Practice4_DA5030

Shivam Verma

```r
library(tm)
```

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

```r
library(SnowballC)
```

```
## Warning: package 'SnowballC' was built under R version 3.6.2
```

```r
library(wordcloud)
```

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

```r
library(e1071)
library(gmodels)
```

Problem - 1

Step-2 Exploring and Preparing the data

```r
# reading the dataset
sms_raw <- read.csv("spammsg.csv", stringsAsFactors = FALSE)
# exploring characteristics of data
str(sms_raw)
```

```
## '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... C
```

```r
sms_raw$type <- factor(sms_raw$type)
table(sms_raw$type)
```

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

Data Preperation - cleaning and standardizing text data

```r
# creating a volatile corpus
sms_corpus <- Corpus(VectorSource(sms_raw$text))
#printing the result
print(sms_corpus)
```

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

```r
#summarize specific messages
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
```

```r
# standardizing the messages to lowercase characters
corpus_clean <- tm_map(sms_corpus, tolower)
```

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

```r
# removing all the numbers from the corpus
corpus_clean <- tm_map (corpus_clean , removeNumbers)
```

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

```r
# removing the stopwords
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
```

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

```r
# removing the punctuation
corpus_clean <- tm_map(corpus_clean, removePunctuation)
```

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

```r
# removing the white spaces
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
```

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

```r
# stemming variants of words
corpus_clean <- tm_map(corpus_clean, stemDocument)
```

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

```r
inspect(corpus_clean[1:3])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 3
##
## [1] go jurong point crazi avail bugi n great world la e buffet cine got amor wat
## [2] ok lar joke wif u oni
## [3] free entri wkli comp win fa cup final tkts st may text fa receiv entri questionstd txt ratetc app
```

```r
# creating DTM by applying tokenization
sms_dtm <- DocumentTermMatrix(corpus_clean)
```

Data Preperation - creating trianing and test datasets

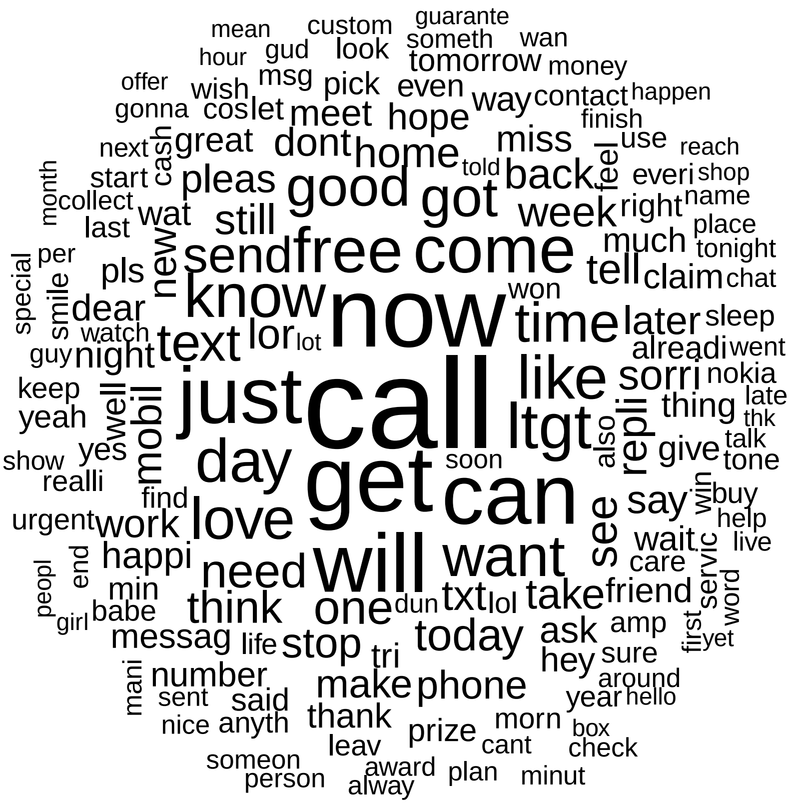
```r
# dividing the data into two portions: 75 percent for training and 25 percent for testing
sms_raw_train <- sms_raw[1:4180,]
sms_raw_test <- sms_raw[4181:5574,]
# creating test and train labels
sms_train_label <- sms_raw[1:4180,]$type
sms_test_label <- sms_raw[4181:5574,]$type
sms_dtm_train <- sms_dtm[1:4180,]
sms_dtm_test <- sms_dtm[4181:5574,]
sms_corpus_train <- corpus_clean[1:4180]
sms_corpus_test <- corpus_clean[4181:5574]
prop.table(table(sms_raw_train$type))
```

```
##
##       ham      spam
## 0.8648325 0.1351675
```

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

```
##
##       ham      spam
## 0.8694405 0.1305595
```

```r
# Visualizing text data - word clouds
wordcloud(corpus_clean, min.freq = 50, random.order = FALSE)
```




```r
# making subsets of raw data by the type
spam <- subset(sms_raw, type=="spam")
ham <- subset(sms_raw, type=="ham")
# creating word cloud for each subset
wordcloud(spam$text, max.words = 40, scale= c(3,0.5))
```

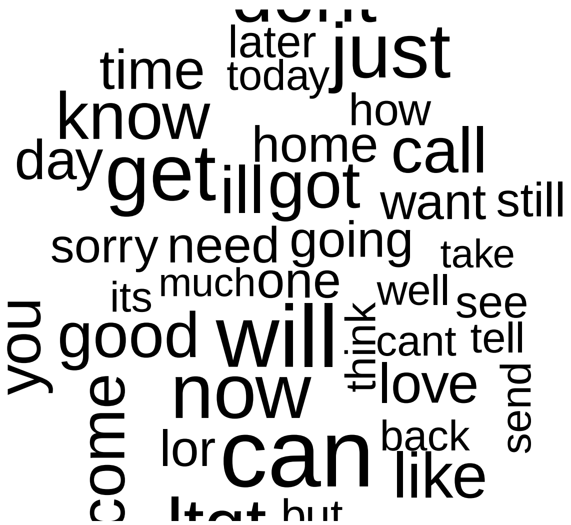
```
## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation): transformation
## drops documents
```

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

```
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
```



Data preperation - creating indicator features for frequent words

```
# eliminating words that appear in less than 5 SMS messages
sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
str(sms_freq_words)
```

```
##  chr [1:1164] "avail" "bugi" "cine" "crazi" "got" "great" "point" "wat" ...
```

```
# filtering DTM test and train data
sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
# function to convert counts to Yes/No strings
convert_counts <- function(x)
{
  x <- ifelse(x>0, "Yes", "No")
}
# Applying the function to all columns
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
sms_test <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```

Step 3 - training a model on the data

```
# Applying Naive Bayes algorithm
sms_classifier <- naiveBayes(sms_train, sms_train_label)
# Making predictions
sms_test_pred <- predict(sms_classifier, sms_test)
```

Step - 4 Evaluating model performance

```r
# generating cross table for performance evaluation
CrossTable(sms_test_pred, sms_test_label, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'actu
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1394
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1203 |        20 |      1223 |
##              |     0.984 |     0.016 |     0.877 |
##              |     0.993 |     0.110 |           |
## -------------|-----------|-----------|-----------|
##         spam |         9 |       162 |       171 |
##              |     0.053 |     0.947 |     0.123 |
##              |     0.007 |     0.890 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1212 |       182 |      1394 |
##              |     0.869 |     0.131 |           |
## -------------|-----------|-----------|-----------|
##
##
```

We can observe only 29 SMS predicted wrong. thus accuracy is approx 98%.

Step 5 - improving the model performance

```r
# trying to improve performance by setting Laplace estimator
sms_classifier2 <- naiveBayes(sms_train, sms_train_label, laplace = 1)
sms_test_pred2 <- predict(sms_classifier2, sms_test)
CrossTable(sms_test_pred, sms_test_label, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'actu
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
```

```
##
## Total Observations in Table:  1394
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1203 |        20 |      1223 |
##              |     0.984 |     0.016 |     0.877 |
##              |     0.993 |     0.110 |           |
## -------------|-----------|-----------|-----------|
##         spam |         9 |       162 |       171 |
##              |     0.053 |     0.947 |     0.123 |
##              |     0.007 |     0.890 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1212 |       182 |      1394 |
##              |     0.869 |     0.131 |           |
## -------------|-----------|-----------|-----------|
##
##
```

It is obsereved the performance remained same even after introducing the Laplace estimator.

Problem - 2

```r
library(klaR)
```

```
## Loading required package: MASS
```

```r
# loading the iris dataset of R
data(iris)
# finding the number of rows in iris data
nrow(iris)
```

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

```r
# Getting the summary of data
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
# Creating a view of data
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
# This statement acts as a loop from 1 to length of dataset to fetch numbers which are divisible by 5.
testidx <- which(1:length(iris[, 1]) %% 5 == 0)

# creating train dataset excluding those rows which are included in test dataset
iristrain <- iris[-testidx,]
# creating test dataset with the previously fetched numbers divisible by 5
iristest <- iris[testidx,]

# applying Naive Bayes Algorithm
nbmodel <- NaiveBayes(Species~., data=iristrain)

# checking for the accuracy
# Using the test data to make prediction for species of test data
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
```

It is observed that there are only two misclassifications which gives accuracy of about 93%