#install.packages('neuralnet', dependencies=TRUE, repos='http://cran.rstudio.com/')

#rf classification YES/NO

df <- read.csv(file.choose(), header=TRUE)

data<-df

str(data)

str(data)

#data$X<-NULL

data$Class = as.factor(data$Class)

table(data$Class)

dt.data <-data

str(dt.data)

table(dt.data$Class)

library(randomForest)

library(caret)

set.seed(123)

inTrain <- createDataPartition(

y = dt.data$Class,

p = .67,

list = FALSE

)

training <- dt.data[ inTrain,]

testing <- dt.data[-inTrain,]

ctrl <- trainControl(method = "repeatedcv", repeats = 1, number = 10)

rfFit <- train(

Class ~ .,

data = training,

method = "rf",

#------->>>> also can use method= "ranger",

tuneLength = 5,

## added:

trControl = ctrl

)

plsClasses <- predict(rfFit, newdata = testing)

results <- data.frame(actual = testing$Class,prediction = plsClasses)

results$prediction <- round(results$prediction, 0)

results

actual = testing$Class

prediction = plsClasses

table\_mat <- table(actual, prediction)

table\_mat

confusionMatrix(prediction, testing$Class)

plot(rfFit, main = "Random forest")

confusionMatrix(table(results))

#SVm Classifer

#SVM

# SVM binary class

data <- read.csv(file.choose(), header=TRUE)

library(caret) # use for train and traincntrl function

library(e1071)

svm.data <- data # undersample dataset

str(svm.data)

svm.data$Class <- as.factor(svm.data$Class)

str(svm.data)

table(svm.data$Class)

set.seed(3)

trainIndex <- createDataPartition(svm.data$Class, p=0.67, list=FALSE)

data\_train <- svm.data[trainIndex,]

data\_test <- svm.data[-trainIndex,]

# to install any R package install.packages('package\_name', dependencies=TRUE, repos='http://cran.rstudio.com/')

dim(svm.data)

summary(svm.data)

model <- svm(Class ~ ., data=data\_train, method="C-classification", kernel="linear", gamma=0.2)

pr = predict(model, type = 'response', newdata = data\_test)

summary(model)

summary(pr)

table(pr, y)

results <- data.frame(actual = data\_test$Class,prediction = pr)

results

results$prediction <- round(results$prediction, 0)

results

table(results)

library(caret)

confusionMatrix(data\_test$Class, pr)

#Neural Network.

data\_down<-read.csv(file.choose(), header=TRUE)

library(neuralnet)

library(caret)

data\_down$X<-NULL

dataset <- data\_down

str(dataset)

library(neuralnet)

str(dataset)

head(dataset)

set.seed(123)

smp\_size <- floor(0.8 \* nrow(dataset))

train\_ind <- sample(seq\_len(nrow(dataset)), size = smp\_size)

trainset <- dataset[train\_ind, ]

testset <- dataset[-train\_ind, ]

maxs <- apply(dataset, 2, max)

mins <- apply(dataset, 2, min)

scaled <- as.data.frame(scale(dataset, center = mins, scale = maxs - mins)) #work when all attributes are numeric

trainNN = scaled[train\_ind , ]

testNN = scaled[-train\_ind , ]

library(doParallel)

registerDoParallel(Cores =8)

creditnet <- neuralnet(Class ~ SessionID + Timestamp + ItemID + Category, trainNN,hidden = 2, lifesign = "minimal",linear.output = FALSE, threshold = 0.1, act.fct="tanh", stepmax=1e+07)

temp\_test <- subset(testNN, select = c("SessionID", "Timestamp", "ItemID", "Category"))

creditnet.results <- compute(creditnet, temp\_test)

head(temp\_test)

results <- data.frame(actual = testNN$Class,prediction = creditnet.results$net.result)

results

results$prediction <- round(results$prediction, 0)

results

table(results)

library(caret)

confusionMatrix(table(results))

# For Methew Corelation Calculation

library(mccr)

mccr(actual,prediction)

# For F-measure Calculation

#cm <- as.matrix(table(testing$Category, plsClasses))

Eval = function(actual=NULL, predicted=NULL, cm=NULL){

n = sum(cm) # number of instances

nc = nrow(cm) # number of classes

diag = diag(cm) # number of correctly classified instances per class

rowsums = apply(cm, 1, sum) # number of instances per class

colsums = apply(cm, 2, sum) # number of predictions per class

p = rowsums / n # distribution of instances over the classes

q = colsums / n # distribution of instances over the predicted classes

#accuracy

accuracy = sum(diag) / n

#per class prf

recall = diag / rowsums

precision = diag / colsums

f1 = 2 \* precision \* recall / (precision + recall)

#macro prf

macroPrecision = mean(precision)

macroRecall = mean(recall)

macroF1 = mean(f1)

#1-vs-all matrix

oneVsAll = lapply(1 : nc,

function(i){

v = c(cm[i,i],

rowsums[i] - cm[i,i],

colsums[i] - cm[i,i],

n-rowsums[i] - colsums[i] + cm[i,i]);

return(matrix(v, nrow = 2, byrow = T))})

s = matrix(0, nrow=2, ncol=2)

for(i in 1:nc){s=s+oneVsAll[[i]]}

#avg accuracy

avgAccuracy = sum(diag(s))/sum(s)

#micro prf

microPrf = (diag(s) / apply(s,1, sum))[1];

#majority class

mcIndex = which(rowsums==max(rowsums))[1] # majority-class index

mcAccuracy = as.numeric(p[mcIndex])

mcRecall = 0\*p; mcRecall[mcIndex] = 1

mcPrecision = 0\*p; mcPrecision[mcIndex] = p[mcIndex]

mcF1 = 0\*p; mcF1[mcIndex] = 2 \* mcPrecision[mcIndex] / (mcPrecision[mcIndex] + 1)

#random/expected accuracy

expAccuracy = sum(p\*q)

#kappa

kappa = (accuracy - expAccuracy) / (1 - expAccuracy)

#random guess

rgAccuracy = 1 / nc

rgPrecision = p

rgRecall = 0\*p + 1 / nc

rgF1 = 2 \* p / (nc \* p + 1)

#random weighted guess

rwgAccurcy = sum(p^2)

rwgPrecision = p

rwgRecall = p

rwgF1 = p

classNames = names(diag)

if(is.null(classNames)) classNames = paste("C",(1:nc),sep="")

metrics = rbind(

Accuracy = accuracy,

Precision = precision,

Recall = recall,

F1 = f1,

MacroAvgPrecision = macroPrecision,

MacroAvgRecall = macroRecall,

MacroAvgF1 = macroF1,

AvgAccuracy = avgAccuracy,

MicroAvgPrecision = microPrf,

MicroAvgRecall = microPrf,

MicroAvgF1 = microPrf,

MajorityClassAccuracy = mcAccuracy,

MajorityClassPrecision = mcPrecision,

MajorityClassRecall = mcRecall,

MajorityClassF1 = mcF1,

Kappa = kappa,

RandomGuessAccuracy = rgAccuracy,

RandomGuessPrecision = rgPrecision,

RandomGuessRecall = rgRecall,

RandomGuessF1 = rgF1,

RandomWeightedGuessAccuracy = rwgAccurcy,

RandomWeightedGuessPrecision = rwgPrecision,

RandomWeightedGuessRecall = rwgRecall,

RandomWeightedGuessF1 = rwgF1)

colnames(metrics) = classNames

return(list(ConfusionMatrix = cm, Metrics = metrics))

}

cm<-as.matrix(table(actual,prediction))

Eval(actual,prediction,cm)

mccr(actual,prediction)