Predicting Fraudulent Transactions

Salih Tuzen

Motivation:

In 2018, unauthorized financial fraud losses across payment cards and remote banking totaled £844.8 million in the United Kingdom. Whereas banks and card companies prevented £1.66 billion in unauthorized fraud in 2018. Objective here is to predict fraudulent transactions based on some transaction related features.

Dataset Description

- Data source: Kaggle
  - Credit Card Transaction over two days in September 2013
  - 284,807 observations
  - All numeric variables except outcome
  - Only class variable categorical
  - Number of features: 31

Name	Description
Time	Time of transaction
V1-V28	Masked data
Amount	Transaction amount
Class	Indicates fraudulent or not

Objective & Methodology

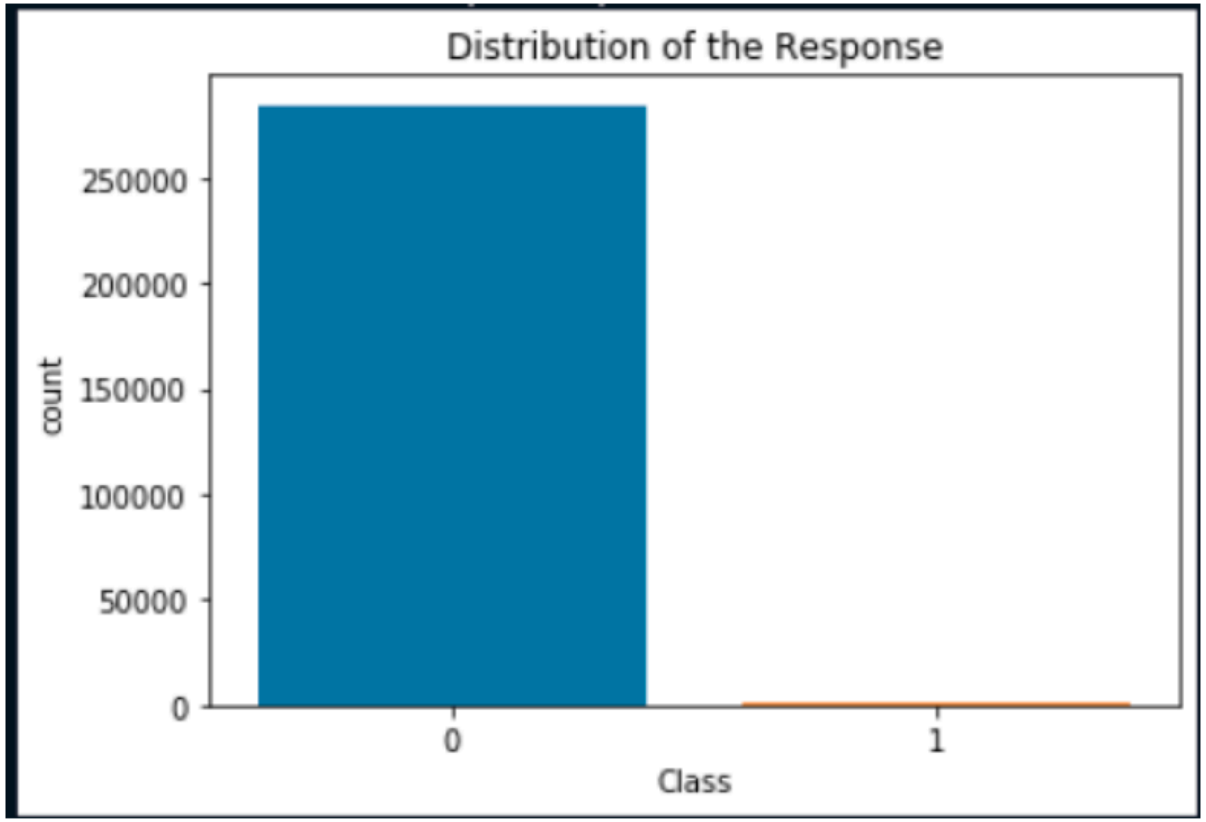
- **Supervised Learning:**
  - Label is Class.
- **Models:**
  - Support Vector Machine
  - Neural Network
  - AdaBoost
  - Random Forest
- **3-Fold cross-validation**
- **Methods for Imbalanced Data**
  - SMOTE
  - RUS
  - Weight Adjustment
- **Scoring metrics used:**
  - ROC area
  - Sensitivity
- **Hyperparameter Optimization**
  - GridsearchCV
- **Standardization:**
  - Robust Standardization
  - Regular Standardization
- **Pipeline Process**

Data Exploration

Distribution of Response variable:

- Data is split into 70% training and 30% testing sets.

- Out of 284,807 observations, only 492 of them are fraudulent. That is only 0.17% of the all observations.



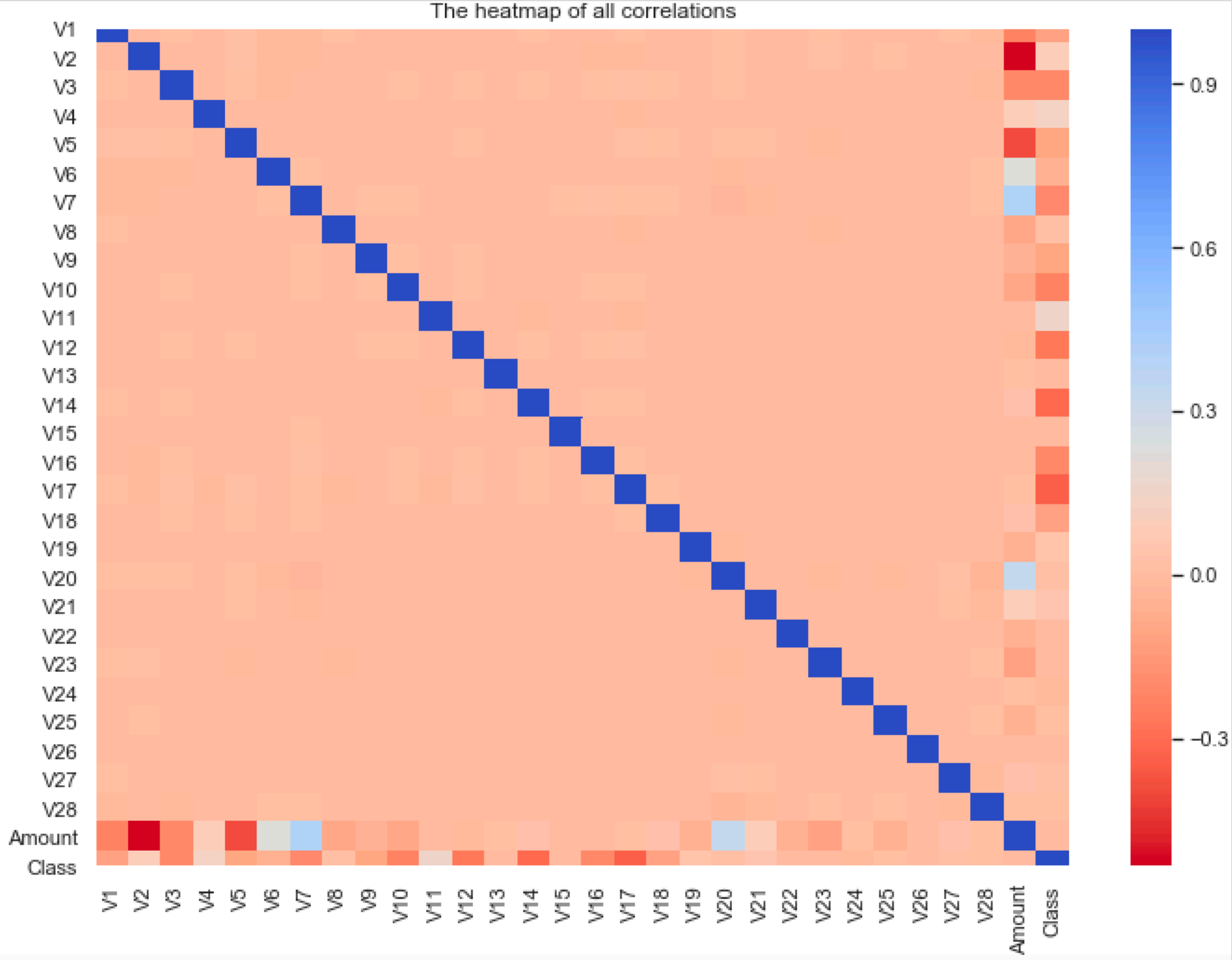
GRAPH 1. DISTRIBUTION OF THE OUTCOME VARIABLE

Data Exploration

Correlation Matrix Heatmap

- We can’t interpret the correlations within predictors because of masked variables.

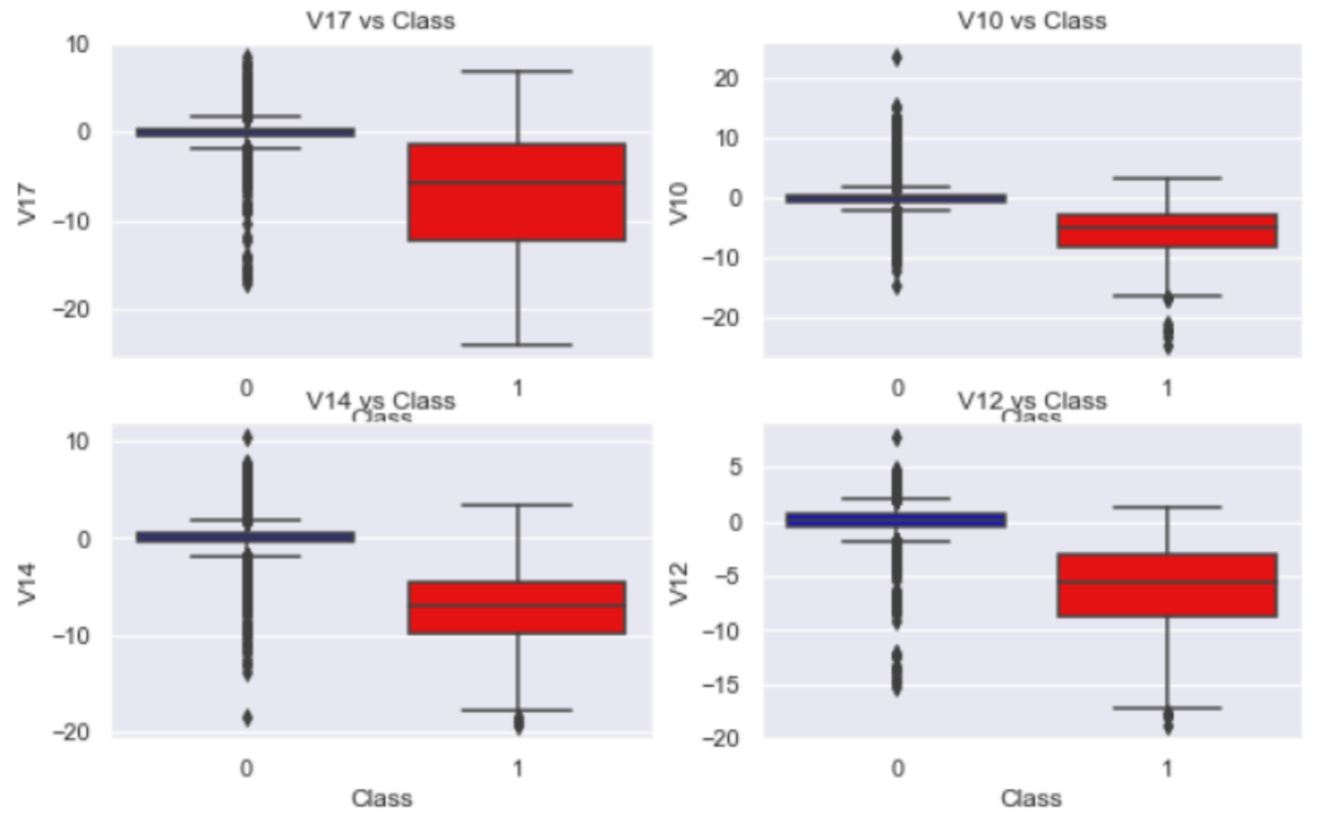
- However, some variables seem to be significant for the classification models.



Data Exploration

Box plot of some V variables

- The graph 5 shows that V10,V17,V14 and V12 can be significant in terms of predicting class variable.



GRAPH 5. DISTRIBUTIONS OF SOME V VARIABLES AGAINST THE CLASS VARIABLE

Hyperparameter Optimization

- Using Grid Search method, I found the optimum parameters in the given hyper-parameter space based on the best cross validation score.

- Support Vector Classifiers:

Kernel: Linear kernel, **Regularization parameter**=1

- AdaBoost optimum Parameters:

Learning rate =1 and **n_estimators** =100.

- Random Forest optimum parameters:
- **criterion**="entropy", **n_estimators**=150, **min_samples_split**=2, **min_samples_leaf**=1

Modeling

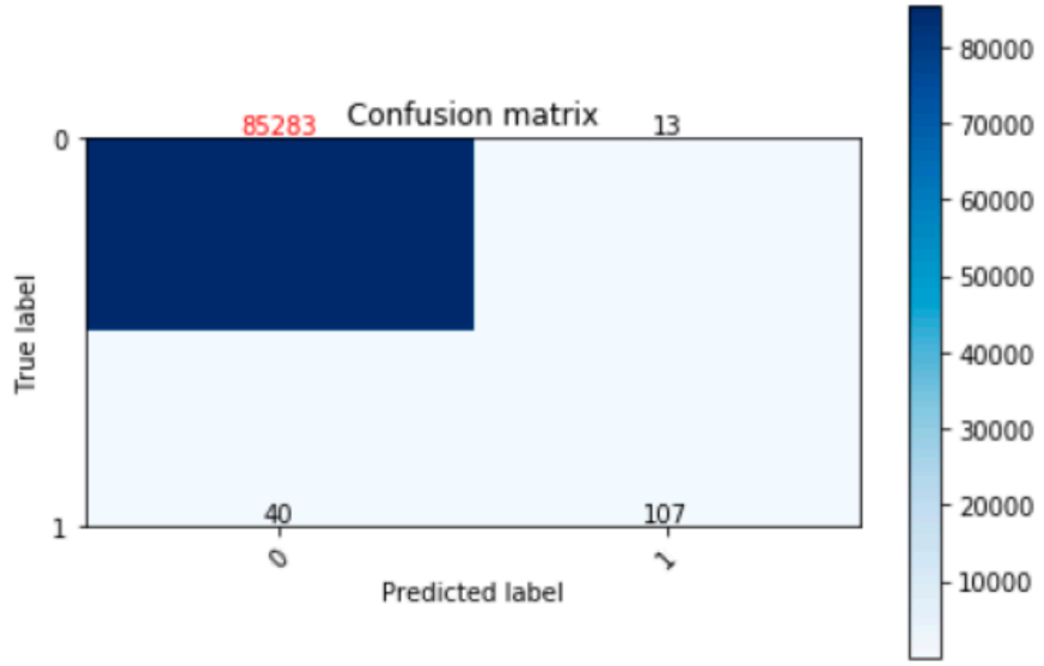
Model 1: Neural Network

- Two hidden layers with 20 nodes in each hidden layer.
- Binary cross-entropy (log loss) function for the loss function.

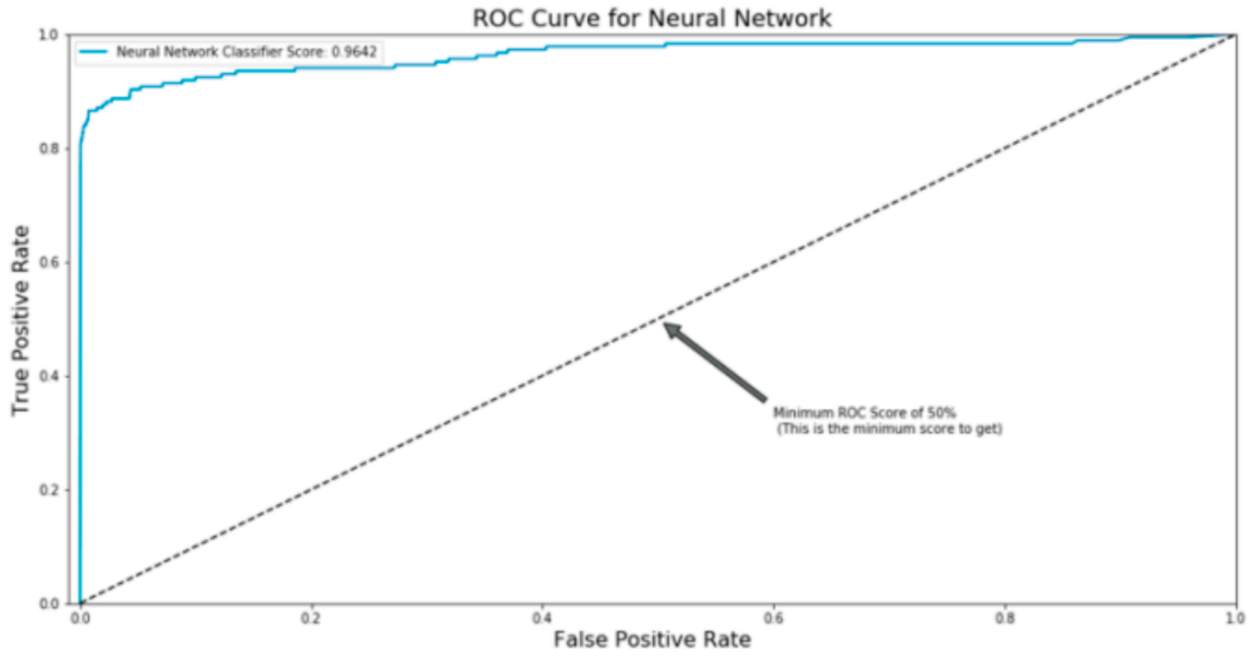
$$Error = \sum_{i=1}^{n} -(p_i \log q_i + (1 - p_i) \log(1 - q_i))$$

Modeling

Model 1: Neural Network



GRAPH 5. CONFUSION MATRIX FOR NEURAL NEWTWORK.



GRAPH 6. ROC AREA FOR NEURAL NETWORK

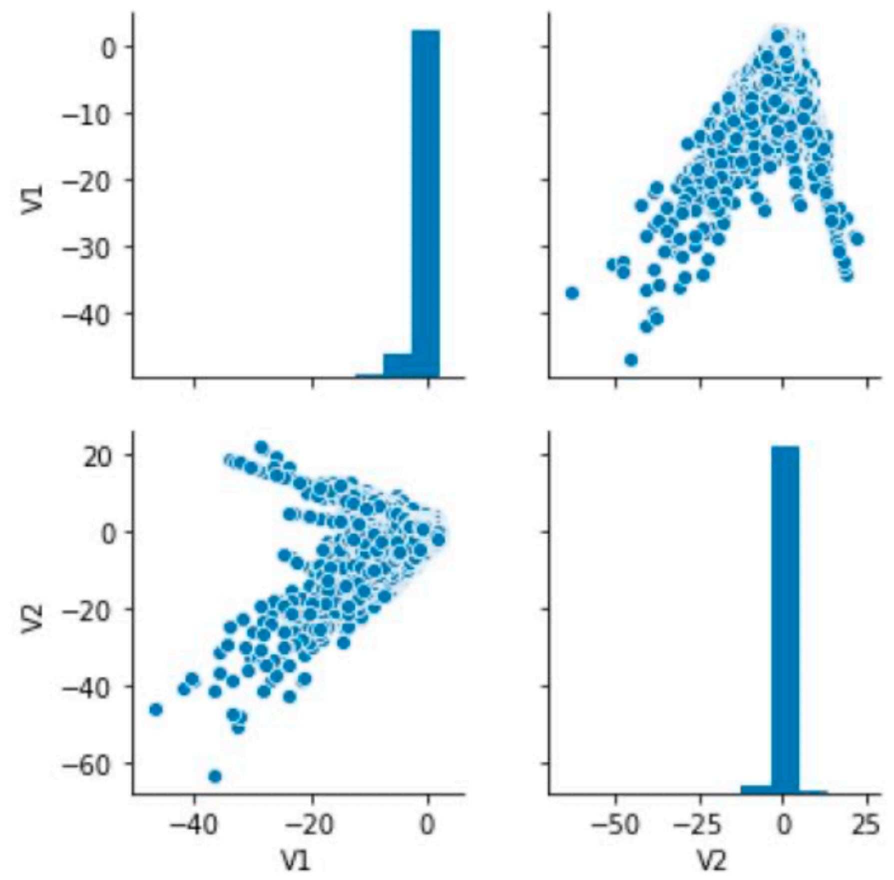
ROC: 0.96 Sensitivity: 40/147=0.73 False Positive:13 Cases

Modeling

Model 2: Support Vector Machine

- Instead of traditional standardization, Robust Standardization is applied.
- Traditional standardization method of subtracting the mean and dividing by the standard deviation may probably impacted by the outliers in the dataset.
- In the robust standardization, the median is subtracted from the sample then it is divided by the IQR.

SVM	Sensitivity	AUC
Split 1	0.77	0.93
Split 2	0.74	0.92
Split 3	0.79	0.95
Average	0.77	0.93



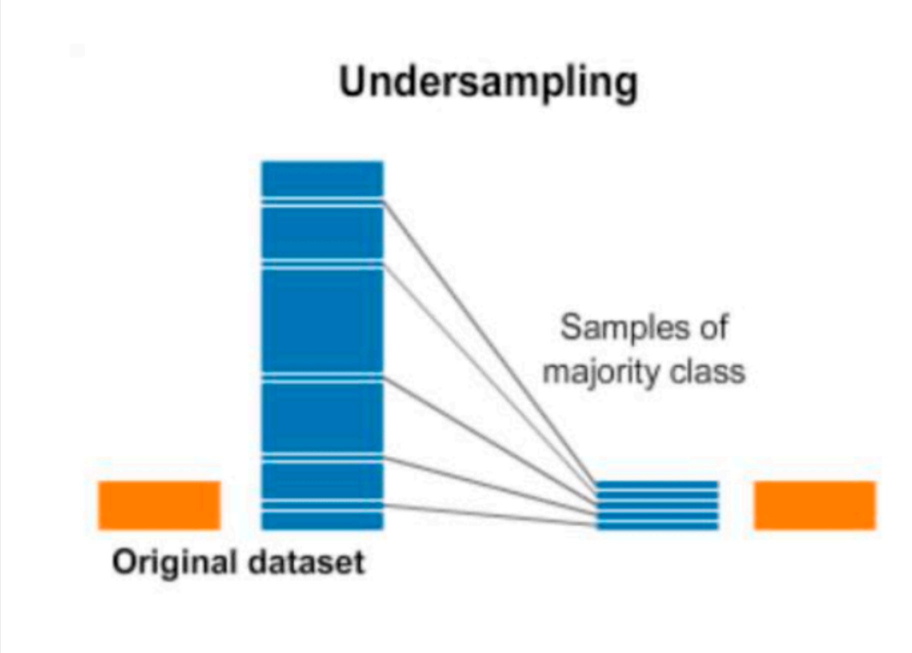
GRAPH 3. DISTRIBUTIONS OF SOME V VARIABLES.

Modeling

Model 3: Support Vector Machine with Random Under Sampling Method

- The RUS is the simplest case of under-sampling, where a simple random sample of the size of the minority class was generated from the majority class.

SVM/RUS	Sensitivity	AUC
Split 1	0.85	0.96
Split 2	0.95	0.98
Split 3	0.93	0.98
Average	0.91	0.97



Modeling

Model 4: Support Vector Machine with SMOTE:

- SMOTE is an over-sampling method where new observations of the minority class are synthesized based on the existing minority observations and parameters defined.

SVM/SMOTE	Sensitivity	AUC
Split 1	0.82	0.94
Split 2	0.95	0.98
Split 3	0.89	0.97
Average	0.89	0.96

Modeling

Model 5: Support Vector Machine with Balanced Weight:

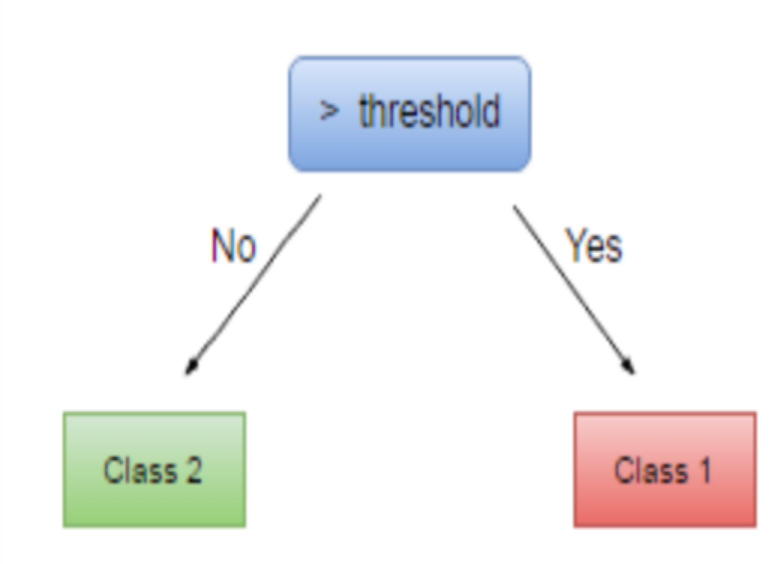
- The classification models traditionally assume all misclassification errors have the same costs.
- Assigned a higher cost to FN misclassification so that a fraudulent transaction wouldn't be easily classified as normal, and therefore model's bias is reduced.
- Adjusted weights inversely proportional to class frequencies in the input data.

SVM/Adj. Weight	Sensitivity	AUC
Split 1	0.77	0.93
Split 2	0.74	0.92
Split 3	0.79	0.95
Average	0.77	0.93

Modeling

Model 6: AdaBoost:

- AdaBoost is a sequential ensemble classifier that combines stumps to form a strong classifier.
- In each iteration of AdaBoost, the sample weights and the stumps' weights are adjusted so that it learns from the mistakes in earlier steps.
- Grid Search method found the optimum parameters as Learning rate=1 and n_estimators =100.



AdaBoost	Sensitivity	AUC
Split 1	0.75	0.94
Split 2	0.79	0.98
Split 3	0.75	0.97
Average	0.76	0.96

Modeling

Model 7: Random Forest:

- Random Forest is a bagging classifier where at each split, a random subset of variables are used. Unlike AdaBoost, all trees in random forest have the same weight.
- Grid Search method found the optimum parameters as

criterion="entropy", n_estimators=150, min_samples_split=2, min_samples_leaf=1

Random Forest	Sensitivity	AUC
Split 1	0.72	0.91
Split 2	0.80	0.97
Split 3	0.76	0.95
Average	0.76	0.94

Model Comparisons

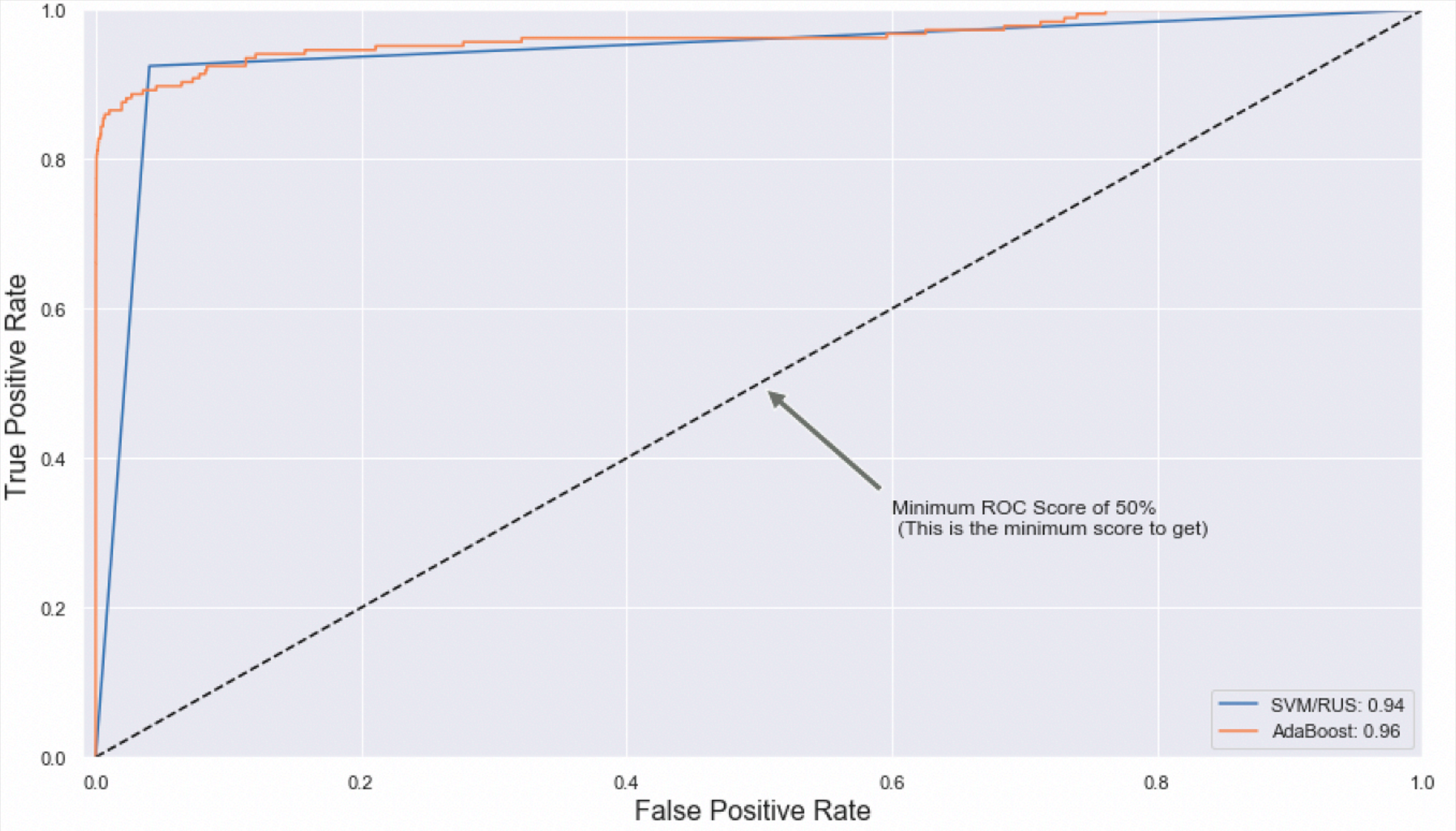
Average Sensitivity and AUC scores

- I explored 4 different models: SVM, AdaBoost, Random Forest and Neural Network.
- This project explored 3 different approaches to handle the imbalanced data. They are RUS, SMOTE, Adjusted Weights.
- When both Sensitivity and AUC considered, the SVM with RUS is the best performing model. The SVM with SMOTE is close second.

Models	Average Sensitivity	Average AUC	Computation Time (min.)
Neural Network	0.72	0.88	0.76
SVM	0.77	0.93	16.36
SVM/RUS	0.91	0.97	0.05
SVM/SMOTE	0.89	0.96	64.21
SVM/ Adj. Weight	0.77	0.93	12.49
Random Forest	0.76	0.94	1.50
AdaBoost	0.77	0.96	1.05

Visual Model Comparison

- The best performing model from SVM family and ensemble family compared on the same ROC plot.



Final Remarks

- The computation time was a big restriction and impacted the decisions I made. Due to computational intensity, only a small range of hypermeters and values were explored.

- Python pipeline process was extremely helpful with code organization and efficiency. Pipeline function simply allows sticking multiple processes into a single fitting.

- Among all complicated models with abundance of hyperparameters, the best performing model was the SVM with Random Under Sampling Method.