DA5030.A4.Parpattedar

Shruti Parpattedar

February 19, 2019

Question 1

Step 1 - Collecting Data

Reading and storing the spam-ham dataset in a data frame

```
spamdata <- read.csv("da5030.spammsgdataset.csv", header = TRUE, stringsAsFactors = FALSE)
```

Step 2 - Exploring and preparing the data

Exploring the dataset and storing the type column as a factor

```
str(spamdata)
```

```
## '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... Cine
```

```
spamdata$type <- factor(spamdata$type)
str(spamdata$type)
```

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

```
table(spamdata$type)
```

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

Step 3 - Data preparation

Cleaning and standardizing text data

```
library(tm)
```

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

```
# VCorpus refers to a volatile source ie stored on memory
# Vector source is being to used to load the text column of the loaded dataset
sms_corpus <- VCorpus(VectorSource(spamdata$text))
print(sms_corpus)
```

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

```
inspect(sms_corpus[1:2])
```

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

```
## Metadata:  7
## Content:  chars: 111
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 29
```

```
# Viewing the text part of the first row of the corpus and then applying the same function
# to the first 2 rows
as.character(sms_corpus[[1]])
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
```

```
lapply(sms_corpus[1:2], as.character)
```

```
## $`1`
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
##
## $`2`
## [1] "Ok lar... Joking wif u oni..."
```

```
# Start of data cleaning
# Converting all the text to lower case for standardization
corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
as.character(sms_corpus[[1]])
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
```

```
as.character(corpus_clean[[1]])
```

```
## [1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine there got
```

```
# Removing all the numbers from the text
# getTransformations()
corpus_clean <- tm_map(corpus_clean, removeNumbers)

# Removing stopwords like: i, me, is, was, should, etc
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())

# Removing all the punctuation marks from the text
corpus_clean <- tm_map(corpus_clean, removePunctuation)

# Loading the SnowballC library which provides us with the wordStem function
# which returns the root of the provided word
library(SnowballC)
wordStem(c("learn", "learned", "learning", "learns"))
```

```
## [1] "learn" "learn" "learn" "learn"
```

```
# Applying stemDocument to replace the derived words by their roots
corpus_clean <- tm_map(corpus_clean, stemDocument)

# Removing the extra whitespace that was introduced due to the text cleaning
corpus_clean <- tm_map(corpus_clean, stripWhitespace)

# Observing the difference between the original and clean data
lapply(sms_corpus[1:3], as.character)
```

```
## $`1`
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
```

```
##
## $`2`
## [1] "Ok lar... Joking wif u oni..."
##
## $`3`
## [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive en
```

```
lapply(corpus_clean[1:3], as.character)
```

```
## $`1`
## [1] "go jurong point crazi avail bugi n great world la e buffet cine got amor wat"
##
## $`2`
## [1] "ok lar joke wif u oni"
##
## $`3`
## [1] "free entri wkli comp win fa cup final tkts st may text fa receiv entri questionstd txt ratetc appl
```

Step 4 - Splitting text documents into words

Generating Document Term Matrices (DTM)

```
# Using the cleaned corpus from the previous step
sms_dtm <- DocumentTermMatrix(corpus_clean)

# Cleaning the corpus while generating the DTM
sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
  tolower = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  removePunctuation = TRUE,
  stemDocument = TRUE
))

# Observing the difference between the 2 DTMs
sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6611)>>
## Non-/sparse entries: 42653/36807061
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
sms_dtm2
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 8371)>>
## Non-/sparse entries: 44243/46615711
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
# Trying to make the second DTM more similar to the first
sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
  tolower = TRUE,
  removeNumbers = TRUE,
  stopwords = function(x) { removeWords(x, stopwords()) },
  removePunctuation = TRUE,
  stemDocument = TRUE
))
```

```
# Observing the difference between the 2 DTMs
sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6611)>>
## Non-/sparse entries: 42653/36807061
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
sms_dtm2
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 7964)>>
## Non-/sparse entries: 43136/44348200
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

Step 5 - Creating training and test datasets

Creating training and testing datasets and their labels. Using prop.table to check the proportions of spam and ham in both the subsets

```
# Approximately 75% of the data is taken as the training set and 25% is taken as the testing set
sms_dtm_train <- sms_dtm[1:4180, ]
sms_dtm_test <- sms_dtm[4181:5574, ]

sms_train_labels <- spamdata[1:4180, ]$type
sms_test_labels <- spamdata[4181:5574, ]$type

# Checking the proportions of spam and ham in both the subsets
prop.table(table(sms_train_labels))
```

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

```
prop.table(table(sms_test_labels))
```

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

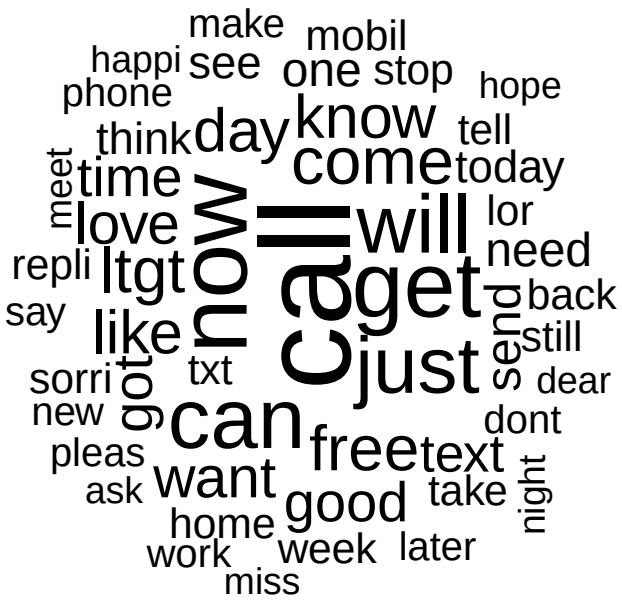
Step 6 - Visualizing text data

Generating word clouds for the text data

```
library(wordcloud)
```

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

```
# Using the cleaned corpus to generate the word cloud of spam and ham texts
wordcloud(corpus_clean, min.freq = 120, random.order = FALSE)
```




```r
# Dividing the dataset based on their type
spam <- subset(spamdata, type == "spam")
Encoding(spam$text) <- ("UTF-8")
ham <- subset(spamdata, type == "ham")

# Generating separate word clouds for spam and ham texts
wordcloud(spam$text, max.words = 50, scale = c(3, 0.5))
```

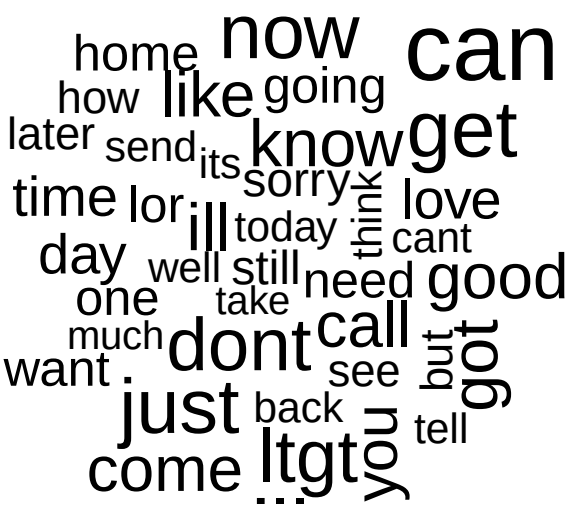
```
## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation):
## transformation drops documents
```

```
## 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
```



Step 7 - Creating indicator features for frequent words

Generating DTMs containing words that occur 5 or more number of times Further, converting counts to yes/no values

```
# findFreqTerms(sms_dtm_train, 5)
sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
str(sms_freq_words)
```

```
##  chr [1:1163] "â£wk" "â\200¦" "â\200"" "abiola" "abl" "abt" "accept" ...
```

```
# Train and test DTM with words of freq 5 or more
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
convert_counts <- function(x)
{
 x <- ifelse(x > 0, "Yes", "No")
}

# Applying function to train and test sets
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
sms_test <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```

Step 8 - Training a model on the data

Generating a Naives Bayes classifier based on the train dataset

```
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

Step 9 - Evaluating model performance

Using the classifer to make predictions, using the test dataset Result shows that (9+20) ie 29 out of 1394 meassages were incorrectly classified.

```
library(gmodels) # Required for CrossTable function
sms_test_pred <- predict(sms_classifier, sms_test)

CrossTable(sms_test_pred, sms_test_labels,
 prop.chisq = FALSE, prop.t = FALSE,
 dnn = c('predicted', 'actual'))
```

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

Step 10 - Improving model performance

Adding the laplace parameter to improve model performance

```
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels,
 laplace = 1)

sms_test_pred2 <- predict(sms_classifier2, sms_test)

CrossTable(sms_test_pred2, sms_test_labels,
 prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
 dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1394
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1205 |        28 |      1233 |
##              |     0.994 |     0.154 |           |
## -------------|-----------|-----------|-----------|
##         spam |         7 |       154 |       161 |
##              |     0.006 |     0.846 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1212 |       182 |      1394 |
##              |     0.869 |     0.131 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Question 2

Using the Naive Bayes function from the klaR package on the iris dataset.

The iris data set comprises of 150 observations with Sepal.length, Sepal.Width, Petal.length, Petal.Width and Species attributes. There are 50 observations for each of the three species - Setosa, Versicolor and Virginica. Species attribute is a factor variable.

In order to divide the dataset uniformly, every fifth number is selected between 1 and 150. Observations of these row numbers are taken as the test dataset. The remaining 120 rows are taken as the training dataset.

The Naive Bayes algorithm is applied on training dataset using Species as the categorical variable and the rest as independent predictor variables. This model is then used to make predictions on the testing dataset.

A tabular representation of the predicted versus the actual values shows that only two observations, of Virginica species were predicted as belonging to Versicolor species, out of the 30 observations in the dataset. This is an efficient model, with an error of 2/30 or 6%.

```
# Loading the klaR package for the Naive Bayes classification function
library(klaR)
```

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

```
# Loading the iris dataset
data(iris)

# Exploring the irir dataset
nrow(iris)
```

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

```
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(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. number of rows in the dataset)
testidx <- which(1:length(iris[, 1]) %% 5 == 0)

# separate into training and testing datasets
# Selecting every 5th row for the test data set and all the other rows for the train dataset
iristrain <- iris[-testidx,]
iristest <- iris[testidx,]

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

# check the accuracy
# Making predictions using the nbmodel for the test dataset
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
```