Predicting the fraud in self-checkout stores

STAT/MATH 8456 - CONTEST #3

🙋‍♂️ Hi, my name is Bikram Maharjan

Presentation Topic: Predicting the fraud in self-checkout stores

Date: Thursday, May 5th 2020

Time: 4:00 PM

University of Nebraska Omaha

The goal of the presentation

- Steps I took to predict the fraud in self-checkout stores or the fraudulent purchases

- Various Machine Learning Models

- The optimal solution I came up with using 3 different models

Model [design] Matrix

- The data was ready to be modeled - I did not introduce new variables

- There were no NA values

```
# Read train and Test dataset
train <- read.csv('../input/train.csv')
test <- read.csv('../input/test.csv')
```

```
sum(is.na(train)) + sum(is.na(test))
```

```
0
```

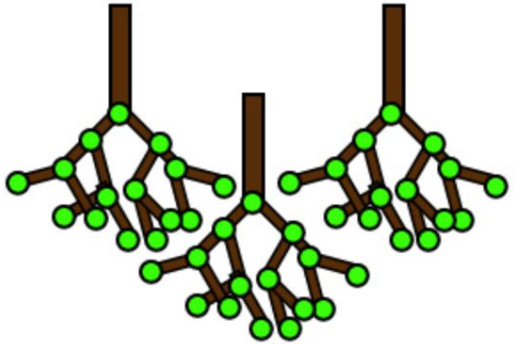
- I used all the available predictors except for **id**

Model Selection

- I attempted to find the best prediction using the following 3 models
    - **Support Vector Matrix**
    - **Random Forest**
    - **XGBoost**
- The optimal prediction was given by XGBoost, followed by Random Forest and then Support Vector Matrix
- The score also was also dependent on the time I kept on each model
    - Such as parameter tuning and feature selections

Model Selection: Random Forest

- This is the first model I used
- Gave an prediction accuracy of **0.97690**



```
# - Using Random Forest
# - Score: 0.97690

rf_model <- randomForest(train$fraud~.,
                         data = train[,-1],
                         mtry= 4, ntree=750,
                         mindev = 0.001,
                         importance=TRUE)

rf_predict <- predict(rf_model,test[,-c(1)])

fitted.results <- ifelse(rf_predict > 0.5,1,0)
```

Parameter Tuning

- **mtry** = 4
- **Ntree** = 750

Kaggle Public Score: **0.97690**

Model Selection: Support Vector Matrix

- AKA SVM
- Didn't put a lot of time on it
- Mostly experimenting it

```
# - Score: 0.6321

SVM_model = svm(train$fraud~.,
                data = train[,-1],
                kernel = "polynomial",
                cost = 5,
                scale = FALSE)

svm_predict <- predict(SVM_model,test[,-c(1)])

fitted.results <- ifelse(svm_predict > 0.5,1,0)
```

- Prediction Accuracy was **0.6321**

Parameter Tuning

- **kernel** = "polynomial"
- **cost** = 5

Kaggle Public Score: **0.6321**

The amazing - XGBoost

- XGBoost stands for e**X**treme **G**radient **Boosting**

- Most of the time was spent using XGBoost

- Highest score:
  - Public Leaderboard: **0.99083**
  - Private Leaderboard: **0.99053**

- Highest score (after presentation)
  - Public Leaderboard: **0.99221**
  - Private Leaderboard: **0.99231**

The amazing - XGBoost

XGBoost

- Why use XGBoost?

- Used for both Regression and Classification problems

- Works really fast - Speed and Execution
    - Memory optimization
    - Cache optimization
    - In comparison with Random Forest
        - **Very Fast**

XGBoost - Data Preparation

XGBoost

Step #1

```
# Read Train and Test dataset
train <- read.csv('../input/train.csv')
test <- read.csv('../input/test.csv')
```

Step #2

```
# - Convert categorical data into factor
# - XGBoost only accepts numerical data

train$credit <- as.factor(train$credit)
test$credit <- as.factor(test$credit)
```

XGBoost - Cross Validation

XGBoost

Step #3

```
# - Use Cross Validation with the Train data
# - Partation the data

split_data <- sample(2, nrow(train), replace = T, prob = c(0.8, 0.2))

trainCV <- train[split_data==1,]

testCV <- train[split_data==2,]
```

Step #4 [One Hot Encoding - Train / Test]

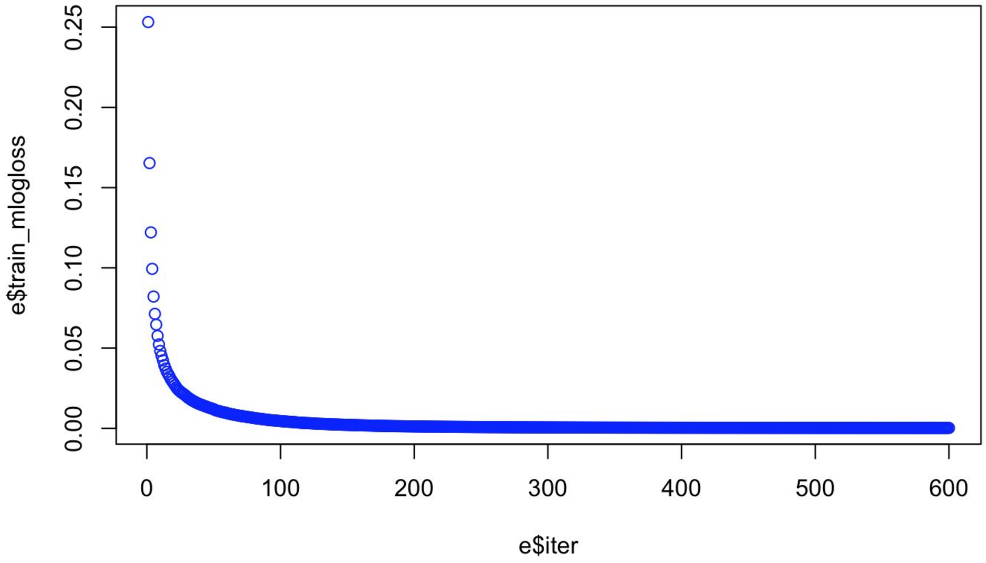
```
# - One Hot Encoding

trainm <- sparse.model.matrix(trainCV$fraud ~
                              credit+duration+total+scans+voidedScans+attemptsWoScan+modifiedQuantities-1,
                              data = trainCV)
train_label <- trainCV[, "fraud"]
train_matrix <- xgb.DMatrix(data = as.matrix(trainm), label = train_label)
```

XGBoost - Cross Validation

Step #5

```
# - Number of class
# - Parameters
nc <- length(unique(train_label))
```

```
# - Best parameter for XGBoost
# - score: 0.99221 [Private Leaderboard]
xgb <- xgboost(data = train_matrix,
    label = train_label,
    eta = 1,
    max_depth = 3,
    nround=600,
    subsample = 1,
    colsample_bytree = 1,
    eval_metric = "mlogloss",
    objective = "multi:softprob",
    num_class = nc,
    nthread = 5)


# Put the evaluation_log into e to plot the numeric data
e <- data.frame(xgb$evaluation_log)
```



XGBoost - Prediction / Confusion Matrix

XGBoost

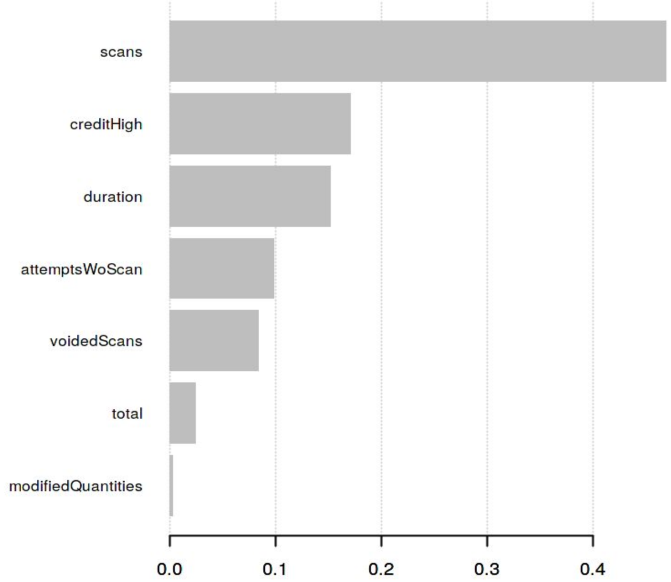
Step #6 - Final Step

```
# - Prediction & confusion matrix - test data

p <- predict(xgb, newdata = test_matrix)

pred <- matrix(p, nrow = nc, ncol = length(p)/nc) %>%
         t() %>%
         data.frame() %>%
         mutate(label = test_label,
                max_prob = max.col(., "last")-1)

table(Prediction = pred$max_prob, Actual = pred$label)
```

```
          Actual
Prediction    0    1
         0 2848   10
         1   11  478
```

0.993725724529429

Feature Importance



XGBoost - Parameter Tuning

XGBoost

- With the information from the Model Validation
- Tested on the entire test dataset
- The best parameter tuning was relevant on the following:

```
# - Best parameter for XGBoost
# - score: 0.99221 [Private Leaderboard]
xgb <- xgboost(data = train_matrix,
    label = train_label,
    eta = 1,
    max_depth = 3,
    nround=600,
    subsample = 1,
    colsample_bytree = 1,
    eval_metric = "mlogloss",
    objective = "multi:softprob",
    num_class = nc,
    nthread = 5)


# Put the evaluation_log into e to plot the numeric data
e <- data.frame(xgb$evaluation_log)
```

XGBoost - Parameter Tuning

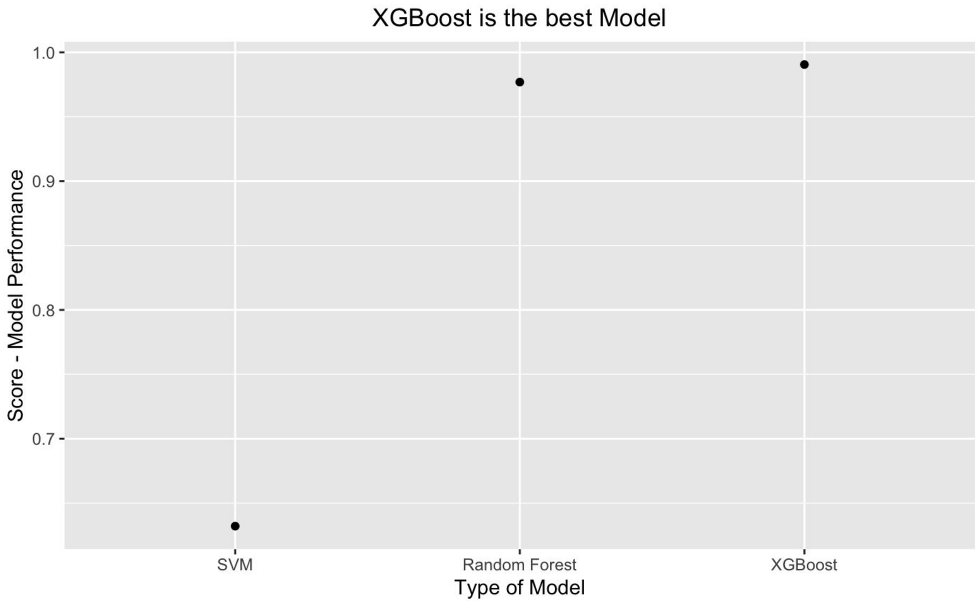
XGBoost

- Focus parameters

Tree Booster

- **eta [default=0.3]**
  - Controls the learning rate, the rate at which our model learns the pattern
- **nrounds[default=100]**
  - Number of iteration - for classification, it's similar to the number of tree to grow
- **max_depth[default=6]**
  - Controls the depth of the tree, larger depth, more complex model, high overfitting
- **subsample[default=1]**
  - Controls number of samples observation to the tree
- **colsample_bytree[default=1]**
  - Controls the number of features variables supplied to the tree

Model Result



XGBoost Public Leaderboard: **0.99083**

Thank you for your time, I look forward to any questions you have, and hearing other presentations.