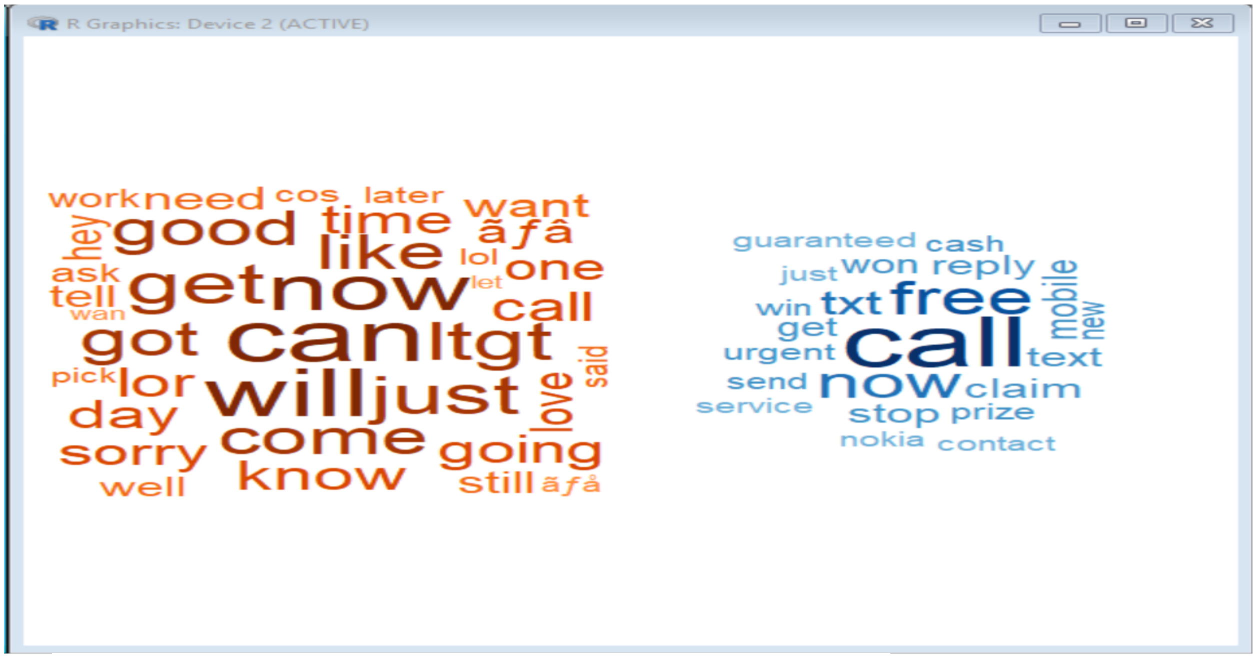
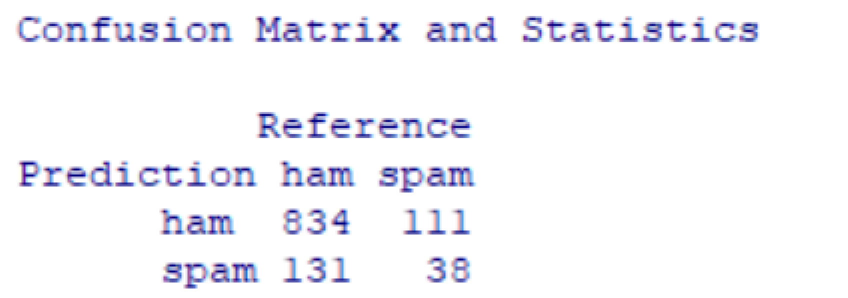
- **Question 3**

Summary:
a

1. The dataset includes 5574 observations (sms) and 2 features/ columns. 1114 are test messages and the rest are train data.
2. There is 86.5% of ham data and only 13.4% of spam which is unbalanced.
3. Document Term Matrix is (documents: 4460, terms: 1096) which indicates there are 1096 of most frequent words in the document.
4. WordCloud for 100 replications for 80% train data of all rows.



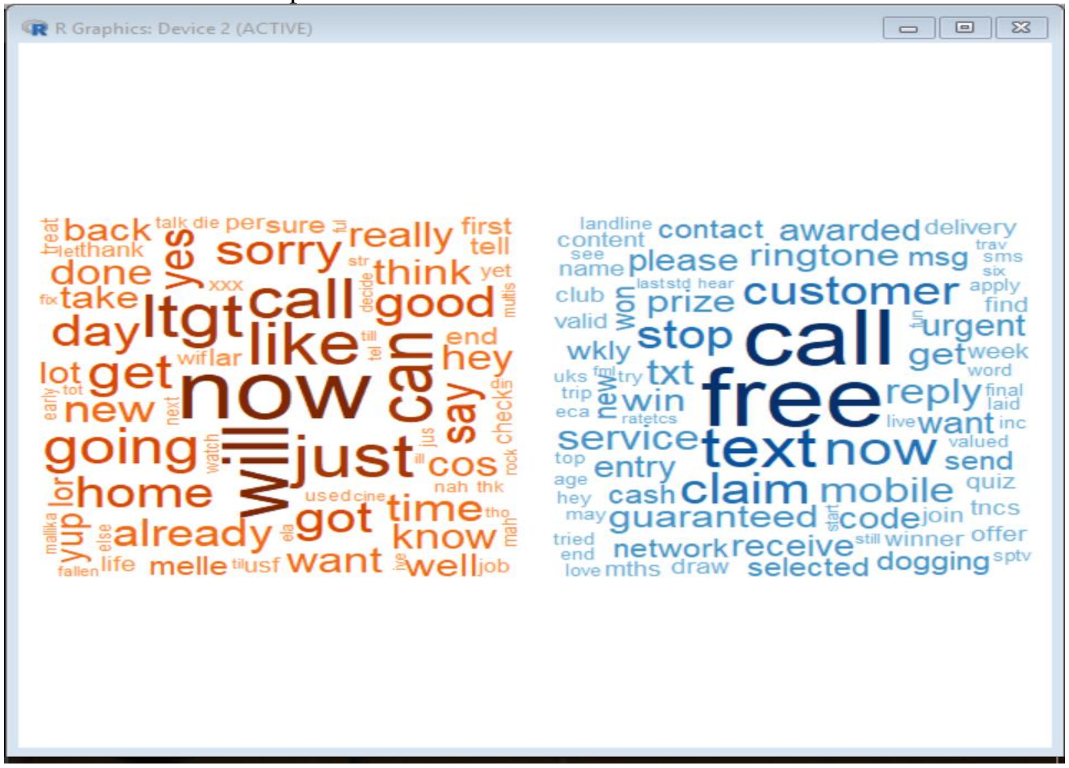


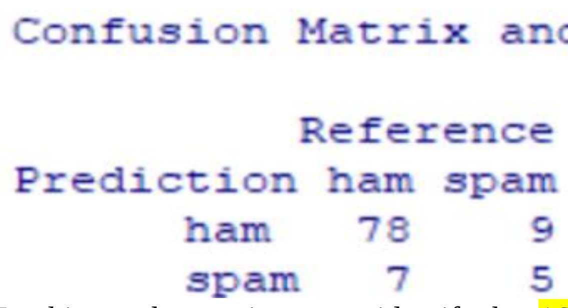
```
Confusion Matrix and Statistics

          Reference
Prediction ham spam
      ham  834  111
      spam 131   38
```

5. Looking at the matrix we can identify that 242 messages are incorrectly classified as ham or spam.
6. 872 messages are correctly classified.
7. When we use the 100 replications to build the model the accuracy is decreased.

b

1. The dataset includes 500 observations (sms) and 2 features/ columns. 100 are test data and the remaining 400 are train data.
2. There is 85.3% of ham data and only 14.2% of spam which is unbalanced
3. DocumentTermMatrix (documents: 401, terms: 115), which indicates there are 115 of most frequent words in the document.
4. WordCloud for 100 replications for 80% train data of all rows.





```
Confusion Matrix and Statistics

          Reference
Prediction ham spam
      ham   78    9
      spam   7    5
```

5. Looking at the matrix we can identify that 16 messages are incorrectly classified as ham or spam.
6. 83 messages are correctly classified among the 100-test data.
7. When we use the 100 replications to build the model the accuracy is decreased.

Observation:

	Accuracy	Precision	Recall
All Messages	0.782	0.882	0.864
500 Messages for seed as 3500	0.8383	0.8965	0.9176
500 Messages for seed as 9399	0.7414	0.848	0.858

1. Compared to the model https://rpubs.com/pparacch/237109, the accuracy is decreased because we are using 20% test take instead of 25%. The replication 100 is executed for spam and ham data are not evenly distributed which decreases the accuracy, precision and Recall.
2. We need huge data for high accuracy, precision, and recall. But in the above model as we are using the 100 replications we observe that accuracy is increased for the seed(3500).
3. When I use the seed(9399) the accuracy is 0.7414 where its decreased.
4. The main disadvantage of using the huge data is the processing time.
5. The accuracy is decreased for 100 replications compared to single replication.

Natural language processing helps in analyzing the text data and simplifies the classification of the huge data. This helps in fast searching of information, build QA model. We basically consider each word as a token and next clean the data before processing. Cleaning the data helps in removing all the unnecessary data which would create the delay in processing time.

In practical problems, a data set doesn't alway come clean. Such is the case with identifying if a message is a spam or not. This, thus, requires you to use natural language processing (NLP) as part of your work to classify data. This assignment to make you familiar with such a practical problem.

. First, do the entire steps discussed in https://rpubs.com/pparacch/237109 to do naive Bayes classification on a dataset consisting of SMS messages. The data set on SMS messages is discussed at http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/ and can be downloaded from http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/smsspamcollection.zip

Warning: the SMS dataset contains offensive words.

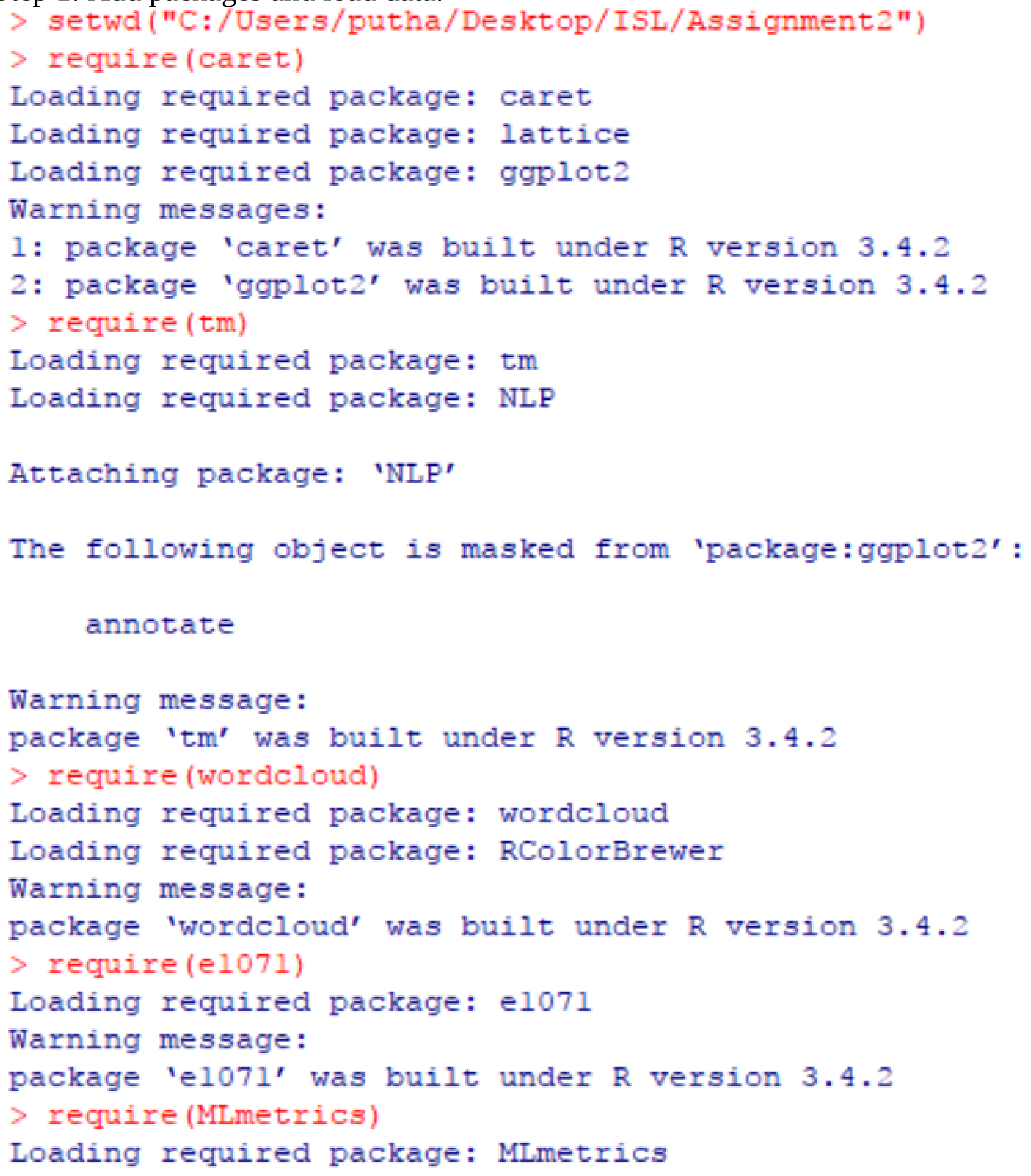
You'll note that the data set as given in the zip file (after unzip) needs to be processed to do the following:

- "\t" is to be replaced by ",".
- Double quote (") in the free text needs to be replaced by single quote (').
- Then, the sms text is to be included in "

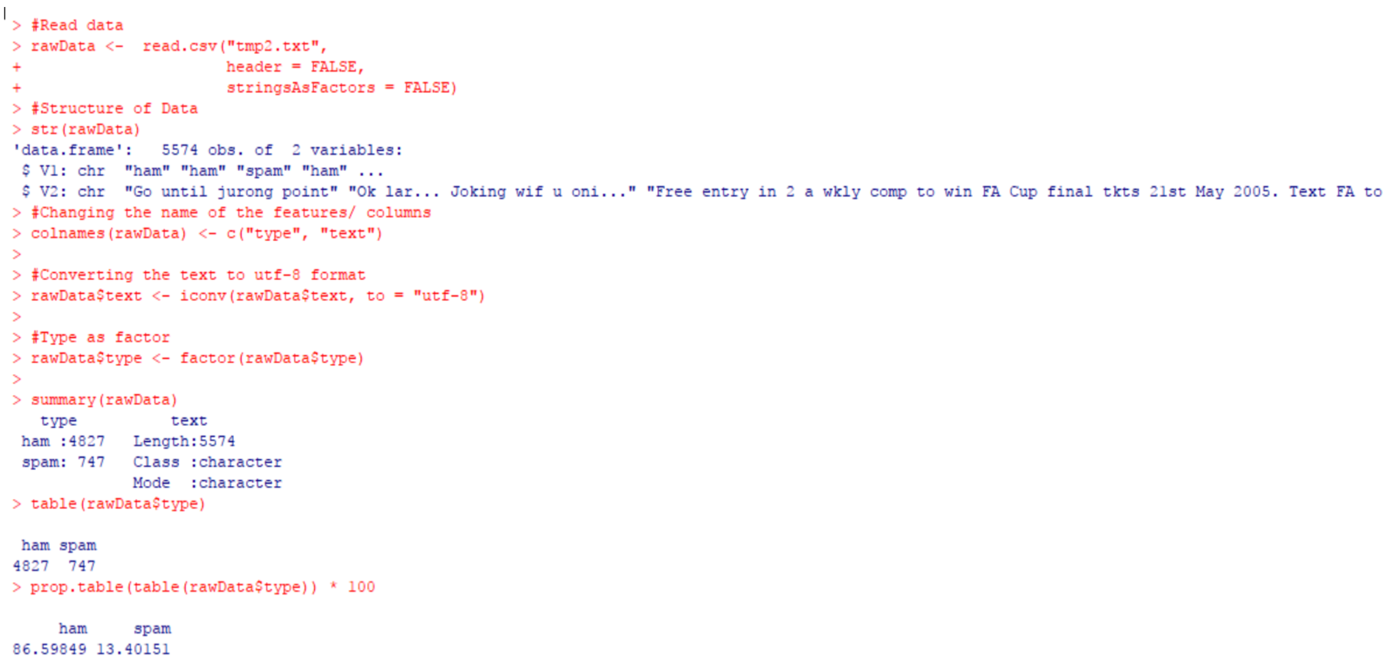
You will need to do this pre-processing yourself. (If it helps, you may use the awk code, available from Start Here, to do this pre-processing). You'll also note that you'd have to install a number of packages as listed as required at the beginning of the link. Be sure to include the wordcloud figure with your submission. Report also if the answers you got are different from the ones available at the link and the possible reason for disparity.

In the below model we are using naïve Bayes to build the model.

Step 1: Add packages and load data.

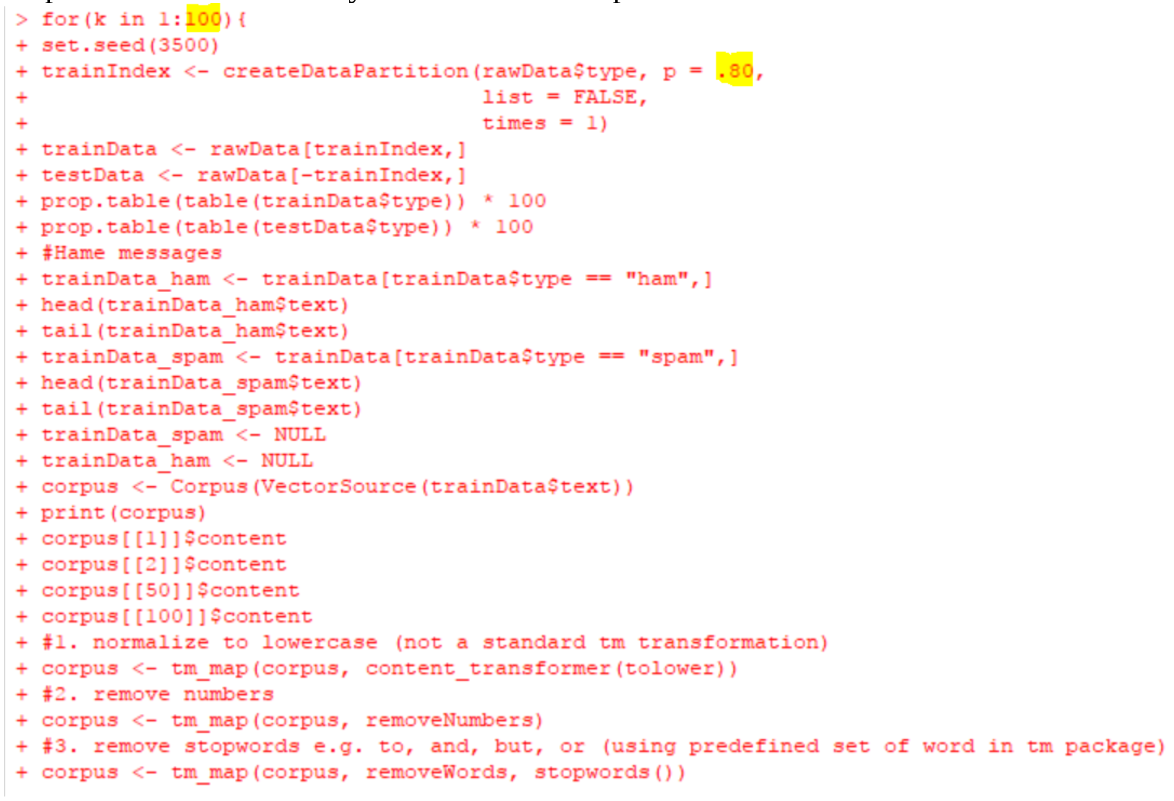


```
> setwd("C:/Users/putha/Desktop/ISL/Assignment2")
> require(caret)
Loading required package: caret
Loading required package: lattice
Loading required package: ggplot2
Warning messages:
1: package 'caret' was built under R version 3.4.2
2: package 'ggplot2' was built under R version 3.4.2
> require(tm)
Loading required package: tm
Loading required package: NLP

Attaching package: 'NLP'

The following object is masked from 'package:ggplot2':

    annotate

Warning message:
package 'tm' was built under R version 3.4.2
> require(wordcloud)
Loading required package: wordcloud
Loading required package: RColorBrewer
Warning message:
package 'wordcloud' was built under R version 3.4.2
> require(e1071)
Loading required package: e1071
Warning message:
package 'e1071' was built under R version 3.4.2
> require(MLmetrics)
Loading required package: MLmetrics
```



```
> #Read data
> rawData <-  read.csv("tmp2.txt",
+                      header = FALSE,
+                      stringsAsFactors = FALSE)
> #Structure of Data
> str(rawData)
'data.frame':   5574 obs. of  2 variables:
 $ V1: chr  "ham" "ham" "spam" "ham" ...
 $ V2: chr  "Go until jurong point" "Ok lar... Joking wif u oni..." "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to
> #Changing the name of the features/ columns
> colnames(rawData) <- c("type", "text")
>
> #Converting the text to utf-8 format
> rawData$text <- iconv(rawData$text, to = "utf-8")
>
> #Type as factor
> rawData$type <- factor(rawData$type)
>
> summary(rawData)
   type          text
 ham :4827   Length:5574
 spam: 747   Class :character
             Mode  :character
> table(rawData$type)

 ham spam
4827  747
> prop.table(table(rawData$type)) * 100

     ham     spam
86.59849 13.40151
```

Step 2: Build the Naïve Bayes model for 100 replications for 80% train data.

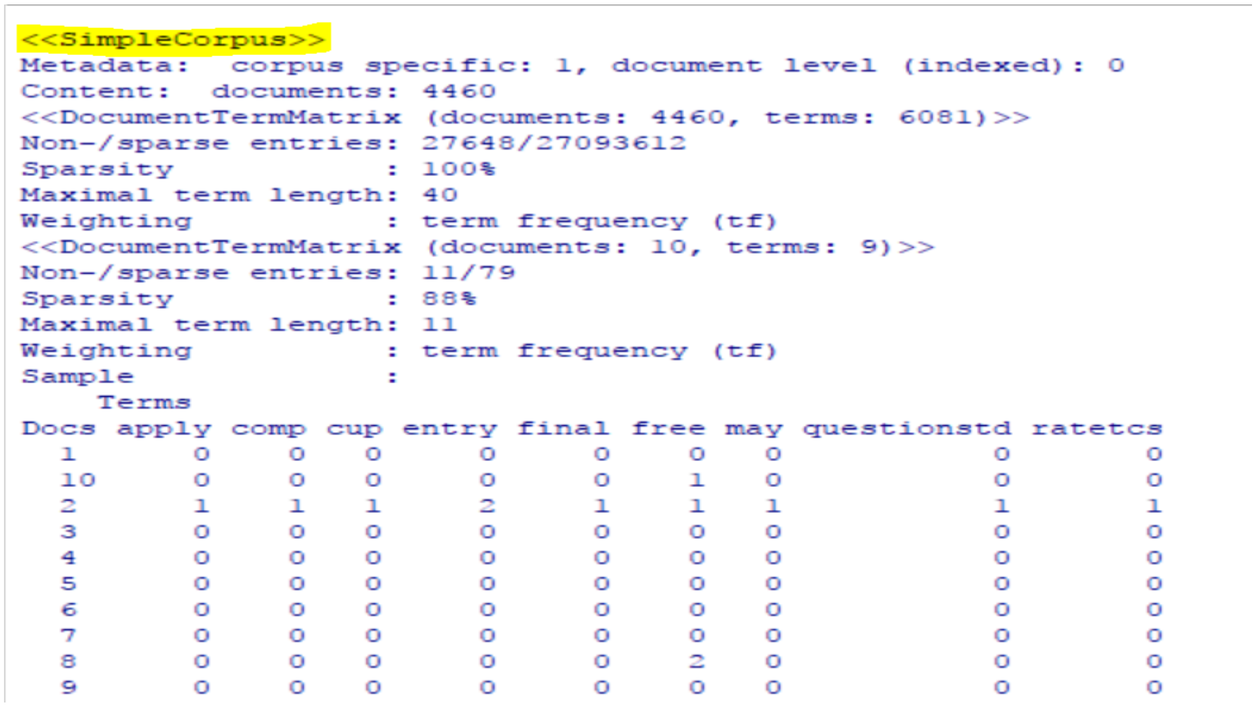


```
> for(k in 1:100){
+ set.seed(3500)
+ trainIndex <- createDataPartition(rawData$type, p = .80,
+                                   list = FALSE,
+                                   times = 1)
+ trainData <- rawData[trainIndex,]
+ testData <- rawData[-trainIndex,]
+ prop.table(table(trainData$type)) * 100
+ prop.table(table(testData$type)) * 100
+ #Hame messages
+ trainData_ham <- trainData[trainData$type == "ham",]
+ head(trainData_ham$text)
+ tail(trainData_ham$text)
+ trainData_spam <- trainData[trainData$type == "spam",]
+ head(trainData_spam$text)
+ tail(trainData_spam$text)
+ trainData_spam <- NULL
+ trainData_ham <- NULL
+ corpus <- Corpus(VectorSource(trainData$text))
+ print(corpus)
+ corpus[[1]]$content
+ corpus[[2]]$content
+ corpus[[50]]$content
+ corpus[[100]]$content
+ #1. normalize to lowercase (not a standard tm transformation)
+ corpus <- tm_map(corpus, content_transformer(tolower))
+ #2. remove numbers
+ corpus <- tm_map(corpus, removeNumbers)
+ #3. remove stopwords e.g. to, and, but, or (using predefined set of word in tm package)
+ corpus <- tm_map(corpus, removeWords, stopwords())
```

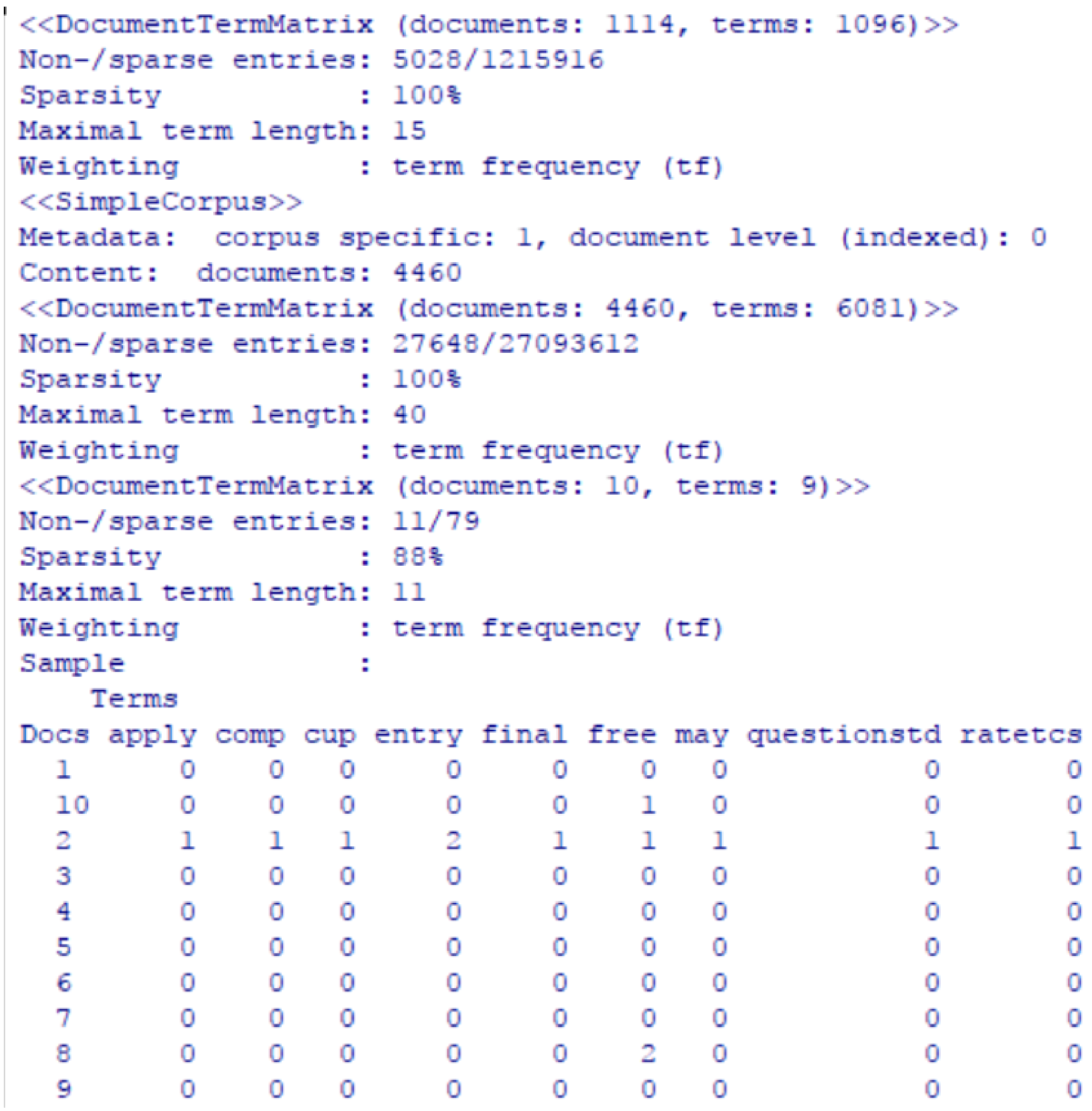
Step 3: Test the model with 20% test data and check the accuracy.

```
+ convert_counts <- function(x){
+   x <- ifelse(x > 0, 1, 0)
+   x <- factor(x, levels = c(0,1), labels = c("No", "Yes"))
+   return (x)
+ }
+ sms_dtm_train <- apply(sms_dtm_train, MARGIN = 2, convert_counts)
+ head(sms_dtm_train[,1:5])
+ sms_classifier <- naiveBayes(sms_dtm_train, trainData$type)
+ sms_classifier[[2]][1:5]
+ corpus <- Corpus(VectorSource(testData$text))
+ #1. normalize to lowercase (not a standard tm transformation)
+ corpus <- tm_map(corpus, content_transformer(tolower))
+ #2. remove numbers
+ corpus <- tm_map(corpus, removeNumbers)
+ #3. remove stopwords e.g. to, and, but, or (using predefined set of word in tm package)
+ corpus <- tm_map(corpus, removeWords, stopwords())
+ #4. remove punctuation
+ corpus <- tm_map(corpus, removePunctuation)
+ #5. normalize whitespaces
+ corpus <- tm_map(corpus, stripWhitespace)
+
+ sms_dtm_test <- DocumentTermMatrix(corpus, list(global = c(2, Inf), dictionary = sms_features))
+ print(sms_dtm_test)
+ sms_dtm_test <- apply(sms_dtm_test, MARGIN = 2, convert_counts)
+ sms_dtm_test[1:10, 5:12]
+ sms_test_pred <- predict(sms_classifier, sms_dtm_test)
+ tablin<-table(testData$type, sms_test_pred)
+ cm=confusionMatrix(sms_test_pred, testData$type)
+ Accuracy[k]=cm$overall[1]
+ Precision[k]=cm$byClass[5]
+ Recall[k]=cm$byClass[6]
+ }
```

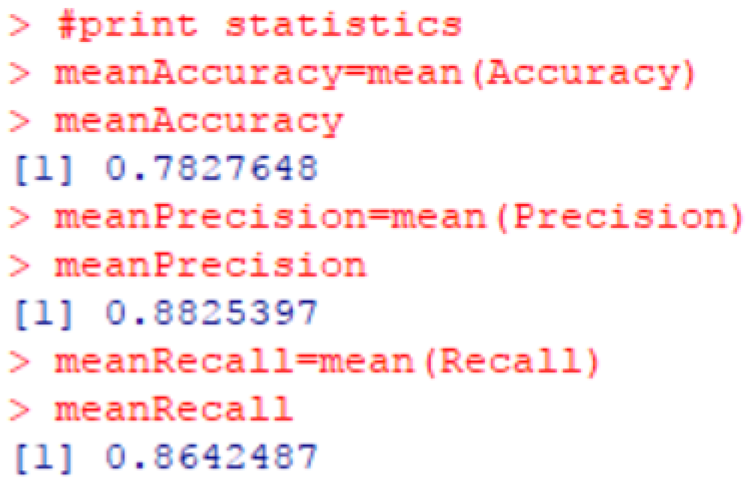
Step 4: Check the accuracy, precision and recall for build model.



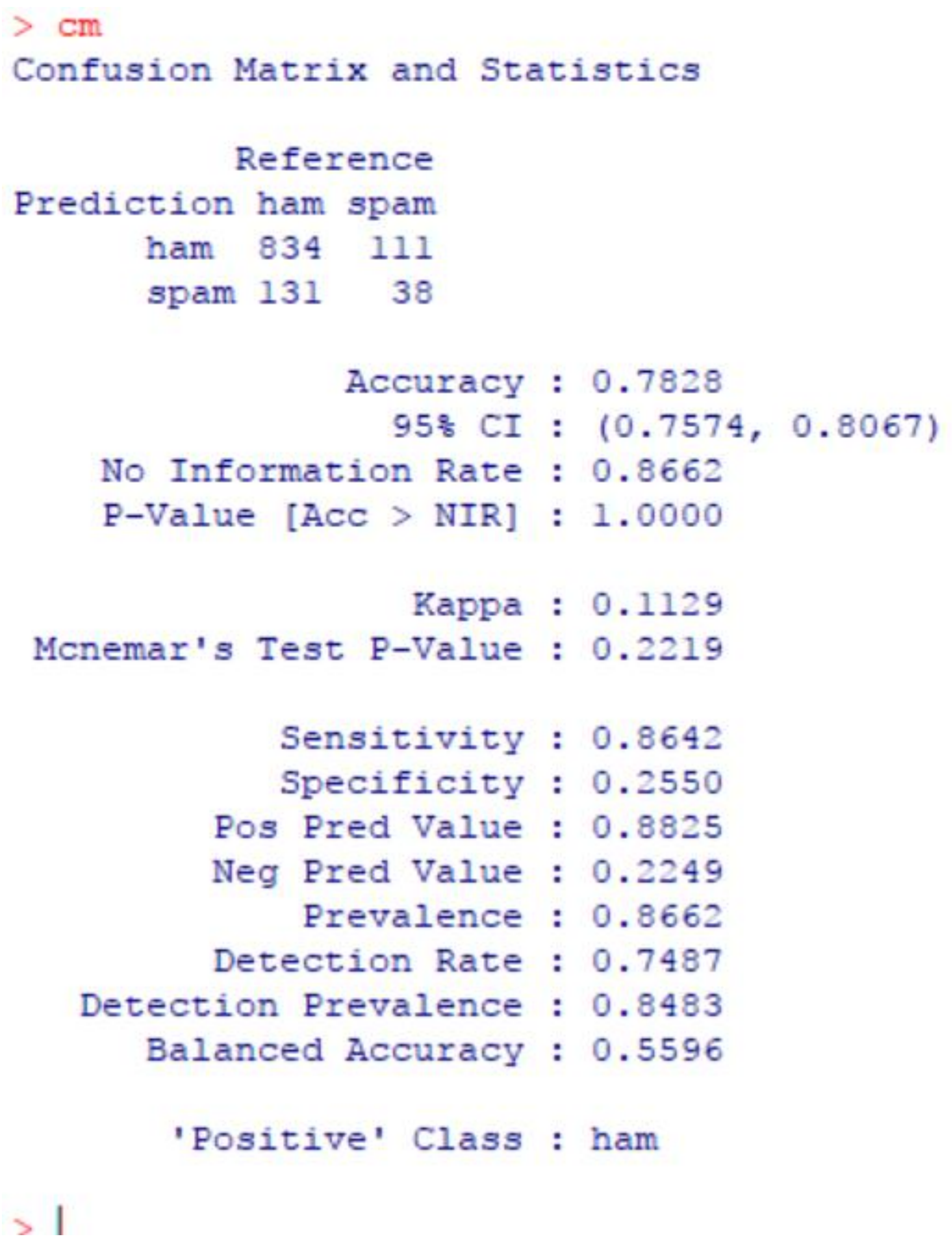
```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 4460
<<DocumentTermMatrix (documents: 4460, terms: 6081)>>
Non-/sparse entries: 27648/27093612
Sparsity           : 100%
Maximal term length: 40
Weighting          : term frequency (tf)
<<DocumentTermMatrix (documents: 10, terms: 9)>>
Non-/sparse entries: 11/79
Sparsity           : 88%
Maximal term length: 11
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs apply comp cup entry final free may questionstd ratetcs
  1      0    0   0     0     0    0   0           0       0
  10     0    0   0     0     0    1   0           0       0
  2      1    1   1     2     1    1   1           1       1
  3      0    0   0     0     0    0   0           0       0
  4      0    0   0     0     0    0   0           0       0
  5      0    0   0     0     0    0   0           0       0
  6      0    0   0     0     0    0   0           0       0
  7      0    0   0     0     0    0   0           0       0
  8      0    0   0     0     0    2   0           0       0
  9      0    0   0     0     0    0   0           0       0
```



```
<<DocumentTermMatrix (documents: 1114, terms: 1096)>>
Non-/sparse entries: 5028/1215916
Sparsity           : 100%
Maximal term length: 15
Weighting          : term frequency (tf)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 4460
<<DocumentTermMatrix (documents: 4460, terms: 6081)>>
Non-/sparse entries: 27648/27093612
Sparsity           : 100%
Maximal term length: 40
Weighting          : term frequency (tf)
<<DocumentTermMatrix (documents: 10, terms: 9)>>
Non-/sparse entries: 11/79
Sparsity           : 88%
Maximal term length: 11
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs apply comp cup entry final free may questionstd ratetcs
  1      0    0   0     0     0    0   0           0       0
  10     0    0   0     0     0    1   0           0       0
  2      1    1   1     2     1    1   1           1       1
  3      0    0   0     0     0    0   0           0       0
  4      0    0   0     0     0    0   0           0       0
  5      0    0   0     0     0    0   0           0       0
  6      0    0   0     0     0    0   0           0       0
  7      0    0   0     0     0    0   0           0       0
  8      0    0   0     0     0    2   0           0       0
  9      0    0   0     0     0    0   0           0       0
```



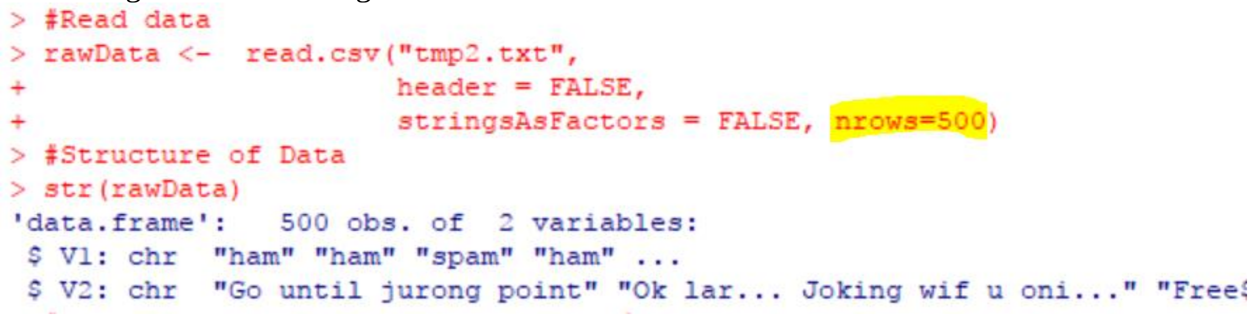
```
> #print statistics
> meanAccuracy=mean(Accuracy)
> meanAccuracy
[1] 0.7827648
> meanPrecision=mean(Precision)
> meanPrecision
[1] 0.8825397
> meanRecall=mean(Recall)
> meanRecall
[1] 0.8642487
```



```
> cm
Confusion Matrix and Statistics

          Reference
Prediction ham spam
      ham  834  111
      spam 131   38

               Accuracy : 0.7828
                 95% CI : (0.7574, 0.8067)
    No Information Rate : 0.8662
    P-Value [Acc > NIR] : 1.0000

                  Kappa : 0.1129
 Mcnemar's Test P-Value : 0.2219

            Sensitivity : 0.8642
            Specificity : 0.2550
         Pos Pred Value : 0.8825
         Neg Pred Value : 0.2249
             Prevalence : 0.8662
         Detection Rate : 0.7487
   Detection Prevalence : 0.8483
      Balanced Accuracy : 0.5596

       'Positive' Class : ham

> |
```

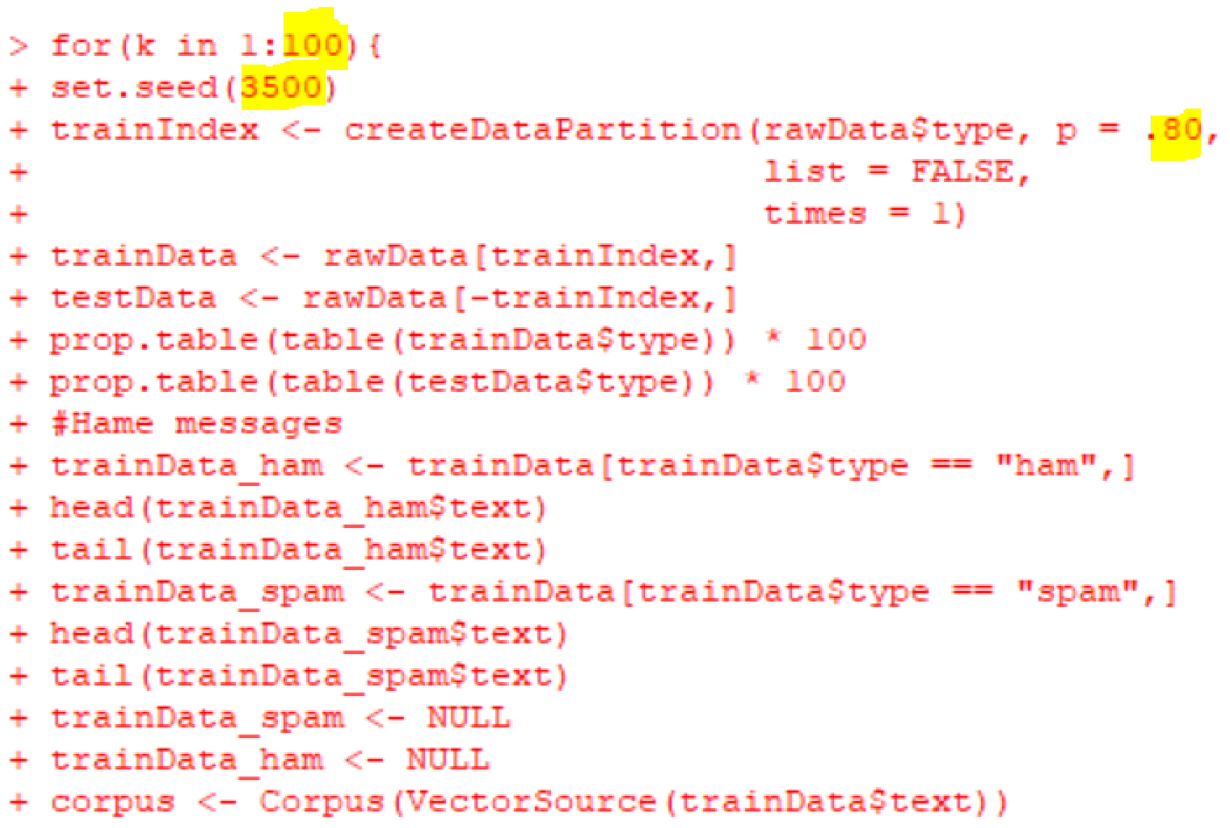
b) Now you are to consider a subset of 500 SMS messages from the original dataset using your last 4 digits of your student ID as the seed (set.seed(nnnn), where nnnn is the last 4 digits of your student ID) through sampling, using 'sample'. On this 500 SMS message in your collection, you'll then do 80/20-rule for training set/test data set split from YOUR data set. And repeat the above work performed in a) above. Report on how the results for your set varies from the original dataset (be sure to include the wordcloud figure for your dataset alongside the original data set for visual comparison).

Step 1: Reading 500 SMS message data.



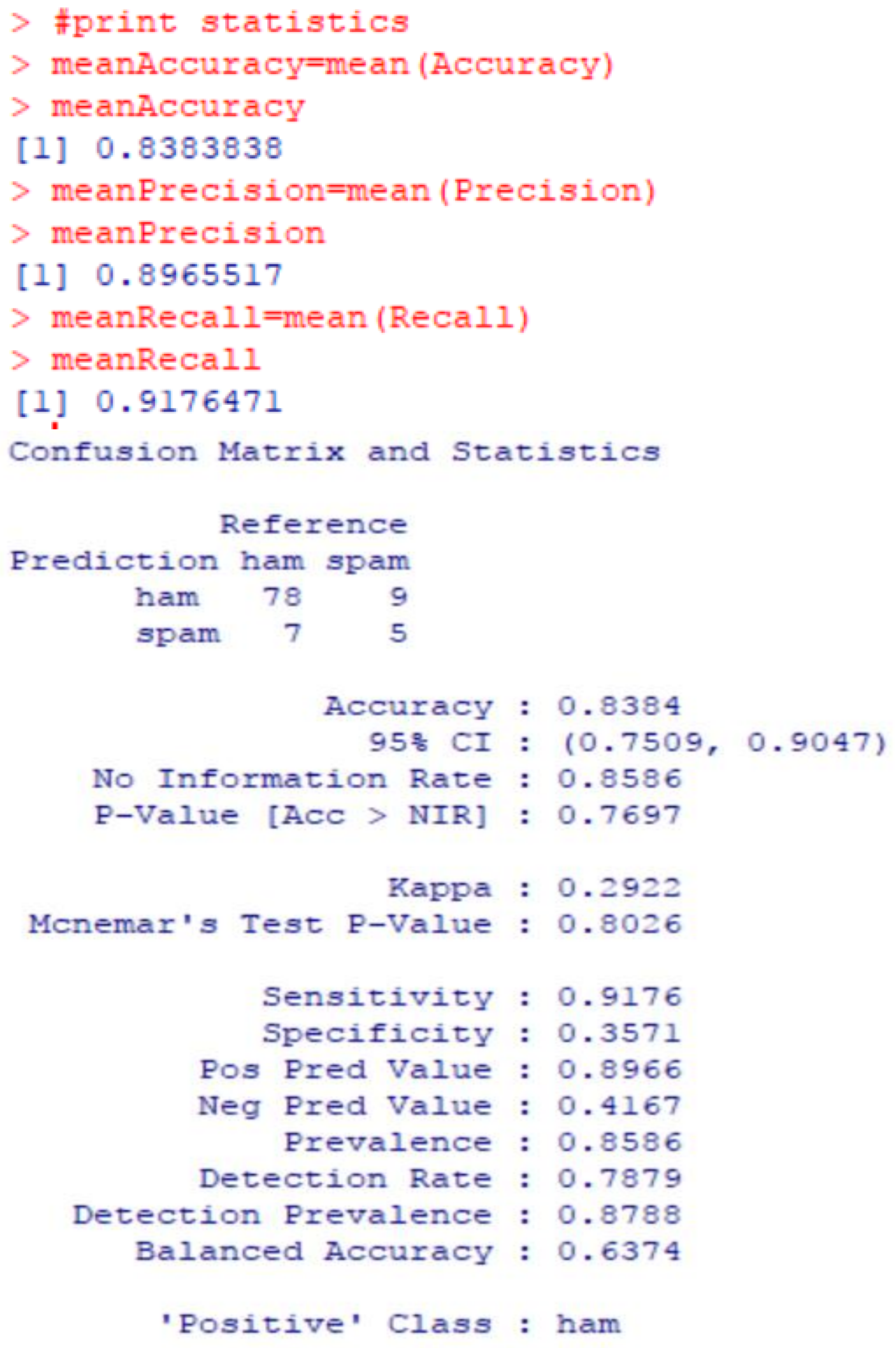
```
> #Read data
> rawData <-  read.csv("tmp2.txt",
+                     header = FALSE,
+                     stringsAsFactors = FALSE, nrows=500)
> #Structure of Data
> str(rawData)
'data.frame':   500 obs. of  2 variables:
 $ V1: chr  "ham" "ham" "spam" "ham" ...
 $ V2: chr  "Go until jurong point" "Ok lar... Joking wif u oni..." "Free
```

Step 2: Cleaning train data and apply the naïve Bayes model for 100 replications.



```
> for(k in 1:100){
+ set.seed(3500)
+ trainIndex <- createDataPartition(rawData$type, p = .80,
+                                   list = FALSE,
+                                   times = 1)
+ trainData <- rawData[trainIndex,]
+ testData <- rawData[-trainIndex,]
+ prop.table(table(trainData$type)) * 100
+ prop.table(table(testData$type)) * 100
+ #Hame messages
+ trainData_ham <- trainData[trainData$type == "ham",]
+ head(trainData_ham$text)
+ tail(trainData_ham$text)
+ trainData_spam <- trainData[trainData$type == "spam",]
+ head(trainData_spam$text)
+ tail(trainData_spam$text)
+ trainData_spam <- NULL
+ trainData_ham <- NULL
+ corpus <- Corpus(VectorSource(trainData$text))
```

Step 3: Resulted Output



```
> #print statistics
> meanAccuracy=mean(Accuracy)
> meanAccuracy
[1] 0.8383838
> meanPrecision=mean(Precision)
> meanPrecision
[1] 0.8965517
> meanRecall=mean(Recall)
> meanRecall
[1] 0.9176471
Confusion Matrix and Statistics

          Reference
Prediction ham spam
      ham   78    9
      spam   7    5

               Accuracy : 0.8384
                 95% CI : (0.7509, 0.9047)
    No Information Rate : 0.8586
    P-Value [Acc > NIR] : 0.7697

                  Kappa : 0.2922
 Mcnemar's Test P-Value : 0.8026

            Sensitivity : 0.9176
            Specificity : 0.3571
         Pos Pred Value : 0.8966
         Neg Pred Value : 0.4167
             Prevalence : 0.8586
         Detection Rate : 0.7879
   Detection Prevalence : 0.8788
      Balanced Accuracy : 0.6374

       'Positive' Class : ham
```