Assignment1

Jordan Whitaker

6/20/2020

#install.packages(c("tidyverse", "caret", "nnet", "rpart", "ranger", "caretEnsemble", "xgboost"))  
  
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$Sector = as.factor(fin$Sector)  
  
fin$Class = as.factor(fin$Class)  
  
fin$Class = (case\_when(fin$Class == '0' ~ 'No',  
 fin$Class == '1' ~ 'Yes'))  
  
fin$Class = as.factor(fin$Class)

fin = drop\_na(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)

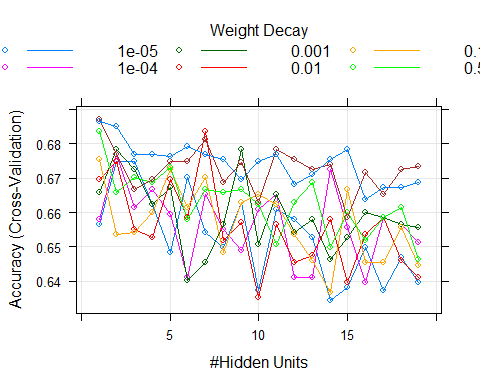
#summary(fin)

set.seed(12345)  
train.rows = createDataPartition(y = fin$Class, p=0.7, list = FALSE)  
train = dplyr::slice(fin, train.rows)  
test = dplyr::slice(fin, -train.rows)

fitControl = trainControl(method = "cv",  
 number = 10)  
nnetGrid = expand.grid(size = 1:19,  
 decay = c(0.5, 0.1, 1e-2, 1e-3, 1e-4, 1e-5,1e-6, 1e-7))  
  
set.seed(1234)  
nnetFit = train(x=fin[,-1], y=fin$Class,  
 method = "nnet",  
 trControl = fitControl,  
 tuneGrid = nnetGrid,  
 verbose = FALSE,  
 trace = FALSE)

saveRDS(nnetFit, "neural\_net.rds")  
rm(nnetFit)  
  
nnetFit = readRDS("neural\_net.rds")

plot(nnetFit)



nnetFit

## Neural Network   
##   
## 1362 samples  
## 13 predictor  
## 2 classes: 'No', 'Yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 1226, 1226, 1226, 1226, 1226, 1226, ...   
## Resampling results across tuning parameters:  
##   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were size = 1 and decay = 0.1.

# Accuracy was used to select the optimal model using the largest value.  
# The final values used for the model were size = 1 and decay = 0.1.  
  
predNet = predict(nnetFit, train)  
  
  
confusionMatrix(predNet, train$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 138 95  
## Yes 183 538  
##   
## Accuracy : 0.7086   
## 95% CI : (0.6786, 0.7373)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 0.001625   
##   
## Kappa : 0.3001   
##   
## Mcnemar's Test P-Value : 1.81e-07   
##   
## Sensitivity : 0.8499   
## Specificity : 0.4299   
## Pos Pred Value : 0.7462   
## Neg Pred Value : 0.5923   
## Prevalence : 0.6635   
## Detection Rate : 0.5639   
## Detection Prevalence : 0.7558   
## Balanced Accuracy : 0.6399   
##   
## 'Positive' Class : Yes   
##

This model has an accuracy of around 71% compared to the naive model (No Information Rate), of 66% this model is performing okay on training data. Now to pump in testing data..

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 47 45  
## Yes 90 226  
##   
## Accuracy : 0.6691   
## 95% CI : (0.6211, 0.7146)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.4397493   
##   
## Kappa : 0.1927   
##   
## Mcnemar's Test P-Value : 0.0001525   
##   
## Sensitivity : 0.8339   
## Specificity : 0.3431   
## Pos Pred Value : 0.7152   
## Neg Pred Value : 0.5109   
## Prevalence : 0.6642   
## Detection Rate : 0.5539   
## Detection Prevalence : 0.7745   
## Balanced Accuracy : 0.5885   
##   
## 'Positive' Class : Yes   
##

The model is not performing well and is actually almost as bad as the naive model. When the model is applied to the testing data, it has an accuracy of 66.91 compared to the naive model at 66.42 this is a poor model.

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

set.seed(111)  
  
model\_list = caretList(  
 x=train[,-1], y = train$Class,  
 metric = "ROC",  
 trControl = control,  
 methodList = c("glm", "ranger", "rpart", "nnet"),  
 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)))

saveRDS(model\_list, "model\_list.rds")  
rm(model\_list)  
model\_list = readRDS("model\_list.rds")

modelCor(resamples(model\_list))

## ranger nn glm ranger.1 rpart nnet  
## ranger 1.0000000 0.6344851 0.72431851 0.8166207 0.28590823 0.7045017  
## nn 0.6344851 1.0000000 0.89036565 0.4834733 -0.20711394 0.9095609  
## glm 0.7243185 0.8903657 1.00000000 0.6921230 0.02909798 0.9907123  
## ranger.1 0.8166207 0.4834733 0.69212303 1.0000000 0.49685372 0.6929567  
## rpart 0.2859082 -0.2071139 0.02909798 0.4968537 1.00000000 -0.0325153  
## nnet 0.7045017 0.9095609 0.99071227 0.6929567 -0.03251530 1.0000000

Between the models that were implemented, there is a high degree of correlation between them. The highest correlated models are nnet and glm with a correlation of nearly 1 (0.99).

ensemble = caretEnsemble(  
 model\_list,  
 metric = "ROC",  
 trControl=control  
)  
  
  
  
  
saveRDS(ensemble, "ensemble.rds")  
rm(ensemble)  
ensemble = readRDS("ensemble.rds")  
  
  
summary(ensemble)

## The following models were ensembled: ranger, nn, glm, ranger.1, rpart, nnet   
## They were weighted:   
## 2.5765 -1.8388 1.0164 -1.2777 -3.2923 0.5243 -0.3121  
## The resulting ROC is: 0.72  
## The fit for each individual model on the ROC is:   
## method ROC ROCSD  
## ranger 0.7283904 0.02415615  
## nn 0.7043567 0.02743524  
## glm 0.6986381 0.02702541  
## ranger.1 0.7326405 0.01970585  
## rpart 0.6358841 0.03522952  
## nnet 0.7009330 0.02649610

The random forest model performed the best with a ROC of .73

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 316 0  
## Yes 5 633  
##   
## Accuracy : 0.9948   
## 95% CI : (0.9878, 0.9983)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.9882   
##   
## Mcnemar's Test P-Value : 0.07364   
##   
## Sensitivity : 0.9844   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.9922   
## Prevalence : 0.3365   
## Detection Rate : 0.3312   
## Detection Prevalence : 0.3312   
## Balanced Accuracy : 0.9922   
##   
## 'Positive' Class : No   
##

#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 37 38  
## Yes 100 233  
##   
## Accuracy : 0.6618   
## 95% CI : (0.6136, 0.7076)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.5647   
##   
## Kappa : 0.1462   
##   
## Mcnemar's Test P-Value : 2.073e-07   
##   
## Sensitivity : 0.27007   
## Specificity : 0.85978   
## Pos Pred Value : 0.49333   
## Neg Pred Value : 0.69970   
## Prevalence : 0.33578   
## Detection Rate : 0.09069   
## Detection Prevalence : 0.18382   
## Balanced Accuracy : 0.56493   
##   
## 'Positive' Class : No   
##

Wow. On the training data, the model was 99% accurate. However, when the model was applied to the testing data, there was significant degredation, nearly 33% drop down to .66. Unfortunately, the model also dipped below the naive model. This is still a poor performing model….

Stacked Model

stack\_control = trainControl(  
 method = "cv",  
 number = 10,  
 classProbs = TRUE)  
  
stack = caretStack(model\_list,  
 method ="glm",  
 metric = "ROC",  
 trControl = stack\_control)

## Warning in train.default(predobs$preds, predobs$obs, ...): The metric "ROC" was  
## not in the result set. Accuracy will be used instead.

print(stack)

## A glm ensemble of 6 base models: ranger, nn, glm, ranger.1, rpart, nnet  
##   
## Ensemble results:  
## Generalized Linear Model   
##   
## 3563 samples  
## 6 predictor  
## 2 classes: 'No', 'Yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 3207, 3206, 3208, 3206, 3207, 3206, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.7038939 0.2533224

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 316 0  
## Yes 5 633  
##   
## Accuracy : 0.9948   
## 95% CI : (0.9878, 0.9983)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.9882   
##   
## Mcnemar's Test P-Value : 0.07364   
##   
## Sensitivity : 0.9844   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.9922   
## Prevalence : 0.3365   
## Detection Rate : 0.3312   
## Detection Prevalence : 0.3312   
## Balanced Accuracy : 0.9922   
##   
## 'Positive' Class : No   
##

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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 37 38  
## Yes 100 233  
##   
## Accuracy : 0.6618   
## 95% CI : (0.6136, 0.7076)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.5647   
##   
## Kappa : 0.1462   
##   
## Mcnemar's Test P-Value : 2.073e-07   
##   
## Sensitivity : 0.27007   
## Specificity : 0.85978   
## Pos Pred Value : 0.49333   
## Neg Pred Value : 0.69970   
## Prevalence : 0.33578   
## Detection Rate : 0.09069   
## Detection Prevalence : 0.18382   
## Balanced Accuracy : 0.56493   
##   
## 'Positive' Class : No   
##

The stacked model performed just as poorly as the other models when applied to the testing data.. Still no improvements. On to the XGBoost model.

XGBoost Model

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

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

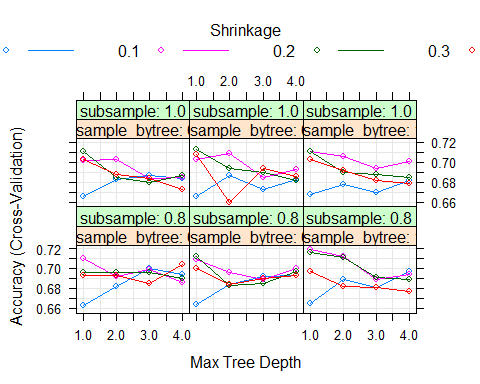
train\_xgb = train\_xgb %>% dplyr::select(-Class.No)  
test\_xgb = test\_xgb %>% dplyr::select(-Class.No)

XGBoost Tuning

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)

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

fitxgb2 = readRDS("fitxgb2.rds")  
plot(fitxgb2)



#fitxgb2

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   
##

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   
##