ML for Smarter Credit Card Fraud Detection

From Raw Transactions to Insights

ISYE 6740: Team 11

Wenxin Jiang, Yu Zheng, Yihan Liao, Qihao Cheng

Motivation & Objective

We live in a world where billions of credit card transactions occur daily. With this massive volume of exchanges comes a critical challenge: **fraud detection.**

We aim to address the challenge of identifying fraudulent transactions within large-scale payment data using machine learning techniques. Our objectives are to **identify the best-performing model** and **enhance the precision of fraud detection** while maintaining efficiency in high-throughput environments.

True	False
True Positive (TP)	False Positive (FP)
False Negative (FN)	True Negative (TN)

Equations:

$$\text{False positive rate (FPR)} = \frac{FP}{FP+TN}$$

$$\text{False negative rate (FNR)} = \frac{FN}{FN+TP}$$

$$\text{Sensitivity} = \frac{TP}{TP+FN}$$

$$\text{Specificity} = \frac{TN}{TN+FP}$$

$$\text{Youden index} = \text{Sensitivity} + \text{Specificity} - 1$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Content

1. **Data Collection**
2. **EDA**
3. **Data Preparation**
4. **Methodology**
    - Logistic-Regression
    - SVM
    - Ensemble Method
    - Neural Network
5. **Discussion**
6. **Conclusion**

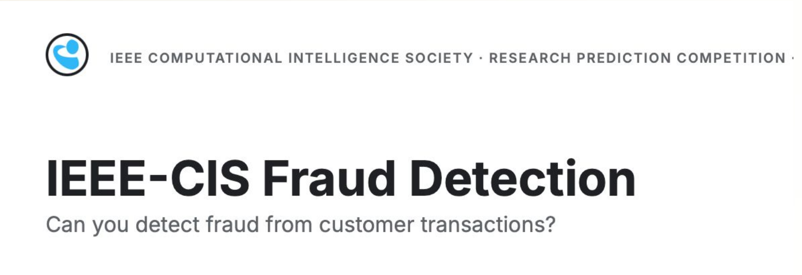
Data Collection

• Dataset Source: Kaggle competition 'IEEE-CIS Fraud Detection', a collaboration between IEEE-CIS and Vesta Corporation.

• Dataset Size & Format: Large dataset derived from real-world e-commerce transactions, including features like device type and product characteristics.

• Scope: Improve fraud detection accuracy; reduce false positives to optimize fraud losses.

• Challenges: Heavy class imbalance typical in fraud detection keeping high false positives.



EDA

Data Overview

432 features【40(identity)+392(transaction)】

- **403** features after dropping high missing value column
- **2278** features after encoding.

1 outcome variable [“isFraud”]

- 569877 Negative
- 20663 Positive

Missing Value

After Review the percentage of missing value of each features.

Null Distribution are roughly shown as below:

100 %: 21 (features)

96 - 97%: 9

68 - 80%: 6

40 - 60%: 151

2 - 18%: 133

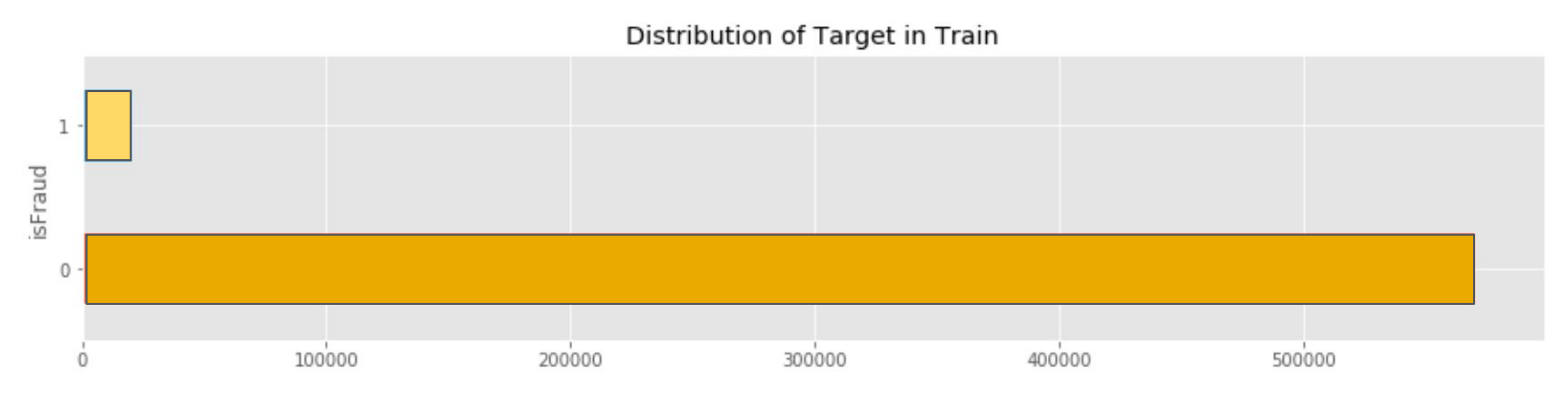
0 - 1%: 92

0: 22 (including outcome variable)

Imbalance Data

Imbalance:

- Train set fraud ratio: 7.48%
- Validation set fraud ratio: 9.33%
- After Standardization, applied SMOTE to deal with the class imbalance by oversampling



Data Preparation Overview

Step 1

✨Missing Value Handling
- Drop columns
- Numerical
- Categorical

Step 2

✨Feature Scaling:
- Standardization of continuous variables

Step 3

✨Class Imbalance Solution:
- SMOTE-NC

Step 4

✨Feature Encoding:
- One-hot encoding

Step 1: Missing Value Handling

• Columns with >80% missing values dropped to prevent excessive noise.

• Numerical Features:
  - Before: Skewed distributions with NaNs.
  - After: Median imputation smoothened the distributions.

• Categorical Features:
  - Before: NaNs leading to incomplete categories.
  - After: Mode imputation preserved feature consistency.

Step 3: Addressing Class Imbalance

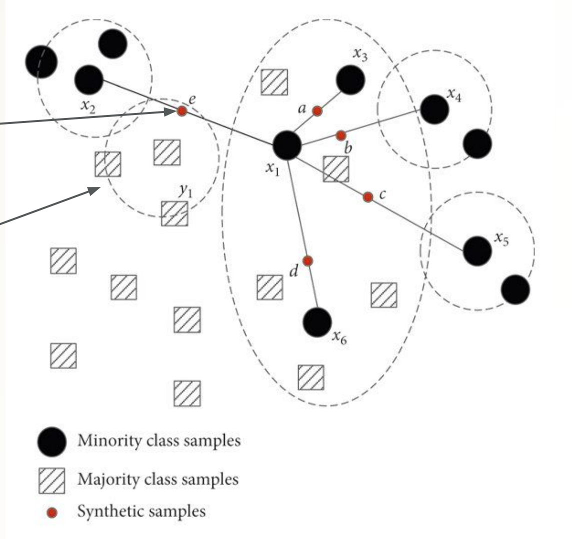
Method: SMOTE-NC (Synthetic Minority Over-sampling Technique for Nominal and Continuous features)

Synthetic Sample Generation:

$$\mathbf{x}' = \mathbf{x} + \lambda(\mathbf{x}_{nn} - \mathbf{x})$$

Handling Categorical Features:

- Instead of interpolation, SMOTE-NC selects the **most frequent (mode)** value of the nominal feature among the **k-nearest neighbors**.



Methodology

Linear Models	Ensemble Methods	Kernel/Nonlinear

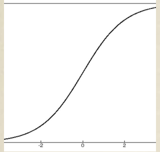
• Logistic Regression	• Bagging
	  ○ Random Forest
	• Boosting
	  ○ LightGBM,
	  ○ XGBoost,
	  ○ CatBoost
		• SVM
		  ○ RBF SVM
		  ○ Polynomial SVM
		• Neural Networks

Logistic Regression

Why Choose Logistic Regression?

Why the simplest?

- Interpretability
- Computational Efficiency
- Regulatory Issue
- Binary Classification



Assumptions: Likelihood maximization

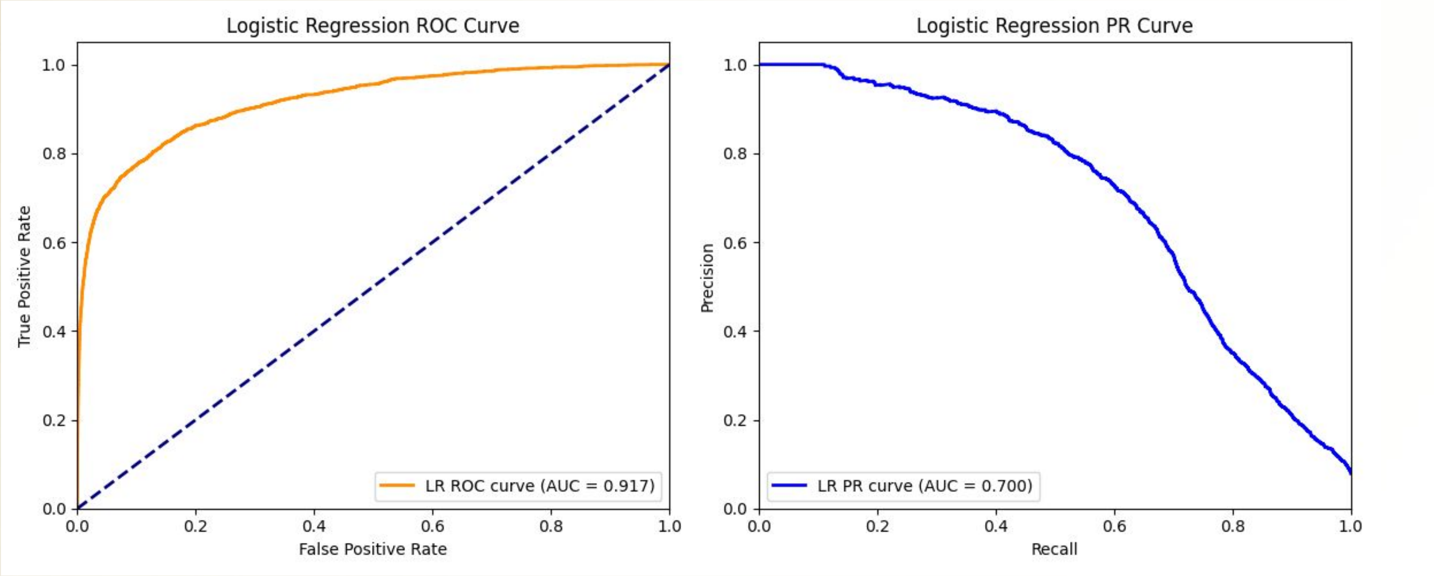
- P(Y|X)

Tuning

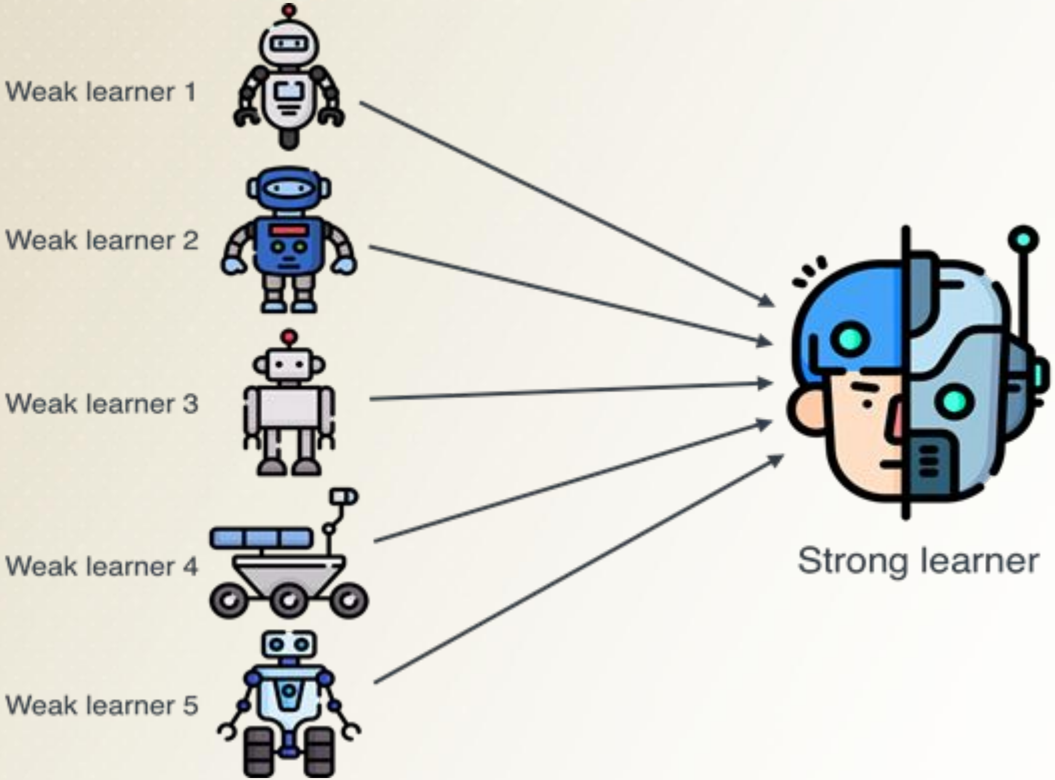
- Regularization
- Encoding

Logistic Regression - Result

- Excellent ROC (0.917)
- Mid PR (0.61 - 0.70 after categorical SMOTE)

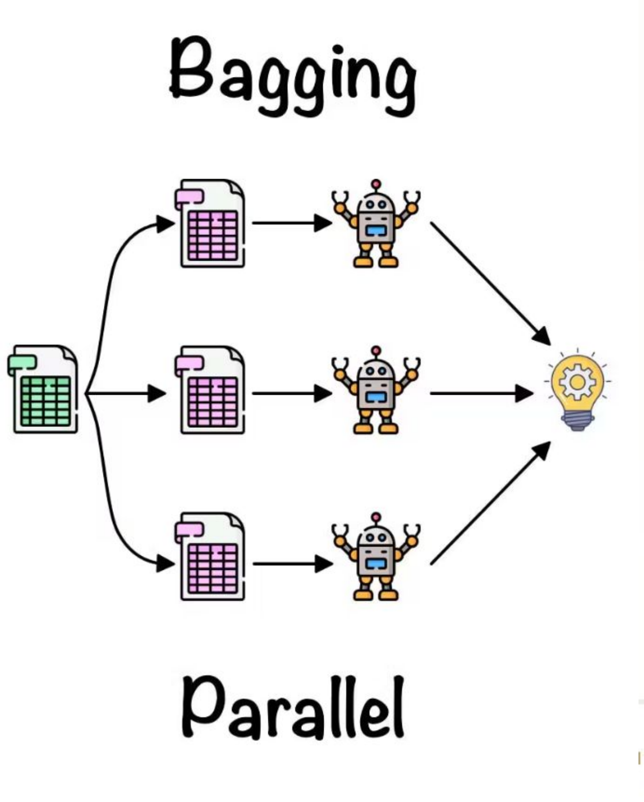


Ensemble Methods



Why Baging?

- Reduce overfitting
- Reduce the impact of outliers or noisy data points



Random Forest - Methodology

1. **Create Bootstrap Samples**
   Randomly sample data points with replacement (bootstrap)

2. **Train Decision Tree**
   Each Tree use a subset of features and bootstrap samples.(sprt, 0.5). Tree grow independently.

3. **Aggregate Prediction**
   Majority Voting

Random Forest - cont.

Why does it work well?

- “Teams” of Decision Trees

Assumptions

- Non Parametric
- Works well with mixed features
- Provide Feature Importance

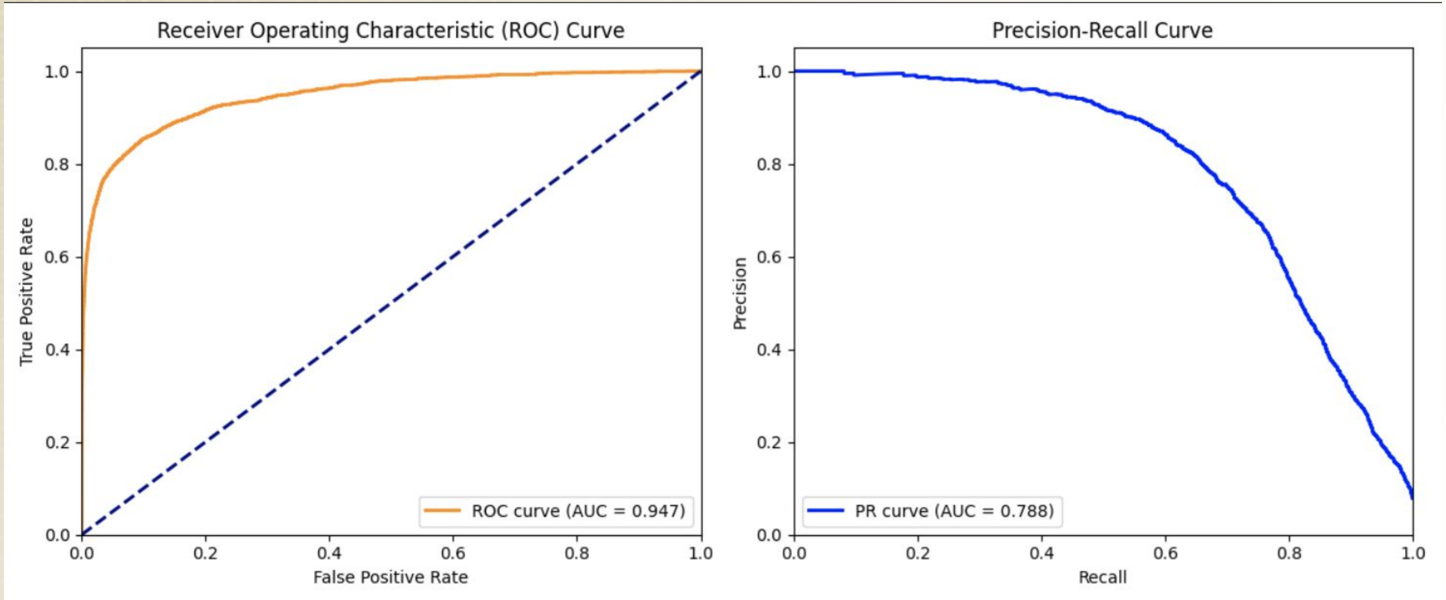
Tuning

```
(criterion='entropy', max_features='sqrt',
max_samples=0.5, min_samples_split=80)
```

Downsides

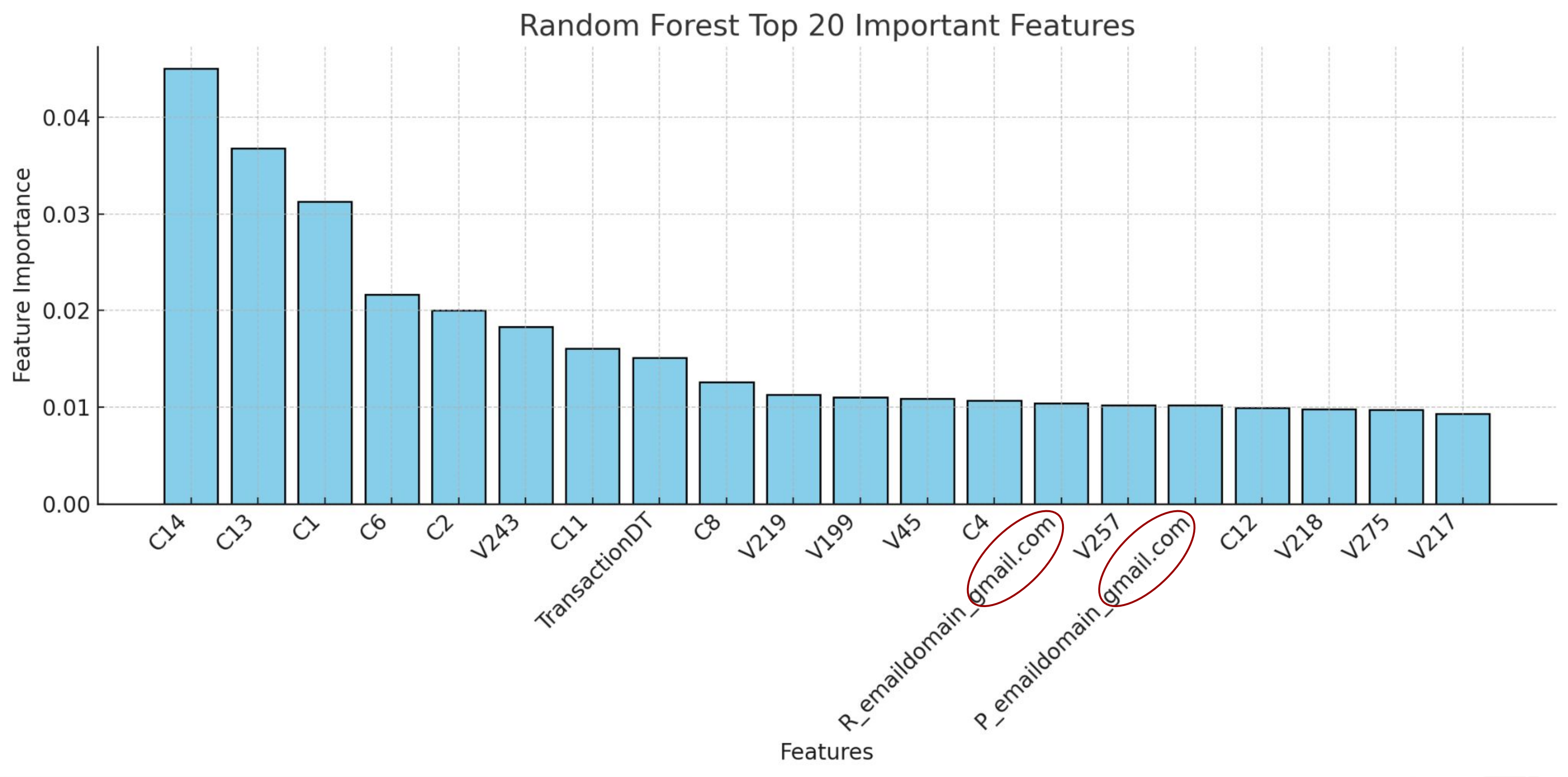
- “Black-box” Characteristics
- Trade-off between Data integrity and Computational force

Random Forest - Result



- Excellent ROC (0.947)
- Mid PR (0.788 w. SMOTE)

Important Features cont.

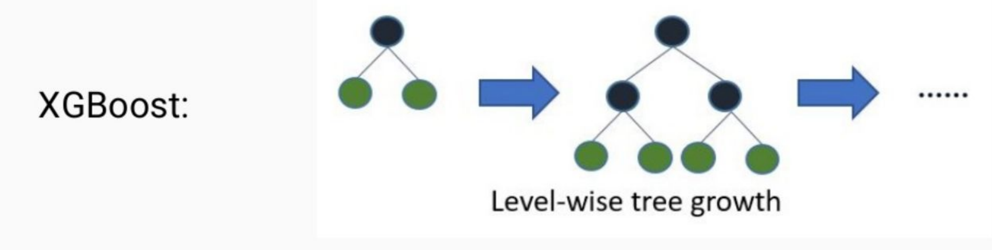


Why Boosting?

Advantages:

- **Handles Complex Relationships**
- **Improves Prediction Accuracy**
- **Reduces Bias and Variance**
- **Adaptable to Different Data Types**
- **Scalable and Efficient**

XGBoost
(EXtreme Gradient Boost)



1. Handling of Categorical Data

This is achieved through **partition-based splits**, where the algorithm determines the optimal way to divide categories into subsets to maximize **information gain.**

This approach reduces the need for extensive preprocessing and can lead to better performance by preserving the inherent relationships within categorical features.

2. Tree Splitting Method

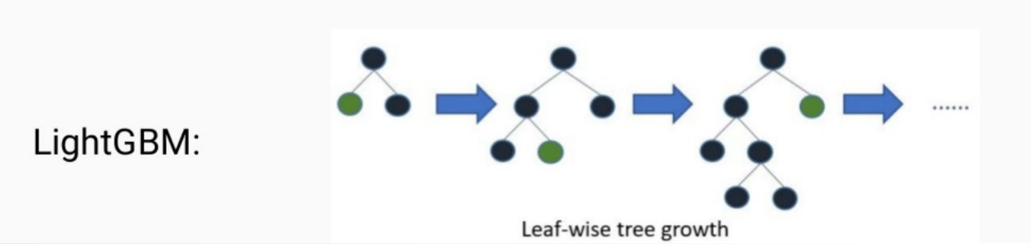
XGBoost constructs trees using a **level-wise** growth strategy, meaning it expands the tree level by level.

At each level, it evaluates all possible splits across all leaves and selects the best splits to add to the tree.

The level-wise method in XGBoost helps maintain balanced trees, which can be beneficial for certain datasets and helps in controlling overfitting.

LightGBM

Light Gradient Boosting Machine



1. Handling of Categorical Data

LightGBM can handle categorical features directly. By specifying which features are categorical, LightGBM applies a specialized algorithm to find the optimal split points for these features, often leading to better performance compared to traditional **one-hot encoding methods**.

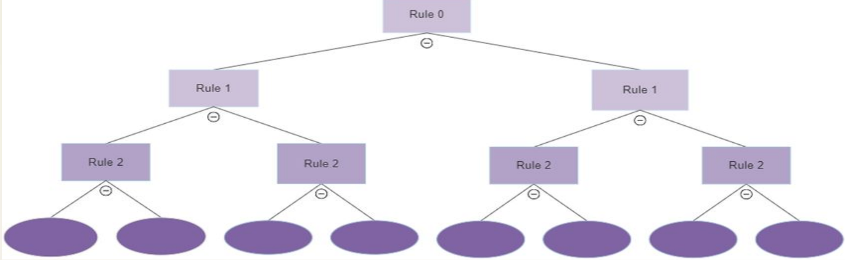
2. Tree Splitting Method

In contrast to XGBoost, LightGBM uses a **leaf-wise** growth strategy, splitting the leaf with the maximum loss reduction (gain).

This approach can lead to **deeper trees and better accuracy**. However, it may also result in overfitting on smaller datasets, so parameters like max_depth are used to control tree depth.

CatBoost

Categorical Boosting



1. Handling of Categorical Data

CatBoost excels in processing categorical features without requiring extensive preprocessing. It employs a technique called **Ordered Target Statistics** to convert categorical data into numerical values during training.

This method calculates statistics (such as mean target values) for each category in a way that prevents data leakage and reduces overfitting. By integrating this approach within the training process, it maintains the **natural ordering and relationships of categorical features, leading to improved model accuracy.**

2.Tree Splitting Method

CatBoost employs a unique tree-building strategy known as **Symmetric Trees**. In this approach, all nodes at the same depth level in the tree are split using the same feature and threshold.

This symmetry simplifies the model and leads to faster predictions, as the structure of the tree is more regular.

Additionally, symmetric trees help in reducing overfitting and improve the model's generalization capabilities.

XGBoost

Loss function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \underbrace{\boxed{\gamma T}}_{\text{Pruning}} + \underbrace{\boxed{\frac{1}{2}\lambda\|w\|^2}}_{\text{Ridge Regression}}$$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

Second order Taylor expansion

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

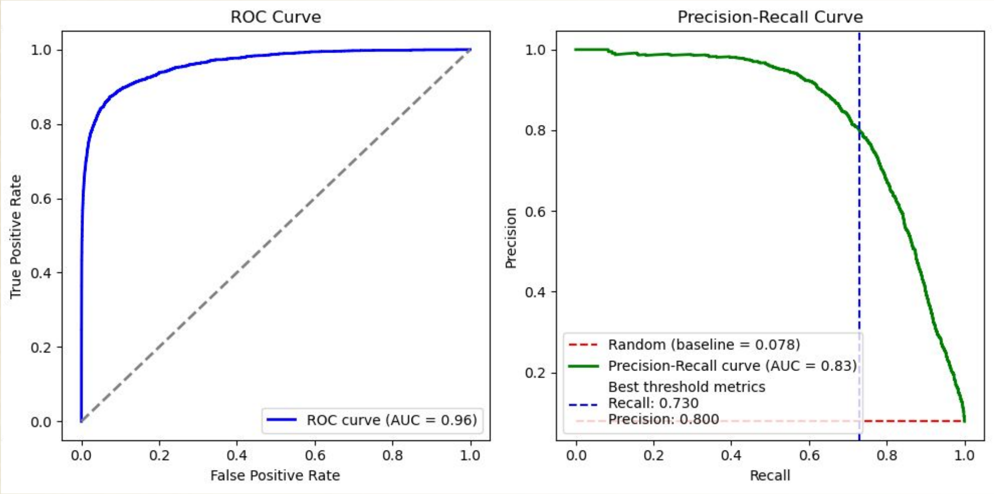
$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

Output Value of leaf *j*

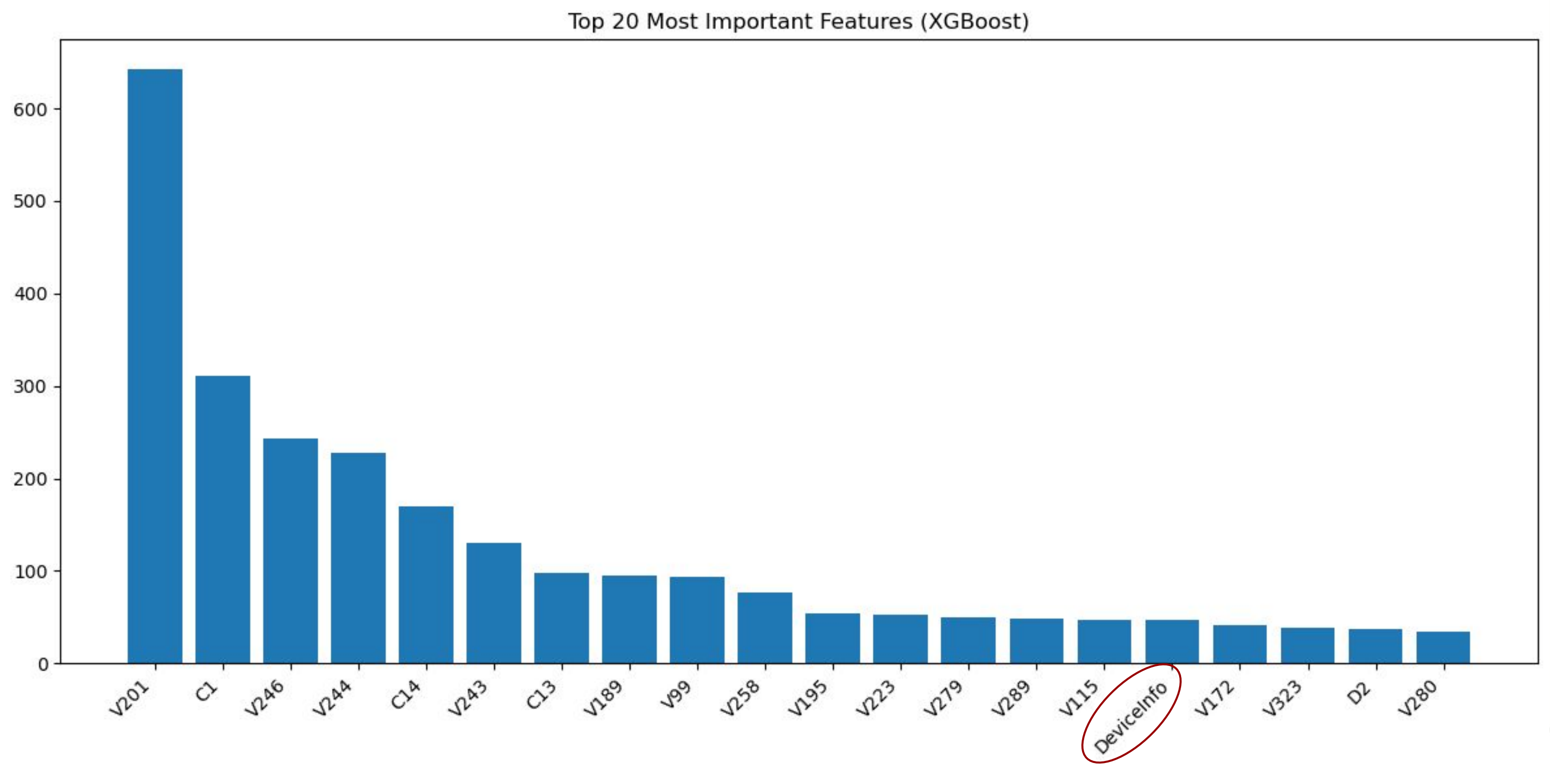
Scores of tree *q* (Impurity score)

XGBoost

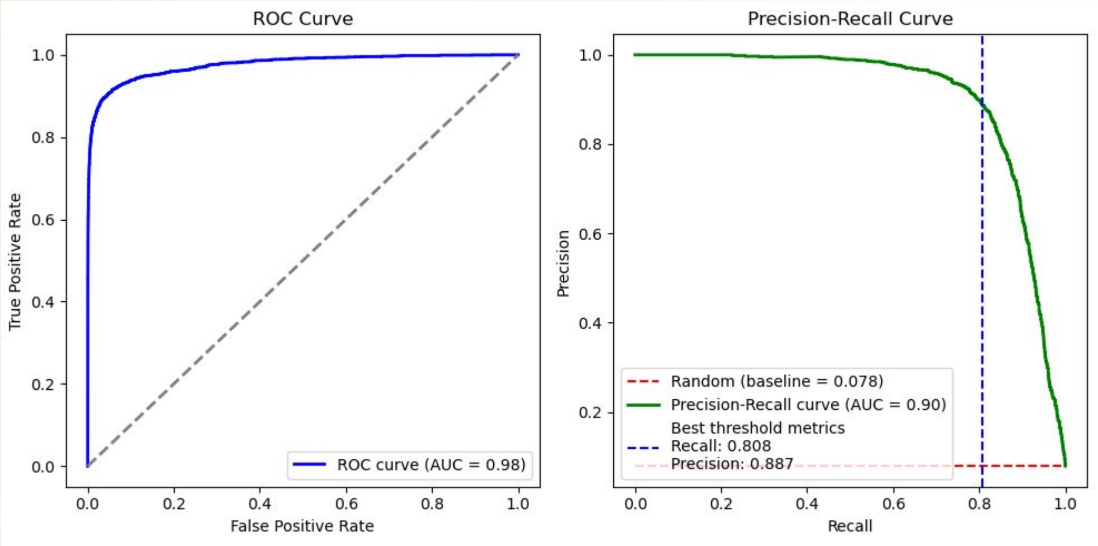


	Accuracy	Best F1-score	AUC PR	AUC ROC	Running Time
1.Preprocessed Data	**0.965**	**0.763**	**0.83**	**0.96**	**19.1s**
2.Oversampled Data	0.964	0.752	0.82	0.95	33.5s

XGBoost

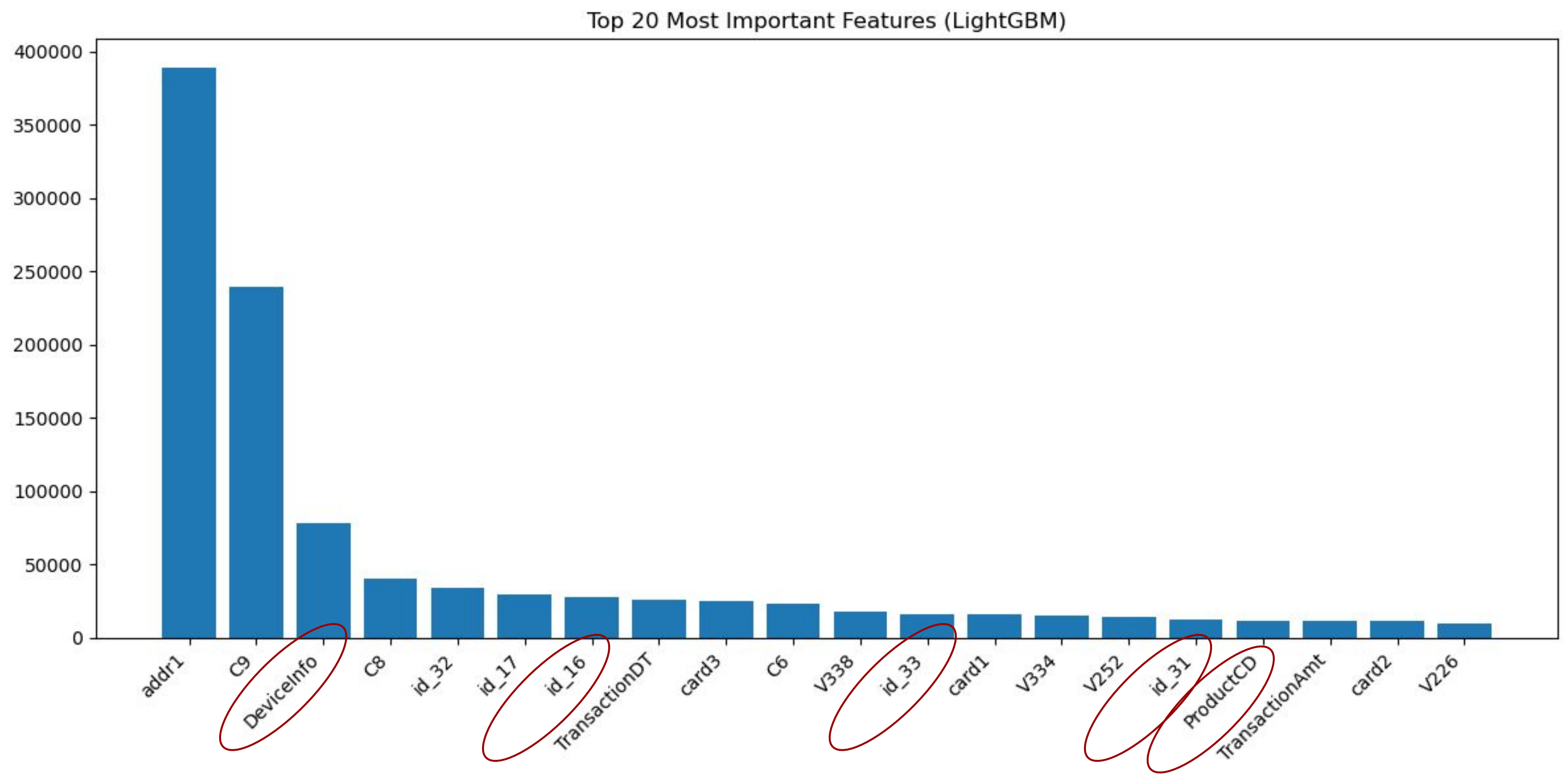

Top 20 Most Important Features (XGBoost)
600
500
400
300
200
100
0
V201
C1
V246
V244
C14
V243
C13
V189
V99
V258
V195
V223
V279
V289
V115
DeviceInfo
V172
V323
D2
V280

LightGBM



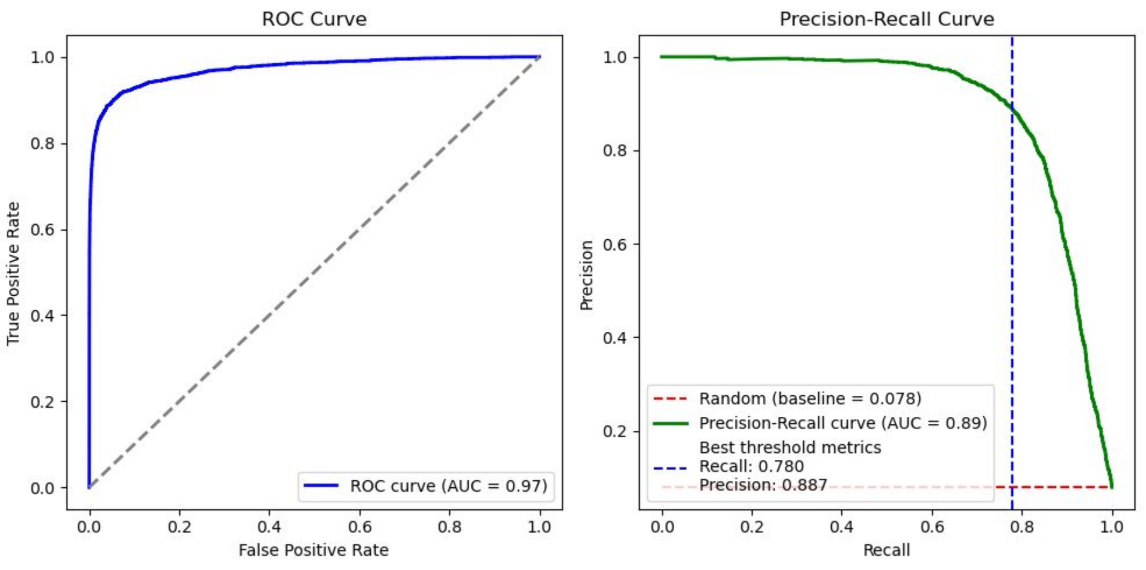
	Accuracy	Best F1-score	AUC PR	AUC ROC	Running Time
1.Preprocessed Data	**0.977**	**0.842**	**0.90**	**0.98**	**11.3s**
2.Oversampled Data	0.973	0.820	0.88	0.97	29.6s

LightGBM



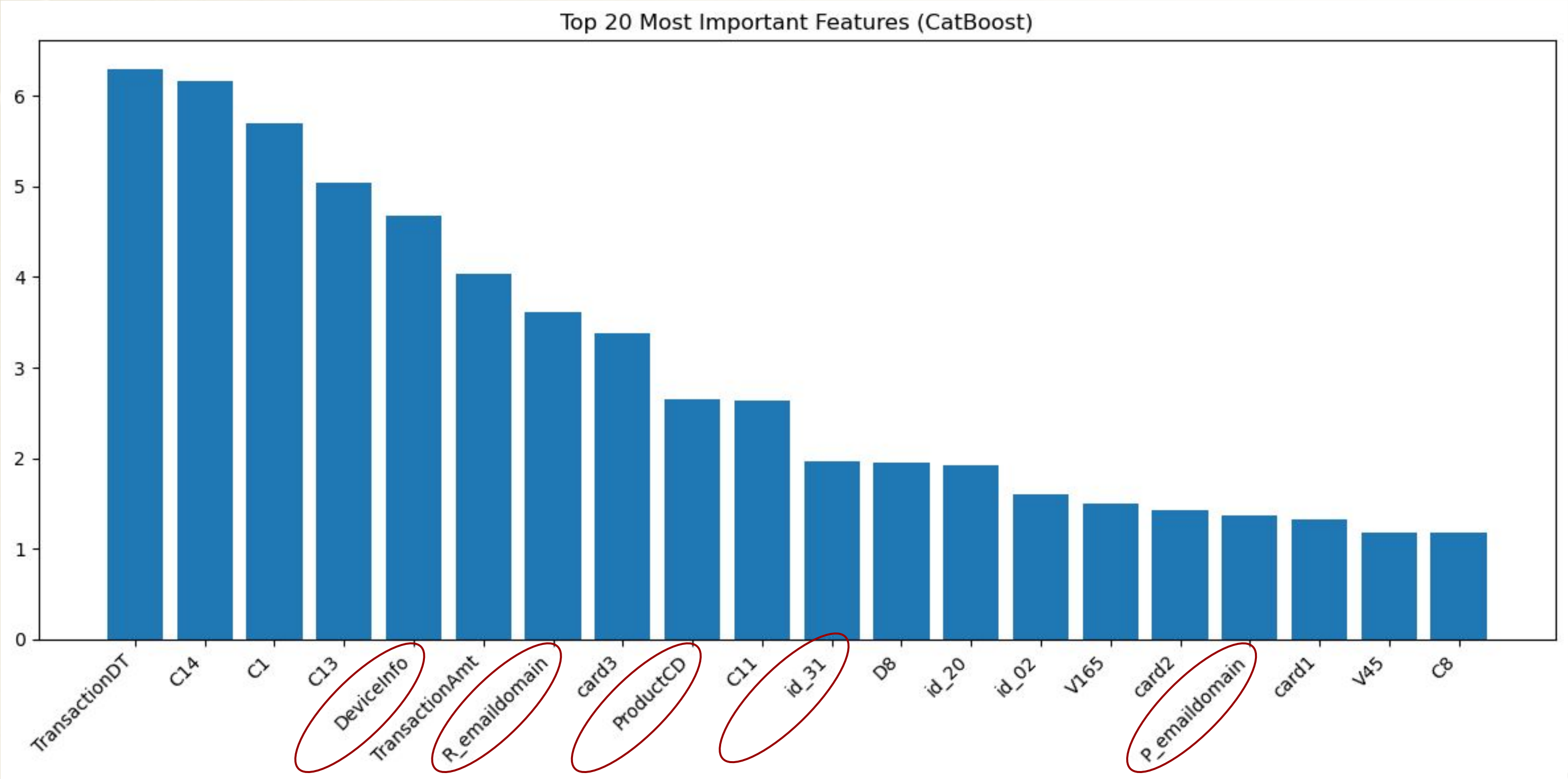
Top 20 Most Important Features (LightGBM)
400000
350000
300000
250000
200000
150000
100000
50000
0
addr1
C9
DeviceInfo
C8
id_32
id_17
id_16
TransactionDT
card3
C6
V338
id_33
card1
V334
V252
id_31
ProductCD
TransactionAmt
card2
V226

CatBoost



	Accuracy	Best F1-score	AUC PR	AUC ROC	Running Time
1.Preprocessed Data	**0.974**	**0.830**	**0.89**	**0.972**	**55.2s**
2.Oversampled Data	0.971	0.813	0.87	0.968	97.6s

CatBoost



Top 20 Most Important Features (CatBoost)
0
1
2
3
4
5
6
TransactionDT
C14
C1
C13
DeviceInfo
TransactionAmt
R_emaildomain
card3
ProductCD
C11
id_31
D8
id_20
id_02
V165
card2
P_emaildomain
card1
V45
C8

Experimental Results

1. LightGBM performs best.
2. Oversample does **not** provide helpful information for boosting model training.

	Accuracy	Best F1-score	AUC PR	AUC ROC	Running Time
XGBoost	0.965	0.763	0.83	0.96	19.1s
LightGBM	**0.976**	**0.842**	**0.90**	**0.98**	**11.3s**
CatBoost	0.974	0.830	0.89	0.97	55.2s

Discussion

- Categorical features are important in classifying the fraud transaction

  The number of categorical features in top 20 most important features (245): Random Forest (2), XGBoost (1), LightGBM(**5**), CatBoost(**5**)

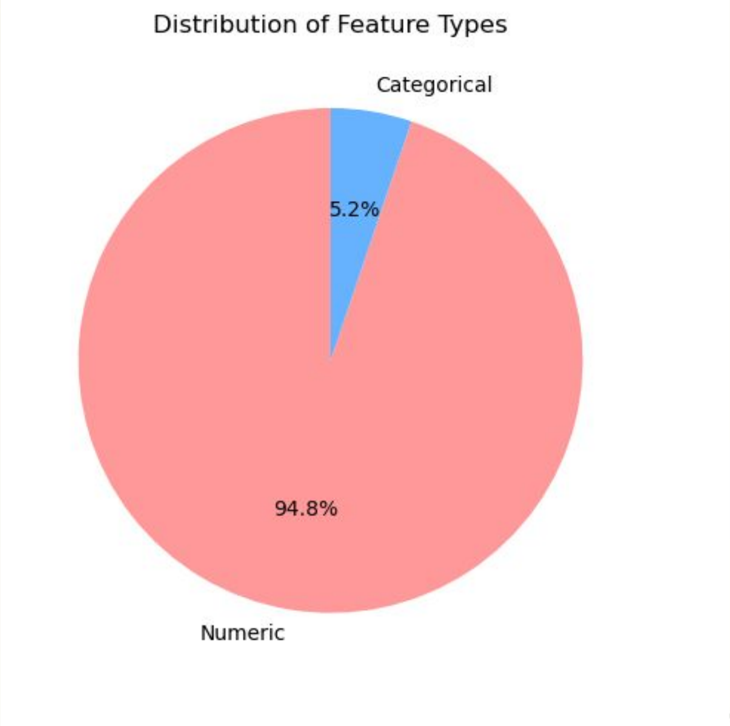
- The original dataset is imbalanced, but due to the large number of samples, boosting models can effectively learn the relationship between transactions and fraud. However, oversampling introduces noise into the data, which boosting models tend to fit, ultimately compromising their performance.

Discussion

Low Proportion of Categorical Features

- The dataset contains **few categorical features (less than 10%)** and is dominated by numerical features, so LightGBM's efficiency in handling numerical data gives it an edge.

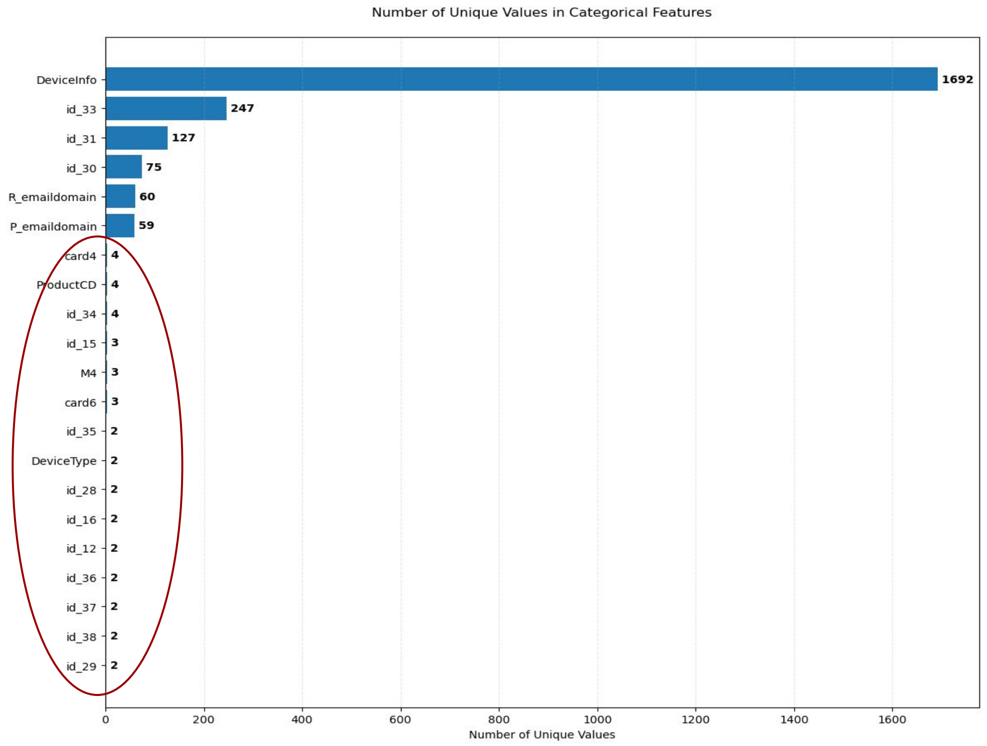
- CatBoost's key advantage lies in its advanced handling of categorical features, so when categorical features are **sparse**, this advantage becomes less impactful.



Discussion

Category Cardinality

- If the categorical features have **low cardinality**, LightGBM's processing can yield comparable results to CatBoost without requiring its specialized encoding.



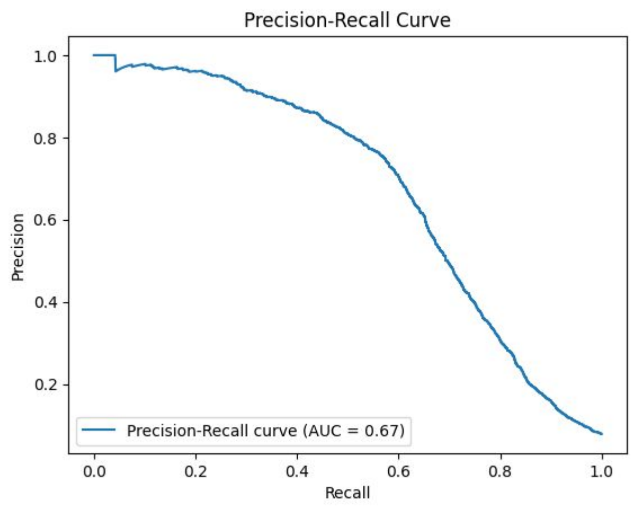
Support Vector Machines(SVMs)

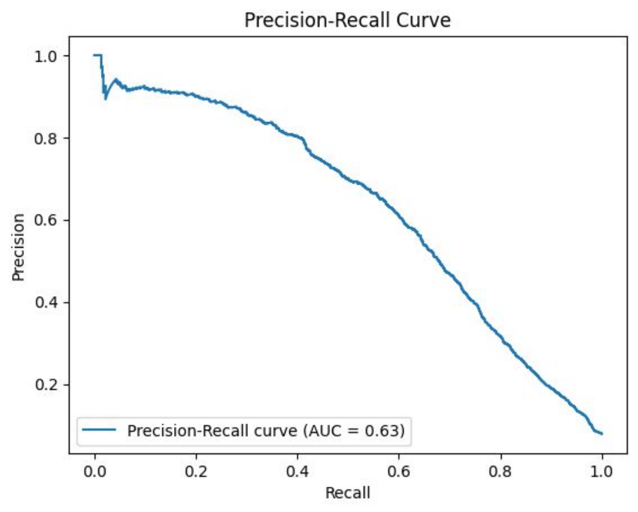
- Radial Basis Function(RBF) SVM
- Polynomial Kernel SVM

RBF Kernel SVM

- Training Model
  - Selected 20% data from train dataset as sample, then PCA was applied to reduce dimensionality while retaining 90% variance.
  - The same process applied to two data sets
    - Preprocessed Data
    - Oversampled Data
  - Grid Search, best model setting C = 100, gamma = 0.001

RBF Kernel SVM





	Accuracy	F1-score	AUC PR	AUC ROC	Running Time
1.Preprocessed Data	**0.9505**	**0.5908**	**0.6733**	**0.8886**	**2m 30s**
2.Oversampled Data	0.8720	0.4857	0.6280	0.9016	15m

Georgia Tech

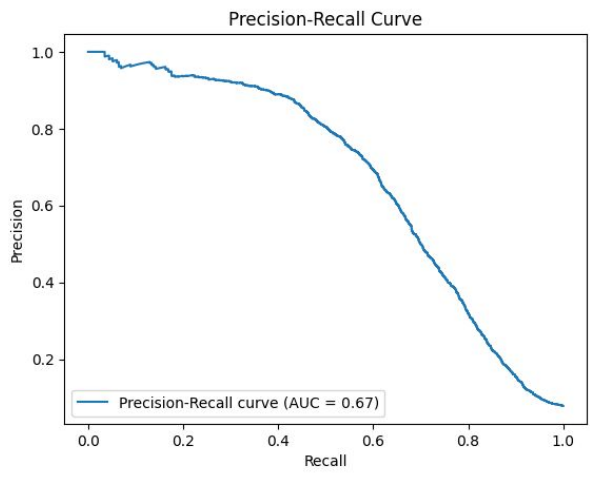
RBF Kernel SVM

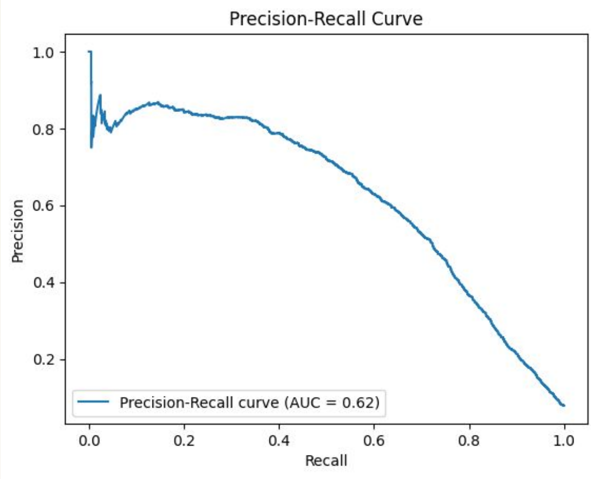
- Performance Evaluation
  - Preprocessed Data:
    - Higher overall accuracy and precision for fraud detection. Poor recall for fraud cases, missing more fraudulent transactions.
  - Oversampled Data:
    - Improved recall for fraud (77% vs. 46%) but at the cost of significantly lower precision. Lower F1 score for fraud indicates the trade-off between precision and recall is less favorable.
  - Both models face difficulties balancing precision and recall due to the highly imbalanced nature of the dataset.

Polynomial Kernel SVM

- Training Model
  - Selected 20% data from train dataset as sample (34026, 2680), then PCA was applied to reduce dimensionality while retaining 90% variance.
  - The same process applied to two data sets
    - Preprocessed Data
    - Oversampled Data
  - Grid Search, best model setting C = 10, degree = 3, gamma = scale, coef0 = 1

Polynomial Kernel SVM





	Accuracy	F1-score	AUC PR	AUC ROC	Running Time
1.Preprocessed Data	**0.9513**	**0.6303**	**0.6733**	**0.8861**	**1m 30s**
2.Oversampled Data	0.8824	0.5137	0.6236	0.9110	10m 20s

Georgia Tech

Polynomial Kernel SVM

- Performance Evaluation
  - Preprocessed Data:
    - Higher overall accuracy, precision, and F1 score for fraud detection. Low recall (53%) indicates many fraud cases are missed.
  - Oversampled Data:
    - Improved recall (79%) for fraud but at the expense of significantly lower precision (38%). Lower accuracy (88.2%) and F1 score (0.5137) for fraud detection due to an increase in false positives.
  - Polynomial SVMs, especially with higher degrees, are prone to overfitting, particularly when the dataset contains synthetic features or noise (as introduced by SMOTE).

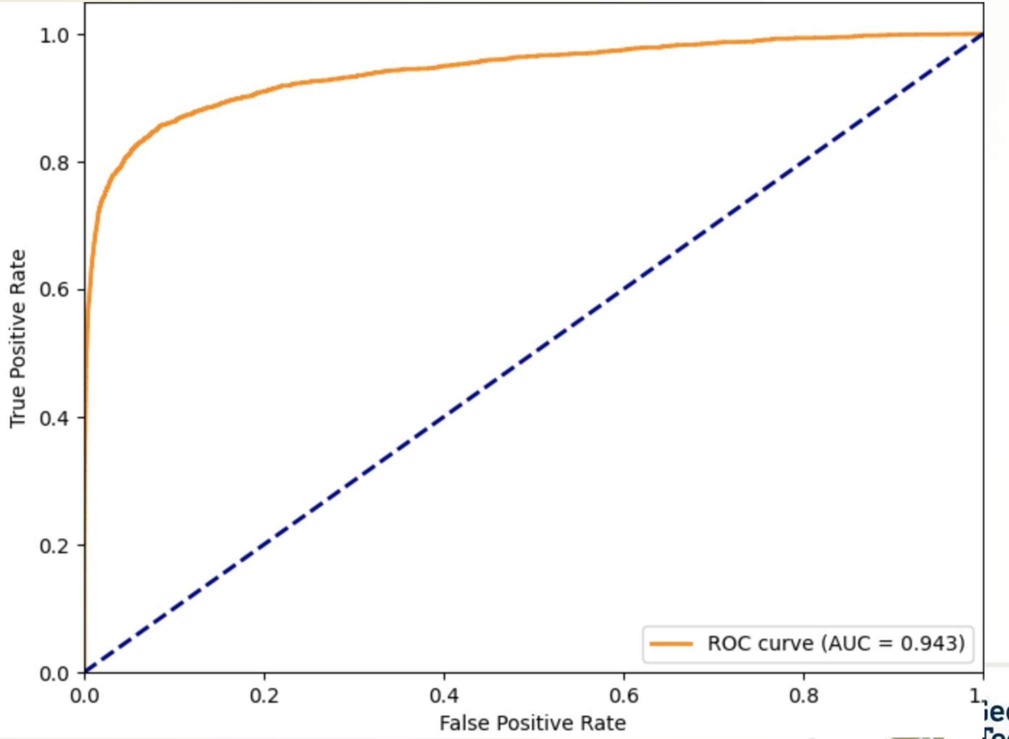
Neural Networks

```python
# Training configurations
class Config:
    # Model parameters
    model_architecture = {
        'hidden_layers': [128, 64, 32, 16],
        'dropout_rate': 0.01,
    }

    # Training parameters
    num_epochs = 50
    steps_per_epoch = 200
    batch_size = 32
    learning_rate = 0.001
    fraud_ratio = 0.5  # For balanced batches

    # Encoder parameters
    encoder_params = {
        'onehot_threshold': 10,
        'outlier_threshold': -999,
        'std_threshold': 5,
        'cache_dir': 'encoder_cache',
        'force_recompute': False
    }
```

AUC: 0.943



Neural Networks

- **Hidden Layers**: Configured with dimensions [128, 64, 32, 16], progressively reducing complexity while retaining the ability to learn deep representations.
- **Activation Function**: Each layer uses ReLU (Rectified Linear Unit) activation to introduce non-linearity, enabling the model to learn complex patterns.
- **Regularization**: A dropout layer with a rate of 0.01 is applied after each hidden layer to mitigate overfitting by randomly deactivating neurons during training.
- **Output Layer**: A single neuron with no activation directly outputs logits, suitable for binary classification tasks when paired with a sigmoid function during evaluation.

Neural Networks

Checkpoint saved at epoch 40

True Positive Rate: 0.7961

True Negative Rate: 0.9435

False Positive Rate: 0.0565

False Negative Rate: 0.2039

Precision: 0.5454

F1 Score: 0.6473

Neural Networks

The model is trained using binary **cross-entropy loss** with logits, which combines a sigmoid activation with cross-entropy for stable binary classification. **The Adam optimizer**, with a learning rate of 0.001, adapts learning rates for efficient convergence. Training is conducted over 50 epochs with 200 steps per epoch, ensuring adequate exposure to patterns in the dataset. **Balanced batch sampling**, with a fraud ratio of 50\%, addresses class imbalance by including sufficient minority class instances during training. To enhance generalization, **dropout regularization** (rate 0.01) mitigates overfitting by randomly deactivating neurons.

The training process incorporates periodic model checkpointing for resumability and generates validation metrics and visualizations, such as AUC history and ROC curves, to monitor progress and ensure robust evaluation. These features make the model both effective and straightforward to extend or adapt for fraud detection tasks.

Discussion

Model Performance Comparison

- Model with High Complexity tends to capture the trend in this high dynamic dataset
  - The Threshold of a good model is set as 0.8 to ensure bearable “false alarm cost”

	Log Reg	SVM	RFC	Boost	NN
ROC-AUC	.917	0.8886	.947	.980	.943
PR-AUC	.700	0.6733	.788	.900	.809

Questions?
&
Thank you!