4\_GBM.R

setwd("")  
library(Metrics)  
########################################################################  
#Definition Gradient Boosting Machine (G.B.M)  
########################################################################  
#QUESTION : What is the main difference between bagged trees and boosted trees?  
#RESPONSE : Boosted trees improve the model fit by considering past fits and bagged trees do not.  
#COMMENT : Boosting is an iterative algorithm that considers past fits to improve performance.  
credit <- read.csv("credit.csv", stringsAsFactors = TRUE)  
########################################################################  
#Split data in 80% 20%  
########################################################################  
# Total number of rows in the credit data frame  
n <- nrow(credit)  
# Number of rows for the training set (80% of the dataset)  
n\_train <- round(0.8 \* n)   
# Create a vector of indices which is an 80% random sample  
set.seed(123)  
train\_indices <- sample(1:n, n\_train)  
# Subset the credit data frame to training indices only  
credit\_train <- credit[train\_indices, ]   
# Exclude the training indices to create the test set  
credit\_test <- credit[-train\_indices, ]  
  
########################################################################  
#Train a GBM model  
########################################################################  
#gbm() function to train a GBM classifier to predict loan default  
#You will train a 10,000-tree GBM on the credit\_train  
  
#Using such a large number of trees (10,000) is probably not optimal for a GBM   
#but we will build more trees than we need and then select the optimal number of trees based on early performance-based stopping.   
  
#For binary classification, gbm() requires the response to be encoded as 0/1 (numeric)  
#so we will have to convert from a "no/yes" factor to a 0/1 numeric   
  
#gbm() function requires the user to specify a distribution argument  
#For a binary classification problem, you should set distribution = "bernoulli"  
  
# Convert "yes" to 1, "no" to 0  
credit\_train$default <- ifelse(credit\_train$default == "yes", 1, 0)  
  
library(gbm)

## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

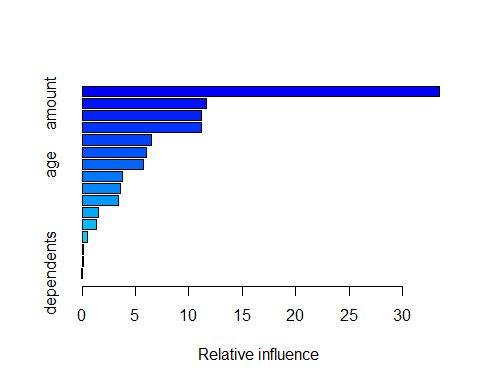
## Loading required package: parallel

## Loaded gbm 2.1.3

# Train a 10000-tree GBM model  
set.seed(1)  
credit\_model <- gbm(formula = default ~ .,   
 distribution = "bernoulli",   
 data = credit\_train,  
 n.trees = 10000)  
  
# Print the model object   
print(credit\_model)

## gbm(formula = default ~ ., distribution = "bernoulli", data = credit\_train,   
## n.trees = 10000)  
## A gradient boosted model with bernoulli loss function.  
## 10000 iterations were performed.  
## There were 16 predictors of which 16 had non-zero influence.

# summary() prints variable importance  
summary(credit\_model)



## var rel.inf  
## checking\_balance checking\_balance 33.49502510  
## amount amount 11.62938098  
## months\_loan\_duration months\_loan\_duration 11.17113439  
## credit\_history credit\_history 11.15698321  
## savings\_balance savings\_balance 6.44293358  
## employment\_duration employment\_duration 6.06266137  
## age age 5.73175696  
## percent\_of\_income percent\_of\_income 3.74219743  
## other\_credit other\_credit 3.56695375  
## purpose purpose 3.38820798  
## housing housing 1.55169398  
## years\_at\_residence years\_at\_residence 1.35255308  
## job job 0.47631930  
## phone phone 0.09142691  
## existing\_loans\_count existing\_loans\_count 0.08924265  
## dependents dependents 0.05152933

########################################################################  
#Prediction using a GBM model  
########################################################################  
# Since we converted the training response col, let's also convert the test response col  
credit\_test$default <- ifelse(credit\_test$default == "yes", 1, 0)  
  
# Generate predictions on the test set  
preds1 <- predict.gbm(object = credit\_model,   
 newdata = credit\_test,  
 n.trees = 10000)  
  
# Generate predictions on the test set (scale to response)  
preds2 <- predict.gbm(object = credit\_model,   
 newdata = credit\_test,  
 n.trees = 10000,  
 type = "response")  
  
# Compare the range of the two sets of predictions  
range(preds1)

## [1] -3.210354 2.088293

range(preds2)

## [1] 0.03877796 0.88976007

########################################################################  
#Evaluate test set AUC  
########################################################################  
#Compute test set AUC of the GBM model for the two sets of predictions  
#We will notice that they are the same value.  
#AUC is a rank-based metric, so changing the actual values does not change the value of the AUC.  
# Generate the test set AUCs using the two sets of preditions & compare  
auc(actual = credit\_test$default, predicted = preds1) #default

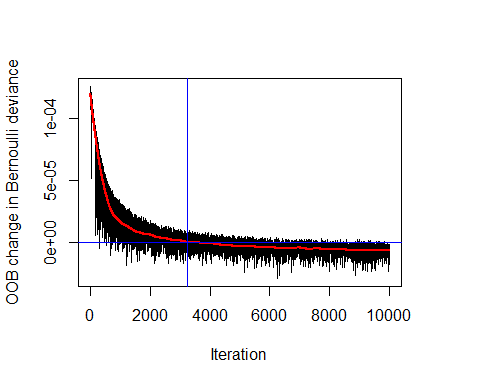
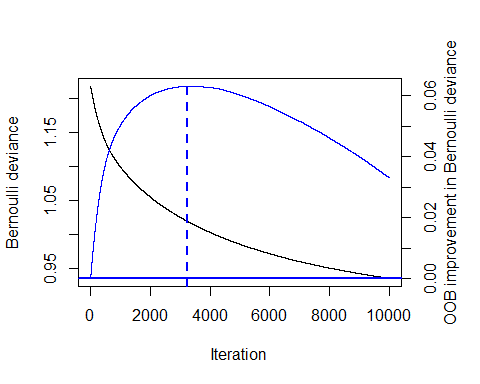
## [1] 0.7875175

auc(actual = credit\_test$default, predicted = preds2) #rescaled

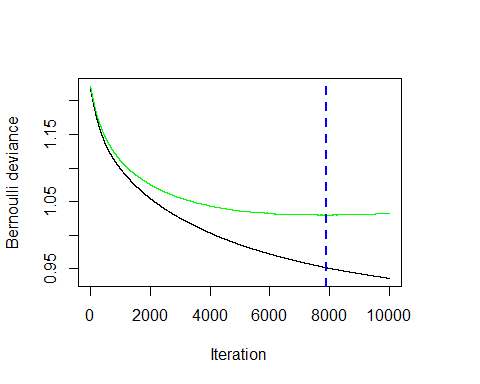
## [1] 0.7875175

########################################################################  
#Early stopping in GBMs  
########################################################################  
#Use the gbm.perf() function to estimate the optimal number of boosting   
#(aka n.trees)  
#using both OOB and CV error  
  
#When you set out to train a large number of trees in a GBM (such as 10,000)   
#you use a validation method to determine an earlier (smaller) number of trees, then that's called "early stopping"  
  
# Optimal ntree estimate based on OOB  
ntree\_opt\_oob <- gbm.perf(object = credit\_model,   
 method = "OOB",   
 oobag.curve = TRUE)

## Warning in gbm.perf(object = credit\_model, method = "OOB", oobag.curve  
## = TRUE): OOB generally underestimates the optimal number of iterations  
## although predictive performance is reasonably competitive. Using cv.folds>0  
## when calling gbm usually results in improved predictive performance.



#get the optimal number of trees based on the OOB error and store that number  
#print(ntree\_opt\_oob)  
#value = 3233  
  
# Train a CV GBM model  
set.seed(1)  
credit\_model\_cv <- gbm(formula = default ~ .,   
 distribution = "bernoulli",   
 data = credit\_train,  
 n.trees = 10000,  
 cv.folds = 2)  
  
# Optimal ntree estimate based on CV  
ntree\_opt\_cv <- gbm.perf(object = credit\_model\_cv,   
 method = "cv")



# Compare the estimates   
print(paste0("Optimal n.trees (OOB Estimate): ", ntree\_opt\_oob))

## [1] "Optimal n.trees (OOB Estimate): 3233"

print(paste0("Optimal n.trees (CV Estimate): ", ntree\_opt\_cv))

## [1] "Optimal n.trees (CV Estimate): 7889"

########################################################################  
#OOB vs CV-based early stopping  
########################################################################  
#Between OOB and CV compare the performance of the models on a test set  
# Generate predictions on the test set using ntree\_opt\_oob number of trees  
preds1 <- predict(object = credit\_model,   
 newdata = credit\_test,  
 n.trees = ntree\_opt\_oob)  
  
# Generate predictions on the test set using ntree\_opt\_cv number of trees  
preds2 <- predict(object = credit\_model,   
 newdata = credit\_test,  
 n.trees = ntree\_opt\_cv)   
#saveRDS(preds2, file = "gbm\_preds")  
  
library(Metrics)  
# Generate the test set AUCs using the two sets of preditions & compare  
auc1 <- auc(actual = credit\_test$default, predicted = preds1) #OOB  
auc2 <- auc(actual = credit\_test$default, predicted = preds2) #CV   
  
# Compare AUC   
print(paste0("Test set AUC (OOB): ", auc1))

## [1] "Test set AUC (OOB): 0.777816736792894"

print(paste0("Test set AUC (CV): ", auc2))

## [1] "Test set AUC (CV): 0.785530621785881"

#Cross-validation's early stop is slightly better than OOB's error.  
  
########################################################################  
#Best GBM model  
########################################################################