Project Report
Applied Statistics and Experimental Design
Credit Card Fraud Detection

Instructor: Dam Quang Tuan

Students:

Hoang Van Nhan	20235542
Le Nguyen Phuoc Thanh	20235561
Tran Quang Trong	20235565
Nguyen Cong Son	20235619
Nguyen Duy Phuc	20235616
Nguyen Duc Manh	20235525

Contents

1 Introduction

Credit card fraud is a significant issue in the financial sector, causing billions of dollars in losses annually for financial institutions and consumers. With the rise of e-commerce and online payments, fraud methods have become increasingly sophisticated and difficult to detect. Developing effective fraud detection systems has become an urgent need to protect users and maintain trust in electronic payment systems.

This project focuses on developing a credit card fraud detection solution based on data from the IEEE-CIS Fraud Detection competition. The primary objective is to build a machine learning model capable of accurately classifying fraudulent and legitimate transactions while achieving high performance on the test dataset.

1.1 Overview of the IEEE-CIS Fraud Detection Competition

The IEEE-CIS Fraud Detection competition, organized by the IEEE Computational Intelligence Society (CIS) and Vesta Corporation on the Kaggle platform, provides a large dataset of real credit card transactions to develop effective fraud detection models. The data includes transaction and identity information, anonymized to protect user privacy.

The main challenges of the competition are:

- Severely imbalanced data (only about 3.5% of transactions are fraudulent)
- Many missing values and anonymized features
- Requirement for high accuracy to minimize both false positives and false negatives
- Complex data structure with many hidden relationships

1.2 Project Objectives

This project was conducted as part of the Applied Statistics and Experimental Design course with the following specific objectives:

- Study and understand the theoretical foundations of credit card fraud detection
- Analyze and preprocess data from the IEEE-CIS Fraud Detection competition
- Develop and evaluate machine learning models for fraud detection
- Improve model performance through feature engineering and parameter optimization
- Evaluate and compare the performance of different methods

Through this project, we not only applied the statistical and experimental design knowledge learned but also developed skills in data analysis and machine learning model development for a practical and meaningful problem.

2 Theoretical Background

2.1 Credit Card Fraud Detection

Credit card fraud detection is a binary classification problem, where the goal is to determine whether a transaction is fraudulent or not. This is a particularly challenging task due to the following reasons:

- **Data Imbalance**: The proportion of fraudulent transactions is typically very low (around 1-3%) compared to the total number of transactions, leading to severe data imbalance.
- **Asymmetric Error Costs**: The cost of missing a fraudulent transaction (false negative) is often much higher than misclassifying a legitimate transaction (false positive).
- **Time-Varying Distribution**: Fraud patterns often change over time as fraudsters develop new methods, leading to the phenomenon of "concept drift."
- **Real-Time Requirement**: Fraud detection systems need to operate near real-time to prevent fraudulent transactions before they are completed.

2.2 Machine Learning Methods in Fraud Detection

Machine learning methods have proven effective in credit card fraud detection. In this project, we focus on Gradient Boosting algorithms, a family of powerful machine learning methods particularly suitable for classification tasks:

2.2.1 Gradient Boosting

Gradient Boosting is a machine learning technique that builds a predictive model as an ensemble of weak models, typically decision trees. The core idea is to construct models sequentially, with each model attempting to correct the errors of the previous ones.

The training process of Gradient Boosting can be described as follows:

1. Initialize the model with a constant value
2. For each iteration $m = 1, 2, ..., M$:
   (a) Compute the gradient of the loss function with respect to the current model's predictions
   (b) Train a weak model (decision tree) to predict the gradient
   (c) Update the model by adding the new weak model with an appropriate weight

In this project, we use three efficient variants of Gradient Boosting:

2.2.2 LightGBM

LightGBM (Light Gradient Boosting Machine) is a Gradient Boosting framework developed by Microsoft, focusing on performance and scalability. Key features of LightGBM include:

- **Histogram-based Algorithm**: Converts continuous feature values into discrete bins, significantly reducing computation time and memory usage.
- **Leaf-wise Tree Growth**: Instead of growing trees level-wise like traditional algorithms, LightGBM grows trees leaf-wise, selecting the leaf with the largest loss to split, leading to faster error reduction.
- **GOSS (Gradient-based One-Side Sampling)**: A sampling technique that retains all samples with large gradients (which contribute significantly to information) and randomly samples from those with small gradients.
- **EFB (Exclusive Feature Bundling)**: A technique to bundle sparse features, reducing the number of features without significant loss of information.

2.2.3 XGBoost

XGBoost (eXtreme Gradient Boosting) is an optimized Gradient Boosting algorithm known for its high performance and ability to handle large datasets. Key features of XGBoost include:

- **Regularization**: Adds regularization terms to the objective function to control model complexity and prevent overfitting.
- **Approximate Greedy Algorithm**: Uses an approximate greedy algorithm to find optimal splits, improving performance on large datasets.
- **Sparsity Awareness**: Efficiently handles sparse data through optimized tree-splitting algorithms.
- **Built-in Cross-Validation**: Supports integrated cross-validation for easy parameter tuning.

2.2.4 CatBoost

CatBoost is a Gradient Boosting algorithm developed by Yandex, particularly effective for handling categorical data. Key features of CatBoost include:

- **Ordered Boosting**: A novel boosting algorithm that minimizes target leakage during training.
- **Automatic Handling of Categorical Features**: Automatically processes categorical features using advanced encoding methods.
- **Fast and Scalable Implementation**: Efficient implementation for both CPU and GPU, enabling fast training on large datasets.
- **Robust to Overfitting**: Designed to minimize overfitting, especially when working with imbalanced data.

2.3 Ensemble Techniques

Ensemble techniques combine multiple models to create a stronger model. In this project, we use Blending, a simplified form of Stacking:

- **Blending**: Combines predictions from multiple models by taking a weighted average. Weights are determined through experimentation on the validation set.
- **Stacking**: Uses predictions from multiple base models as input to a meta-learner. In this project, we use LogisticRegression as the meta-learner.

2.4 Performance Evaluation

In credit card fraud detection, selecting appropriate evaluation metrics is crucial due to the imbalanced nature of the data. We use AUC-ROC (Area Under the Receiver Operating Characteristic Curve) as the primary metric:

- **AUC-ROC**: Measures the model's ability to distinguish between positive and negative classes. AUC-ROC is unaffected by data imbalance and classification thresholds.
- **Validation Strategy**: Uses k-fold cross-validation and time-based validation to reliably evaluate model performance.

3 Data Analysis

3.1 Data Description

The data from the IEEE-CIS Fraud Detection competition includes two main types of information:

- **Transaction Data**: Contains information about transactions, such as amount, time, and other transaction-related features.
- **Identity Data**: Contains information about the user performing the transaction, such as email address, device, IP address, etc.

The training dataset includes 590,540 transactions, of which 20,663 are fraudulent (approximately 3.5%). The test dataset includes 506,691 transactions to be classified.

3.2 Feature Analysis

3.2.1 Target Variable Distribution

The target variable `isFraud` is severely imbalanced, with only about 3.5% of transactions labeled as fraudulent.

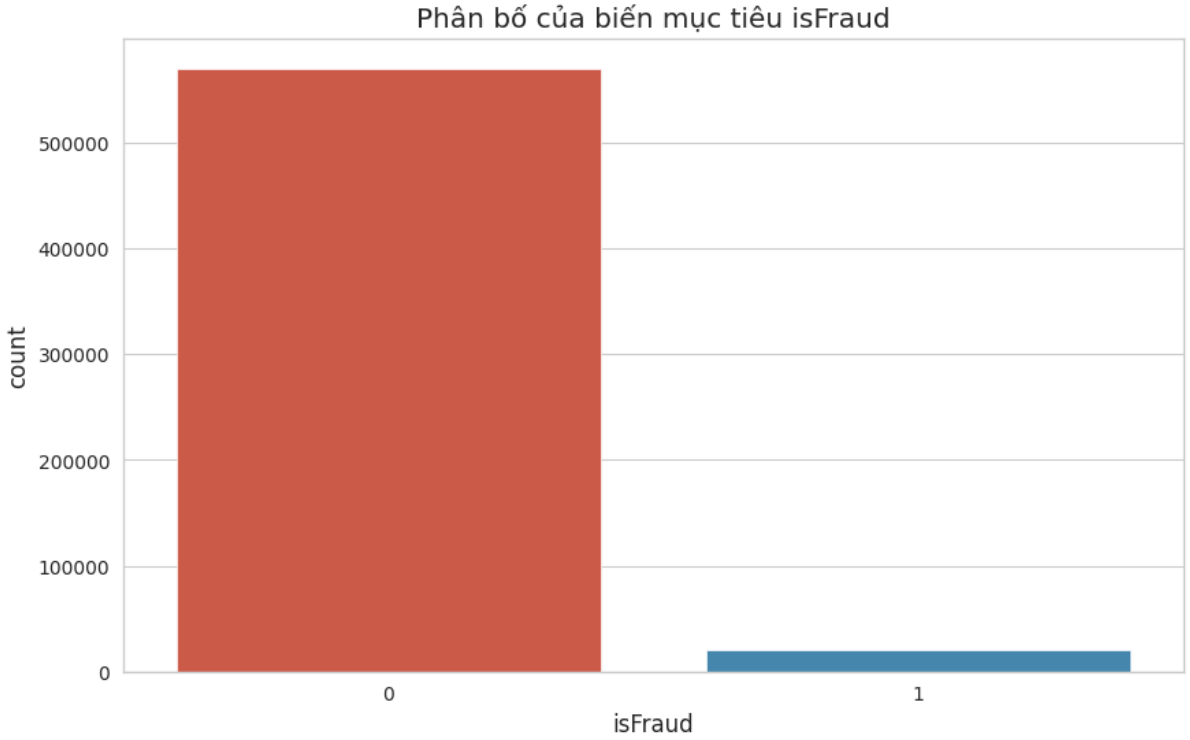

Figure 1: Distribution of the Target Variable isFraud

3.2.2 Transaction Amount Analysis

The transaction amount (`TransactionAmt`) is a critical feature in fraud detection. Analysis shows that fraudulent transactions often have a different amount distribution compared to legitimate transactions.

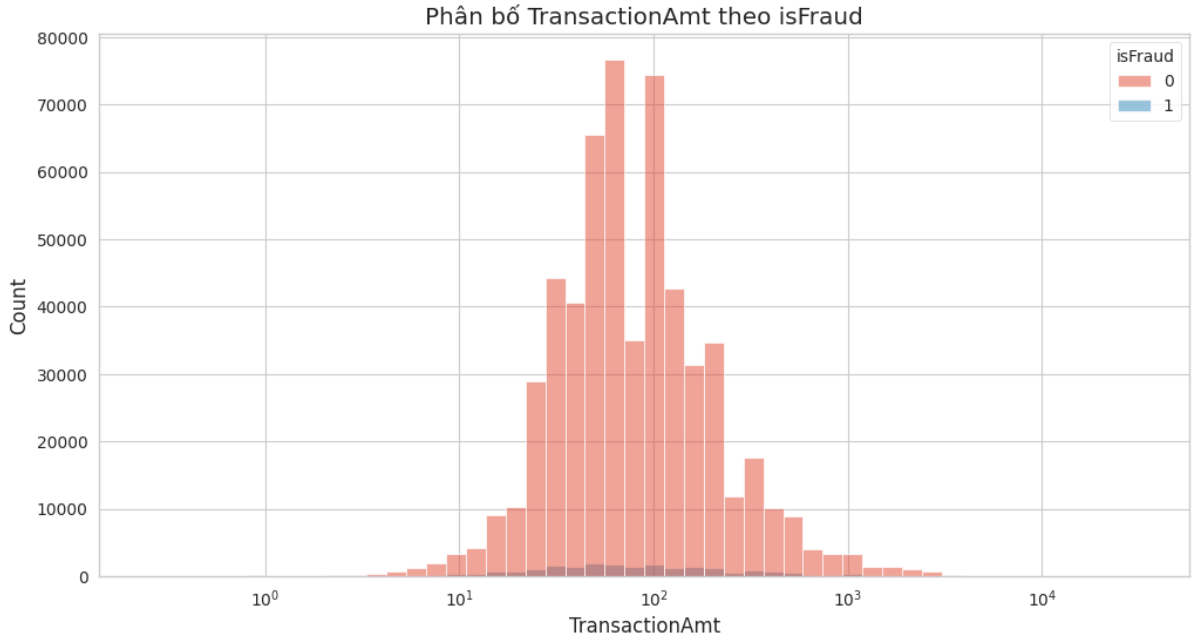

Figure 2: Distribution of TransactionAmt by isFraud

Although significantly fewer in number, the distribution of fraudulent transactions (isFraud = 1) shows some important differences in shape compared to legitimate transactions (isFraud = 0).

Concentration: The distribution of legitimate transactions has a very high and sharp peak, strongly concentrated at low transaction values (around 10^2). In contrast, the distribution of fraudulent transactions appears flatter and more spread out, without a distinct sharp peak.

In particular, the decimal part of the transaction amount also shows significant differences between fraudulent and legitimate transactions:

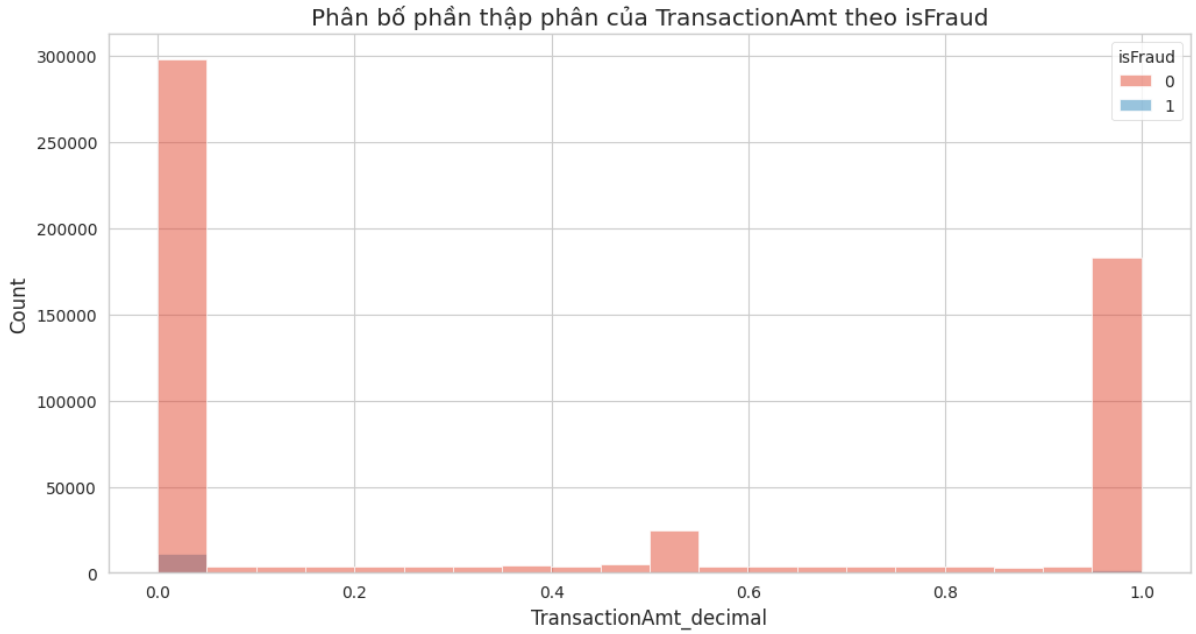

Figure 3: Distribution of the Decimal Part of TransactionAmt by isFraud

The chart shows a complete difference in the distribution of decimal parts between legitimate and fraudulent transactions.

For Legitimate Transactions (isFraud = 0): The distribution has a complex and diverse structure, reflecting real consumer behavior. There are three very distinct main clusters:

- A huge peak at the value 0.0 (corresponding to transactions with even amounts like $50.00).
- Another large peak near the value 1.0 (corresponding to 'psychological pricing' like $19.99, $24.95).
- A smaller but significant peak at the value 0.5 (corresponding to values like $12.50).

For Fraudulent Transactions (isFraud = 1): In complete contrast, the distribution is extremely simple and lacks diversity. Almost all fraudulent transactions have a decimal part of 0.0. The bars in the chart at any other decimal values are virtually non-existent.

This sharp contrast is undeniable evidence: while legitimate transactions have a diverse and patterned 'decimal footprint,' fraudulent transactions exhibit a rigid and monotonous pattern, almost exclusively whole numbers. This is an extremely strong fraud detection signal.

3.2.3 Time-Based Analysis

The fraud rate varies over time during the day, with certain hours showing significantly higher fraud rates:

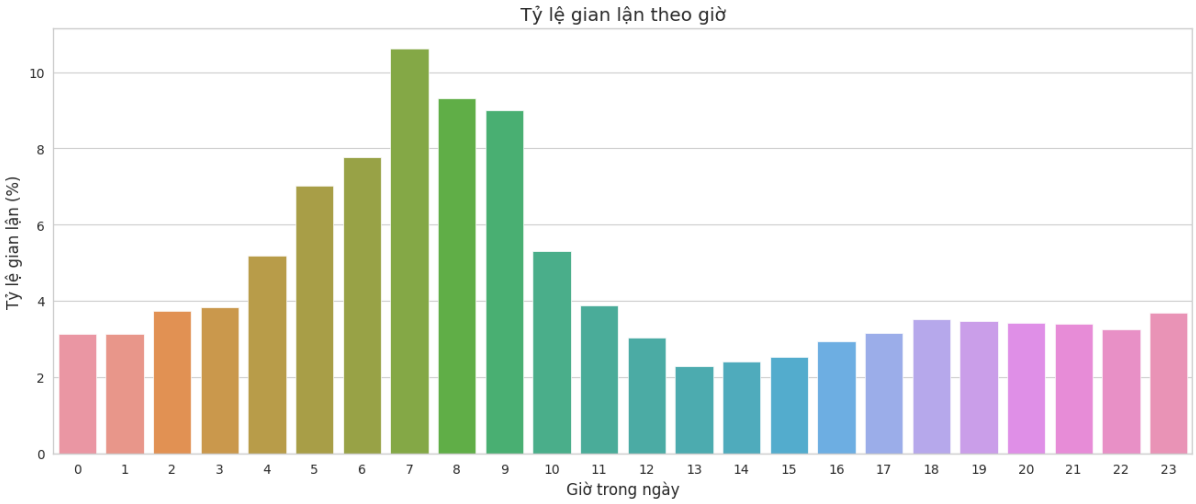

Figure 4: Fraud Rate by Hour

Looking at the chart, we can see a clear story about fraudulent behavior throughout the 24 hours:

Early Morning (Around 4-10 AM): This is the "red zone" of fraudulent activity. Starting at 4 AM, the fraud rate surges abruptly, peaking at 7 AM with the highest rate, exceeding 10%. The period from 7 to 9 AM remains very high. This indicates that early morning is the riskiest time.

Noon and Afternoon (Around 11 AM-4 PM): After the morning peak, the fraud rate drops sharply and reaches its lowest point around 1 PM (1 PM). This is the most "peaceful" time of the day with the lowest fraud rate.

Evening and Night (Around 5 PM-3 AM): The fraud rate tends to rise slightly in the evening but remains relatively low and stable, much lower than the alarming morning levels.

The visual evidence from the chart is undeniable. It not only confirms that the fraud rate fluctuates significantly throughout the day but also precisely identifies the "prime time" for criminals as the morning, particularly from 7 to 9 AM.

3.2.4 Customer Identity Analysis

A key finding is the relationship between the number of transactions per customer (identified by UID) and the fraud rate:

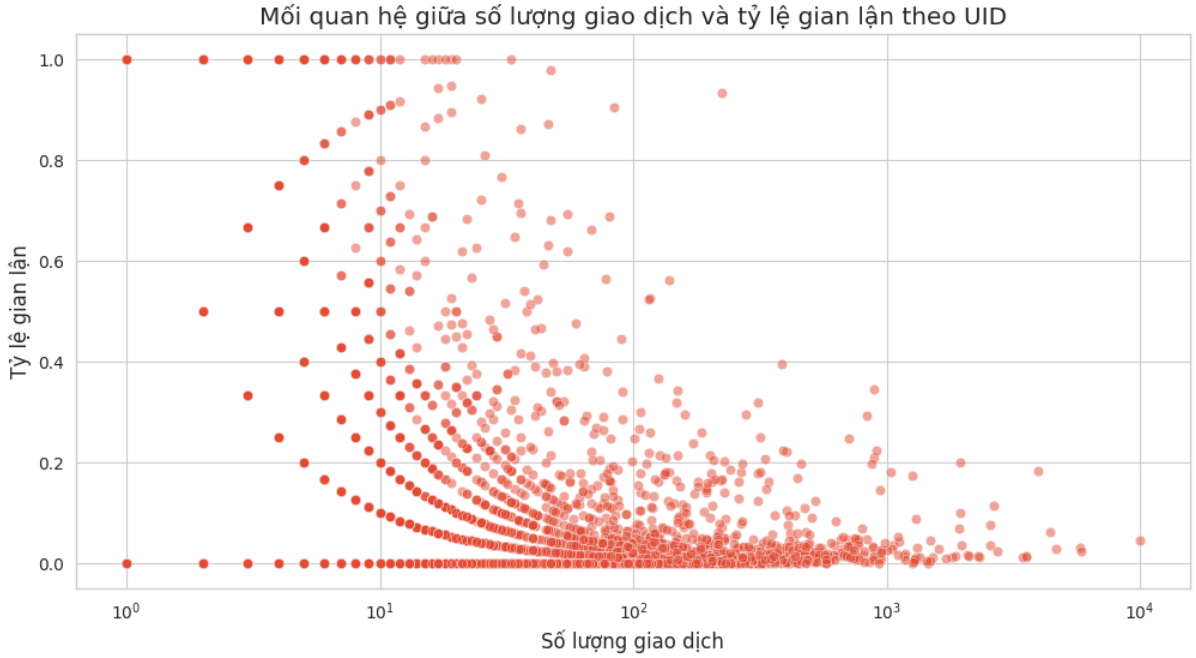

Figure 5: Relationship between Transaction Volume and Fraud Rate by UID

The chart shows that customers with fewer transactions tend to have higher fraud rates, while those with more transactions have lower fraud rates. This suggests that classifying by customer rather than individual transactions may improve model performance.

4 Methodology

4.1 Data Preprocessing

4.1.1 Handling Missing Values

The dataset contains many missing values, especially in the identity data. We applied the following strategies to handle missing values:

- For numerical features: Replace with -999 (a special value not present in the original data)
- For categorical features: Replace with the string 'missing'
- Create missing value indicator features to allow the model to learn from the pattern of missing data

4.1.2 Feature Encoding

Categorical features were encoded using Label Encoding to convert them into a numerical format that machine learning algorithms can process:

- Use `LabelEncoder` from scikit-learn to convert categorical values into integers
- Apply Frequency Encoding for categorical features with many unique values

4.2 Feature Engineering

4.2.1 Creating Customer Identifier (UID)

One of the most important techniques was creating a unique customer identifier (UID) by combining features related to cards and addresses:

- `uid1 = card1 + addr1`: Combines card number and primary address
- `uid2 = card1 + addr1 + addr2`: Adds secondary address
- `uid3 = card1 + card2 + card3`: Combines card information
- `uid4 = card1 + card2 + addr1`: Combines card and address information
- `uid5 = P_emaildomain + card1 + addr1`: Combines email domain, card, and address

4.2.2 Aggregated Features

Based on the created UIDs, we computed aggregated features to capture customer behavior:

- `uid_count`: Number of transactions per UID
- `uid_amt_mean/std/max/min`: Statistics of transaction amounts per UID
- `uid_hour_nunique`: Number of unique hours in which the UID made transactions
- `uid_day_nunique`: Number of unique days in which the UID made transactions

4.2.3 Time-Based Features

We converted `TransactionDT` (transaction time in seconds) into meaningful time-based features:

- `DT_hour`: Hour of the day (0-23)
- `DT_day`: Day of the week (0-6)
- `DT_M`: Month (1-12)
- `DT_W`: Week of the year
- `is_weekend`: Marks transactions on weekends
- `is_night`: Marks transactions at night

Additionally, we created cyclical time features using sine and cosine transformations:

- `hour_sin/cos`: Cyclical representation of the hour of the day
- `day_sin/cos`: Cyclical representation of the day of the week
- `month_sin/cos`: Cyclical representation of the month of the year

4.2.4 Transaction Amount Features

We created additional features from the transaction amount:

- `TransactionAmt_decimal`: Decimal part of the amount
- `TransactionAmt_decimal_len`: Length of the decimal part
- `TransactionAmt_round/round_10/round_100`: Rounded values
- `TransactionAmt_outlier`: Marks transactions with unusual amounts

4.2.5 Interaction Features

We created interaction features between the transaction amount and other features:

- `TransactionAmt_to_mean_card1`: Ratio of the transaction amount to the mean amount of card1
- `TransactionAmt_to_std_card1`: Ratio of the transaction amount to the standard deviation of card1
- `TransactionAmt_to_mean_addr1`: Ratio of the transaction amount to the mean amount of addr1

4.2.6 Feature reduction through correlation matrix

Reduction Methods: Selected a maximum-sized subset of uncorrelated columns from each group: A correlation matrix was calculated for the columns in each group. Columns with high correlation (e.g., above 0.7) were removed, leaving only those that weren't too similar. This resulted in a smaller set of columns, each adding distinct value.

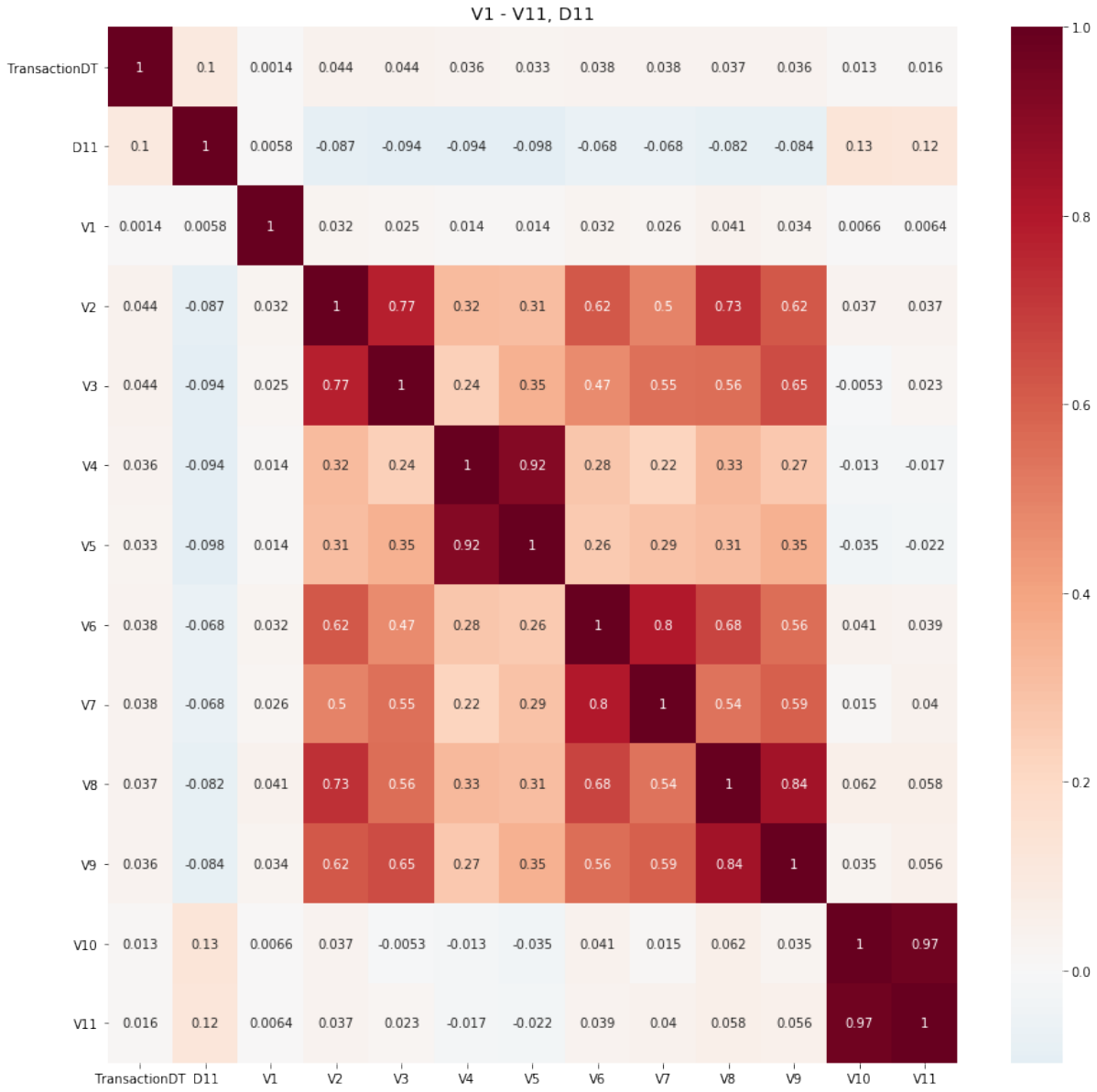

Figure 6: Corelation between V1 - V11, D11, TransactionDT

Variable Pair	**Correlation Coefficient**
V2 and V8	0.73
V2 and V3	0.77
V6 and V7	0.80
V8 and V9	0.84
V4 and V5	0.92
V10 and V11	0.97

All listed pairs exhibit **strong positive correlation**. To mitigate multicollinearity, we decide to **remove** one variable from each highly correlated pair. The decisions are as follows:

- **Remove V8**: Due to high correlation with V2 (0.73) and V9 (0.84).
- **Remove V3**: Due to high correlation with V2 (0.77).
- **Remove V5**: Due to high correlation with V4 (0.92).
- **Remove V10**: Due to high correlation with V11 (0.97).

Conclusion: The variables V8, V3, V5, and V10 will be removed to ensure the independence of the remaining variables in the model.

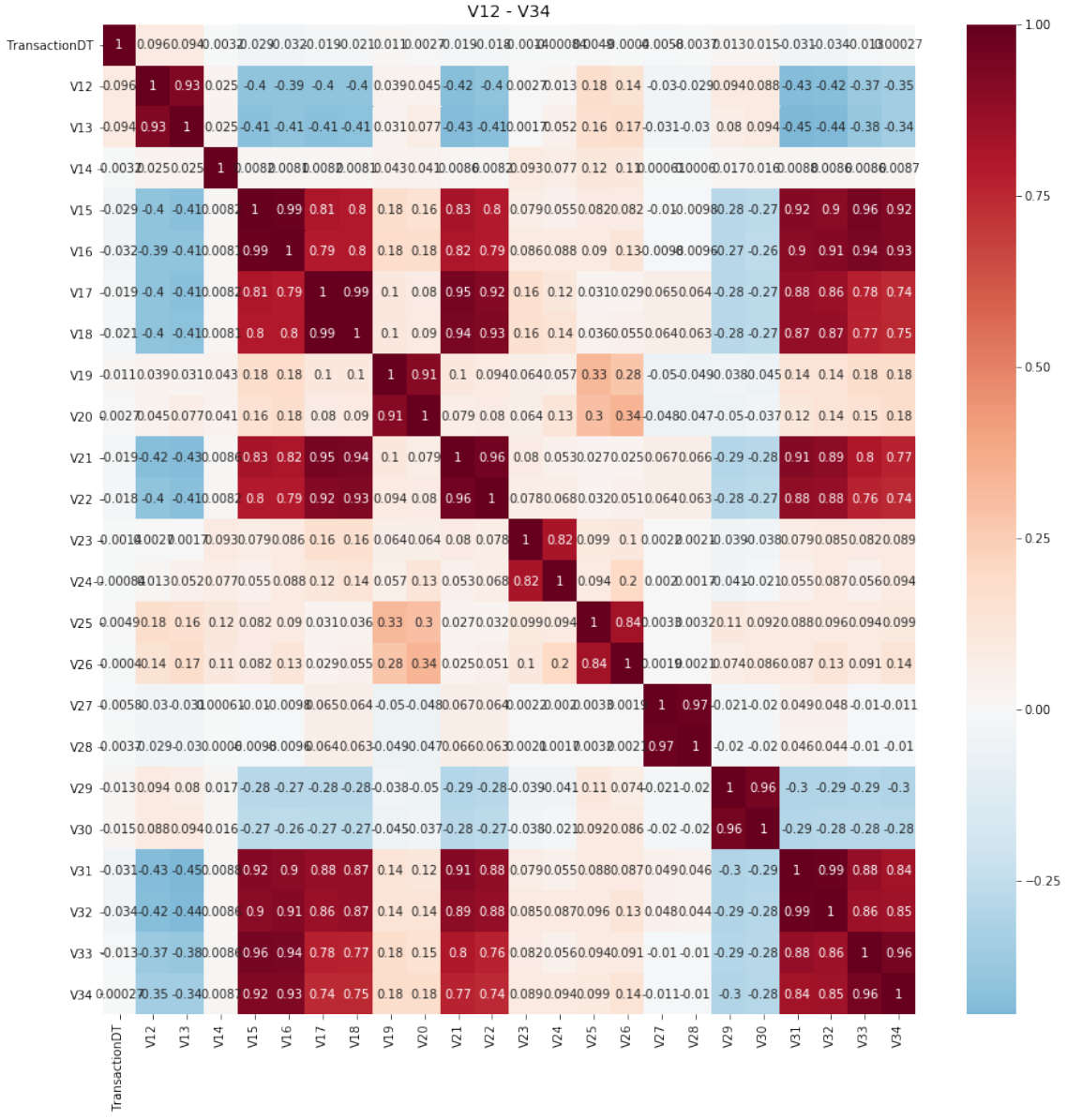

Figure 7: Corelation between V12 - V34, TransactionDT

- **Feature 13**: Retained from the pair [12, 13] with correlation 0.93.
- **Feature 17**: Retained from the group [15, 16, 17, 18, 21, 22, 31, 32, 33, 34] with correlations in [0.74–0.96].
- **Feature 20**: Retained from the pair [19, 20] with correlation 0.91.
- **Feature 23**: Retained from the pair [23, 24] with correlation 0.82.
- **Feature 26**: Retained from the pair [25, 26] with correlation 0.84.
- **Feature 27**: Retained from the pair [27, 28] with correlation 0.97.
- **Feature 30**: Retained from the pair [29, 30] with correlation 0.96.

Conclusion: The features [13, 17, 20, 23, 26, 27, 30] are selected to ensure the independence of variables in the model while preserving representativeness.

4.3 Validation Strategy

We tested three different validation strategies:

1. **Simple Train-Test Split**: Split the data into 80% training and 20% validation
2. **StratifiedKFold**: Split the data into 5 folds, ensuring the same fraud ratio in each fold
3. **Time-Based Cross-Validation**: Split the data by time to simulate real-world scenarios

Time-Based Cross-Validation provided the best results as it more accurately simulates how the model will be used in practice, predicting new transactions based on historical data.

4.4 Model Training

Since the credit card fraud detection problem is a binary classification problem, it makes sense to use binary log loss (also known as logistic loss or cross entropy loss). This loss function measures the difference between the probability predicted by the model and the actual label of each transaction.

Binary log loss not only penalizes incorrect predictions, but also heavily penalizes predictions that the model is overconfident about but does not match the actual label. This helps the model learn to fine-tune the probabilities, leading to better discrimination between fraudulent and legitimate transactions. This is also the reason why boosting algorithms such as LightGBM, XGBoost, and CatBoost are designed with this loss function when applied to this problem.

The Binary Log Loss, also known as Binary Cross-Entropy Loss, is defined as follows:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{1}$$

where:

- $y_i \in \{0, 1\}$ is the true label for the i-th sample,
- $\hat{y}_i \in [0, 1]$ is the predicted probability for the i-th sample,
- N is the total number of samples.

We trained three Gradient Boosting models—LightGBM, XGBoost, and CatBoost—with tuned parameters:

4.4.1 LightGBM

```
lgb_params = {
    "objective": "binary",
    "boosting_type": "gbdt",
    "metric": "auc",
    "n_jobs": -1,
    "learning_rate": 0.01,
    "num_leaves": 256,
    "max_depth": 8,
    "tree_learner": "serial",
    "colsample_bytree": 0.7,
    "subsample": 0.7,
    "min_child_weight": 1,
    "min_child_samples": 20,
    "scale_pos_weight": 1,
    "reg_alpha": 0.1,
    "reg_lambda": 0.1,
    "verbose": -1,
    "seed": 42
}
```

4.4.2 XGBoost

```
xgb_params = {
    "objective": "binary:logistic",
    "eval_metric": "auc",
    "eta": 0.02,
    "max_depth": 8,
    "min_child_weight": 1,
    "subsample": 0.7,
    "colsample_bytree": 0.7,
    "alpha": 0.1,
    "lambda": 0.1,
    "tree_method": "hist",
    "nthread": -1,
    "scale_pos_weight": 1,
    "seed": 42
}
```

4.4.3 CatBoost

```
cat_params = {
    "iterations": 10000,
    "learning_rate": 0.02,
    "depth": 8,
    "l2_leaf_reg": 10,
    "bootstrap_type": "Bernoulli",
    "subsample": 0.7,
    "scale_pos_weight": 1,
    "eval_metric": "AUC",
    "metric_period": 100,
    "od_type": "Iter",
    "od_wait": 200,
    "random_seed": 42,
    "allow_writing_files": False,
    "task_type": "GPU",
    "verbose": 100
}
```

4.5 Model Ensemble

We tested two model ensemble methods:

1. **Blending**: Combined predictions from the three models with different weights
2. **Stacking**: Used LogisticRegression as a meta-learner on out-of-fold (OOF) predictions from the three models

We experimented with various weights and selected the best weights based on AUC on the validation set.

5 Experimental Results

5.1 Model Improvement Process

We developed three model versions with continuous improvements:

5.1.1 Version 1: Simple Model Ensemble

The first version used a simple train-test split and combined the three models with manually tuned weights:

- Weights: LightGBM (0.1), XGBoost (0.8), CatBoost (0.1)
- AUC on Private Test: 0.85402
- AUC on Public Test: 0.890363

5.1.2 Version 2: Using StratifiedKFold

The second version improved the validation strategy by using StratifiedKFold:

- Weights: LightGBM (0.1), XGBoost (0.6), CatBoost (0.3)
- AUC on Private Test: 0.890895
- AUC on Public Test: 0.916814

5.1.3 Version 3: Time-Based Cross-Validation and Stacking

The final version used Time-Based Cross-Validation and tested both Stacking and Blending:

- Best Weights: LightGBM (0.2), XGBoost (0.6), CatBoost (0.2)
- AUC on Private Test: 0.900404
- AUC on Public Test: 0.925547

5.1.4 Version 4: Feature reduction

This is a special version that differs from all previous versions in that it reduces the number of features to reduce the load on the model. This version is to test how applying Feature reduction will affect the final result.

Data Preprocessing: Feature selection

V Columns: more than 300 columns Vxxx → Complicated → Should be Reduced.

Grouping: Identified groups of V columns with similar NaN structures → Feature reduction.

Time consistency: train a single model using a single feature (or small group of features) on the first month of train dataset and predict isFraud for the last month of train dataset.

Discovery: 95% features training is consistent over time. But AUC around 0.60 and validation AUC 0.40. In other words some features found patterns in the present that did not exist in the future, V322 – V399 in particular. We should eliminate these ones.

Model Training and Validating

Cross-Validation with GroupKFold using months as groups: This ensures that the temporal structure of the data is preserved, allowing us to test the model across different time periods while avoiding data leakage between training and validation sets.

Analysis Based on UIDs: Evaluating model performance by analyzing how well it classified transactions associated with known versus unknown clients.

- Model is used in this version: XGBoost. The model outperforms clients with prior transaction history.
- AUC on Private Test: 0.913516
- AUC on Public Test: 0.946695

5.2 Performance Comparison

The table below summarizes the performance of the four model versions:

Version	Validation Strategy	Private AUC	Public AUC
Version 1	Train-Test Split	0.85402	0.890363
Version 2	StratifiedKFold	0.890895	0.916814
Version 3	Time-Based CV	**0.900404**	**0.925547**
Version 4	Feature reduction	**0.913516**	**0.946695**

Table 1: Performance Comparison of Model Versions

5.3 Feature Importance Analysis

We analyzed feature importance from all three models to better understand the factors influencing fraud detection:

5.3.1 LightGBM Feature Importance

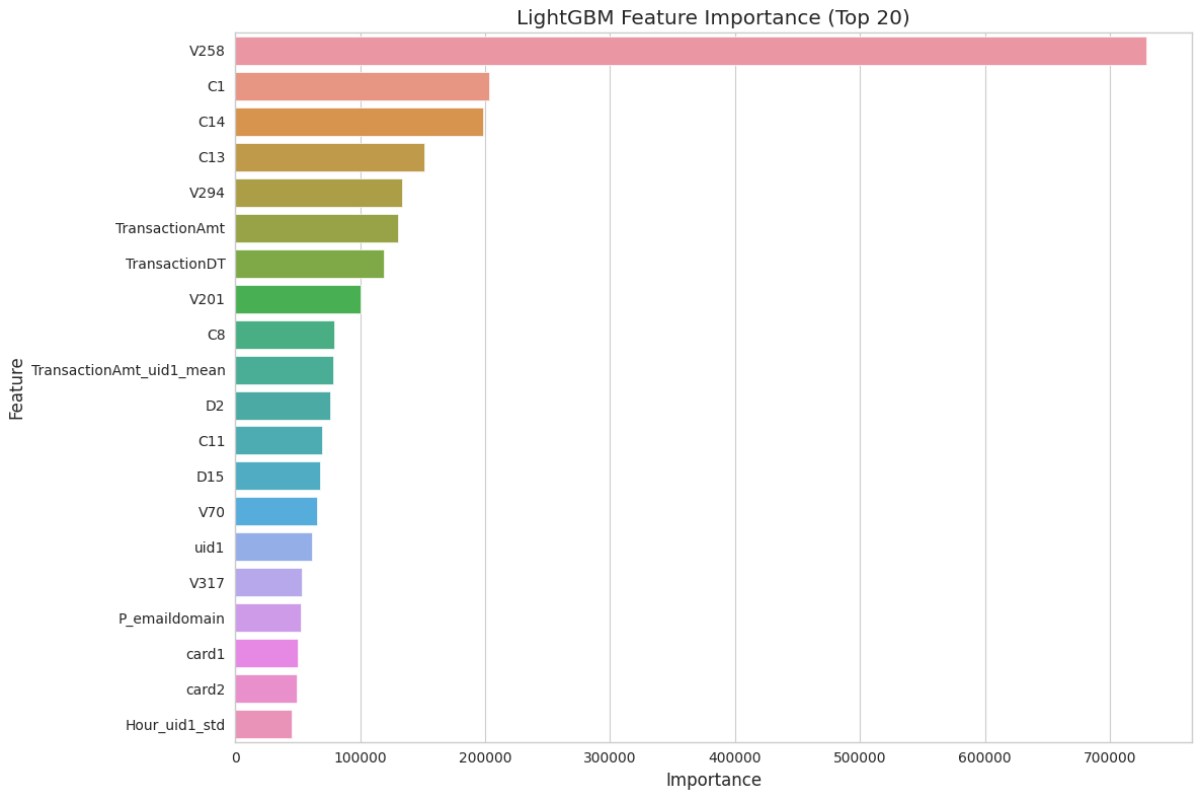

Figure 8: LightGBM Feature Importance (Top 20)

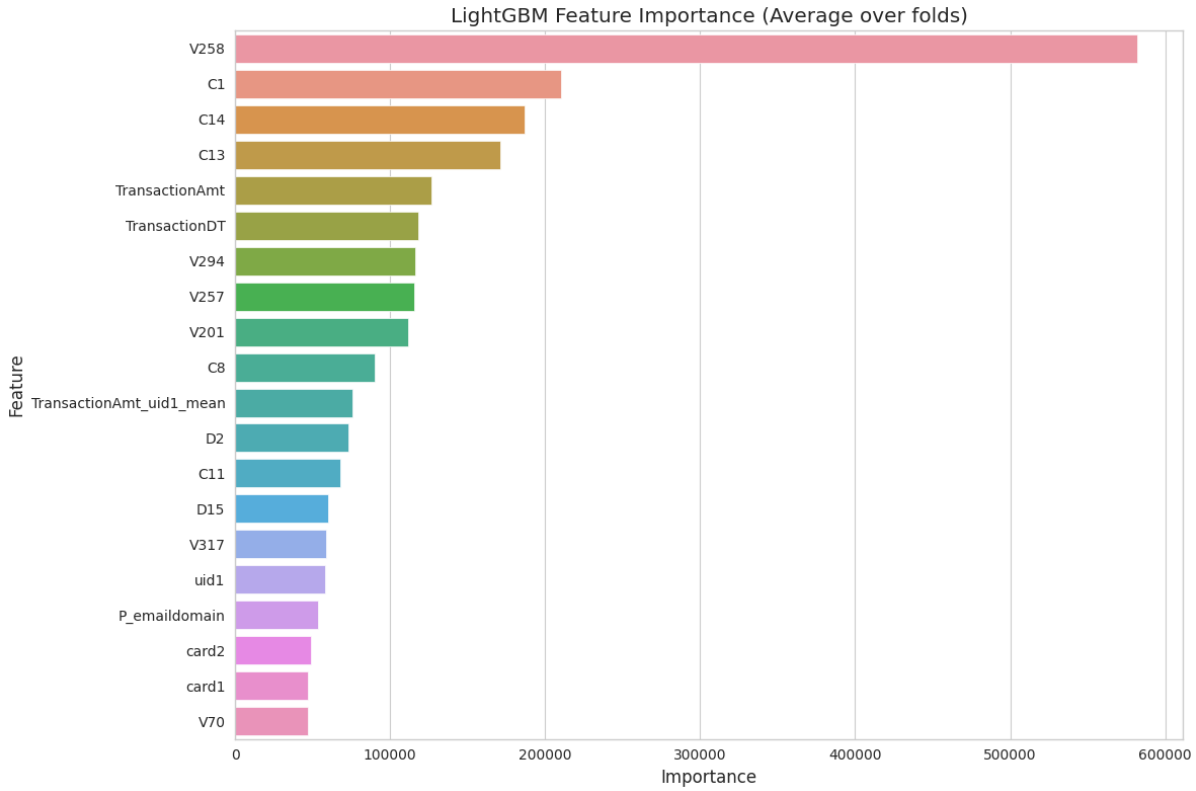

Figure 9: LightGBM Feature Importance (Average over Folds)

5.3.2 XGBoost Feature Importance

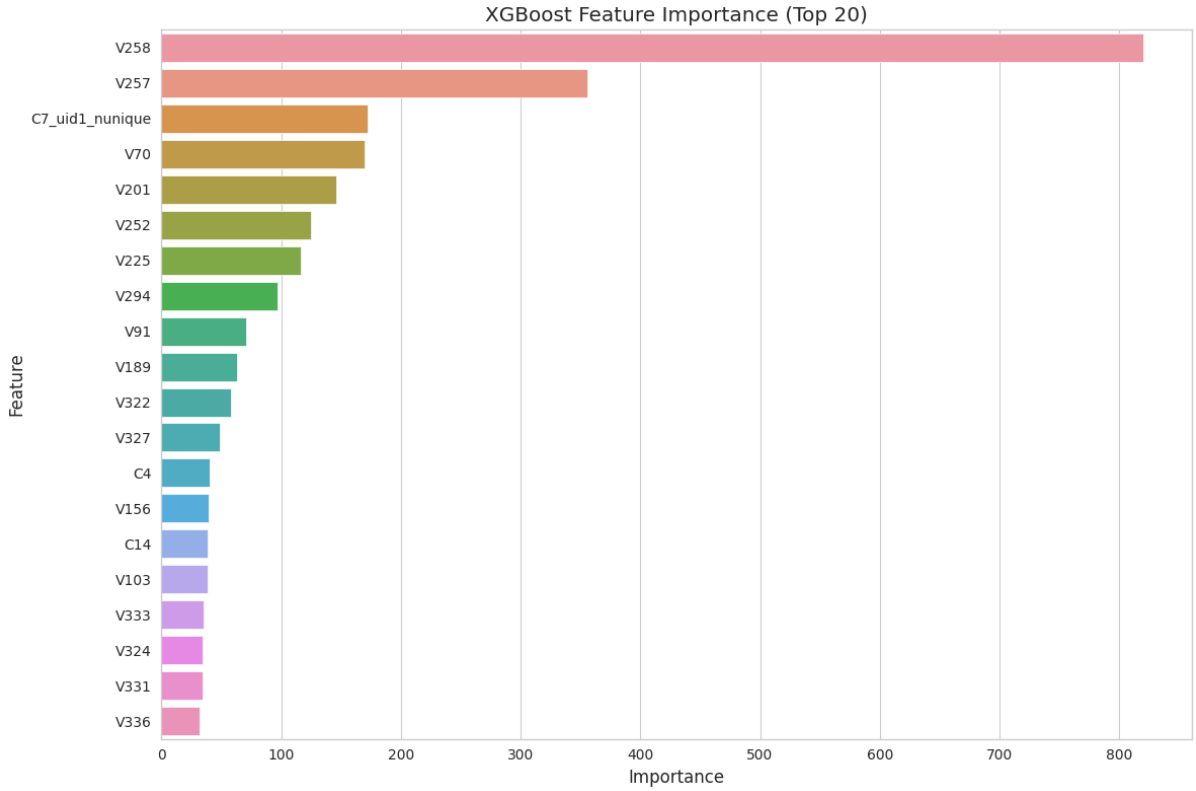

Figure 10: XGBoost Feature Importance (Top 20)

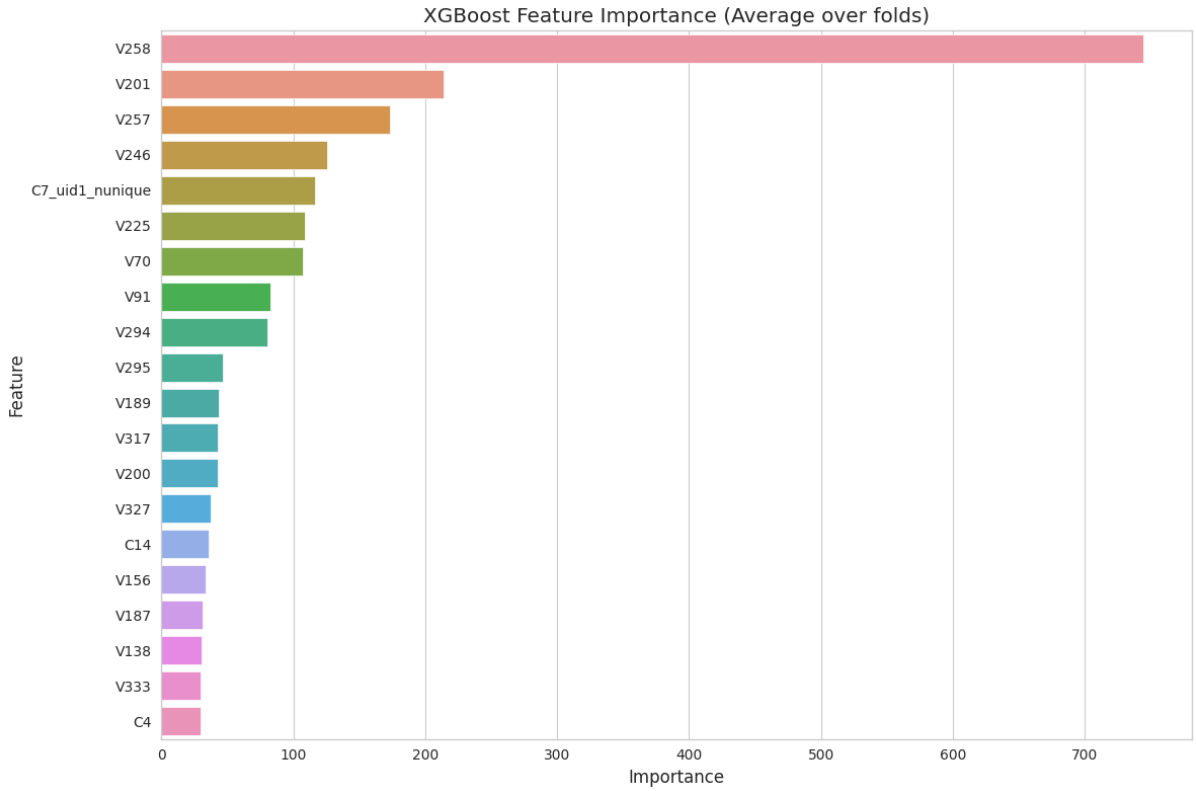

Figure 11: XGBoost Feature Importance (Average over Folds)

5.3.3 CatBoost Feature Importance

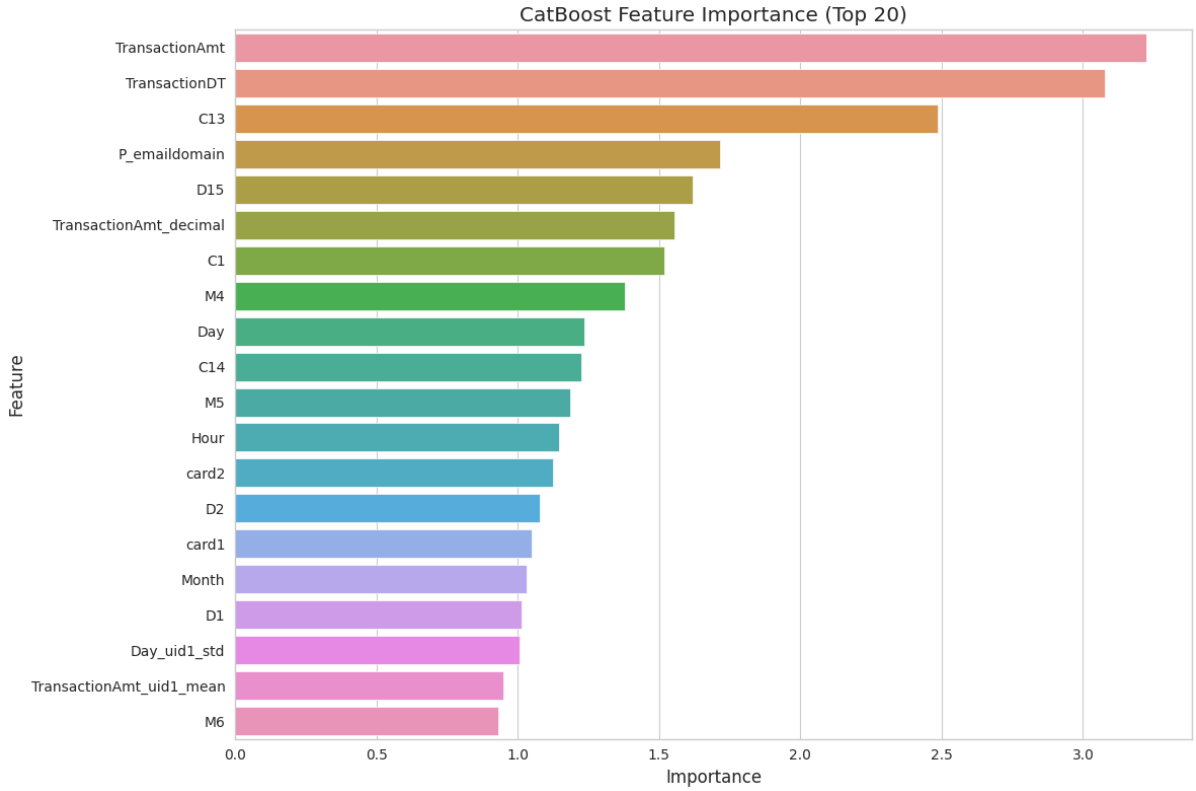

Figure 12: CatBoost Feature Importance (Top 20)

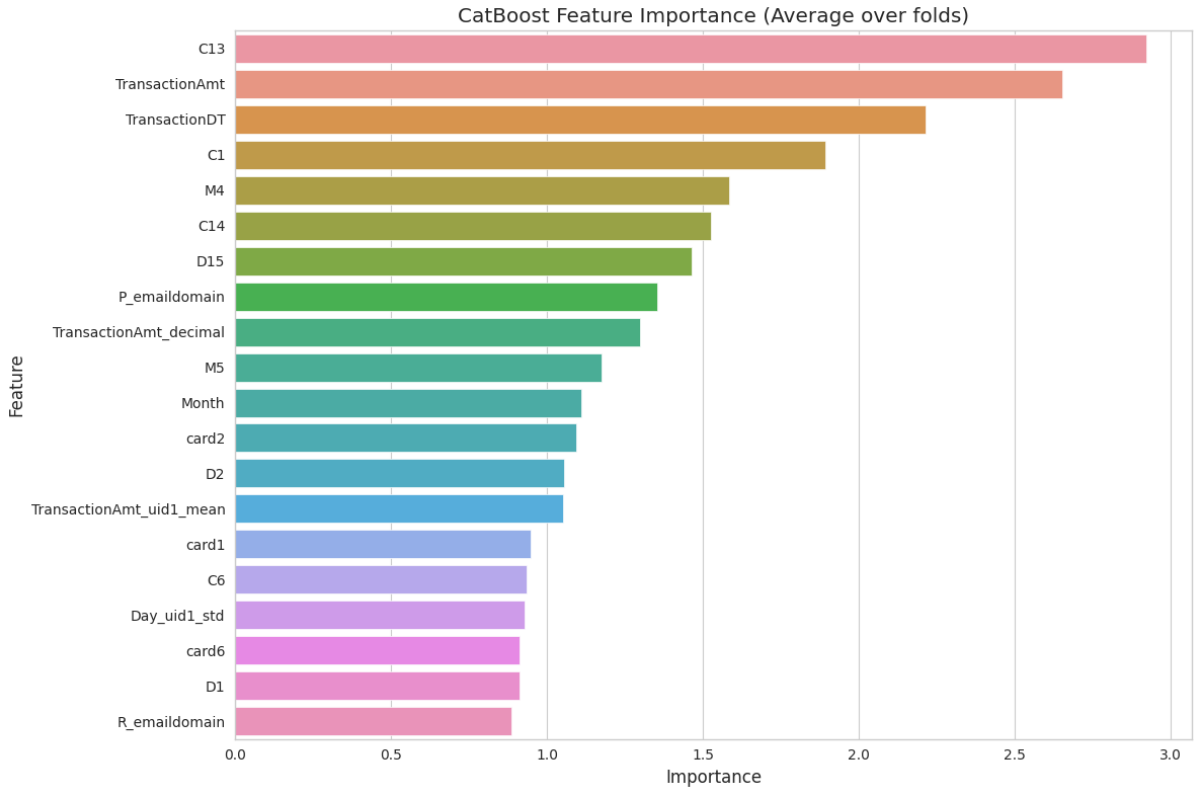

Figure 13: CatBoost Feature Importance (Average over Folds)

From the charts above, we can identify some of the most important features in fraud detection:

- **V258**: An anonymized feature, but highly important in both LightGBM and XGBoost
- **TransactionAmt and TransactionDT**: Transaction amount and time are significant features in CatBoost
- **C1, C13, C14**: Card-related features are highly important in multiple models
- **uid1 features**: The presence of uid1 attributes in the top attributes reflects the importance of customer-based classification.
- **TransactionAmt_decimal**: Decimal part of the transaction amount, a key feature in CatBoost

The contribution of the features in CatBoost is quite even while in LightGBM and XGBoost, they are completely dependent on the V258 feature. This will have a negative impact if the real-world testing data is no longer heavily influenced by V258.

So in the long run, if we want to apply it in practice (the kaggle dataset is only provided by Vesta), we can try to remove some features during the training process to avoid overfitting the model.

6 Discussion

6.1 Result Analysis

The experimental results highlight several key points:

- **Validation Strategy**: Time-Based Cross-Validation provided the best results, underscoring the importance of accurately simulating real-world model usage.
- **Model Ensemble**: Combining the three Gradient Boosting models with appropriate weights significantly improved performance compared to individual models.
- **Feature Engineering**: Aggregated UID-based features and time-based features played a critical role in improving model performance.
- **Continuous Improvement**: The improvement process from Version 1 to Version 3 demonstrates the importance of iterative experimentation and refinement.
- **Feature reduction**: We reduced over 300 features by grouping those with similar missing data patterns and testing for temporal consistency. Features in the V322–V399 range, which showed a marked drop in validation performance (from 0.60 in training to 0.40), were removed to simplify the model and improve its robustness.

6.2 Comparison with Top Solutions

Compared to the first-place solution in the competition, our model shares some similarities and differences:

- **Similarities**: Both use an ensemble of multiple Gradient Boosting models and focus on customer-based feature engineering.
- **Differences**: The first-place solution used more complex post-processing, replacing individual transaction predictions with customer-level average predictions. In contrast, we focused on improving validation strategies and advanced time-based feature engineering.

6.3 Limitations and Future Improvements

Despite achieving good results, our model has some limitations:

- **Parameter Optimization**: We did not use automated parameter optimization tools like Optuna to find optimal parameters for each model.
- **Post-Processing**: We did not fully implement advanced post-processing techniques as in the first-place solution.
- **Advanced Features**: More complex features, such as transaction graph-based features or time-series analysis, could be created.

Future improvements include:

- Using Optuna for automated parameter optimization
- Implementing advanced post-processing techniques
- Experimenting with deep learning models like Neural Networks
- Creating additional features based on time-series and transaction graph analysis

7 Conclusion

In this project, we developed a credit card fraud detection solution based on data from the IEEE-CIS Fraud Detection competition. We applied advanced techniques in data preprocessing, feature engineering, and model ensembling to achieve high performance.

The continuous improvement process from Version 1 to Version 3 significantly enhanced model performance, with the AUC on the Private Test increasing from 0.85402 to 0.900404. This highlights the importance of selecting appropriate validation strategies, effective feature engineering, and combining multiple models.

Along with the Feature reduction technique in Version 4 (0.913516 on private test), it showed the importance of considering the correlation of features and then being able to select a more suitable feature set for the model.

The results and analyses from this project are not only valuable for addressing the credit card fraud detection problem but also provide deep insights into applying statistical and machine learning methods to real-world imbalanced classification problems.

8 References

1. First-place solution in the IEEE-CIS Fraud Detection competition. https://www.kaggle.com/competitions/ieee-fraud-detection/discussion/111308
2. Solution from a high-scoring participant.

   https://github.com/shejz/IEEE-CIS-Fraud-Detection/tree/master
3. Detailed guide on IEEE-CIS Fraud Detection. https://towardsdatascience.com/ieee-cis-fraud-detection-top-5-solution-5488fc66e95f
4. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30.
5. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794).
6. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. Advances in neural information processing systems, 31.
7. Dal Pozzolo, A., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In 2015 IEEE Symposium Series on Computational Intelligence (pp. 159-166).

8. Breiman, L. (1996). Stacked regressions. Machine learning, 24(1), 49-64.

You can also view my code and submission file for the competition on Credit-card-fraud-detection