Startup Investments - Predictive Analysis

Group 6

Vineeth Penugonda, Sai Rachana Bandi, Chandra Vardhan

12 May 2020

library(caret)  
library(doParallel)  
library(pROC)

#memory.limit(24000)  
memory.limit(10 \* 10^10)

## Warning: 'memory.limit()' is Windows-specific

## [1] Inf

# Parallelization

We paralleled the operations for faster results.

cl <- makeCluster(detectCores())  
registerDoParallel(cl)

#dat <- readRDS("../Dataset/Data\_Cleansed.rds")  
dat <- readRDS("../Dataset/Data\_CE\_Filtered.rds")  
datCopy <- dat  
datCopy$post\_ipo\_equity <- NULL  
datCopy$status <- NULL  
#datCopy$name <- NULL

# Splitting Data

We split the data into train data and test data. We split the data into 70% train data and 30% test data.

# post\_success  
'%ni%' <- Negate('%in%') # define 'not in' func  
options(scipen=999) # prevents printing scientific notations.  
  
# Prep Training and Test data.  
set.seed(100)  
trainDataIndex <- createDataPartition(datCopy$post\_success, p=0.7, list = F) # 70% training data  
trainData <- datCopy[trainDataIndex, ]  
testData <- datCopy[-trainDataIndex, ]  
table(trainData$post\_success)

##   
## 0 1   
## 11060 1389

# Logistic Regression

# Run algorithms using 10-fold cross validation  
fitControl <- trainControl(method="cv", number= 10, allowParallel = TRUE, verboseIter = TRUE)

## Train a logistic regression model with 10-fold cross-validation  
set.seed(100)  
logit\_fit <- train(post\_success ~ ., data = trainData,  
 trControl = fitControl,  
 method="glm", family=binomial(link='logit'))

## Aggregating results  
## Fitting final model on full training set

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

print(logit\_fit)

## Generalized Linear Model   
##   
## 12449 samples  
## 56 predictor  
## 2 classes: '0', '1'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 11204, 11204, 11204, 11204, 11204, 11204, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.8790276 0.2069696

# In-sample performance  
confusionMatrix(logit\_fit)

## Cross-Validated (10 fold) Confusion Matrix   
##   
## (entries are percentual average cell counts across resamples)  
##   
## Reference  
## Prediction 0 1  
## 0 85.7 8.9  
## 1 3.2 2.2  
##   
## Accuracy (average) : 0.879

# Out-of-sample performance  
confusionMatrix(predict(logit\_fit, newdata = testData),  
 testData$post\_success, positive = '1')

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 4680 510  
## 1 60 85  
##   
## Accuracy : 0.8932   
## 95% CI : (0.8846, 0.9013)   
## No Information Rate : 0.8885   
## P-Value [Acc > NIR] : 0.1431   
##   
## Kappa : 0.1945   
##   
## Mcnemar's Test P-Value : <0.0000000000000002  
##   
## Sensitivity : 0.14286   
## Specificity : 0.98734   
## Pos Pred Value : 0.58621   
## Neg Pred Value : 0.90173   
## Prevalence : 0.11153   
## Detection Rate : 0.01593   
## Detection Prevalence : 0.02718   
## Balanced Accuracy : 0.56510   
##   
## 'Positive' Class : 1   
##

# Gradient Boosting Machine

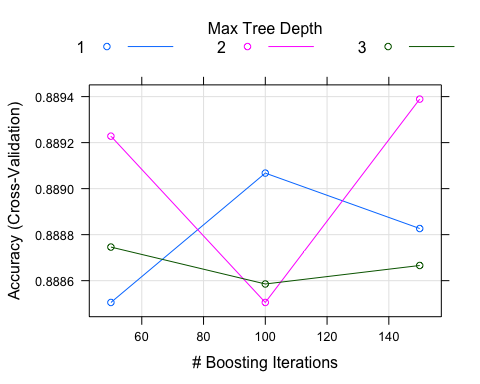
## Train a GBM model with 10-fold cross-validation  
set.seed(100)  
gbm\_fit <- train(post\_success ~ ., data = trainData,  
 trControl = fitControl, method = "gbm",  
 verbose=TRUE)

## Aggregating results  
## Selecting tuning parameters  
## Fitting n.trees = 150, interaction.depth = 2, shrinkage = 0.1, n.minobsinnode = 10 on full training set

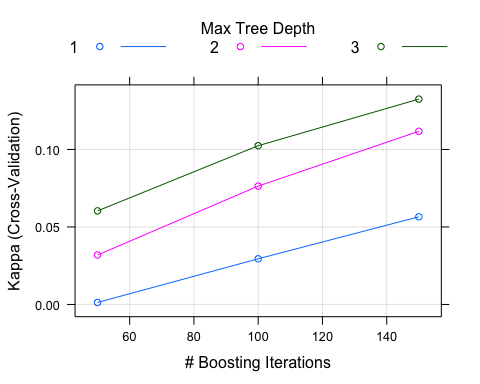
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =  
## "bernoulli", : variable 90: round\_H has no variation.

## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 0.6879 nan 0.1000 0.0054  
## 2 0.6733 nan 0.1000 0.0071  
## 3 0.6617 nan 0.1000 0.0057  
## 4 0.6528 nan 0.1000 0.0041  
## 5 0.6453 nan 0.1000 0.0037  
## 6 0.6369 nan 0.1000 0.0041  
## 7 0.6313 nan 0.1000 0.0026  
## 8 0.6274 nan 0.1000 0.0017  
## 9 0.6245 nan 0.1000 0.0011  
## 10 0.6205 nan 0.1000 0.0017  
## 20 0.5904 nan 0.1000 0.0014  
## 40 0.5710 nan 0.1000 0.0003  
## 60 0.5589 nan 0.1000 0.0003  
## 80 0.5524 nan 0.1000 0.0000  
## 100 0.5467 nan 0.1000 0.0000  
## 120 0.5424 nan 0.1000 -0.0001  
## 140 0.5379 nan 0.1000 -0.0001  
## 150 0.5357 nan 0.1000 -0.0000

# Plot resampling profile by accuracy   
plot(gbm\_fit)



# Plot resampling profile by kappa statistic   
plot(gbm\_fit, metric = "Kappa")



# In-sample performance  
confusionMatrix(gbm\_fit)

## Cross-Validated (10 fold) Confusion Matrix   
##   
## (entries are percentual average cell counts across resamples)  
##   
## Reference  
## Prediction 0 1  
## 0 88.1 10.3  
## 1 0.8 0.9  
##   
## Accuracy (average) : 0.8894

# Out-of-sample performance  
confusionMatrix(predict(gbm\_fit, newdata = testData),  
 testData$post\_success, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 4699 536  
## 1 41 59  
##   
## Accuracy : 0.8918   
## 95% CI : (0.8832, 0.9001)   
## No Information Rate : 0.8885   
## P-Value [Acc > NIR] : 0.224   
##   
## Kappa : 0.1423   
##   
## Mcnemar's Test P-Value : <0.0000000000000002  
##   
## Sensitivity : 0.09916   
## Specificity : 0.99135   
## Pos Pred Value : 0.59000   
## Neg Pred Value : 0.89761   
## Prevalence : 0.11153   
## Detection Rate : 0.01106   
## Detection Prevalence : 0.01874   
## Balanced Accuracy : 0.54525   
##   
## 'Positive' Class : 1   
##

# KNN

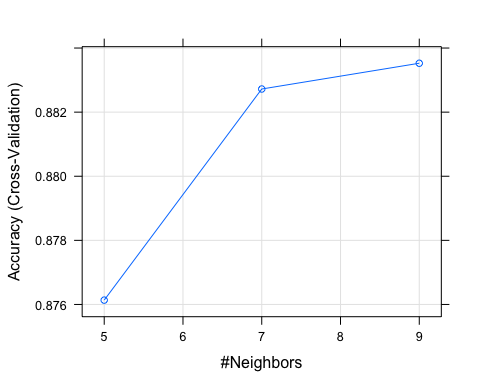
## Train a KNN model with 10-fold cross-validation  
set.seed(100)  
knn\_fit <- train(post\_success ~ ., data = trainData, method="knn", trControl=fitControl)

## Aggregating results  
## Selecting tuning parameters  
## Fitting k = 9 on full training set

# In-sample performance  
confusionMatrix(knn\_fit)

## Cross-Validated (10 fold) Confusion Matrix   
##   
## (entries are percentual average cell counts across resamples)  
##   
## Reference  
## Prediction 0 1  
## 0 87.8 10.6  
## 1 1.0 0.6  
##   
## Accuracy (average) : 0.8835

# Plot accuracy across different n values  
plot(knn\_fit)



# Out-of-sample performance  
confusionMatrix(predict(knn\_fit, newdata = testData),  
 testData$post\_success, positive = '1')

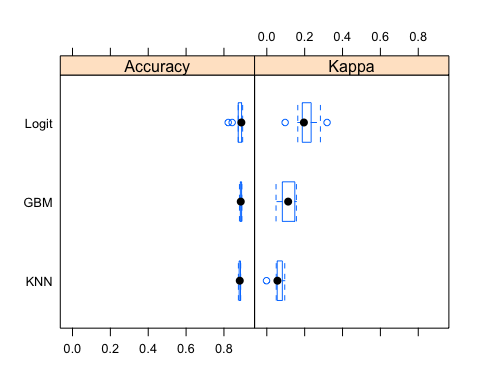
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 4691 555  
## 1 49 40  
##   
## Accuracy : 0.8868   
## 95% CI : (0.878, 0.8952)   
## No Information Rate : 0.8885   
## P-Value [Acc > NIR] : 0.662   
##   
## Kappa : 0.0906   
##   
## Mcnemar's Test P-Value : <0.0000000000000002  
##   
## Sensitivity : 0.067227   
## Specificity : 0.989662   
## Pos Pred Value : 0.449438   
## Neg Pred Value : 0.894205   
## Prevalence : 0.111528   
## Detection Rate : 0.007498   
## Detection Prevalence : 0.016682   
## Balanced Accuracy : 0.528445   
##   
## 'Positive' Class : 1   
##

# Comparing the models using performance metrics

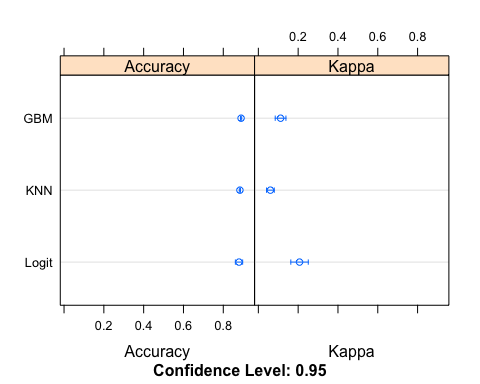
resamps <- resamples(list(Logit=logit\_fit, GBM=gbm\_fit, KNN=knn\_fit))  
  
# Summarize the resamples  
summary(resamps)

##   
## Call:  
## summary.resamples(object = resamps)  
##   
## Models: Logit, GBM, KNN   
## Number of resamples: 10   
##   
## Accuracy   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## Logit 0.8224900 0.8781124 0.8919679 0.8790276 0.8931512 0.8987952 0  
## GBM 0.8819277 0.8871486 0.8887550 0.8893891 0.8931727 0.8946945 0  
## KNN 0.8763052 0.8805221 0.8835341 0.8835251 0.8870808 0.8883534 0  
##   
## Kappa   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## Logit 0.096850736 0.18828043 0.1959385 0.20696964 0.22648819 0.31827664 0  
## GBM 0.048753866 0.08439806 0.1127729 0.11171169 0.14658147 0.15631066 0  
## KNN -0.001577617 0.05481420 0.0560808 0.06057304 0.07842618 0.09475873 0

# Boxplots of resamples  
bwplot(resamps)



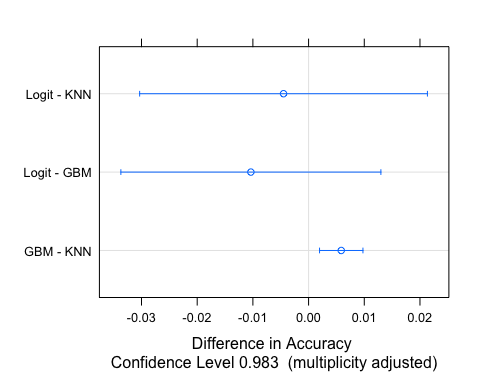
# Dot plots of resamples  
dotplot(resamps)



difValues <- diff(resamps)  
  
summary(difValues)

##   
## Call:  
## summary.diff.resamples(object = difValues)  
##   
## p-value adjustment: bonferroni   
## Upper diagonal: estimates of the difference  
## Lower diagonal: p-value for H0: difference = 0  
##   
## Accuracy   
## Logit GBM KNN   
## Logit -0.010362 -0.004498  
## GBM 0.675736 0.005864  
## KNN 1.000000 0.005052   
##   
## Kappa   
## Logit GBM KNN   
## Logit 0.09526 0.14640  
## GBM 0.002896 0.05114  
## KNN 0.00004291 0.005046

#bwplot(difValues, layout = c(3, 1))  
dotplot(difValues)



accu <- c(confusionMatrix(predict(logit\_fit, newdata=testData),testData$post\_success)$overall['Accuracy'],  
 confusionMatrix(predict(gbm\_fit, newdata=testData),testData$post\_success)$overall['Accuracy'],  
 confusionMatrix(predict(knn\_fit, newdata=testData),testData$post\_success)$overall['Accuracy'])

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

kappa <- c(confusionMatrix(predict(logit\_fit, newdata=testData),testData$post\_success)$overall['Kappa'],  
 confusionMatrix(predict(gbm\_fit, newdata=testData),testData$post\_success)$overall['Kappa'],  
 confusionMatrix(predict(knn\_fit, newdata=testData),testData$post\_success)$overall['Kappa'])

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

sensi <- c(confusionMatrix(predict(logit\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Sensitivity'],  
 confusionMatrix(predict(gbm\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Sensitivity'],  
 confusionMatrix(predict(knn\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Sensitivity'])

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

speci <- c(confusionMatrix(predict(logit\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Specificity'],  
 confusionMatrix(predict(gbm\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Specificity'],  
 confusionMatrix(predict(knn\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Specificity'])

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

preci <- c(confusionMatrix(predict(logit\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Precision'],  
 confusionMatrix(predict(gbm\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Precision'],  
 confusionMatrix(predict(knn\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['Precision'])

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

f1 <- c(confusionMatrix(predict(logit\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['F1'],  
 confusionMatrix(predict(gbm\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['F1'],  
 confusionMatrix(predict(knn\_fit, newdata=testData),testData$post\_success,positive='1')$byClass['F1'])

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

auc <- c(roc(testData$post\_success, as.numeric(predict(logit\_fit, newdata=testData)))$auc,  
 roc(testData$post\_success, as.numeric(predict(gbm\_fit, newdata=testData)))$auc,  
 roc(testData$post\_success, as.numeric(predict(knn\_fit, newdata=testData)))$auc)

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

data.frame(Accuracy = round(accu,3),  
 Kappa = round(kappa,3),  
 Precision = round(preci,3),  
 Recall = round(sensi,3),   
 Specificity = round(speci,3),  
 F1 = round(f1,3),  
 AUC = round(auc,3),  
 Balanced.Accuracy = round((sensi + speci)/2,3),  
 row.names = c('Logit','GBM','KNN'))

## Accuracy Kappa Precision Recall Specificity F1 AUC Balanced.Accuracy  
## Logit 0.893 0.195 0.586 0.143 0.987 0.230 0.565 0.565  
## GBM 0.892 0.142 0.590 0.099 0.991 0.170 0.545 0.545  
## KNN 0.887 0.092 0.466 0.067 0.990 0.117 0.529 0.529

logit\_fit\_roc <- roc(testData$post\_success, as.numeric(predict(logit\_fit, newdata=testData)))

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

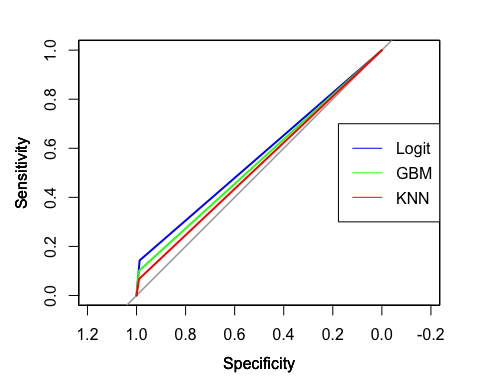
gbm\_fit\_roc <- roc(testData$post\_success, as.numeric(predict(gbm\_fit, newdata=testData)))

## Setting levels: control = 0, case = 1  
## Setting direction: controls < cases

knn\_fit\_roc <- roc(testData$post\_success, as.numeric(predict(knn\_fit, newdata=testData)))

## Setting levels: control = 0, case = 1  
## Setting direction: controls < cases

plot(logit\_fit\_roc, col = "blue")  
par(new = TRUE)  
plot(gbm\_fit\_roc, col = "green", xaxt = "n", yaxt = "n")  
par(new = TRUE)  
plot(knn\_fit\_roc, col = "red", xaxt = "n", yaxt = "n")  
  
legend("right", legend = c("Logit", "GBM", "KNN"), col = c("blue", "green", "red"), lty = 1)



# RESULTS

We used 3 algorithms, namely the KNN , logistic regression and the gradient boosting model (GBM) to train 3 different machine learning models. The features obtained during the data preprocessing phase were used to fit the 3 different models.

The AUC score for each of the machine learning models were compared and the model with the maximum AUC score is considered for the given dataset.

Based on AUC score, we can observe that our best model is Gradient Boosting Machine (GBM). Logistic Regression (GLM) is the second best model.

stopCluster(cl)