**STA 141 Assignment 5**

Junxiao Bu

999452701

" I certify that I have acknowledged any code that I used from any other person in the class, from Piazza or any Web site or book or other source. Any other work is my own. *"*

**Training Data**

1. **KNN**

With 10 folds cross validation, for KNN, I used three kinds of distance: Euclidean distance, Manhattan distance and Minkowski distance. I tried k from 1 to 30. The total average error rate for each combination of k and distance is in the following graph:

Macintosh HD:Users:Bruce:Desktop:Rplot01.pdf

From the graph, I can find that all the Apply three kinds of distance, I can find that using Manhattan distance can give the lowest prediction error rate when k=3. So I choose to use Manhattan distance and let k=3.

Here I will firstly define my version of **Type I error** and **Type II error**. In the rest part of the report, I will use this version of error.

**Type I error**: isSpam is True but prediction is False

**Type II error**: isSpam is False but prediction is True

The confusion matrix for 6541 observations in the training data is:

|  |  |  |
| --- | --- | --- |
|  | TRUE | FALSE |
| TRUE (predict) | 1384 | 187 |
| FALSE (predict) | 293 | 4677 |

The type I error rate: 293/(1384+293) = 17.47%.

The type II error rate: 187/(187+4677) = 3.844%.

Total prediction error rate: (293+187)/6541 = 7.338%.

The classification rate is 92.662%.

It seems that most of the misspecification cases are type I error. It suggests that we may consider more SPAM as HAM rather than considering HAM as SPAM. That makes sense. Because the number of HAM are far more than number of SPAM.

**Misclassification Cases Characteristics**

Now I can see some characteristics for these misclassified cases. For k=3, check the misclassified cases. There are 480 misclassified cases. 293 of them are considered as HAM while being SPAM. 187 of them are considered as SPAM while being HAM. I will write a function to get the indexes of all the misclassified cases. The function is in the appendix.

For all the logical variables, I get all the mosaic plot of correctly classified cases and misclassified cases. I try to find the difference between two kinds of cases in the same variable.

Here I present two logical variables’ pairwise mosaic plots:

(1) isInReplyTo:

Macintosh HD:Users:Bruce:Desktop:Rplot03.pdf

For this variable, in the misclassified cases, it seems that when an email doesn't have isInReplyTo, it is more possible to be SPAM. But for correctly classified cases, it is more possible to be HAM.

(2) fromNunericEnd:

Macintosh HD:Users:Bruce:Desktop:Rplot04.pdf

For this variable, in the misclassified cases, it seems that when an email doesn't have numeric end, it is more possible to be SPAM. But for correctly classified cases, it is more possible to be HAM.

I find similar pattern for almost every logical variables. This pattern suggests that my model is corresponding to misclassify some of the HAM to SPAM.

For all the numeric variables, I get all the barplots of correctly classified cases and misclassified cases. I try to find the difference between two kinds of cases in the same variable by comparing the mean values of two sets of variables.

Next I present two numeric variables’ pairwise barplots:

1. subjectExclamationCount:

Macintosh HD:Users:Bruce:Desktop:Rplot08.pdf

For this variable, in the misclassified cases, it seems that the average of exclamation marks of SPAM is higher than HAM. In the correctly classified cases, the pattern is reversed.

(2)subjectQuestionCount:

Macintosh HD:Users:Bruce:Desktop:Rplot09.pdf

For this variable, in the misclassified cases, it seems that the average of question marks of SPAM is higher than HAM. In the correctly classified cases, the pattern is reversed.

There are a few numeric variables, which have similar patterns of (1) and (2). Other numeric variables have same patterns for both misclassified cases and correctly classified cases.

1. **Classification Tree**

Using the R package **rpart,** the result is in the following graph:

Macintosh HD:Users:Bruce:Desktop:Rplot.pdf

The confusion matrix for 6541 observations in the training data is:

|  |  |  |
| --- | --- | --- |
|  | TRUE | FALSE |
| TRUE (predict) | 1319 | 233 |
| FALSE (predict) | 358 | 4631 |

The type I error rate: 293/(1384+293) = 21.34%.

The type II error rate: 187/(187+4677) = 4.79%.

Total prediction error rate: (293+187)/6541 = 9.035%.

Obviously, the error rate is higher than the best model using KNN. The type I error rate and type II error rate are also higher than the best model using KNN.

**Transformation**

Now I can transform original data to check if it would affect the result of my best model. I will fit three models. The first two are to get the square of all the variables and (1/ variable). These transformations are monotonic. Another one is **sin** function. This transformation is not monotonic. All the transformations will be used only for **numeric** variables.

1. **Square of variables:**

The confusion matrix for 6541 observations in the training data is:

|  |  |  |
| --- | --- | --- |
|  | TRUE | FALSE |
| TRUE (predict) | 1319 | 233 |
| FALSE (predict) | 358 | 4631 |

I can find that monotonic transformation won’t affect the prediction error rate. Even each element of the confusion matrix is same. So I can confirm that monotonic transformation **won’t affect the best model.**

1. **Sin (variable):**

The confusion matrix for 6541 observations in the training data is:

|  |  |  |
| --- | --- | --- |
|  | TRUE | FALSE |
| TRUE (predict) | 582 | 240 |
| FALSE (predict) | 1095 | 4624 |

**Now all the error rates change.** The type I error is 65.295%. The type II error rate is 4.934%. The total prediction error rate is 20.41%. If I transform variables using sin function, the type I error rate is ridiculous high. The total prediction error rate is significant higher than before.

(3) **1/variables**

The confusion matrix for 6541 observations in the training data is:

|  |  |  |
| --- | --- | --- |
|  | TRUE | FALSE |
| TRUE (predict) | 1181 | 155 |
| FALSE (predict) | 496 | 4709 |

**Now all the error rates change.** The type I error is 29.5766%. The type II error rate is 3.1866%. The total prediction error rate is 9.995%. This transformation is still monotonic. But the order is reversed. So the error rates should change.

If the transformation is monotonic, the best model won’t be affected. Otherwise, the best model’s error rates will change.

**Misclassification Cases Characteristics**

Now I can see some characteristics for these misclassified cases. There are 591 misclassified cases. 358 of them are considered as HAM while being SPAM. 233 of them are considered as SPAM while being HAM.

For all the logical variables, I get all the mosaic plot of correctly classified cases and misclassified cases. I try to find the difference between two kinds of cases in the same variable.

Here I present two logical variables’ pairwise mosaic plots. In order to compare with each other, I choose the same variables as in the KNN method.

(1) isInReplyTo:

Macintosh HD:Users:Bruce:Desktop:Rplot10.pdf

For this variable, in the misclassified cases, it seems that when an email doesn't have isInReplyTo, it is more possible to be SPAM. But for correctly classified cases, it is more possible to be HAM.

(2) fromNunericEnd:

Macintosh HD:Users:Bruce:Desktop:Rplot11.pdf

For this variable, in the misclassified cases, it seems that when an email doesn't have numeric end, it is more possible to be SPAM. But for correctly classified cases, it is more possible to be HAM.

I find similar pattern for almost every logical variables. This pattern suggests that my model is corresponding to misclassify some of the HAM to SPAM.

Next I present two numeric variables’ pairwise barplots:

1. subjectExclamationCount:

Macintosh HD:Users:Bruce:Desktop:Rplot13.pdf

For this variable, the pattern is different with the pattern in KNN. It seems that the average of exclamation marks of SPAM is higher than HAM in both correctly classified and misclassified cases.

1. subjectQuestionCount:

Macintosh HD:Users:Bruce:Desktop:Rplot12.pdf

For this variable, in the misclassified cases, it seems that the average of question marks of SPAM is higher than HAM. In the correctly classified cases, the pattern is reversed.

(3)numLinesInBody

Macintosh HD:Users:Bruce:Desktop:Rplot14.pdf

For this variable, in the misclassified cases, it seems that the average of lines in body part of SPAM is higher than HAM. In the correctly classified cases, the pattern is reversed.

There are a few numeric variables, which have similar patterns of (3) and (2). Other numeric variables have same patterns for both misclassified cases and correctly classified cases.

**Test Data**

1. **KNN**

For KNN, the confusion matrix for 2000 observations in the test data is:

|  |  |  |
| --- | --- | --- |
|  | TRUE | FALSE |
| TRUE (predict) | 406 | 107 |
| FALSE (predict) | 83 | 1404 |

The type I error rate: 83/(406+83) = 16.973%.

The type II error rate: 107/(107+1404) = 7.08%.

Total prediction error rate: (83+107)/2000 = 9.5%.

The type I error rate is lower in the test data than training data. The type II error is relatively higher. The total average rate is higher in the test data than training data. Since the procedure of cross validation in these two methods may not be same, the error rates are reasonably not same. After looking through both the training data and test data, I find that the HAM emails are 74.4% of the total emails in the training data, while the percentage is 75.6% in the test data.

The classification rate is 90.5%, which is lower than the classification rate in KNN.

1. **Classification Tree**

For classification tree, the confusion matrix for 2000 observations in the test data is:

|  |  |  |
| --- | --- | --- |
|  | TRUE | FALSE |
| TRUE (predict) | 333 | 88 |
| FALSE (predict) | 156 | 1423 |

The type I error rate: 156/(156+333) = 31.902%.

The type II error rate: 88/(88+1423) = 5.824%.

Total prediction error rate: (88+156)/2000 = 12.2%.

The type I error rate is higher than the training data. The type II error rate is also higher than the training data. The total prediction rate is 12.2%, which is 9.035% in the training data.

**Same for Misclassified and Classified Cases**

Now I can check how many misclassified cases are same using both methods. Besides, I can also check how many same cases are correctly specified using both methods. Code is in the appendix. The result is in the following table:

|  |  |
| --- | --- |
| Same classified cases | Same misclassified cases |
| 1661 | 95 |

From the result, I can find that most of misclassified cases in KNN method are also misclassified in the classification tree model. Most of classified cases in classification tree model are also classified in the KNN method. So KNN model is better.

Comparing the result above, I find that the KNN method gives the lower prediction rate both in training data and test data. So I choose KNN method to predict the blind test data. In the KNN method, the k=3 and the method is Manhattan distance.

**Blind Data**

Using KNN method, the prediction result is saved in the **prediction.rda**.

**Reference:**

1. Class notes.
2. Discussion notes: week 6&7.

**APPENDIX:**

**R CODE:**

############# KNN model ###################################

## Fit models using training data.

##Step 1 : CALCULATE THE DISTANCE USING DIST FUNCTION AND NEAREST DISTANCE MATRIX

load("~/Desktop/sta 141 assignment 5/trainVariables.rda")

real\_class = as.matrix(trainVariables[,"isSpam"])

### use scale for all variables

data\_dist = scale(trainVariables[,-ncol(trainVariables)])

## euclidean distance

distance = as.matrix(dist(data\_dist,upper=FALSE,diag=FALSE))

##Manhattan distance

distance\_man = as.matrix(dist(data\_dist,method='manhattan', upper=T))

## minkowski distance

distance\_min = as.matrix(dist(data\_dist,method="minkowski",p=3))

## euclidean distance order matrix

nearest\_euc = apply(distance,2,order)

##Manhattan distance order matrix

nearest\_man = apply(distance\_man,2,order)

##minkowski distance order matrix

nearest\_min = apply(distance\_min,2,order)

##Step 2: CROSS VALIDATION FUNCTION----10 folds

## l indicates which group of 10 I want. data indicates the order matrix.

cross\_validation = function (data,l){

#Create 10 folds

folds = rep(1:10,length = nrow(data))

#Segement your data by fold using the which() function

Indexes <- which(folds==l,arr.ind=TRUE)

Indexes

}

##STEP 3: get the kth nearest neighbors.

## index indicates the indexes of the specific folder.

## k indicates the how many points we want to choose to calculate nearest distance.

## data indicates the order matrix

KNN = function(index,k,data){

## generate folder index

test\_matrix = data[,index]

## In the order matrix, the first row always represents the distance to itself.

## So I need to delete the first row to get the nearest distance.

Knn = apply(test\_matrix,2,drop\_first,drop=index)

Knn[1:k,]

}

##Write a function to drop the first row

drop\_first = function(x,drop){

x[-match(drop,x)]

}

##STEP 4: KNN prediction function.

##For this function, I need to use the last KNN function's result to calculate

##the prediction for the specific KNN result.

## real\_class indicates the true class for each email.

## index indicates the indexes of the specific folder.

## k indicates the how many points we want to choose to calculate nearest distance.

## data indicates the order matrix

KNN\_predict = function(data,real\_class,k,index){

## matrix to store each column's probability to be a SPAM

percent = c()

knn = matrix(KNN(index,k,data),nrow = k)

## write a loop to get each row's prediction

for(i in 1:ncol(knn)){

percent[i] = sum(real\_class[knn[,i],]) / k

}

## if percent >=0.5, we predict it as a SPAM

predict = (percent >= 0.5)

## consider the case that probability == 0.5

predict[percent==0.5] = rbinom(n=1,size=1, prob=0.5)

predict

}

##STEP 5: KNN prediction check function.

##We can get the type I and type II errors from this functions.

##This function will return the confusion matrix.

##KNN\_predict is the prediction calculated in step 4.

KNN\_error = function(KNN\_predict,real\_class,index){

identity = real\_class[index]

##index to check the difference between true class and prediction

value = KNN\_predict == real\_class[index]

## correct prediction for spam

correct\_spam = table(value[identity==TRUE])[2]

## Type 1 error: real:spam & prediction: ham

TYPE1 = table(value[identity==TRUE])[1]

## correct prediction for Ham

correct\_ham = table(value[identity==FALSE])[2]

## Type 2 error: real:ham & prediction: spam

TYPE2 = table(value[identity==FALSE])[1]

## confusion matrix

ERROR = matrix(c(correct\_spam,TYPE2,TYPE1,correct\_ham),nrow=2,ncol=2,byrow=TRUE)

colnames(ERROR) = c("TRUE","FALSE")

rownames(ERROR) = c("TRUE","FALSE")

## return the confusion matrix

ERROR

}

##STEP 6: For a given k , apply step 3-5 for all 10 folders.

##This function will generate the confusion matrix for a given k.

##I will let the function return type I error rate, type II error rate and the total error rate.

KNN\_given\_k = function(k,data){

## initiate the confusion matrix

confusion = matrix(c(0,0,0,0),nrow=2,ncol=2)

colnames(confusion) = c("TRUE","FALSE")

rownames(confusion) = c("TRUE","FALSE")

## 10 folds

for(i in 1:10){

## store the index for ith test group

group = cross\_validation(data,i)

knn\_predict\_mid = KNN\_predict(data,real\_class,k,group)

confusion = confusion + KNN\_error(knn\_predict\_mid,real\_class,group)

}

## return type I error, type II error and total error rate

c(confusion[2,1] / (confusion[1,1] + confusion[2,1] ),

confusion[1,2] / (confusion[1,2] +confusion[2,2]) ,

(confusion[2,1] + confusion[1,2]) /sum(confusion))

}

##STEP 7: Given different k(1-30), get the result of error rate using three kinds of distances.

## calculate error rate for euclidean distance

ERROR\_RATE\_euc = sapply(1:30,KNN\_given\_k,nearest\_euc)

## calculate error rate for manhattan distance

ERROR\_RATE\_man = sapply(1:30,KNN\_given\_k,nearest\_man)

## calculate error rate for minkowski distance(p=3)

ERROR\_RATE\_min = sapply(1:30,KNN\_given\_k,nearest\_min)

par(mfrow=c(1,1))

## plot error rates on one graph

plot(ERROR\_RATE\_euc[3,],col="red",xlab="different K",

ylab="ERROR RATE for different distance",ylim=c(0,0.15))

lines(ERROR\_RATE\_euc[3,],col="red")

points(ERROR\_RATE\_man[3,],col="blue")

lines(ERROR\_RATE\_man[3,],col="blue")

points(ERROR\_RATE\_min[3,],col="yellow")

lines(ERROR\_RATE\_min[3,],col="yellow")

legend(x="topright",c("ERROR\_RATE\_euc","ERROR\_RATE\_man","ERROR\_RATE\_min"),

col=c("red","blue","yellow"),lwd=c(1,1,1),cex=0.5)

## Best model: k=3; method=manhattan

## get the error rate of best model

KNN\_given\_k(3,nearest\_man)

##check the misclassified cases

## This function returns the indexes of all the misclassified cases

mis\_indexes = function(k,real\_class,data){

mis\_index = c()

for(i in 1:10){

group = cross\_validation(data,i)

knn\_predict\_mid = KNN\_predict(data,real\_class,k,group)

identity = real\_class[group]

value = knn\_predict\_mid == real\_class[group]

mis\_index = c(mis\_index, group[which(value==FALSE)])

}

mis\_index

}

## The misclassified cases indexes in the training data:

mis = mis\_indexes(3,real\_class,nearest\_man)

## misclassified data

mis\_data = trainVariables[mis,]

## correctly classified data

cor\_data = trainVariables[-mis,]

## draw plots for misclassified cases

# all non-numeric variables

nums = sapply(trainVariables, is.numeric)

mis\_log\_name = lapply(mis\_data[!nums], table, mis\_data$isSpam)

cor\_log\_name = lapply(cor\_data[!nums], table, cor\_data$isSpam)

## logical cases

par(mfrow = c(1, 2))

for(i in 1:(30 - sum(nums) - 1)){

mosaicplot(mis\_log\_name[[i]], main = "Comparison for Misclassified Cases",

xlab = names(mis\_log\_name)[i], ylab = "isSpam",shade = TRUE)

mosaicplot(cor\_log\_name[[i]], main = "Comparison for Correct Cases",

xlab = names(cor\_log\_name)[i], ylab = "isSpam",shade = TRUE)

}

## numeric cases

mis\_num\_name = lapply(1:sum(nums), function(i) by(mis\_data[nums][, i],

mis\_data$isSpam, mean, na.rm = TRUE))

cor\_num\_name = lapply(1:sum(nums), function(i) by(cor\_data[nums][, i], cor\_data$isSpam, mean, na.rm = TRUE))

name = names(which(nums=="TRUE"))

for(i in 1:sum(nums)){

barplot(mis\_num\_name[[i]], main = "Misclassified Cases",

xlab = "isSpam", ylab = name[i], col = rainbow(2))

barplot(cor\_num\_name[[i]], main = "Comparison for Correct Cases",

xlab = "isSpam", ylab = name[i], col = rainbow(2))

}

### TEST DATA PART

load("~/Desktop/sta 141 assignment 5/testData.rda")

## scale test data

scale\_test = scale(testVariables[,-30])

## scale version of training data

data\_dist

## compute the distance

distance\_man\_new = as.matrix(dist(rbind(scale\_test,data\_dist),method='manhattan', upper=T))

## drop first 2000 rows since I need to use training data to predict

distance\_man\_new = distance\_man\_new[-(1:2000),]

## get the order matrix

nearest\_man\_new = apply(distance\_man\_new,2,order)

## The indexes for test data are from 1-2000

new\_index = c(1:2000)

## get the prediction of test data

new\_predict = KNN\_predict(nearest\_man\_new,real\_class,3,new\_index)

## get the confusion matrix for test data

new\_confusion = KNN\_error(new\_predict,real\_class\_new,new\_index)

new\_confusion

## For the test data, I can look through the misclassified cases.

check = new\_predict==real\_class\_new[1:2000,]

##Index for misclassified cases in KNN--test data

mis\_k7 = which(check==FALSE)

##Index for classified cases in KNN--test data

cor\_k7 = which(check==TRUE)

### Using Blind data to predict the SPAM or HAM.

load("~/Desktop/sta 141 assignment 5/blindTestData.rda")

scale\_blind = scale(blindTestVariables)

distance\_man\_blind = as.matrix(dist(rbind(scale\_blind,data\_dist),method='manhattan', upper=T))

## The first 808 obs are blind, which we need to predict.

distance\_man\_blind = distance\_man\_blind[-(1:808),]

nearest\_man\_blind = apply(distance\_man\_blind,2,order)

## The indexes for blind data are from 1-808

new\_index\_blind = c(1:808)

new\_predict\_blind = KNN\_predict(nearest\_man\_blind,real\_class,7,new\_index\_blind)

head(new\_predict\_blind)

new\_predict\_blind

## save into a .rda file

save(new\_predict\_blind,file="prediction.rda")

############################ Classification TREE################################

### TRAINING DATA PART

library(rpart)

log\_Vars = c()

## get all the logical variables

for (i in 1:ncol(trainVariables)) {

if (mode(trainVariables[,i]) == "logical") log\_Vars = c(log\_Vars,i)}

der = trainVariables

## factor all the logical variables.

for (i in log\_Vars) {der[,i] = as.factor(trainVariables[,i])}

par(mfrow=c(1,1))

## fit the model

tree = rpart(isSpam~., data = der, method="class")

plot(tree,uniform=TRUE)

text(tree)

## predict each email of the training dataset.

predict = predict(tree,der[-30],type="class")

##Now I need to write a funtion to calculate the confusion matrix for the classification tree.

## predict indicates the prediction result of classification tree.

confusion\_matrix = function(predict,data){

trueclass = data[,"isSpam"]

## check the differences between prediction and true class of each email.

classify = trueclass == predict

## correct prediction for spam

correct\_spam1 = table(classify[trueclass==TRUE])[2]

## Type 1 error: real:spam & prediction: ham

TYPE1\_1 = table(classify[trueclass==TRUE])[1]

## correct prediction for Ham

correct\_ham1 = table(classify[trueclass==FALSE])[2]

## Type 2 error: real:ham & prediction: spam

TYPE2\_1 = table(classify[trueclass==FALSE])[1]

## confusion matrix

ERROR\_1 = matrix(c(correct\_spam1,TYPE2\_1,TYPE1\_1,correct\_ham1),nrow=2,ncol=2,byrow=TRUE)

colnames(ERROR\_1) = c("TRUE","FALSE")

rownames(ERROR\_1) = c("TRUE","FALSE")

ERROR\_1

}

##Using the confusion matrix, the type I error rate,

##type II error rate and total prediction error rate are :

## write a function to calculate the error rates in the confusion matrix

error\_rate = function(ERROR\_1){

c(ERROR\_1[2,1] / (ERROR\_1[1,1] + ERROR\_1[2,1]),ERROR\_1[1,2] / (ERROR\_1[1,2] +ERROR\_1[2,2]) ,

(ERROR\_1[2,1] + ERROR\_1[1,2]) /sum(ERROR\_1))

}

##confusion matrix for training data

ERROR\_1 = confusion\_matrix(predict,trainVariables)

## error rates for training data

error\_rate(ERROR\_1)

### monotonic transformation: square

num\_Vars = c()

for (i in 1:ncol(trainVariables)) {

if (mode(trainVariables[,i]) != "logical") num\_Vars = c(num\_Vars,i)}

der\_square = trainVariables

for (i in num\_Vars) {der\_square[,i] = (trainVariables[,i])^2}

for (i in log\_Vars) {der\_square[,i] = as.factor(trainVariables[,i])}

tree\_square = rpart(isSpam~., data = der\_square, method="class")

predict\_square = predict(tree\_square,der\_square[-30],type="class")

error\_square = confusion\_matrix(predict\_square,trainVariables)

error\_square

### not monotonic transformation: sin

der\_sin = trainVariables

for (i in num\_Vars) {der\_sin[,i] = sin(trainVariables[,i])}

for (i in log\_Vars) {der\_sin[,i] = as.factor(trainVariables[,i])}

tree\_sin = rpart(isSpam~., data = der\_square, method="class")

predict\_sin = predict(tree\_sin,der\_sin[-30],type="class")

error\_sin = confusion\_matrix(predict\_sin,trainVariables)

error\_sin

### 1/ variables

der\_over = trainVariables

for (i in num\_Vars) {der\_over[,i] = 1/ (trainVariables[,i])}

for (i in log\_Vars) {der\_over[,i] = as.factor(trainVariables[,i])}

tree\_over = rpart(isSpam~., data = der\_over, method="class")

predict\_over = predict(tree\_over,der\_over[-30],type="class")

error\_over = confusion\_matrix(predict\_over,trainVariables)

error\_over

error\_rate(error\_over)

##Now I can see some characteristics for these misclassified cases in the training data set.

check0 = trainVariables[,"isSpam"] == predict

## index for misclassified obs

mis\_c = which(check0==FALSE)

## misclassified data

mis\_cla = trainVariables[mis\_c,]

## correctly classified data

cor\_cla = trainVariables[-mis\_c,]

## draw plot for misclassified cases

# all non-numeric variables

nums = sapply(trainVariables, is.numeric)

mis\_log\_tree = lapply(mis\_cla[!nums], table, mis\_cla$isSpam)

cor\_log\_tree = lapply(cor\_cla[!nums], table, cor\_cla$isSpam)

## logical cases

par(mfrow = c(1, 2))

for(i in 1:(30 - sum(nums) - 1)){

mosaicplot(mis\_log\_tree[[i]], main = "Comparison for Misclassified Cases",

xlab = names(mis\_log\_tree)[i], ylab = "isSpam",shade = TRUE)

mosaicplot(cor\_log\_tree[[i]], main = "Comparison for Correct Cases",

xlab = names(cor\_log\_tree)[i], ylab = "isSpam",shade = TRUE)

}

## numeric cases

mis\_num\_tree = lapply(1:sum(nums), function(i) by(mis\_cla[nums][, i],

mis\_cla$isSpam, mean, na.rm = TRUE))

cor\_num\_tree = lapply(1:sum(nums), function(i) by(cor\_cla[nums][, i],

cor\_cla$isSpam, mean, na.rm = TRUE))

name = names(which(nums=="TRUE"))

# compare the two different cases

for(i in 1:sum(nums)){

barplot(mis\_num\_tree[[i]], main = "Misclassified Cases",

xlab = "isSpam", ylab = name[i], col = rainbow(2))

barplot(cor\_num\_tree[[i]], main = "Comparison for Correct Cases",

xlab = "isSpam", ylab = name[i], col = rainbow(2))

}

### TEST DATA PART

log\_Var\_test = c()

## get all the logical variables

for (i in 1:ncol(testVariables)) {

if (mode(testVariables[,i]) == "logical") log\_Var\_test = c(log\_Var\_test,i)}

ders = testVariables

## factor all the logical variables.

for (i in log\_Var\_test) {ders[,i] = as.factor(testVariables[,i])}

## using the results in the training data, calculate the prediction for test data.

predict\_test = predict(tree,newdata=ders[-30],method="class")

##The confusion matrix for test data is:

logical = rep(0,2000)

predict\_test = cbind(predict\_test,logical)

## generate a new column to indicate if this observation is SPAM or HAM.

predict\_test[,3] = predict\_test[,1] < predict\_test[,2]

## confusion matrix for test data

error\_test = confusion\_matrix(predict\_test[,3],testVariables)

error\_test

## error rate for test data

error\_rate(error\_test)

##Now Let's check the misclassified cases.

##I will compare the reult with the misclassified cases using KNN.

## generate the logical matrix to check the difference between true class and prediction

check1 = testVariables[,"isSpam"] == predict\_test[,3]

## Index for misclassified cases in classification tree

mis\_class = which(check1==FALSE)

## Index for classified cases in classification tree

corr\_class = which(check1==TRUE)

```

##Now I can check if how many misclassified cases are same using both methods.Besides,

##I can also check how many same cases are correclty specified using both methods.

## check how many common cases are correclty classified.

common\_cor = Reduce(intersect,list(corr\_class,cor\_k7))

length(common\_cor)

## check how many common cases are misclassified.

common\_mis = Reduce(intersect,list(mis\_class,mis\_k7))

length(common\_mis)