Question 3

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 is to make you familiar with such a practical problem.

a. 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 here process_sms.awk , 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.

(ADDED REQUIREMENTS:)
For the part 'Evaluate the Model', use either 80/20-rule with randomization for 100 replications, or k-fold cross-validation.

```r
# Importing the libraries

library(caTools)
library(ggplot2)
library(MASS)
library(tm)
library(wordcloud)
library(caret)
library(e1071)
library(MLmetrics)
library(stringr)

# Importing the dataset

input_data=input_data<-read.csv("C:/Users/bvkka/Desktop/ISL-Deep Medhi/SMSSpamCollection.csv",
                                header = FALSE,
```

```r
                                stringsAsFactors = FALSE)

str(input_data)

#Changing the name of the features/ columns

colnames(input_data) <- c("type", "text")

#Converting the text to utf-8 format

input_data$text <- iconv(input_data$text, to = "utf-8")

#Type as factor

input_data$type <- factor(input_data$type)

summary(input_data)

table(input_data$type)
prop.table(table(input_data$type)) * 100

set.seed(123)

# Create a training set containing 80% of the data (with stratified sampling)

trainIndex <- createDataPartition(input_data$type, p = .8,
                                  list = FALSE,
                                  times = 1)
trainData <- input_data[trainIndex,]
testData <- input_data[-trainIndex,]

# proportion in train dataset

prop.table(table(trainData$type)) * 100

##      ham     spam
## 86.58591 13.41409

# proportion in test dataset
```

```r
prop.table(table(testData$type)) * 100

# Ham messages

trainData_ham <- trainData[trainData$type == "ham",]
head(trainData_ham$text)
tail(trainData_ham$text)

# spam messages

trainData_spam <- trainData[trainData$type == "spam",]
head(trainData_spam$text)

# Removing the trainData_ham and trainData_spam

trainData_spam <- NULL
trainData_ham <- NULL

# create the corpus

corpus <- Corpus(VectorSource(trainData$text))

# basic info about the corpus

print(corpus)

#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
```

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

#5. normalize whitespaces

corpus <- tm_map(corpus, stripWhitespace)

# Visualizing the data

pal1 <- brewer.pal(9,"YlGn")
pal1 <- pal1[-(1:4)]

pal2 <- brewer.pal(9,"Reds")
pal2 <- pal2[-(1:4)]

#min.freq initial settings -> around 10% of the number of docs in the corpus (40 times)

par(mfrow = c(1,2))
wordcloud(corpus[trainData$type == "ham"], min.freq = 40, random.order = FALSE, colors = pal1)
wordcloud(corpus[trainData$type == "spam"], min.freq = 40, random.order = FALSE, colors = pal2)

# Creation of the DTM considering terms with at least 2 chars

sms_dtm <- DocumentTermMatrix(corpus, control = list(global = c(2, Inf)))

# Basic information about the sparse matrix

print(sms_dtm)

inspect(sms_dtm[1:10, 5:13])

sms_features <- findFreqTerms(sms_dtm, 5) #find words that appears at least 5 times
summary(sms_features)

head(sms_features)

sms_dtm_train <- DocumentTermMatrix(corpus, list(global = c(2, Inf), dictionary = sms_features))
print(sms_dtm_train)
```

```r
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])

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]

#Evaluating the Model

#sms_classifier <- naiveBayes(sms_dtm_train, trainData$type)

sms_classifier <- train(sms_dtm_train, trainData$type, method = "nb", trControl = trainControl(method = "cv",
number = 10)) #k fold cross validation

sms_classifier[[2]][1:5]

sms_test_pred <- predict(sms_classifier$finalModel, sms_dtm_test)$class

#table actual (row) vs. predicted (col): confusion matrix
```

```
Accuracy(sms_test_pred, testData$type)

F1_Score(sms_test_pred, testData$type)
```

```
##**Accuracy and Score***###
```

```
> Accuracy(sms_test_pred, testData$type)
[1] 0.7836625
>
> F1_Score(sms_test_pred, testData$type)
[1] 0.8771035
>
```

```
Sample          :
    Terms
Docs cine crazy got great joking jurong point wat world
  1    1     1   1     1      0      1     1   1     1
  10   0     0   0     0      0      0     0   0     0
  2    0     0   0     0      1      0     0   0     0
  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      0     0   0     0
  9    0     0   0     0      0      0     0   0     0
<<DocumentTermMatrix (documents: 4458, terms: 1303)>>
Non-/sparse entries: 25907/5782867
Sparsity          : 100%
Maximal term length: 19
Weighting         : term frequency (tf)
```

#####*******TEXT SUMMARIZATION*******######

1. We Observe that the world cloud figure is different from the URL source. The first one I attached is after 10 iterations and the second image I attached is after 100 iterations. We can see the difference how the words which are most frequently seen under spam and ham groups.
2. Note that the Accuracy and F1_score obtained is different from the URL source. I took the mean of all 100 accuracy values and stored it under acc variable. Similarly, I took mean of all 100 F1_Score values and stored it under F1_Score variable.

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).

(ADDED REQUIREMENTS:)

For the part 'Evaluate the Model', use either 80/20-rule with randomization for 100 replications, or k-fold cross-validation.

(ADDED NOTE-2):

First include a text summarizing your KEY observations and any issues (this can be a page or so in single-space). Following this, include the output from R. From the text, you may include some pointers to the R output where your observation comes from.

```r
# Importing the libraries

library(caTools)
library(ggplot2)
library(MASS)
library(tm)
library(wordcloud)
library(caret)
library(e1071)
library(MLmetrics)
library(stringr)



# Importing the libraries

library(caTools)
library(ggplot2)
library(MASS)
library(tm)
library(wordcloud)
library(caret)
library(e1071)
library(MLmetrics)
library(stringr)

# Importing the dataset

input_data=input_data<-read.csv("C:/Users/bvkka/Desktop/ISL-Deep Medhi/SMSSpamCollection.csv",
                                header = FALSE,
                                stringsAsFactors = FALSE)

str(input_data)

#Changing the name of the features/ columns

colnames(input_data) <- c("type", "text")

#Converting the text to utf-8 format
```

```r
input_data$text <- iconv(input_data$text, to = "utf-8")

#Type as factor

input_data$type <- factor(input_data$type)

# Cleaning the raw data by removing the na's

input_data = na.omit(input_data)

# Taking only 500 messages from the main dataset

data = input_data[sample(nrow(input_data), 500), ]

summary(data)

table(data$type)
prop.table(table(data$type)) * 100

set.seed(0698) ###****MY STUDENT ID LAST FOUR DIGITS****###

# Create a training set containing 80% of the data (with stratified sampling)

trainIndex <- createDataPartition(data$type, p = .8,
                                  list = FALSE,
                                  times = 1)
trainData <- data[trainIndex,]
testData <- data[-trainIndex,]

# proportion in train dataset

prop.table(table(trainData$type)) * 100

##      ham     spam
## 86.58591 13.41409

# proportion in test dataset
prop.table(table(testData$type)) * 100
```

```r
# Ham messages

trainData_ham <- trainData[trainData$type == "ham",]
head(trainData_ham$text)
tail(trainData_ham$text)

# spam messages

trainData_spam <- trainData[trainData$type == "spam",]
head(trainData_spam$text)

# Removing the trainData_ham and trainData_spam

trainData_spam <- NULL
trainData_ham <- NULL

# create the corpus

corpus <- Corpus(VectorSource(trainData$text))

# basic info about the corpus

print(corpus)

#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)

# Visualizing the data

pal1 <- brewer.pal(9,"YlGn")
pal1 <- pal1[-(1:4)]

pal2 <- brewer.pal(9,"Reds")
pal2 <- pal2[-(1:4)]

#min.freq initial settings -> around 10% of the number of docs in the corpus (40 times)

par(mfrow = c(1,2))
wordcloud(corpus[trainData$type == "ham"], min.freq = 40, random.order = FALSE, colors = pal1)
wordcloud(corpus[trainData$type == "spam"], min.freq = 40, random.order = FALSE, colors = pal2)

# Creation of the DTM considering terms with at least 2 chars

sms_dtm <- DocumentTermMatrix(corpus, control = list(global = c(2, Inf)))

# Basic information about the sparse matrix

print(sms_dtm)

inspect(sms_dtm[1:10, 5:13])

sms_features <- findFreqTerms(sms_dtm, 5) #find words that appears at least 5 times
summary(sms_features)

head(sms_features)

sms_dtm_train <- DocumentTermMatrix(corpus, list(global = c(2, Inf), dictionary = sms_features))
print(sms_dtm_train)

convert_counts <- function(x){
```

```r
  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])

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]

#Evaluating the Model

#sms_classifier <- naiveBayes(sms_dtm_train, trainData$type)

sms_classifier <- train(sms_dtm_train, trainData$type, method = "nb",
trControl = trainControl(method = "cv", number = 10)) # K fold Cross
Validation where K=10

sms_classifier[[2]][1:5]

sms_test_pred <- predict(sms_classifier$finalModel, sms_dtm_test)$class
```

```r
#table actual (row) vs. predicted (col): confusion matrix

Accuracy(sms_test_pred, testData$type)

F1_Score(sms_test_pred, testData$type)
```

```
>
> Accuracy(sms_test_pred, testData$type)
[1] 0.8787879
>
> F1_Score(sms_test_pred, testData$type)
[1] 0.9354839
>
```

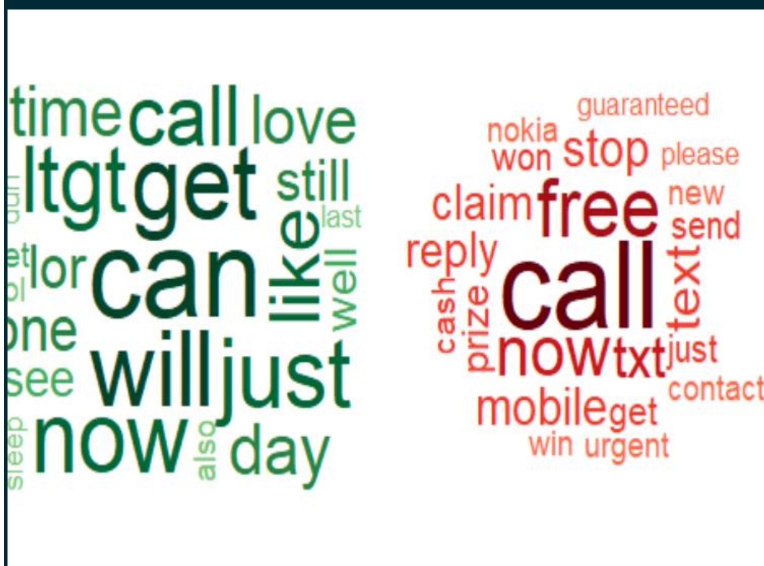
	Accuracy	F1_Score
Whole data set	0.7836625	0.8771035
500 data subset points and seed value set to 0698	0.8787879	0.9354

Text Summarization:

I USED K FOLD CROSS VALIDATION WHERE K=10

The data set when considered completely and applied k fold cross validation with training as 80% and testing as 20% of data has more accuracy and F1_Score values when compared to the model where 500 points are sampled and applied k fold cross validation with training as 80% and testing as 20% of data and seed value set. Hence, whole data set fit model is the best fit and good classifier model.

#####******COMPARING TWO WORDCLOUD FIGURE****####



PART A

PART B