# Module 5 Assignment 1

## Miguel Fernandez

## BAN-502

# Import libraries  
library(tidyverse)  
library(caret)  
library(nnet)  
library(rpart)  
library(ranger)  
library(caretEnsemble)  
library(xgboost)

# Read in data  
fin = read\_csv("2018Fin.csv")

# Preview structure and summary of dataframe  
# str(fin)  
# summary(fin)

# Data cleaning  
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) %>%  
 drop\_na() %>%  
 mutate(Class = as.factor(Class),  
 Class = fct\_recode(Class, "No" = "0", "Yes" = "1"),  
 Sector = as.factor(Sector)  
 ) %>%  
 filter(`Revenue Growth` <= 1) %>%  
 filter(`EPS Diluted` >= -10, `EPS Diluted` <= 10) %>%  
 filter(`EBITDA Margin` >= -5, `EBITDA Margin` <= 5) %>%  
 filter(priceBookValueRatio >= 0, priceBookValueRatio <= 5) %>%  
 filter(debtEquityRatio >= -1, debtEquityRatio <= 2) %>%  
 filter(debtRatio <= 1) %>%  
 filter(`PE ratio` <= 100) %>%  
 filter(returnOnAssets >= -5, returnOnAssets <= 5) %>%  
 filter(returnOnEquity >= -5, returnOnEquity <= 5) %>%  
 filter(returnOnCapitalEmployed >= -2, returnOnCapitalEmployed <= 2) %>%  
 filter(quickRatio <= 20)

#### Task 1

# Split data into train and test sets  
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)

#### Task 2

# To measure run time  
# start\_time = Sys.time()  
#   
# # Set controls  
# set.seed(1234)  
# fitControl = trainControl(method = "cv",   
# number = 10)  
#   
# # Create grid  
# nnetGrid = expand.grid(size = 1:23,  
# decay = c(0.5, 0.1, 1e-2, 1e-3,  
# 1e-4, 1e-5, 1e-6, 1e-7))  
# set.seed(1234)  
# # Fit the model  
# nnetFit = train(x = fin[ , -1], y = fin$Class,   
# method = "nnet",  
# trControl = fitControl,  
# tuneGrid = nnetGrid,  
# trace = FALSE,  
# verbose = FALSE  
# )  
#   
# end\_time = Sys.time()  
# end\_time-start\_time

# # Save and remove model  
# saveRDS(nnetFit,"nnetfit.rds")  
# rm(nnetFit)

# Read in model  
nnetFit = readRDS("nnetfit.rds")

#### Task 3

# Create predictions on the train set  
predNet = predict(nnetFit, train)  
  
# Create confusion matrix  
confusionMatrix(predNet, train$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 186 57  
## Yes 135 576  
##   
## Accuracy : 0.7987   
## 95% CI : (0.7719, 0.8238)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5206   
##   
## Mcnemar's Test P-Value : 2.745e-08   
##   
## Sensitivity : 0.9100   
## Specificity : 0.5794   
## Pos Pred Value : 0.8101   
## Neg Pred Value : 0.7654   
## Prevalence : 0.6635   
## Detection Rate : 0.6038   
## Detection Prevalence : 0.7453   
## Balanced Accuracy : 0.7447   
##   
## 'Positive' Class : Yes   
##

The neural network created above has an accuracy of approximately 80 percent on the train data. This is a fairly strong score given a naive model of around 66 percent. Looking at the lower-level metrics, the model has a sensitivity of 91 percent and a specificity of 58 percent. The model does very well predicting which stocks will increase in price which is a good start to any trading strategy. But how will the model perform on the test data set.

#### Task 3

# Create predictions on the test set  
predNetTest = predict(nnetFit, test)  
  
# Create confusion matrix  
confusionMatrix(predNetTest, test$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 68 30  
## Yes 69 241  
##   
## Accuracy : 0.7574   
## 95% CI : (0.7128, 0.7982)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 2.753e-05   
##   
## Kappa : 0.4148   
##   
## Mcnemar's Test P-Value : 0.0001339   
##   
## Sensitivity : 0.8893   
## Specificity : 0.4964   
## Pos Pred Value : 0.7774   
## Neg Pred Value : 0.6939   
## Prevalence : 0.6642   
## Detection Rate : 0.5907   
## Detection Prevalence : 0.7598   
## Balanced Accuracy : 0.6928   
##   
## 'Positive' Class : Yes   
##

There is slight degradation in model performance on the test set compared to the train set. Here the accuracy drops to 76 percent but does outperform the naive model. Again, sensitivity is high while specificity drops below 50 percent.

#### Task 5

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

# start\_time = Sys.time()  
# # Build ensemble  
# set.seed(111)  
# model\_list = caretList(  
# x = train[ , -1], y = train$Class,  
# metric = "ROC",  
# trControl = ctrl,  
# 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

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

# Read in model  
model\_list = readRDS("model\_list.rds")

#### Task 6

# Create matrix for model correlation  
modelCor(resamples(model\_list))

## ranger nn glm rpart  
## ranger 1.0000000 0.6492490 0.6157734 0.6609865  
## nn 0.6492490 1.0000000 0.8932273 0.4375211  
## glm 0.6157734 0.8932273 1.0000000 0.4969120  
## rpart 0.6609865 0.4375211 0.4969120 1.0000000

The correlation matrix reveals a strong positive correlation coefficient of 0.89 between the neural network and the logistic regression model. There are also moderately strong relationships between the random forest and the three other models; the neural network, the logistic regression and the classification tree. The coefficients range from 0.61 to 0.66. The weakest correlation is between the neural network and the classification tree, with a coefficient of 0.43. Ideally, the correlations should be weak, near zero, but that is not always the case.

#### Task 7

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

## The following models were ensembled: ranger, nn, glm, rpart   
## They were weighted:   
## 2.7135 -5.1218 0.9214 -1.42 0.0262  
## The resulting ROC is: 0.6876  
## The fit for each individual model on the ROC is:   
## method ROC ROCSD  
## ranger 0.7242414 0.02083234  
## nn 0.7027510 0.01908417  
## glm 0.7009915 0.02468394  
## rpart 0.6454993 0.01967674

The table above shows that the random forest has the highest AUC of 0.72. The neural network and logistic regression performed nearly as well, both scoring an AUC of approximately 0.7. The classification tree performed the worst with an AUC of 0.64. How does the ensemble model perform on the train and test data?

# Predictions for train set  
pred\_ensemble = predict(ensemble, train, type = "raw")  
confusionMatrix(pred\_ensemble, train$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 116 23  
## Yes 205 610  
##   
## Accuracy : 0.761   
## 95% CI : (0.7326, 0.7878)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 3.519e-11   
##   
## Kappa : 0.3778   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9637   
## Specificity : 0.3614   
## Pos Pred Value : 0.7485   
## Neg Pred Value : 0.8345   
## Prevalence : 0.6635   
## Detection Rate : 0.6394   
## Detection Prevalence : 0.8543   
## Balanced Accuracy : 0.6625   
##   
## 'Positive' Class : Yes   
##

# Predictions for test set  
pred\_ensemble\_test = predict(ensemble, test, type = "raw")  
confusionMatrix(pred\_ensemble\_test, test$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 31 24  
## Yes 106 247  
##   
## Accuracy : 0.6814   
## 95% CI : (0.6337, 0.7264)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.2488   
##   
## Kappa : 0.1616   
##   
## Mcnemar's Test P-Value : 1.21e-12   
##   
## Sensitivity : 0.9114   
## Specificity : 0.2263   
## Pos Pred Value : 0.6997   
## Neg Pred Value : 0.5636   
## Prevalence : 0.6642   
## Detection Rate : 0.6054   
## Detection Prevalence : 0.8652   
## Balanced Accuracy : 0.5689   
##   
## 'Positive' Class : Yes   
##

The ensemble model has a slightly lower accuracy than the neural network from above. The sensitivity is 96 percent in the train data which is close to near-perfect predictions of stocks that increased in value. Using the test data set, the ensemble model’s accuracy is approaching the naive model’s performance. However, measures of sensitivity are high in both the train and the test data sets. This model could represent a solid approach for value investors who are looking to identify stocks to buy and hold over a long period of time.

#### Task 8

# Set control  
ctrl2 = trainControl(  
 method = "cv",  
 number = 10,  
 savePredictions = "final",  
 classProbs = TRUE,  
 summaryFunction = twoClassSummary,  
 index=createResample(train$Class)  
)  
  
stack = caretStack(  
 model\_list,  
 method ="glm",  
 metric ="ROC",  
 trControl = ctrl2  
)  
  
print(stack)

## A glm ensemble of 4 base models: ranger, nn, glm, rpart  
##   
## Ensemble results:  
## Generalized Linear Model   
##   
## 3529 samples  
## 4 predictor  
## 2 classes: 'No', 'Yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 954, 954, 954, 954, 954, 954, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.6849777 0.2517901 0.8971418

summary(stack)

##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1581 -1.1112 0.5916 0.8596 1.9459   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 2.71348 0.11655 23.281 <2e-16 \*\*\*  
## ranger -5.12182 0.55198 -9.279 <2e-16 \*\*\*  
## nn 0.92137 0.79554 1.158 0.2468   
## glm -1.41995 0.65633 -2.163 0.0305 \*   
## rpart 0.02624 0.21678 0.121 0.9037   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4456.6 on 3528 degrees of freedom  
## Residual deviance: 3987.3 on 3524 degrees of freedom  
## AIC: 3997.3  
##   
## Number of Fisher Scoring iterations: 3

# Predictions for train set  
pred\_stack = predict(stack, train, type = "raw")  
confusionMatrix(pred\_stack, train$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 116 23  
## Yes 205 610  
##   
## Accuracy : 0.761   
## 95% CI : (0.7326, 0.7878)  
## No Information Rate : 0.6635   
## P-Value [Acc > NIR] : 3.519e-11   
##   
## Kappa : 0.3778   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9637   
## Specificity : 0.3614   
## Pos Pred Value : 0.7485   
## Neg Pred Value : 0.8345   
## Prevalence : 0.6635   
## Detection Rate : 0.6394   
## Detection Prevalence : 0.8543   
## Balanced Accuracy : 0.6625   
##   
## 'Positive' Class : Yes   
##

# Predictions for test set  
pred\_stack\_test = predict(stack, test, type = "raw")  
confusionMatrix(pred\_stack\_test, test$Class, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 31 24  
## Yes 106 247  
##   
## Accuracy : 0.6814   
## 95% CI : (0.6337, 0.7264)  
## No Information Rate : 0.6642   
## P-Value [Acc > NIR] : 0.2488   
##   
## Kappa : 0.1616   
##   
## Mcnemar's Test P-Value : 1.21e-12   
##   
## Sensitivity : 0.9114   
## Specificity : 0.2263   
## Pos Pred Value : 0.6997   
## Neg Pred Value : 0.5636   
## Prevalence : 0.6642   
## Detection Rate : 0.6054   
## Detection Prevalence : 0.8652   
## Balanced Accuracy : 0.5689   
##   
## 'Positive' Class : Yes   
##

The model performance is identical between the ensemble model and the stacked model.

#### Task 9

# Create train and test set for xgboost model  
set.seed(12345)   
train.rows = createDataPartition(y = fin$Class, p = 0.7, list = FALSE)  
train.boost = dplyr::slice(fin, train.rows)  
test.boost = dplyr::slice(fin, -train.rows)

# Create dummy variables in train set  
train\_dummy.boost = dummyVars(" ~ .", data = train.boost)  
train\_xgb = data.frame(predict(train\_dummy.boost, newdata = train.boost))  
  
# Preview structure  
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.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.Cyclical : num 1 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Consumer.Defensive : num 0 0 0 0 0 1 0 0 0 0 ...  
## $ Sector.Energy : num 0 0 0 0 0 0 0 0 1 0 ...  
## $ Sector.Financial.Services : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Healthcare : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Sector.Industrials : 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.Technology : num 0 1 1 0 1 0 1 1 0 1 ...  
## $ 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 ...

# Create dummy variables in test set  
test\_dummy.boost = dummyVars(" ~ .", data = test.boost)  
test\_xgb = data.frame(predict(test\_dummy.boost, newdata = test.boost))

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

# # Measure model run time  
# start\_time = Sys.time()  
#   
# set.seed(999)  
# # Set control  
# ctrl3 = trainControl(method = "cv",  
# number = 5)  
#   
# # Tune the model model  
# 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)  
# )  
#   
# # Build the model  
# fitxgb = train(as.factor(Class.Yes) ~ .,  
# data = train\_xgb,  
# method="xgbTree",  
# tuneGrid = tgrid,  
# trControl=ctrl3)  
#   
# end\_time = Sys.time()  
# end\_time-start\_time

# # Save and remove model  
# saveRDS(fitxgb,"fitxgb.rds")  
# rm(fitxgb)

# Load model  
fitxgb = readRDS("fitxgb.rds")

predxgbtrain = predict(fitxgb, train\_xgb)  
confusionMatrix(as.factor(train\_xgb$Class.Yes), predxgbtrain, 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   
##

predxgbtest = predict(fitxgb, test\_xgb)  
confusionMatrix(as.factor(test\_xgb$Class.Yes), predxgbtest, 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   
##

The xgb model produced inadequate results. Among the train data and the test data, the xgb model’s accuracy did not score as well as the naive model. This means that the model is more accurate if it simply predicted “Yes” for all values in the data set. Using the train data set, the model accuracy is 74 percent while the naive model’s accuracy is 83 percent, nearly ten points higher. The difference is even larger in the test data set. The xgb model scored 66 percent accurate and the naive model is 84 percent accurate. There is also significant degradation among sensitivity between the xgb model and the ensemble and stacked models above. Considering the higher chance of not identifying stocks that are likely to increase in price, the xgb model is not an ideal candidate for this data.