CASE STUDY

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.

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

Feature name	Variable type	Type	Description
Duration	C	1	No. of seconds of the connection
Protocol_type	D	1	Type of protocol E.g.: TCP,UDP ,ICMP
Service	D	1	Network service on the destination E.g.: http, telnet
Flag	D	1	Normal or error status of the connection
src_bytes	C	1	Number of data bytes from source to destination
dst_bytes	C	1	Number of data bytes from destination to source

Land	D	1	1-connection is from the same host/port: 0-otherwise
Wrong_fragment	C	1	No. of 'wrong' fragments
Urgent	C	1	No of urgent fragments
Hot	C	2	The count of access to system directories, creation and execution of programs
Num_failed_logins	C	2	No. of failed login attempts
Logged_in	D	2	1-successfully logged in 0-otherwise
num_compromised	C	2	No. of compromised conditions
Root_shell	C	2	1-root shell is obtained;0 otherwise
Su_attempted	C	2	1-'su root' command attempted;0 otherwise
Num_root	C	2	No .of root accesses
num_file_creations	C	2	Number of file creation operations
Num_shells	C	2	No of shell prompts
Num_access_files	C	2	No. of write ,delete and create operations on access control files
Num_outbound_cmds	C	2	No. of outbound commands in an ftp session
Is_hot_login	D	2	1-the login belongs to the 'hot' list 0: otherwise
Count	C	3	No. of connections to the same host as the current connection in the past seconds
Srv_count	C	3	No of connections to the same host as the current connection in the past 2 seconds
serror_rate	C	3	% of connections that have 'SYN' errors to the same host
Srv_serror_rate	C	3	% of connections that have 'SYN' errors to the same service
Rerror_rate	C	3	% of connections that have 'REJ' errors to the same host
Srv_diff_host_rate	C	3	% of connections to different services and to the same host
Dst_host_count	C	3	No of connections to the same host to the destination host as the current connection in the past 2 seconds
Dst_host_srv_count	C	3	No of connections from the same service to the destination host as the

			current connection in the past 2 seconds
dst_host_srv_count	C	3	No. of connections from the same service to the destination host as the current connection in the past 2 seconds
Dst_host_srv_count	C	3	No. of connections from the same service to the destination host as the current connection in the past 2 seconds
Dst_host_same_srv_rate	C	3	% of connections from the same service to the destination host
Dst_host_diff_srv_rate	C	3	% of connections from the different services to the destination host
Dst_host_same_src_port_rate	C	3	% of connections from the port services to the destination host
Dst_host_srv_diff_host_rate	C	3	% of connections from the different hosts from the same service to destination host
Dst_host_serror_rate	C	3	% of connections that have ‘SYN” errors to same host to the destination host
dst_host_srv_serror_rate	C	3	% of connections that have ‘SYN’ errors from the same service to the destination host
Dst_host_rerror_rate	C	3	% of connections that have ‘REJ’ errors from the same host to destination host
Dst_host_srv_rerror_rate	C	3	% of connections that have ‘REJ’ errors from the same service to the destination host

Code Execution:

1.Data Imputation:

1.1 Set working directory & Set Seed:

```
setwd("E:/santosh_self/Working/DS/Imarticus/Imarticus/DSP23/3.Chapter(R)/3.12_Project 3 - Network Intrusion Detection using Decision Tree & Ensemble Learning in R")

#set the random number generation to a fixed value

set.seed(123)


rm(list = ls())

#Clean the workspace

gc()


#Console:

#> setwd("E:/santosh_self/Working/DS/Imarticus/Imarticus/DSP23/3.Chapter(R)/3.9_Project 2  Default Modelling using SVM in R/Project 3 - SVM Credit Risk Analytics in R")

#> rm(list = ls())

#> gc()

#          used (Mb) gc trigger (Mb) max used (Mb)

#Ncells  575388 30.8    1228650 65.7  1228650 65.7

#Vcells 2834692 21.7    8388608 64.0  3260580 24.9.0
```

1.2 Read Dataset:

```
# Read in the training datasets ad save it to object named "nw_train, nw_test,nw_valid"

nw_train<-read.csv("Network_Intrusion_Train_data.csv")

nw_test<-read.csv("Network_Intrusion_Test_data.csv")

nw_valid<-read.csv("Network_Intrusion_Validate_data.csv")

#Environment

#25192 obs. Of 42 variables
```

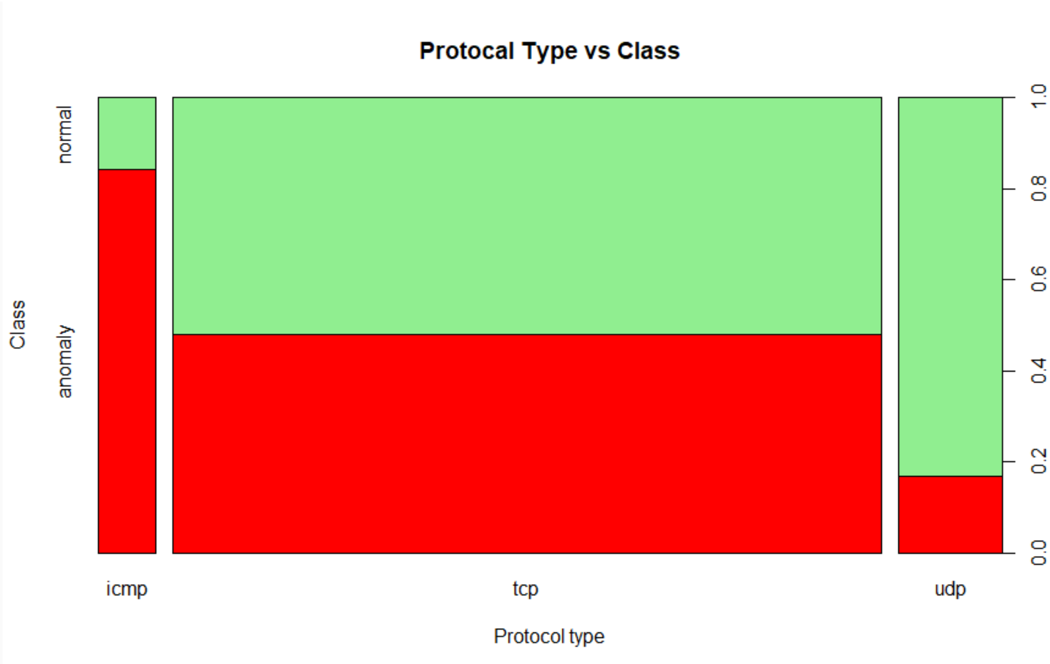
```
#22544 Obs. Of 41 variables
#22544 Obs. of 42 variables

#Check the structure of dataset
str(nw_train)
str(nw_test)
str(nw_valid)

# Check the summary of datasets
summary(nw_train)
summary(nw_test)
summary(nw_valid)
```

```
# use the below function  on the training dataset
dim(nw_train)
head(nw_train)
str(nw_train)
summary(nw_train)
```

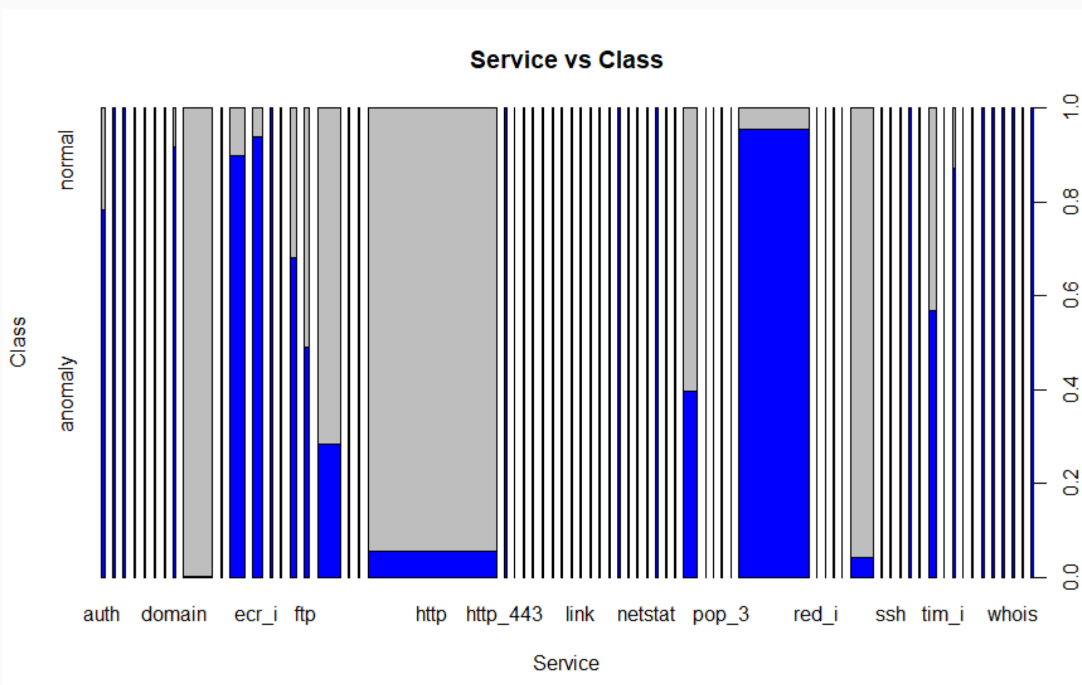
1.3 Plot the relationship between discrete variables and the output variable

Ideally, the proportion of events and non-events in the Y variable should approximately be the same. So, lets first check the proportion of classes in the dependent variables Gender

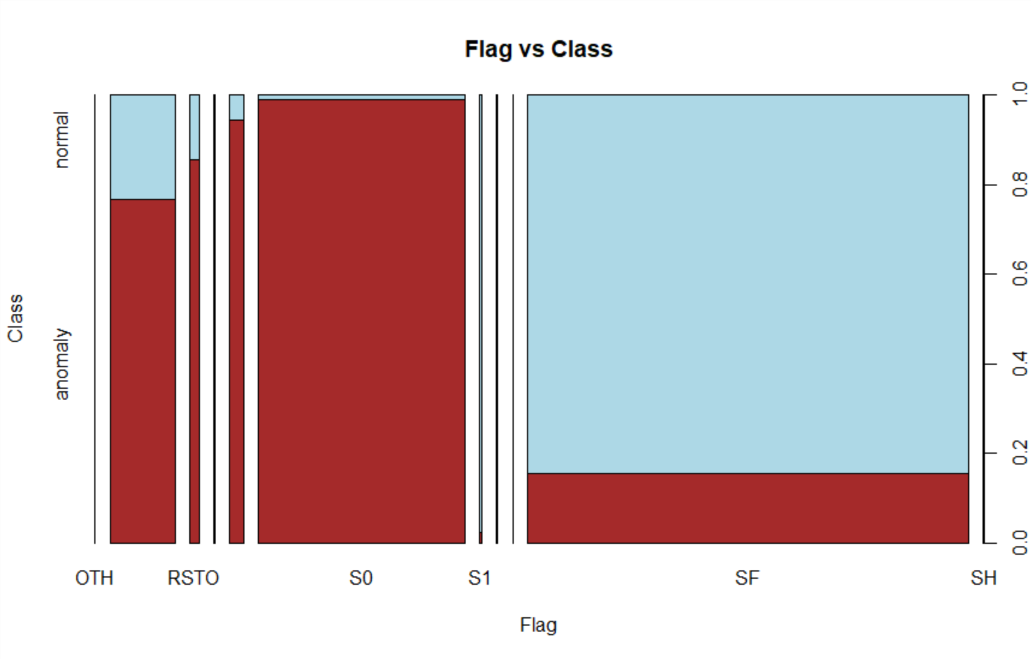
```
plot(nw_train$protocol_type, nw_train$class, main="Protocal Type vs Class",xl
ab="Protocol type", ylab="Class", col=c("red","lightgreen"))
```



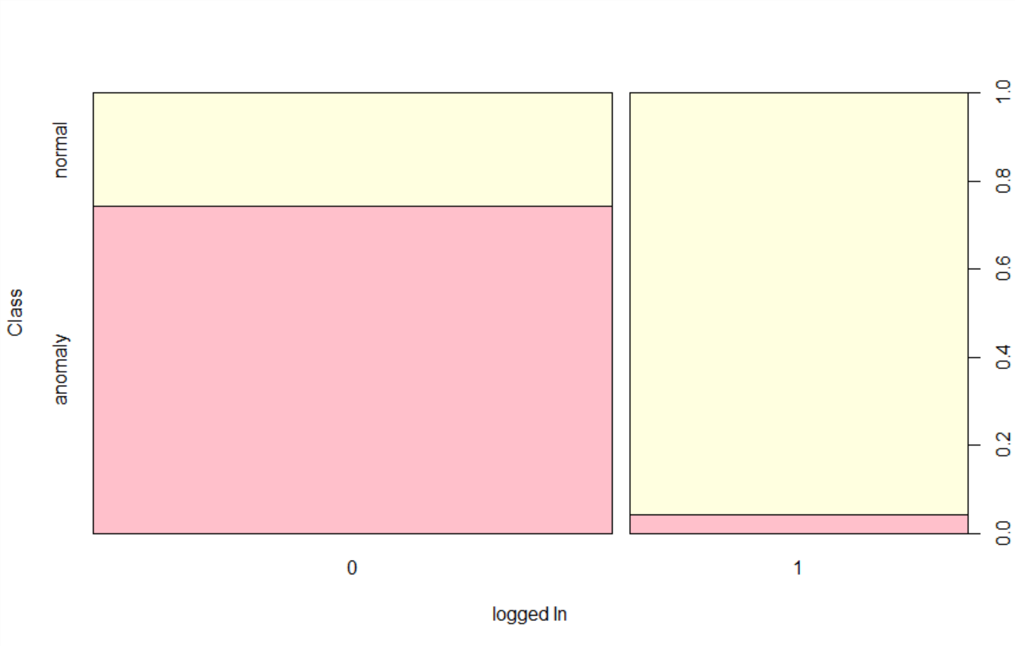

```
plot(nw_train$service, nw_train$class, main="Service vs Class",xlab="Service"
, ylab="Class", col=c("blue","grey"))
```



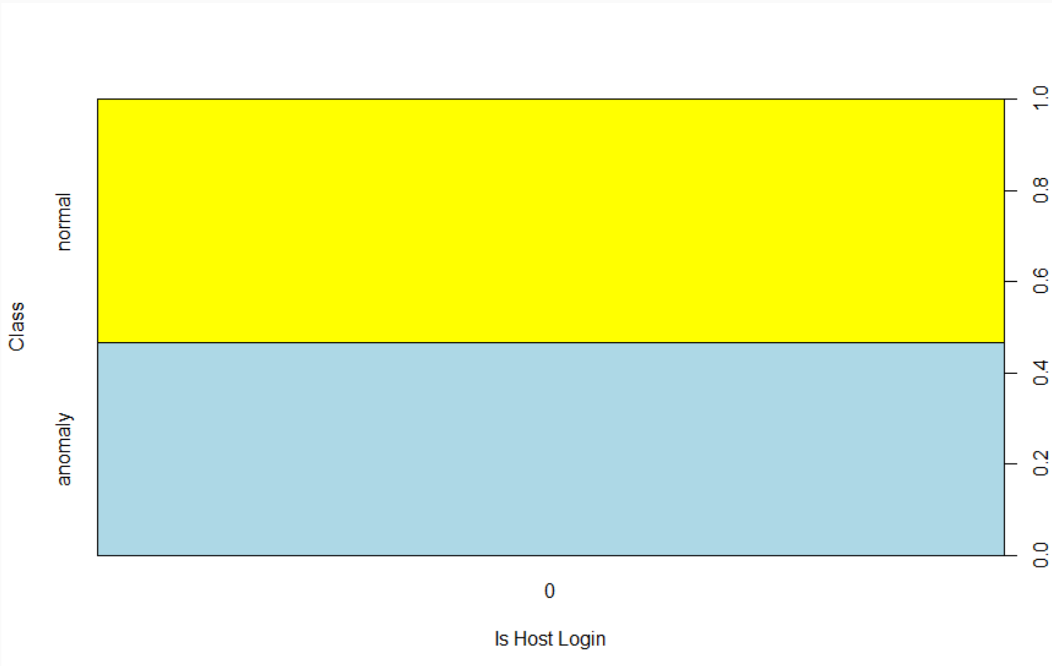

```
plot(nw_train$flag, nw_train$class, main="Flag vs Class",xlab="Flag", ylab="C
lass", col=c("brown","lightblue"))
```



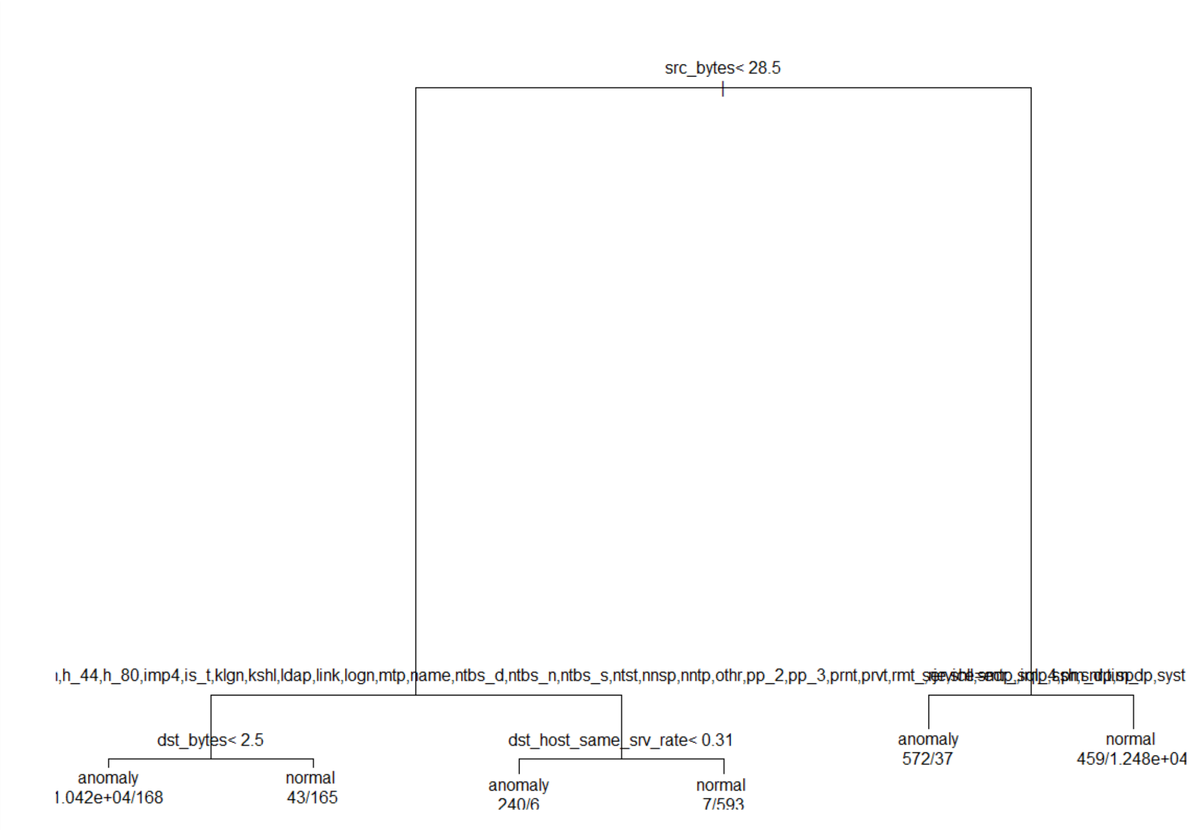

```
plot(as.factor(nw_train$logged_in),nw_train$class, xlab="logged In", ylab="Cl
ass", col=c("pink","lightyellow"))
```



```
plot(as.factor(nw_train$is_host_login),nw_train$class, xlab="Is Host Login",
ylab="Class", col=c("lightblue","yellow"))"))
```

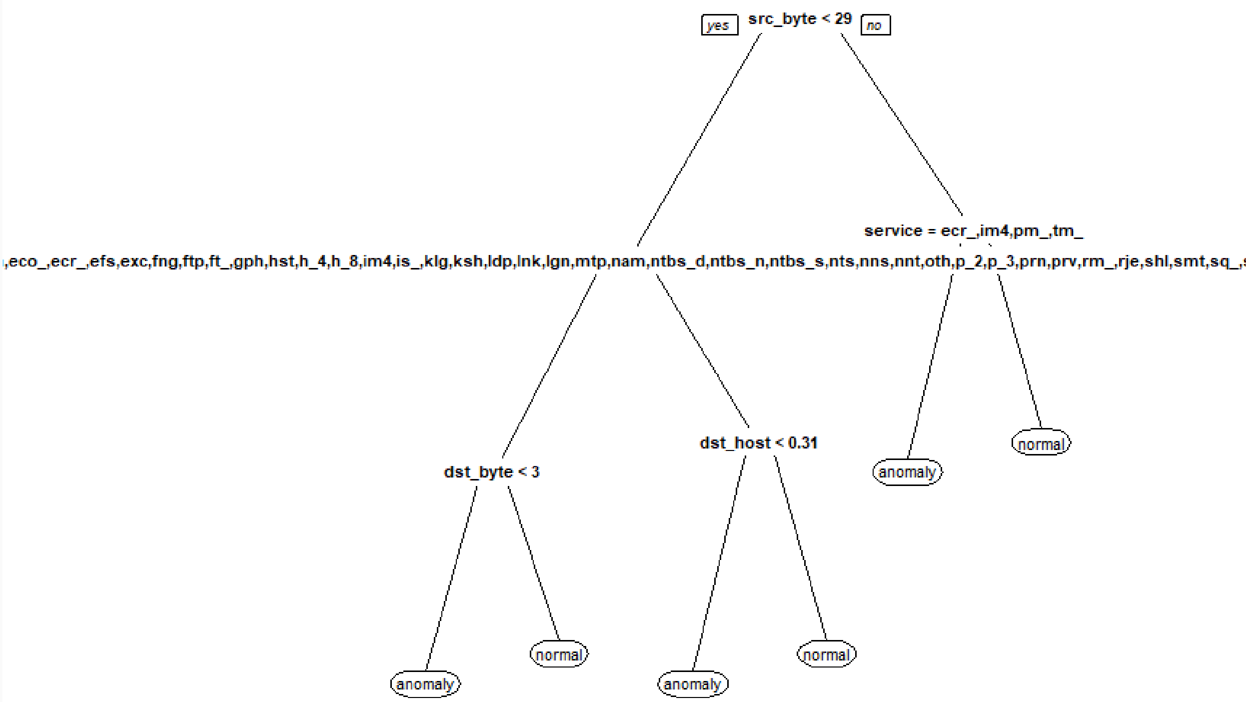



```
#Create the cart model using rpart
library(rpart)

cart_mod<-rpart(class~.,data=nw_train, method="class")
summary(cart_mod)

#Global Environment
Large rpart(15 elements, 1.8Mb)

# Plotting Decision Tree
plot(cart_mod,margin = 0.01)
text(cart_mod, use.n = T, pretty = T, cex=1)
```




```
#Lets do the predictions on the validation dataset using cart model


#Resolve error

levels(nw_valid$service)=levels(nw_train$service)

# Perform the prediction on the validation dataset using the model on decisio
n tree

pred_on_valid<-predict(cart_mod, newdata = nw_valid, type = "class")

pred_on_valid

table(nw_valid$class, pred_on_valid)


# calculate the accuracy ratio

accuracy_ratio_of_cart_mod<-(7672+9380)/(7672+331+5161+9380)

accuracy_ratio_of_cart_mod #<-[1] 0.7563875 accuracy
```

```
#Console:
#> table(nw_valid$class, pred_on_valid)
#         pred_on_valid
#          anomaly normal
#  anomaly    7672   5161
#  normal      331   9380
#> accuracy_ratio_of_cart_mod<-(7672+9380)/(7672+331+5161+9380)
#> accuracy_ratio_of_cart_mod
#[1] 0.7563875

# Perform the prediction for test dataset
levels(nw_test$service)=levels(nw_train$service)
pred_on_nw_test<-predict(cart_mod,newdata = nw_test, typr="class")
pred_on_nw_test

table(pred_on_nw_test)
#Console:
#> table(pred_on_nw_test)
#pred_on_nw_test
#0.0116666666666667 0.0158640226628895  0.024390243902439 0.0354741479248783
0.0607553366174056  0.206730769230769
#              263               7066                154              12560
783              1718
# 0.793269230769231  0.939244663382594  0.964525852075122  0.975609756097561
0.984135977337111  0.988333333333333
#             1718                783              12560                154
7066               263

# Alternate method of plotting
library(rpart.plot)
library(RColorBrewer)
prp(cart_mod)
```



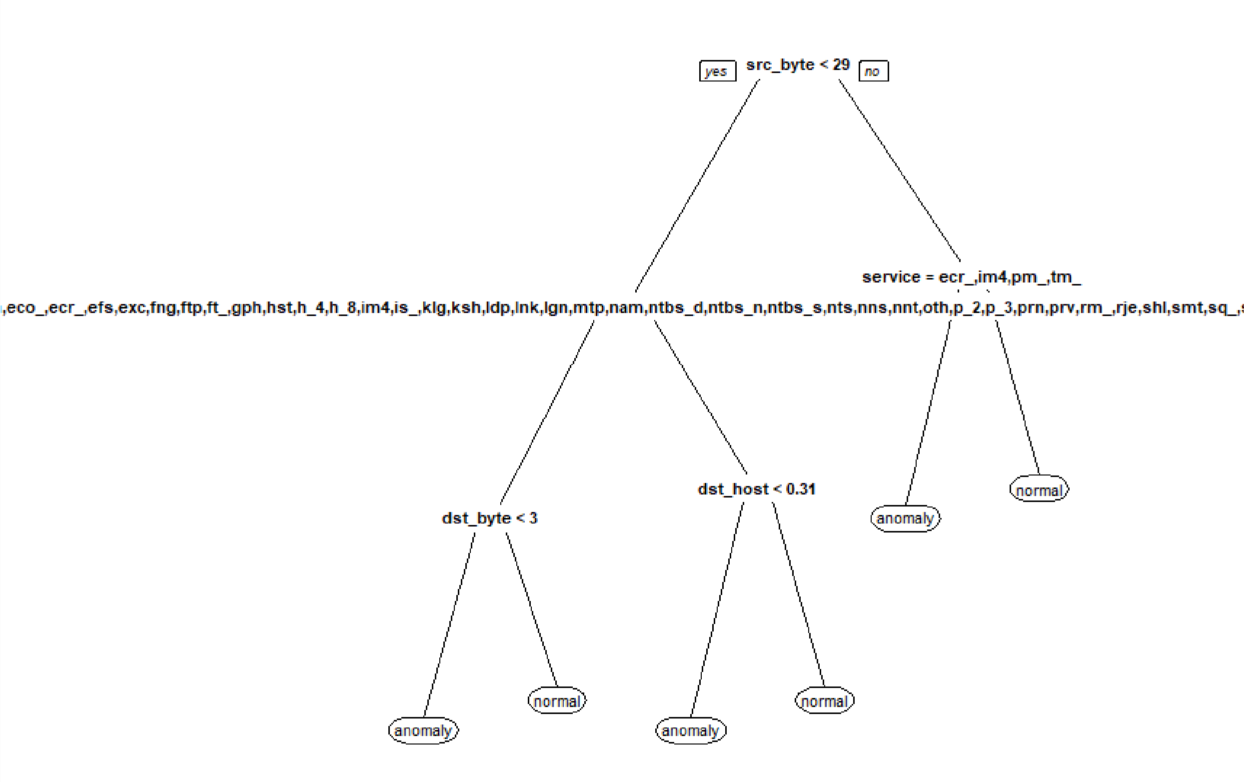

```
#find out the cp parameter  corresponding to the least cross validation error


printcp(cart_mod)


#Console:
#> printcp(cart_mod)
#Classification tree:
#rpart(formula = class ~ ., data = nw_train, method = "class")
#Variables actually used in tree construction:
#[1] dst_bytes              dst_host_same_srv_rate service                  src
_bytes
#Root node error: 11743/25192 = 0.46614
#n= 25192
#         CP nsplit rel error   xerror      xstd
#1 0.832837      0   1.000000 1.000000 0.0067425
#2 0.045559      1   0.167163 0.167163 0.0036230
```

```
#3 0.029975       2   0.121604 0.121604 0.0031255
#4 0.019927       3   0.091629 0.091714 0.0027343
#5 0.010389       4   0.071702 0.071873 0.0024322
#6 0.010000       5   0.061313 0.065912 0.0023325
```

1.4Tree Model pruning

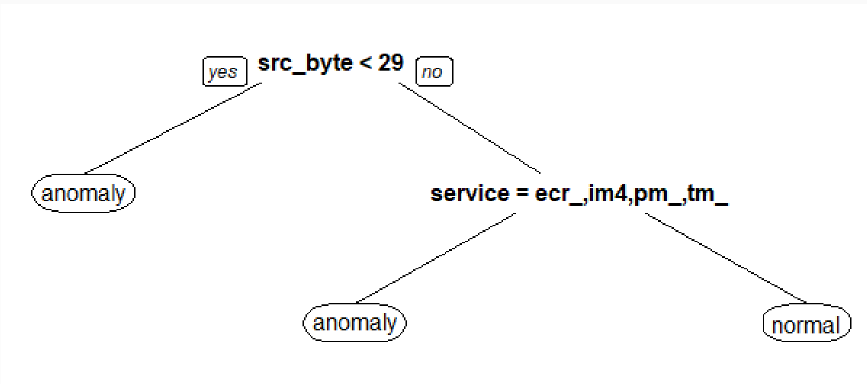
The technique of setting constraint is a greedy-approach. In other words, it will check for the best split instantaneously and move forward until one of the specified stopping condition is reached

```
#Prune the tree using cp parameter corresponding to the least cross validatio
n error

cart_mod_1<-prune(cart_mod, cp=0.01)

prp(cart_mod_1)
```



```
# prediction on validation data using pruned tree
pred_on_valid_1<-predict(cart_mod_1, newdata = nw_valid, type = "class")
table(pred_on_valid_1)
table(nw_valid$class,pred_on_valid_1)
accuracy_model_1<-(7672+9380)/(7672+331+5161+9380)
accuracy_model_1 #[1] 0.7563875

#Console:
#> table(pred_on_valid_1)
#pred_on_valid_1
#anomaly  normal
#   8003   14541
#> table(nw_valid$class,pred_on_valid_1)
#         pred_on_valid_1
#          anomaly normal
#  anomaly    7672   5161
#  normal      331   9380
#> accuracy_model_1<-(7672+9380)/(7672+331+5161+9380)
#> accuracy_model_1
#[1] 0.7563875

# There is no change in model by default model using the least cross validati
on error and the corresponding cp parameters
```

```
# create a new model by using a different cp parameter
cart_mod_2<-prune(cart_mod, cp=0.045559)
prp(cart_mod_2)
```




```
# Prediction on validation dataset using model2
pred_on_valid_2<-predict(cart_mod_2,newdata = nw_valid, type = "class")
table(pred_on_valid_2)
table(nw_valid$class, pred_on_valid_2)

accuracy_model_2<-(9468+9195)/(9768+516+3365+9195)
accuracy_model_2 #[1] 0.816976

#Console:
#> table(pred_on_valid_2)
#pred_on_valid_2
#anomaly  normal
#   9984   12560
#> table(nw_valid$class, pred_on_valid_2)
#         pred_on_valid_2
#          anomaly normal
#  anomaly    9468   3365
#  normal      516   9195
```

```
#> accuracy_model_2<-(9468+9195)/(9768+516+3365+9195)
#> accuracy_model_2
#[1] 0.816976

# Prediction on test dataset using model2
pred_on_nw_test_1<-predict(cart_mod_2,newdata = nw_test, type = "class")
pred_on_nw_test_1
table(pred_on_nw_test_1)

results<-data.frame(Duration=nw_test$duration, Protocal_type=nw_test$protocol_type, Service=nw_test$service,
                    flag=nw_test$flag, Predicted_class=pred_on_nw_test_1)

head(results,10)

#>Console:
#> table(pred_on_nw_test_1)
#pred_on_nw_test_1
#anomaly  normal
#   9984   12560
#> results<-data.frame(Duration=nw_test$duration, Protocal_type=nw_test$protocol_type, Service=nw_test$service,
+                      flag=nw_test$flag, Predicted_class=pred_on_nw_test_1)
#> head(results,10)
#   Duration Protocal_type  Service flag Predicted_class
#1         0           tcp  printer  REJ         anomaly
#2         0           tcp  printer  REJ         anomaly
#3         2           tcp ftp_data   SF          normal
#4         0          icmp    eco_i   SF         anomaly
#5         1           tcp   supdup RSTO         anomaly
#6         0           tcp     http   SF          normal
#7         0           tcp      rje   SF          normal
```

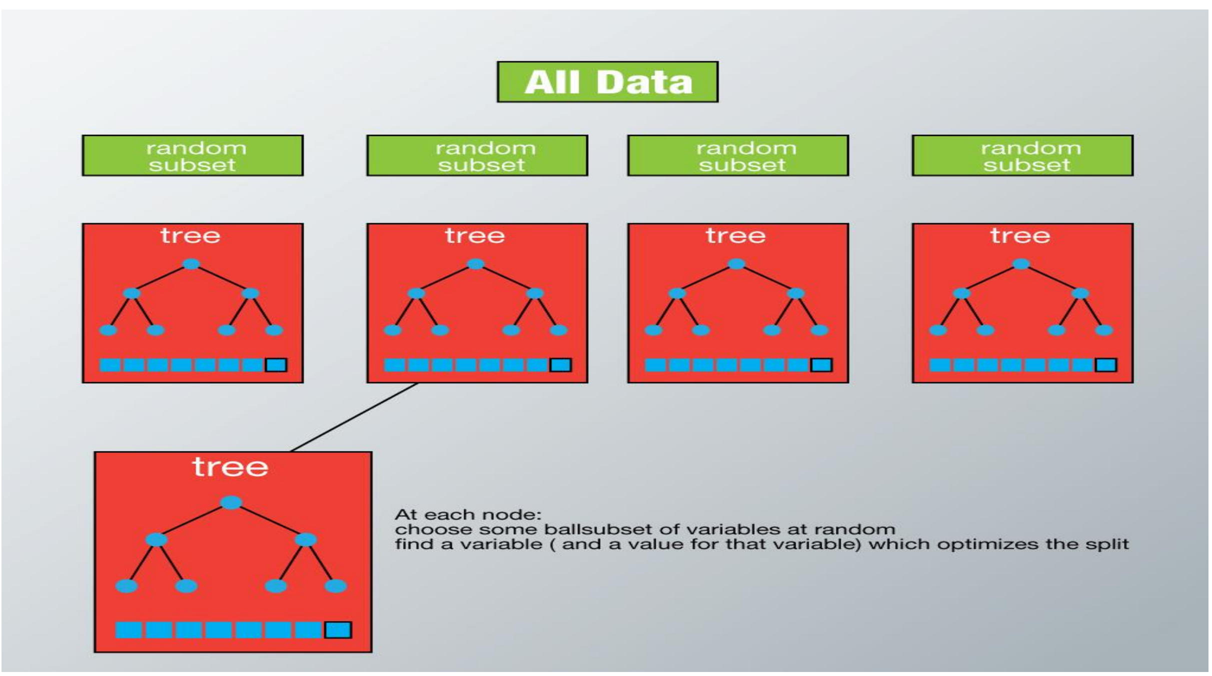
```
#8          0          tcp   supdup   SF          normal
#9          0          tcp     http   SF          normal
#10         0          tcp      ftp   SF         anomaly
```

1.5 Random Forest

Random Forest is considered to be a panacea of all data science problems. On a funny note, when you can’t think of any algorithm (irrespective of situation), use random forest!

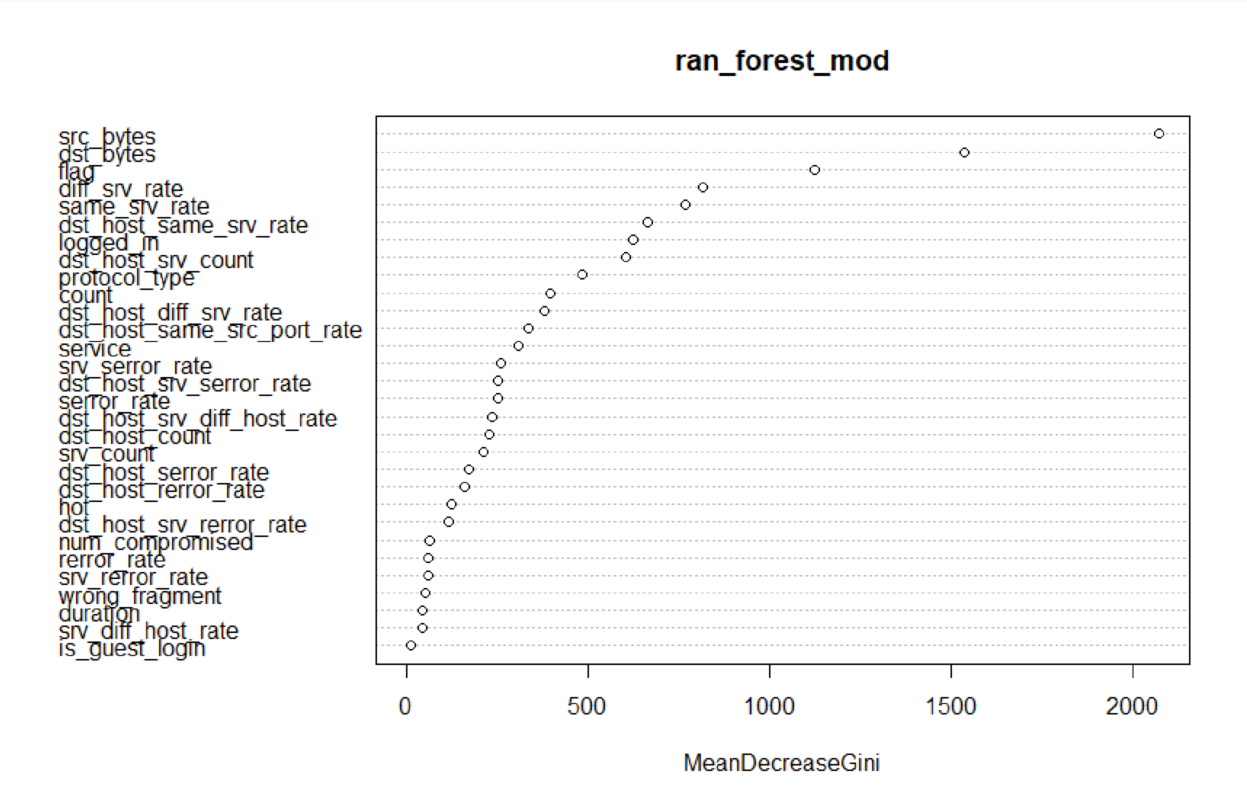
Random Forest is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model.
It works in the following manner. Each tree is planted & grown as follows:

1. Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<M is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.
3. Each tree is grown to the largest extent possible and there is no pruning.
4. Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).



```
#Random Forest (Ensemble learning)

install.packages("randomForest")
library(randomForest)

levels(nw_test$service)=levels(nw_train$service)
ran_forest_mod<-randomForest(class~.,data = nw_train, method="class")

#Random forest cannot handle more than 53 categorical predictors

str(nw_train)
nw_train$service<-as.numeric(nw_train$service)
nw_test$service<-as.numeric(nw_test$service)
nw_valid$service<-as.numeric(nw_valid$service)

ran_forest_mod<-randomForest(class~.,data = nw_train, method="class")

# Apply Random forest model on validation dataset
pred_rnd_forest<-predict(ran_forest_mod,newdata=nw_valid,type="class")

# Create the confusion matrix
table(nw_valid$class, pred_rnd_forest)

# Find the accuracy of random forest model
ran_forest_mod_accuracy<-(8153+9441)/(8153+270+4680+9441)
ran_forest_mod_accuracy #[1] 0.7804294

#Console:
# >  Create the confusion matrix
#> table(nw_valid$class, pred_rnd_forest)
```

```
#         pred_rnd_forest
#          anomaly normal
#  anomaly    8167   4666
#  normal      274   9437

#> ran_forest_mod_accuracy<-(8153+9441)/(8153+270+4680+9441)
#> ran_forest_mod_accuracy
#[1] 0.7804294

# Identification of important variables
varImpPlot(ran_forest_mod)
```



```
# Do the prediction of Important variables
pred_rnd_forest_test_data<-predict(ran_forest_mod, newdata = nw_test,type = "
class")


table(pred_rnd_forest_test_data)


# Store the results in new data frame called Network intrusion Random Forest
Network_Intution_RF<-data.frame(Duration=nw_test$duration,Protocol_Type=nw_te
st$protocol_type,Service=as.factor(nw_test$service),Flag=nw_test$flag,Predict
ed_class=pred_rnd_forest_test_data)
head(Network_Intution_RF)


write.csv(Network_Intution_RF, "Network_Anamoly_Detection_Random_forest.csv",
row.names = F)


#Console:
#> table(pred_rnd_forest_test_data)
#pred_rnd_forest_test_data
#anomaly  normal
#   8421   14123
#> head(Network_Intution_RF)
#  Duration Protocol_Type Service Flag Predicted_class
#1        0          tcp      44  REJ         anomaly
#2        0          tcp      44  REJ         anomaly
#3        2          tcp      17   SF          normal
#4        0         icmp      11   SF         anomaly
#5        1          tcp      54 RSTO          normal
#6        0          tcp      20   SF          normal
```


1.6 Conclusion

Hence Decision tree model having good accuracy score as 81.69% and Random Forest model got an accuracy as 78%

Both the models are best fit to the case study.