Novel Classifier and Regressor with Random Forests

Table of Contents

### Include libraries

suppressMessages(library(dplyr))  
suppressMessages(library(pROC))  
suppressMessages(library(party))  
suppressMessages(library(caret))  
suppressMessages(library(xgboost))  
suppressMessages(library(randomForest))  
suppressMessages(library(miscTools))

### Load data

setwd("..")  
customerInfo = read.csv("Data/customerInfo.csv")  
tableE = read.csv("Data/tableE.csv")

### Clean data

set.seed(123) # For reproducibility  
colnames(tableE)[colnames(tableE) == "Company"] = "Customer.ID"  
  
pool = merge(customerInfo, tableE, by = "Customer.ID")  
rm(customerInfo, tableE)  
  
pool$Customer.Size = as.factor(pool$Customer.Size)  
pool$Closed\_Time = as.integer(as.Date(pool$Closed) - as.Date(pool$Open))  
  
pool = pool %>% select(Customer.Size, Open\_Year, Open\_Month,   
 Open\_Day, Closed\_Year, Closed\_Month, Closed\_Day, Closed\_Time)  
  
pool = na.omit(pool)  
n\_classes = length(unique(pool$Customer.Size))

* NOTE: Merging all tables results in an OOM error for tables B and D.

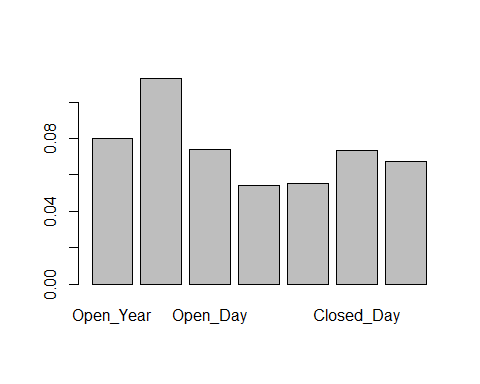
### Split data

split\_index = floor(0.8 \* nrow(pool))  
  
set.seed(123) # For reproducibility  
train\_ind = sample(seq\_len(nrow(pool)), size = split\_index)  
train = pool[train\_ind, ]  
test = pool[-train\_ind, ]  
  
rm(pool)

### Random Forest predicting Customer.Size

#### Model creation and feature importance

set.seed(123) # For reproducibility  
rf\_model = cforest(Customer.Size ~ ., data = train, control = cforest\_unbiased(mtry = 2,   
 ntree = 50))  
vi\_cs = varimp(rf\_model) # get variable importance, based on mean decrease in accuracy  
barplot(vi\_cs)

 \* Higher scores imply more importance in classifying Customer.Size. Open\_Month seems to be the most important while the rest are around the same amount of importance.

#### Get training predictions

set.seed(123) # For reproducibility  
predicted = predict(rf\_model, newdata = train, OOB = TRUE, type = "response")

#### ROC Curve

# ROC curve  
rf\_cs\_auc = auc(as.numeric(train$Customer.Size), as.numeric(predicted))

## Warning in roc.default(response, predictor, auc = TRUE, ...): 'response'  
## has more than two levels. Consider setting 'levels' explicitly or using  
## 'multiclass.roc' instead

## Setting levels: control = 1, case = 2

## Setting direction: controls < cases

rf\_cs\_auc

## Area under the curve: 0.6932

* AUC for the ROC curve is above. This implies that the model has that probability that it will be able to distinguish between positive class and negative class.

#### In-sample Error

# Compute in-sample results  
rf\_cs\_cm = confusionMatrix(predicted, train$Customer.Size)  
rf\_cs\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3  
## 0 115 0 1 0  
## 1 1 228 8 7  
## 2 7 11 564 18  
## 3 77 240 380 3779  
##   
## Overall Statistics  
##   
## Accuracy : 0.862   
## 95% CI : (0.8526, 0.8711)  
## No Information Rate : 0.6998   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6548   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3  
## Sensitivity 0.57500 0.47599 0.5918 0.9934  
## Specificity 0.99981 0.99677 0.9920 0.5729  
## Pos Pred Value 0.99138 0.93443 0.9400 0.8443  
## Neg Pred Value 0.98402 0.95166 0.9196 0.9740  
## Prevalence 0.03679 0.08812 0.1753 0.6998  
## Detection Rate 0.02116 0.04194 0.1038 0.6952  
## Detection Prevalence 0.02134 0.04489 0.1104 0.8234  
## Balanced Accuracy 0.78740 0.73638 0.7919 0.7832

* Accuracy is roughly 84%.

#### Get test predictions

set.seed(123) # For reproducibility  
predicted\_test = predict(rf\_model, newdata = test, OOB = TRUE,   
 type = "response")

#### In-sample Error

# Compute in-sample results  
rf\_cs\_cm = confusionMatrix(data = predicted\_test, reference = test$Customer.Size)  
rf\_cs\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3  
## 0 32 1 0 1  
## 1 0 65 3 4  
## 2 1 5 152 15  
## 3 23 83 95 879  
##   
## Overall Statistics  
##   
## Accuracy : 0.83   
## 95% CI : (0.809, 0.8496)  
## No Information Rate : 0.6615   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.617   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3  
## Sensitivity 0.57143 0.42208 0.6080 0.9778  
## Specificity 0.99847 0.99419 0.9811 0.5630  
## Pos Pred Value 0.94118 0.90278 0.8786 0.8139  
## Neg Pred Value 0.98189 0.93085 0.9174 0.9283  
## Prevalence 0.04121 0.11332 0.1840 0.6615  
## Detection Rate 0.02355 0.04783 0.1118 0.6468  
## Detection Prevalence 0.02502 0.05298 0.1273 0.7947  
## Balanced Accuracy 0.78495 0.70813 0.7945 0.7704

* Test accuracy is roughly 82%, or about 2% lower than that of the training set. There likely isn’t much room for improvement through hyperparameter tuning.

### Random Forest predicting Closed Time

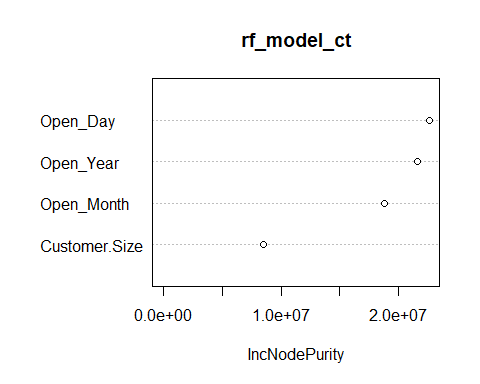
#### Model creation and feature importance

set.seed(123) # For reproducibility  
  
train = train %>% select(Customer.Size, Open\_Year, Open\_Month,   
 Open\_Day, Closed\_Time)  
test = test %>% select(Customer.Size, Open\_Year, Open\_Month,   
 Open\_Day, Closed\_Time)  
  
rf\_model\_ct <- randomForest(Closed\_Time ~ ., data = train, mtry = 3,   
 importance = TRUE, na.action = na.omit)  
print(rf\_model\_ct)

##   
## Call:  
## randomForest(formula = Closed\_Time ~ ., data = train, mtry = 3, importance = TRUE, na.action = na.omit)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 3  
##   
## Mean of squared residuals: 7303.433  
## % Var explained: 55.7

### Feature Importance

varImpPlot(rf\_model\_ct, type = 2)

 \* Importance scores (higher is better).

#### Train Metrics

set.seed(123) # For reproducibility  
rf\_ct\_train\_predictions = predict(rf\_model\_ct, train)  
(r2 = rSquared(train$Closed\_Time, train$Closed\_Time - rf\_ct\_train\_predictions))

## [,1]  
## [1,] 0.7687596

(mse = mean((train$Closed\_Time - rf\_ct\_train\_predictions)^2))

## [1] 3812.327

(rmse = sqrt(mse))

## [1] 61.74405

* Above training performance metrics are above. With a root mean squared error of about 60 days, this may be acceptable for predicting close dates depending on the application.

#### Test Metrics

set.seed(123) # For reproducibility  
rf\_ct\_test\_predictions = predict(rf\_model\_ct, test)  
(mse = mean((test$Closed\_Time - rf\_ct\_test\_predictions)^2))

## [1] 8245.751

(rmse = sqrt(mse))

## [1] 90.80612

* Testing performance metrics are above. This shows that the model might be tuned for better generalization as the train and test performance has a small gap. RMSE for the test set is about 90 days higher than that of the training set.

### Conclusions

It appears that this novel application of random forests shows that some of the data can be well generalized by machine and statistical learning models, but are these applications particularly useful? One feature was engineered, Closed\_Time. This was an elapsed days amount of time between open and closed and it seems that predicting it can be done to 60/90 days RMSE (train/test).

Future work can examine other feature from tables A, B, and D which when joined, cause memory problems on low memory machines. Since we are limited to R, we likely have three paths to work around this memory issue. We could subset from the database, divide data and train different models eventually joining the models, or we can compress the data in the data and then pull, which likely isn’t going to help all that much. Yet, the features from other variables might be improve model performance.

Otherwise, we can also explore other models and algorithms but there may be tradeoffs between interpretability and performance. We can also explore other features such as lag variables which can inform algorithms of a customer’s past transactions.