Filtering mobile phone spam with the Naive Bayes algorithm

Group-7 **Members:-**

201751003-Aditya Prakash

201751031-Parishkrit Goel

201751038-Praveen Kumar

201752002-Abhimanyu Raj

Abstract

Due to the growth of worldwide mobile usage, a new avenue for electronic junk mail has opened for disreputable marketers. These advertisers utilize Short Message Service (SMS) text messages to target potential consumers with unwanted advertising known as SMS spam. SMS messages are often limited to 160 characters, reducing the amount of text that can be used to identify whether a message is junk. The limit, combined with SMS shorthand lingo, further blurs the line between legitimate messages and spam. One way spam emails are sorted is by using a Naive Bayes classifier. This algorithm will classify each object by looking at all of it's features individually. The posterior probability of the object is calculated for each feature and then these probabilities are multiplied together to get a final probability. This probability is calculated for the other class as well. Whichever has the greater probability that ultimately determines what class the object is in.

Data Collection

To develop the Naive Bayes classifier, we used data adapted from the SMS Spam Collection at http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/. This dataset includes the text of SMS messages along with a label. Junk messages are labeled spam, while legitimate messages are labeled ham. Some examples of spam and ham are shown below

##Sample SMS ham

- Better. Made up for Friday and stuffed myself like a pig yesterday. Now I feel bleh. But, at least, its not writhing pain kind of bleh.
- If he started searching, he will get job in few days. He has great potential and talent.
- I got another job! The one at the hospital, doing data analysis or something, starts on Monday! Not sure when my thesis will finish.

##Sample SMS spam

- Congratulations ur awarded 500 of CD vouchers or 125 gift guaranteed & Free entry 2 100 wkly draw txt MUSIC to 87066.
- December only! Had your mobile 11mths+? You are entitled to update to the latest colour camera mobile for Free! Call The Mobile Update Co FREE on 08002986906.
- Valentines Day Special! Win over £1000 in our quiz and take your partner on the trip of a lifetime! Send GO to 83600 now. 150 p/msg rcvd.

One notable characteristic is most of the spam messages use the word "free," yet the word does not appear in most of the ham messages.

Data Exploration

The first step towards constructing our classifier involves processing the raw data for analysis. We will transform our data into a representation known as bag-of-words, which ignores word order and simply provides a variable indicating whether the word appears at all.

We'll begin by importing the CSV data and saving it in a data frame:

```
sms_raw <- read.csv("spam1.csv", stringsAsFactors = FALSE)
```

Using the str() function, we see that the sms_raw data frame includes 5,559 total SMS messages with two features: type and text. The SMS type has been coded as either ham or spam. The text element stores the full raw SMS text.

```
str(sms_raw)

## 'data.frame':    5572 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 there got amore wat..." "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 87121 to receive entry
```

```
question("| __truncated__ "U dun say so early hor... U c already then
say..." ...
```

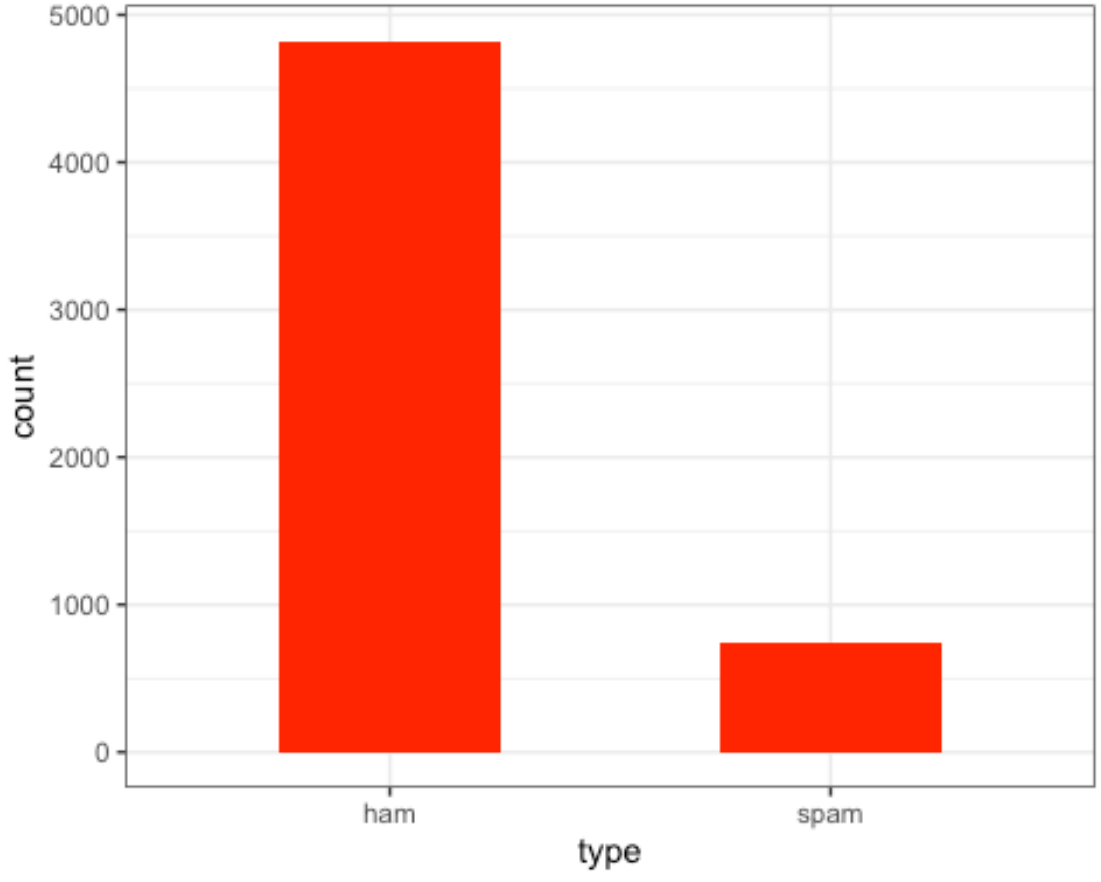
The type element is currently a character vector. Since this is a categorical variable, it would be better to convert it into a factor.

```
sms_raw$type <- factor(sms_raw$type)
```

Examining this with the str() and table() functions, we see that 747 (about 13 percent) of SMS messages in our data were labeled as spam, while the others were labeled as ham:

```
str(sms_raw$type)

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

table(sms_raw$type)

##
##  ham spam
## 4825  747

library(tm)#vcorpus

## Loading required package: NLP

library(SnowballC)
library(klaR)

## Loading required package: MASS

library(ggplot2)

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
##
##     annotate

library(pROC)

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var


library(RColorBrewer)

#checking the distribution of type of messages
theme_set(theme_bw())
ggplot(aes(x=type),data=sms_raw) + geom_bar(fill="red",width=0.5)
```



Data preparation - cleaning and standardizing text data

It involves removing numbers and punctuation, stopwords: this functionality has been provided by the text mining package titled tm. The first step in processing text data involves creating a corpus, which is a collection of text documents. We create a corpus, using the VCorpus() function in the tm package. The resulting corpus object is saved with the name sms_corpus.

```
library(tm)
sms_corpus <- VCorpus(VectorSource(sms_raw$text))
```

By printing the corpus, we see that it contains documents for each of the 5,559 SMS messages in the training data:

```
print(sms_corpus)

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5572
```

To view multiple documents, we use lapply() command to apply as.character() to a subset of corpus elements is as follows:

```
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..."
```

We start by standardizing the messages to use only lowercase characters

```
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

Text before

```
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..."
```

Text after 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..."
```

As expected, uppercase letters have been replaced by lowercase versions of the same.

Now we continue our cleanup by removing numbers from the SMS messages. we'll strip all the numbers from the corpus.

```
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.

```
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
```

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

```
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

Another standardization for text data involves reducing words to their root form in a process called stemming. The stemming process takes words like learned, learning, and learns, and strips the suffix in order to transform them into the base form, learn. Now we reduce the text to its root form using stemming provided in the SnowballC package.

```
library(SnowballC)
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

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

```
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

The following table shows the first three messages in the SMS corpus before and after the cleaning process. The messages have been limited to the most interesting words, and punctuation and capitalization have been removed:

##SMS messages before cleaning

```
as.character(sms_corpus[1:3])
```

```
## [1] "list(list(content = \"Go until jurong point, crazy.. Available
only in bugis n great world la e buffet... Cine there got amore wat...
\", meta = list(author = character(0), datetimestamp = list(sec =
49.4880828857422, min = 48, hour = 17, mday = 22, mon = 9, year = 120,
wday = 4, yday = 295, isdst = 0), description = character(0), heading
= character(0), id = \"1\", language = \"en\", origin =
character(0))), list(content = \"Ok lar... Joking wif u oni...\", meta
= list(author = character(0), datetimestamp = list(\n    sec =
49.4896850585938, min = 48, hour = 17, mday = 22, mon = 9, year = 120,
wday = 4, yday = 295, isdst = 0), description = character(0), heading
= character(0), id = \"2\", language = \"en\", origin =
character(0))), list(content = \"Free entry in 2 a wkly comp to win FA
Cup final tkts 21st May 2005. Text FA to 87121 to receive entry
question(std txt rate)T&C's apply 08452810075over18's\", meta =
list(author = character(0), datetimestamp = list(sec =
49.4899399280548, min = 48, hour = 17, mday = 22, \n    mon = 9, year
= 120, wday = 4, yday = 295, isdst = 0), description = character(0),
heading = character(0), id = \"3\", language = \"en\", origin =
character(0))))"
## [2] "list()"
## [3] "list()"
```

##SMS messages after cleaning

```
as.character(sms_corpus_clean[1:3])

## [1] "list(list(content = \"go jurong point crazi avail bugi n great
world la e buffet cine got amor wat\", meta = list(author =
character(0), datetimestamp = list(sec = 49.4880828857422, min = 48,
hour = 17, mday = 22, mon = 9, year = 120, wday = 4, yday = 295, isdst
= 0), description = character(0), heading = character(0), id = \"1\",
language = \"en\", origin = character(0))), list(content = \"ok lar
joke wif u oni\", meta = list(author = character(0), datetimestamp =
list(sec = 49.4896850585938, min = 48, \n    hour = 17, mday = 22, mon
= 9, year = 120, wday = 4, yday = 295, isdst = 0), description =
character(0), heading = character(0), id = \"2\", language = \"en\",
origin = character(0))), list(content = \"free entri wkli comp win fa
cup final tkts st may text fa receiv entri questionstd txt ratetc
appli s\", meta = list(author = character(0), datetimestamp = list(sec
= 49.4899399280548, min = 48, hour = 17, mday = 22, mon = 9, year =
120, wday = 4, yday = 295, isdst = 0), description = character(0),
heading = character(0), \n    id = \"3\", language = \"en\", origin =
character(0))))"
```

```
## [2] "list()"
## [3] "list()"
```

Data preparation - splitting text documents into words

Now the messages are split into individual components through a process called tokenization. The DocumentTermMatrix() function takes a corpus and creates a data structure called a Document Term Matrix (DTM) in which rows indicate documents (SMS messages) and columns indicate terms (words).

```
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
```

This will create an sms_dtm object that contains the tokenized corpus using the default settings, which apply minimal processing.

Another method to perform preprocessing directly:

```
sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
  tolower = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  removePunctuation = TRUE,
  stemming = TRUE))
```

This applies the same preprocessing steps to the SMS corpus in the same order as done earlier.

Data preparation

creating training and test datasets

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 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 4,169 for training and leave the remaining 1,390 for testing.

```
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test <- sms_dtm[4170:5559, ]
```

It is also helpful to save a pair of vectors with labels for each of the rows in the training and testing matrices. These labels are not stored in the DTM, so we pull them from the original sms_raw data frame:

```
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels <- sms_raw[4170:5559, ]$type
```

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

```
prop.table(table(sms_train_labels))

## sms_train_labels
##       ham      spam
## 0.8647158 0.1352842

prop.table(table(sms_test_labels))

## sms_test_labels
##       ham      spam
## 0.8697842 0.1302158
```

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

A word cloud is composed of words scattered somewhat randomly around the figure. Words appearing more often in the text are shown in a larger font, while less common terms are shown in smaller fonts. Since we specified random.order = FALSE, the cloud will be arranged in a nonrandom order with higher frequency words placed closer to the center.

```
library(wordcloud)
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```

Now we create two subsets (i.e. clouds) for SMS spam and ham.

```
spam <- subset(sms_raw, type == "spam")

ham <- subset(sms_raw, type == "ham")
```

We use the max.words parameter to look at the 40 most common words in each of the two sets. The scale parameter allows us to adjust the maximum and minimum font size for words in the cloud.

```
wordcloud(spam$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, 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
```

Notice 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 keywords to differentiate between the classes.

Data preparation

creating indicator features for frequent words

The final step in the data preparation process is to transform the DTM into a data structure that can be used to train a Naive Bayes classifier. To reduce the number of features, we eliminate any word that appears in less than five SMS messages.

```
findFreqTerms(sms_dtm_train, 5)
```

```
##    [1] "£wk"          "abiola"       "abl"          "abt"
##    [5] "accept"       "access"       "account"      "across"
##    [9] "activ"        "actual"       "add"          "address"
##   [13] "admir"        "adult"        "advanc"       "aft"
##   [17] "afternoon"    "aftr"         "age"          "ago"
##   [21] "ahead"        "aight"        "aint"         "air"
##   [25] "aiyah"        "alex"         "almost"       "alon"
##   [29] "alreadi"      "alright"      "alrit"        "also"
##   [33] "alway"        "amp"          "angri"        "announc"
##   [37] "anoth"        "answer"       "anybodi"      "anymor"
##   [41] "anyon"        "anyth"        "anytim"       "anyway"
##   [45] "apart"        "app"          "appli"        "appoint"
##   [49] "appreci"      "april"        "ard"          "area"
##   [53] "argument"     "arm"          "around"       "arrang"
##   [57] "arrest"       "arriv"        "asap"         "ask"
##   [61] "askd"         "asleep"       "ass"          "attempt"
##   [65] "auction"      "avail"        "ave"          "avoid"
##   [69] "await"        "award"        "away"         "awesom"
##   [73] "babe"         "babi"         "back"         "bad"
##   [77] "bag"          "bak"          "balanc"       "bank"
##   [81] "bare"         "bath"         "batteri"      "bcoz"
##   [85] "bcum"         "bday"         "beauti"       "becom"
##   [89] "bed"          "bedroom"      "begin"        "believ"
##   [93] "belli"        "best"         "better"       "bid"
##   [97] "big"          "bill"         "bird"         "birthday"
##  [101] "bit"          "black"        "blank"        "bless"
##  [105] "blue"         "bluetooth"    "bodi"         "bold"
##  [109] "bonus"        "boo"          "book"         "bore"
##  [113] "boss"         "bother"       "bout"         "bowl"
##  [117] "box"          "boy"          "boytoy"       "brand"
##  [121] "break"        "breath"       "brilliant"    "bring"
##  [125] "brother"      "bslvyl"       "btnationalr"  "budget"
##  [129] "bugi"         "bus"          "busi"         "buy"
##  [133] "buzz"         "cabin"        "cafe"         "cal"
##  [137] "call"         "caller"       "callertun"    "camcord"
##  [141] "came"         "camera"       "can"          "cancel"
##  [145] "cant"         "car"          "card"         "care"
##  [149] "carlo"        "case"         "cash"         "cashbal"
##  [153] "catch"        "caus"         "chanc"        "chang"
##  [157] "charact"      "charg"        "chariti"      "chat"
```

```
## [1061] "ure"           "urgent"        "urself"        "use"
## [1065] "usf"           "usual"         "uve"           "valentin"
## [1069] "valid"         "valu"          "vari"          "verifi"
## [1073] "via"           "video"         "visit"         "voic"
## [1077] "voucher"       "wait"          "wake"          "walk"
## [1081] "wan"           "wana"          "wanna"         "want"
## [1085] "wap"           "warm"          "wast"          "wat"
## [1089] "watch"         "water"         "way"           "weak"
## [1093] "wear"          "weather"       "wed"           "wednesday"
## [1097] "weed"          "week"          "weekend"       "weight"
## [1101] "welcom"        "well"          "wen"           "went"
## [1105] "wer"           "wet"           "what"          "whatev"
## [1109] "whenev"        "whole"         "wid"           "wif"
## [1113] "wife"          "wil"           "will"          "win"
## [1117] "wine"          "winner"        "wish"          "wit"
## [1121] "within"        "without"       "wiv"           "wkli"
## [1125] "wnt"           "woke"          "won"           "wonder"
## [1129] "wont"          "word"          "work"          "workin"
## [1133] "world"         "worri"         "worth"         "wot"
## [1137] "wow"           "write"         "wrong"         "wun"
## [1141] "wwwgetzedcouk" "xmas"          "xxx"           "yahoo"
## [1145] "yar"           "yeah"          "year"          "yep"
## [1149] "yes"           "yest"          "yesterday"     "yet"
## [1153] "yoga"          "yogasana"      "yrs"           "yun"
## [1157] "yup"
```

The result of the function is a character vector, so we save our frequent words for later on:

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

A peek into the contents of the vector shows us that there are 1,136 terms appearing in at least five SMS messages:

```
str(sms_freq_words)

##  chr [1:1157] "£wk" "abiola" "abl" "abt" "accept" "access"
"account" ...
```

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

```
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 1,136 features, which correspond to words appearing in at least five messages.

The Naive Bayes classifier is typically trained on data with categorical features.So, we now change this to a categorical variable that simply indicates yes or no depending on whether the word appears at all, by using convert_counts() function to convert counts to Yes/No strings.

```
convert_counts <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}
```

We now apply convert_counts() to each of the columns in our sparse matrix.

```
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2,
convert_counts)

sms_test <- apply(sms_dtm_freq_test, MARGIN = 2,
convert_counts)
```

The result is a two character type matrix, 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 that we have transformed the raw SMS messages into a format that can be represented by a statistical model. We build our model on the sms_train matrix, using the naive bayes function in library e1071

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

The sms_classifier object now contains a naiveBayes classifier object that can be used to make predictions.

Step 4 - evaluating model performance

To evaluate the SMS classifier, we need to test its predictions on unseen messages in the test data. The predict() function is used to make the predictions. We store these in a vector named sms_test_pred.

```
sms_test_pred <- predict(sms_classifier, sms_test)
```

To compare the predictions to the true values, we'll use the CrossTable() function in the gmodels package:

```
library(gmodels)

##
## Attaching package: 'gmodels'

## The following object is masked from 'package:pROC':
##
##     ci

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:  1390
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1200 |        20 |      1220 |
##              |     0.984 |     0.016 |     0.878 |
##              |     0.993 |     0.110 |           |
## -------------|-----------|-----------|-----------|
##         spam |         9 |       161 |       170 |
##              |     0.053 |     0.947 |     0.122 |
##              |     0.007 |     0.890 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1209 |       181 |      1390 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Now, we see that a total of only 20 + 9 = 29 of the 1,390 SMS messages were incorrectly classified (2.6 percent). Among the errors were 9 out of 1,209 ham messages that were misidentified as spam, and 20 of the 181 spam messages were incorrectly labeled as ham.

```
library(caret)

## Loading required package: lattice

# summarize results
confusionMatrix(sms_test_pred, sms_test_labels)

## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  ham spam
##       ham  1200   20
##       spam    9  161
##
##                Accuracy : 0.9791
##                  95% CI : (0.9702, 0.986)
##     No Information Rate : 0.8698
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9055
##
##  Mcnemar's Test P-Value : 0.06332
##
##             Sensitivity : 0.9926
##             Specificity : 0.8895
##          Pos Pred Value : 0.9836
##          Neg Pred Value : 0.9471
##              Prevalence : 0.8698
##          Detection Rate : 0.8633
##    Detection Prevalence : 0.8777
##       Balanced Accuracy : 0.9410
##
##        'Positive' Class : ham
##
```

Summary

Naive Bayes algorithm constructs tables of probabilities that are used to estimate the likelihood that new examples belong to various classes. The probabilities are calculated using a formula known as Bayes' theorem, which specifies how dependent events are related. Although Bayes' theorem can be computationally expensive, a simplified version that makes so-called "naive" assumptions about the independence of features is capable of handling extremely large datasets. The Naive Bayes classifier is often used for text classification. To illustrate its effectiveness, we employed Naive Bayes on a classification task involving spam SMS messages. Preparing the text data for analysis required the use of specialized R packages for text processing and visualization. Ultimately, the model was able to classify over 97 percent of all the SMS messages correctly as spam or ham.