## Daniel Sanchez

## BAN 502

## Dr. Stephen Hill

## Module 5 Assignment 1

## June 22nd, 2020

library(tidyverse)  
library(caret)  
library(nnet)  
library(rpart)  
library(ranger)  
library(caretEnsemble)  
library(xgboost)

fin=read\_csv("2018Fin.csv")

## Warning: Missing column names filled in: 'X1' [1]

## Parsed with column specification:  
## cols(  
## .default = col\_double(),  
## X1 = col\_character(),  
## Sector = col\_character()  
## )

## See spec(...) for full column specifications.

#str(fin)  
#summary(fin)  
fin=fin %>%select(Class,`Revenue Growth`,`EPS Diluted`,`EBITDA Margin`,priceBookValueRatio,debtEquityRatio,debtRatio,`PE ratio`,Sector,`5Y Revenue Growth (per Share)`,returnOnAssets,returnOnEquity,returnOnCapitalEmployed,quickRatio)  
  
fin = fin %>% mutate(Class = as\_factor(as.character(Class))) %>%  
mutate(Class = fct\_recode(Class,  
"No" = "0",  
"Yes" = "1"))  
  
fin = fin %>% mutate(Sector = as\_factor(as.character(Sector)))  
  
fin = fin %>% drop\_na()  
  
fin = fin %>% filter(`Revenue Growth` <= 1)  
fin = fin %>% filter(`EPS Diluted` >= -10, `EPS Diluted` <= 10)  
fin = fin %>% filter(`EBITDA Margin` >= -5, `EBITDA Margin` <= 5)  
fin = fin %>% filter(priceBookValueRatio >= 0, priceBookValueRatio <= 5)  
fin = fin %>% filter(debtEquityRatio >= -1, debtEquityRatio <= 2)  
fin = fin %>% filter(debtRatio <= 1)  
fin = fin %>% filter(`PE ratio` <= 100)  
fin = fin %>% filter(returnOnAssets >= -5, returnOnAssets <= 5)  
fin = fin %>% filter(returnOnEquity >= -5, returnOnEquity <= 5)  
fin = fin %>% filter(returnOnCapitalEmployed >= -2, returnOnCapitalEmployed <= 2)  
fin = fin %>% filter(quickRatio <= 20)

**Task 1: Split the data into training and testing sets. Your training set should have 70% of the data. Use a random number (set.seed) of 12345. Recall that your predictor variable is Class. Hint: Use dplyr::slice to avoid conflicts with xgboost package**

set.seed(12345)  
split = createDataPartition(y=fin$Class, p = .7, list = FALSE)  
train = dplyr::slice(fin,split)  
test = dplyr::slice(fin,-split)

**Task 2: Create a neural network to predict stock performance (given b the Class variable). Use a grid to search sizes 1 through an appropriate maximum (recall that you can get the “rule of thumb” size by counting the number of predictor variables, including factor levels) and decay rates of 0.5, 0.1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7. Use caret to implement 10-fold k-fold cross-validation. Use a random number seed of 1234. To suppress all of the text describing model convergence, add the command: trace = FALSE after verbose = FALSE in your train function.**

start\_time = Sys.time()  
fitControl = trainControl(method = "cv",   
 number = 10)  
  
nnetGrid = expand.grid(size = 1:14,  
 decay = c(0.5, 0.1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7))  
set.seed(1234)  
nnetFit = train(x=as.data.frame(train[,-1]),y=train$Class,   
 method = "nnet",  
 trControl = fitControl,  
 tuneGrid = nnetGrid,  
 verbose = FALSE,  
 trace = FALSE)  
end\_time = Sys.time()  
end\_time-start\_time

## Time difference of 3.401369 mins

**Task 3: Use your model from Task 2 to develop predictions on the training set. Use caret’s confusionMatrix function to evaluate the model quality. Comment on the model quality.**

predNet = predict(nnetFit, train)  
confusionMatrix(predNet, train$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 609 233  
## No 24 88  
##   
## Accuracy : 0.7306   
## 95% CI : (0.7012, 0.7585)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 4.741e-06   
##   
## Kappa : 0.2814   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9621   
## Specificity : 0.2741   
## Pos Pred Value : 0.7233   
## Neg Pred Value : 0.7857   
## Prevalence : 0.6635   
## Detection Rate : 0.6384   
## Detection Prevalence : 0.8826   
## Balanced Accuracy : 0.6181   
##   
## 'Positive' Class : Yes   
##

The 73.06% accuracy of this model is higher than the naive model’s accuracy of 66.35%. This indicates a solid improvement on the neural network’s predictions.

**Task 4: Use your model from Task 3 to develop predictions on the testing set. Use the confusionMatrix command to assess and comment on the quality of the model.**

predNetTest = predict(nnetFit, test)  
confusionMatrix(predNetTest, test$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 250 109  
## No 21 28  
##   
## Accuracy : 0.6814   
## 95% CI : (0.6337, 0.7264)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.2488   
##   
## Kappa : 0.1508   
##   
## Mcnemar's Test P-Value : 2.34e-14   
##   
## Sensitivity : 0.9225   
## Specificity : 0.2044   
## Pos Pred Value : 0.6964   
## Neg Pred Value : 0.5714   
## Prevalence : 0.6642   
## Detection Rate : 0.6127   
## Detection Prevalence : 0.8799   
## Balanced Accuracy : 0.5634   
##   
## 'Positive' Class : Yes   
##

When applied to the testing set, the accuracy of the model still outperforms the naive model. However, there is some degradation, with an accuracy of 68.14%.

**Task 5: Let’s build an ensemble model. The ensemble model will contain the following models: a logistic regression model (use all variables as predictors), a classification tree, a random forest (with the range package), and a new neural network. Use 5 folds to save time (rather than 10). Your random number seed should be set to 111. Maximizing AUC will be our objective.**

start\_time = Sys.time()  
  
control = trainControl(  
 method = "cv",  
 number = 5,  
 savePredictions = "final",  
 classProbs = TRUE,  
 summaryFunction = twoClassSummary,  
 index=createResample(train$Class),  
 verbose=FALSE  
 )  
  
set.seed(111)  
model\_list = caretList(  
 x=as.data.frame(train[-1]), y=train$Class,  
 metric = "ROC",  
 trControl= control,  
 methodList=c("glm","rpart"),  
tuneList=list(  
ranger = caretModelSpec(method="ranger", max.depth = 5, tuneGrid =  
expand.grid(mtry = 1:13,  
splitrule = c("gini","extratrees","hellinger"),  
min.node.size=1:5)),  
nn = caretModelSpec(method="nnet", tuneGrid =  
expand.grid(size = 1:23,  
decay = c(0.5, 0.1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7)),trace=FALSE)))  
  
end\_time = Sys.time()  
end\_time - start\_time

## Time difference of 13.34775 mins

**Task 6: Discuss the correlation that exists (or does not exist) between the models in the ensemble.**

modelCor(resamples(model\_list))

## ranger nn glm rpart  
## ranger 1.0000000 0.5053779 0.3568218 0.3173736  
## nn 0.5053779 1.0000000 0.9415648 0.2286096  
## glm 0.3568218 0.9415648 1.0000000 0.1818420  
## rpart 0.3173736 0.2286096 0.1818420 1.0000000

GLM and NN have the highest correlation in the group (.93). The next closest correlation exists between Ranger and NN (.43). The other models do not have as much correlation with each other, less than .25 in all cases.

**Task 7: Build the ensemble model (using the caretEnsemble function) and comment on the results. How does the model perform on the training and testing sets?**

ensemble = caretEnsemble(  
 model\_list,   
 metric="ROC",  
 trControl=control)

summary(ensemble)

## The following models were ensembled: ranger, nn, glm, rpart   
## They were weighted:   
## 5.1908 -7.9829 -0.4523 -0.5654 0.0795  
## The resulting ROC is: 0.6996  
## The fit for each individual model on the ROC is:   
## method ROC ROCSD  
## ranger 0.7308913 0.01880511  
## nn 0.6990850 0.02232856  
## glm 0.6939204 0.01885028  
## rpart 0.6376733 0.02786347

#training set  
pred\_ensemble = predict(ensemble, train, type = "raw")  
confusionMatrix(pred\_ensemble,train$Class)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 582 200  
## No 51 121  
##   
## Accuracy : 0.7369   
## 95% CI : (0.7077, 0.7646)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 5.95e-07   
##   
## Kappa : 0.3347   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9194   
## Specificity : 0.3769   
## Pos Pred Value : 0.7442   
## Neg Pred Value : 0.7035   
## Prevalence : 0.6635   
## Detection Rate : 0.6101   
## Detection Prevalence : 0.8197   
## Balanced Accuracy : 0.6482   
##   
## 'Positive' Class : Yes   
##

#testing set  
pred\_ensemble\_test = predict(ensemble, test, type = "raw")  
confusionMatrix(pred\_ensemble\_test,test$Class)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 242 97  
## No 29 40  
##   
## Accuracy : 0.6912   
## 95% CI : (0.6439, 0.7357)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.1352   
##   
## Kappa : 0.2108   
##   
## Mcnemar's Test P-Value : 2.39e-09   
##   
## Sensitivity : 0.8930   
## Specificity : 0.2920   
## Pos Pred Value : 0.7139   
## Neg Pred Value : 0.5797   
## Prevalence : 0.6642   
## Detection Rate : 0.5931   
## Detection Prevalence : 0.8309   
## Balanced Accuracy : 0.5925   
##   
## 'Positive' Class : Yes   
##

The ensemble model outperforms the naive model on the training set, with an accuracy of 74.21%, over 66.35%. When run on the testing set, the resulting accuracy comes to 68.63%.

**Task 8: Build a stacked model and use the stacked model to make predictions on the training and testing set. Remember to use a new trainControl object. Use 10 fold k-fold cross-validation in this object.**

start\_time = Sys.time()  
  
stack = caretStack(  
 model\_list,   
 method ="glm",   
 metric ="ROC",   
   
 trControl=trainControl(  
 method="cv",  
 number=10,  
 savePredictions="final",  
 classProbs=TRUE,  
 summaryFunction=twoClassSummary  
 )  
)  
end\_time = Sys.time()  
end\_time - start\_time

## Time difference of 1.288742 secs

print(stack)

## A glm ensemble of 4 base models: ranger, nn, glm, rpart  
##   
## Ensemble results:  
## Generalized Linear Model   
##   
## 3516 samples  
## 4 predictor  
## 2 classes: 'Yes', 'No'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 3164, 3164, 3166, 3164, 3165, 3165, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.7206063 0.899112 0.3049494

summary(stack)

##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.3865 -0.8588 -0.6236 1.1192 2.0575   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 5.19078 0.37012 14.025 <2e-16 \*\*\*  
## ranger -7.98295 0.81800 -9.759 <2e-16 \*\*\*  
## nn -0.45235 0.76340 -0.593 0.553   
## glm -0.56545 0.65129 -0.868 0.385   
## rpart 0.07952 0.25495 0.312 0.755   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4496.5 on 3515 degrees of freedom  
## Residual deviance: 4037.3 on 3511 degrees of freedom  
## AIC: 4047.3  
##   
## Number of Fisher Scoring iterations: 4

**Now use the stacked model to make predictions on the training and testing set.**

pred\_stack = predict(stack, train, type = "raw")  
confusionMatrix(pred\_stack,train$Class)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 582 200  
## No 51 121  
##   
## Accuracy : 0.7369   
## 95% CI : (0.7077, 0.7646)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 5.95e-07   
##   
## Kappa : 0.3347   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9194   
## Specificity : 0.3769   
## Pos Pred Value : 0.7442   
## Neg Pred Value : 0.7035   
## Prevalence : 0.6635   
## Detection Rate : 0.6101   
## Detection Prevalence : 0.8197   
## Balanced Accuracy : 0.6482   
##   
## 'Positive' Class : Yes   
##

pred\_stack\_test = predict(stack, test, type = "raw")  
confusionMatrix(pred\_stack\_test,test$Class)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 242 97  
## No 29 40  
##   
## Accuracy : 0.6912   
## 95% CI : (0.6439, 0.7357)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.1352   
##   
## Kappa : 0.2108   
##   
## Mcnemar's Test P-Value : 2.39e-09   
##   
## Sensitivity : 0.8930   
## Specificity : 0.2920   
## Pos Pred Value : 0.7139   
## Neg Pred Value : 0.5797   
## Prevalence : 0.6642   
## Detection Rate : 0.5931   
## Detection Prevalence : 0.8309   
## Balanced Accuracy : 0.5925   
##   
## 'Positive' Class : Yes   
##

**Task 9: Build an xgboost model. First you will need to create new training and tests sets in which the categorical variables are one-hot encoded to dummy variables. Use select to exclude the NEGATIVE level of the Class variable from your training and testing sets.**

train\_dummy = dummyVars(" ~ .", data = train)  
train\_xgb = data.frame(predict(train\_dummy, newdata = train))  
str(train\_xgb)

## 'data.frame': 954 obs. of 25 variables:  
## $ Class.Yes : num 1 1 1 1 1 1 1 1 0 0 ...  
## $ Class.No : num 0 0 0 0 0 0 0 0 1 1 ...  
## $ X.Revenue.Growth. : num 0.1115 0.1289 0.3735 0.0636 0.0421 ...  
## $ X.EPS.Diluted. : num 2.53 4.48 7.57 2.85 0.85 3.67 1.56 4.88 -0.02 1.89 ...  
## $ X.EBITDA.Margin. : num 0.31 0.456 0.531 0.355 0.438 0.248 0.323 0.172 0.039 0.166 ...  
## $ priceBookValueRatio : num 2.16 2.86 4.48 1.13 4.08 ...  
## $ debtEquityRatio : num 1.56 0.353 0 0.959 1.307 ...  
## $ debtRatio : num 0.444 0.206 0 0.332 0.44 ...  
## $ X.PE.ratio. : num 13.3 10.3 17.1 10 53.7 ...  
## $ Sector.Consumer.Cyclical : num 1 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Energy : num 0 0 0 0 0 0 0 1 0 1 ...  
## $ Sector.Technology : num 0 1 1 0 1 0 1 0 1 0 ...  
## $ Sector.Industrials : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Financial.Services : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Basic.Materials : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Communication.Services : num 0 0 0 1 0 0 0 0 0 0 ...  
## $ Sector.Consumer.Defensive : num 0 0 0 0 0 1 0 0 0 0 ...  
## $ Sector.Healthcare : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Real.Estate : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Utilities : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ X.5Y.Revenue.Growth..per.Share..: num 0.1094 0.077 0.4281 -0.0081 0.0416 ...  
## $ returnOnAssets : num 0.303 0.344 0.325 0.143 0.057 ...  
## $ returnOnEquity : num 0.1638 0.2824 0.2628 0.1052 0.0774 ...  
## $ returnOnCapitalEmployed : num 0.0531 0.1444 0.3165 0.0352 0.1495 ...  
## $ quickRatio : num 0.54 1.105 6.94 0.492 3.786 ...

test\_dummy = dummyVars(" ~ .", data = test)  
test\_xgb = data.frame(predict(test\_dummy, newdata = test))

train\_xgb = train\_xgb %>% dplyr::select(-Class.No)  
test\_xgb = test\_xgb %>% dplyr::select(-Class.No)

str(train\_xgb)

## 'data.frame': 954 obs. of 24 variables:  
## $ Class.Yes : num 1 1 1 1 1 1 1 1 0 0 ...  
## $ X.Revenue.Growth. : num 0.1115 0.1289 0.3735 0.0636 0.0421 ...  
## $ X.EPS.Diluted. : num 2.53 4.48 7.57 2.85 0.85 3.67 1.56 4.88 -0.02 1.89 ...  
## $ X.EBITDA.Margin. : num 0.31 0.456 0.531 0.355 0.438 0.248 0.323 0.172 0.039 0.166 ...  
## $ priceBookValueRatio : num 2.16 2.86 4.48 1.13 4.08 ...  
## $ debtEquityRatio : num 1.56 0.353 0 0.959 1.307 ...  
## $ debtRatio : num 0.444 0.206 0 0.332 0.44 ...  
## $ X.PE.ratio. : num 13.3 10.3 17.1 10 53.7 ...  
## $ Sector.Consumer.Cyclical : num 1 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Energy : num 0 0 0 0 0 0 0 1 0 1 ...  
## $ Sector.Technology : num 0 1 1 0 1 0 1 0 1 0 ...  
## $ Sector.Industrials : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Financial.Services : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Basic.Materials : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Communication.Services : num 0 0 0 1 0 0 0 0 0 0 ...  
## $ Sector.Consumer.Defensive : num 0 0 0 0 0 1 0 0 0 0 ...  
## $ Sector.Healthcare : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Real.Estate : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Utilities : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ X.5Y.Revenue.Growth..per.Share..: num 0.1094 0.077 0.4281 -0.0081 0.0416 ...  
## $ returnOnAssets : num 0.303 0.344 0.325 0.143 0.057 ...  
## $ returnOnEquity : num 0.1638 0.2824 0.2628 0.1052 0.0774 ...  
## $ returnOnCapitalEmployed : num 0.0531 0.1444 0.3165 0.0352 0.1495 ...  
## $ quickRatio : num 0.54 1.105 6.94 0.492 3.786 ...

str(test\_xgb)

## 'data.frame': 408 obs. of 24 variables:  
## $ Class.Yes : num 1 0 1 0 1 1 1 1 1 0 ...  
## $ X.Revenue.Growth. : num 0.032 0.021 0.187 0.381 -0.152 ...  
## $ X.EPS.Diluted. : num 0.66 1.87 3.23 1.29 4.17 5.32 7.74 1.33 2.55 -4.27 ...  
## $ X.EBITDA.Margin. : num 0.453 0.34 0.312 0.622 0.465 0.429 0.245 0.229 0.318 -0.109 ...  
## $ priceBookValueRatio : num 1.027 3.987 4.428 0.979 3.761 ...  
## $ debtEquityRatio : num 1.108 0.658 0.776 0.453 1.278 ...  
## $ debtRatio : num 0.473 0.262 0.301 0.258 0.429 ...  
## $ X.PE.ratio. : num 23.3 22.97 9.52 11.03 14.89 ...  
## $ Sector.Consumer.Cyclical : num 0 0 0 0 0 0 0 0 1 0 ...  
## $ Sector.Energy : num 1 0 0 1 0 1 1 0 0 1 ...  
## $ Sector.Technology : num 0 0 1 0 0 0 0 0 0 0 ...  
## $ Sector.Industrials : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Financial.Services : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Basic.Materials : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Communication.Services : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Consumer.Defensive : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Healthcare : num 0 1 0 0 1 0 0 1 0 0 ...  
## $ Sector.Real.Estate : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Utilities : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ X.5Y.Revenue.Growth..per.Share..: num -0.1402 0.0383 0.2221 -0.1418 0.184 ...  
## $ returnOnAssets : num 0.0446 0.3546 0.4096 0.0613 0.1866 ...  
## $ returnOnEquity : num 0.0478 0.1759 0.484 0.0904 0.2551 ...  
## $ returnOnCapitalEmployed : num 0.0339 0.0634 0.3096 0.0578 0.2003 ...  
## $ quickRatio : num 0.632 0.843 1.62 1.387 3.151 ...

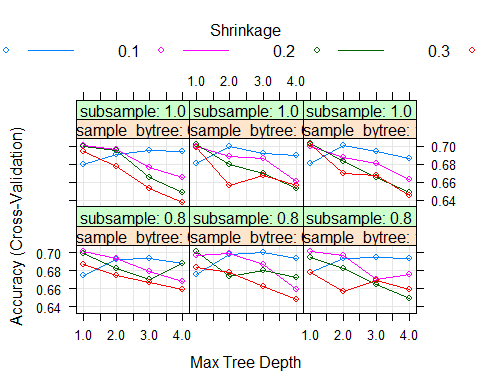
start\_time = Sys.time()  
  
set.seed(999)  
ctrl = trainControl(method = "cv",  
 number = 5)  
tgrid = expand.grid(  
 nrounds = 100,  
 max\_depth = c(1,2,3,4),  
 eta = c(0.01, 0.1, 0.2, 0.3),  
 gamma = 0,  
 colsample\_bytree = c(0.6, 0.8, 1),  
 min\_child\_weight = 1,  
 subsample = c(0.8, 1)  
)  
  
fitxgb2 = train(as.factor(Class.Yes)~.,  
 data = train\_xgb,  
 method="xgbTree",  
 tuneGrid = tgrid,  
 trControl=ctrl)  
  
end\_time = Sys.time()  
end\_time-start\_time

## Time difference of 33.52669 secs

fitxgb2

## eXtreme Gradient Boosting   
##   
## 954 samples  
## 23 predictor  
## 2 classes: '0', '1'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 763, 763, 763, 763, 764   
## Resampling results across tuning parameters:  
##   
## eta max\_depth colsample\_bytree subsample Accuracy Kappa   
## 0.01 1 0.6 0.8 0.6750455 0.05626173  
## 0.01 1 0.6 1.0 0.6792339 0.07404054  
## 0.01 1 0.8 0.8 0.6760926 0.06408438  
## 0.01 1 0.8 1.0 0.6802811 0.07800084  
## 0.01 1 1.0 0.8 0.6781868 0.07004025  
## 0.01 1 1.0 1.0 0.6802811 0.07800084  
## 0.01 2 0.6 0.8 0.6918159 0.16461039  
## 0.01 2 0.6 1.0 0.6907688 0.16700025  
## 0.01 2 0.8 0.8 0.6981042 0.18623267  
## 0.01 2 0.8 1.0 0.6991458 0.18666760  
## 0.01 2 1.0 0.8 0.6928686 0.17563739  
## 0.01 2 1.0 1.0 0.7001929 0.19040504  
## 0.01 3 0.6 0.8 0.6928630 0.18804049  
## 0.01 3 0.6 1.0 0.6949628 0.19841511  
## 0.01 3 0.8 0.8 0.7002039 0.21384439  
## 0.01 3 0.8 1.0 0.6918214 0.19049832  
## 0.01 3 1.0 0.8 0.6949628 0.19679385  
## 0.01 3 1.0 1.0 0.6939157 0.19591868  
## 0.01 4 0.6 0.8 0.6876385 0.17992482  
## 0.01 4 0.6 1.0 0.6939212 0.20071448  
## 0.01 4 0.8 0.8 0.6928686 0.20828420  
## 0.01 4 0.8 1.0 0.6897217 0.19710733  
## 0.01 4 1.0 0.8 0.6939212 0.20594987  
## 0.01 4 1.0 1.0 0.6865803 0.19182982  
## 0.10 1 0.6 0.8 0.7012455 0.23459682  
## 0.10 1 0.6 1.0 0.7001874 0.21498579  
## 0.10 1 0.8 0.8 0.6970681 0.22674969  
## 0.10 1 0.8 1.0 0.6991403 0.20867174  
## 0.10 1 1.0 0.8 0.7012731 0.24233086  
## 0.10 1 1.0 1.0 0.6991403 0.20897428  
## 0.10 2 0.6 0.8 0.6938991 0.24069803  
## 0.10 2 0.6 1.0 0.6960099 0.23115421  
## 0.10 2 0.8 0.8 0.6991458 0.25682344  
## 0.10 2 0.8 1.0 0.6886746 0.20831444  
## 0.10 2 1.0 0.8 0.6970626 0.24699922  
## 0.10 2 1.0 1.0 0.6876274 0.20215913  
## 0.10 3 0.6 0.8 0.6792284 0.21358513  
## 0.10 3 0.6 1.0 0.6760871 0.18915400  
## 0.10 3 0.8 0.8 0.6865858 0.22711778  
## 0.10 3 0.8 1.0 0.6865803 0.21660653  
## 0.10 3 1.0 0.8 0.6697878 0.19585647  
## 0.10 3 1.0 1.0 0.6802921 0.19859139  
## 0.10 4 0.6 0.8 0.6677156 0.19562523  
## 0.10 4 0.6 1.0 0.6656048 0.17076936  
## 0.10 4 0.8 0.8 0.6593387 0.17279749  
## 0.10 4 0.8 1.0 0.6603527 0.17006005  
## 0.10 4 1.0 0.8 0.6760761 0.22233133  
## 0.10 4 1.0 1.0 0.6635161 0.17653108  
## 0.20 1 0.6 0.8 0.6991623 0.25339418  
## 0.20 1 0.6 1.0 0.6991513 0.23783973  
## 0.20 1 0.8 0.8 0.7012731 0.24797908  
## 0.20 1 0.8 1.0 0.7012565 0.24481923  
## 0.20 1 1.0 0.8 0.6949738 0.23496056  
## 0.20 1 1.0 1.0 0.7033618 0.24900872  
## 0.20 2 0.6 0.8 0.6824084 0.23144740  
## 0.20 2 0.6 1.0 0.6949738 0.24508966  
## 0.20 2 0.8 0.8 0.6729402 0.20922014  
## 0.20 2 0.8 1.0 0.6792395 0.21301484  
## 0.20 2 1.0 0.8 0.6823808 0.23203501  
## 0.20 2 1.0 1.0 0.6834279 0.22047726  
## 0.20 3 0.6 0.8 0.6698154 0.20655369  
## 0.20 3 0.6 1.0 0.6656048 0.19294880  
## 0.20 3 0.8 0.8 0.6802921 0.23974620  
## 0.20 3 0.8 1.0 0.6698209 0.20344318  
## 0.20 3 1.0 0.8 0.6645688 0.20714935  
## 0.20 3 1.0 1.0 0.6656214 0.19192907  
## 0.20 4 0.6 0.8 0.6876109 0.25256297  
## 0.20 4 0.6 1.0 0.6488289 0.16269745  
## 0.20 4 0.8 0.8 0.6719151 0.21474315  
## 0.20 4 0.8 1.0 0.6530284 0.16907165  
## 0.20 4 1.0 0.8 0.6488564 0.15831443  
## 0.20 4 1.0 1.0 0.6488289 0.15854420  
## 0.30 1 0.6 0.8 0.6865858 0.23100398  
## 0.30 1 0.6 1.0 0.6939212 0.23239307  
## 0.30 1 0.8 0.8 0.6834390 0.22321427  
## 0.30 1 0.8 1.0 0.6981207 0.24523522  
## 0.30 1 1.0 0.8 0.6781923 0.21038293  
## 0.30 1 1.0 1.0 0.7012565 0.25437909  
## 0.30 2 0.6 0.8 0.6750510 0.22302469  
## 0.30 2 0.6 1.0 0.6771177 0.22431729  
## 0.30 2 0.8 0.8 0.6781703 0.24825118  
## 0.30 2 0.8 1.0 0.6561808 0.17311706  
## 0.30 2 1.0 0.8 0.6572389 0.18848007  
## 0.30 2 1.0 1.0 0.6698044 0.20368415  
## 0.30 3 0.6 0.8 0.6666465 0.21439165  
## 0.30 3 0.6 1.0 0.6530504 0.17403788  
## 0.30 3 0.8 0.8 0.6624635 0.21409648  
## 0.30 3 0.8 1.0 0.6676826 0.21875174  
## 0.30 3 1.0 0.8 0.6687572 0.21897855  
## 0.30 3 1.0 1.0 0.6676770 0.21173740  
## 0.30 4 0.6 0.8 0.6592891 0.19902905  
## 0.30 4 0.6 1.0 0.6373216 0.13188940  
## 0.30 4 0.8 0.8 0.6478148 0.17271477  
## 0.30 4 0.8 1.0 0.6561367 0.18961533  
## 0.30 4 1.0 0.8 0.6593331 0.19396695  
## 0.30 4 1.0 1.0 0.6456875 0.16656118  
##   
## Tuning parameter 'nrounds' was held constant at a value of 100  
## Tuning  
## parameter 'gamma' was held constant at a value of 0  
## Tuning  
## parameter 'min\_child\_weight' was held constant at a value of 1  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were nrounds = 100, max\_depth = 1, eta  
## = 0.2, gamma = 0, colsample\_bytree = 1, min\_child\_weight = 1 and subsample = 1.

plot(fitxgb2)

   
**Develop predictions on the training and testing sets.**

predxgbtrain2 = predict(fitxgb2, train\_xgb)  
confusionMatrix(as.factor(train\_xgb$Class.Yes), predxgbtrain2,positive="1")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 122 199  
## 1 45 588  
##   
## Accuracy : 0.7442   
## 95% CI : (0.7153, 0.7717)  
## No Information Rate : 0.8249   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3504   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.7471   
## Specificity : 0.7305   
## Pos Pred Value : 0.9289   
## Neg Pred Value : 0.3801   
## Prevalence : 0.8249   
## Detection Rate : 0.6164   
## Detection Prevalence : 0.6635   
## Balanced Accuracy : 0.7388   
##   
## 'Positive' Class : 1   
##

predxgbtest2 = predict(fitxgb2, test\_xgb)  
confusionMatrix(as.factor(test\_xgb$Class.Yes), predxgbtest2,positive="1")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 40 97  
## 1 33 238  
##   
## Accuracy : 0.6814   
## 95% CI : (0.6337, 0.7264)  
## No Information Rate : 0.8211   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.1924   
##   
## Mcnemar's Test P-Value : 3.286e-08   
##   
## Sensitivity : 0.7104   
## Specificity : 0.5479   
## Pos Pred Value : 0.8782   
## Neg Pred Value : 0.2920   
## Prevalence : 0.8211   
## Detection Rate : 0.5833   
## Detection Prevalence : 0.6642   
## Balanced Accuracy : 0.6292   
##   
## 'Positive' Class : 1   
##