# Chicago Crime Model - Phase 2

### Miguel Fernandez

### BAN-502

Phase 2 of the BAN-502 course project contains model building efforts using the 2018 crime data set from Chicago, Illinois. Data cleaning and exploratory data analysis was conducted in Phase 1. Several models will be constructed and their performance evaluated before making a recommendation. We will begin by reading in the data and performing feature engineering.

library(tidyverse)  
library(caret)  
library(nnet)  
library(rpart)  
library(ranger)  
library(caretEnsemble)  
library(lubridate)

# Read in data  
df = read\_csv("chicago\_crimes.csv")

# Convert variables to factors  
df <- df %>%  
 mutate(  
 Arrest = as\_factor(as.character(Arrest)),  
 Arrest = fct\_recode(Arrest, "Yes" = "TRUE", "No" = "FALSE"),  
 Domestic = as\_factor(as.character(Domestic)),  
 Domestic = fct\_recode(Domestic, "Yes" = "TRUE", "No" = "FALSE"),  
 Primary.Type = as\_factor(Primary.Type),  
 Location.Description = as\_factor(Location.Description),  
 date.only = as.Date(Date),  
 hr = hour(Date),  
 hr = as\_factor(hr),  
 mnth = month(Date),  
 mnth = as\_factor(mnth),  
 day.of.week = weekdays(Date),  
 day.of.week = as\_factor(day.of.week),  
 day.of.month = day(Date),  
 day.of.month = as\_factor(day.of.month),  
 street.type = str\_extract(Block, '\\w+$'),  
 street.type = as\_factor(street.type),  
 hr = as\_factor(hr),  
 day.of.week = as\_factor(day.of.week),  
 street.type = as\_factor(street.type)  
 )

## Warning: Unknown levels in `f`: TRUE, FALSE  
  
## Warning: Unknown levels in `f`: TRUE, FALSE

# Reduce the number of factor levels   
levels(df$Primary.Type)[table(df$Primary.Type) < 200] <- 'Other'  
levels(df$Location.Description)[table(df$Location.Description) < 200] <- 'Other'  
levels(df$street.type)[table(df$street.type) < 200] <- 'Other'

We will select a subset of the data for modeling.

df2 = df %>%  
 select(Arrest, Domestic, hr, day.of.week,  
 Primary.Type, Location.Description,  
 day.of.month, mnth, street.type,)

The data will be split into a train set and a test set using the createDataPartition function.

# Split data into train and test sets  
set.seed(1234)  
train.rows <- createDataPartition(y=df2$Arrest,  
 p=0.7,  
 list = FALSE)  
train <- dplyr::slice(df2, train.rows)  
test <- dplyr::slice(df2, -train.rows)

Now that the data set is split, we will begin by building a random forest model. But first a control object will be used to provide a set of parameters to the model.

# Set control  
ctrl = trainControl(  
 method = "cv",  
 number = 10,  
 savePredictions = "final",  
 classProbs = TRUE,  
 summaryFunction = twoClassSummary,  
 index=createResample(train$Arrest)  
)

# # Measure model run time  
# start\_time = Sys.time()  
#   
# # Fit random forest model to train data  
# rf\_fit = train(x = as.matrix(train[, -1]),  
# y=as.matrix(train$Arrest),  
# method = "ranger",  
# importance = "permutation",  
# num.trees = 100,  
# trControl = ctrl)  
#   
# # Measure model run time  
# end\_time = Sys.time()  
# end\_time-start\_time

The random forest model took approximately 34 seconds to run. The model will be saved for efficiency.

# # Save and remove model  
# saveRDS(rf\_fit, "rf\_fit.rds")  
# rm(rf\_fit)  
  
# Read in model  
rf\_fit = readRDS("rf\_fit.rds")

# Display variable importance  
varImp(rf\_fit)

## ranger variable importance  
##   
## Overall  
## Primary.Type 100.00000  
## Location.Description 18.51267  
## Domestic 6.13812  
## day.of.month 1.11183  
## hr 0.59396  
## mnth 0.51502  
## street.type 0.02248  
## day.of.week 0.00000

The table above contains scores for variable importance in the model. The Primary Type variable appears to be the most important in the random forest model. The variables for hour of the day, month of the year, street type and day of week appear to be inconsequential in the model. Using the predict function, we will generate predictions on the train data set and use a confusion matrix to evaluate performance.

# Create predictions on train set  
pred.train = predict(rf\_fit, train)  
head(pred.train)

## [1] No No No No No No  
## Levels: No Yes

# Create confusion matrix  
confusionMatrix(pred.train, train$Arrest, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 8049 1288  
## Yes 298 826  
##   
## Accuracy : 0.8484   
## 95% CI : (0.8414, 0.8552)  
## No Information Rate : 0.7979   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4303   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.39073   
## Specificity : 0.96430   
## Pos Pred Value : 0.73488   
## Neg Pred Value : 0.86205   
## Prevalence : 0.20208   
## Detection Rate : 0.07896   
## Detection Prevalence : 0.10745   
## Balanced Accuracy : 0.67751   
##   
## 'Positive' Class : Yes   
##

The random forest model has an accuracy of 0.8484 which is greater than the naive model. The model correctly identified 39 percent of the true positive values. Note that the specificity of the model is very strong. The unbalanced values of sensitivity and specificity are likely caused by a heavily skewed data set. Recall that in Phase 1 we discovered that 80 percent of the crimes in the data set have had no arrests made. This is also visible in the confusion matrix which displays a large number of true negative values.

# Make predictions on test set  
pred.test = predict(rf\_fit, test)  
head(pred.test)

## [1] No No No No Yes No   
## Levels: No Yes

# Create confusion matrix  
confusionMatrix(pred.test, test$Arrest, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 3427 544  
## Yes 150 361  
##   
## Accuracy : 0.8452   
## 95% CI : (0.8342, 0.8556)  
## No Information Rate : 0.7981   
## P-Value [Acc > NIR] : 3.121e-16   
##   
## Kappa : 0.4263   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.39890   
## Specificity : 0.95807   
## Pos Pred Value : 0.70646   
## Neg Pred Value : 0.86301   
## Prevalence : 0.20192   
## Detection Rate : 0.08054   
## Detection Prevalence : 0.11401   
## Balanced Accuracy : 0.67848   
##   
## 'Positive' Class : Yes   
##

The model performed nearly identical on both, the train and the test data. This phenomenon is likely evidence that the model is fit well to the data. While performance is fairly strong, we will try to improve the model’s sensitivity using a neural network.

# # Create grid  
# nnetGrid = expand.grid(size = 1:8,  
# decay = c(0.5, 0.1, 1e-2, 1e-3,  
# 1e-4,1e-5,1e-6,1e-7))  
#   
# # Measure model run time  
# start\_time = Sys.time()  
#   
#   
# set.seed(1234)  
# # Fit the model  
# nnetFit = train(x = as.data.frame(df2[ , -1]), y = df2$Arrest,  
# method = "nnet",  
# trControl = ctrl,  
# tuneGrid = nnetGrid,  
# trace = FALSE,  
# verbose = FALSE  
# )  
#   
# # Measure model run time  
# end\_time = Sys.time()  
# end\_time-start\_time

The neural network took about 48 minutes to run. We will save the model so it will not have to be run again and it can be loaded into the analysis.

# # Save and remove model  
# saveRDS(nnetFit, "nnetFit.rds")  
# rm(nnetFit)  
  
# Read in model  
nnetFit = readRDS("nnetFit.rds")

# Create predictions on train set  
pred.train.nnet = predict(nnetFit, train)  
head(pred.train.nnet)

## [1] No No No No No No  
## Levels: No Yes

# Create confusion matrix  
confusionMatrix(pred.train.nnet, train$Arrest, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 8128 1073  
## Yes 219 1041  
##   
## Accuracy : 0.8765   
## 95% CI : (0.87, 0.8827)  
## No Information Rate : 0.7979   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.549   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.49243   
## Specificity : 0.97376   
## Pos Pred Value : 0.82619   
## Neg Pred Value : 0.88338   
## Prevalence : 0.20208   
## Detection Rate : 0.09951   
## Detection Prevalence : 0.12045   
## Balanced Accuracy : 0.73310   
##   
## 'Positive' Class : Yes   
##

The neural network model has an accuracy of 0.8765 on the train data set. The naive model has an accuracy of 0.7979. When we compare the accuracy of the neural network to the random forest created above with an accuracy of 0.8484, we see there is some model improvement using the neural network. The specificity measure is very strong, likely the result of the skewed data set favoring crimes with no arrests made. There is also an improvement in sensitivity. The neural network was able to correctly identify another 10 percent of positive values. Let us examine how well the neural network performs on the test data.

# Create predictions on train set  
pred.test.nnet = predict(nnetFit, test)  
head(pred.test.nnet)

## [1] No No No No No No  
## Levels: No Yes

# Create confusion matrix  
confusionMatrix(pred.test.nnet, test$Arrest, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 3470 457  
## Yes 107 448  
##   
## Accuracy : 0.8742   
## 95% CI : (0.8641, 0.8837)  
## No Information Rate : 0.7981   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5436   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.49503   
## Specificity : 0.97009   
## Pos Pred Value : 0.80721   
## Neg Pred Value : 0.88363   
## Prevalence : 0.20192   
## Detection Rate : 0.09996   
## Detection Prevalence : 0.12383   
## Balanced Accuracy : 0.73256   
##   
## 'Positive' Class : Yes   
##

The neural network performed very similarly between the train and test data sets. As with the random forest, this lends confidence to model performance and confirms that the model did not over fit the data. In a final attempt, we will build an ensemble model combining a logistic regression model, a classification tree, a random forest and a neural network. We are increasing model complexity with the hopes of improving our results.

# start\_time = Sys.time()  
# # Build ensemble  
# set.seed(111)  
# model\_list = caretList(  
# x = as.data.frame(train[ , -1]), y = train$Arrest,  
# metric = "ROC",  
# trControl = ctrl,  
# methodList = c("glm","rpart"),  
# tuneList = list(ranger = caretModelSpec(  
# method = "ranger",  
# max.depth = 5,  
# tuneGrid = expand.grid(  
# mtry = 1:8,  
# splitrule = c("gini",  
# "extratrees",  
# "hellinger"),  
# min.node.size = 1:3)),  
# nn = caretModelSpec(  
# method = "nnet",  
# tuneGrid = expand.grid(  
# size = 1:8,  
# decay = c(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")

The ensemble model has been built and saved. It took nearly 51 minutes to run. A correlation matrix will reveal relationships between the four models in the ensemble.

modelCor(resamples(model\_list))

## ranger nn glm rpart  
## ranger 1.0000000 -0.496069 0.8110474 0.4057137  
## nn -0.4960690 1.000000 -0.3329580 -0.1360430  
## glm 0.8110474 -0.332958 1.0000000 0.3180751  
## rpart 0.4057137 -0.136043 0.3180751 1.0000000

The strongest correlation exists between the random forest and the logistic regression model. Interestingly, the neural network has a moderately strong negative correlation with the random forest. The neural network is also negatively correlated with the logistic regression and classification tree models. Typically, we would like to see no correlation among the models.

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

## The following models were ensembled: ranger, nn, glm, rpart   
## They were weighted:   
## 5.5716 -7.3748 -1.1799 -0.611 0.1795  
## The resulting ROC is: 0.8511  
## The fit for each individual model on the ROC is:   
## method ROC ROCSD  
## ranger 0.8512361 0.006261732  
## nn 0.8210637 0.008584103  
## glm 0.8363802 0.007223749  
## rpart 0.8012157 0.031542498

The table above shows that the random forest has the highest AUC of 0.85. There is some degradation looking at the three remaining models. We will generate predictions and analyze model performance.

# Create predictions on train set  
pred.train.ensemble = predict(ensemble, train)  
head(pred.train.ensemble)

## [1] No No No No No No  
## Levels: No Yes

# Create confusion matrix  
confusionMatrix(pred.train.ensemble, train$Arrest, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 8142 1127  
## Yes 205 987  
##   
## Accuracy : 0.8727   
## 95% CI : (0.8661, 0.879)  
## No Information Rate : 0.7979   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5284   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.46689   
## Specificity : 0.97544   
## Pos Pred Value : 0.82802   
## Neg Pred Value : 0.87841   
## Prevalence : 0.20208   
## Detection Rate : 0.09435   
## Detection Prevalence : 0.11395   
## Balanced Accuracy : 0.72116   
##   
## 'Positive' Class : Yes   
##

The single neural network from earlier performed slightly better than the model ensemble on the train data set. The neural network also has a sensitivity that is three points higher than the model ensemble. It appears that adding model complexity has caused some degradation in performance. This is at least true among the train data but let us examine how the ensemble model performs on the test data.

# Create predictions on train set  
pred.test.ensemble = predict(ensemble, test)  
head(pred.test.ensemble)

## [1] No No No No No No  
## Levels: No Yes

# Create confusion matrix  
confusionMatrix(pred.test.ensemble, test$Arrest, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 3478 486  
## Yes 99 419  
##   
## Accuracy : 0.8695   
## 95% CI : (0.8593, 0.8792)  
## No Information Rate : 0.7981   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.518   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.46298   
## Specificity : 0.97232   
## Pos Pred Value : 0.80888   
## Neg Pred Value : 0.87740   
## Prevalence : 0.20192   
## Detection Rate : 0.09349   
## Detection Prevalence : 0.11557   
## Balanced Accuracy : 0.71765   
##   
## 'Positive' Class : Yes   
##

As in the previous models, the test data set scores equally as well as on the train data set. We can then conclude that out of the three models that were built, the neural network performed best. Model accuracy is fairly strong but further work needs to be done to increase the model’s sensitivity before it is deployed. The neural network model has a sensitivity of 0.49. This metric is particularly weak as the model only identified 49 percent of crimes where an actual arrest was made. With the model as is, we cannot confidently identify circumstances where and when reports of crimes requiring arrests will happen. This will lead to ineffective allocation of officers and resources.