# Train a Random Forest

library(randomForest)

library(rpart)

library(caret)

library(e1071)

library(ipred)

library(caret)

library(Metrics)

library(readr)

library(gbm)

library(dplyr)

library(randomForest)

library(ROCR)

?randomForest

set.seed(1) # for reproducibility

# credit\_train\_r=credit\_train

#

# library(dplyr)

# credit\_train\_r=credit\_train %>% mutate\_if(is.character, as.factor)

library(randomForest)

credit\_model\_randomForest <- randomForest(formula = default ~ .,

data = credit\_train)

# Print the model output

print(credit\_model\_randomForest)

names(credit\_model\_randomForest)

head(credit\_model\_randomForest$predicted)

head(credit\_model\_randomForest$importance)

credit\_model\_randomForest$classes

# Generate predicted classes using the model object

# Grab OOB error matrix & take a look

err <- credit\_model\_randomForest$err.rate

head(err)

tail(err)

# Look at final OOB error rate (last row in err matrix)

oob\_err <- err[nrow(err), "OOB"]

print(oob\_err)

# Plot the model trained in the previous exercise

plot(credit\_model\_randomForest)

# Add a legend since it doesn't have one by default

legend(x = "bottom",

legend = colnames(err),

fill = 1:ncol(err))

# Generate predicted classes using the model object

pred\_randomForest <- predict(object = credit\_model\_randomForest, # model object

newdata = credit\_test, # test dataset

type = "class") # return classification labels

# Calculate the confusion matrix for the test set

cm <- confusionMatrix(data = pred\_randomForest, # predicted classes

reference = credit\_test$default) # actual classes

print(cm)

# Compare test set accuracy to OOB accuracy

paste0("Test Accuracy: ", cm$overall[1])

paste0("OOB Accuracy: ", 1 - oob\_err)

# Generate predictions on the test set

pred\_randomForest1 <- predict(object = credit\_model\_randomForest,

newdata = credit\_test,

type = "prob")

# `pred` is a matrix

class(pred)

# Look at the pred format

head(pred)

# Compute the AUC (`actual` must be a binary 1/0 numeric vector)

auc\_randomForest=auc(actual = ifelse(credit\_test$default == "yes", 1, 0),

predicted = pred\_randomForest1[,"yes"])

auc\_randomForest

# Execute the tuning process

set.seed(1)

res <- tuneRF(x = subset(credit\_train, select = -default),

y = credit\_train$default,

ntreeTry = 500)

print(res)

# Establish a list of possible values for mtry, nodesize and sampsize

# Look at results

print(res)

# Find the mtry value that minimizes OOB Error

mtry\_opt <- res[,"mtry"][which.min(res[,"OOBError"])]

print(mtry\_opt)

mtry <- seq(4, ncol(credit\_train) \* 0.8, 2)

mtry

nodesize <- seq(3, 8, 2)

sampsize <- nrow(credit\_train) \* c(0.7, 0.8)

# Create a data frame containing all combinations

hyper\_grid <- expand.grid(mtry = mtry, nodesize = nodesize, sampsize = sampsize)

head(hyper\_grid)

# Create an empty vector to store OOB error values

oob\_err <- c()

# Write a loop over the rows of hyper\_grid to train the grid of models

for (i in 1:nrow(hyper\_grid)) {

# Train a Random Forest model

model <- randomForest(formula = default ~ .,

data = credit\_train,

mtry = hyper\_grid$mtry[i],

nodesize = hyper\_grid$nodesize[i],

sampsize = hyper\_grid$sampsize[i])

# Store OOB error for the model

oob\_err[i] <- model$err.rate[nrow(model$err.rate), "OOB"]

}

# Identify optimal set of hyperparmeters based on OOB error

opt\_i <- which.min(oob\_err)

opt\_i

min(oob\_err)

print(hyper\_grid[opt\_i,])

credit\_model\_randomForest\_update <- randomForest(formula = default ~ .,

data = credit\_train,

mtry = 6,

nodesize =3,

sampsize = 560)

pred\_randomForest\_update<-predict(credit\_model\_randomForest\_update,credit\_test,method="class")

pred\_randomForest\_update1<-predict(credit\_model\_randomForest\_update,credit\_test,method="prob")

head(pred\_randomForest\_update1)

cm\_RandomForest\_update <- confusionMatrix(data = pred\_randomForest\_update, # predicted classes

reference = credit\_test$default) # actual classes

print(cm\_RandomForest\_update)

auc\_randomForest\_update1=auc(actual = ifelse(credit\_test$default == "yes", 1, 0),

predicted = pred\_randomForest\_update1)

auc\_randomForest\_update1

summary(credit\_model\_randomForest)