Credit Card Fraud Detection Project Report

July 22, 2024

1 Introduction

This project focuses on developing a model to detect credit card fraud using a highly imbalanced dataset from Kaggle, which includes 284,807 credit card transactions with only 492 identified as fraudulent. The extreme imbalance, **where fraudulent transactions make up only 0.17% of the total**, poses a significant challenge for traditional machine learning techniques.

To address the class imbalance and improve the detection of fraudulent transactions, **this project implements three approaches: a voting classifier, a neural network with focal loss, and a synthetic minority oversampling technique (SMOTE).**

2 Data Analysis

It **contains only numerical input variables** which are the result of a PCA transformation. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

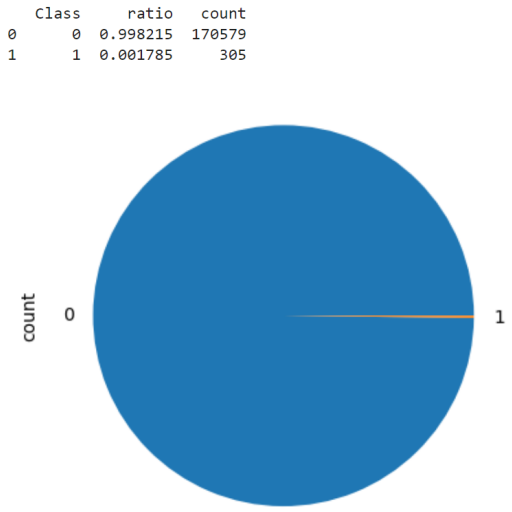

Figure 1: check class imbalance

3 Data Correlation

As you can see in Figure 2, There are different correlations or patterns between the feature and each class.

Fraud Transactions (Left Heatmap)

Strong Correlations:

- There are noticeable blocks of strong positive correlations among several variables.
- Variables such as V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, and V12 show significant correlations among themselves. This is evident from the bright yellow blocks in these regions.
- Variables like V16, V17, and V18 also show strong correlations with each other.

Non-Fraud Transactions (Right Heatmap)

Weak Correlations:

- The non-fraud transactions heatmap shows mostly weak correlations among variables. The majority of the correlations are close to zero, indicated by the dominance of green and blue colors.

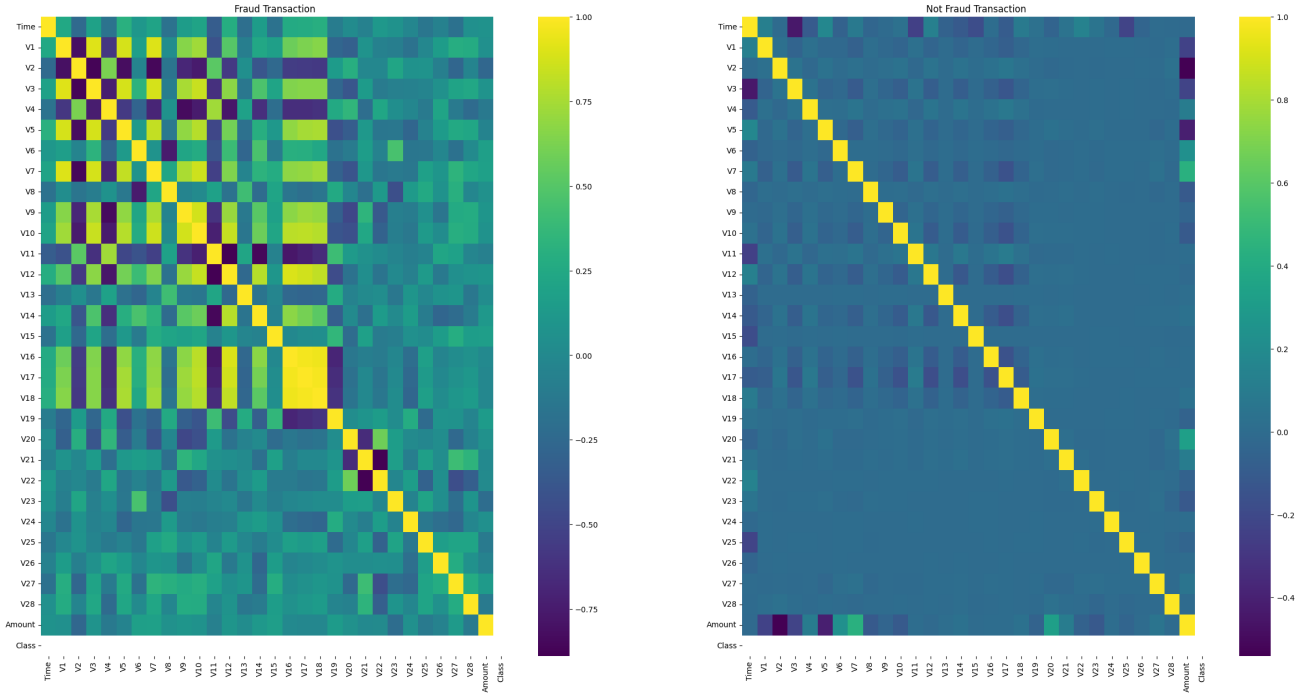

Figure 2: Correlation matrix separate class

4 Data visualization using PCA

As we can see from Figure 3 and Figure 4, the dataset is highly imbalanced, with Class 0 dominating. There's no clear separation between most of the data points and a few outliers. **However, we cannot conclude any definitive information because this is just a projection in low dimension.**

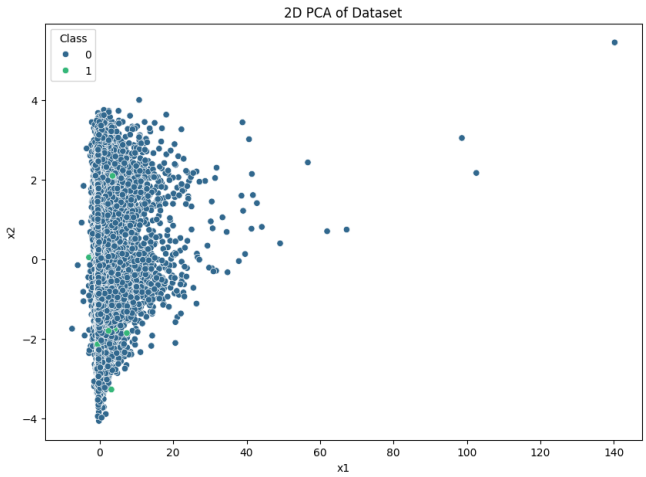

Figure 3: 2D PCA of Dataset

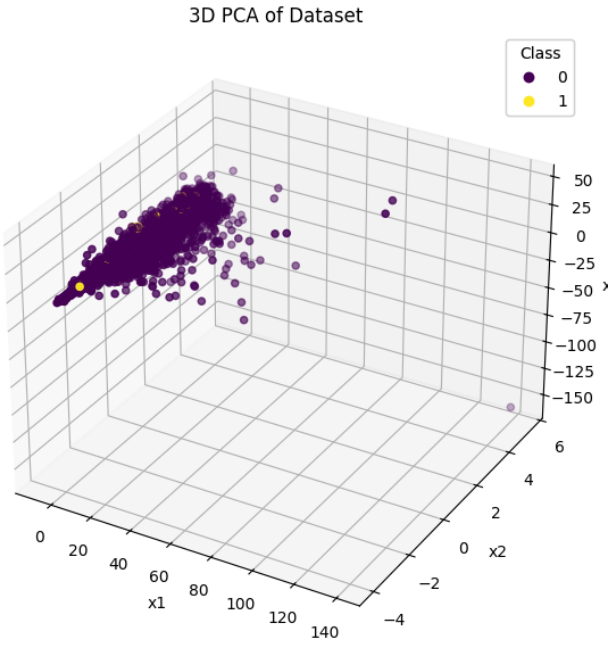

Figure 4: 3D PCA of Dataset

5 Analysis of Random Forest Feature Importances

The feature importance plot from our Random Forest model provides valuable insights into which features are most influential in predicting credit card fraud. Here's a detailed analysis:

5.1 Most Important Features

- V17 is by far the most important feature, with a relative importance of approximately 0.2 (20%).
- V14 is the second most important, with importance around 0.17 (17%).
- V12 is third, with importance close to 0.14 (14%).

5.2 Least Important Features

Many features (V21, V1, V5, V6, V2, V19, V26, Amount, V13, V8, Time, V15, V20, V25, V22, V28, V24, V23, V27) have very low importance, barely visible on the chart.

5.3 Key Observations

- The 'Amount' feature has surprisingly low importance, suggesting that the transaction amount alone is not a strong predictor of fraud in this model.
- 'Time' also has very low importance, indicating that the time of transaction may not be a crucial factor in detecting fraud.
- The features are named V1, V2, etc., which suggests they might be derived features or principal components, possibly from a dimensionality reduction technique applied before the Random Forest model.

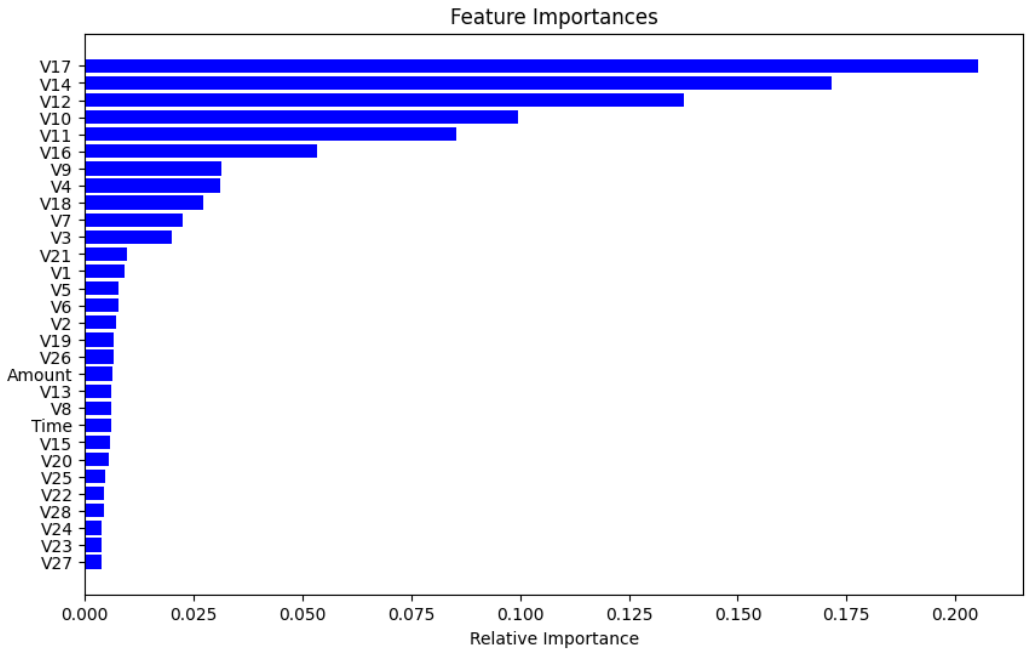

Figure 5: Random Forest Feature Importances

6 Modeling

6.1 Voting Classifier

I built a voting classifier using three different models: Random Forest, Logistic Regression, and Neural Network.

Random Forest I performed a Randomized Search with Stratified K-Fold cross-validation. Additionally, I tried an Extra Trees Classifier. Key observations include:

- Random Forests easily overfit the training data.
- Regularization techniques proved very important.
- Increasing the minimum number of samples per leaf helped prevent overfitting.
- Normal Random forest outperforms then Extra Trees Classifier.

Logistic Regression and KNN These classifiers initially achieved poor performance. Notable improvements came from:

- Using RobustScaler, which helped more than other scaling techniques.
- Setting the `class_weight` parameter, which was crucial for performance.

Multi-Layer Perceptrons (MLP) - Neural networks The MLP achieved the most balance between training and validation performance. Key points from the randomized search with Stratified K-Fold:

- Stochastic Gradient Descent (SGD) optimizer performed better than Adam.
- The `tanh` activation function worked better than ReLU.

6.2 Results

Figure 6: Model comparison on validation data

Note:
The optimal threshold is calculated by finding the highest F1-score and its threshold in training data only.

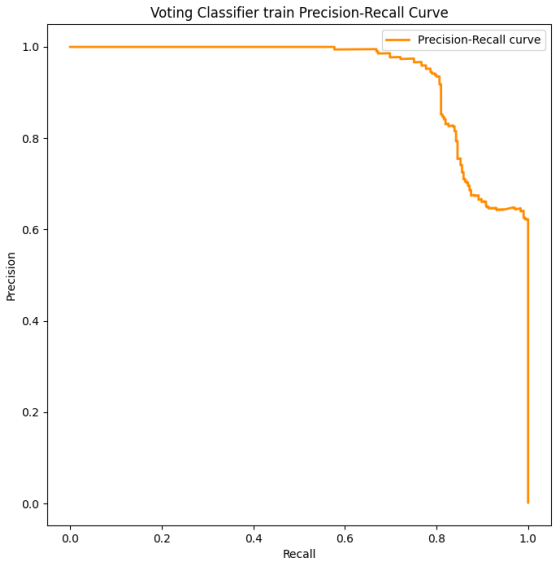

Figure 7: Voting Classifier PR-AUC

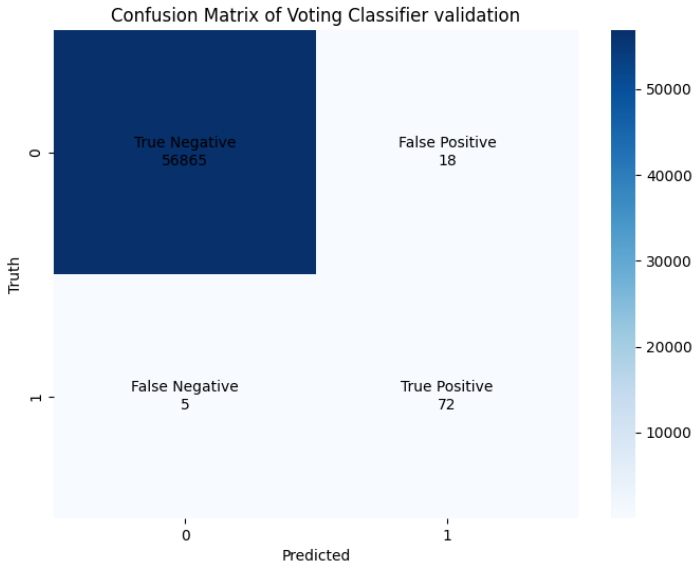

Figure 8: Voting classifier validation confusion matrix

Observation:
I also observed using K-Nearest Neighbors (KNN) with the `ball_tree` algorithm achieved a performance close to that of Random Forest and the neural network but required significantly high computational power.

6.3 Neural network with focal loss

Focal Loss is a specialized loss function designed to address the class imbalance problem commonly encountered in tasks like object detection. It was introduced in the paper "Focal Loss for Dense Object Detection." The main idea is to focus more on hard-to-classify examples while reducing the loss contribution from easy-to-classify examples. This is achieved through two parameters α (alpha) and γ (gamma).

$$\mathrm{FL}(p_\mathrm{t}) = -\alpha_\mathrm{t}(1 - p_\mathrm{t})^\gamma \log(p_\mathrm{t}).$$

Figure 9: Focal Loss formula

Listing 1: Focal Loss Implementation in PyTorch

```
class FocalLoss(nn.Module):
    def __init__(self, gamma=2, alpha=0.25):
        super(FocalLoss, self).__init__()
        self.gamma = gamma
        self.alpha = alpha

    def forward(self, pred_logits, target):
        BCELoss = F.binary_cross_entropy_with_logits(pred_logits, target, red
        prob = pred_logits.sigmoid()
        alpha_t = torch.where(target == 1, self.alpha, (1 - self.alpha))
        pt =  torch.where(target == 1, prob, 1 - prob)
        loss = alpha_t * ((1 - pt) ** self.gamma) * BCELoss
        return loss.sum()
```

Observation and Notes:

- I experimented with different combinations of α (0.80–0.99, increment of 0.05) and γ (0–4, increment of 1).
- The best result was achieved with $\alpha = 0.75$ and $\gamma = 2$.
- α and γ sometimes cause instability during training. Using Batch Normalization mitigated this effect, and switching from Adam to SGD also helped.
- A high γ (5–7) results in a very noisy loss curve.
- A high α gives very high recall for positive class with less precision and vice versa.

6.4 Results

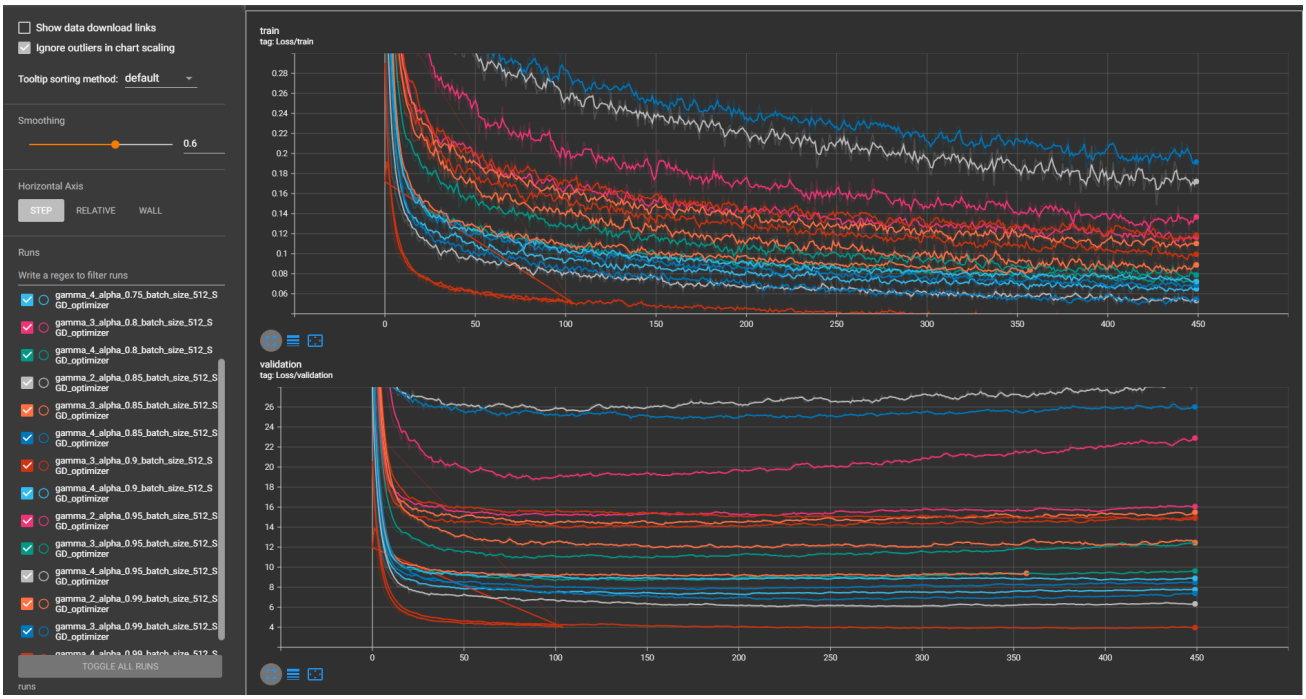

Figure 10: Different alpha and gamma effect Loss curve

```
Checkpoint loaded from epoch 400
FraudDetectionNN train Classification Report
              precision    recall  f1-score   support

           0    0.99977   0.99974   0.99975    170579
           1    0.85531   0.87213   0.86364       305

    accuracy                        0.99951    170884
   macro avg    0.92754   0.93593   0.93170    170884
weighted avg    0.99951   0.99951   0.99951    170884

FraudDetectionNN valdtion Classification Report
              precision    recall  f1-score   support

           0    0.99975   0.99975   0.99975     56870
           1    0.84444   0.84444   0.84444        90

    accuracy                        0.99951     56960
   macro avg    0.92210   0.92210   0.92210     56960
weighted avg    0.99951   0.99951   0.99951     56960
```

Figure 11: Focal loss classification report

7 Smote and under-sampling technique

SMOTE (Synthetic Minority Over-sampling Technique) is an oversampling method used to generate synthetic samples for the minority class. Despite experimenting with SMOTE, random over-sampling, and under-sampling techniques, the results on the validation data were poor.

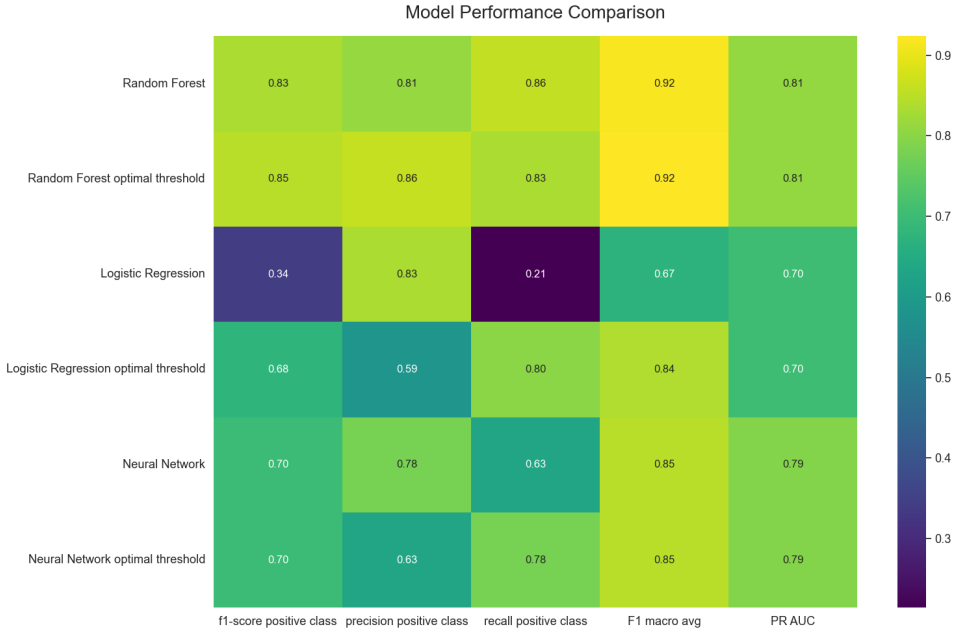

Figure 12: Smote (0.05-ratio)

Figure 13: Random-Under-Sampler (0.05-ratio) result