

## Project: Naïve Bayes

**Objective:** to classify SMS messages as spam or not spam (ham)

**Data set:** SMSSpamCollection.txt – a sms messages collection downloaded from

<http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>

Prepare the environment, set the working directory, and load the libraries using the following:

```
> rm(list=ls()) #Clear the environment
> setwd("YOUR_PATH") #Set working directory for the assignment
> getwd() #Check working directory
[1] "YOUR_PATH"
>
> ###Use Naive Bayes to create a SMS spam filter
> ###Data set SMSSpamCollection.txt from http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection
>
> #Load packages
>
> library("tm")
> library("SnowballC")
> library("wordcloud")
> library("RColorBrewer")
> library("e1071")
> library("gmodels")
> library("ggplot2")
```

Next, I loaded the data (previously obtained at

<https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>) into a table using the following

code :

```
> #Load Data into a table
> raw.data <- read.table("SMSSpamCollection.txt", header = FALSE, sep="\t", quote="", stringsAsFactors = FALSE)
```

For convenience, I added column names:

```
> #Add column names
```

```
> names(raw.data) <- c("Label", "Text")
```

To make sure that the data was loaded correctly, I checked the structure of the data table and its content:

```
> #Check the structure and content of the data table
> summary(raw.data)
  Label      Text
Length:5574 Length:5574
Class :character Class :character
Mode  :character Mode  :character
> table(raw.data$Label)

ham spam
4827  747
```

The first column (“Label”) contains either “spam” or “ham” label, so it is convenient to convert it to factors:

```
> #Change Label type from char to factor
> raw.data$Label <- factor(raw.data$Label)
```

The data was loaded correctly. Since the sms spam data set was originally compiled from three different sources, I used `sample()` command to randomize it:

```
> #Randomize data using sample() command
> set.seed(2018)
> raw.data <- raw.data[sample(nrow(raw.data)),]
> view(raw.data) #Check data
```

Preview of the results is below:

	Label	Text
1874	ham	Oh ok i didnt know what you meant. Yep i am baby jontin
2585	ham	Goodmorning, today i am late for 1hr.
338	ham	Cool. So how come you havent been wined and dined b...
1100	ham	NO GIFTS!! You trying to get me to throw myself off a clif...
2642	ham	Pandy joined 4w technologies today.he got job..
1677	ham	Painful words- "I thought being Happy was the most to...
3379	ham	Yup. Wun believe wat? U really neva c e msg i sent shuhui?
724	ham	That is wondar full flim.
5336	ham	Neither [in stern voice] - i'm studying. All fine with me! N...
3044	ham	Slaaaaave ! Where are you ? Must I summon you to me al...
2202	ham	Haha... can... But i'm having dinner with my cousin...
3697	ham	Hello, As per request from &lt;
5463	spam	December only! Had your mobile 11mths+? You are entitl...
3772	ham	Love it! The girls at the office may wonder why you are s...
4482	ham	Y cant u try new invention to fly..i'm not joking.,
3526	ham	Yeah that'd pretty much be the best case scenario
1505	ham	Ill be there on &lt;

In order to proceed with the modeling, I constructed a corpus object using the VCorpus() function and used the VectorSource() function to create a source object from the existing Text field:

```
> #Create corpus using VCorpus() from the tm package
>
> corpus.data <- VCorpus(VectorSource(raw.data$Text))
```

To check the results, print the corpus:

```
> #Print the corpus
> print(corpus.data)
<<VCorpus>>
Metadata: corpus specific: 0, document level (indexed): 0
Content: documents: 5574
```

The corpus contains 5574 documents, one for each sms message in the original data set.

Then I used inspect() function to look at the first two elements of the corpus:

```
> inspect(corpus.data[1:2])
<<VCorpus>>
```

```
Metadata: corpus specific: 0, document level (indexed): 0
Content: documents: 2
```

```
[[1]]
<<PlainTextDocument>>
Metadata: 7
Content: chars: 55
```

```
[[2]]
<<PlainTextDocument>>
Metadata: 7
Content: chars: 37
```

The content of the first two messages:

```
> lapply(corpus.data[1:2], as.character)
$`1`
[1] "Oh ok i didnt know what you meant. Yep i am baby jontin"

$`2`
[1] "Goodmorning, today i am late for 1hr."
```

Next step is to transform the text while preparing it for analysis. I used the following code to convert all messages to lower case:

```
> ###Text transformations
>
> #Change to lower case
> corpus.clean <- tm_map(corpus.data, content_transformer(tolower))
```

Check results by comparing the same message before and after the command:

```
> #Check results
> #original message
> as.character(corpus.data[[1]])
[1] "Oh ok i didnt know what you meant. Yep i am baby jontin"
> #processed message
> as.character(corpus.clean[[1]])
[1] "oh ok i didnt know what you meant. yep i am baby jontin"
```

I removed all numbers using the following command:

```
> #Remove numbers
> corpus.clean <- tm_map(corpus.clean, removeNumbers)
```

In the next step I removed stop words using the following code:

```
> #Remove stop words
> corpus.clean <- tm_map(corpus.clean, removeWords, stopwords())
```

I used the removePunctuation function built-in tm package to remove all punctuation from the messages:

```
> #Remove punctuation
> corpus.clean<- tm_map(corpus.clean, removePunctuation)
```

Next, I substituted all words in the messages with their root form (stemming):

```
> #Stemming
> corpus.clean <- tm_map(corpus.clean, stemDocument)
```

In the next last transformation step I removed extra white spaces from the text messages:

```
> #Stemming
> corpus.clean <- tm_map(corpus.clean, stemDocument)
```

To check the results, I compare the first five messages before (corpus.data) and after the processing (corpus.clean):

```
> #Check results by comparing first five messages before and after processing
> lapply(corpus.data[1:5], as.character)
$`1`
[1] "Oh ok i didnt know what you meant. Yep i am baby jontin"

$`2`
[1] "Goodmorning, today i am late for 1hr."

$`3`
[1] "Cool. So how come you havent been wined and dined before?"

$`4`
[1] "NO GIFTS!! You trying to get me to throw myself off a cliff or something?"

$`5`
[1] "Pandy joined 4w technologies today.he got job.."

> lapply(corpus.clean[1:5], as.character)
$`1`
[1] "oh ok didnt know meant yep babi jontin"

$`2`
[1] "goodmorn today late hr"

$`3`
[1] "cool come havent wine dine"
```

```
$`4`
[1] "gift tri get throw cliff someth"

$`5`
[1] "pandi join w technolog today got job"
```

The above output showed that the text transformations worked as they were supposed to.

Next step is to create a Document-Term-Matrix using the following command:

```
> #Create DTM
> dtm.all <- DocumentTermMatrix(corpus.clean)
```

With created a matric with 5574 rows (one for each document) and 6452 columns

(unique words) as the following output shows:

```
> #Check results
> dtm.all
<<DocumentTermMatrix (documents: 5574, terms: 6452)>>
Non-/sparse entries: 41294/35922154
Sparsity           : 100%
Maximal term length: 40
weighting           : term frequency (tf)
```

In order to proceed with the model, we need to split the data into the training and testing data sets. Since the initial data was randomized, we can just split it into two parts for training (80%) and testing purposes (20%) without running risk of misrepresenting either spam or ham messages.

```
> ###Split data into the training and testing sets
> dtm.train <- dtm.all[1:4459, ] #80% training data
> dtm.test <- dtm.all[4460:5574, ] #20% testing data
```

For training and testing the algorithm we will need vectors with the labels for each row:

```
> #Create vectors with labels for each row
> labels.train <- raw.data[1:4459,]$Label
> labels.test <- raw.data[4460:5574,]$Label
```

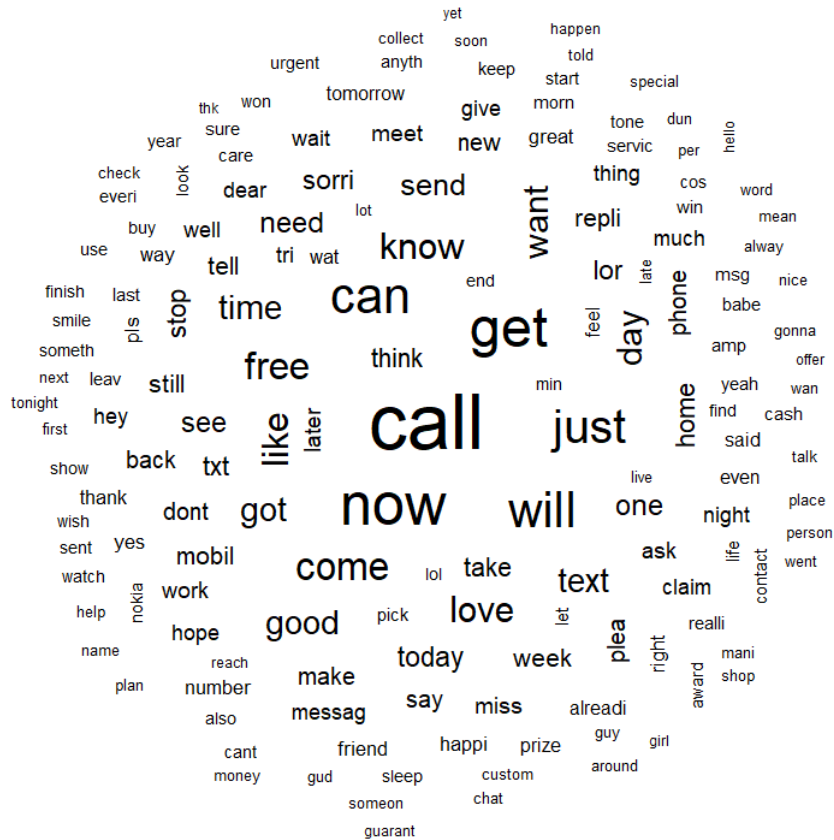
The proportions between the ham and spam in the training and testing data sets remained approximately the same:

```
> #Compare proportions between spam and ham in testing and training data
> prop.table(table(labels.train))
labels.train
      ham      spam
0.8697017 0.1302983
> prop.table(table(labels.test))
labels.test
      ham      spam
0.8511211 0.1488789
```

Next, I look at the most popular word in the text messages. Using the following code I created a word cloud for the 50 most popular words in the initial data set, including both spam and ham messages:

```
> #Top 50 words for the combine data set
> wordcloud(corpus.clean, min.freq = 50, random.order = FALSE)
```

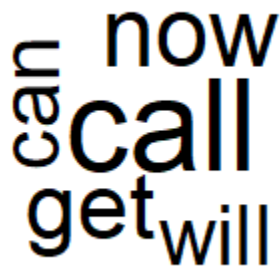
The output in below:



```
> #Top five words for all messages
> wordcloud(corpus.clean, max.words = 5, random.order = FALSE)
```

The five most popular words for all messages on average were the words “call”, “now”, “get”, “will” and “can”.





I wanted to see if the top five words are different for the spam and ham messages. So, I created two subsets (for spam and ham messages) and created separate word clouds for these two groups using the following code:

```
> #create a subset of spam messages
> spam <-subset(raw.data, Label == "spam")
> ham <- subset(raw.data, Label == "ham")
>
> #Create word clouds with the 5 most frequent words for spam and ham separately
> wordcloud(spam$Text, max.words = 5, colors=brewer.pal(8, "Dark2"))
Warning messages:
1: In tm_map.SimpleCorpus(corpus, tm::removePunctuation) :
  transformation drops documents
2: In tm_map.SimpleCorpus(corpus, function(x) tm::removewords(x, tm::stopwords())) :
  transformation drops documents
> wordcloud(ham$Text, max.words = 5, colors=brewer.pal(8, "Dark2"))
Warning messages:
1: In tm_map.SimpleCorpus(corpus, tm::removePunctuation) :
  transformation drops documents
2: In tm_map.SimpleCorpus(corpus, function(x) tm::removewords(x, tm::stopwords())) :
  transformation drops documents
```

The output for spam:



The output for ham messages:



The above output showed that out of top 5 messages one word – “now” is the same for both spam and ham messages. If we look on all most frequent words in all messages in the data set (see results below for frequency > 200) , then we notice that all five most popular words from the ham messages are included in this list. However, only three out of five most popular spam words are included in this list.

```
> #Find the most frequent terms in all messages
> findFreqTerms(dtm.all, lowfreq = 200)
[1] "call" "can" "come" "day" "free" "get" "good" "got" "just" "know" "like" "love" "now" "
text" "time" "want" "will"
```

To see how many times each popular word is found in the data set I used the following code:

```
> #Create a list of frequency words
> freq <- colSums(as.matrix(dtm.all))
> #Sort in descending order
> ord <- order(freq, decreasing = TRUE)
> #Display the most frequently occurring words
> freq[head(ord, 20)]
call now get can will just come free know like love day want got time good text send need o
ne
650 481 439 392 381 364 295 274 263 251 251 245 241 237 237 233 223 198 186 1
81
```

The most popular word “call” (650) appeared in the data set almost three times more than the tenth popular word “like” (251).

The least popular words may potentially contain interesting information, unique characteristics of a particular message, but as shown by the following output, in this case the list mostly included misspelled words:

```
> #Show the least frequently occurring words
> freq[tail(ord, 20)]
```

	youuuuu	youwanna	yovil	yowif	yoyyooo	ystrdayic	yummmm	yuou	yupz	z
ac	zاهر	zealand	zebra	zero						
	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1					
	zhong	zindgi	zogtorius	zoom	zouk	zyada				
	1	1	1	1	1	1				

Next, I proceeded to prepare the data for model fitting.

I created indicators for frequently used words:

```
> ###Creating indicators for frequent words
> train_freq_words <- findFreqTerms(dtm.train, 5)
> str(train_freq_words)
```

```
chr [1:1182] "âfwk" "â200|" "â200"" "abiola" "abl" "abt" "accept" "access" "account" "across"
"activ" "actual" "add" "address" ...
```

I used it to filter both training and testing data set to include only the words that are found at least 5 times.:

```
> #Filter DTM to keep only the words in train_freq_words
>
> dtm.train.freq <- dtm.train[, train_freq_words]
> dtm.test.freq <- dtm.test[, train_freq_words]
```

As Naïve Bayes model works best with categorical data, I created a function to convert counts for every word to a factor (with “Yes” meaning count > 0 and “No” meaning count = 0) and applied it to both training and testing sets.

```
> #Function to convert word counts to categorical
> convert_counts <- function(x) {
+   x <- ifelse (x > 0, "Yes", "No")
+ }
```

```

+ }
>
> #Convert the training data set
> dtm.train.model <- apply(dtm.train.freq, MARGIN = 2, convert_counts)
>
> #convert the testing data set
> dtm.test.model <- apply(dtm.test.freq, MARGIN=2, convert_counts)

```

Next step – model fitting and predictions. I used the following code:

```

> #Fit the model
> nb_model <- naiveBayes(dtm.train.model, labels.train)
> #Use the model for predictions
> test_prediction <- predict(nb_model, newdata = dtm.test.model)

```

To evaluate the results I used CrossTable() function from the gmodels package:

```

> #Evaluate model performance
> CrossTable(test_prediction, labels.test, prop.chisq = FALSE, prop.t = FALSE
, prop.r = FALSE, dnn = c("predicted", "actual"))

```

Cell Contents	
	N
N / Col Total	

Total Observations in Table: 1115

predicted	actual		Row Total
	ham	spam	
ham	944 0.995	17 0.102	961
spam	5 0.005	149 0.898	154
Column Total	949 0.851	166 0.149	1115

The above output shows that the model accurately predicted 944 ham and 149 spam messages. In means the overall accuracy of about 98.03%.  $((944 + 149) / 1115)$ . The model classified 5 ham messages as spam and 17 genuine messages were classified as spam. It shows a

very high accuracy of the model as overall less than two percent of all messages are classified incorrectly. For ham messages the accuracy is  $944/949 = 99.47\%$  and for the spam messages the accuracy was a little lower  $89.76\%$ .

In order to try to improve the accuracy even more, I fitted another model adding a Laplace estimator using the following commands:

```
> #Adjusting the model
> #Add Laplace estimator
>
> nb_model2 <- naiveBayes(dtm.train.model, labels.train, laplace = 1)
>
> #Use the second model for predictions
> test_prediction2 <- predict(nb_model2, newdata = dtm.test.model)
>
> #Evaluate performance for model 2
> CrossTable(test_prediction2, labels.test, prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("predicted", "actual"))
```

Cell Contents

	N
N / Col Total	

Total Observations in Table: 1115

predicted	actual		Row Total
	ham	spam	
ham	945 0.996	18 0.108	963
spam	4 0.004	148 0.892	152
Column Total	949 0.851	166 0.149	1115

The above output shows that the overall accuracy of the second model is  $(945+148)/1115 = 98.03\%$ , which is exactly the same as in the previous case. The difference is that is further improved accuracy for the ham messages (one less was misclassified as spam), but further

decreased accuracy for the spam messages. One more actual spam message got misclassified as ham. However, users are usually more tolerant to an occasional spam message getting onto their inbox compared to a good message being deleted or marked as spam. I would also emphasize that both models showed overall very impressive accuracy.

Simply increasing the training data set size may not bring considerable improvements in accuracy of the model. However, if we look at each type of the messages, there is some room for improvement in spam classification. Therefore, using a training set with additional spam messages might allow the algorithm to improve its classification accuracy.

Another potential way to increase accuracy is to experiment with data transformations that were applied before fitting the model. In text messages excessive punctuation plays a different role from punctuation in regular texts. In SMS it is sometimes used to convey additional meanings or intentionally mistype brand names etc., so simply deleting it might not be helping in classification of spam. Intentionally misspelled words can also be used in spam messages to circumvent spam detection, so finding a way to deal with it could potentially improve classification accuracy. In addition, a more compressive spam filter can be build using word associations or word combinations that are commonly used together in spam and in ham messages.