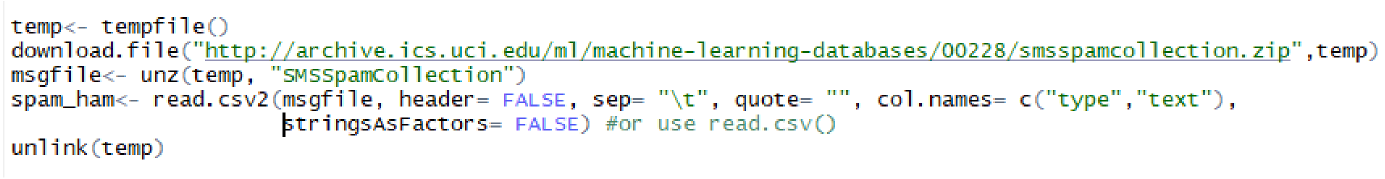
MSDS 680 Machine Learning
Alla Topp

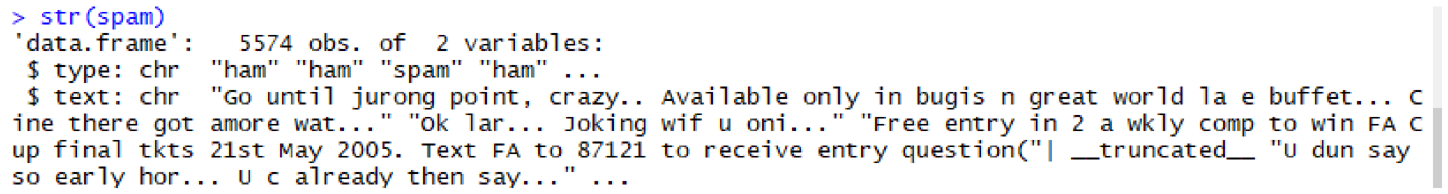
Objective: to classify SMS message as spam or not spam (ham).

Step 1 - collecting data



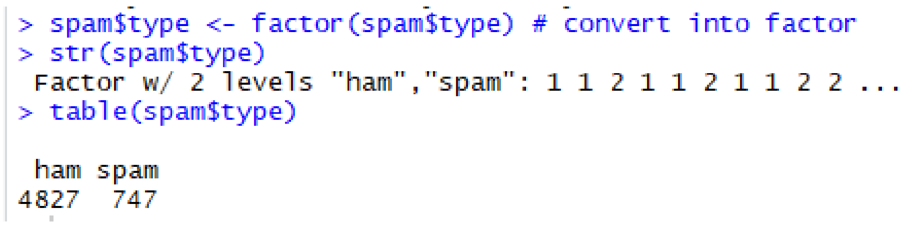
```
temp<- tempfile()
download.file("http://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip",temp)
msgfile<- unz(temp, "SMSSpamCollection")
spam_ham<- read.csv2(msgfile, header= FALSE, sep= "\t", quote= "", col.names= c("type","text"),
                     stringsAsFactors= FALSE) #or use read.csv()
unlink(temp)
```

Step 2 - exploring and preparing the data



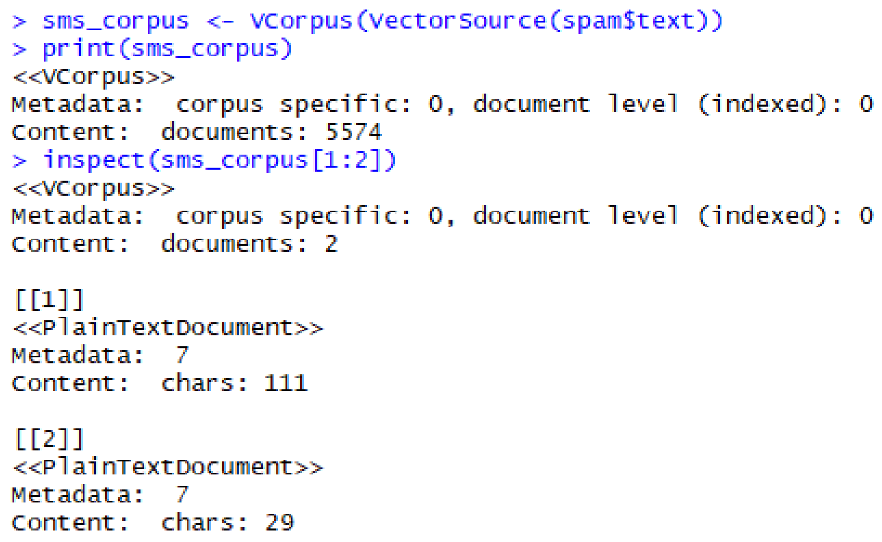
```
> str(spam)
'data.frame':	5574 obs. of  2 variables:
 $ type: chr  "ham" "ham" "spam" "ham" ...
 $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... C
ine there got amore wat..." "Ok lar... Joking wif u oni..." "Free entry in 2 a wkly comp to win FA C
up final tkts 21st May 2005. Text FA to 87121 to receive entry question("| __truncated__ "U dun say
so early hor... U c already then say..." ...
```

Examining this with the str() and table() functions, we see that type has now been appropriately recoded as a factor. Additionally, we see that 747 (about 15 percent) of SMS messages in our data were labeled as spam, while the others were labeled as ham:



```
> spam$type <- factor(spam$type) # convert into factor
> str(spam$type)
 Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
> table(spam$type)

 ham spam 
4827  747 
```

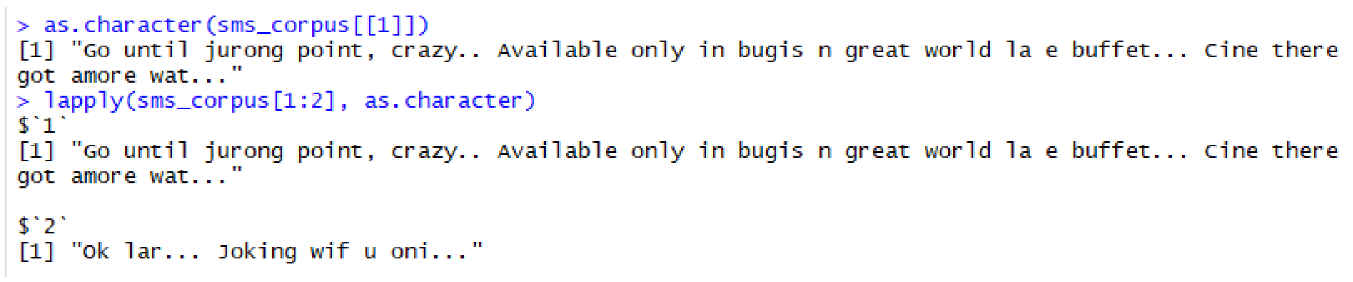
The first step in processing text data involves creating a corpus, which is a collection of text documents. The documents can be short or long, from individual news articles, pages in a book or on the web, or entire books. In our case, the corpus will be a collection of SMS messages.



```
> sms_corpus <- VCorpus(VectorSource(spam$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
```

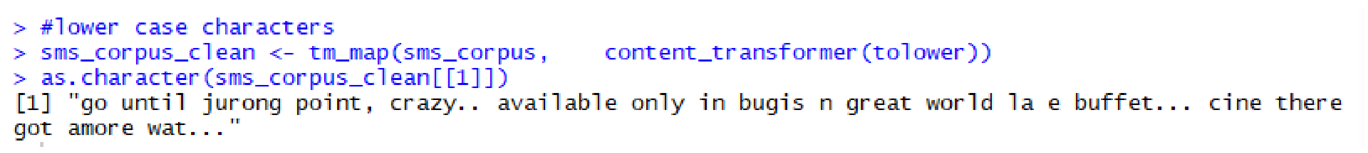
To view the actual message text, the as.character() function must be applied to the desired messages. To view one message, use the as.character() function on a single list element, noting that the double-bracket notation is required:

To view multiple documents, we'll need to use as.character() on several items in the sms_corpus object.



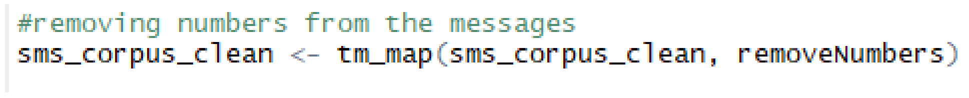
```
> as.character(sms_corpus[[1]])
[1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
got amore wat..."
> lapply(sms_corpus[1:2], as.character)
$`1`
[1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
got amore wat..."

$`2`
[1] "Ok lar... Joking wif u oni..."
```

Our first order of business will be to standardize the messages to use only lowercase characters. To this end, R provides a tolower() function that returns a lowercase version of text strings.



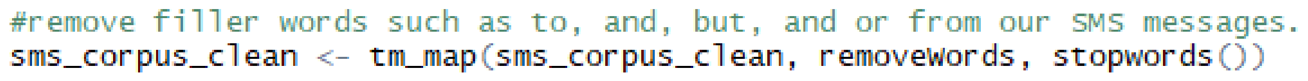
```
> #lower case characters
> sms_corpus_clean <- tm_map(sms_corpus,    content_transformer(tolower))
> as.character(sms_corpus_clean[[1]])
[1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine there
got amore wat..."
```

Removing numbers from the SMS messages.



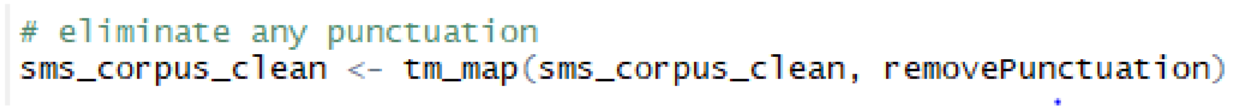
```
#removing numbers from the messages
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
```

Our next task is to remove filler words such as to, and, but, and or from our SMS messages. These terms are known as stop words and are typically removed prior to text mining. This is due to the fact that although they appear very frequently, they do not provide much useful information for machine learning.



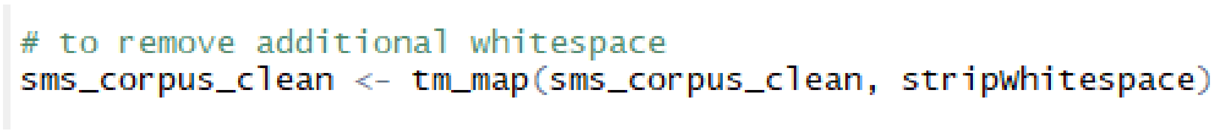
```
#remove filler words such as to, and, but, and or from our SMS messages.
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
```

Continuing with our cleanup process, we can also eliminate any punctuation from the text messages using the built-in removePunctuation() transformation:



```
# eliminate any punctuation
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

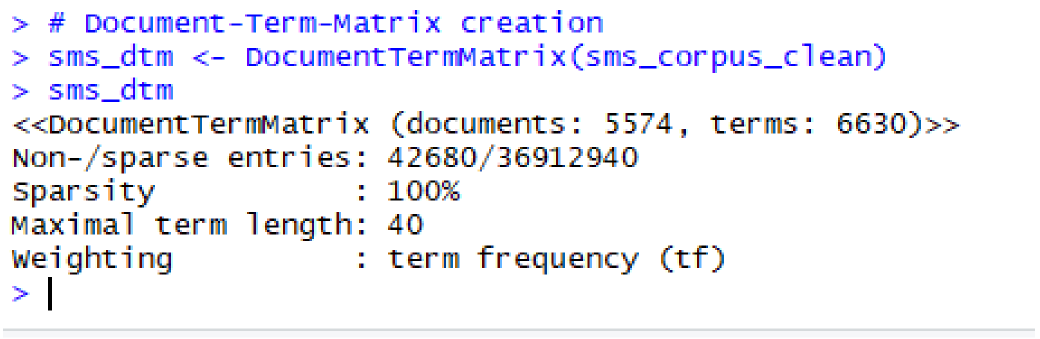
The final step in our text cleanup process is to remove additional whitespace, using the built-in stripWhitespace() transformation:



```
# to remove additional whitespace
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

Data preparation – splitting text documents into words

This will create an sms_dtm object that contains the tokenized corpus using the default settings, which apply minimal processing. The default settings are appropriate because we have already prepared the corpus manually.

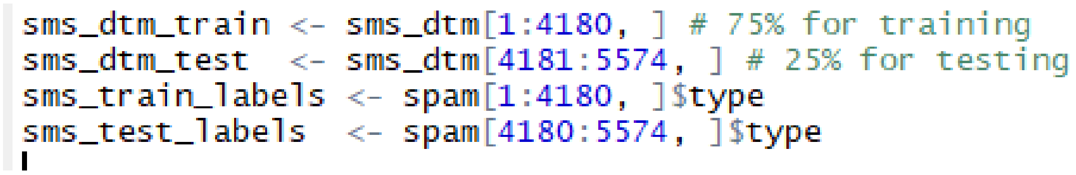


```
> # Document-Term-Matrix creation
> sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
> sms_dtm
<<DocumentTermMatrix (documents: 5574, terms: 6630)>>
Non-/sparse entries: 42680/36912940
Sparsity           : 100%
Maximal term length: 40
Weighting          : term frequency (tf)
> 
```

Data preparation – creating training and test datasets

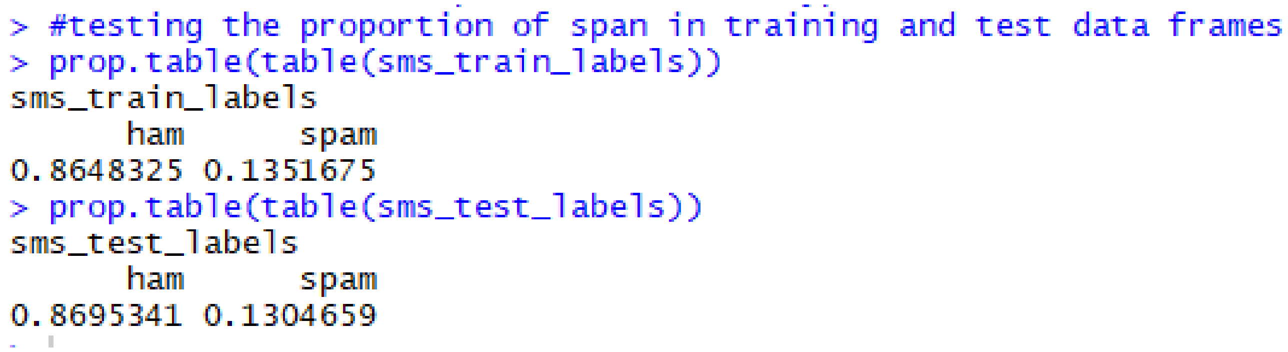
With our data prepared for analysis, we now need to split the data into training and test datasets, so that once our spam classifier is built, it can be evaluated on data it has not previously seen.

We'll divide the data into two portions: 75 percent for training and 25 percent for testing. Since the SMS messages are sorted in a random order, we can simply take the first 4,180 for training and leave the remaining 1,394 for testing.



```
sms_dtm_train <- sms_dtm[1:4180, ] # 75% for training
sms_dtm_test  <- sms_dtm[4181:5574, ] # 25% for testing
sms_train_labels <- spam[1:4180, ]$type
sms_test_labels  <- spam[4180:5574, ]$type
```

To confirm that the subsets are representative of the complete set of SMS data, let's compare the proportion of spam in the training and test data frames:

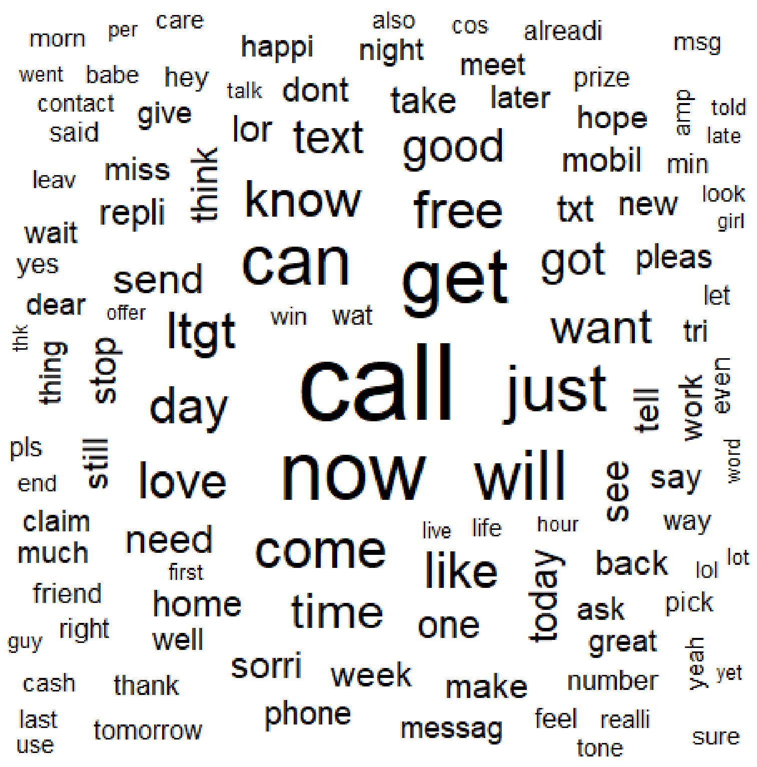


```
> #testing the proportion of span in training and test data frames
> prop.table(table(sms_train_labels))
sms_train_labels
      ham      spam 
0.8648325 0.1351675 
> prop.table(table(sms_test_labels))
sms_test_labels
      ham      spam 
0.8695341 0.1304659 
```

Both the training data and test data contain about 13 percent spam. This suggests that the spam messages were divided evenly between the two datasets.

Visualizing text data – word clouds

```
# visualizing text data
install.packages("wordcloud")
library(wordcloud)
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE) #creates worldcloud
```



random.order = FALSE, the cloud will be arranged in a nonrandom order with higher frequency words placed closer to the center.

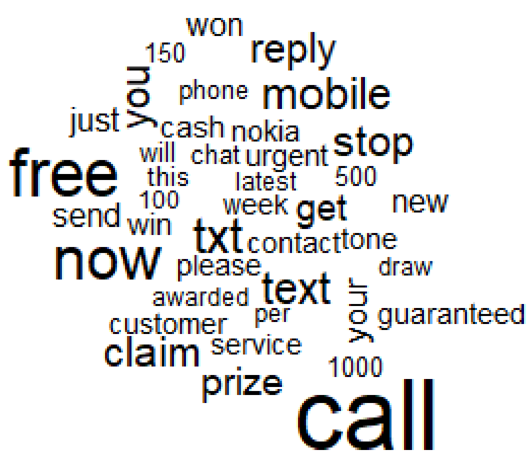
The min.freq parameter specifies the number of times a word must appear in the corpus before it will be displayed in the cloud.

Now we are going to create world cloud of spam data and ham data separately.

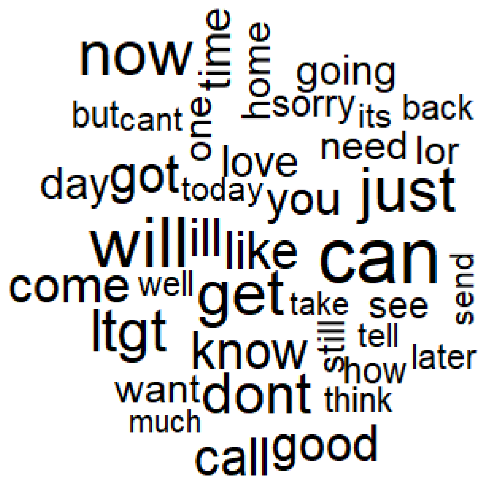
```
#creating wordcloud with ham and spam data
wordcloud(spam_1$text, max.words = 40, scale = c(3, 0.5))
wordcloud(ham_1$text, max.words = 40, scale = c(3, 0.5))
```

MSDS 680 Machine Learning
Alla Topp

Spam



Ham

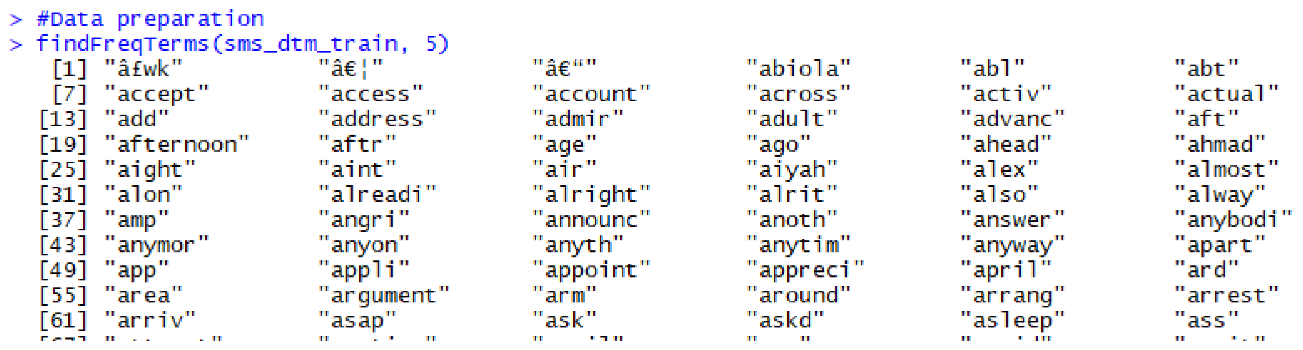


The spam cloud is on the left. Spam messages include words such as urgent, free, mobile, claim, and stop; these terms do not appear in the ham cloud at all. Instead, ham messages use words such as can, sorry, need, and time. These stark differences suggest that our Naive Bayes model will have some strong key words to differentiate between the classes.

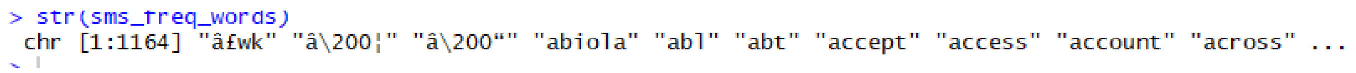
Data preparation – creating indicator features for frequent words

Now we will eliminate any words that appear in less than five SMS messages, or less than about 0.1 percent of records in the training data.

Take a document term matrix and returns a character vector containing the words appearing at least a specified number of times.

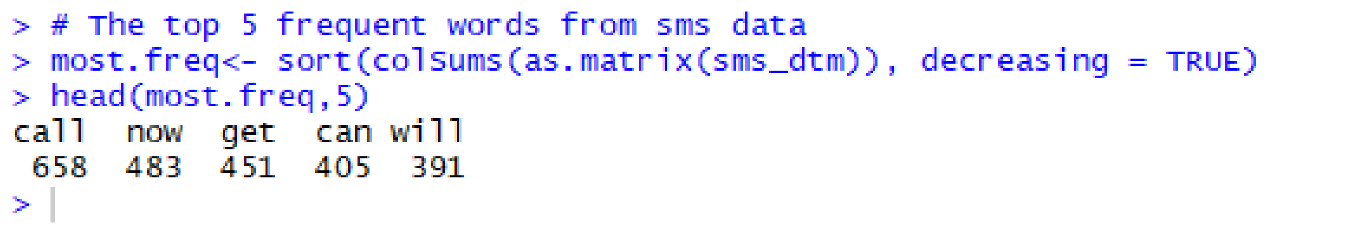


```
> #Data preparation
> findFreqTerms(sms_dtm_train, 5)
  [1] "â£wk"         "â€¦"          "â€“"          "abiola"       "abl"          "abt"
  [7] "accept"       "access"       "account"      "across"       "activ"        "actual"
 [13] "add"          "address"      "admir"        "adult"        "advanc"       "aft"
 [19] "afternoon"    "aftr"         "age"          "ago"          "ahead"        "ahmad"
 [25] "aight"        "aint"         "air"          "aiyah"        "alex"         "almost"
 [31] "alon"         "alreadi"      "alright"      "alrit"        "also"         "alway"
 [37] "amp"          "angri"        "announc"      "anoth"        "answer"       "anybodi"
 [43] "anymor"       "anyon"        "anyth"        "anytim"       "anyway"       "apart"
 [49] "app"          "appli"        "appoint"      "appreci"      "april"        "ard"
 [55] "area"         "argument"     "arm"          "around"       "arrang"       "arrest"
 [61] "arriv"        "asap"         "ask"          "askd"         "asleep"       "ass"
```

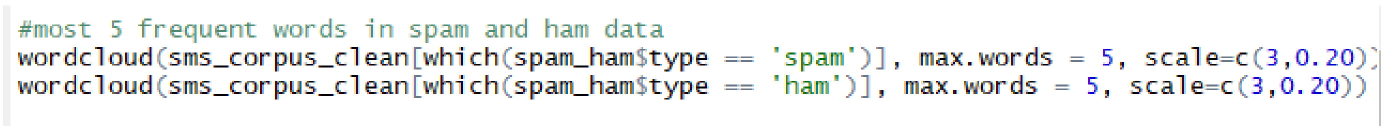


```
> str(sms_freq_words)
 chr [1:1164] "â£wk" "â\200¦" "â\200“" "abiola" "abl" "abt" "accept" "access" "account" "across" ...
```

there are 1,164 terms appearing in at least five SMS messages.



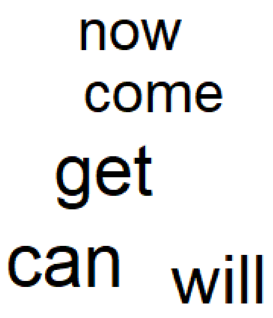
```
> # The top 5 frequent words from sms data
> most.freq<- sort(colSums(as.matrix(sms_dtm)), decreasing = TRUE)
> head(most.freq,5)
call  now  get  can will
 658  483  451  405  391
> 
```

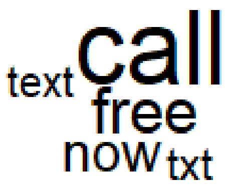


```
#most 5 frequent words in spam and ham data
wordcloud(sms_corpus_clean[which(spam_ham$type == 'spam')], max.words = 5, scale=c(3,0.20))
wordcloud(sms_corpus_clean[which(spam_ham$type == 'ham')], max.words = 5, scale=c(3,0.20))
```

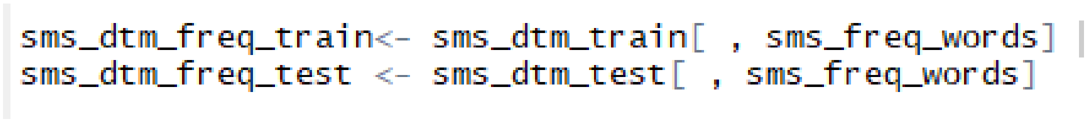
Output:

Ham **spam**





We now need to filter our DTM to include only the terms appearing in a specified vector.

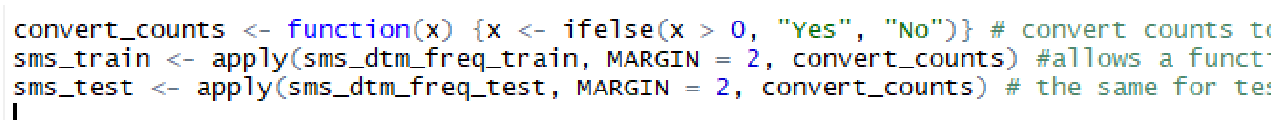


```
sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

The training and test datasets now include 1,164 features, which correspond to words appearing in at least five messages.

We need to change this to a categorical variable that simply indicates yes or no depending on whether the word appears at all.

The following defines a convert_counts() function to convert counts to Yes/No strings:

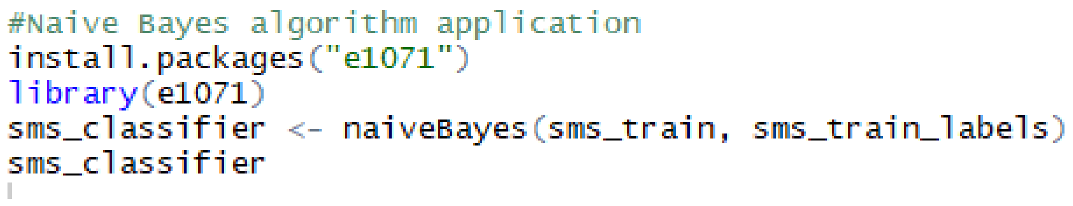


```
convert_counts <- function(x) {x <- ifelse(x > 0, "Yes", "No")} # convert counts t
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts) #allows a funct
sms_test <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts) # the same for te
```

The result will be two-character type matrixes, each with cells indicating "Yes" or "No" for whether the word represented by the column appears at any point in the message represented by the row.

Step 3 – training a model on the data

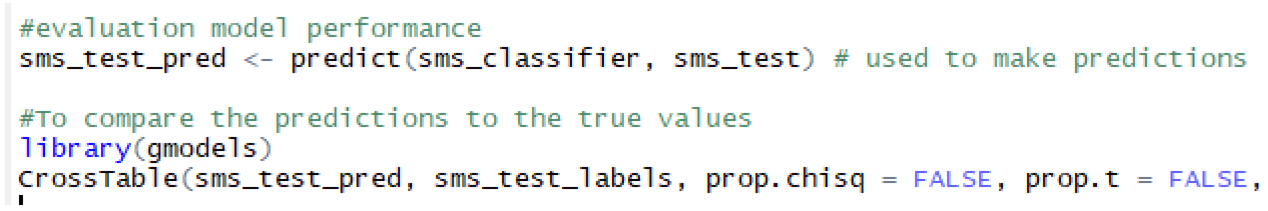
Now we can apply the Naive Bayes algorithm. The algorithm will use the presence or absence of words to estimate the probability that a given SMS message is spam.



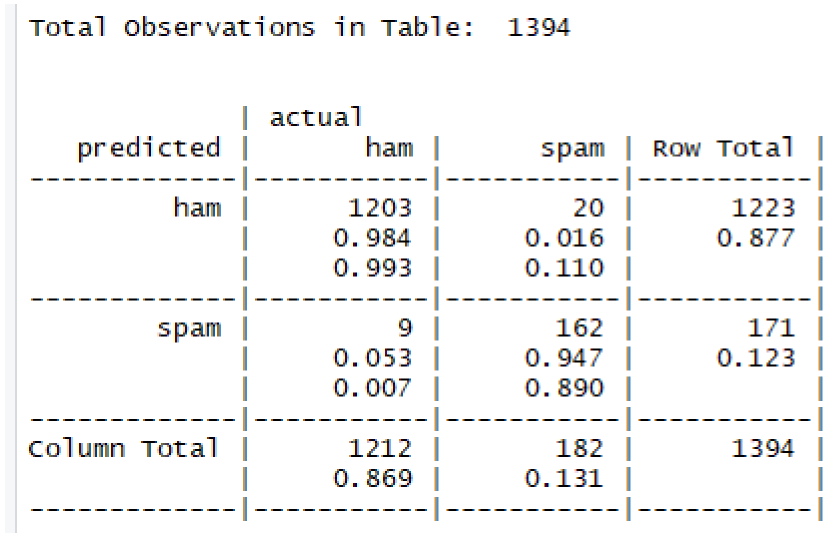
```
#Naive Bayes algorithm application
install.packages("e1071")
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
sms_classifier
```

Step 4 – evaluating model performance

To evaluate the SMS classifier, we need to test its predictions on unseen messages in the test data.



```
#evaluation model performance
sms_test_pred <- predict(sms_classifier, sms_test) # used to make predictions

#To compare the predictions to the true values
library(gmodels)
CrossTable(sms_test_pred, sms_test_labels, prop.chisq = FALSE, prop.t = FALSE,
```

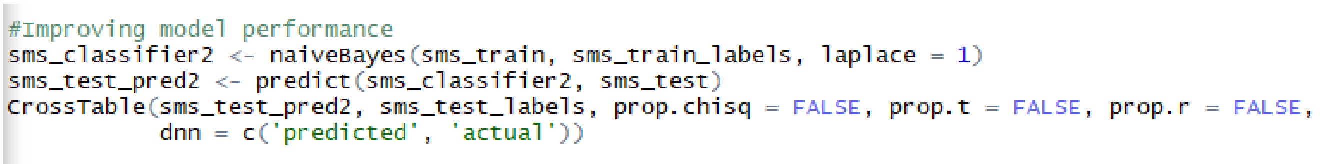


```
Total Observations in Table:  1394


             | 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 |           |
-------------|-----------|-----------|-----------|
```

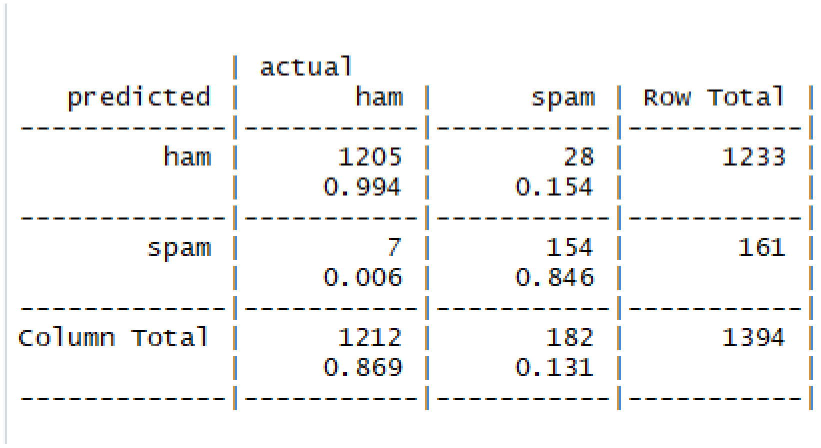
Looking at the table, we can see that a total of only 9 + 20 = 29 of the 1,394 SMS messages were incorrectly classified (2.1 percent). Among the errors were 9 out of 1,212 ham messages that were misidentified as spam, and 20 of the 182 spam messages were incorrectly labeled as ham. Since 9 messages were incorrectly classified as a spam it could create a big problem in the result, so we can try to improve the model performance. It leaves us to state that 97.9 % of the messages were correctly classified.

Step 5 – improving model performance

We didn't set a value for the Laplace estimator while training our model. This allows words that appeared in zero spam or zero ham messages to have an indisputable say in the classification process. Just because the word “ringtone” only appeared in the spam messages in the training data, it does not mean that every message with this word should be classified as spam.



```r
#Improving 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'))
```



```
             | 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 |           | 
-------------|-----------|-----------|-----------|
```

We can see a slight improvement, but only for a spam value on the table.

Adding the Laplace estimator reduced the number of false positives (ham messages erroneously classified as spam) from 9 to 7 and the number of false negatives from 20 to 28 (which increased). Although this seems like a small change, it’s substantial considering that the model’s accuracy was already quite impressive. 2.5% of all messages were incorrectly classified in this case which is an increased value in comparison with the model without Laplace.