Practice-4

Harsh

13/06/2020

---Problem 1: SMS message filtering---

Step 1 & 2 – collecting, exploring and preparing the data

```
#Importing Data using read.csv() function
sms_data <- read.csv("C:\\Users\\harsh\\Desktop\\Introduction to Machine learning and Data Mining\\Pract

#Exploring Data using head and str function. We can see that we have 2 features with 5574 number of tot
head(sms_data)
```

```
##   type
## 1  ham
## 2  ham
## 3 spam
## 4  ham
## 5  ham
## 6 spam
##
## 1                                             Go until jurong point, crazy.. Available only in bugis
## 2
## 3 Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive en
## 4
## 5                                                                                                Nah
## 6        FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up
```

```
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... C
```

```
#sms_data$type is character vector, since it is a categorical variable we convert it to factors with 2
sms_data$type <- as.factor(sms_data$type)

#verifying the datatype using str() function
str(sms_data)
```

```
## 'data.frame':    5574 obs. of  2 variables:
##  $ type: Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
##  $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... C
```

```r
#We count the total spam and ham messages using table function
table(sms_data$type)
```

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

Data preparation – processing text data for analysis

```r
#The tm text mining package is installed using install.packages() function
#install.packages("tm")
library(tm)
```

```
## Warning: package 'tm' was built under R version 3.6.3
```

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

```r
#We create a collection of text documents called  corpus. Since we have used vector data we use VectorS
sms_corpus <- Corpus(VectorSource(sms_data$text))

#Using print we get a statement as A corpus with 5574 text documents
print(sms_corpus)
```

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

```r
#To observe the content we use inspect() function
inspect(sms_corpus[1:2])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 2
##
## [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...
```

```r
#We remove all the numbers and punctuations using tm_map() function. It is used to transform data.
corpus_clean <- tm_map(sms_corpus, tolower)
```

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

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

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

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

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

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

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

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

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

```r
#We verify using inspect whether all unwanted characters are removed
inspect(corpus_clean[1:2])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 2
##
## [1] go jurong point crazy available bugis n great world la e buffet cine got amore wat
## [2] ok lar joking wif u oni
```

```r
#Now we split sentences into individual words by using the process of tokenization. This is done by usi
sms_dtm <- DocumentTermMatrix(corpus_clean)
```

Data preparation – creating training and test datasets

```r
#We split 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]

#We compare 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
```

Visualizing text data – word clouds

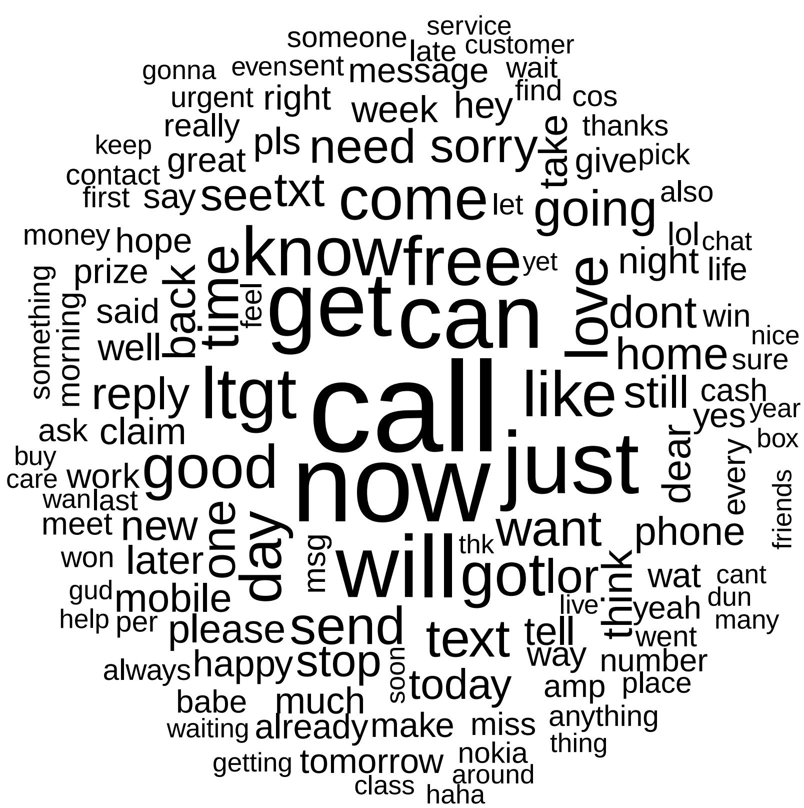
```r
#Using the wordcloud package we visually depict the frequency at which words appear in text data.
#install.packages("wordcloud")
library(wordcloud)
```

```
## Warning: package 'wordcloud' was built under R version 3.6.3
```

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

```r
library(stringr)

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



```r
#Now to visualize spam and ham of train data seperately we create a subset of them individually
spam <- subset(sms_train_data, type == "spam")
ham <- subset(sms_test_data, type == "ham")

#Since an error is generated because of a unknown graph element we replace that using str_replace funct
#Solution provided by Annie Bryant
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))
```

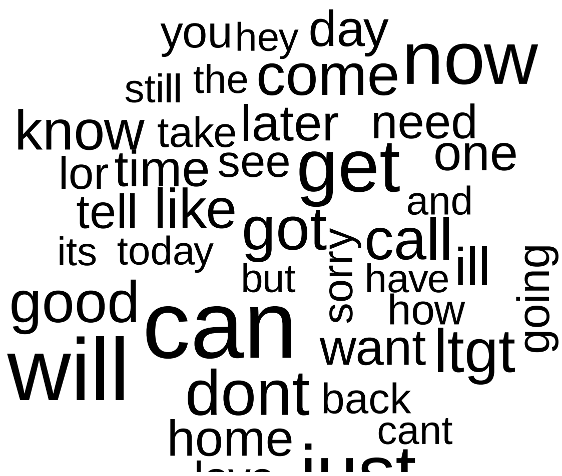
```
## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation): transformation
## drops documents

## 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
```




```
#We can observe that the most frequently used words in spam are call, free, stop, prize.
```

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"
```

```
#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
```

```r
sms_train <- apply(sms_train, MARGIN = 2, convert_counts)
sms_test <- apply(sms_test, MARGIN = 2, convert_counts)
```

Step 3 – training a model on the data

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

```
## Warning: package 'gmodels' was built under R version 3.6.3
```

```r
#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)
```

Step 4 – evaluating model performance

```r
library(e1071)
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. We can observe that 6 of the ham mess
CrossTable(sms_test_pred, sms_test_data$type, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted',
```

```
##
##
##    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 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Step 5 – improving model performance

```r
#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 |      1207 |        30 |      1237 |
##              |     0.997 |     0.165 |           |
## -------------|-----------|-----------|-----------|
##         spam |         4 |       152 |       156 |
##              |     0.003 |     0.835 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1211 |       182 |      1393 |
##              |     0.869 |     0.131 |           |
## -------------|-----------|-----------|-----------|
##
##
```

---Problem 2: Classification of the built-in iris data using Naive Bayes---

```r
#We test the naiveBayes function using a different library called klaR package
#install.packages("klaR")
library(klaR)
```

```
## Warning: package 'klaR' was built under R version 3.6.3
```

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

```r
#Loading the built-in dataset iris. We observe that the data consists of 5 features namely Sepal.Length
data(iris)

#Calculating the total number of rows present in the iris data using nrow() function
nrow(iris)
```

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

```r
#Summary function helps in providing a detailed statistics of the data. It shows the mean,median,min,ma
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
#Head is used to show the top 6 rows of the data. This helps in exploring the 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
#With the help of which() function and a logic which basically means that every fifth row is stored in
testidx <- which(1:length(iris[, 1]) %% 5 == 0)


#Separate into training and testing datasets
#Training data makes use of 80% of the data. This is done by using inverse of the testidx.
iristrain <- iris[-testidx,]

#In testing data we use testidx which is 20% of the data.
iristest <- iris[testidx,]

#Apply Naive Bayes
#Using the NaiveBayes() function from the klaR library. We specify the target variable i.e species and
nbmodel <- NaiveBayes(Species~., data=iristrain)

#Check the accuracy
#Prediction of the model is done using predict() function and we use the testing data without the last
prediction <- predict(nbmodel, iristest[,-5])

#To calculate the accuracy we create a table to observe actual and predicted values
table(prediction$class, iristest[,5])
```

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

```r
#We can see that only 2 virginica flowers were predicted as versicolor. We get the accuracy as 93.33%
acc <- ((10+10+8)/(10+10+10))*100
sprintf("The accuracy of the model is %s",acc)
```

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