Practice 4

Smit Patil

—Problem 1—-

Build an R Notebook of the SMS message filtering example in the textbook on pages 103 to 123 (data set). Show each step and add appropriate documentation. Note that the attached data set differs slightly from the one used on the book; the number of cases differ.

```
#Step 1 & 2: Collecting, Exploring and Preparing and Data

#Importing Data
sms_data <- read.csv("da5030-spammsgdataset.csv")

#Looking at the structure of the data
str(sms_data)
```

```
## 'data.frame':    5574 obs. of  2 variables:
##  $ type: chr  "ham" "ham" "spam" "ham" ...
##  $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Ci
```

```
#Coverting the type column to factors
sms_data$type <- as.factor(sms_data$type)

#Checking whether the as.factor function is applied properly
str(sms_data$type)
```

```
##  Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
```

```
#Creating a table of ham and spam
table(sms_data$type)
```

```
##
##  ham spam
## 4827  747
```

```
#Data preparation - processing text data for analysis

#Importing libararies to create corpus of the spam data
library(NLP)
library(tm)

#Creating corpus and printing the results
sms_corpus <- Corpus(VectorSource(sms_data$text))
print(sms_corpus)
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 5574
```

```
#Inspecting the corpus data
inspect(sms_corpus[1:3])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 3
##
## [1] Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
## [2] Ok lar... Joking wif u oni...
## [3] Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
```

```
#Removing Numbers, Stopwords, Punctuation and Whitespace using tm_map() function
corpus_clean <- tm_map(sms_corpus, tolower)
```

```
## Warning in tm_map.SimpleCorpus(sms_corpus, tolower): transformation drops
## documents
```

```
corpus_clean <- tm_map(corpus_clean, removeNumbers)
```

```
## Warning in tm_map.SimpleCorpus(corpus_clean, removeNumbers): transformation
## drops documents
```

```
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
```

```
## Warning in tm_map.SimpleCorpus(corpus_clean, removeWords, stopwords()):
## transformation drops documents
```

```
corpus_clean <- tm_map(corpus_clean, removePunctuation)
```

```
## Warning in tm_map.SimpleCorpus(corpus_clean, removePunctuation): transformation
## drops documents
```

```
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
```

```
## Warning in tm_map.SimpleCorpus(corpus_clean, stripWhitespace): transformation
## drops documents
```

```
#Inspecting data after cleaning
inspect(corpus_clean[1:3])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 3
##
## [1] go jurong point crazy available bugis n great world la e buffet cine got amore wat
## [2] ok lar joking wif u oni
## [3] free entry wkly comp win fa cup final tkts st may text fa receive entry questionstd txt ratetcs
```

```r
#Splitting the sentences into words using DocumentTermMatrix() function
sms_dtm <- DocumentTermMatrix(corpus_clean)
```

```r
#Splitting the sms_data in 75:25 ratio and create train and test objects
sms_train_data <- sms_data[1:4181, ]
sms_test_data <- sms_data[4182:5574, ]

#Similarly we split tokenized data into train and test objects
sms_train_dtm <- sms_dtm[1:4181, ]
sms_test_dtm <- sms_dtm[4182:5574, ]

#Similarly we split corpus data into train and test objects
sms_train_corpus <- corpus_clean[1:4181]
sms_test_corpus <- corpus_clean[4182:5574]

#Comparing the proportion of spam in the training and test data frames
prop.table(table(sms_train_data$type))
```

```
##
##       ham      spam
## 0.8648649 0.1351351
```

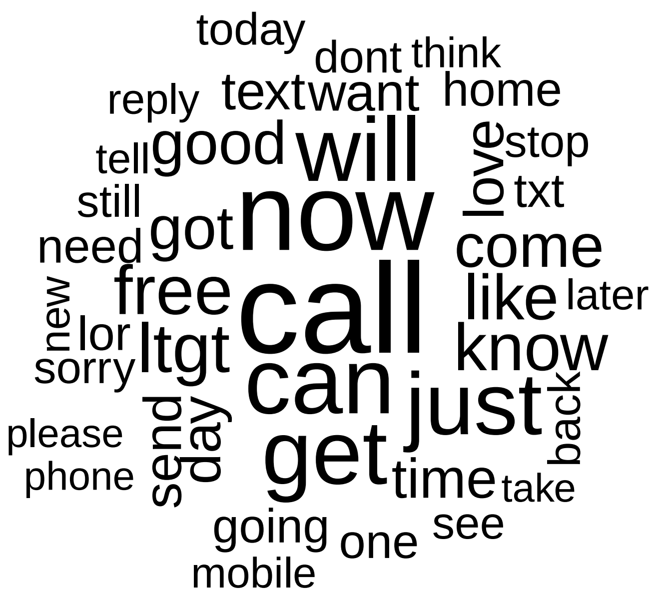
```r
prop.table(table(sms_test_data$type))
```

```
##
##       ham      spam
## 0.8693467 0.1306533
```

```r
#Visualizing text data - word clouds

#Importing libraries to create wordcloud of the sms data
library(RColorBrewer)
library(wordcloud)
library(stringr)

#A wordcloud is created using the train corpus data, we set the minimum word frequency as 40
wordcloud(corpus_clean, min.freq = 120, random.order = FALSE)
```




```r
#Now to visualize spam and ham of train data seperately we create a subset of them individually
spam <- subset(sms_data, type == "spam")
ham <- subset(sms_data, type == "ham")
spam$text <- str_replace_all(spam$text,"[^[:graph:]]", " ")
ham$text <- str_replace_all(ham$text,"[^[:graph:]]", " ")

#Visualization of spam and ham individually and we set the maximum words as 40 most common words
wordcloud(spam$text, max.words = 40, scale = c(3,0.5))
```

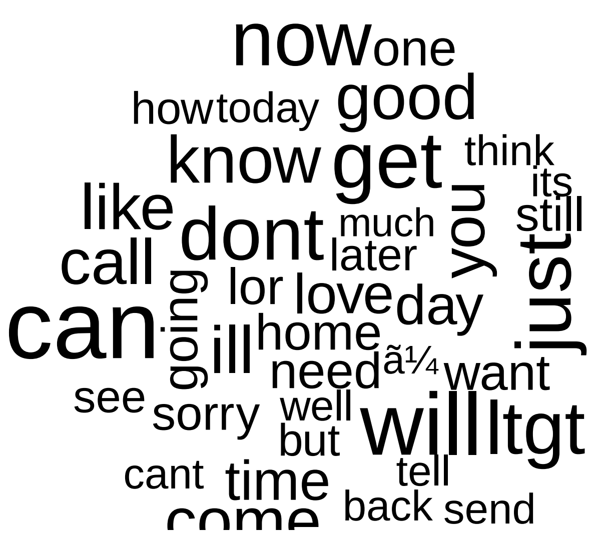
```
## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation): transformation
## drops documents
```

```
## Warning in tm_map.SimpleCorpus(corpus, function(x) tm::removeWords(x,
## tm::stopwords())): transformation drops documents
```


```r
wordcloud(ham$text, max.words = 40, scale = c(3,0.5))
```

```
## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation): transformation
## drops documents

## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation): transformation
## drops documents
```




```r
#Data preparation - creating indicator features for frequent words

library(tm)

#We find the word which have a frequency of 5 or more using findFreqTerms() function from tm library an
sms_dict <- findFreqTerms(sms_train_dtm, 5)
head(sms_dict)
```

```
## [1] "available" "bugis"     "cine"      "crazy"     "got"       "great"
```

```r
#We create a sparse matrix of both train and test corpus data which have frequent words
sms_train <- DocumentTermMatrix(sms_train_corpus, list(dictionary = sms_dict))
sms_test <- DocumentTermMatrix(sms_test_corpus, list(dictionary = sms_dict))

#convert_counts functions is used to convert sparse matrix element numbers to a factor with Yes and No 
convert_counts <- function(x)
  {
    x <- ifelse(x > 0, 1, 0)
    x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
    return(x)
  }

#Using apply() function we convert the sparse matrix elements by calling the convert_counts() function
sms_train <- apply(sms_train, MARGIN = 2, convert_counts)
sms_test <- apply(sms_test, MARGIN = 2, convert_counts)
```

```r
#Step 3 - training a model on the data

#Importing library to use naiveBayes() function
library(e1071)

#First we build our model using naiveBayes() function from the e1071 library. We use the training data
sms_classifier <- naiveBayes(sms_train, sms_train_data$type)
```

```r
#Step 4 - evaluating model performance

#Importing libraries to use predict() function for model evaluation
library(gmodels)

#Here for prediction we have used testing sms data along with the predict() function to evaluate the pe
sms_test_pred <- predict(sms_classifier, sms_test)

#To calculate the accuracy of the model we generate a crosstable
CrossTable(sms_test_pred, sms_test_data$type, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted','a
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1393
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1205 |        28 |      1233 |
##              |     0.977 |     0.023 |     0.885 |
##              |     0.995 |     0.154 |           |
## -------------|-----------|-----------|-----------|
##         spam |         6 |       154 |       160 |
##              |     0.037 |     0.963 |     0.115 |
##              |     0.005 |     0.846 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1211 |       182 |      1393 |
##              |     0.869 |     0.131 |           |
## -------------|-----------|-----------|-----------|
##
##
```

```r
#Step 5 - improving model performance

#We try to improve the performance of the model by using laplace = 1 in the naiveBayes() function. It h
sms_classifier2 <- naiveBayes(sms_train, sms_train_data$type, laplace = 1)
```

```
#We test the new improved model
sms_test_pred2 <- predict(sms_classifier2, sms_test)

#We use crosstable to observe the improved performance of the model. We can observe that number of ham
CrossTable(sms_test_pred2, sms_test_data$type, prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE, dnn
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1393
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1189 |        10 |      1199 |
##              |     0.982 |     0.055 |           |
## -------------|-----------|-----------|-----------|
##         spam |        22 |       172 |       194 |
##              |     0.018 |     0.945 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1211 |       182 |      1393 |
##              |     0.869 |     0.131 |           |
## -------------|-----------|-----------|-----------|
##
##
```

—Problem 2—

Install the requisite packages to execute the following code that classifies the built-in iris data using Naive Bayes. Build an R Notebook and explain in detail what each step does. Be sure to look up each function to understand how it is used.

```
#Importing libraries to test Naive Bayes using klaR package
library(MASS)
library(klaR)

#Loading the built-in iris dataset
data(iris)

#Calculating number of rows in iris dataset
nrow(iris)
```

```
## [1] 150
```

```r
#Summarising the iris dataset
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
## 
```

```r
#Printing the header of the iris dataset
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
#Selecting every 5th number between 1 and 150 (i.e. 20% of the dataset)
testidx <- which(1:length(iris[, 1]) %% 5 == 0)

#Creating training and testing dataset
iristrain <- iris[-testidx,]
iristest <- iris[testidx,]

#Applying the Naive Bayes algorithm from the klaR package, using the Species as the categorical variabl
nbmodel <- NaiveBayes(Species~., data=iristrain)

#Check the accuracy
#Prediction of the model is done using predict() function
prediction <- predict(nbmodel, iristest[,-5])
table(prediction$class, iristest[,5])
```

```
##             
##              setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         2
##   virginica       0          0         8
```

```r
#Printing the accuracy
accuracy <- ((10+10+8)/(30))*100
sprintf("The accuracy of the model is %s percent",accuracy)
```

```
## [1] "The accuracy of the model is 93.3333333333333 percent"
```