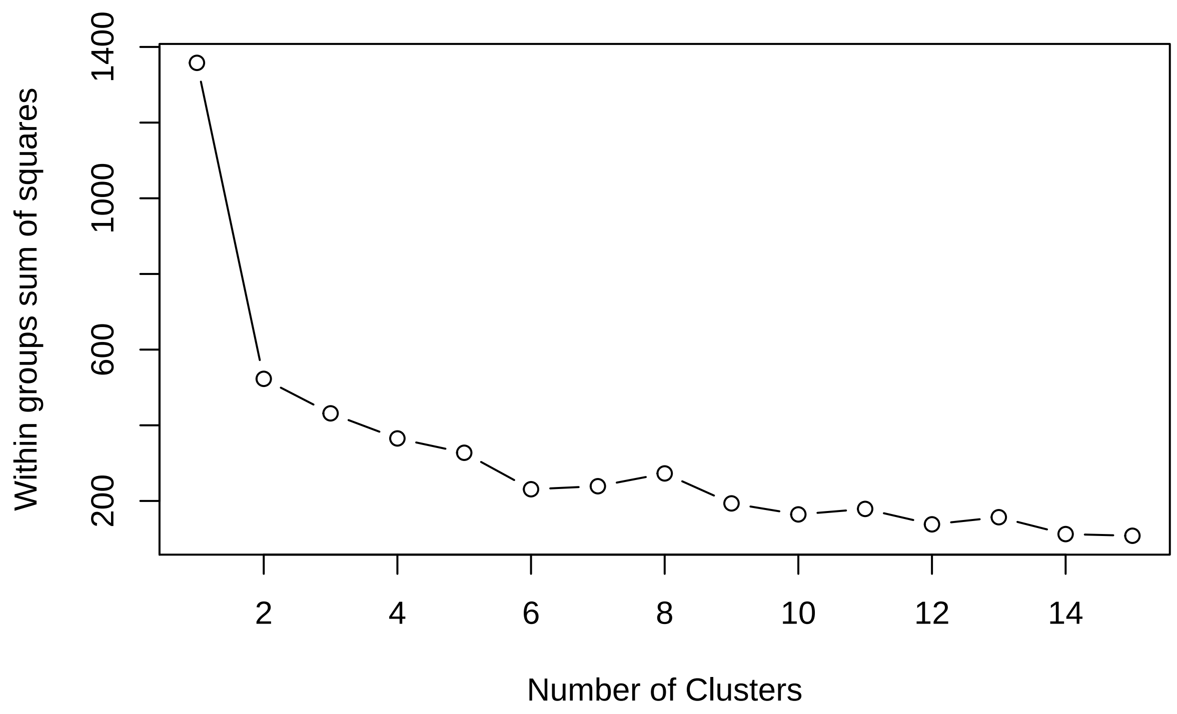
```
clusters = 3


#k means clustering
distDtm <- dist(dtm2 , method = "euclidean")

groups <- hclust(distDtm, method = "ward.D")
groups
```

```
##
## Call:
## hclust(d = distDtm, method = "ward.D")
##
## Cluster method   : ward.D
## Distance         : euclidean
## Number of objects: 15
```

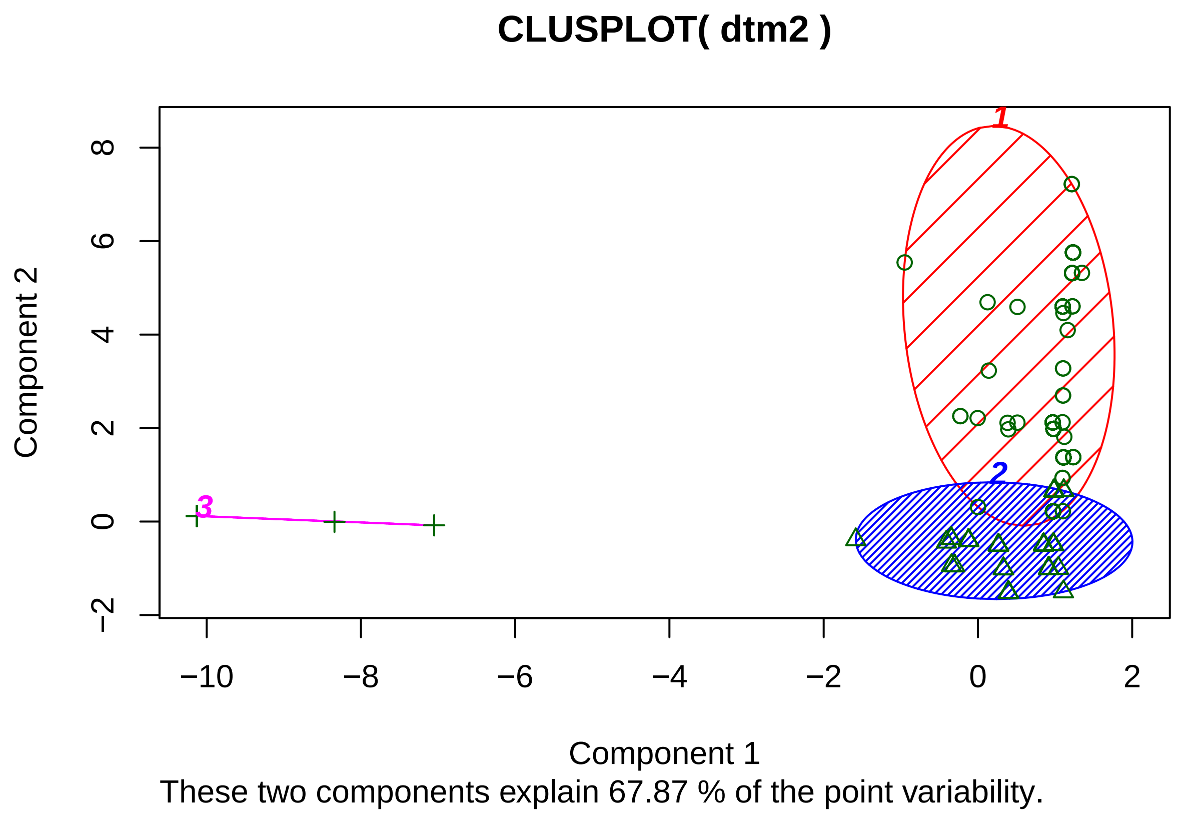
```
plot(groups)
rect.hclust(groups, k=clusters)
```



```r
dtm2 <- t(dtm2)


# Using elbow method to determine number of clusters
x <- data.frame(dtm2)
wss <- (nrow(x)-1)*sum(apply(x,2,var))
for (i in 2:15) wss[i] <- sum(kmeans(x, centers = i)$withinss)

plot(1:15, wss, type="b", xlab="Number of Clusters", ylab="Within groups sum of squares")
```

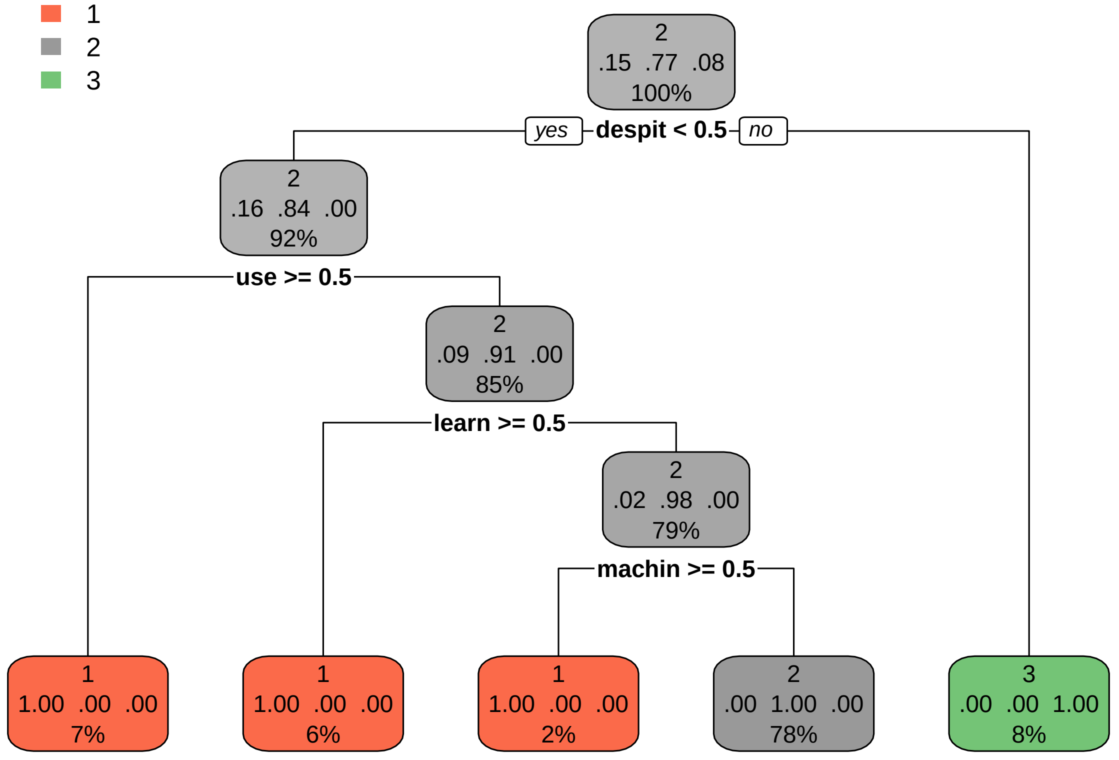


```
#clusterplot principle component analysis
rsltKmeans <- kmeans(dtm2, clusters)
clusplot(dtm2, rsltKmeans$cluster, color = TRUE, shade = TRUE, labels = 4, lines = 0)
```



It seems like the model checks whether the stem word *despit* is used, in which case it belongs to the third cluster.Then it checks Whether any of the words *use*, *learn* or *machin* are being used, if any of them are used, it belongs to the first cluster. Anything else and it belongs to the second cluster, the one that solely contained the word *human*.

```
library(rpart)
library(rpart.plot)

#add clusters to the existing dataframe
dfcluster <- data.frame(dtm2, k = as.factor(rsltKmeans$cluster))

#train the tree
tree <- rpart(k ~ ., data = dfcluster, method = "class")
rpart.plot(tree)
```



Part (b)

(b) Manually label all the tweets in your data set as positive (label = 1), negative (label = -1), and neutral (label = 0). Thereafter, using 10-fold cross validation, report the overall accuracy of a Naive Bayes model when predicting the target (labels). Tip: you might (or might not) have to reduce the sparsity of the term frequency matrix to get higher accuracy.

We will now proceed with the sentiment analysis for twitter. We first import the file and clean the text in the same way as in **Part (a)**. After we have obtained the DTM we append the manual sentiment labels to create a dataframe that contains the usage of the terms and the sentiment as columns. Here we find the first variable where we have a choice, *sparcity*. Using a trail an error method, we found that the optimal value for this is 0.96.

We then apply ten fold cross validation. Here we find the second variable we can tweak, the threshold of the prediction for the Naive Bayes model. Again, using the trail and error method we found a optimal threshold of 0.7. Finally to evaluate the performance we use accuracy and a confusion matrix. The code below was used:

```
#load libraries
library("e1071")
library("plyr")
library(naivebayes)
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

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

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
#set the values for the sparcity and the threshold for NB (easier tweaking)
spar = 0.96
nbt = 0.7



#import the dataset and create corpus
AITweets <- read_delim("C:/Users/Loic RW/Google Drive/Big Data and Business Analytics/Assignments/Assig
                       ">", escape_double = FALSE, trim_ws = TRUE)
```

```
## Parsed with column specification:
## cols(
##   text = col_character(),
##   retweetCount = col_integer(),
##   count = col_integer(),
##   sentiment = col_integer()
## )
```

```r
AITweets$sentiment <- as.factor(AITweets$sentiment)

#remove emoji's
AITweets$text <- sapply(AITweets$text, function(row) iconv(row, "latin1", "ASCII", sub = ""))

#create corpus
vecdata <- VectorSource(AITweets$text)
myCorpus <- VCorpus(vecdata)

#remove the links
removeLinks <- function (x) {
  gsub("http [^[:blank:]]+","",x)
}
```

```r
myCorpus <- tm_map (myCorpus, content_transformer(removeLinks))

#only lower case
tweets_corpus_clean <- tm_map(myCorpus, content_transformer(tolower))

#remove punctuation
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removePunctuation)

#remove numbers
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removeNumbers)

#remove stopwords
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removeWords, stopwords("english"))
tweets_corpus_clean <- tm_map(tweets_corpus_clean, removeWords, c("userexperienceu", "artificial", "int

#remove unnecessary white space
tweets_corpus_clean <- tm_map(tweets_corpus_clean, stripWhitespace)

#stemming
tweets_corpus_clean <- tm_map(tweets_corpus_clean, stemDocument, language = "english")

myCorpus <- tweets_corpus_clean



#make term frequency matrix
dtm <- TermDocumentMatrix(tweets_corpus_clean)
dtm2 <- removeSparseTerms(dtm, sparse = spar)
dtm2 <- t(as.matrix(dtm2))

#turn the dtm into a data frame
dfsentiment <- data.frame(dtm2)
dfsentiment$target <- as.factor(AITweets$sentiment)

#delete any potential missing values due to improper manual labeling
dfsentiment <- dfsentiment[complete.cases(dfsentiment),]

#Randomize the data
dfsentiment <- dfsentiment[sample(1:nrow(dfsentiment)), ]



#Create 10 equally sized folds
nFolds <- 10
folds <- cut(seq(1, nrow(dfsentiment)),
             breaks = nFolds,
             labels = FALSE)

#Vectors to store the results initialized with zeros
pdNB <- NULL
actual <- NULL

#define the model
```

```r
mdl <- as.factor(target) ~ .

#Perform 10 fold cross validation
for(i in 1:nFolds){

  #Segment the data by fold using which-function
  testIndexes <- which(folds == i, arr.ind = FALSE)
  testData <- dfsentiment[testIndexes, ]
  trainData <- dfsentiment[-testIndexes, ]

  #train the model
  rsltNB <- naiveBayes(mdl, data = trainData)

  #Make predictions on the test set
  NBClass <- predict(rsltNB, testData, type="class", threshold = nbt)

  #save the results to calculate accuracy later on
  pdNB <- c(as.character(pdNB), as.character(NBClass))
  actual <- c(actual, as.character(testData$target))
}

meanAccNB = mean(pdNB == actual)
meanAccNB
```

```
## [1] 0.6967509
```

```r
actual <- as.factor(actual)
pdNB <- as.factor(pdNB)

confusionMatrix(pdNB, actual, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  -1   0   1
##         -1   0   4   4
##         0   11 144  40
##         1    1  24  49
##
## Overall Statistics
##
##                Accuracy : 0.6968
##                  95% CI : (0.6389, 0.7503)
##     No Information Rate : 0.6209
##     P-Value [Acc > NIR] : 0.005062
##
##                   Kappa : 0.3574
##  Mcnemar's Test P-Value : 0.028418
##
## Statistics by Class:
##
##                      Class: -1 Class: 0 Class: 1
## Sensitivity            0.00000   0.8372   0.5269
## Specificity            0.96981   0.5143   0.8641
## Pos Pred Value         0.00000   0.7385   0.6622
## Neg Pred Value         0.95539   0.6585   0.7833
## Prevalence             0.04332   0.6209   0.3357
## Detection Rate         0.00000   0.5199   0.1769
## Detection Prevalence   0.02888   0.7040   0.2671
## Balanced Accuracy      0.48491   0.6757   0.6955
```

As can be seen the accuracy of the model is 69.8%, simply choosing the most common factor would result in an accuracy of 62.1%. This shows that our model does not show great performance, but it is better than simply predicting neutral all the time. In the confusion matrix we see that the model classifies 0 most of the time correctly (83.7%). The sensitivity class 1 and -1 is poor, with 52.7% and 0% respectively.