Credit Card Transaction Fraud Classifier

Mu Niu

2023-12-10

Abstract

In this project, I utilized a credit card transaction dataset from Kaggle to develop a fraud detection classifier. The Exploratory Data Analysis revealed an imbalanced distribution in the fraud response variable, with no missing values. To address this imbalance, I implemented SMOTE, enhancing the model's ability to classify the minority class effectively. Our primary model was XGBoost, complemented by an SVM trained on normalized data for comparative analysis. The XGBoost model outperformed the SVM in AUC, and also offered advantages in interpretability, computational efficiency, and robustness. The conclusion favored XGBoost as the superior model for this scenario. Further evaluation using the confusion matrix highlighted that adjusting the XGBoost's probability-to-class label threshold could further refine the model, particularly by reducing the false positive rate.

Introduction

(1)Problem

- Problem Formulation:

In the rapidly evolving landscape of digital payments, the challenge of detecting fraud is escalating, particularly with trillions of card transactions processed daily. This project aims to tackle this challenge by focusing on fraud classification using a dataset with 7 predictor variables. I will primarily utilize XGBoost for this task. Additionally, to assess and validate our approach, I plan to train a Support Vector Machine as a secondary model and compare the performance of both models using the Area Under the Curve metric.

- Statistical Learning Algorithms Discussion:

The Main Machine Learning Algorithm we are going to apply is XGBoost, which is a sophisticated ensemble machine learning technique based on decision trees. Operating within a gradient boosting framework, XGBoost constructs a series of weak learners sequentially. Each model in the series aims to rectify the errors of its predecessors, leading to a robust and accurate composite model.

On the other hand, we will train a Support Vector Machine for comparative analysis. The SVM is a powerful supervised learning algorithm designed to find an optimal hyperplane in an N-dimensional space(where N represents the number of features). This hyperplane effectively categorizes data points into distinct classes, with a focus on maximizing the margin between the data points and the hyperplane, thereby enhancing classification accuracy. By comparing the performance of XGBoost and SVM, particularly through the lens of AUC, I aim to find the most effective model for fraud detection in our dataset.

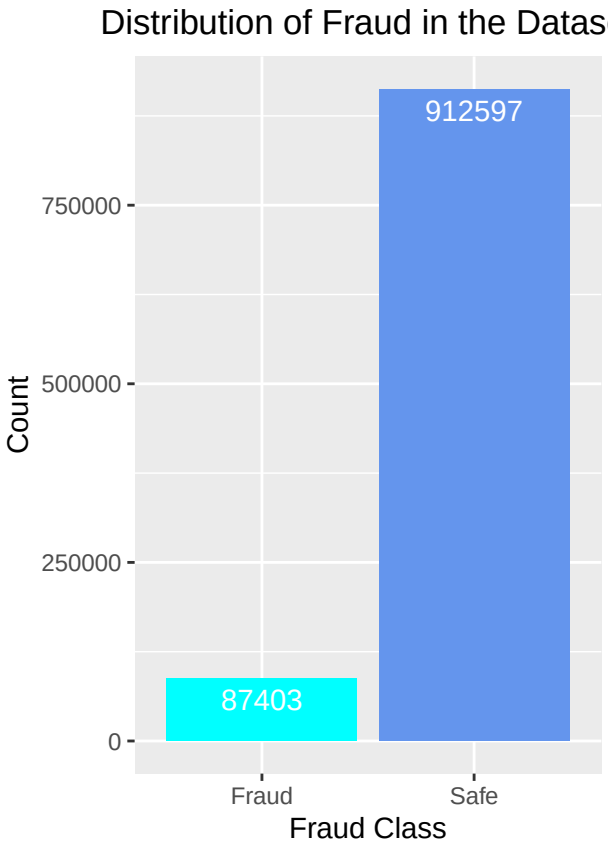
(2)Data Set

- Source(link): Credit Card Fraud

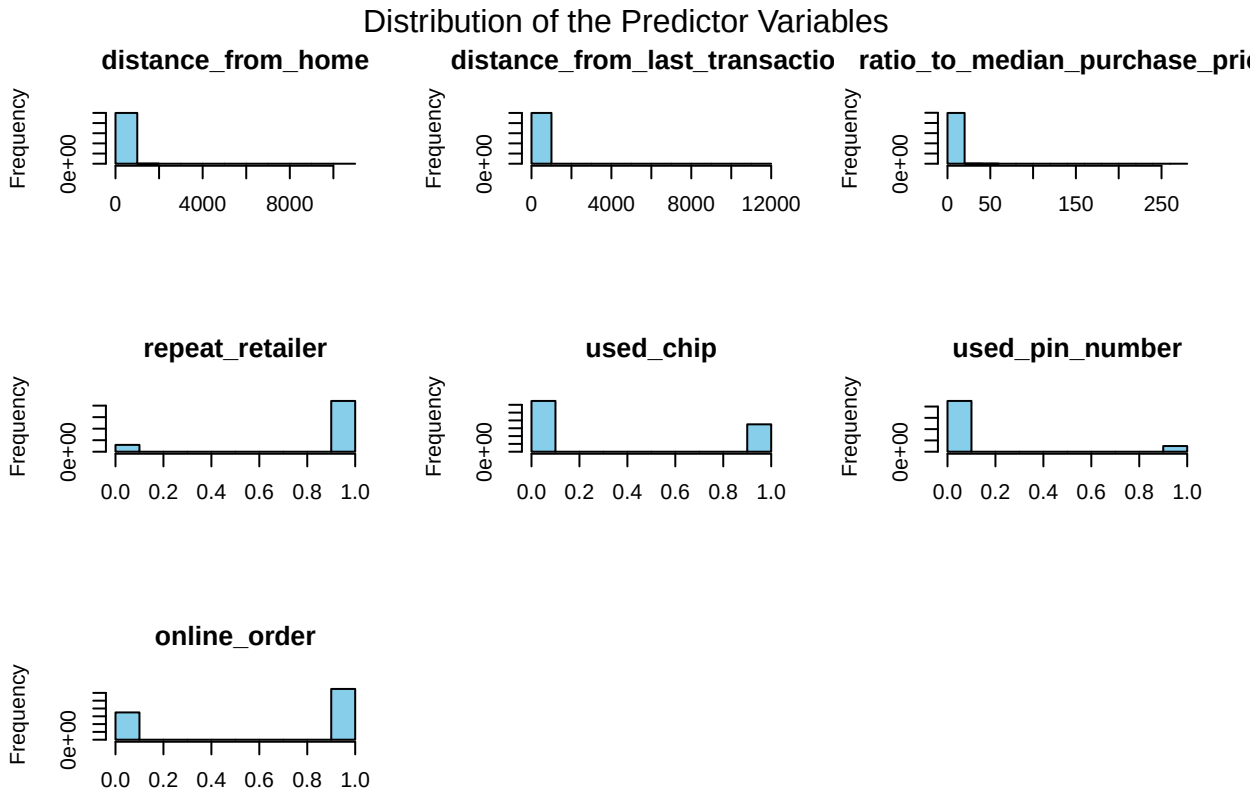
- Feature Explanation:

Variable Name	Feature Explanation	Type
distance_from_home	The distance from home where the transaction happened	Continuous
distance_from_last_transaction	The distance from last transaction happened	Continuous
ratio_to_median_purchase_price	Ratio of purchased price transaction to median purchase price	Continuous
repeat_retailer	Is the transaction happened from same retailer	Discrete
used_chip	Is the transaction through credit card chip	Discrete
used_pin_number	Is the transaction happened by using PIN number	Discrete
online_order	Is the transaction an online order	Discrete
fraud	Is the transaction fraudulent	Discrete

- Exploratory Data Analysis and Visualizations



	Fraud	Safe
.	87403	912597
	8.74%	91.26%



.	Observation	Variable
	1000000	8

```
# Check missing values
data %>% is.na() %>% sum()
```

```
## [1] 0
```

- Data Description

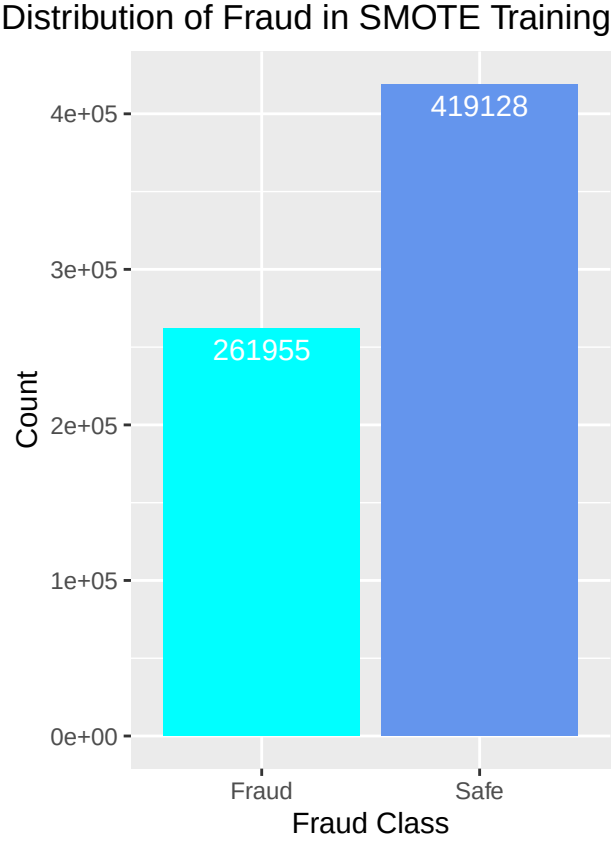
The "Credit Card Fraud" dataset, sourced from Kaggle and published by Dhanush Narayanan R, comprises 1,000,000 observations and 8 variables. It includes continuous variables like 'distance_from_home' and 'distance_from_last_transaction', as well as categorical variables such as 'repeat_retailer', 'used_chip', and 'fraud', which are represented numerically(1 means Yes/0 means No). Our Exploratory Data Analysis(EDA) revealed no missing values, but highlighted an imbalanced response variable. To address this and the varying scales of continuous variables, we will apply over-sampling techniques like SMOTE and normalization before training our models, ensuring robustness, especially for scale-sensitive algorithms like SVM.

Data Preparation

The dataset was divided into training(60%), validation(20%), and testing(20%) subsets, with stratification on the 'fraud' variable to maintain representation of the original dataset across all subsets. This validation set is particularly crucial for our analysis using the **xgb.train()** function in R, which has a **watchlist**

parameter containing both training and validation set. This parameter allows us to monitor log-loss on both training and validation sets during training, aiding in the prevention of overfitting. In the data description section, we noted the varying scales of continuous variables. Given our intention to use the SVM algorithm, which is sensitive to scale differences, we decided to normalize each dataset separately to enhance model performance.

However, our Exploratory Data Analysis revealed an imbalance in the 'fraud' variable, potentially affecting model performance on the minority class. To address this, we will apply SMOTE(Synthetic Minority Over-sampling Technique) to the training set. SMOTE will generate additional observations for the minority class, alleviating the imbalance issue. The downside of the SMOTE is that it may create instances in noisy areas, which can lead to the creation of observations that are not representative of the minority class and may negatively impact classification performance. Hence, we aim for a ratio of approximately 2:1 between the majority and minority classes, avoiding a perfect 1:1 ratio that might introduce too much synthetic data and potentially harm the classification performance. This approach ensures a more balanced dataset after performing SMOTE, which could improve model performance on the minority class.



Fraud	Safe
261955	419128
38.46%	61.54%

Models Training and Outcomes

Main Model: XGBoost

- XGBoost Model Training and Parameter Tunning

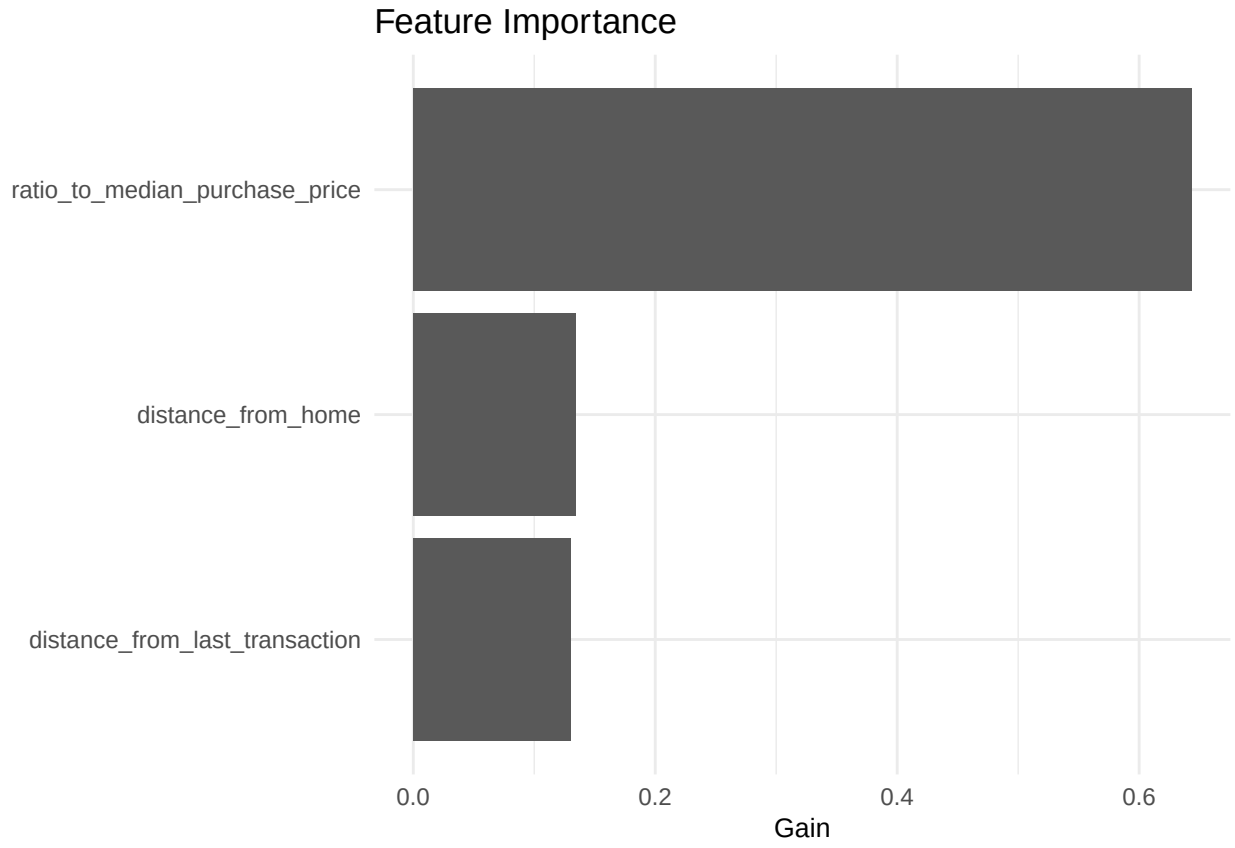
In our model training process, we utilized the watchlist feature, incorporating both training and validation datasets. This approach enables continuous monitoring of the model's performance on both sets, serving as a safeguard to prevent overfitting. Additionally, we implemented the early_stopping_rounds parameter, set

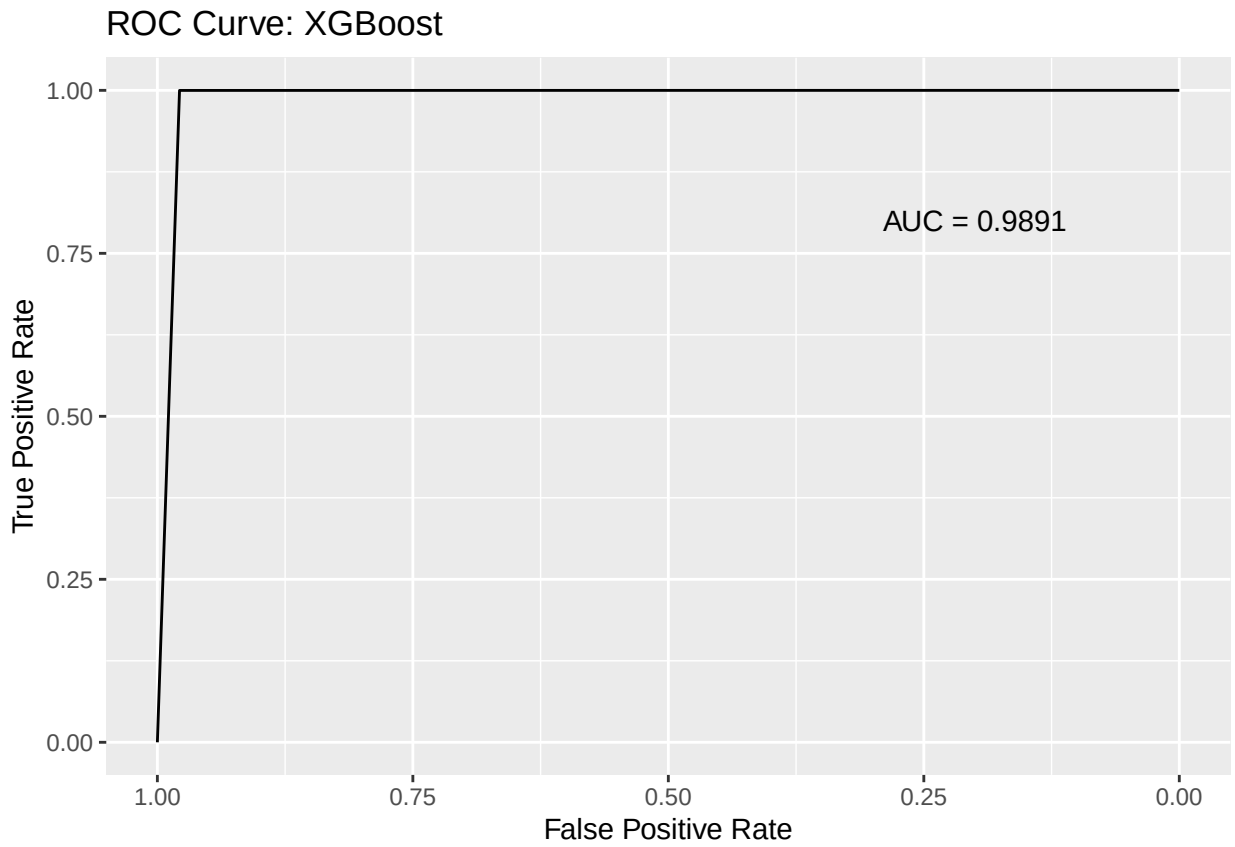
to halt training if there's no improvement in the model's validation performance for 50 consecutive rounds. This strategy is effective in determining the optimal number of boosting rounds to avoid overfitting while maximizing performance. In our case, the model identified 353 as the best number of boosting rounds, indicating that further training beyond this point led to overfitting. This early stopping mechanism is a critical component in fine-tuning our model, ensuring that it achieves robust performance without overfitting to the training data.

- Final XGBoost Model Result and Presentation

Our final XGBoost model, fine-tuned with the optimal number of boosting rounds, demonstrated feature importance insights. As a tree-based method, it highlighted the top three influential features in fraud detection. The most critical among these is the ratio of the transaction purchase price to the median purchase price. This finding suggests that transactions with amounts significantly deviating from the median are more likely to be fraudulent. Additionally, the distances from home and from the last transaction location to the current transaction's location are 2 nearly equally important features. Transactions occurring at locations far from the usual transaction points(home or previous transaction location) are also more likely to be identified as fraudulent.

Evaluating the model's performance on the testing dataset using the AUC-ROC plot and the confusion matrix, we observed an impressive AUC of 0.9891. This high score indicates the model's excellent capability in classifying the testing data and its strong generalization to new, unseen data. Building on these insights, our next step involves training a Support Vector Machine(SVM) and comparing its performance to that of the XGBoost model, to further validate the effectiveness of our chosen approach in fraud detection.





Confusion Matrix

	Actual	
Predicted	0	1
0	178462	2
1	3954	17582

Comparison Model: Support Vector Machine

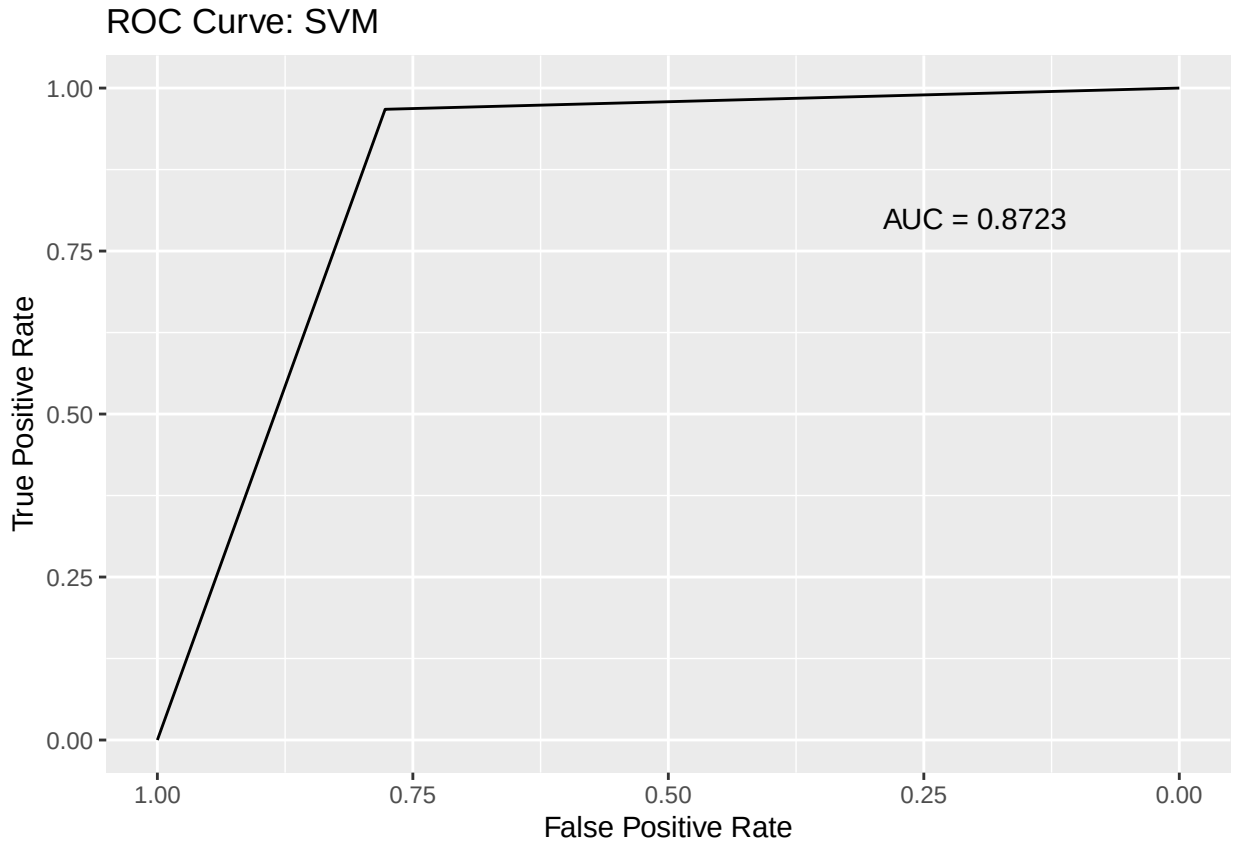
- SVM Training and Parameter Tunning

In this section, I am focusing on training a Support Vector Machine(SVM) to classify fraud and measuring its performance through AUC. Given SVM's sensitivity to data scaling, I am using normalized data to ensure uniform contribution of each feature. I've chosen a radial kernel(RBF) for the SVM since it's a good fit for our dataset's non-linear relationship between predictors and response variable. In training our SVM, I utilized a randomly sampled portion of the normalized training data due to the extensive size of the dataset with 681,083 observations. This strategy is efficient as SVMs primarily rely on support vectors, which are a subset of data, for determining the decision boundary. Such sampling not only captures essential model-building information effectively but also significantly lowers the demands for computational power, which allows me to train the model on my laptop. Moreover, it accelerates hyperparameter tuning and model validation, allowing more efficient experimentations.

To optimize SVM performance, I am utilizing the tune() function for 10-fold cross-validation, aimed at finding the best cost and gamma values. To be more specific on these parameters, the cost is a regularization parameter in SVM. It controls the trade-off between achieving a low training error and a low testing error,

so a small value of cost could potentially underfitting the training data, while a large value of cost might cause overfitting. The gamma parameter defines how far the influence of a single training example reaches. That is, the decision boundary will be influenced more by the support vectors with a large gamma value, leading to a more complex, decision boundary, which might be prone to overfitting. However, with a small gamma value, the decision boundary will be smoother. This can be good for generalization but might also lead to underfitting. The cross-validation process not only identifies these optimal parameters but also yields a model fine-tuned with them, as shown in the table below.

	cost	gamma
16	1000	10



Conclusions and Future Directions for Improvement

Reviewing the ROC-AUC plots for both the XGBoost and SVM models, it's evident that XGBoost outperforms SVM in this dataset. XGBoost's superiority extends beyond the accuracy. As a tree-based method, it demonstrates robustness, showing less sensitivity to data scaling compared to SVM. Furthermore, XGBoost excels in interpretability, particularly in identifying key features influencing the model. Additionally, XGBoost is computationally more efficient, operating faster than SVM, which is a substantial advantage in practical applications. These factors collectively affirm that XGBoost has better performance over SVM for this specific dataset.

However, there's room for improvement in the XGBoost model, especially regarding its false positive rate. The confusion matrix reveals that the model occasionally misclassifies safe transactions as fraudulent. This over-caution can disrupt customer experience in real-world applications, such as shopping, which potentially

harm customer trust. A promising area for improvement is to adjust the threshold of mapping predicted probability to the class label. The currently threshold is set at 50%, where transactions with a fraud probability over this are marked as fraud, and a lower threshold could reduce false positives. Gaining more domain knowledge through research in related fields is a valuable approach for further improvement. With this enhanced understanding, we can more effectively adjust the threshold used to classify transactions as fraudulent. Additionally, we can empirically test different thresholds on the current model, comparing performances to identify the most effective setting. This practical approach will help us minimize the false positive rate and thereby improve the classification accuracy.

	Actual	
Predicted	0	1
0	178462	2
1	3954	17582

Appendix

```
# load the packages
library(tidymodels)
library(xgboost)
library(tidyverse)
library(pROC)
library(ggplot2)
library(gridExtra)
library(DMwR)
library(caret)
library(e1071)
# load the data
data <- read_csv("card_transdata.csv")
# plot distribution of the response
plot1 <- ggplot(data, aes(x = ifelse(data$fraud == 0, 'Safe', 'Fraud'))) +
  geom_bar(fill = c('cyan','cornflowerblue')) +
  labs(title="Distribution of Fraud in the Dataset",
       x="Fraud Class", y="Count") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_text(aes(label = after_stat(count)), stat = "count",
            vjust = 1.5, colour = "white")
table1 <- table(ifelse(data$fraud == 0, 'Safe', 'Fraud')) %>%
  rbind(c('8.74%','91.26%')) %>% as.data.frame() %>% tableGrob()
grid.arrange(arrangeGrob(plot1, table1, ncol=2))
# plot distribution of the predictor variables
vars <- data %>% select(-fraud)
par(mfrow = c(3, 3))
for (col in colnames(vars)) {
  hist(vars[[col]], main = col, xlab = '', col = "skyblue",
       border = "black", breaks = 10)}
mtext('Distribution of the Predictor Variables',
      side = 3, line = -1.2, outer = TRUE)
# dimension of the data set
c('Observation','Variable') %>% rbind(dim(data)) %>% knitr::kable()
# Check missing values
data %>% is.na() %>% sum()
```

```r
# set seed for reproducibility
set.seed(131)
# split the data
data.split <- data %>%
  initial_split(prop = 0.6, strata = fraud)
test.split <- initial_split(testing(data.split), prop = 0.5, strata = fraud)
data.train <- training(data.split)
data.val <- training(test.split)
data.test <- testing(test.split)
# convert discrete variables to categorical variable(required by SMOTE function)
smote.data <- data.train %>% select(1:3) %>%
  cbind('repeat_retailer' = as.factor(data.train$repeat_retailer)) %>%
  cbind('used_chip' = as.factor(data.train$used_chip)) %>%
  cbind('used_pin_number' = as.factor(data.train$used_pin_number)) %>%
  cbind('online_order' = as.factor(data.train$online_order)) %>%
  cbind('fraud' = as.factor(data.train$fraud))
# compute parameter for a ratio of 1.6:1 between majority class and minority class
perc.over <- round(((0.5 * 547609 - 52391) / 52391) * 100, 0)
# apply SMOTE for imbalanced data
smote <- SMOTE(fraud ~., smote.data, perc.over = perc.over, k =5)
# normalize each data set for SVM
norm.train <- as.data.frame(lapply(smote, function(x) {
  if (is.numeric(x)) (x - min(x)) / (max(x) - min(x))
  else x}))
norm.val <- as.data.frame(lapply(data.val, function(x) {
  if (is.numeric(x)) (x - min(x)) / (max(x) - min(x))
  else x}))
norm.test <- as.data.frame(lapply(data.test, function(x) {
  if (is.numeric(x)) (x - min(x)) / (max(x) - min(x))
  else x}))
# plot distribution of the smote response
plot2 <- ggplot(smote, aes(x = ifelse(smote$fraud == 0, 'Safe', 'Fraud'))) +
geom_bar(fill = c('cyan','cornflowerblue')) +
labs(title="Distribution of Fraud in SMOTE Training Dataset",
x="Fraud Class", y="Count") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_text(aes(label = after_stat(count)), stat = "count", vjust = 1.5,
            colour = "white")
table2 <- table(ifelse(smote$fraud == 0, 'Safe', 'Fraud')) %>%
  rbind(c('38.46%','61.54%')) %>% as.data.frame() %>% tableGrob()
grid.arrange(arrangeGrob(plot2, table2, ncol=2))
# convert categorical variable to numeirc(required by XGBoost function)
xgb.train.data <- smote %>% select(1:3) %>%
  cbind('repeat_retailer' = as.numeric(smote$repeat_retailer) - 1) %>%
  cbind('used_chip' = as.numeric(smote$used_chip) - 1) %>%
  cbind('used_pin_number' = as.numeric(smote$used_pin_number) - 1) %>%
  cbind('online_order' = as.numeric(smote$online_order) - 1) %>%
  cbind('fraud' = as.numeric(smote$fraud) - 1)
# define predictor and response variables in training set
# NOTE: XGBoost only use matrix data
xgb.train.x <- data.matrix(xgb.train.data %>% select(-fraud))
xgb.train.y <- xgb.train.data %>% pull(fraud)
# define predictor and response variables in validation set
```

```r
xgb.val.x <- data.matrix(data.val %>% select(-fraud))
xgb.val.y <- data.val %>% pull(fraud)
# define predictor and response variables in testing set
xgb.test.x <- data.matrix(data.test %>% select(-fraud))
xgb.test.y <- data.test %>% pull(fraud)
# define xgb.DMatirx: a specialized data structure xgboost uses for efficiency
xgb.train <- xgb.DMatrix(data = xgb.train.x, label = xgb.train.y)
xgb.val <- xgb.DMatrix(data = xgb.val.x, label = xgb.val.y)
xgb.test <- xgb.DMatrix(data = xgb.test.x, label = xgb.test.y)
# define watchlist to monitor training process & prevent overfitting
watchlist = list(train = xgb.train, validation = xgb.val)
# define params
params <- list(
  objective = "binary:logistic", # set goal to predict probability of fraud
  eta = 0.3 # learning rate: default 0.3 can prevent overfitting
  )
#fit XGBoost model and display training and testing data at each round
set.seed(131)
model <- xgb.train(params = params,
                   data = xgb.train, # Training data
                   # Size of each individual tree: rule of thumb is 3 to prevent overfitting
                   max.depth = 3,
                   # Track model performance on train/validation
                   watchlist = watchlist,
                   # Number of boosting iterations: more observation, more rounds
                   nrounds = 500,
                   # threshold to stop training: usually 10% of nrounds
                   early_stopping_rounds = 50,
                   # silent: don't show processing result
                   verbose = 0)
set.seed(131)
# Define final model
# The verbose = 0 tells R don't show training and testing error for each round.
final <- xgboost(params = params, data = xgb.train,
                 max.depth = 3, nrounds = 353, verbose = 0)
### Feature Importance
importance <- xgb.importance(feature_names = colnames(xgb.train.x), model = final)
# plot 3 most important features
importance_df <- as.data.frame(importance)[1:3,]
feature.plot <- ggplot(importance_df, aes(x = reorder(Feature, Gain), y = Gain)) +
  geom_col() +
  coord_flip() +
  theme_minimal() +
  theme(axis.title.y = element_blank(),
        axis.title.x = element_text(size = 10)) +
  labs(title = "Feature Importance", x = "Gain")
# Use model to make predictions on test data
pred.y <- predict(final, xgb.test.x)
# Label test data according to the predicted probability
pred.label <- ifelse(pred.y > 0.5, 1, 0)
# Confusion Matrix
confusion.matrix <- table(Predicted = pred.label, Actual = xgb.test.y)
# AUC-ROC
```

```r
roc <- roc(xgb.test.y, pred.label)
auc <- auc(roc)
# Visualization
ggroc(roc) +
  labs(title = "ROC Curve: XGBoost", x = "False Positive Rate",
       y = "True Positive Rate") +
  annotate("text", x = 0.2, y = 0.8, label = paste("AUC =", round(auc, 4)))
# XGBoost Confusion Matrix
confusion.matrix
# scatter plot to determine if linear relationship exists
par(mfrow = c(3, 3))
for (col in colnames(vars)) {
  plot(data$fraud, vars[[col]], main = col, xlab = '', ylab = '',
       col = "skyblue", border = "black") }
# Train SVM
set.seed(131)
svm.train <- norm.train %>% select(1:3) %>%
  cbind('repeat_retailer' = as.numeric(smote$repeat_retailer) - 1) %>%
  cbind('used_chip' = as.numeric(smote$used_chip) - 1) %>%
  cbind('used_pin_number' = as.numeric(smote$used_pin_number) - 1) %>%
  cbind('online_order' = as.numeric(smote$online_order) - 1) %>%
  cbind('fraud' = smote$fraud)
# sample training data due to limited computation power
sample_indices <- sample(1:nrow(svm.train),
                         size = 0.01 * nrow(svm.train))
svm.sample <- svm.train[sample_indices, ]
# find optimal parameters
tune.out <- tune(svm, fraud ~., data = svm.sample,
                 kernel = 'radial', scale = FALSE,
                 ranges = list(cost=c(1,5,100,1000),gamma=c(0.1,1,5,10)))
# show best parameters
summary(tune.out)$"best.parameters" %>% knitr::kable()
# test data set for SVM
svm.test = data.frame(norm.test %>% select(-fraud),
fraud = norm.test %>% pull(fraud) %>% as.factor())
# predict base on test x
svm.pred = predict(tune.out$best.model, svm.test)
# confusion matrix
table(Predicted = svm.pred, Actual = svm.test$fraud)
# ROC-AUC
roc2 <- roc(svm.test$fraud, svm.pred %>% as.numeric())
auc2 <- auc(roc2)
# Visualization
ggroc(roc2) +
  labs(title = "ROC Curve: SVM", x = "False Positive Rate",
       y = "True Positive Rate") +
  annotate("text", x = 0.2, y = 0.8, label = paste("AUC =", round(auc2, 4)))
```