

**Network Intrusion Case study -Decision Trees and Random Forest**

* **Apeksha Agrawal**

**BA-COE**

Contents

[Business Use Case/Problem Statement: 3](#_Toc20233995)

[Understanding the Data: 3](#_Toc20233996)

[Solution: 4](#_Toc20233997)

[Use Case Objectives: 4](#_Toc20233998)

[Tools & Technologies: 5](#_Toc20233999)

[Process followed: 5](#_Toc20234000)

[Detail steps with code: 6](#_Toc20234002)

## **Business Use Case/Problem Statement:**

**Background**

The dataset to be audited was provided which consists of a wide variety of intrusions simulated in a military network environment. It created an environment to acquire raw TCP/IP dump data for a network by simulating a typical US Air Force LAN. The LAN was focused like a real environment and blasted with multiple attacks. A connection is a sequence of TCP packets starting and ending at some time duration between which data flows to and from a source IP address to a target IP address under some well-defined protocol. Also, each connection is labelled as either normal or as an attack with exactly one specific attack type. Each connection record consists of about 100 bytes.

**Problem statement** – To identify if the new connection setup is normal or anomalous

## **Understanding the Data:**

For each TCP/IP connection, 41 quantitative and qualitative features are obtained from normal and attack data (3 qualitative and 38 quantitative features) .The class variable has two categories:

* Normal
* Anomalous

Data basically represents the packet data for a time duration of 2 seconds.

1-9 Columns: basic features of packet (type 1)

10-22 columns: employ the content features (type 2)

23-31 columns: employ the traffic features with 2 seconds of time window (type 4)

32-41 columns: employ the host based features

C: Continuous data

D: Discrete data

## **Solution:**

A word document containing basic guidelines to follow was provided along with case study to find optimal solution. While the solution covers all the steps mentioned, it has added important steps that were used for Exploratory Data Analysis, Feature Engineering- Creation of new Features and its selection, Model iterations etc. Please find attached the word Document.



## **Use Case Objectives:**

The objectives of this use case are to implement: -

* **Exploratory Data Analysis**-
  + Understanding data structure
  + Identifying and treating missing values
  + Checking the variance and distribution of continuous variables
  + Checking the frequency of categorical variables
  + Finding outliers within the variables and treating them
* **Feature Engineering- Feature Creation and Selection**
  + Feature Creation
  + Feature Selection process
    1. Fitter Method-
       - Using correlation and p values
         * Removing zero variance numeric columns and single level categorical column
         * Removing Effect related columns columns
         * Retaining columns with high correlation with dependent variable and are significant contributor
         * Removing independent variables that have high correlation within themselves and retaining one with high correlation with dependent variable
       - Searching for automated packages to perform it
    2. Wrapper Method- using model fitting method to select important features
       - Boruta Method
       - RFE Method
* Model Building and iterations to improve
  + Decision tree classifier
    1. Building base tree
    2. Model improvement techniques
       - Pre Pruning
       - Post Pruning
  + Random tree Classifier
    1. Building base tree
    2. Model improvement techniques
       - Hyper parameter tuning- Using caret package
       - Hyper parameter tuning- Using h2o package
       - Grid Search and Random Search CV methods

## **Tools & Technologies:**

* R studio



## **Process followed:**

Based on use case objectives, different iterations of feature engineering along with model building has been tried. The details for all the iterations done are provided below.

However only the steps for iteration with best model accuracy is explained in subsequent code-

1. Iteration 1- EDA + Decision tree -pre/post prune + Random forest - tuned using caret, h2o, grid search cv
2. Iteration 2- EDA + Feature Engineering- Fitter Method + Decision tree -pre/post prune + Random forest - tuned using caret, h2o, grid search cv
3. Iteration 3- EDA + Feature Engineering- Wrapper Method- Boruta + Decision tree -pre/post prune + Random forest - tuned using caret, h2o
4. Iteration 4- EDA + Feature Engineering- Wrapper Method- RFE + Decision tree -pre/post prune + Random forest - tuned using caret, h2o, grid search cv (Due to high run time and unforeseen issues this iteration could not be tested. But code is in working condition)

Please find attached excel with summary of all iterations using accuracy to test the model-

## 

**Iteration 2 gave best accuracy and is explained in following document**

***Note-*** All the iterations tried are well commented and hence self-explanatory. PFA codes

****

## **Details steps with code and output**

1. **Download the dataset csv**



1. **Setting working directory or creating Project**

Place the csv files in a folder and set this folder as the current working directory in Rstudio. Save the workspace as a project to avoid setting of working folder again and again.

1. **Reading data in R memory (training, validation and testing datasets)**

train\_data = read.csv("Network\_Intrusion\_Train\_data.csv",header = T )

train\_data1 = read.csv("Network\_Intrusion\_Train\_data.csv",header = T )

test\_data = read.csv("Network\_Intrusion\_Test\_data.csv",header = T )

validate\_data = read.csv("Validate\_data.csv",header = T )

1. **Checking the structure of the data**

> str(train\_data)

'data.frame': 25192 obs. of 42 variables:

$ duration : int 0 0 0 0 0 0 0 0 0 0 ...

$ protocol\_type : Factor w/ 3 levels "icmp","tcp","udp": 2 3 2 2 2 2 2 2 2 2 ...

$ service : Factor w/ 66 levels "auth","bgp","courier",..: 17 40 45 20 20 45 45 45 47 45 ...

$ flag : Factor w/ 11 levels "OTH","REJ","RSTO",..: 10 10 6 10 10 2 6 6 6 6 ...

$ src\_bytes : int 491 146 0 232 199 0 0 0 0 0 ...

$ dst\_bytes : int 0 0 0 8153 420 0 0 0 0 0 ...

$ land : int 0 0 0 0 0 0 0 0 0 0 ...

$ wrong\_fragment : int 0 0 0 0 0 0 0 0 0 0 ...

$ urgent : int 0 0 0 0 0 0 0 0 0 0 ...

$ hot : int 0 0 0 0 0 0 0 0 0 0 ...

$ num\_failed\_logins : int 0 0 0 0 0 0 0 0 0 0 ...

$ logged\_in : int 0 0 0 1 1 0 0 0 0 0 ...

$ num\_compromised : int 0 0 0 0 0 0 0 0 0 0 ...

$ root\_shell : int 0 0 0 0 0 0 0 0 0 0 ...

$ su\_attempted : int 0 0 0 0 0 0 0 0 0 0 ...

$ num\_root : int 0 0 0 0 0 0 0 0 0 0 ...

$ num\_file\_creations : int 0 0 0 0 0 0 0 0 0 0 ...

$ num\_shells : int 0 0 0 0 0 0 0 0 0 0 ...

$ num\_access\_files : int 0 0 0 0 0 0 0 0 0 0 ...

$ num\_outbound\_cmds : int 0 0 0 0 0 0 0 0 0 0 ...

$ is\_host\_login : int 0 0 0 0 0 0 0 0 0 0 ...

$ is\_guest\_login : int 0 0 0 0 0 0 0 0 0 0 ...

$ count : int 2 13 123 5 30 121 166 117 270 133 ...

$ srv\_count : int 2 1 6 5 32 19 9 16 23 8 ...

$ serror\_rate : num 0 0 1 0.2 0 0 1 1 1 1 ...

$ srv\_serror\_rate : num 0 0 1 0.2 0 0 1 1 1 1 ...

$ rerror\_rate : num 0 0 0 0 0 1 0 0 0 0 ...

$ srv\_rerror\_rate : num 0 0 0 0 0 1 0 0 0 0 ...

$ same\_srv\_rate : num 1 0.08 0.05 1 1 0.16 0.05 0.14 0.09 0.06 ...

$ diff\_srv\_rate : num 0 0.15 0.07 0 0 0.06 0.06 0.06 0.05 0.06 ...

$ srv\_diff\_host\_rate : num 0 0 0 0 0.09 0 0 0 0 0 ...

$ dst\_host\_count : int 150 255 255 30 255 255 255 255 255 255 ...

$ dst\_host\_srv\_count : int 25 1 26 255 255 19 9 15 23 13 ...

$ dst\_host\_same\_srv\_rate : num 0.17 0 0.1 1 1 0.07 0.04 0.06 0.09 0.05 ...

$ dst\_host\_diff\_srv\_rate : num 0.03 0.6 0.05 0 0 0.07 0.05 0.07 0.05 0.06 ...

$ dst\_host\_same\_src\_port\_rate: num 0.17 0.88 0 0.03 0 0 0 0 0 0 ...

$ dst\_host\_srv\_diff\_host\_rate: num 0 0 0 0.04 0 0 0 0 0 0 ...

$ dst\_host\_serror\_rate : num 0 0 1 0.03 0 0 1 1 1 1 ...

$ dst\_host\_srv\_serror\_rate : num 0 0 1 0.01 0 0 1 1 1 1 ...

$ dst\_host\_rerror\_rate : num 0.05 0 0 0 0 1 0 0 0 0 ...

$ dst\_host\_srv\_rerror\_rate : num 0 0 0 0.01 0 1 0 0 0 0 ...

$ class : Factor w/ 2 levels "anomaly","normal": 2 2 1 2 2 1 1 1 1 1 ...

summary(test\_data)

summary(validate\_data**)**

1. **If data contains NA, then perform imputation.**

**#a. For categorical data – replace NA with mode**

**#b. For numeric data – replace NA with median**

**#Method 1**

train\_data\_NA = for (i in length(train\_data)){

ifelse(is.element('TRUE',is.na(train\_data[,i]))=='TRUE', print(colnames(train\_data)[i]) , NA)}

NULL

test\_data\_NA = for (i in length(test\_data)){

ifelse(is.element('TRUE',is.na(test\_data[,i]))=='TRUE', print(colnames(test\_data)[i]) , NA)}

NULL

validate\_data\_NA= for (i in length(validate\_data)){

ifelse(is.element('TRUE',is.na(validate\_data[,i]))=='TRUE', print(colnames(validate\_data)[i]) , NA)}

NULL

**#Method 2**

indx\_train <- apply(train\_data, 2, function(x) any(is.na(x)))

colnames(indx\_train)

NULL

indx\_test <- apply(test\_data, 2, function(x) any(is.na(x)))

colnames(indx\_test)

NULL

indx\_validate <- apply(validate\_data, 2, function(x) any(is.na(x)))

colnames(indx\_validate)

NULL

**#Method 3**

**##visualize the patterns of NAs**

#install.packages("mice")

library(mice)

aqty2=train\_data

md.pattern(aqty2)

#111 observations with no values

#install.packages("VIM")

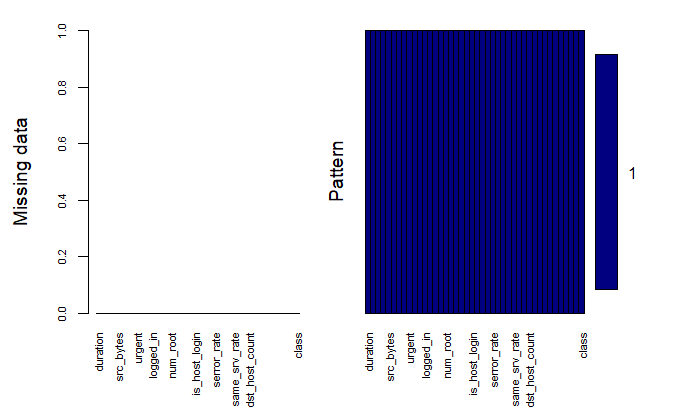
library(VIM) #visualize the pattern of NAs

mp <- aggr(aqty2, col=c('navyblue','yellow'),

numbers=TRUE, sortVars=TRUE,

labels=names(aqty2), cex.axis=.7,

gap=3, ylab=c("Missing data","Pattern"))



**#checking if any na**

summary(train\_data)

summary(test\_data)

summary(validate\_data)

**#creating a vector for character variables**

vec= vector()

for(i in 1:ncol(train\_data)){

if(!is.numeric(train\_data[,i]) & class(train\_data[,i])=="factor" ){

vec[i]= colnames(train\_data)[i]

}else{print("numeric")}}

vec <- vec[!is.na(vec)]

vec= as.vector(vec)

class(vec)

**#replacing numerical with median**

**#method 1 for numeric**

impute.med <- function(x) {

z <- median(x, na.rm = TRUE)

x[is.na(x)] <- z

return(x)

}

train\_data\_numeric= train\_data[ , !(names(train\_data) %in% vec)]

train\_data\_numeric <- sapply(train\_data\_numeric, function(x){

if(is.numeric(x) & any(is.na(x))){

impute.med(x)

} else {

x

}

}

)

train\_data\_numeric= as.data.frame(train\_data\_numeric)

**#Method 2 for numeric**

train\_data\_numeric= train\_data[ , !(names(train\_data) %in% vec)]

summary(train\_data\_numeric)

library(plyr)

impute.med <- function(x) replace(x, is.na(x), median(x, na.rm = TRUE))

train\_data\_numeric <- sapply(train\_data\_numeric, function(x){

if(is.numeric(x)){

impute.med(x)

} else {

x

}

}

)

train\_data\_numeric= data.frame(train\_data\_numeric)

summary(train\_data\_numeric)

**#replacing categorical with mode**

train\_data\_factor= train\_data[ , (names(train\_data) %in% vec)]

impute.mode <- function (x, na.rm) {

xtab <- table(x)

xmode <- names(which(xtab == max(xtab)))

if (length(xmode) > 1)

xmode <- ">1 mode"

return(xmode) }

#class(train\_data\_factor$protocol\_type)

summary(train\_data\_factor)

train\_data\_factor <- sapply(train\_data\_factor, function(x){

if(!is.numeric(x) & class(x)=="factor" & any(is.na(x))){

impute.mode(x, na.rm='TRUE')

} else {

x

}

}

)

summary(train\_data\_factor)

train\_data\_factor= as.data.frame(train\_data\_factor)

train\_data= cbind(train\_data\_factor, train\_data\_numeric)

summary(train\_data)

#install.packages("dplyr")

library(dplyr)

train\_data= as.data.frame(train\_data)

**#Repeating same set of steps as above for validate and test dataset**

**# treating NA in validate**

#Method 1

validate\_data\_NA = for (i in length(validate\_data)){

ifelse(is.element('TRUE',is.na(validate\_data[,i]))=='TRUE', print(colnames(validate\_data)[i]) , NA)}

test\_data\_NA = for (i in length(test\_data)){

ifelse(is.element('TRUE',is.na(test\_data[,i]))=='TRUE', print(colnames(test\_data)[i]) , NA)}

validate\_data\_NA= for (i in length(validate\_data)){

ifelse(is.element('TRUE',is.na(validate\_data[,i]))=='TRUE', print(colnames(validate\_data)[i]) , NA)}

#Method 2

indx\_train <- apply(validate\_data, 2, function(x) any(is.na(x)))

colnames(indx\_train)

indx\_test <- apply(test\_data, 2, function(x) any(is.na(x)))

colnames(indx\_test)

indx\_validate <- apply(validate\_data, 2, function(x) any(is.na(x)))

colnames(indx\_validate)

**#**Method 3

##visualize the patterns of NAs

#install.packages("mice")

library(mice)

aqty2=validate\_data

md.pattern(aqty2)

#111 observations with no values

#install.packages("VIM")

library(VIM) #visualize the pattern of NAs

mp <- aggr(aqty2, col=c('navyblue','yellow'),

numbers=TRUE, sortVars=TRUE,

labels=names(aqty2), cex.axis=.7,

gap=3, ylab=c("Missing data","Pattern"))

#checking if any na

summary(validate\_data)

summary(test\_data)

summary(validate\_data)

#creating a vector for character variables

vec= vector()

for(i in 1:ncol(validate\_data)){

if(!is.numeric(validate\_data[,i]) & class(validate\_data[,i])=="factor" ){

vec[i]= colnames(validate\_data)[i]

}else{print("numeric")}}

vec <- vec[!is.na(vec)]

vec= as.vector(vec)

class(vec)

#replacing numerical with median

#method 1 for numeric

impute.med <- function(x) {

z <- median(x, na.rm = TRUE)

x[is.na(x)] <- z

return(x)

}

validate\_data\_numeric= validate\_data[ , !(names(validate\_data) %in% vec)]

validate\_data\_numeric <- sapply(validate\_data\_numeric, function(x){

if(is.numeric(x) & any(is.na(x))){

impute.med(x)

} else {

x

}

}

)

validate\_data\_numeric= as.data.frame(validate\_data\_numeric)

#Method 2 for numeric

validate\_data\_numeric= validate\_data[ , !(names(validate\_data) %in% vec)]

summary(validate\_data\_numeric)

library(plyr)

impute.med <- function(x) replace(x, is.na(x), median(x, na.rm = TRUE))

validate\_data\_numeric <- sapply(validate\_data\_numeric, function(x){

if(is.numeric(x)){

impute.med(x)

} else {

x

}

}

)

validate\_data\_numeric= data.frame(validate\_data\_numeric)

summary(validate\_data\_numeric)

#replacing categorical with mode

#Method 1

validate\_data\_factor= validate\_data[ , (names(validate\_data) %in% vec)]

impute.mode <- function (x, na.rm) {

xtab <- table(x)

xmode <- names(which(xtab == max(xtab)))

if (length(xmode) > 1)

xmode <- ">1 mode"

return(xmode) }

#class(validate\_data\_factor$protocol\_type)

summary(validate\_data\_factor)

validate\_data\_factor <- sapply(validate\_data\_factor, function(x){

if(!is.numeric(x) & class(x)=="factor" & any(is.na(x))){

impute.mode(x, na.rm='TRUE')

} else {

x

}

}

)

summary(validate\_data\_factor)

validate\_data\_factor= as.data.frame(validate\_data\_factor)

validate\_data= cbind(validate\_data\_factor, validate\_data\_numeric)

summary(validate\_data)

#install.packages("dplyr")

library(dplyr)

validate\_data= as.data.frame(validate\_data)

**#treating NA in test**

#Method 1

test\_data\_NA = for (i in length(test\_data)){

ifelse(is.element('TRUE',is.na(test\_data[,i]))=='TRUE', print(colnames(test\_data)[i]) , NA)}

test\_data\_NA = for (i in length(test\_data)){

ifelse(is.element('TRUE',is.na(test\_data[,i]))=='TRUE', print(colnames(test\_data)[i]) , NA)}

test\_data\_NA= for (i in length(test\_data)){

ifelse(is.element('TRUE',is.na(test\_data[,i]))=='TRUE', print(colnames(test\_data)[i]) , NA)}

#Method 2

indx\_train <- apply(test\_data, 2, function(x) any(is.na(x)))

colnames(indx\_train)

indx\_test <- apply(test\_data, 2, function(x) any(is.na(x)))

colnames(indx\_test)

indx\_validate <- apply(test\_data, 2, function(x) any(is.na(x)))

colnames(indx\_validate)

#method 3

##visualize the patterns of NAs

#install.packages("mice")

library(mice)

aqty2=test\_data

md.pattern(aqty2)

#111 observations with no values

#install.packages("VIM")

library(VIM) #visualize the pattern of NAs

mp <- aggr(aqty2, col=c('navyblue','yellow'),

numbers=TRUE, sortVars=TRUE,

labels=names(aqty2), cex.axis=.7,

gap=3, ylab=c("Missing data","Pattern"))

#checking if any na

summary(test\_data)

summary(test\_data)

summary(test\_data)

#creating a vector for character variables

vec= vector()

for(i in 1:ncol(test\_data)){

if(!is.numeric(test\_data[,i]) & class(test\_data[,i])=="factor" ){

vec[i]= colnames(test\_data)[i]

}else{print("numeric")}}

vec <- vec[!is.na(vec)]

vec= as.vector(vec)

class(vec)

#replacing numerical with median

#method 1 for numeric

impute.med <- function(x) {

z <- median(x, na.rm = TRUE)

x[is.na(x)] <- z

return(x)

}

test\_data\_numeric= test\_data[ , !(names(test\_data) %in% vec)]

test\_data\_numeric <- sapply(test\_data\_numeric, function(x){

if(is.numeric(x) & any(is.na(x))){

impute.med(x)

} else {

x

}

}

)

test\_data\_numeric= as.data.frame(test\_data\_numeric)

#Method 2 for numeric

test\_data\_numeric= test\_data[ , !(names(test\_data) %in% vec)]

summary(test\_data\_numeric)

library(plyr)

impute.med <- function(x) replace(x, is.na(x), median(x, na.rm = TRUE))

test\_data\_numeric <- sapply(test\_data\_numeric, function(x){

if(is.numeric(x)){

impute.med(x)

} else {

x

}

}

)

test\_data\_numeric= data.frame(test\_data\_numeric)

summary(test\_data\_numeric)

#replacing categorical with mode

#Method 1

test\_data\_factor= test\_data[ , (names(test\_data) %in% vec)]

impute.mode <- function (x, na.rm) {

xtab <- table(x)

xmode <- names(which(xtab == max(xtab)))

if (length(xmode) > 1)

xmode <- ">1 mode"

return(xmode) }

#class(test\_data\_factor$protocol\_type)

summary(test\_data\_factor)

test\_data\_factor <- sapply(test\_data\_factor, function(x){

if(!is.numeric(x) & class(x)=="factor" & any(is.na(x))){

impute.mode(x, na.rm='TRUE')

} else {

x

}

}

)

summary(test\_data\_factor)

test\_data\_factor= as.data.frame(test\_data\_factor)

test\_data= cbind(test\_data\_factor, test\_data\_numeric)

summary(test\_data)

#install.packages("dplyr")

library(dplyr)

test\_data= as.data.frame(test\_data)

colnames(train\_data)

**#---------------------------Feature Creation-------------------------------------------#**

Creating flags for all numeric variables as val > 0 then flag= 1 orelse if val= 0 then flag= 1. This is done as each numeric column has almost 10% outliers which are in very different range and cannot be treated as they carry information. Hence creating flgs for zero and non zero values.

**#creating flags for numeric variables**

for(i in 1:length(train\_data)){

if(is.numeric(train\_data[,i])==TRUE){

train\_data[,paste(colnames(train\_data)[i],"Flag",sep="\_")] = ifelse(train\_data[,i] > 0,1,0)

}else{print("NA")}}

#validate\_data

#creating flags for numeric variables

for(i in 1:length(validate\_data)){

if(is.numeric(validate\_data[,i])==TRUE){

validate\_data[,paste(colnames(validate\_data)[i],"Flag",sep="\_")] = ifelse(validate\_data[,i] > 0,1,0)

}else{print("NA")}}

#test\_data

#creating flags for numeric variables

for(i in 1:length(test\_data)){

if(is.numeric(test\_data[,i])==TRUE){

test\_data[,paste(colnames(test\_data)[i],"Flag",sep="\_")] = ifelse(test\_data[,i] > 0,1,0)

}else{print("NA")}}

colnames(train\_data)

**output-** colnames(train\_data)

[1] "protocol\_type" "service"

[3] "flag" "class"

[5] "duration" "src\_bytes"

[7] "dst\_bytes" "land"

[9] "wrong\_fragment" "urgent"

[11] "hot" "num\_failed\_logins"

[13] "logged\_in" "num\_compromised"

[15] "root\_shell" "su\_attempted"

[17] "num\_root" "num\_file\_creations"

[19] "num\_shells" "num\_access\_files"

[21] "num\_outbound\_cmds" "is\_host\_login"

[23] "is\_guest\_login" "count"

[25] "srv\_count" "serror\_rate"

[27] "srv\_serror\_rate" "rerror\_rate"

[29] "srv\_rerror\_rate" "same\_srv\_rate"

[31] "diff\_srv\_rate" "srv\_diff\_host\_rate"

[33] "dst\_host\_count" "dst\_host\_srv\_count"

[35] "dst\_host\_same\_srv\_rate" "dst\_host\_diff\_srv\_rate"

[37] "dst\_host\_same\_src\_port\_rate" "dst\_host\_srv\_diff\_host\_rate"

[39] "dst\_host\_serror\_rate" "dst\_host\_srv\_serror\_rate"

[41] "dst\_host\_rerror\_rate" "dst\_host\_srv\_rerror\_rate"

[43] "duration\_Flag" "src\_bytes\_Flag"

[45] "dst\_bytes\_Flag" "land\_Flag"

[47] "wrong\_fragment\_Flag" "urgent\_Flag"

[49] "hot\_Flag" "num\_failed\_logins\_Flag"

[51] "logged\_in\_Flag" "num\_compromised\_Flag"

[53] "root\_shell\_Flag" "su\_attempted\_Flag"

[55] "num\_root\_Flag" "num\_file\_creations\_Flag"

[57] "num\_shells\_Flag" "num\_access\_files\_Flag"

[59] "num\_outbound\_cmds\_Flag" "is\_host\_login\_Flag"

[61] "is\_guest\_login\_Flag" "count\_Flag"

[63] "srv\_count\_Flag" "serror\_rate\_Flag"

[65] "srv\_serror\_rate\_Flag" "rerror\_rate\_Flag"

[67] "srv\_rerror\_rate\_Flag" "same\_srv\_rate\_Flag"

[69] "diff\_srv\_rate\_Flag" "srv\_diff\_host\_rate\_Flag"

[71] "dst\_host\_count\_Flag" "dst\_host\_srv\_count\_Flag"

[73] "dst\_host\_same\_srv\_rate\_Flag" "dst\_host\_diff\_srv\_rate\_Flag"

[75] "dst\_host\_same\_src\_port\_rate\_Flag" "dst\_host\_srv\_diff\_host\_rate\_Flag"

[77] "dst\_host\_serror\_rate\_Flag" "dst\_host\_srv\_serror\_rate\_Flag"

[79] "dst\_host\_rerror\_rate\_Flag" "dst\_host\_srv\_rerror\_rate\_Flag"

**#-------------------------------- feature selection----------------------#**

**# checking for distinct values and the variance of columns with numeric data and frequency distribution for categorical**

**##examine data distribution**

**# removing columns with zero variation**

summary(train\_data)

vec1= vector()

for(i in 1:length(train\_data)){

print(paste(i,"column= ",names(train\_data)[i],sep=""))

if(is.numeric(train\_data[,i])){

print(paste("variance= ", var(train\_data[,i]),sep=""))

print(unique(train\_data[,i]))

#printing histogram and boxplot for numeric

hist(get(colnames(train\_data)[i],train\_data),main=paste("histogram",names(train\_data)[i], sep=" ") ) # distribution of a varaibles

boxplot(train\_data[,i],main=paste("boxplot",names(train\_data)[i], sep=" "),boxwex=0.9)

#removing 0 variance columns

if(var(train\_data[,i])==0){

print(paste("Remove column ", names(train\_data)[i], sep=""))

vec1[i]= colnames(train\_data)[i]

}else{print("Non zero variance")}

}else{ print(unique(train\_data[,i]))

print(table(train\_data[,i]), sep="")

#printing barplot for categorical

barplot(table(train\_data[,i]), main=paste("barplot",names(train\_data)[i],sep=" "))

#removing 0 variance columns

if(length(unique(train\_data[,i]))==1){

print(paste("Remove column", names(train\_data)[i], sep=""))

vec1[i]= colnames(train\_data)[i]

}else{print("Non zero variance")}

}

}

vec1 <- vec1[!is.na(vec1)]

vec1= as.vector(vec1)

class(vec1)

train\_data= train\_data[ , !(names(train\_data) %in% vec1)]

train\_data= as.data.frame(train\_data)

colnames(train\_data)

**Sample output**

[1] "1column= protocol\_type"

[1] tcp udp icmp

Levels: icmp tcp udp

icmp tcp udp

1655 20526 3011

[1] "Non zero variance"

[1] "3column= flag"

[1] SF S0 REJ RSTR SH RSTO S1 RSTOS0 S3 S2 OTH

Levels: OTH REJ RSTO RSTOS0 RSTR S0 S1 S2 S3 SF SH

OTH REJ RSTO RSTOS0 RSTR S0 S1 S2 S3 SF SH

5 2216 304 21 497 7009 88 21 15 14973 43

[1] "Non zero variance"

[1] "4column= class"

[1] normal anomaly

Levels: anomaly normal

anomaly normal

11743 13449

"8column= land"

[1] "variance= 7.93871310958176e-05"

[1] 0 1

[1] "Non zero variance"

[1] "9column= wrong\_fragment"

[1] "variance= 0.0677148528714738"

[1] 0 3 1

[1] "Non zero variance"

[1] "10column= urgent"

[1] "variance= 3.96951413147031e-05"

[1] 0 1

[1] "Non zero variance"

[1] "11column= hot"

[1] "variance= 4.64058525384324"

[1] 0 5 6 4 2 1 28 30 22 24 14 3 15 25 19 18 77 17 11 7 20 12

[1] "Non zero variance"

[1] "12column= num\_failed\_logins"

[1] "variance= 0.00206281109812301"

[1] 0 2 1 3 4

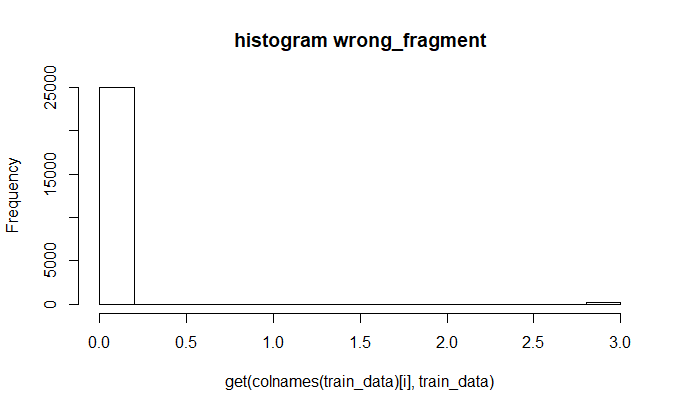
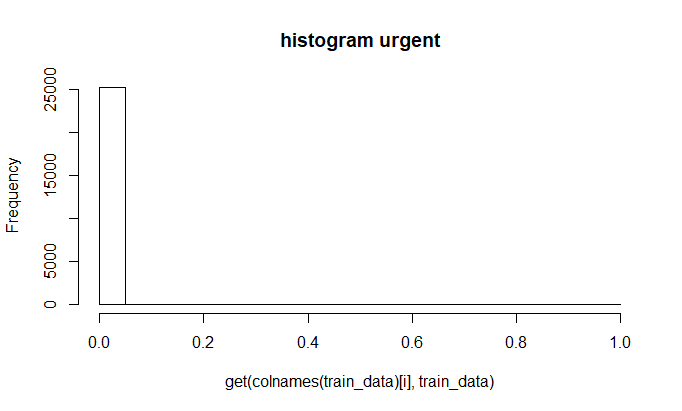
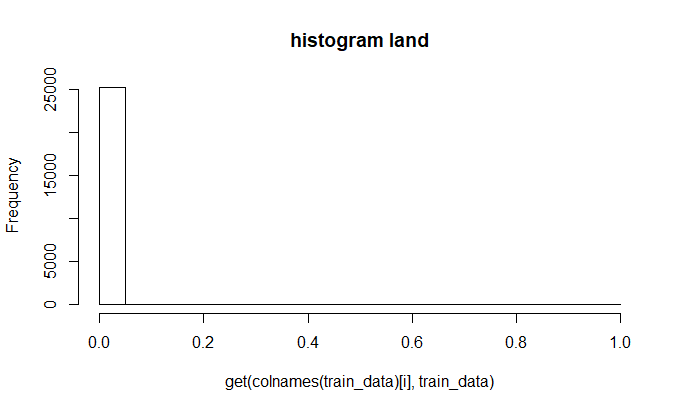
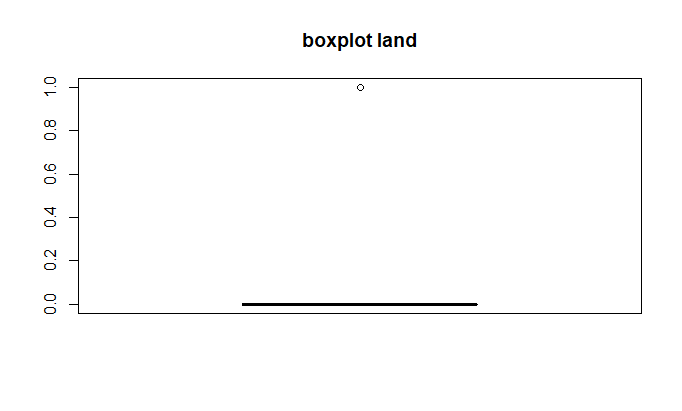
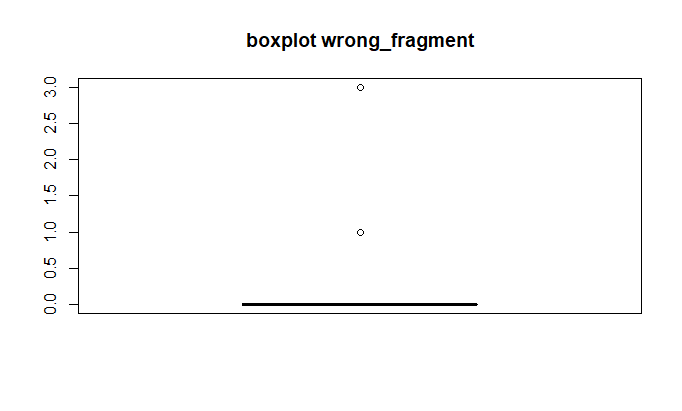
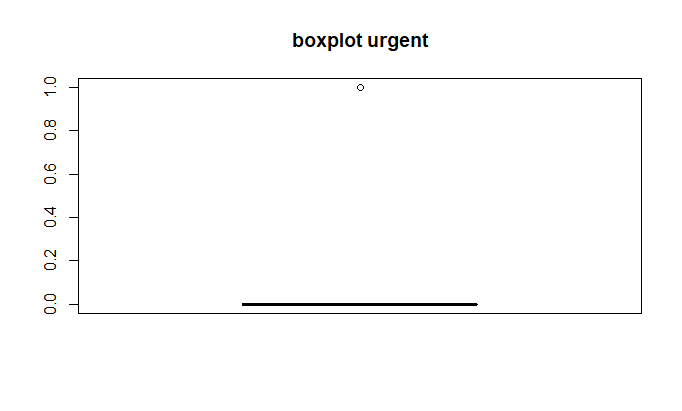
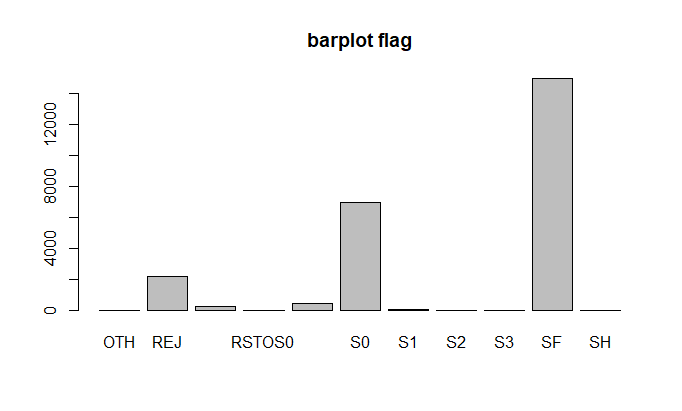
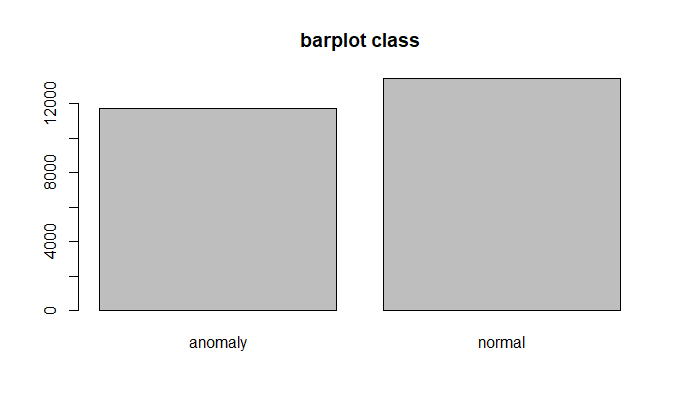
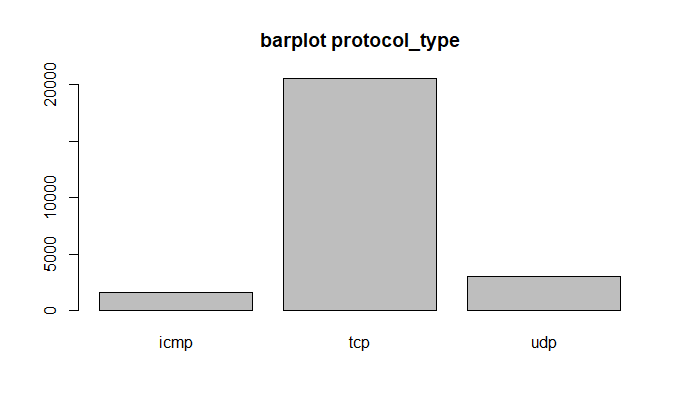
[1] "Non zero variance"

[1] "13column= logged\_in"

[1] "variance= 0.238935748726664"

[1] 0 1

[1] "Non zero variance"

****

**Columns removed=** *"num\_outbound\_cmds" , "is\_host\_login" , "num\_outbound\_cmds\_Flag" , "is\_host\_login\_Flag" , "count\_Flag" , "srv\_count\_Flag"*

**#Checking correlation of class variable(dependent) with individual**

**Independent variables**

df <- data.frame(matrix(ncol = 3, nrow = ncol(train\_data)))

df$X1= "aa"

df$X2= 0

df$X3= 0

for (i in 1:length(train\_data)) {

df[i,grep("X1", colnames(df))]= colnames(train\_data)[i]

if(is.numeric(train\_data[,i])== TRUE){

train\_data$class1= ifelse(train\_data$class=="anomaly", 1, 0)

#saving p value for numeric

df[i,grep("X2", colnames(df))]= cor.test(train\_data[,i], train\_data$class1)$p.value

#saving correlation value for numeric

df[i,grep("X3", colnames(df))]= cor.test(train\_data[,i], train\_data$class1)$estimate

}

else{

#chi-square test for categorical variables

# Create a table with the needed variables

a = table(train\_data[,i], train\_data$class)

#print(a)

# Perform the Chi-Square test.

df[i,grep("X2", colnames(df))]=(chisq.test(a))$p.value

df[i,grep("X3", colnames(df))]= NA

}

df$X4= ifelse(df$X2<0.05,"Significant", "Not Significant")

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **column** | **p value** | **co relation** | **sig/non sig** |
| **1** | protocol\_type | 0.00E+00 | *NA* | Significant |
| **2** | service | 0.00E+00 | *NA* | Significant |
| **3** | flag | 0.00E+00 | *NA* | Significant |
| **4** | class | 0.00E+00 | *NA* | Significant |
| **5** | duration | 6.27E-16 | 5.09E-02 | Significant |
| **6** | src\_bytes | 3.62E-01 | 5.74E-03 | Not Significant |
| **7** | dst\_bytes | 8.22E-02 | -1.09E-02 | Not Significant |
| **8** | land | 9.24E-01 | 6.05E-04 | Not Significant |
| **9** | wrong\_fragment | 2.13E-54 | 9.76E-02 | Significant |
| **10** | urgent | 2.85E-01 | 6.74E-03 | Not Significant |
| **11** | hot | 4.16E-02 | -1.28E-02 | Significant |
| **12** | num\_failed\_logins | 9.96E-01 | 2.77E-05 | Not Significant |
| **13** | logged\_in | 0.00E+00 | -6.88E-01 | Significant |
| **14** | num\_compromised | 3.12E-03 | -1.86E-02 | Significant |
| **15** | root\_shell | 3.19E-03 | -1.86E-02 | Significant |
| **16** | su\_attempted | 4.07E-05 | -2.59E-02 | Significant |
| **17** | num\_root | 1.72E-03 | -1.98E-02 | Significant |
| **18** | num\_file\_creations | 3.64E-03 | -1.83E-02 | Significant |
| **19** | num\_shells | 3.27E-02 | -1.35E-02 | Significant |
| **20** | num\_access\_files | 4.25E-09 | -3.70E-02 | Significant |
| **21** | is\_guest\_login | 8.34E-10 | -3.87E-02 | Significant |
| **22** | count | 0.00E+00 | 5.79E-01 | Significant |
| **23** | srv\_count | 7.07E-01 | 2.37E-03 | Not Significant |
| **24** | serror\_rate | 0.00E+00 | 6.50E-01 | Significant |
| **25** | srv\_serror\_rate | 0.00E+00 | 6.48E-01 | Significant |
| **26** | rerror\_rate | 0.00E+00 | 2.57E-01 | Significant |
| **27** | srv\_rerror\_rate | 0.00E+00 | 2.56E-01 | Significant |
| **28** | same\_srv\_rate | 0.00E+00 | -7.49E-01 | Significant |
| **29** | diff\_srv\_rate | 4.09E-211 | 1.94E-01 | Significant |
| **30** | srv\_diff\_host\_rate | 2.58E-82 | -1.21E-01 | Significant |
| **31** | dst\_host\_count | 0.00E+00 | 3.69E-01 | Significant |
| **32** | dst\_host\_srv\_count | 0.00E+00 | -7.19E-01 | Significant |
| **33** | dst\_host\_same\_srv\_rate | 0.00E+00 | -6.92E-01 | Significant |
| **34** | dst\_host\_diff\_srv\_rate | 7.905050e-322 | 2.38E-01 | Significant |
| **35** | dst\_host\_same\_src\_port\_rate | 1.75E-49 | 9.30E-02 | Significant |
| **36** | dst\_host\_srv\_diff\_host\_rate | 1.57E-23 | 6.29E-02 | Significant |
| **37** | dst\_host\_serror\_rate | 0.00E+00 | 6.51E-01 | Significant |
| **38** | dst\_host\_srv\_serror\_rate | 0.00E+00 | 6.54E-01 | Significant |
| **39** | dst\_host\_rerror\_rate | 0.00E+00 | 2.56E-01 | Significant |
| **40** | dst\_host\_srv\_rerror\_rate | 0.00E+00 | 2.57E-01 | Significant |
| **41** | duration\_Flag | 1.10E-129 | -1.52E-01 | Significant |
| **42** | src\_bytes\_Flag | 0.00E+00 | -7.49E-01 | Significant |
| **43** | dst\_bytes\_Flag | 0.00E+00 | -8.11E-01 | Significant |
| **44** | land\_Flag | 9.24E-01 | 6.05E-04 | Not Significant |
| **45** | wrong\_fragment\_Flag | 1.59E-58 | 1.01E-01 | Significant |
| **46** | urgent\_Flag | 2.85E-01 | 6.74E-03 | Not Significant |
| **47** | hot\_Flag | 3.46E-05 | 2.61E-02 | Significant |
| **48** | num\_failed\_logins\_Flag | 9.07E-01 | 7.34E-04 | Not Significant |
| **49** | logged\_in\_Flag | 0.00E+00 | -6.88E-01 | Significant |
| **50** | num\_compromised\_Flag | 1.48E-11 | 4.25E-02 | Significant |
| **51** | root\_shell\_Flag | 3.19E-03 | -1.86E-02 | Significant |
| **52** | su\_attempted\_Flag | 1.83E-05 | -2.70E-02 | Significant |
| **53** | num\_root\_Flag | 2.96E-24 | -6.40E-02 | Significant |
| **54** | num\_file\_creations\_Flag | 2.87E-07 | -3.23E-02 | Significant |
| **55** | num\_shells\_Flag | 3.27E-02 | -1.35E-02 | Significant |
| **56** | num\_access\_files\_Flag | 1.18E-13 | -4.67E-02 | Significant |
| **57** | is\_guest\_login\_Flag | 8.34E-10 | -3.87E-02 | Significant |
| **58** | serror\_rate\_Flag | 0.00E+00 | 6.42E-01 | Significant |
| **59** | srv\_serror\_rate\_Flag | 0.00E+00 | 6.12E-01 | Significant |
| **60** | rerror\_rate\_Flag | 0.00E+00 | 2.61E-01 | Significant |
| **61** | srv\_rerror\_rate\_Flag | 0.00E+00 | 2.43E-01 | Significant |
| **62** | same\_srv\_rate\_Flag | 3.39E-136 | -1.56E-01 | Significant |
| **63** | diff\_srv\_rate\_Flag | 0.00E+00 | 7.14E-01 | Significant |
| **64** | srv\_diff\_host\_rate\_Flag | 0.00E+00 | -3.51E-01 | Significant |
| **65** | dst\_host\_count\_Flag | 3.50E-01 | 5.89E-03 | Not Significant |
| **66** | dst\_host\_srv\_count\_Flag | 3.50E-01 | 5.89E-03 | Not Significant |
| **67** | dst\_host\_same\_srv\_rate\_Flag | 3.12E-177 | -1.77E-01 | Significant |
| **68** | dst\_host\_diff\_srv\_rate\_Flag | 0.00E+00 | 4.88E-01 | Significant |
| **69** | dst\_host\_same\_src\_port\_rate\_Flag | 0.00E+00 | -3.62E-01 | Significant |
| **70** | dst\_host\_srv\_diff\_host\_rate\_Flag | 0.00E+00 | -4.36E-01 | Significant |
| **71** | dst\_host\_serror\_rate\_Flag | 0.00E+00 | 5.91E-01 | Significant |
| **72** | dst\_host\_srv\_serror\_rate\_Flag | 0.00E+00 | 5.52E-01 | Significant |
| **73** | dst\_host\_rerror\_rate\_Flag | 1.37E-230 | 2.02E-01 | Significant |
| **74** | dst\_host\_srv\_rerror\_rate\_Flag | 2.09E-189 | 1.83E-01 | Significant |

**# removing non significant columns**

sig\_col= df[df$X4=="Significant",]

sig\_col\_vec= sig\_col$X1

train\_data= train\_data[ , (names(train\_data) %in% sig\_col\_vec)]

colnames(train\_data)

**output- columns retained**

[1] "protocol\_type" "service"

[3] "flag" "class"

[5] "duration" "wrong\_fragment"

[7] "hot" "logged\_in"

[9] "num\_compromised" "root\_shell"

[11] "su\_attempted" "num\_root"

[13] "num\_file\_creations" "num\_shells"

[15] "num\_access\_files" "is\_guest\_login"

[17] "count" "serror\_rate"

[19] "srv\_serror\_rate" "rerror\_rate"

[21] "srv\_rerror\_rate" "same\_srv\_rate"

[23] "diff\_srv\_rate" "srv\_diff\_host\_rate"

[25] "dst\_host\_count" "dst\_host\_srv\_count"

[27] "dst\_host\_same\_srv\_rate" "dst\_host\_diff\_srv\_rate"

[29] "dst\_host\_same\_src\_port\_rate" "dst\_host\_srv\_diff\_host\_rate"

[31] "dst\_host\_serror\_rate" "dst\_host\_srv\_serror\_rate"

[33] "dst\_host\_rerror\_rate" "dst\_host\_srv\_rerror\_rate"

[35] "duration\_Flag" "src\_bytes\_Flag"

[37] "dst\_bytes\_Flag" "wrong\_fragment\_Flag"

[39] "hot\_Flag" "logged\_in\_Flag"

[41] "num\_compromised\_Flag" "root\_shell\_Flag"

[43] "su\_attempted\_Flag" "num\_root\_Flag"

[45] "num\_file\_creations\_Flag" "num\_shells\_Flag"

[47] "num\_access\_files\_Flag" "is\_guest\_login\_Flag"

[49] "serror\_rate\_Flag" "srv\_serror\_rate\_Flag"

[51] "rerror\_rate\_Flag" "srv\_rerror\_rate\_Flag"

[53] "same\_srv\_rate\_Flag" "diff\_srv\_rate\_Flag"

[55] "srv\_diff\_host\_rate\_Flag" "dst\_host\_same\_srv\_rate\_Flag"

[57] "dst\_host\_diff\_srv\_rate\_Flag" "dst\_host\_same\_src\_port\_rate\_Flag"

[59] "dst\_host\_srv\_diff\_host\_rate\_Flag" "dst\_host\_serror\_rate\_Flag"

[61] "dst\_host\_srv\_serror\_rate\_Flag" "dst\_host\_rerror\_rate\_Flag"

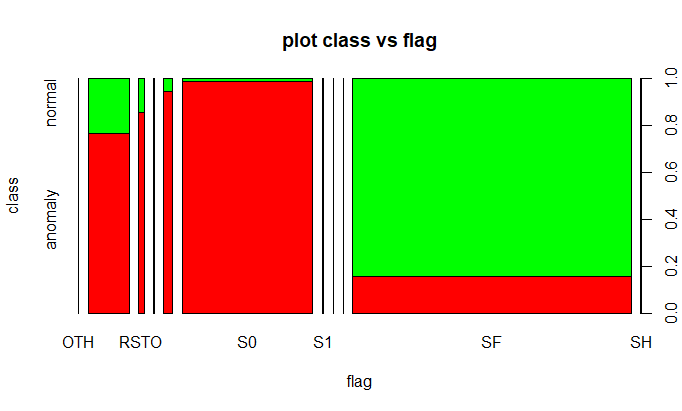
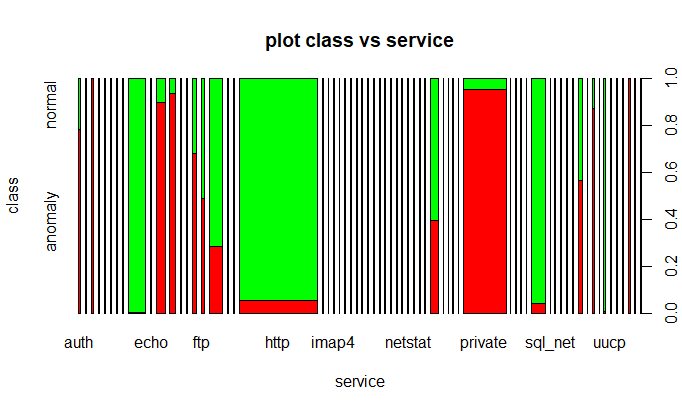
[63] "dst\_host\_srv\_rerror\_rate\_Flag"

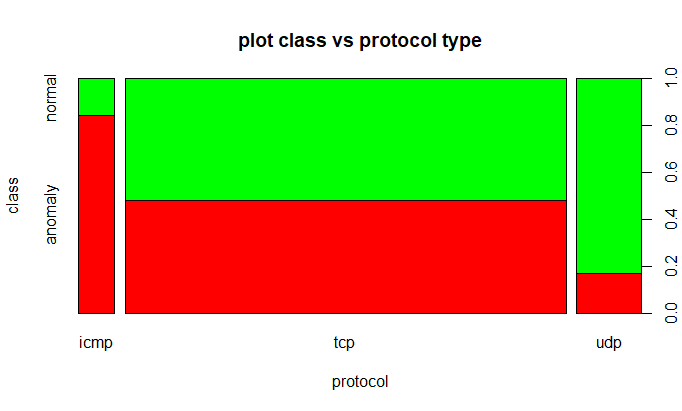
**#5) Different plots to identify relationship between variables for all the variables within independent variables**

#PLOTTING ONLY CATEGORICAL VARIABLES AGAINST CLASS

plot(train\_data$protocol\_type,train\_data$class, col=c("red","green"), main=("plot class vs protocol type"), xlab="protocol", ylab="class")

plot(train\_data$service,train\_data$class, col=c("red","green"), main=("plot class vs service"), xlab="service", ylab="class")

plot(train\_data$flag,train\_data$class, col=c("red","green"), main=("plot class vs flag"), xlab="flag", ylab="class")



These graphs show that there is relationship between class variable and all categorical variables

**#finding correlation between all independent variables along with p values of corelation**

#A t test is available to test the null hypothesis that the correlation coefficient is zero

#Note that the P value derived from the test provides no information on how strongly the 2 variables are related

#install.packages("Hmisc")

#the p-value is less than 0.05, there is strong evidence against the null hypothesis.

#As a result, reject the null hypothesis and accept the alternative hypothesis that there is co-relation between 2 variables

#Creating different vector for numeric and categorical variables

vec\_num=vector()

vec\_char=vector()

for (i in 1:length(train\_data)) {

print(colnames(train\_data)[i])

if(is.numeric(train\_data[,i])== TRUE){

vec\_num[i]= colnames(train\_data)[i]

}else{

vec\_char[i]= colnames(train\_data)[i]}

}

library("Hmisc")

# ++++++++++++++++++++++++++++

# flattenCorrMatrix

# ++++++++++++++++++++++++++++

# cormat : matrix of the correlation coefficients

# pmat : matrix of the correlation p-values

flattenCorrMatrix <- function(cormat, pmat) {

ut <- upper.tri(cormat)

data.frame(

row = rownames(cormat)[row(cormat)[ut]],

column = rownames(cormat)[col(cormat)[ut]],

cor =(cormat)[ut],

p = pmat[ut]

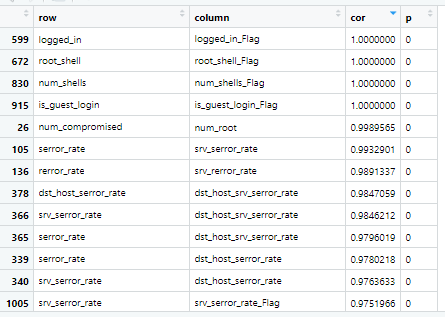
)

}

res3<-rcorr(as.matrix(train\_data[,names(train\_data) %in% vec\_num]))

res4 =flattenCorrMatrix(res3$r, res3$P)

**Sample output**



**# variables correlation>0.90 or correlation< (-.0.9) val pvalue <0.05 are having strong coreltion**

res5= subset(res4,(res4$cor>0.90 & res4$p<0.05))

names(res5)[names(res5)== "row"]= "col1"

names(res5)[names(res5)== "column"]= "col2"

**#checking correlation of both variables with the dependent variable and removing one with**

**low correlation value**

res6=data.frame()

res6= merge(res5, df, by.x = "col1", by.y = "X1", all.x = T )

names(res6)[names(res6)== "X3"]= "row\_cor"

res6= subset(res6, select = c("col1","col2","row\_cor" ))

res6= merge(res6, df, by.x = "col2", by.y = "X1", all.x = T )

names(res6)[names(res6)== "X3"]= "column\_cor"

res6= subset(res6, select = c("col1","col2","row\_cor","column\_cor" ))

rem\_var=vector()

res6$rem\_var= ifelse(res6$row\_cor> res6$column\_cor, as.character(res6$col2) , as.character(res6$col1) )

vec2=vector()

vec2= unique(res6$rem\_var)

train\_data= train\_data[ , !(names(train\_data) %in% vec2)]

colnames(train\_data)

[1] "protocol\_type" "service"

[3] "flag" "class"

[5] "duration" "hot"

[7] "num\_compromised" "su\_attempted"

[9] "num\_file\_creations" "num\_access\_files"

[11] "count" "same\_srv\_rate"

[13] "diff\_srv\_rate" "srv\_diff\_host\_rate"

[15] "dst\_host\_count" "dst\_host\_srv\_count"

[17] "dst\_host\_same\_srv\_rate" "dst\_host\_diff\_srv\_rate"

[19] "dst\_host\_same\_src\_port\_rate" "dst\_host\_srv\_diff\_host\_rate"

[21] "dst\_host\_srv\_serror\_rate" "duration\_Flag"

[23] "src\_bytes\_Flag" "dst\_bytes\_Flag"

[25] "wrong\_fragment\_Flag" "hot\_Flag"

[27] "logged\_in\_Flag" "num\_compromised\_Flag"

[29] "root\_shell\_Flag" "num\_root\_Flag"

[31] "num\_file\_creations\_Flag" "num\_shells\_Flag"

[33] "num\_access\_files\_Flag" "is\_guest\_login\_Flag"

[35] "rerror\_rate\_Flag" "same\_srv\_rate\_Flag"

[37] "diff\_srv\_rate\_Flag" "srv\_diff\_host\_rate\_Flag"

[39] "dst\_host\_same\_srv\_rate\_Flag" "dst\_host\_diff\_srv\_rate\_Flag"

[41] "dst\_host\_same\_src\_port\_rate\_Flag" "dst\_host\_srv\_diff\_host\_rate\_Flag"

[43] "dst\_host\_serror\_rate\_Flag" "dst\_host\_rerror\_rate\_Flag"

[45] "dst\_host\_srv\_rerror\_rate\_Flag"

**#---------------Making data ready to fit model--------------------------------#**

# handling dependent var

train\_data$class1= ifelse(train\_data$class=="anomaly", 1, 0)

train\_data$class= NULL

names(train\_data)[names(train\_data)== "class1"]= "y\_var"

validate\_data$class1= ifelse(validate\_data$class=="anomaly", 1, 0)

validate\_data$class= NULL

names(validate\_data)[names(validate\_data)== "class1"]= "y\_var"

validate\_data= subset(validate\_data,validate\_data$service !="tftp\_u" )

validate\_data= validate\_data[, names(validate\_data) %in% names(train\_data)]

test\_data= subset(test\_data,test\_data$service !="tftp\_u" )

test\_data= test\_data[, names(test\_data) %in% names(test\_data)]

**#--------------------------Building Classifier----Decision Tree------------------------#**

**# Iteration 2**

**# Fit Decision Tree model on training data**

**## iteration 2**

class(train\_data)

train\_data= as.data.frame(train\_data)

**#no need of feature scaling in decision trees as not ecudiliean distance algo**

**# Fitting Decision Tree Classification to the Training set**

**#install.packages('rpart')**

library(rpart)

classifier = rpart(formula = y\_var~.,

data = train\_data, method="class")

**Sample output-**

n= 25192

node), split, n, loss, yval, (yprob)

\* denotes terminal node

1) root 25192 11743 0 (0.53385996 0.46614004)

2) service=domain\_u,ftp,ftp\_data,http,IRC,ntp\_u,other,pop\_3,red\_i,smtp,urh\_i,urp\_i,X11 14149 1427 0 (0.89914482 0.10085518)

4) flag=REJ,S1,S2,S3,SF 13283 687 0 (0.94827976 0.05172024)

8) same\_srv\_rate>=0.075 12981 432 0 (0.96672059 0.03327941)

16) num\_compromised< 0.5 12806 264 0 (0.97938466 0.02061534) \*

17) num\_compromised>=0.5 175 7 1 (0.04000000 0.96000000) \*

9) same\_srv\_rate< 0.075 302 47 1 (0.15562914 0.84437086) \*

5) flag=OTH,RSTO,RSTOS0,RSTR,S0 866 126 1 (0.14549654 0.85450346) \*

3) service=auth,bgp,courier,csnet\_ns,ctf,daytime,discard,domain,echo,eco\_i,ecr\_i,efs,exec,finger,gopher,hostnames,http\_443,http\_8001,imap4,iso\_tsap,klogin,kshell,ldap,link,login,mtp,name,netbios\_dgm,netbios\_ns,netbios\_ssn,netstat,nnsp,nntp,pm\_dump,pop\_2,printer,private,remote\_job,rje,shell,sql\_net,ssh,sunrpc,supdup,systat,telnet,tim\_i,time,uucp,uucp\_path,vmnet,whois,Z39\_50 11043 727 1 (0.06583356 0.93416644)

6) dst\_bytes\_Flag>=0.5 586 51 0 (0.91296928 0.08703072) \*

7) dst\_bytes\_Flag< 0.5 10457 192 1 (0.01836091 0.98163909) \*

**# Predicting the Test set results**

y\_pred = predict(classifier, newdata = validate\_data[,!(names(validate\_data) %in% c("y\_var"))], type = 'class')

**# Making the Confusion Matrix to test accuracy**

cm = table(validate\_data$y\_var, y\_pred)

**Sample output-**

y\_pred

0 1

0 9398 312

1 4271 8562

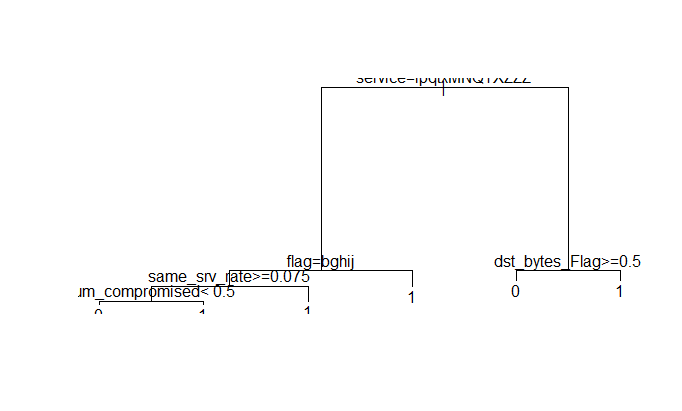
accuracy= (cm[1,1] + cm[2,2])/ sum(cm)

print(accuracy)

**#0.7966996**

plot(classifier)

text(classifier)



**# improve the visualization**

**#install.packages("rattle")**

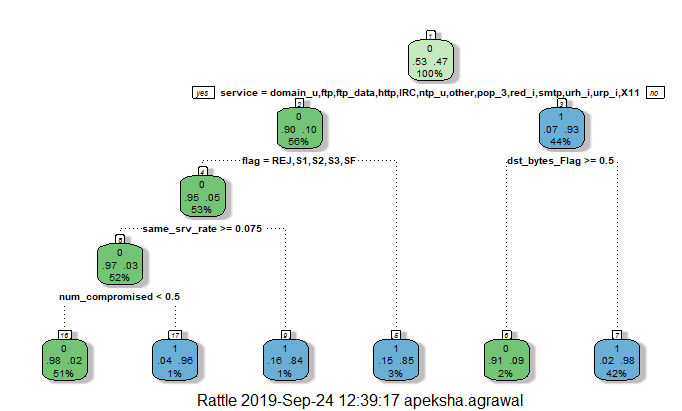
**#install.packages("https://cran.r-project.org/bin/windows/contrib/3.3/RGtk2\_2.20.31.zip", repos=NULL)**

library(rattle)

library(rpart.plot)

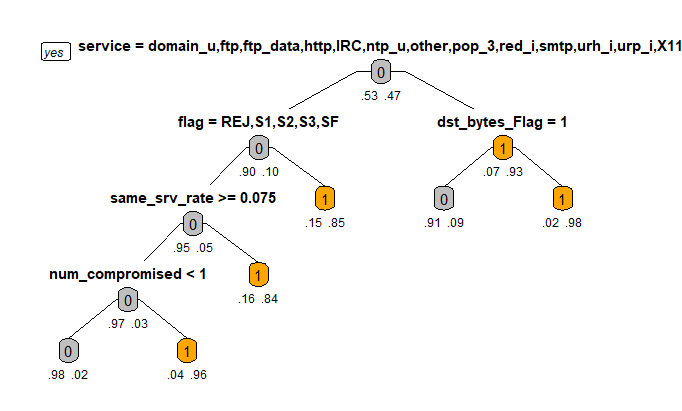
library(RColorBrewer)

fancyRpartPlot(classifier)



**#Method 1**

prp(classifier,box.col=c("Grey", "Orange")[classifier$frame$yval],varlen=0,faclen=0, type=1,extra=4,under=TRUE)



**# post pruning using cp- complexity parameter and the cross validation error values**

printcp(classifier)

Classification tree:

rpart(formula = y\_var ~ ., data = train\_data, method = "class")

Variables actually used in tree construction:

[1] dst\_bytes\_Flag flag num\_compromised same\_srv\_rate service

Root node error: 11743/25192 = 0.46614

n= 25192

CP nsplit rel error xerror xstd

1 0.816572 0 1.000000 1.000000 0.0067425

2 0.052286 1 0.183428 0.183428 0.0037795

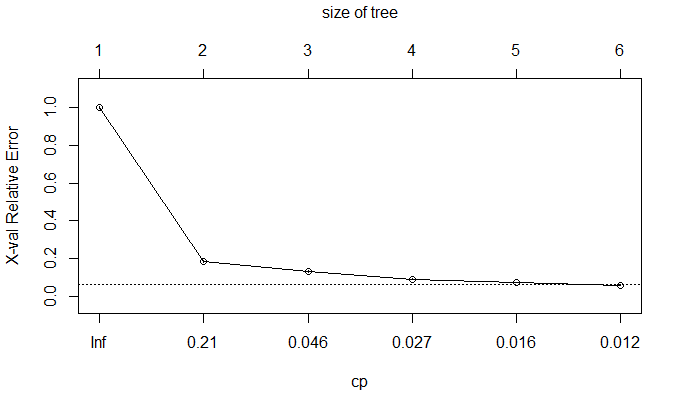
3 0.041216 2 0.131142 0.131312 0.0032400

4 0.017713 3 0.089926 0.090096 0.0027111

5 0.013710 4 0.072213 0.072554 0.0024433

6 0.010000 5 0.058503 0.058844 0.0022076

plotcp(classifier)



#select the one having the least cross-validated error- xerror and use it to prune the tree

#From the above mentioned list of cp values-

#Prune the tree to create an optimal decision tree :

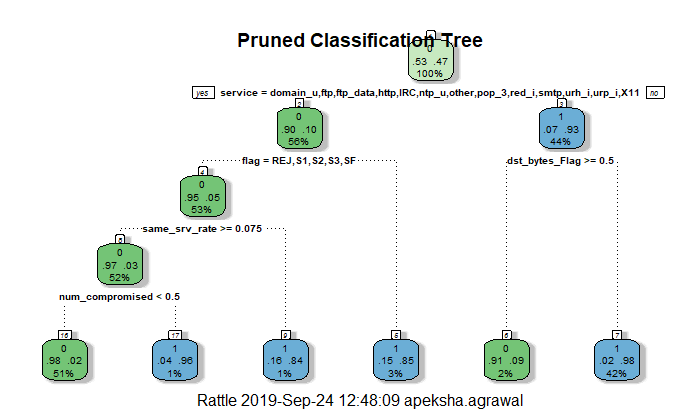
cp\_min= min(classifier$cptable[,"xerror"])

cp\_min

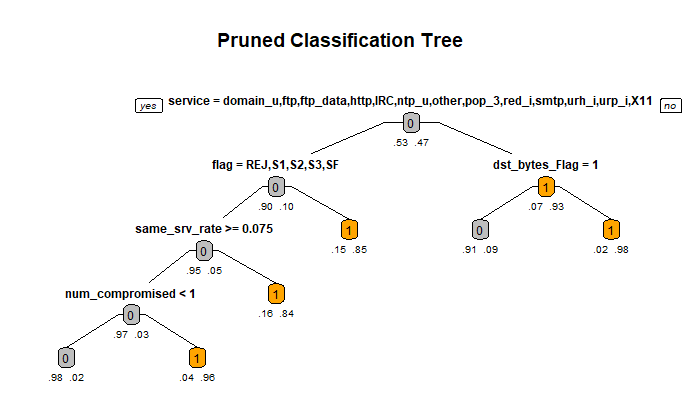
#0.05884357

ptree<- prune(classifier, cp= classifier$cptable[which.min(classifier$cptable[,"xerror"]),"CP"])

fancyRpartPlot(ptree, uniform=TRUE,main="Pruned Classification Tree")



prp(ptree,box.col=c("Grey", "Orange")[ptree$frame$yval],varlen=0,faclen=0, type=1,extra=4,under=TRUE, main="Pruned Classification Tree")



# Predicting the Test set results using pruned tree

y\_pred\_prune = predict(ptree, newdata = validate\_data[,!(names(validate\_data) %in% c("y\_var"))], type = 'class')

# Making the Confusion Matrix to test accuracy

cm = table(validate\_data$y\_var, y\_pred\_prune)

y\_pred\_prune

0 1

0 9398 312

1 4271 8562

accuracy= (cm[1,1] + cm[2,2])/ sum(cm)

print(accuracy)

#0.7966996

#prepruning

# Grow a tree with minsplit of 100 and max depth of 8

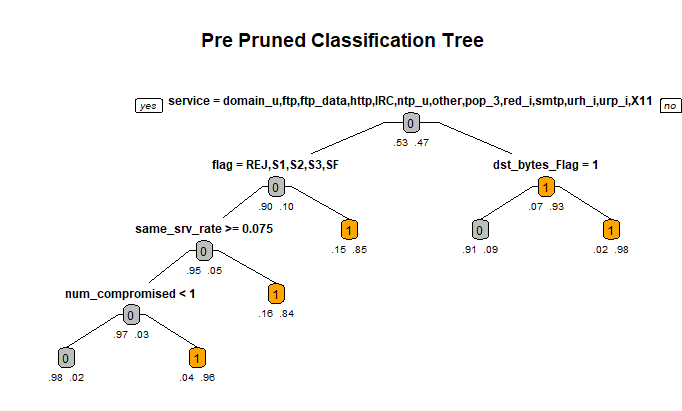
classifier <- rpart(formula = y\_var~.,

data = train\_data, method="class", control=rpart.control(cp = 0.01, maxdepth = 8,minsplit = 100))

prp(classifier,box.col=c("Grey", "Orange")[classifier$frame$yval],varlen=0,faclen=0, type=1,extra=4,under=TRUE, main="Pruned Classification Tree")

# Compute the accuracy of the pruned tree

y\_pred\_prune\_pre = predict(classifier, newdata = validate\_data[,!(names(validate\_data) %in% c("y\_var"))], type = 'class')



cm = table(validate\_data$y\_var, y\_pred\_prune\_pre)

y\_pred\_prune\_pre

0 1

0 9398 312

1 4271 8562

accuracy= (cm[1,1] + cm[2,2])/ sum(cm)

print(accuracy)

#0.8042408

**#---------------------------RANDOM FOREST------------------------------#**

Random forest with RandomForest package ( base and tuned) and H2O package (base

and tuned) are tried

#random forest

# first time H20 instalment

#Remove any previously installed packages for R

#if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }

#if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

#download packages that H2O depends on.

#pkgs <- c("RCurl","jsonlite")

#for (pkg in pkgs) {

# if (! (pkg %in% rownames(installed.packages()))) { install.packages(pkg) }

#}

#Download and install the latest H2O package for R.

# install.packages("h2o", type="source", repos=(c("http://h2o-release.s3.amazonaws.com/h2o/latest\_stable\_R")))

# install.packages("rsample")

# install.packages("pROC")

library(rsample) # data splitting

library(randomForest) # basic implementation

library(ranger) # a faster implementation of randomForest

library(caret) # an aggregator package for performing many machine learning model

library(h2o) # an extremely fast java-based platform

# Basic Implementation using randomForest package

# for reproduciblity

set.seed(123)

train\_data$service= as.numeric(train\_data$service)

validate\_data$service= as.numeric(validate\_data$service)

test\_data$service= as.numeric(test\_data$service)

train\_data$class1= ifelse(train\_data$y\_var==1,"anomaly", "Normal")

train\_data$y\_var= NULL

names(train\_data)[names(train\_data)== "class1"]= "y\_var"

validate\_data$class1= ifelse(validate\_data$y\_var==1,"anomaly", "Normal")

validate\_data$y\_var= NULL

names(validate\_data)[names(validate\_data)== "class1"]= "y\_var"

train\_data$y\_var= as.factor(train\_data$y\_var)

validate\_data$y\_var= as.factor(validate\_data$y\_var)

class(train\_data$y\_var)

"factor"

# default RF model

m1 <- randomForest(

formula = train\_data$y\_var ~ .,

data = train\_data

)

m1

**Sample output-**

Call:

randomForest(formula = train\_data$y\_var ~ ., data = train\_data)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 6

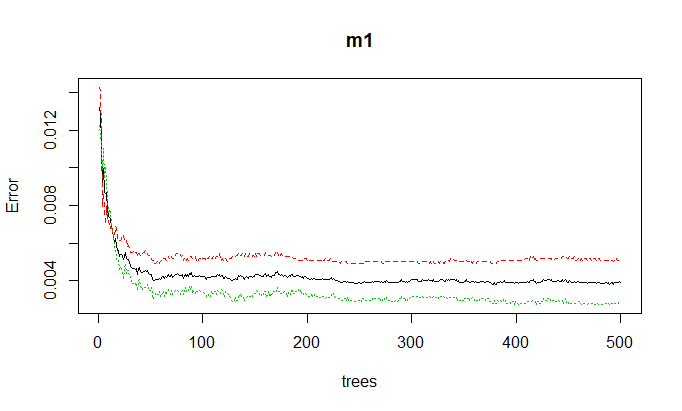
OOB estimate of error rate: 0.39%

Confusion matrix:

anomaly Normal class.error

anomaly 11683 60 0.005109427

Normal 38 13411 0.002825489

plot(m1) 

# Number of tree with lowest error rate

min(m1$err.rate)

#0.002751134

which.min(m1$err.rate)

#1398

y\_pred = predict(m1, newdata = validate\_data[,!(names(validate\_data) %in% c("y\_var"))],type ="class")

# Making the Confusion Matrix to test accuracy

cm = table(validate\_data$y\_var, y\_pred)

accuracy= (cm[1,1] + cm[2,2])/ sum(cm)

# 0.7605909

print(accuracy)

# 0.7765604 ntree=400

**##Tuning parameters of Random Forest**

#ntree

#mtry

#sampsize

#nodesize

#maxnodes

# tuning the mtry parameter we can use randomForest::tuneRF for a quick and easy tuning assessment.

# tuneRf will start at a value of mtry that you supply and increase

# by a certain step factor until the OOB error stops improving be a specified amount.

# names of features

features <- setdiff(names(train\_data), "y\_var")

set.seed(123)

m2 <- tuneRF(

x = train\_data[features],

y = train\_data$y\_var,

ntreeTry = 400,

mtryStart = 5,

stepFactor = 2,

improve = 0.01,

trace = TRUE # to not show real-time progress

)

**Console output**

mtry = 5 OOB error = 0.53%

Searching left ...

mtry = 3 OOB error = 0.77%

-0.4586466 0.01

Searching right ...

mtry = 10 OOB error = 0.32%

0.3984962 0.01

mtry = 20 OOB error = 0.33%

-0.0375 0.01

This shows that mtry= 20 is having lest error when ntree= 500

m1 <- randomForest(

formula = train\_data$y\_var ~ .,

data = train\_data,

ntreeTry = 500,

mtry =20

)

**Sample output**

**m1**

Call:

randomForest(formula = train\_data$y\_var ~ ., data = train\_data, ntreeTry = 500, mtry = 20)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 20

OOB estimate of error rate: 0.33%

Confusion matrix:

anomaly Normal class.error

anomaly 11693 50 0.004257856

Normal 32 13417 0.002379359

y\_pred = predict(m1, newdata = validate\_data[,!(names(validate\_data) %in% c("y\_var"))],type ="class")

# Making the Confusion Matrix to test accuracy

cm = table(validate\_data$y\_var, y\_pred)

y\_pred

anomaly Normal

anomaly 8074 4759

Normal 244 9466

accuracy= (cm[1,1] + cm[2,2])/ sum(cm)

print(accuracy)

0.7780686

#0.7781129 ntree=500 mtry=20

#0.7644502 ntree=400 mtry=40

#printing important variables

library(caret)

RM\_IMP=varImp(m1, scale= F)

|  |  |
| --- | --- |
| dst\_bytes\_Flag | 4430.01 |
| flag | 2051.48 |
| src\_bytes\_Flag | 848.22 |
| same\_srv\_rate | 636.58 |
| protocol\_type | 584.21 |
| count | 519.67 |
| dst\_host\_same\_src\_port\_rate | 442.18 |
| service | 390.11 |
| diff\_srv\_rate | 337.67 |
| dst\_host\_same\_srv\_rate | 318.51 |
| diff\_srv\_rate\_Flag | 252.14 |
| dst\_host\_diff\_srv\_rate | 249.43 |
| dst\_host\_srv\_count | 208.77 |
| dst\_host\_srv\_diff\_host\_rate | 148.14 |
| dst\_host\_count | 133.28 |
| hot | 130.60 |
| hot\_Flag | 114.42 |
| logged\_in\_Flag | 97.70 |
| duration | 91.89 |
| dst\_host\_rerror\_rate\_Flag | 91.36 |
| wrong\_fragment\_Flag | 83.23 |
| dst\_host\_srv\_serror\_rate | 77.52 |
| dst\_host\_diff\_srv\_rate\_Flag | 58.00 |
| dst\_host\_same\_src\_port\_rate\_Flag | 44.48 |
| num\_compromised\_Flag | 37.34 |
| dst\_host\_srv\_diff\_host\_rate\_Flag | 33.76 |
| num\_compromised | 33.05 |
| rerror\_rate\_Flag | 16.14 |
| srv\_diff\_host\_rate | 14.25 |
| dst\_host\_srv\_rerror\_rate\_Flag | 10.70 |
| dst\_host\_serror\_rate\_Flag | 8.83 |
| duration\_Flag | 8.20 |
| srv\_diff\_host\_rate\_Flag | 7.87 |
| num\_root\_Flag | 5.49 |
| is\_guest\_login\_Flag | 4.63 |
| dst\_host\_same\_srv\_rate\_Flag | 3.88 |
| num\_file\_creations | 1.70 |
| num\_access\_files | 1.59 |
| num\_access\_files\_Flag | 1.53 |
| root\_shell\_Flag | 1.40 |
| num\_file\_creations\_Flag | 1.14 |
| same\_srv\_rate\_Flag | 0.92 |
| su\_attempted | 0.49 |
| num\_shells\_Flag | 0.09 |

#To get the area under the ROC curve for each predictor, the filterVarImp function can be used

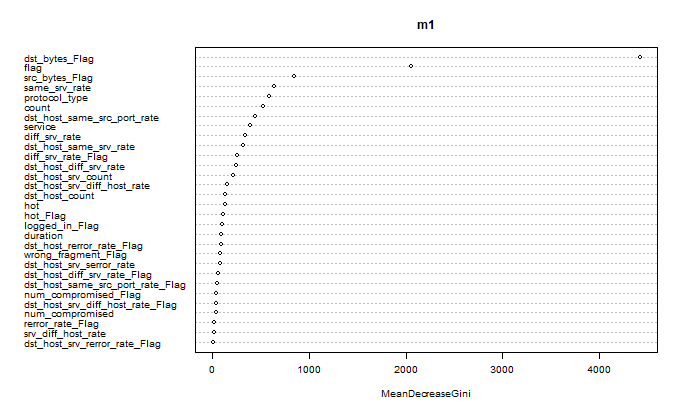
roc\_imp =filterVarImp(x = train\_data[, -ncol(train\_data)], y = train\_data$y\_var)

roc\_imp

anomaly Normal

|  |  |  |
| --- | --- | --- |
|  | anomaly | Normal |
| dst\_bytes\_Flag | 0.9052895 | 0.9052895 |
| dst\_host\_srv\_count | 0.8908315 | 0.8908315 |
| same\_srv\_rate | 0.8723819 | 0.8723819 |
| src\_bytes\_Flag | 0.8666435 | 0.8666435 |
| dst\_host\_same\_srv\_rate | 0.8642143 | 0.8642143 |
| flag | 0.8592038 | 0.8592038 |
| diff\_srv\_rate\_Flag | 0.8498343 | 0.8498343 |
| diff\_srv\_rate | 0.8420721 | 0.8420721 |
| logged\_in\_Flag | 0.8371099 | 0.8371099 |
| dst\_host\_diff\_srv\_rate | 0.8248347 | 0.8248347 |
| count | 0.8207797 | 0.8207797 |
| dst\_host\_srv\_serror\_rate | 0.7838931 | 0.7838931 |
| dst\_host\_serror\_rate\_Flag | 0.783633 | 0.783633 |
| dst\_host\_diff\_srv\_rate\_Flag | 0.7363288 | 0.7363288 |
| dst\_host\_count | 0.7044021 | 0.7044021 |
| dst\_host\_srv\_diff\_host\_rate\_Flag | 0.7020398 | 0.7020398 |
| dst\_host\_same\_src\_port\_rate\_Flag | 0.6813339 | 0.6813339 |
| dst\_host\_srv\_diff\_host\_rate | 0.6811461 | 0.6811461 |
| srv\_diff\_host\_rate\_Flag | 0.6470976 | 0.6470976 |
| srv\_diff\_host\_rate | 0.637861 | 0.637861 |
| dst\_host\_same\_src\_port\_rate | 0.630386 | 0.630386 |
| service | 0.6250169 | 0.6250169 |
| protocol\_type | 0.6107479 | 0.6107479 |
| rerror\_rate\_Flag | 0.5871804 | 0.5871804 |
| dst\_host\_rerror\_rate\_Flag | 0.5776429 | 0.5776429 |
| dst\_host\_srv\_rerror\_rate\_Flag | 0.5660878 | 0.5660878 |
| duration\_Flag | 0.5413509 | 0.5413509 |
| duration | 0.5405969 | 0.5405969 |
| dst\_host\_same\_srv\_rate\_Flag | 0.5404517 | 0.5404517 |
| same\_srv\_rate\_Flag | 0.5226416 | 0.5226416 |
| wrong\_fragment\_Flag | 0.5095376 | 0.5095376 |
| num\_root\_Flag | 0.5046628 | 0.5046628 |
| num\_compromised\_Flag | 0.5044033 | 0.5044033 |
| num\_compromised | 0.5043644 | 0.5043644 |
| hot\_Flag | 0.5037172 | 0.5037172 |
| is\_guest\_login\_Flag | 0.5036857 | 0.5036857 |
| hot | 0.5036375 | 0.5036375 |
| num\_access\_files\_Flag | 0.502618 | 0.502618 |
| num\_access\_files | 0.5026179 | 0.5026179 |
| num\_file\_creations | 0.5016562 | 0.5016562 |
| num\_file\_creations\_Flag | 0.5016562 | 0.5016562 |
| su\_attempted | 0.5007807 | 0.5007807 |
| root\_shell\_Flag | 0.5007321 | 0.5007321 |
| num\_shells\_Flag | 0.5002548 | 0.5002548 |

varImpPlot(m1,type=2, pretty=T, cex=0.6)



**#hyper parameter tuning using H2O package**

# start up h2o

# Activate h2o package for using:

library(h2o)

h2o.init(nthreads = 20, max\_mem\_size = "16g")

h2o.no\_progress()

# Convert to h2o Frame and identify inputs and output:

test <- as.h2o(validate\_data)

train <- as.h2o(train\_data)

y <- "y\_var"

x <- setdiff(names(train), y)

#Default Random Forest

# Train default Random Forest:

default\_rf <- h2o.randomForest(x = x, y = y,

training\_frame = train,

stopping\_rounds = 5,

stopping\_tolerance = 0.001,

stopping\_metric = "AUC",

seed = 29,

balance\_classes = FALSE,

nfolds = 10)

default\_rf

Model Details:

==============

H2OBinomialModel: drf

Model ID: DRF\_model\_R\_1569318231672\_1

Model Summary:

number\_of\_trees number\_of\_internal\_trees model\_size\_in\_bytes min\_depth max\_depth mean\_depth

1 50 50 157886 16 20 18.78000

min\_leaves max\_leaves mean\_leaves

1 210 284 245.02000

H2OBinomialMetrics: drf

\*\* Reported on training data. \*\*

\*\* Metrics reported on Out-Of-Bag training samples \*\*

MSE: 0.003147487

RMSE: 0.05610247

LogLoss: 0.01933628

Mean Per-Class Error: 0.003425369

AUC: 0.9997053

pr\_auc: 0.4590492

Gini: 0.9994106

R^2: 0.987352

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

Normal anomaly Error Rate

Normal 13421 28 0.002082 =28/13449

anomaly 56 11687 0.004769 =56/11743

Totals 13477 11715 0.003334 =84/25192

Maximum Metrics: Maximum metrics at their respective thresholds

metric threshold value idx

1 max f1 0.533333 0.996419 196

2 max f2 0.261905 0.996871 255

3 max f0point5 0.666774 0.997295 167

4 max accuracy 0.533333 0.996666 196

5 max precision 1.000000 1.000000 0

6 max recall 0.000003 1.000000 399

7 max specificity 1.000000 1.000000 0

8 max absolute\_mcc 0.533333 0.993302 196

9 max min\_per\_class\_accuracy 0.442141 0.996423 215

10 max mean\_per\_class\_accuracy 0.533333 0.996575 196

11 max tns 1.000000 13449.000000 0

12 max fns 1.000000 5393.000000 0

13 max fps 0.000003 13449.000000 399

14 max tps 0.000003 11743.000000 399

15 max tnr 1.000000 1.000000 0

16 max fnr 1.000000 0.459252 0

17 max fpr 0.000003 1.000000 399

18 max tpr 0.000003 1.000000 399

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

H2OBinomialMetrics: drf

\*\* Reported on cross-validation data. \*\*

\*\* 10-fold cross-validation on training data (Metrics computed for combined holdout predictions) \*\*

MSE: 0.003162394

RMSE: 0.05623517

LogLoss: 0.01664645

Mean Per-Class Error: 0.003404395

AUC: 0.9998083

pr\_auc: 0.2537805

Gini: 0.9996166

R^2: 0.9872921

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

Normal anomaly Error Rate

Normal 13425 24 0.001785 =24/13449

anomaly 59 11684 0.005024 =59/11743

Totals 13484 11708 0.003295 =83/25192

Maximum Metrics: Maximum metrics at their respective thresholds

metric threshold value idx

1 max f1 0.554000 0.996461 186

2 max f2 0.319354 0.996835 234

3 max f0point5 0.683791 0.997636 160

4 max accuracy 0.554000 0.996705 186

5 max precision 0.999983 1.000000 0

6 max recall 0.000008 1.000000 399

7 max specificity 0.999983 1.000000 0

8 max absolute\_mcc 0.554000 0.993383 186

9 max min\_per\_class\_accuracy 0.433077 0.996208 207

10 max mean\_per\_class\_accuracy 0.473927 0.996623 199

11 max tns 0.999983 13449.000000 0

12 max fns 0.999983 2982.000000 0

13 max fps 0.000008 13449.000000 399

14 max tps 0.000008 11743.000000 399

15 max tnr 0.999983 1.000000 0

16 max fnr 0.999983 0.253939 0

17 max fpr 0.000008 1.000000 399

18 max tpr 0.000008 1.000000 399

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

Cross-Validation Metrics Summary:

mean sd cv\_1\_valid cv\_2\_valid cv\_3\_valid

accuracy 0.99714416 9.274678E-4 0.9967506 0.9987775 0.99763685

auc 0.9997602 4.6892744E-4 0.9999416 0.999992 0.9999689

err 0.0028558217 9.274678E-4 0.0032493907 0.001222494 0.002363135

err\_count 7.2 2.3475757 8.0 3.0 6.0

f0point5 0.9973942 9.409508E-4 0.99647886 0.9979065 0.99850225

f1 0.9969331 9.931794E-4 0.99647886 0.99869055 0.99750626

f2 0.9964739 0.0016382475 0.99647886 0.9994758 0.99651223

lift\_top\_group 2.1463132 0.053250656 2.1672535 2.145105 2.107054

logloss 0.016587088 0.008199789 0.012556453 0.0105226915 0.011567972

max\_per\_class\_error 0.0044385567 0.0014096908 0.0035211267 0.0022900763 0.004149378

mcc 0.9942632 0.0018617458 0.99346226 0.9975472 0.9952657

mean\_per\_class\_accuracy 0.9970736 9.759807E-4 0.99673116 0.99885494 0.9975505

mean\_per\_class\_error 0.002926437 9.759807E-4 0.003268859 0.0011450382 0.0024495013

mse 0.0031594378 7.377715E-4 0.002849403 0.001960692 0.0025541056

precision 0.9977026 0.0013062503 0.99647886 0.9973845 0.9991674

r2 0.9872977 0.0029646344 0.9885341 0.99212116 0.9897571

recall 0.9961688 0.0021681988 0.99647886 1.0 0.9958506

rmse 0.05586253 0.006567192 0.0533798 0.044279702 0.05053816

specificity 0.9979783 0.0012060009 0.9969834 0.99770993 0.99925035

cv\_4\_valid cv\_5\_valid cv\_6\_valid cv\_7\_valid cv\_8\_valid

accuracy 0.996892 0.99605054 0.99805826 0.99762285 0.9956556

auc 0.99844986 0.9999099 0.99990463 0.99995404 0.9998114

err 0.0031080032 0.003949447 0.0019417476 0.002377179 0.0043443916

err\_count 8.0 10.0 5.0 6.0 11.0

f0point5 0.9970186 0.9968062 0.99864244 0.9979906 0.9966583

f1 0.9964943 0.99580187 0.99788046 0.9974895 0.9954109

f2 0.99597055 0.99479955 0.9971196 0.99698895 0.9941667

lift\_top\_group 2.2539403 2.1223805 2.1803555 2.1103678 2.1082432

logloss 0.038729653 0.014837253 0.013443243 0.012981504 0.018278621

max\_per\_class\_error 0.004378284 0.005867561 0.0033869601 0.0033444816 0.0066611157

mcc 0.9937042 0.9920783 0.9960917 0.9952335 0.99129426

mean\_per\_class\_accuracy 0.99676335 0.995946 0.9979478 0.99757475 0.99554247

mean\_per\_class\_error 0.003236628 0.004054019 0.0020521602 0.002425253 0.00445753

mse 0.0033688731 0.0037248004 0.0027493557 0.0028288143 0.004553979

precision 0.9973684 0.9974769 0.9991511 0.99832493 0.99749166

r2 0.98635125 0.9850511 0.9889268 0.9886537 0.98173594

recall 0.99562174 0.99413246 0.996613 0.9966555 0.9933389

rmse 0.058041994 0.06103114 0.0524343 0.0531866 0.06748317

specificity 0.997905 0.9977595 0.99928266 0.99849397 0.99774605

cv\_9\_valid cv\_10\_valid

accuracy 0.996782 0.99721557

auc 0.99969864 0.9999715

err 0.003218021 0.0027844072

err\_count 8.0 7.0

f0point5 0.99805 0.9958882

f1 0.9964602 0.9971182

f2 0.99487543 0.9983512

lift\_top\_group 2.1941748 2.0742574

logloss 0.018450288 0.014503209

max\_per\_class\_error 0.006178288 0.004608295

mcc 0.9935221 0.99443275

mean\_per\_class\_accuracy 0.9965413 0.99728334

mean\_per\_class\_error 0.003458693 0.0027166887

mse 0.003800974 0.0032033809

precision 0.99911267 0.99506986

r2 0.9846761 0.98717004

recall 0.99382174 0.9991749

rmse 0.06165204 0.056598417

specificity 0.9992609 0.9953917

**# Function for collecting cross-validation results:**

results\_cross\_validation <- function(h2o\_model) {

h2o\_model@model$cross\_validation\_metrics\_summary %>%

as.data.frame() %>%

select(-mean, -sd) %>%

t() %>%

as.data.frame() %>%

mutate\_all(as.character) %>%

mutate\_all(as.numeric) %>%

select(Accuracy = accuracy,

AUC = auc,

Precision = precision,

Specificity = specificity,

Recall = recall,

Logloss = logloss) %>%

return()

}

**# Use function**:

results\_cross\_validation(default\_rf) -> ket\_qua\_default

ket\_qua\_default

Accuracy AUC Precision Specificity Recall Logloss

1 0.9967506 0.9999416 0.9964789 0.9969834 0.9964789 0.01255645

2 0.9987775 0.9999920 0.9973845 0.9977099 1.0000000 0.01052269

3 0.9976369 0.9999689 0.9991674 0.9992503 0.9958506 0.01156797

4 0.9968920 0.9984499 0.9973684 0.9979050 0.9956217 0.03872965

5 0.9960505 0.9999099 0.9974769 0.9977595 0.9941325 0.01483725

6 0.9980583 0.9999046 0.9991511 0.9992827 0.9966130 0.01344324

7 0.9976228 0.9999540 0.9983249 0.9984940 0.9966555 0.01298150

8 0.9956556 0.9998114 0.9974917 0.9977461 0.9933389 0.01827862

9 0.9967820 0.9996986 0.9991127 0.9992609 0.9938217 0.01845029

10 0.9972156 0.9999715 0.9950699 0.9953917 0.9991749 0.01450321

**# Model Performance by Graph:**

theme\_set(theme\_minimal())

plot\_results <- function(df\_results) {

df\_results %>%

gather(Metrics, Values) %>%

ggplot(aes(Metrics, Values, fill = Metrics, color = Metrics)) +

geom\_boxplot(alpha = 0.3, show.legend = FALSE) +

theme(plot.margin = unit(c(1, 1, 1, 1), "cm")) +

scale\_y\_continuous(labels = scales::percent) +

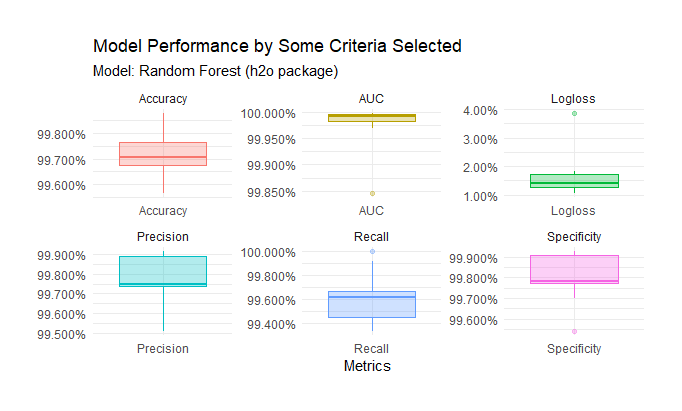
facet\_wrap(~ Metrics, scales = "free") +

labs(title = "Model Performance by Some Criteria Selected", y = NULL)

}

plot\_results(ket\_qua\_default) +

labs(subtitle = "Model: Random Forest (h2o package)")



# Model performance based on test data:

pred\_class <- h2o.predict(default\_rf, test) %>% as.data.frame() %>% pull(predict)

library(caret)

confusionMatrix(pred\_class, validate\_data$y\_var) #Accuracy : 0.7849

Confusion Matrix and Statistics

Reference

Prediction anomaly Normal

anomaly 7736 256

Normal 5097 9454

Accuracy : 0.7625

95% CI : (0.7569, 0.7681)

No Information Rate : 0.5693

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.5435

Mcnemar's Test P-Value : < 2.2e-16

Sensitivity : 0.6028

Specificity : 0.9736

Pos Pred Value : 0.9680

Neg Pred Value : 0.6497

Prevalence : 0.5693

Detection Rate : 0.3432

Detection Prevalence : 0.3545

Balanced Accuracy : 0.7882

'Positive' Class : anomaly

**# ROC curve and AUC:**

library(pROC)

**# Function calculates AUC:**

auc\_for\_test <- function(model\_selected) {

actual <- validate\_data$y\_var

pred\_prob <- h2o.predict(model\_selected, test) %>% as.data.frame() %>% pull(anomaly)

return(roc(actual, pred\_prob))

}

**# Use this function:**

my\_auc <- auc\_for\_test(default\_rf)

my\_auc$auc

**#Area under the curve: 0.931**

**# Graph ROC and AUC:**

sen\_spec\_df <- data\_frame(TPR = my\_auc$sensitivities, FPR = 1 - my\_auc$specificities)

sen\_spec\_df %>%

ggplot(aes(x = FPR, ymin = 0, ymax = TPR))+

geom\_polygon(aes(y = TPR), fill = "red", alpha = 0.3)+

geom\_path(aes(y = TPR), col = "firebrick", size = 1.2) +

geom\_abline(intercept = 0, slope = 1, color = "gray37", size = 1, linetype = "dashed") +

theme\_bw() +

coord\_equal() +

labs(x = "FPR (1 - Specificity)",

y = "TPR (Sensitivity)",

title = "Model Performance for RF Classifier based on Test Data",

subtitle = paste0("AUC Value: ", my\_auc$auc %>% round(2)))



**# #=================================**

**# Full Cartesian Grid Search**

**#=================================**

**# Set hyperparameter grid:**

hyper\_grid.h2o <- list(ntrees = seq(250, 550, by = 50),

mtries = seq(5, 40, by = 5)

# max\_depth = seq(10, 30, by = 10),

# min\_rows = seq(1, 3, by = 1),

# nbins = seq(20, 30, by = 10),

#sample\_rate = c(0.55, 0.632, 0.75)

)

# The number of models is 56:

sapply(hyper\_grid.h2o, length) %>% prod()

# Train Random Forest Models:

system.time(grid\_cartesian <- h2o.grid(algorithm = "randomForest",

grid\_id = "rf\_grid1",

x = x,

y = y,

seed = 29,

nfolds = 10,

training\_frame = train,

stopping\_metric = "AUC",

hyper\_params = hyper\_grid.h2o,

search\_criteria = list(strategy = "Cartesian")))

# Save an CARTESIAN RF MODEL OUTPUT object to a file

**Sample output-**

system.time(grid\_cartesian <- h2o.grid(algorithm = "randomForest",

+ grid\_id = "rf\_grid1",

+ x = x,

+ y = y,

+ seed = 29,

+ nfolds = 10,

+ training\_frame = train,

+ stopping\_metric = "AUC",

+ hyper\_params = hyper\_grid.h2o,

+ search\_criteria = list(strategy = "Cartesian")))

# user system elapsed

# 57.51 10.13 24327.88

saveRDS(grid\_cartesian, file = "RF MODELS USING GRID SEARCH 2\_ITERATION.rds")

#h2o.shutdown()

grid\_cartesian <- readRDS("RF MODELS USING GRID SEARCH 2\_ITERATION.rds")

grid\_cartesian

H2O Grid Details

================

Grid ID: rf\_grid1

Used hyper parameters:

- mtries

- ntrees

Number of models: 56

Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss

mtries ntrees model\_ids logloss

1 10 550 rf\_grid1\_model\_50 0.01160796683396694

2 20 550 rf\_grid1\_model\_52 0.012286464394837857

3 10 450 rf\_grid1\_model\_34 0.012524680543018231

4 10 500 rf\_grid1\_model\_42 0.012533799283187376

5 10 400 rf\_grid1\_model\_26 0.012607095426678844

---

mtries ntrees model\_ids logloss

51 40 400 rf\_grid1\_model\_32 0.022576408108815813

52 40 500 rf\_grid1\_model\_48 0.02259628551604085

53 40 550 rf\_grid1\_model\_56 0.022622765902021714

54 40 350 rf\_grid1\_model\_24 0.02262698983448474

55 40 250 rf\_grid1\_model\_8 0.02379758089078724

56 40 300 rf\_grid1\_model\_16 0.023814621585695663

**# Collect the results and sort by our model performance metric of choice:**

grid\_perf <- h2o.getGrid(grid\_id = "rf\_grid1",

sort\_by = "accuracy",

decreasing = TRUE)

grid\_perf

# H2O Grid Details

# ================

#

# Grid ID: rf\_grid1

# Used hyper parameters:

# - mtries

# - ntrees

# Number of models: 56

# Number of failed models: 0

#

# Hyper-Parameter Search Summary: ordered by decreasing accuracy

# mtries ntrees model\_ids accuracy

# 1 20 300 rf\_grid1\_model\_12 0.9971816449666561

# 2 20 350 rf\_grid1\_model\_20 0.9971816449666561

# 3 20 450 rf\_grid1\_model\_36 0.9971419498253413

# 4 20 550 rf\_grid1\_model\_52 0.9971419498253413

# 5 20 500 rf\_grid1\_model\_44 0.9971419498253413

#

# ---

# mtries ntrees model\_ids accuracy

# 51 40 400 rf\_grid1\_model\_32 0.996268656716418

# 52 40 500 rf\_grid1\_model\_48 0.996268656716418

# 53 40 350 rf\_grid1\_model\_24 0.996268656716418

# 54 40 450 rf\_grid1\_model\_40 0.996268656716418

# 55 40 250 rf\_grid1\_model\_8 0.9961892664337885

# 56 40 300 rf\_grid1\_model\_16 0.9961495712924738

# Best model chosen by validation error:

best\_model <- h2o.getModel(grid\_perf@model\_ids[[1]])

# Model Details:

# ==============

#

# H2OBinomialModel: drf

# Model ID: rf\_grid1\_model\_12

# Model Summary:

# number\_of\_trees number\_of\_internal\_trees model\_size\_in\_bytes min\_depth max\_depth mean\_depth

# 1 300 300 623272 13 20 17.23333

# min\_leaves max\_leaves mean\_leaves

# 1 129 194 159.83667

#

#

# H2OBinomialMetrics: drf

# \*\* Reported on training data. \*\*

# \*\* Metrics reported on Out-Of-Bag training samples \*\*

#

# MSE: 0.002417252

# RMSE: 0.04916556

# LogLoss: 0.01097108

# Mean Per-Class Error: 0.002721878

# AUC: 0.9998924

# pr\_auc: 0.06036745

# Gini: 0.9997848

# R^2: 0.9902864

#

# Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

# Normal anomaly Error Rate

# Normal 13409 40 0.002974 =40/13449

# anomaly 29 11714 0.002470 =29/11743

# Totals 13438 11754 0.002739 =69/25192

#

# Maximum Metrics: Maximum metrics at their respective thresholds

# metric threshold value idx

# 1 max f1 0.400000 0.997063 202

# 2 max f2 0.306874 0.997549 224

# 3 max f0point5 0.633929 0.997778 156

# 4 max accuracy 0.400000 0.997261 202

# 5 max precision 1.000000 1.000000 0

# 6 max recall 0.000001 1.000000 399

# 7 max specificity 1.000000 1.000000 0

# 8 max absolute\_mcc 0.400000 0.994498 202

# 9 max min\_per\_class\_accuracy 0.425227 0.997175 196

# 10 max mean\_per\_class\_accuracy 0.400000 0.997278 202

# 11 max tns 1.000000 13449.000000 0

# 12 max fns 1.000000 710.000000 0

# 13 max fps 0.000001 13449.000000 399

# 14 max tps 0.000001 11743.000000 399

# 15 max tnr 1.000000 1.000000 0

# 16 max fnr 1.000000 0.060462 0

# 17 max fpr 0.000001 1.000000 399

# 18 max tpr 0.000001 1.000000 399

#

# Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

#

# H2OBinomialMetrics: drf

# \*\* Reported on cross-validation data. \*\*

# \*\* 10-fold cross-validation on training data (Metrics computed for combined holdout predictions) \*\*

#

# MSE: 0.002621235

# RMSE: 0.051198

# LogLoss: 0.01459354

# Mean Per-Class Error: 0.00283404

# AUC: 0.9997688

# pr\_auc: 0.08778887

# Gini: 0.9995375

# R^2: 0.9894668

#

# Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

# Normal anomaly Error Rate

# Normal 13414 35 0.002602 =35/13449

# anomaly 36 11707 0.003066 =36/11743

# Totals 13450 11742 0.002818 =71/25192

#

# Maximum Metrics: Maximum metrics at their respective thresholds

# metric threshold value idx

# 1 max f1 0.450005 0.996977 175

# 2 max f2 0.342222 0.997106 196

# 3 max f0point5 0.716667 0.997928 118

# 4 max accuracy 0.450005 0.997182 175

# 5 max precision 1.000000 1.000000 0

# 6 max recall 0.000000 1.000000 399

# 7 max specificity 1.000000 1.000000 0

# 8 max absolute\_mcc 0.450005 0.994337 175

# 9 max min\_per\_class\_accuracy 0.433889 0.997020 177

# 10 max mean\_per\_class\_accuracy 0.450005 0.997166 175

# 11 max tns 1.000000 13449.000000 0

# 12 max fns 1.000000 1033.000000 0

# 13 max fps 0.000000 13449.000000 399

# 14 max tps 0.000000 11743.000000 399

# 15 max tnr 1.000000 1.000000 0

# 16 max fnr 1.000000 0.087967 0

# 17 max fpr 0.000000 1.000000 399

# 18 max tpr 0.000000 1.000000 399

#

# Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

# Cross-Validation Metrics Summary:

# mean sd cv\_1\_valid cv\_2\_valid cv\_3\_valid

# accuracy 0.9976197 7.985348E-4 0.9967506 0.999185 0.99763685

# auc 0.99977 3.4329726E-4 0.9999187 0.99999535 0.999972

# err 0.0023803273 7.985348E-4 0.0032493907 8.149959E-4 0.002363135

# err\_count 6.0 2.0 8.0 2.0 6.0

# f0point5 0.99750704 6.771867E-4 0.99752825 0.99860334 0.99652374

# f1 0.99744487 8.5620576E-4 0.99647266 0.9991266 0.9975145

# f2 0.9973837 0.0014661337 0.9954193 0.9996505 0.9985072

# lift\_top\_group 2.1463132 0.053250656 2.1672535 2.145105 2.107054

# logloss 0.014586449 0.009609783 0.011049904 0.006853847 0.007787449

# max\_per\_class\_error 0.0035422083 0.0010572101 0.00528169 0.0015267176 0.0037481259

# mcc 0.9952176 0.0016035072 0.99346596 0.99836403 0.9952672

# mean\_per\_class\_accuracy 0.997588 8.4825105E-4 0.996605 0.99923664 0.997711

# mean\_per\_class\_error 0.0024120486 8.4825105E-4 0.0033949928 7.633588E-4 0.0022890007

# mse 0.002619952 6.795504E-4 0.0029329236 0.0015272287 0.002041509

# precision 0.9975491 9.538492E-4 0.9982332 0.9982548 0.99586433

# r2 0.98946655 0.0027301975 0.98819804 0.993863 0.9918128

# recall 0.9973435 0.0019309199 0.9947183 1.0 0.9991701

# rmse 0.050793175 0.0066671083 0.054156475 0.039079774 0.04518306

# specificity 0.9978324 9.212008E-4 0.9984917 0.9984733 0.9962519

# cv\_4\_valid cv\_5\_valid cv\_6\_valid cv\_7\_valid cv\_8\_valid

# accuracy 0.9980575 0.99723536 0.9976699 0.99762285 0.9964455

# auc 0.9993258 0.9999377 0.99991554 0.99996066 0.9997285

# err 0.001942502 0.002764613 0.0023300971 0.002377179 0.0035545023

# err\_count 5.0 7.0 6.0 6.0 9.0

# f0point5 0.9980722 0.9978162 0.99796504 0.99699396 0.9965006

# f1 0.9978099 0.9970625 0.9974576 0.99749374 0.9962516

# f2 0.99754775 0.99630994 0.9969507 0.997994 0.9960027

# lift\_top\_group 2.2539403 2.1223805 2.1803555 2.1103678 2.1082432

# logloss 0.025318911 0.0112300785 0.010562367 0.00831061 0.017545445

# max\_per\_class\_error 0.0026269702 0.004191115 0.0033869601 0.003012048 0.0041631972

# mcc 0.996065 0.9944544 0.99530834 0.99523425 0.99287224

# mean\_per\_class\_accuracy 0.99798816 0.99715763 0.9975892 0.99765784 0.9964158

# mean\_per\_class\_error 0.002011809 0.0028423835 0.0024108402 0.0023421445 0.0035842282

# mse 0.0025702917 0.0030531236 0.0024512107 0.0020695538 0.003923906

# precision 0.99824715 0.9983193 0.99830365 0.9966611 0.99666667

# r2 0.98958665 0.9877468 0.9901276 0.9916991 0.9842629

# recall 0.99737304 0.9958089 0.996613 0.99832773 0.9958368

# rmse 0.050698046 0.055255078 0.049509704 0.045492347 0.062641084

# specificity 0.99860334 0.99850637 0.99856526 0.99698794 0.99699473

# cv\_9\_valid cv\_10\_valid

# accuracy 0.9971842 0.9984089

# auc 0.99898005 0.9999658

# err 0.0028157684 0.0015910899

# err\_count 7.0 4.0

# f0point5 0.99770033 0.99736667

# f1 0.99690676 0.9983525

# f2 0.99611443 0.99934036

# lift\_top\_group 2.1941748 2.0742574

# logloss 0.037004884 0.010200987

# max\_per\_class\_error 0.0044130627 0.0030721966

# mcc 0.99432576 0.99681914

# mean\_per\_class\_accuracy 0.9970544 0.9984639

# mean\_per\_class\_error 0.0029456296 0.0015360983

# mse 0.0031552515 0.0024745204

# precision 0.9982301 0.99671054

# r2 0.98727936 0.99008924

# recall 0.99558693 1.0

# rmse 0.056171626 0.04974455

# specificity 0.9985218 0.9969278

**# Use best model for making predictions:**

confusionMatrix(h2o.predict(best\_model, test) %>% as.data.frame() %>% pull(predict),

validate\_data$y\_var)

# Confusion Matrix and Statistics

#

# Reference

# Prediction anomaly Normal

# anomaly 8701 278

# Normal 4132 9432

#

# Accuracy : 0.8044

# 95% CI : (0.7991, 0.8095)

# No Information Rate : 0.5693

# P-Value [Acc > NIR] : < 2.2e-16

#

# Kappa : 0.6195

#

# Mcnemar's Test P-Value : < 2.2e-16

#

# Sensitivity : 0.6780

# Specificity : 0.9714

# Pos Pred Value : 0.9690

# Neg Pred Value : 0.6954

# Prevalence : 0.5693

# Detection Rate : 0.3860

# Detection Prevalence : 0.3983

# Balanced Accuracy : 0.8247

#

# 'Positive' Class : anomaly

**Using Best model to make predictions- pre-pruned decision tree**

**#predicting test values for best model**

y\_test = predict(classifier, newdata = test\_data, type = 'class')

y\_test\_fin= as.character(ifelse(y\_test==1, "Anomaly", "Normal"))

test\_data= cbind(test\_data,y\_test)

write.csv(test\_data, "Test Data with Predictions.csv")