## Week 5 Assigment 1

### DEY, Sankha

#### Binary Classification Stock Market Data

# message=FALSE done before final knit to reduce overall dcoument size.  
library(tidyverse)  
library(caret)  
library(rpart)  
library(nnet)  
library(caretEnsemble)  
library(ranger)  
library(xgboost)

fin <- read\_csv("2018Fin.csv")

## 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 %>% dplyr::select(Class, `Revenue Growth`,`EPS Diluted`,`EBITDA Margin`, priceBookValueRatio,debtEquityRatio,debtRatio,`PE ratio`,Sector,`5Y Revenue Growth (per Share)`,returnOnAssets,returnOnEquity,returnOnCapitalEmployed,quickRatio)  
#str(fin) #Commented before final knit to reduce overall dcoument size.

fin = fin %>% mutate(Class = as\_factor(Class)) %>%   
 mutate(Class= fct\_recode(Class, "No" = "0", "Yes" = "1" )) %>%  
 mutate(Sector = as\_factor(Sector))

Mising Data

fin = fin %>% drop\_na()  
#str(fin)

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)  
str(fin)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 1362 obs. of 14 variables:  
## $ Class : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 1 2 2 ...  
## $ Revenue Growth : num 0.1115 0.032 0.1289 0.3735 0.0636 ...  
## $ EPS Diluted : num 2.53 0.66 4.48 7.57 2.85 0.85 3.67 1.87 1.56 3.23 ...  
## $ EBITDA Margin : num 0.31 0.453 0.456 0.531 0.355 0.438 0.248 0.34 0.323 0.312 ...  
## $ priceBookValueRatio : num 2.16 1.03 2.86 4.48 1.13 ...  
## $ debtEquityRatio : num 1.56 1.108 0.353 0 0.959 ...  
## $ debtRatio : num 0.444 0.473 0.206 0 0.332 ...  
## $ PE ratio : num 13.3 23.3 10.3 17.1 10 ...  
## $ Sector : Factor w/ 11 levels "Consumer Cyclical",..: 1 2 3 3 7 3 8 9 3 3 ...  
## $ 5Y Revenue Growth (per Share): num 0.1094 -0.1402 0.077 0.4281 -0.0081 ...  
## $ returnOnAssets : num 0.3033 0.0446 0.3444 0.3245 0.1429 ...  
## $ returnOnEquity : num 0.1638 0.0478 0.2824 0.2628 0.1052 ...  
## $ returnOnCapitalEmployed : num 0.0531 0.0339 0.1444 0.3165 0.0352 ...  
## $ quickRatio : num 0.54 0.632 1.105 6.94 0.492 ...

#### Task 1

Training/testing split

set.seed(12345)   
train.rows = createDataPartition(y = fin$Class, p=0.7, list = FALSE) #70% in training  
train = dplyr::slice(fin,train.rows) #conflict with slice function  
test = dplyr::slice(fin,-train.rows)

#### Task 2

Neural Network

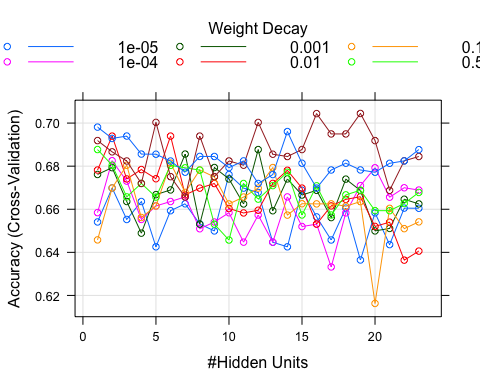
# start\_time = Sys.time() #for timing  
# fitControl = trainControl(method = "cv",   
# number = 10)  
#   
# nnetGrid = expand.grid(size = seq(from = 1, to = 23, by = 1), #rule of thumb --> between # of input and # of output layers (input can be considered number of predictors, output is usually 1 for classification)  
# decay = c(0.5, 0.1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7))  
# set.seed(1234)  
# nnetFit2 = 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

# saveRDS(nnetFit2,"nnetfit2.rds")  
# rm(nnetFit2)

nnetFit2 = readRDS("nnetfit2.rds")

#### Task 3

#nnetFit2 ##Commented before final knit to reduce overall dcoument size.  
plot(nnetFit2)



Accuracy has been maximized when size = 19 and decay = 0.1. Accuracy was = 0.7044627

Predictions

predNet2 = predict(nnetFit2, train)

Confusion matrix

confusionMatrix(predNet2, train$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 222 50  
## Yes 99 583  
##   
## Accuracy : 0.8438   
## 95% CI : (0.8192, 0.8663)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6365   
##   
## Mcnemar's Test P-Value : 8.413e-05   
##   
## Sensitivity : 0.9210   
## Specificity : 0.6916   
## Pos Pred Value : 0.8548   
## Neg Pred Value : 0.8162   
## Prevalence : 0.6635   
## Detection Rate : 0.6111   
## Detection Prevalence : 0.7149   
## Balanced Accuracy : 0.8063   
##   
## 'Positive' Class : Yes   
##

Model quality is looking good. Acuracy is 0.8438 which is better than Naive model (0.6635).

#### Task 4

Test Dataset

predNet2 = predict(nnetFit2, test)

Confusion matrix

confusionMatrix(predNet2, test$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 48 54  
## Yes 89 217  
##   
## Accuracy : 0.6495   
## 95% CI : (0.601, 0.6958)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.753140   
##   
## Kappa : 0.1613   
##   
## Mcnemar's Test P-Value : 0.004466   
##   
## Sensitivity : 0.8007   
## Specificity : 0.3504   
## Pos Pred Value : 0.7092   
## Neg Pred Value : 0.4706   
## Prevalence : 0.6642   
## Detection Rate : 0.5319   
## Detection Prevalence : 0.7500   
## Balanced Accuracy : 0.5756   
##   
## 'Positive' Class : Yes   
##

This model is not looking good on test data with accuracy dropped significantly from training. Accuracy (0.6495) in test set is even lower than the Naive model (0.6642). This model shouldn’t be recommended in real-life prediction scenario.

#### Task 5

Ensemble Model

control = trainControl(  
 method = "cv",  
 number = 5,  
 savePredictions = "final",  
 classProbs = TRUE,   
 summaryFunction = twoClassSummary,   
 index=createResample(train$Class)   
 )

# start\_time = Sys.time()  
# set.seed(111)  
# model\_list = caretList(  
# x=as.data.frame(train[,-1]), y=train$Class, #use all variables (except Class) as predictors  
# metric = "ROC", #specify that maximizing AUC is our objective  
# trControl= control, #using the previously defined trControl object  
# methodList=c("glm","rpart"), #specifying the model methods to use  
# 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

# saveRDS(model\_list,"model\_list.rds")  
# rm(model\_list)

model\_list = readRDS("model\_list.rds")

#### Task 6

as.data.frame(predict(model\_list, newdata=head(train)))

## ranger nn glm rpart  
## 1 0.2089846 0.20422299 0.22056786 0.2286213  
## 2 0.1474833 0.08162977 0.10481112 0.2286213  
## 3 0.2709432 0.05845163 0.08503933 0.2286213  
## 4 0.2898835 0.61142715 0.57672520 0.2286213  
## 5 0.2137929 0.08813134 0.10852095 0.2286213  
## 6 0.1577264 0.19350372 0.21416066 0.2286213

Ideally, our models should each exhibit “good” performance, but be uncorrelated with each other. We can check model correlation with the following code.

modelCor(resamples(model\_list)) #show model correlation

## ranger nn glm rpart  
## ranger 1.0000000 0.7127101 0.6548628 0.5486364  
## nn 0.7127101 1.0000000 0.9223977 0.6182542  
## glm 0.6548628 0.9223977 1.0000000 0.6167166  
## rpart 0.5486364 0.6182542 0.6167166 1.0000000

The neural network and glm models are strongly correlated (0.922). Neural and ranger are also somewhat correlated (0.713). Weaker correlation are seen between other models.

#### Task 7

ensemble = caretEnsemble(  
 model\_list,   
 metric="ROC",  
 trControl=control #we already defined the trControl object  
 )

Examine the ensemble.

summary(ensemble)

## The following models were ensembled: ranger, nn, glm, rpart   
## They were weighted:   
## 2.6597 -4.3496 -1.3718 -0.1362 0.3739  
## The resulting ROC is: 0.6975  
## The fit for each individual model on the ROC is:   
## method ROC ROCSD  
## ranger 0.7217493 0.01552264  
## nn 0.7079702 0.01861469  
## glm 0.7056371 0.02018710  
## rpart 0.6270063 0.02283864

From the summary, we see that the resulting AUC (shown as ROC) for the ensemble is ~0.7. Ranger, neural network and glm have better ROC individually. ROC of rpart is relatively low.

We can then evaluate the performance of the ensemble on the training and testing sets.

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 119 33  
## Yes 202 600  
##   
## Accuracy : 0.7537   
## 95% CI : (0.725, 0.7807)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 9.251e-10   
##   
## Kappa : 0.3661   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.3707   
## Specificity : 0.9479   
## Pos Pred Value : 0.7829   
## Neg Pred Value : 0.7481   
## Prevalence : 0.3365   
## Detection Rate : 0.1247   
## Detection Prevalence : 0.1593   
## Balanced Accuracy : 0.6593   
##   
## 'Positive' Class : No   
##

Model is doing well on training set with accuracy 0.7537 which is higher than the Naive model (0.6635). However we had 84% accuracy in neural network model earlier.

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 33 24  
## Yes 104 247  
##   
## Accuracy : 0.6863   
## 95% CI : (0.6388, 0.731)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.1868   
##   
## Kappa : 0.178   
##   
## Mcnemar's Test P-Value : 2.896e-12   
##   
## Sensitivity : 0.24088   
## Specificity : 0.91144   
## Pos Pred Value : 0.57895   
## Neg Pred Value : 0.70370   
## Prevalence : 0.33578   
## Detection Rate : 0.08088   
## Detection Prevalence : 0.13971   
## Balanced Accuracy : 0.57616   
##   
## 'Positive' Class : No   
##

On test set, the accuracy dropped to 0.6863. It’s somewhat better than the Naive.

#### Task 8 (Stacking)

start\_time = Sys.time()  
stack = caretStack(  
 model\_list, #use the list of models already specified  
 method ="glm", #stack models linearly  
 metric ="ROC", #maximize AUC  
 trControl = trainControl(  
 method="cv",  
 number=10,  
 savePredictions="final",  
 classProbs=TRUE,  
 summaryFunction=twoClassSummary  
 )  
 )  
end\_time = Sys.time()  
end\_time - start\_time

## Time difference of 2.0048 secs

print(stack)

## A glm ensemble of 4 base models: ranger, nn, glm, rpart  
##   
## Ensemble results:  
## Generalized Linear Model   
##   
## 3498 samples  
## 4 predictor  
## 2 classes: 'No', 'Yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 3148, 3149, 3148, 3149, 3147, 3148, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.719946 0.3105022 0.8834297

ROC is improved to 0.7186. This is slightly better than the RUC we’ve seen from ensemble.

summary(stack)

##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1098 -1.1084 0.5894 0.8694 1.9346   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 2.6597 0.1224 21.723 < 2e-16 \*\*\*  
## ranger -4.3496 0.5540 -7.851 4.12e-15 \*\*\*  
## nn -1.3718 0.6370 -2.153 0.0313 \*   
## glm -0.1362 0.6181 -0.220 0.8256   
## rpart 0.3739 0.2450 1.526 0.1270   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4439.1 on 3497 degrees of freedom  
## Residual deviance: 3979.0 on 3493 degrees of freedom  
## AIC: 3989  
##   
## Number of Fisher Scoring iterations: 3

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 119 33  
## Yes 202 600  
##   
## Accuracy : 0.7537   
## 95% CI : (0.725, 0.7807)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 9.251e-10   
##   
## Kappa : 0.3661   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.3707   
## Specificity : 0.9479   
## Pos Pred Value : 0.7829   
## Neg Pred Value : 0.7481   
## Prevalence : 0.3365   
## Detection Rate : 0.1247   
## Detection Prevalence : 0.1593   
## Balanced Accuracy : 0.6593   
##   
## 'Positive' Class : No   
##

Using stancked model accuracy is 0.7537 on train set. same accurace that was derived from enesemble model. No such improvements.

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 33 24  
## Yes 104 247  
##   
## Accuracy : 0.6863   
## 95% CI : (0.6388, 0.731)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.1868   
##   
## Kappa : 0.178   
##   
## Mcnemar's Test P-Value : 2.896e-12   
##   
## Sensitivity : 0.24088   
## Specificity : 0.91144   
## Pos Pred Value : 0.57895   
## Neg Pred Value : 0.70370   
## Prevalence : 0.33578   
## Detection Rate : 0.08088   
## Detection Prevalence : 0.13971   
## Balanced Accuracy : 0.57616   
##   
## 'Positive' Class : No   
##

Using stancked model accuracy is 0.6863 on test set. same accurace that was derived from enesemble model. No difference.

#### Task 9 (XGBOOST Model)

Training and tests sets in which the categorical variables are one-hot encoded to dummy variables.

train\_dummy = dummyVars(" ~ .", data = train) #creates dummy labels  
train\_xgb = data.frame(predict(train\_dummy, newdata = train)) #converts variables in dataset to dummies  
test\_dummy = dummyVars(" ~ .", data = test) #creates dummy labels  
test\_xgb = data.frame(predict(test\_dummy, newdata = test)) #converts variables in dataset to dummies  
str(train\_xgb)

## 'data.frame': 954 obs. of 25 variables:  
## $ Class.No : num 0 0 0 0 0 0 0 0 0 1 ...  
## $ Class.Yes : num 1 1 1 1 1 1 1 1 1 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 3.23 4.88 -0.02 ...  
## $ X.EBITDA.Margin. : num 0.31 0.456 0.531 0.355 0.438 0.248 0.323 0.312 0.172 0.039 ...  
## $ 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 0 1 0 ...  
## $ Sector.Technology : num 0 1 1 0 1 0 1 1 0 1 ...  
## $ 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 ...

We only need one variable for Class. Let’s keep the Class.yes, remove Class.no

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 1 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 3.23 4.88 -0.02 ...  
## $ X.EBITDA.Margin. : num 0.31 0.456 0.531 0.355 0.438 0.248 0.323 0.312 0.172 0.039 ...  
## $ 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 0 1 0 ...  
## $ Sector.Technology : num 0 1 1 0 1 0 1 1 0 1 ...  
## $ 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 ...

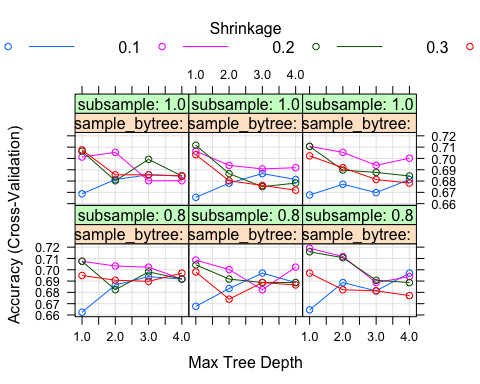
XGBOOST Model

# start\_time = Sys.time() #for timing  
#   
# set.seed(999)  
# ctrl = trainControl(method = "cv",  
# number = 5) #5 fold, k-fold cross-validation  
#   
# tgrid = expand.grid(  
# nrounds = 100, #50, 100, and 150 in default tuning  
# max\_depth = c(1,2,3,4), #1, 2, and 3 in default tuning  
# eta = c(0.01, 0.1, 0.2, 0.3), #0.3 and 0.4 in default tuning  
# gamma = 0, #fixed at 0 in default tuning  
# colsample\_bytree = c(0.6, 0.8, 1), #0.6 and 0.6 in default tuning  
# min\_child\_weight = 1, #fixed at 1 in default tuning  
# subsample = c(0.8, 1) #0.5, 0.75, and 1 in default tuning, we don't have much data so can choose a larger value  
# )  
#   
# fitxgb2 = train(as.factor(Class.Yes)~.,  
# data = train\_xgb,  
# method="xgbTree",  
# tuneGrid = tgrid,  
# trControl=ctrl)  
#   
# end\_time = Sys.time()  
# end\_time-start\_time

# saveRDS(fitxgb2,"fitxgb2.rds")  
# rm(fitxgb2)

fitxgb2 = readRDS("fitxgb2.rds")

#fitxgb2 #Commented before final knit to reduce overall dcoument size.  
plot(fitxgb2)



Best accuracy is 0.7190245 when max\_depth = 1, eta = 0.1, gamma = 0, colsample\_bytree = 1, min\_child\_weight = 1 and subsample = 0.8.

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 116 205  
## 1 45 588  
##   
## Accuracy : 0.7379   
## 95% CI : (0.7088, 0.7656)  
## No Information Rate : 0.8312   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3309   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.7415   
## Specificity : 0.7205   
## Pos Pred Value : 0.9289   
## Neg Pred Value : 0.3614   
## Prevalence : 0.8312   
## Detection Rate : 0.6164   
## Detection Prevalence : 0.6635   
## Balanced Accuracy : 0.7310   
##   
## 'Positive' Class : 1   
##

Accurcy on training set is significantly lower than Naive model. Not a good predcting model.

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 32 105  
## 1 33 238  
##   
## Accuracy : 0.6618   
## 95% CI : (0.6136, 0.7076)  
## No Information Rate : 0.8407   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.1285   
##   
## Mcnemar's Test P-Value : 1.504e-09   
##   
## Sensitivity : 0.6939   
## Specificity : 0.4923   
## Pos Pred Value : 0.8782   
## Neg Pred Value : 0.2336   
## Prevalence : 0.8407   
## Detection Rate : 0.5833   
## Detection Prevalence : 0.6642   
## Balanced Accuracy : 0.5931   
##   
## 'Positive' Class : 1   
##

Accurcy on testing set is significantly lower than Naive model. Also, testing accuracy is lower than training accuracy. Overall this doesn’t seem to be a good model for predicting Class.