****

Machine Learning

B9DA104

|  |  |
| --- | --- |
| Student Names | Student ID number (email) |
| 1-Brijith Elizabeth Joseph | [10393634@mydbs.ie](mailto:10393634@mydbs.ie) |
| 2-Jayaramakrishnan Balakrishnan | [10393364@mydbs.ie](mailto:10393364@mydbs.ie) |

**Assignment : CA1**

**Naive Bayes Algorithm using R**

**JayaramaKrishnan BalaKrishnan**

**10393364**

**The Naive Bayes algorithm**

The Naive Bayes algorithm describes a simple method to apply Bayes' theorem to classification problems. Although it is not the only machine learning method that utilizes Bayesian methods, it is the most common one. The Naive Bayes algorithm is named as such because it makes some "naive“ assumptions about the data. In particular, Naive Bayes assumes that all of the features in the dataset are equally important and independent. These assumptions are rarely true in most real-world applications.

Bayesian classifiers are best applied to problems in which the information from numerous attributes should be considered simultaneously in order to estimate the overall probability of an outcome. Many machine learning algorithms ignore features that have weak effects, Bayesian methods utilize all the available evidence to subtly change the predictions. If large number of features have relatively minor effects, taken together, their combined impact could be quite large. Bayesian probability theory is rooted in the idea that the estimated likelihood of an event, or a potential outcome, should be based on the evidence at hand across multiple trials, or opportunities for the event to occur. The probability of all the possible outcomes of a trial must always sum to 1, because a trial always results in some outcome happening. Thus, if the trial has two outcomes that cannot occur simultaneously.we are interested in monitoring several non mutually exclusive events for the same trial. If certain events occur with the event of interest, we may be able to use them to make predictions

They have been used successfully for:

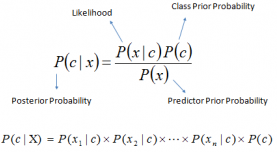
• Text classification, such as junk e-mail (spam) filtering

• Intrusion or anomaly detection in computer networks

• Diagnosing medical conditions given a set of observed symptoms

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naïve Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).



* P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
* P(c) is the prior probability of class.
* P(x|c) is the likelihood which is the probability of predictor given class.
* P(x) is the prior probability of predictor.

***Pros:***

* It is easy and fast to predict class of test data set. It also perform well in multi class prediction
* When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
* It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

***Cons:***

* If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
* On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict\_proba are not to be taken too seriously.
* Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

**Dataset**

The dataset ‘Spam Filter ’is took from Kaggle. It is an open dataset with 5728 records. The file is in the form of excel. It has two fields’ text and spam. The field text contains the message, and Spam contains binary data. 1 represents the message is spam and 0 represents the message is ham.

A spam filter is a program that is used to detect unsolicited and unwanted email and prevent those messages from getting to a user's inbox. Like other types of filtering programs, a spam filter looks for certain criteria on which it bases judgments. For example, the simplest and earliest versions (such as the one available with Microsoft's Hotmail) can be set to watch for particular words in the subject line of messages and to exclude these from the user's inbox. This method is not especially effective, too often omitting perfectly legitimate messages (these are called false positives) and letting actual spam through. More sophisticated programs, such as Bayesian filters or other heuristic filters, attempt to identify spam through suspicious word patterns or word frequency.

The url is : <https://www.kaggle.com/karthickveerakumar/spam-filter>

Using the dataset, we created a model that can predict whether a message is spam or ham. As per the content in the message, we are classifying the messages into two categories, Spam and Ham

**Code with explanation**

**Step 1:**

# Install Required Packages

install.packages("tm")

library(tm)

install.packages("SnowballC")

library(SnowballC)

install.packages("wordcloud")

library('wordcloud')

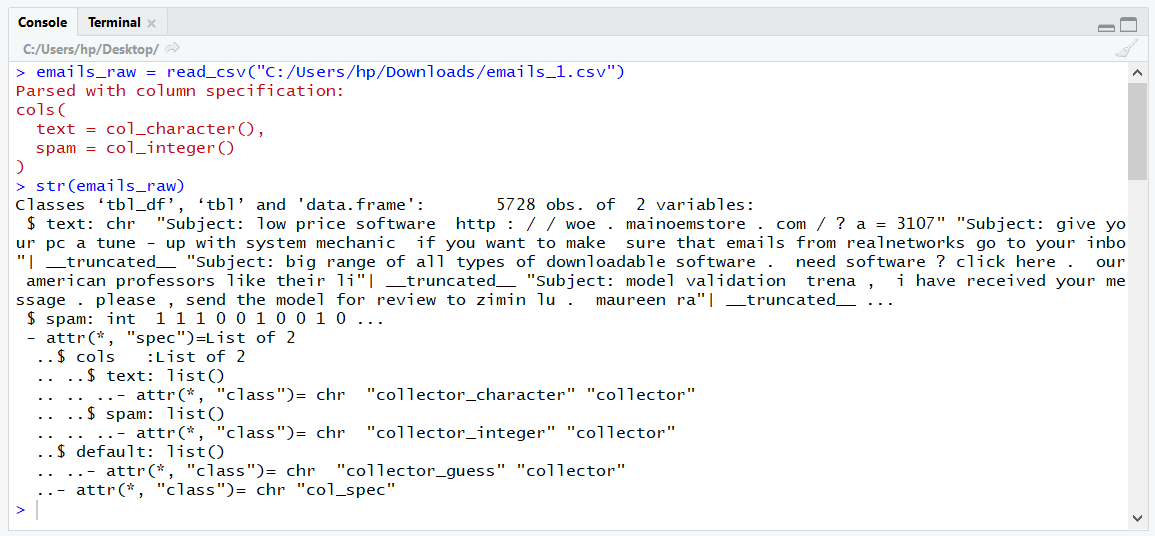
library(readr)

**Step 2:**

**# Load the dataset and check the Stucture of the data**

emails\_raw = read\_csv("C:/Users/hp/Downloads/emails\_1.csv")

str(emails\_raw)warnings.filterwarnings('ignore')

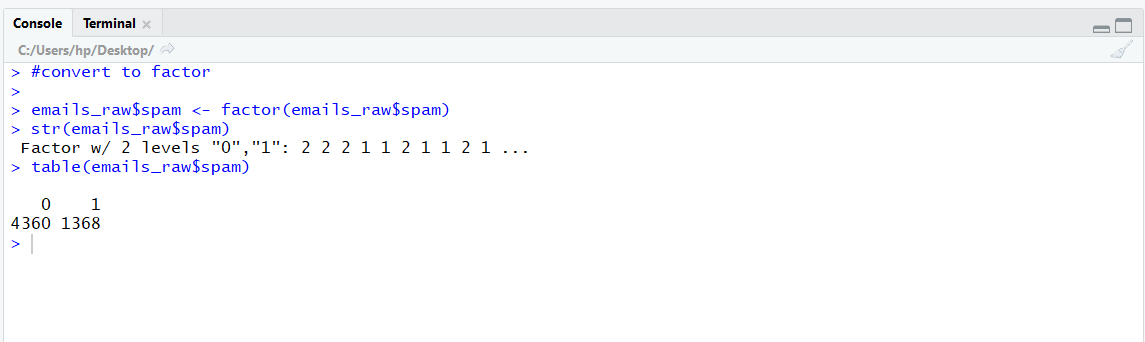


**#convert to factor and check the count values of the dataset**

emails\_raw$spam <- factor(emails\_raw$spam)

str(emails\_raw$spam)

table(emails\_raw$spam)



This out put giving us how many values are spam and ham

0 – It’s mentioning Ham emails

1 – It’s mentioning Spam emails

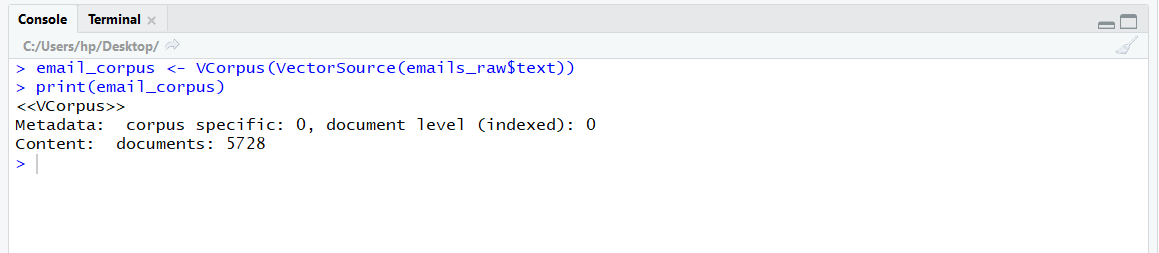
**# Step 3**

**# Pre-processing**

# For Pre-processing step we are using a corpus for the text values

email\_corpus <- VCorpus(VectorSource(emails\_raw$text))

print(email\_corpus)



**#To receive a summary of specific email, we can use the inspect() function with list operators**

inspect(email\_corpus[1:2])

**#to view the email**

as.character(email\_corpus[[2]])

lapply(email\_corpus[1:2], as.character)

**# The tm\_map() function provides a method to apply a transformation. We will use this function to clean up our corpus using a series of transformations 1st convert all to lower**

email\_corpus\_clean <- tm\_map(email\_corpus,

content\_transformer(tolower))

**# Removing stopwords, Numbers, Punctuation and Whitespaces**

email\_corpus\_clean <- tm\_map(email\_corpus\_clean,

removeWords, stopwords("english"))

email\_corpus\_clean <- tm\_map(email\_corpus\_clean, removeWords,c("subject"))

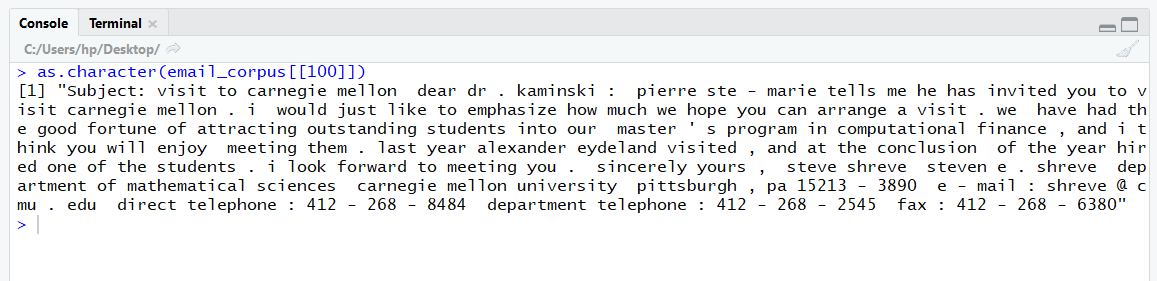
email\_corpus\_clean <- tm\_map(email\_corpus\_clean, removeNumbers)

email\_corpus\_clean <- tm\_map(email\_corpus\_clean, removePunctuation)

email\_corpus\_clean <- tm\_map(email\_corpus\_clean, stripWhitespace)

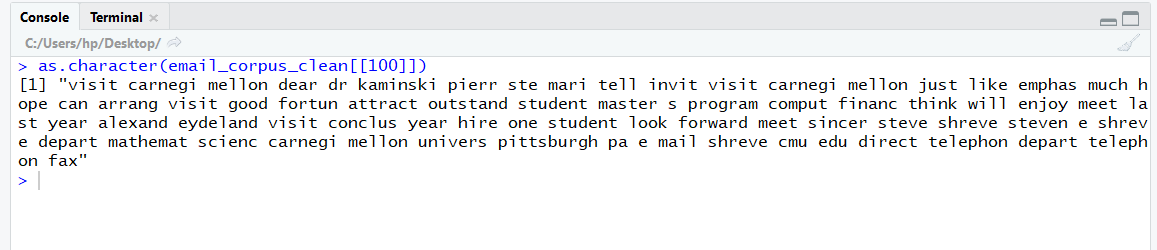
**#check it worked**

as.character(email\_corpus[[1]])



This above picture shows the text Before Pre-processing

as.character(email\_corpus\_clean[[1]])



This above picture shows the text After Pre-processing

**# Standardization for text data involves reducing words to their root form in a process called stemming.**

email\_corpus\_clean <- tm\_map(email\_corpus\_clean, stemDocument)

**# Splitting text documents into words using Document Term Matrix (DTM)**

email\_dtm <- DocumentTermMatrix(email\_corpus\_clean)

**#DTM cleaning Method 2**

email\_dtm2 <- DocumentTermMatrix(email\_corpus, control = list(

tolower = TRUE,

removeNumbers = TRUE,

stopwords = TRUE,

removePunctuation = TRUE,

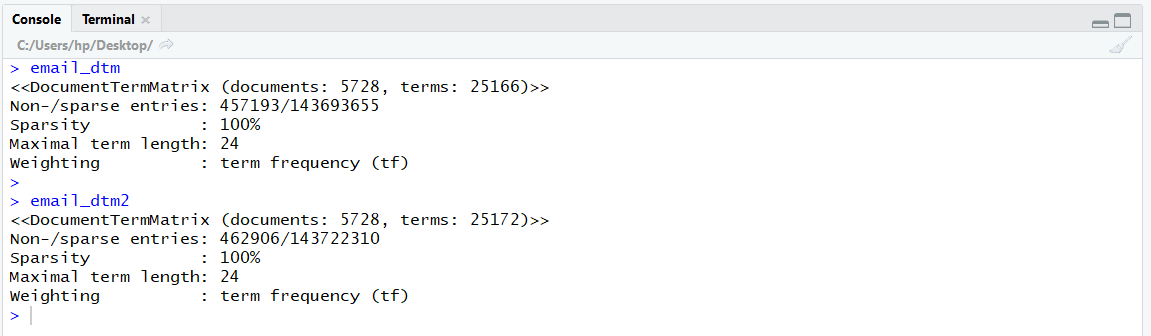
stemming = TRUE

))

**# Differences (Two types of Pre-processing differences)**

email\_dtm

email\_dtm2



**# Step 4**

**# Data preparation - creating training and test datasets (75% for train and 25% for test)**

email\_dtm\_train <- email\_dtm[1:4296, ]

email\_dtm\_test <- email\_dtm[4297:5728, ]

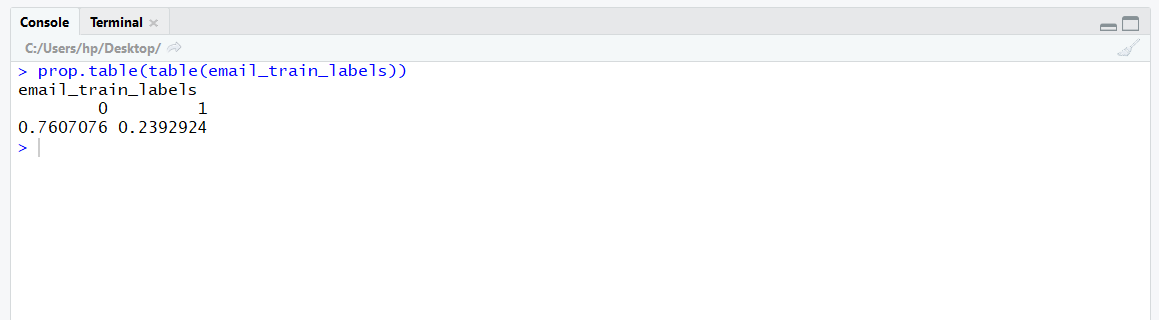
**# Labels**

email\_train\_labels <- emails\_raw[1:4296, ]$spam

email\_test\_labels <- emails\_raw[4297:5728, ]$spam

prop.table(table(email\_train\_labels))

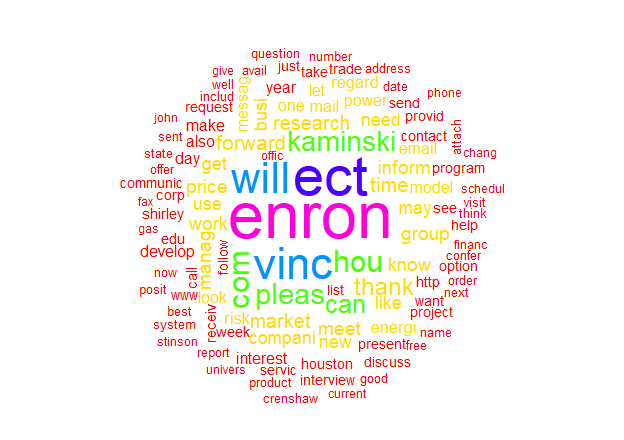
prop.table(table(email\_test\_labels))



Property table for train labels

**# Visualizing text data - word clouds**

wordcloud(email\_corpus\_clean, min.freq = 1000, random.order = FALSE, col=rainbow(7))



This above picture shows the word frequency more than 1000 times

**# Subset where the email type is spam:**

spam <- subset(emails\_raw, spam == "1")

ham <- subset(emails\_raw, spam == "0")

wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))

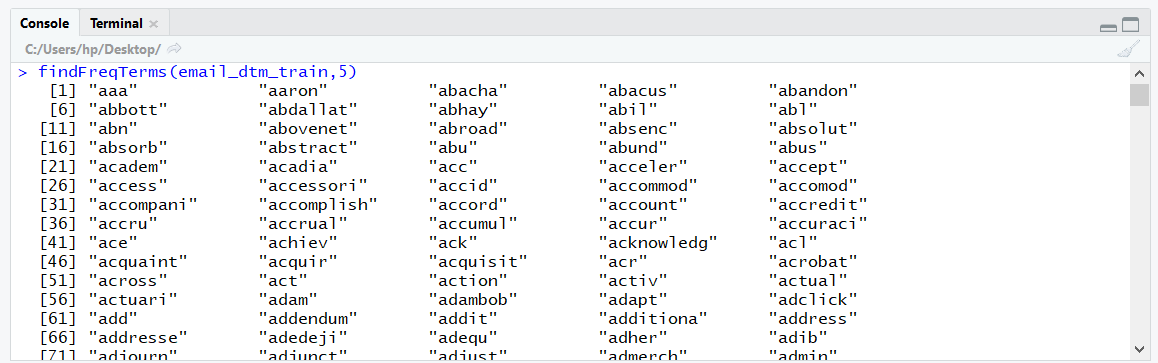
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))

**# Creating indicator features for frequent words. Words appearing aleast 5 time below**

findFreqTerms(email\_dtm\_train,5)

email\_freq\_words <- findFreqTerms(email\_dtm\_train, 5)

str(email\_freq\_words)



**# Filter only appearing in certain vector**

email\_dtm\_freq\_train<- email\_dtm\_train[ , email\_freq\_words]

email\_dtm\_freq\_test <- email\_dtm\_test[ , email\_freq\_words]

**#The following defines a convert\_counts() function to convert counts to Yes/No strings:**

convert\_counts <- function(x) {

x <- ifelse(x > 0, "Yes", "No")

}

**#We need to apply convert\_counts() to each of the columns in our sparse. MARGIN parameter to specify either rows or columns. MARGIN = 2columns, MARGIN = 1 is rows**

email\_train <- apply(email\_dtm\_freq\_train, MARGIN = 2,

convert\_counts)

email\_test <- apply(email\_dtm\_freq\_test, MARGIN = 2,

convert\_counts)

**# Step:5**

**# The Naive Bayes implementation**

install.packages("e1071")

library(e1071)

install.packages("caret")

library(caret)

library(gmodels)

**# Naïve Bayes Algorithm**

email\_classifier <- naiveBayes(email\_train, email\_train\_labels)

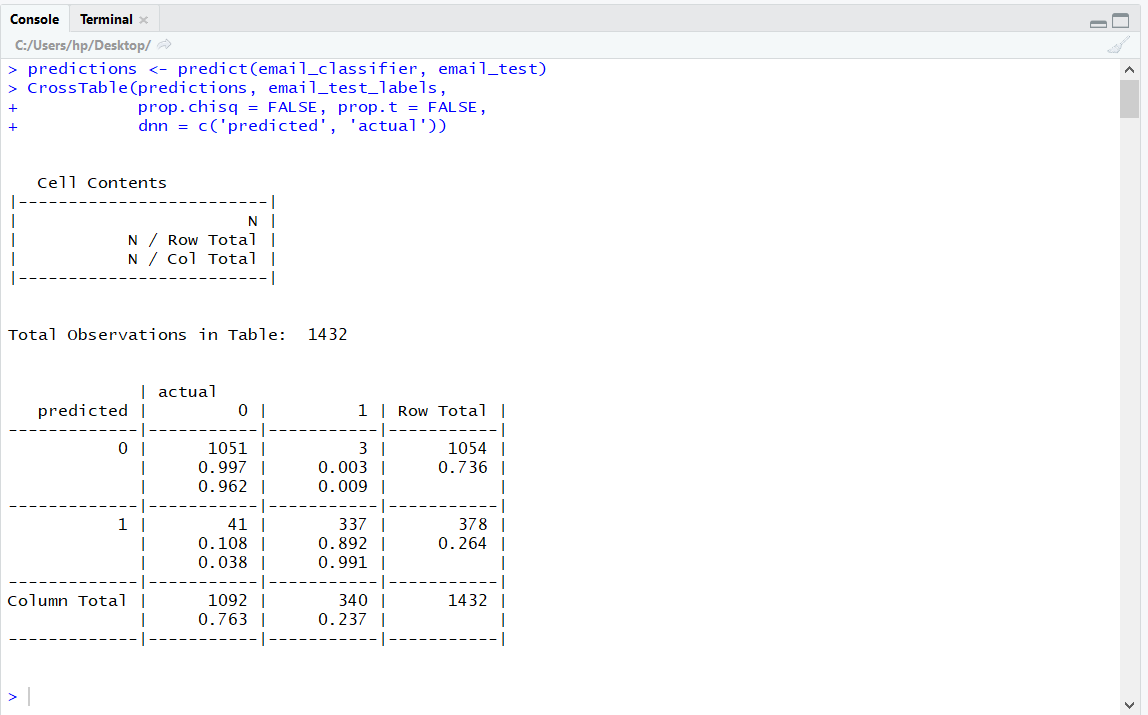
**# Evaluating model performance make the predictions and Cross Table Validation**

predictions <- predict(email\_classifier, email\_test)

CrossTable(predictions, email\_test\_labels,

prop.chisq = FALSE, prop.t = FALSE,

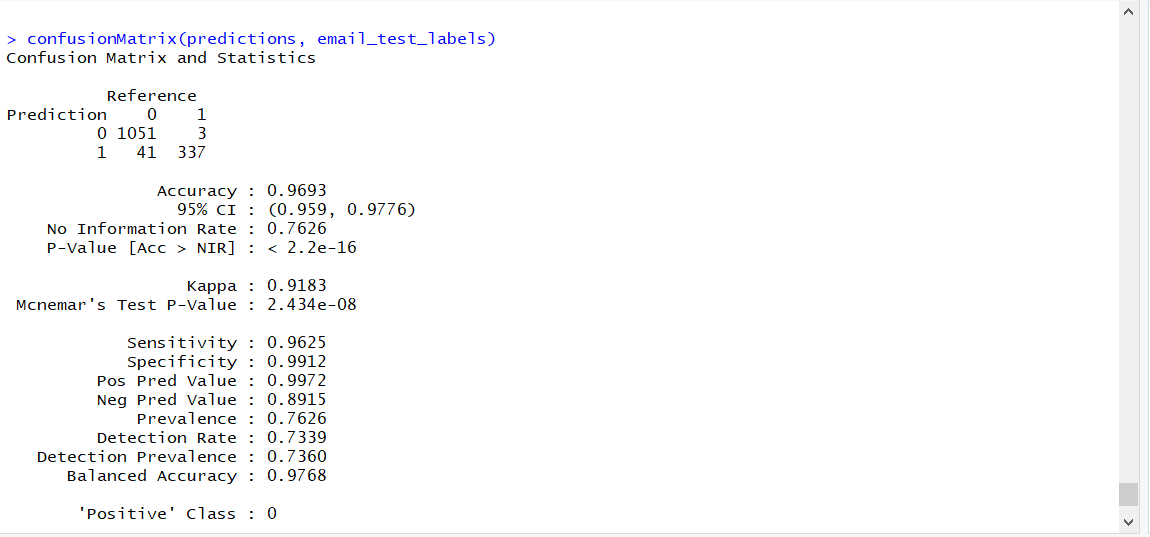
dnn = c('predicted', 'actual'))



This above picture giving the observations based on our model prediction

**# Summarize Results (confusion Matrix and Accuracy)**

confusionMatrix(predictions, email\_test\_labels)



This naive model gives 96.93% accuracy

**# Step 6**

**# Improving model performance**

email\_classifier\_1 <- naiveBayes(email\_train, email\_train\_labels,

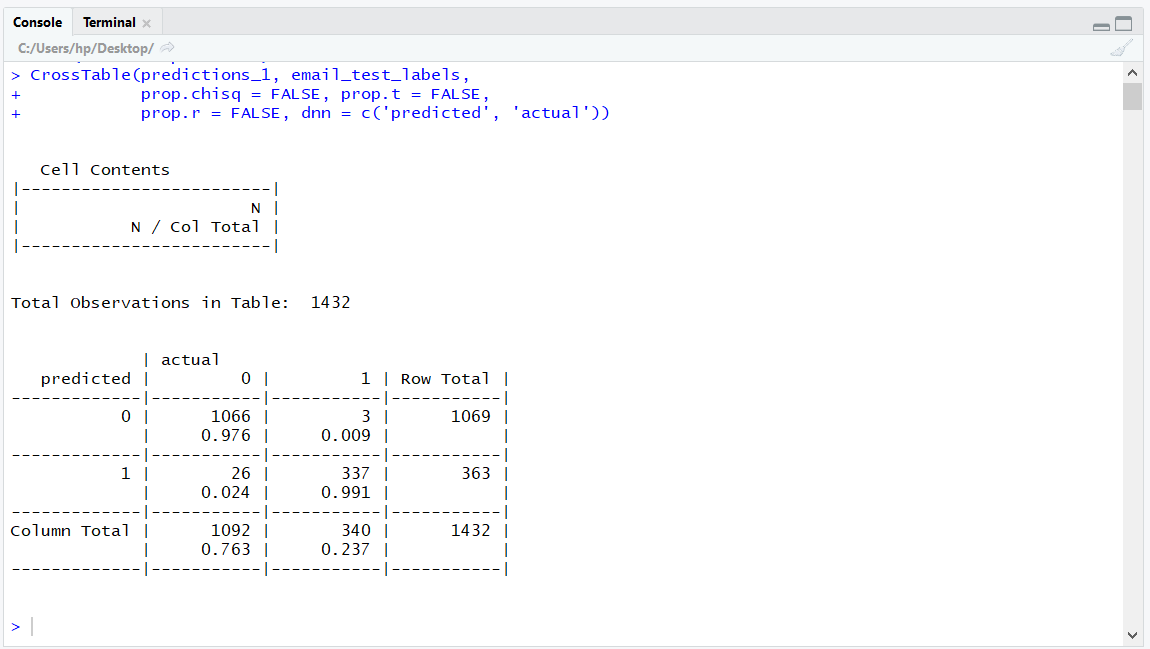
laplace = 1)

predictions\_1 <- predict(email\_classifier\_1, email\_test)

CrossTable(predictions\_1, email\_test\_labels,

prop.chisq = FALSE, prop.t = FALSE,

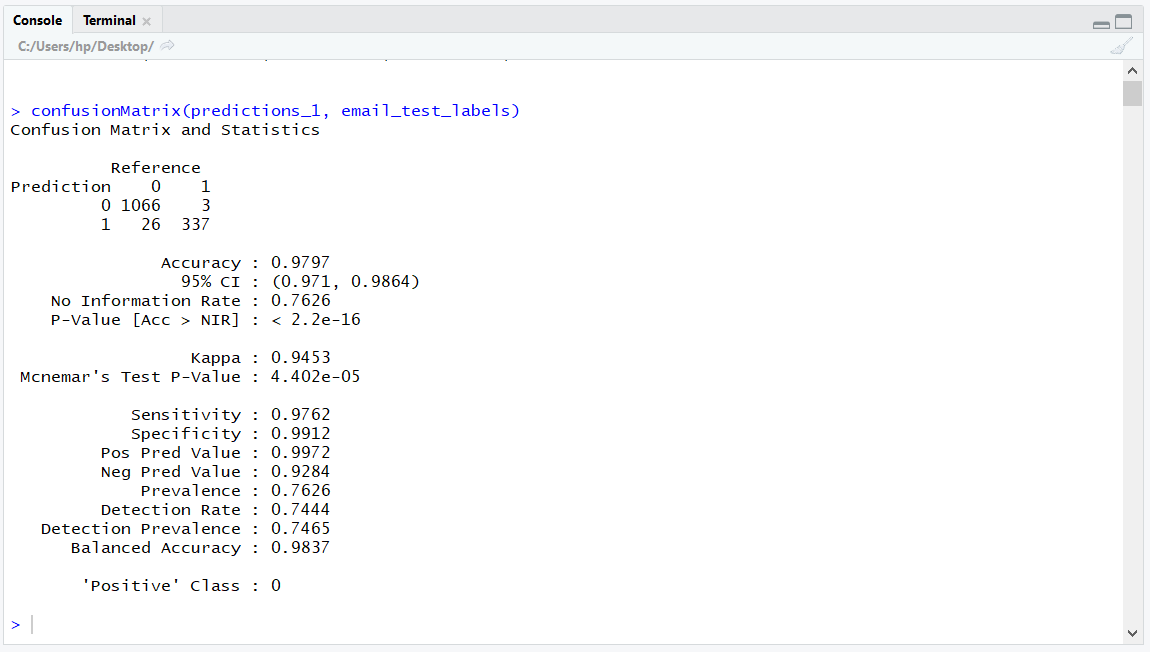
prop.r = FALSE, dnn = c('predicted', 'actual'))



This above picture giving the observations based on our improved model prediction. Compared model one it gives better result

**# Improved Model performace Summarize Results (confusion Matrix and Accuracy)**

confusionMatrix(predictions\_1, email\_test\_labels)



This improved naive model gives 97.97% accuracy. Compared to our first model it gives the better accuracy when we use some parameter tuning

**# Step 7**

# Realtime Test

test.data <- data.frame(Text="Hi Pallu, I sent the assignment file. Check it and let me know")

test\_pred = predict(email\_classifier\_1, newdata=test.data, interval="prediction")

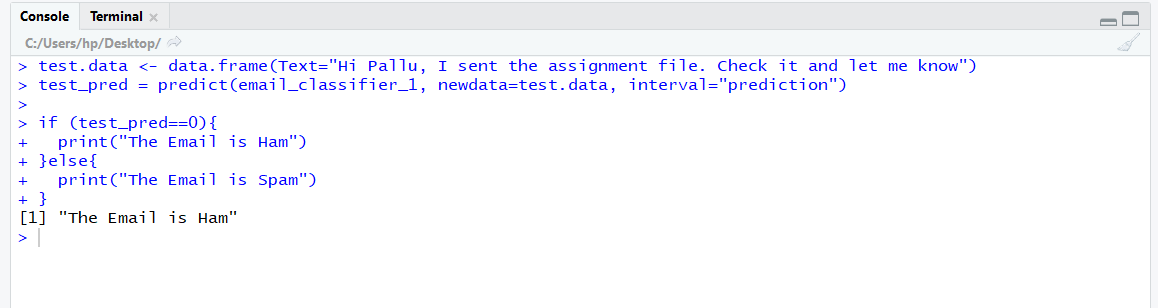
if (test\_pred==0){

print("The Email is Ham")

} else {

print("The Email is Spam")

}



The realtime test gives the prediction based on our model. The above picture our our test data predict the email is Ham