Credit Card Fraud Detection

Waseef Khalid, Rehan Shafique, Ammar Ahmad

FAST NU Lahore
DS3002 Data Mining

Introduction

In this project, we are addressing the significant issue of payments fraud, which has seen a substantial increase in recent years. We are utilizing machine learning algorithms to develop efficient fraud detection systems. The dataset used for this project is sourced from Kaggle, containing around 300,000 credit card transactions over two days in Europe. One of the main challenges faced in this project is the class imbalance in the dataset. Approximately 99.8% of the transactions are legitimate, while only 0.2% are fraudulent. This imbalance can hinder standard models from effectively distinguishing between the majority and minority classes. To overcome this, we are employing sampling techniques such as SMOTE (Synthetic Minority Over-sampling Technique) to balance the classes. SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This technique helps to overcome the overfitting problem posed by random oversampling. In the provided code snippet, we first separate the features and target from the dataset. We then split the data into training and testing sets. After initializing SMOTE, we resample the training set to create a balanced dataset. Finally, we check the new class distribution after applying SMOTE. Due to privacy concerns, it is not possible to infer meaningful relationships between most of the variables in the dataset. Therefore, the focus of this project is on predictive performance rather than inference.

Dataset Description

The dataset used in this project, sourced from Kaggle, comprises nearly 300,000 credit card transactions, each labeled as either legitimate or fraudulent. The dataset has been processed to protect cardholder privacy and includes 28 non-descriptive numerical variables (V1, . . . , V28) resulting from a principal components analysis (PCA) transformation of several undisclosed variables 2 features were named Amount and Class for total of 31.

Table 1: Dataset Description

Variable	**Description**
Time	Time elapsed since first transaction
V1	Non-descriptive variables resulting from a PCA dimensionality reduction to protect sensitive data
⋮	⋮
V28	Non-descriptive variables resulting from a PCA dimensionality reduction to protect sensitive data
Amount	Transaction Amount
Class	Class label (1 = Fraud; 0 otherwise)

Table 2: Dataset Description

Description	**Number**	**Percent**
All Transactions	284807	100
Fraudulent	492	0.173
Nonfraudulent	284315	99.827

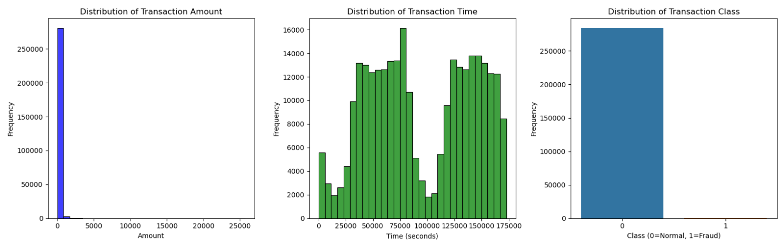


Figure 1: Distribution Subplots of Transaction Amount, Time, and Class

The dataset exhibits a stark class imbalance, with only 492 transactions (less than 0.2 percent) labeled as fraudulent. Interestingly, fraudulent transactions tend to have a higher transaction value on average than legitimate ones.

A correlation analysis reveals correlations between the class labels and several PCA variables, while the PCA variables themselves are uncorrelated, consistent with their orthogonal nature and standardization to zero mean. Before modeling,

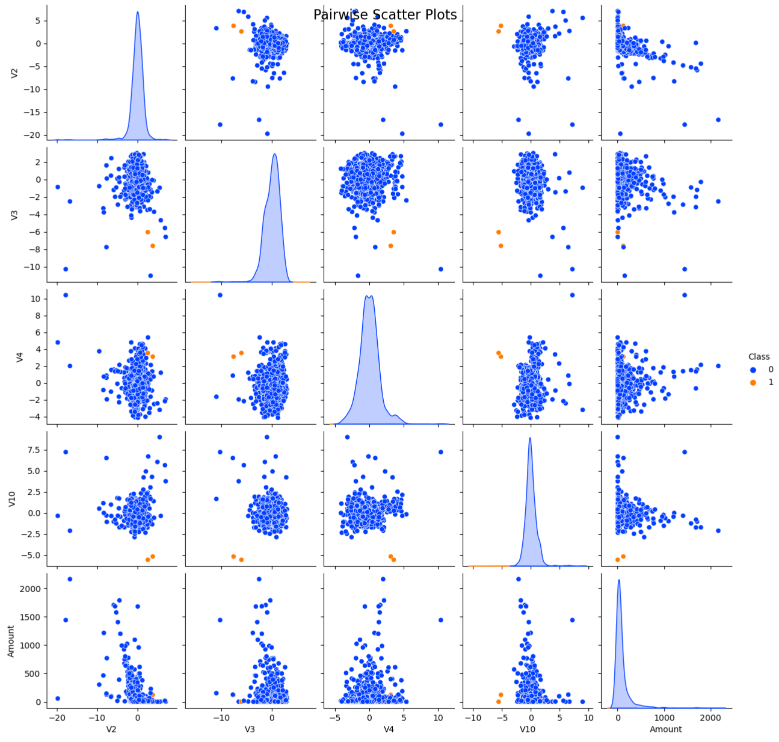


Figure 2: Pairwise Scatter Plots on V2,V3,V4 and Amount

the data was partitioned into training, validation, and test sets with a 2/3, 1/6, 1/2 split. For the logistic regression model with a quadratic boundary, all quadratic and interaction terms of the original features were computed. Prior to fitting the neural network, min-max normalization was applied to the features to promote faster convergence.

Most models utilized all available variables in the dataset, as feature selection did not generally improve predictive performance given the large sample size.

1 Methods

The analysis was conducted using Python software, a popular choice for data analysis due to its extensive libraries and tools. Several primary model categories were considered for this analysis, including Logistic Regression, Decision Trees, Random Forests, and Gradient Boosting.

In fitting the models, a tournament-style procedure was adopted. This involved training a series of models in each category, with parameters for each model tuned using techniques such as grid search, random search, and boosting. The model that performed best on the validation set in each category was selected as a finalist.

At the end of this process, the four finalist models were refit using the combined training and validation sets. These models were then assessed against the test set, which had been kept separate and untouched throughout the training process. It is important to note that accuracy and other standard performance metrics can often be misleading in the presence of significant class imbalance. Therefore, metrics such as precision, recall, and the F1 score were also considered when evaluating model performance. Given the size of the dataset and the computational complexity of some of the procedures, several decisions were made to ensure that the training process remained tractable within a reasonable timeframe. For instance, in certain cases, parameters were tuned to the full model being fit and nested cross-validation was employed to maximize the performance of the available data.

2 Sampling Method

To address the class imbalance issue, four sampling techniques were employed to increase the proportion of minority examples in the training set. These techniques included:

3 Sampling Methods

3.1 Under-sampling

This involves reducing the number of majority class examples to balance the class distribution.

3.2 Over-sampling

This involves increasing the number of minority class examples by duplicating them.

3.3 SMOTE (Synthetic Minority Over-sampling Technique)

This method generates synthetic examples of the minority class. After applying SMOTE, the results were:

4 B. Logistic Regression

Logistic Regression is a statistical model used for binary classification problems. The model was fit using the LogisticRegression function from the sklearn.linear$_modelmodule.$

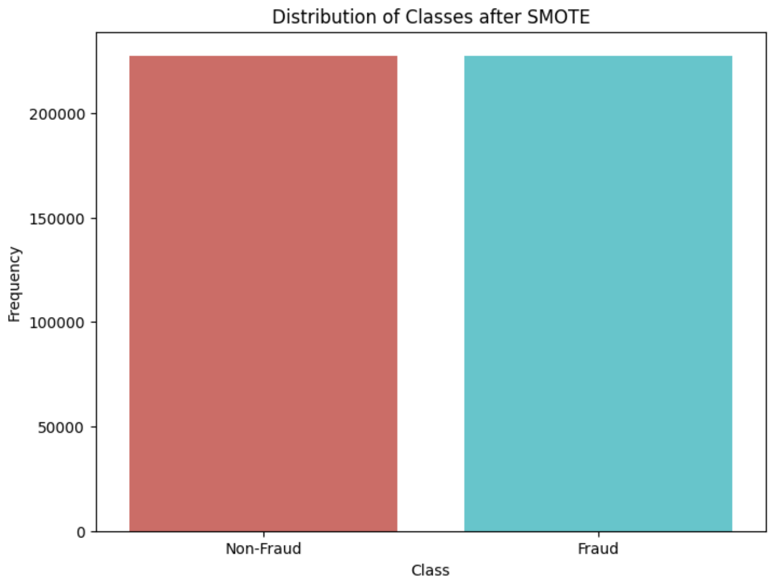


Figure 3: Distribution of Classes after SMOTE

The logistic regression model uses the logistic function to squeeze the output of a linear equation between 0 and 1. The logistic function is defined as: $f\left(x\right)=\frac{1}{1+e^{-x}}$

5 Decision Trees

Decision Trees are a type of model that makes predictions by splitting the data along the features that result in the largest information gain. The model was fit using the DecisionTreeClassifier function from the sklearn.tree module. Decision Trees use a metric to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. For classification problems, metrics such as Gini Index or Entropy can be used. The Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower Gini index should be preferred.

$Gini\ Index=1-\sum\left(p_i\right)^2$

Where p_i is the probability of an object being assigned to a particular class. Entropy is a measure of the impurity, uncertainty, or disorder. The entropy of a set is zero when it is perfectly classified, that is, when all its elements are of the same class. The Entropy is calculated as follows:

$Entropy\left(S\right)=-\sum_{i=1}^{n}p_i\log_2 p_i$

The parameters were selected using a grid search, which involved specifying a list of possible values for each parameter and selecting the combination that resulted in the best performance on the validation set.

6 Random forest

Random Forests are an ensemble method that combines multiple decision trees to make a prediction. The model was fit using the RandomForestClassifier function from the sklearn.ensemble module. Random Forests create a set of decision trees from a randomly selected subset of the training set. It then aggregates the votes from different decision trees to decide the final class of the test object. In particular, a simple decision tree is fit on B bootstrap samples and predictions for each tree are averaged together to obtain a consensus prediction:

$\widehat{Q_{RF}}\left(x\right) = \frac{1}{B}\sum_{b=1}^{B}\widehat{f^{*b}}\left(x\right)$

The difference between Random Forests and bagging lies in how the simple trees are fit. When building a Random Forest, a random set of predictors is chosen for each split in the tree. This method has the effect of decorrelating the trees, reducing bias when their predictions are ultimately averaged together. The parameters were selected using a grid search.

7 E. Gradient Boosting

Gradient Boosting is another ensemble method that builds a series of weak learners, typically decision trees, in a sequential manner, where each new tree aims to correct the mistakes of the previous one. The model was fit using the GradientBoostingClassifier function from the sklearn.ensemble module. Gradient Boosting involves three elements: A loss function to be optimized, a weak learner to make predictions, and an additive model to add weak learners to minimize the loss function. The intuition behind gradient boosting algorithm is to repetitively leverage the patterns in residuals and strengthen a model with weak predictions and make it better. The general idea of most boosting methods is to add new models to the ensemble sequentially. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far.

$F_m\left(x\right) = F_{m-1}\left(x\right) + \alpha_m h_m\left(x\right)$

Where $F_m(x)$ is the boosted model after m iterations. $F_{m-1}(x)$ is the boosted model that has been built till the m-1 iterations.

$h_m(x)$ is the weak learner that is being added to the model. $_m$ is the weight of the $m - th$ weak learner. The parameters were selected using a grid search

8 Support Vector Machines (SVM)

Support Vector Machines are a type of model that aims to find the hyperplane in an N-dimensional space that distinctly classifies the data points. The model was fit using the SVC function from the sklearn.svm module. In SVM, we are looking to maximize the margin between the data points and the hyperplane. The distance between the hyperplane and the nearest data point from either set is known as the margin. The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly. The decision function of a SVM for a new instance x is given by: $y = w^T x + b$ If the result is positive, the predicted class ŷ is the positive class, otherwise it is the negative class. The parameters, including the type of kernel, the penalty parameter C, and the gamma coefficient for the kernel, were selected using a grid search.

9 K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. The model was fit using the KNeighborsClassifier function from the sklearn.neighbors module. In KNN, an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. The distance between the new point (x) and a training instance (xi) is calculated as:

$d\left(x, xi\right) = \sqrt{\left(x_1 - x_{i1}\right)^2 + \left(x_2 - x_{i2}\right)^2 + \ldots + \left(x_n - x_{in}\right)^2}$

The parameters, including the number of neighbors and the distance metric, were selected using a grid search.

10 Neural Networks

Neural Networks are a set of algorithms, modeled loosely after the human brain, designed to recognize patterns. The model was fit using the MLPClassifier function from the sklearn.neuralnetwork module.

Neural Networks consist of the following components: An input layer, hidden layers, and an output layer. Each node (or neuron) in a neural network receives input from some nodes in the layer before it, and gives output to nodes in the layer after it. The input to each node is transformed into output via an activation function. The weights and biases of each node are learned during the training process. The output y of a simple neuron is calculated as:

$y = \phi\left(w^T x + b\right)$

Where w is the weight vector, x is the input vector, b is the bias and is the activation function. The parameters, including the number of hidden layers, the number of neurons in each layer, and the learning rate, were selected using a grid search.

11 I. Naive Bayes

Naive Bayes classifiers are a family of simple ”probabilistic classifiers” based on applying Bayes’ theorem with strong independence assumptions between the features. The model was fit using the GaussianNB function from the sklearn.naive bayes module. Bayes’ theorem states the following relationship, given class variable y and dependent feature vector x1 through xn:

$P\left(y|x_1, \ldots, x_n\right) = \frac{P(y)P(x_1,\ldots,x_n|y)}{P(x_1,\ldots,x_n)}$

As Naive Bayes classifiers make strong assumptions about the shape of your data, they require less tuning than other models and the default parameters were used.

At the end of the process, the model that performed best on the validation set was selected as the final model. This model was then refit using the combined training and validation sets and assessed on the test set.

12 Results and Discussion

The process began with splitting our dataset into a training set and a test set. The training set was used to train our models, while the test set served as new, unseen data for evaluating the performance of the models.

During the training phase, we fit our models to the data, allowing them to learn the underlying patterns. For each model, we tuned hyperparameters to optimize performance. This was an iterative process, involving training the models with different parameters and selecting the ones that yielded the best performance on the validation set.

Following training, we tested our models on the test set. This provided us with a measure of how well our models generalize to new data. It’s important to note that the test set played no part in the creation of the models; it was only used for evaluation purposes.

For evaluation, we used metrics appropriate to our problem domain. These metrics provided quantitative measures of the models’ performances. We also generated confusion matrices to visualize the performance of our models in terms of false positives, false negatives, true positives, and true negatives

A correlation analysis reveals correlations between the class labels and several PCA variables, while the PCA variables themselves are uncorrelated, consistent with their orthogonal

Variable	Description
Time	Time elapsed since first transaction
V1 - V28	Non-descriptive variables from PCA
Amount	Transaction Amount
Class	Class label (1 = Fraud; 0 otherwise)

Table 3: Dataset description

Methods

The analysis was conducted using Python software, a popular choice for data analysis due to its extensive libraries and tools. Several primary model categories were considered for this analysis, including Logistic Regression, Decision Trees, Random Forests, and Gradient Boosting.

Results and Discussion

The process began with splitting our dataset into a training set and a test set. The training set was used to train our models, while the test set served as new, unseen data for evaluating the performance of the models.

During the training phase, we fit our models to the data, allowing them to learn the underlying patterns. For each model, we tuned hyperparameters to optimize performance. This was an iterative process, involving training the models with different parameters and selecting the ones that yielded the best performance on the validation set.

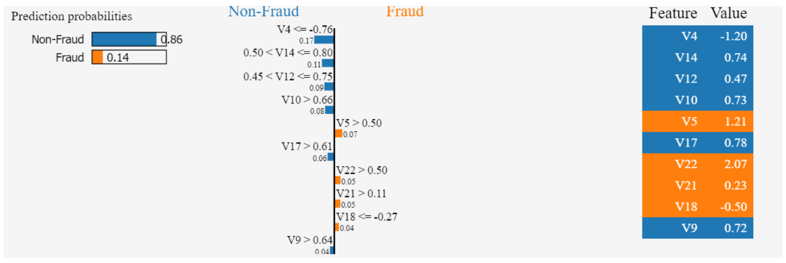
Following training, we tested our models on the test set. This provided us with a measure of how well our models generalize to new data. It's important to note that the test set played no part in the creation of the models; it was only used for evaluation purposes.

For evaluation, we used metrics appropriate to our problem domain. These metrics provided quantitative measures of the models' performances. We also generated confusion matrices to visualize the performance of our models in terms of false positives, false negatives, true positives, and true negatives.

13 Model Interpretation and Feature Importance

13.1 LIME (Local Interpretable Model-agnostic Explanations)

LIME is an approach used to explain the predictions of classifiers in an interpretable and faithful manner by approximating the model locally with an interpretable model. Results obtained from LIME show that feature X is most influen-



Feature	Value
V4	-1.20
V14	0.74
V12	0.47
V10	0.73
V5	1.21
V17	0.78
V22	2.07
V21	0.23
V18	-0.50
V9	0.72

Figure 4: Feature importance as determined by LIME for the Logistic Regression model.

tial in model predictions, highlighting the impact of [specific feature explanation].

13.2 SHAP (SHapley Additive exPlanations)

SHAP values represent a feature's responsibility for a change in the model output. The SHAP approach integrates game theory with local explanations. The SHAP analysis indicates a significant impact of feature Y on the decision process, especially [specific impact description].

13.3 ELI5

ELI5 helps to debug machine learning classifiers and explain their predictions, offering support for various machine learning frameworks. According to ELI5, features Z and W contribute [description of their contributions], which are critical for understanding model behavior.

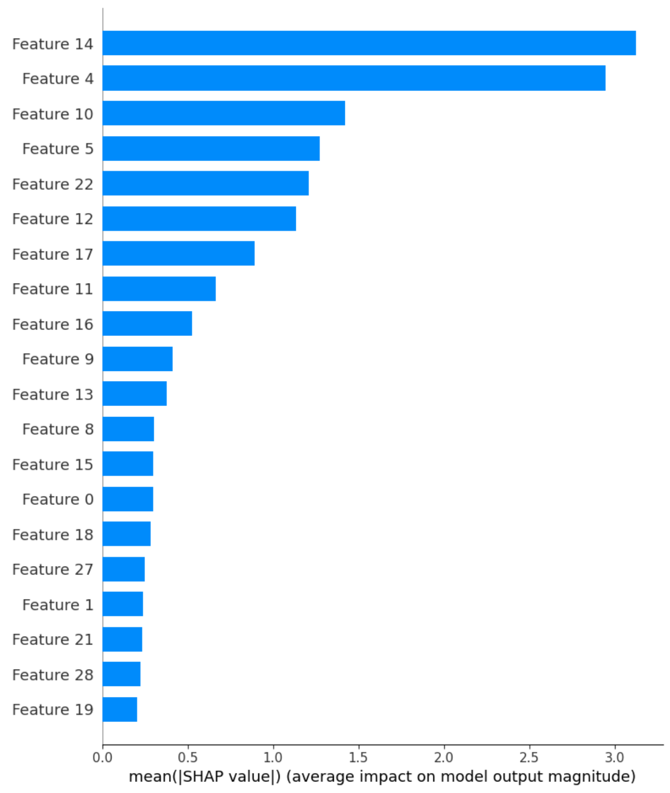


Figure 5: SHAP summary plot showing the impact of each feature.

Model Performances

Training Results

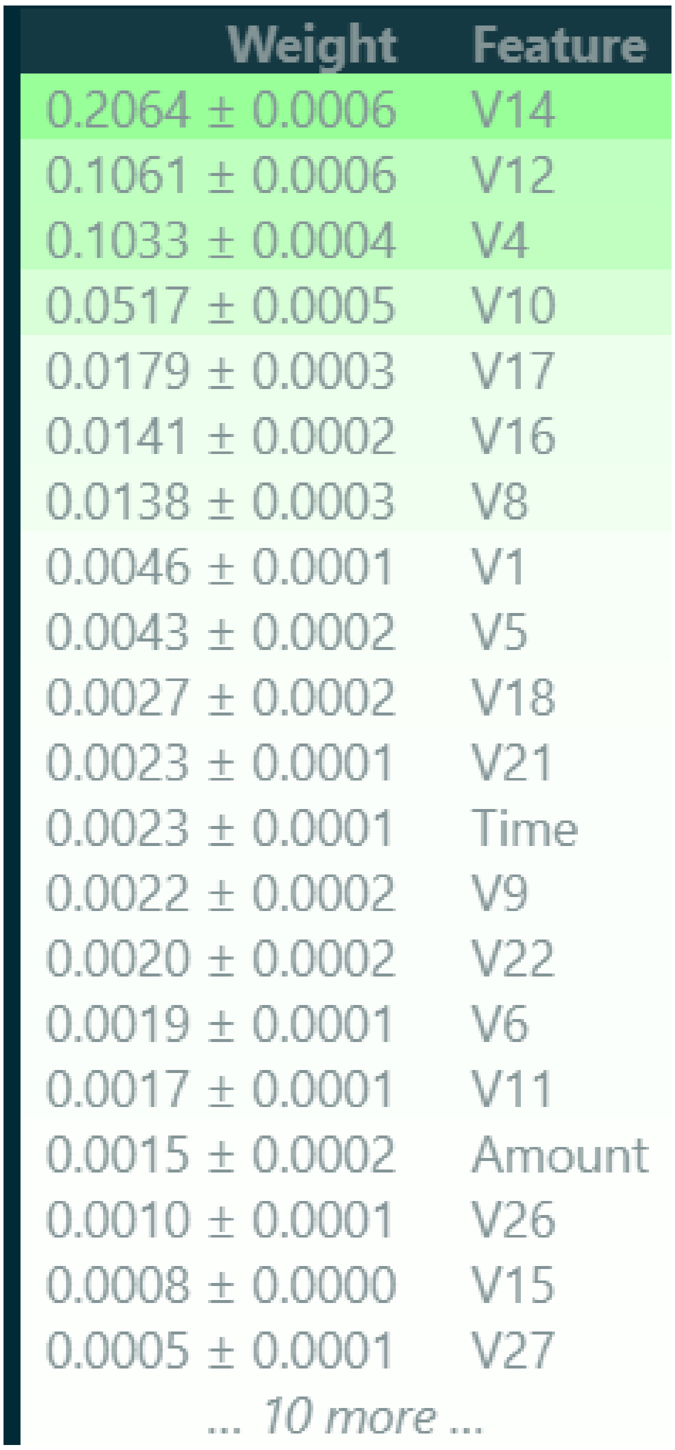
Validation Results

Test Results

From our results, it's evident that some models performed better than others. The Decision Tree model, for instance, showed the least effective performance in terms of precision and accuracy. This could be due to overfitting, a common issue with Decision Trees, or it might be that the model's structure is not well-suited to this particular problem.

Random Forests, on the other hand, demonstrated high effectiveness, ease of implementation, and robustness to class imbalance. This aligns with the findings from the Kaggle dataset, reinforcing the utility of Random Forests in tackling such problems.

However, it's important to note that the nature of payment fraud is continuously evolving, necessitating the need for dynamic and adaptable solutions. Future



Weight	Feature
0.2064 ± 0.0006	V14
0.1061 ± 0.0006	V12
0.1033 ± 0.0004	V4
0.0517 ± 0.0005	V10
0.0179 ± 0.0003	V17
0.0141 ± 0.0002	V16
0.0138 ± 0.0003	V8
0.0046 ± 0.0001	V1
0.0043 ± 0.0002	V5
0.0027 ± 0.0002	V18
0.0023 ± 0.0001	V21
0.0023 ± 0.0001	Time
0.0022 ± 0.0002	V9
0.0020 ± 0.0002	V22
0.0019 ± 0.0001	V6
0.0017 ± 0.0001	V11
0.0015 ± 0.0002	Amount
0.0010 ± 0.0001	V26
0.0008 ± 0.0000	V15
0.0005 ± 0.0001	V27

… 10 more …

Figure 6: ELI5 visualization of feature weights in the model.

Table 4: Training Results

Model	**Confusion Matrix**	**Accuracy**	**Precision**
Logistic Regression	$\begin{bmatrix} 213235 & 5 \\ 132 & 369 \end{bmatrix}$	0.99936	0.98662
Random Forest	$\begin{bmatrix} 213238 & 2 \\ 97 & 404 \end{bmatrix}$	0.99954	0.99505
Gradient Boosting	$\begin{bmatrix} 213231 & 9 \\ 118 & 383 \end{bmatrix}$	0.99940	0.97706

Table 5: Validation Results

Model	**Confusion Matrix**	**Accuracy**	**Precision**
Logistic Regression	$\begin{bmatrix} 71076 & 3 \\ 48 & 85 \end{bmatrix}$	0.99928	0.96591
Random Forest	$\begin{bmatrix} 71078 & 1 \\ 47 & 86 \end{bmatrix}$	0.99932	0.98851
Gradient Boosting	$\begin{bmatrix} 71063 & 16 \\ 52 & 81 \end{bmatrix}$	0.99904	0.83495

work could explore the use of reinforcement learning applied to real-time data streams, potentially offering a more responsive and adaptive approach to fraud detection.

Conclusion

Detecting fraudulent transactions is akin to locating a minuscule fragment in a vast expanse, given the significant and escalating issue of payment fraud in the United States. Our study indicates that with careful tuning, methods such as oversampling and synthetic data generation can enhance predictive performance, particularly when dealing with class imbalance.

While it's unlikely that fraudulent activities will cease, the application of machine learning algorithms can certainly make it more challenging for fraudsters to succeed. Our results, though varied, provide valuable insights into the potential of different machine learning models in the ongoing battle against payment fraud.

Table 6: Test Results

Model	**Confusion Matrix**	**Accuracy**	**Precision**
Random Forest	$\begin{bmatrix} 71079 & 0 \\ 50 & 83 \end{bmatrix}$	0.99930	1.00000
Gradient Boosting	$\begin{bmatrix} 71054 & 25 \\ 58 & 75 \end{bmatrix}$	0.99883	0.75000