澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

Faculty of Business Administration

ISOM7031 - MACHINE LEARNING WITH BUSINESS APPLICATIONS

Group Project Report

(First Semester, 2024/2025)

Theme - Credit Card Fraud Detection in Machine Learning

Group member

(1) Chinese Name: 陳慧藍 English Name: Huilan Chen Student ID: MC439936

(2) Chinese Name: 林靜 English Name: Jing Lin Student ID: MC439764

(3) Chinese Name: 談霈琳 English Name: Peilin Tan Student ID: MC439755

(4) Chinese Name: 周忻如 English Name: Xinru Zhou Student ID: MC439963

Contents

Credit Card Fraud Detection in Machine Learning

Abstract

This project addresses the challenge of detecting fraudulent activities in an imbalanced dataset of credit card transactions. The dataset, comprising 284,807 transactions by European cardholders over a two-day period in 2013, includes only 492 fraudulent transactions, representing a mere 0.172% of the total. To tackle this imbalance, we implemented and compared various machine learning algorithms, including Logistic Regression, Linear SVM, SVM with RBF Kernel, Random Forest, Neural Networks, and XGBoost. We employed techniques such as SMOTE for oversampling, random under-sampling, and algorithmic adjustments of class weights to handle the class imbalance. Additionally, extensive hyperparameter tuning was performed for models like Neural Networks and XGBoost to optimize their performance. The methodologies developed in this project are not only applicable to credit card fraud detection but can also be extended to other domains facing similar challenges, such as healthcare, cybersecurity, and manufacturing failure prediction. The results demonstrate the effectiveness of these approaches in enhancing the accuracy and robustness of fraud detection models.

Keywords: Fraud Detection, Imbalanced Dataset, Credit Card Transactions, Logistic Regression, Support Vector Machine (SVM), Random Forest, Neural Networks, XGBoost, SMOTE, Hyperparameter Tuning

I. Introduction

This project focuses on detecting fraudulent activities in an imbalanced dataset of credit card transactions. The dataset records transactions by European cardholders over a two-day period in 2013, comprising 284,807 transactions, of which only 492 were flagged as fraudulent. This significant imbalance, with fraudulent transactions representing just 0.172% of the total, poses a unique challenge for developing effective fraud detection models.

II. Goals and Innovation point

The primary goal of this project is to showcase and compare different methodologies for fraud detection in imbalanced datasets. These methodologies are not only applicable to credit card fraud detection but can also be extended to other domains facing similar challenges, such as healthcare, cybersecurity, and manufacturing failure prediction.

Innovative Points:

Comprehensive Model Comparison: We implemented and compared several machine learning algorithms, including Logistic Regression, Linear SVM, SVM with RBF Kernel, Random Forest, Neural Networks, and XGBoost. This comprehensive comparison helps identify the most effective models for fraud detection.

Handling Imbalance: To address the class imbalance, we employed various techniques:

- **Oversampling**: Using SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic samples for the minority class.
- **Under-sampling**: Randomly reducing the number of samples in the majority class.
- **Algorithmic Approaches**: Adjusting class weights within algorithms to give more importance to the minority class.

Hyperparameter Tuning: For models like Neural Networks and XGBoost, we performed extensive hyperparameter tuning to achieve optimal performance. This demonstrates our commitment to fine-tuning and optimizing models for better accuracy and robustness in detecting fraudulent transactions.

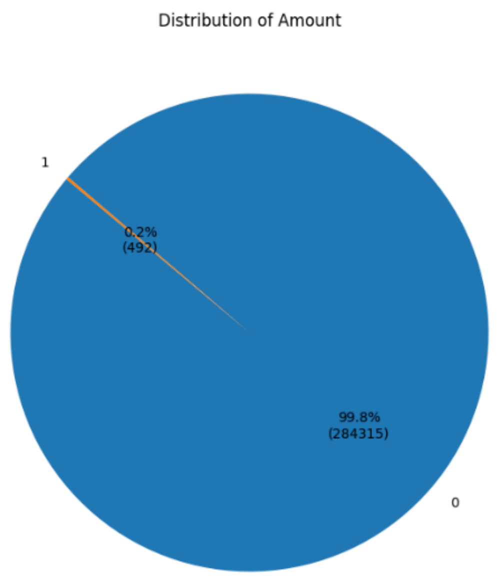
By addressing these goals and incorporating innovative techniques, this project aims to develop a robust machine learning model capable of accurately detecting fraudulent credit card transactions, despite the significant class imbalance. The methodologies and findings from this project can contribute to enhancing financial security and can be adapted to other fields facing similar challenges.

III. Understanding our data

1. Description:

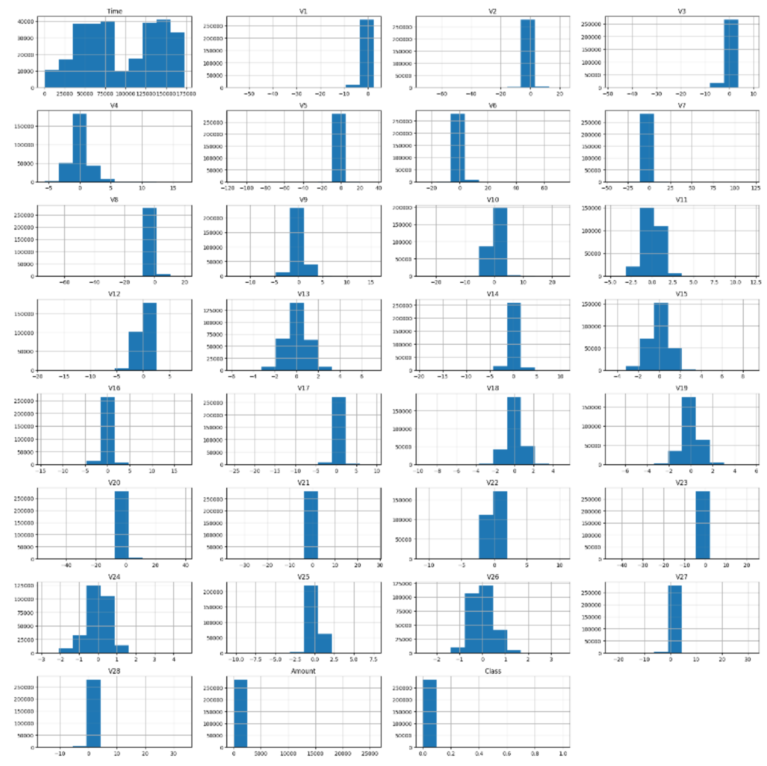
The dataset records credit card transactions by European cardholders in 2013, covering transactions that occurred over a two-day period. Out of a total of 284,807 transactions, 492 were flagged as fraudulent. The dataset is significantly unbalanced as fraudulent transactions represent only 0.172% of all transactions.

Figure 1.1 Distribution Pie Chart of the 2013 Credit Card Transactions dataset



2.Key Features:

Figure 1.2 Histogram matrix of Distribution of different features in the 2023 Credit Card Fraud Transactions dataset



(1) Time: The graph in the upper left corner shows the distribution of the temporal characteristics of the transactions. The horizontal axis is time (in seconds) and the vertical axis is the number of samples. There are differences in the number of trades in different time periods, reflecting the distribution of trading activity over time.

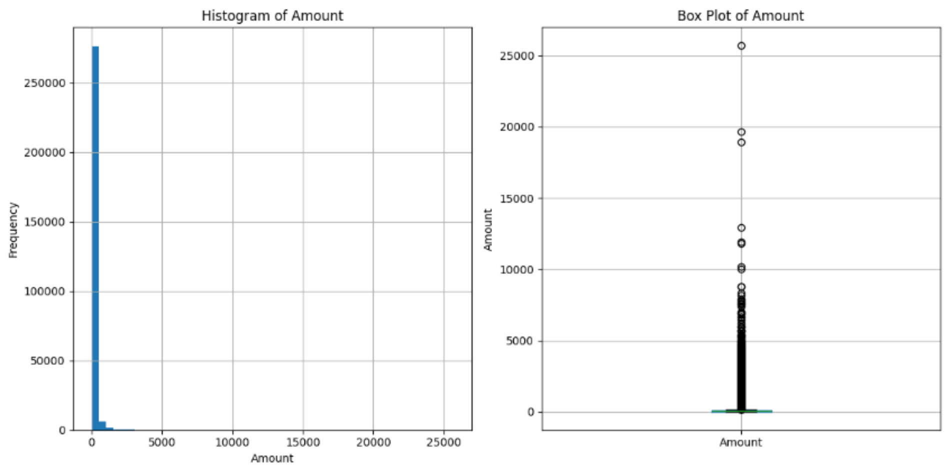
2. PCA converted features (V1 - V28): These features (from V1 to V28) are obtained by Principal Component Analysis (PCA) transformation. Since the dataset is related to user privacy, the original features are transformed into nonspecific principal components. The histogram shows most of the features have a concentrated distribution, mostly centered on zero, with some showing a right-skewed or left-skewed distribution.

3. Label (Class): The bottom right graph shows the distribution of the target variable Class, where a value of 0 indicates a normal transaction and a value of 1 indicates a fraudulent transaction. As can be seen, there are far more samples of normal transactions than fraudulent transactions, and the dataset suffers from a severe class imbalance.

It shows that there are far fewer samples of fraudulent transactions than normal transactions, which implies that there is a high degree of imbalance in the dataset, and special attention needs to be paid to this issue when the model is being trained to avoid the model being biased towards predicting the majority of classes (normal transactions).

4. Amount: The histogram of transaction amount shows that most of the transactions have relatively small amounts and only a few transactions have very large amounts. This feature is an important piece of information used to distinguish normal transactions from fraudulent ones. In Figure 1.3 Points in the box plot where the amount is particularly large may be unusual transactions and maybe need to be further analyzed for fraudulent transactions.

Figure 1.3 Histograms and Box Plots of Transaction Amounts



IV. Approach and Results

1. Logistic Regression

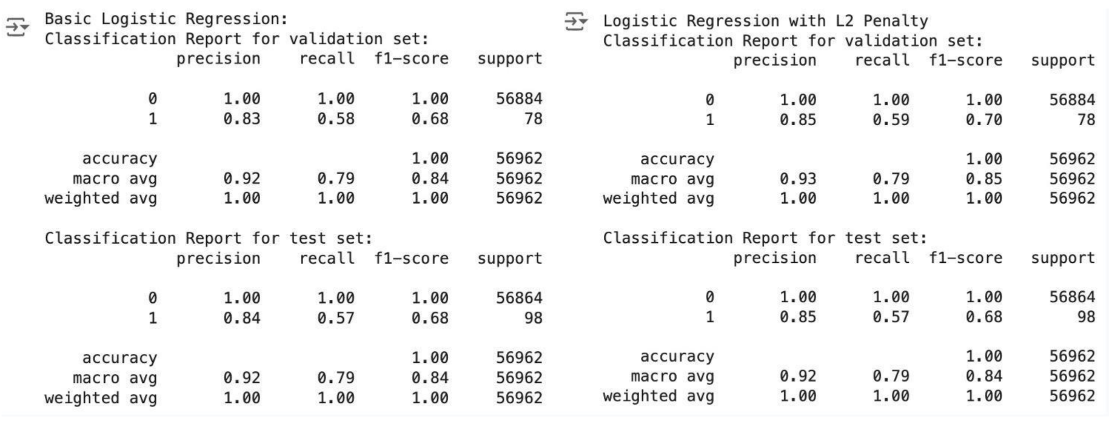
Logistic Regression is a popular statistical method used for binary classification problems, making it a suitable choice for credit card fraud detection. It has several advantages: it is simple to implement and interpret, making it a good starting point for binary classification problems; it is computationally efficient, suitable for large datasets; it provides probabilistic outputs, which can be useful for threshold-based decision making; and it serves as a strong baseline model to compare with more complex models. However, Logistic Regression also has some disadvantages: it assumes a linear relationship between the features, which might not always be true; it can struggle with highly imbalanced datasets, often requiring additional techniques like resampling or adjusting class weights; and it requires careful feature engineering and selection, as it can be sensitive to irrelevant or highly correlated features.

1.1 Basic vs. L2 Regularized Logistic Regression

We first compare basic logistic regression with logistic regression using L2 penalty parameter (penalty='l2'). L2 regularization, or Ridge regression, adds a penalty equal to the sum of the squared coefficients to the loss function.

As shown in Figure 1.1, basic logistic regression provides balanced performance with decent precision and recall. However, it exhibits slightly lower precision and recall compared to the L2 model on the validation set. Logistic regression with L2 penalty demonstrates marginal improvements in precision and recall on the validation set, resulting in a slightly higher F1-score. However, the performance on the test set is almost identical to the basic model. Overall, the L2 regularized model offers slight improvements in precision and recall on the validation set, but both models perform similarly on the test set.

Figure 1.1



```
Basic Logistic Regression:
Classification Report for validation set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.83      0.58      0.68        78

    accuracy                           1.00     56962
   macro avg       0.92      0.79      0.84     56962
weighted avg       1.00      1.00      1.00     56962

Classification Report for test set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.84      0.57      0.68        98

    accuracy                           1.00     56962
   macro avg       0.92      0.79      0.84     56962
weighted avg       1.00      1.00      1.00     56962
```

```
Logistic Regression with L2 Penalty
Classification Report for validation set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.85      0.59      0.70        78

    accuracy                           1.00     56962
   macro avg       0.93      0.79      0.85     56962
weighted avg       1.00      1.00      1.00     56962

Classification Report for test set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.85      0.57      0.68        98

    accuracy                           1.00     56962
   macro avg       0.92      0.79      0.84     56962
weighted avg       1.00      1.00      1.00     56962
```

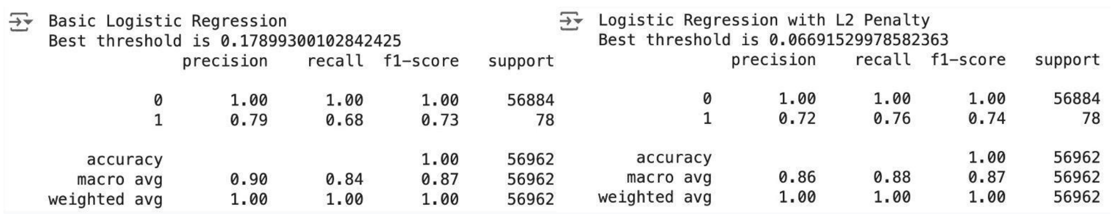
As shown in Figure 1.2, the best threshold for the logistic regression with L2 Penalty is much lower (0.0669) compared to the basic logistic regression (0.1790). This indicates that the L2 regularized model is more sensitive and requires a lower probability to classify an instance as the minority class.

Logistic regression with L2 penalty shows balanced precision (0.72) and recall (0.76) for the minority class, leading to a slightly higher F1-score (0.74). This model is better at identifying true positives while maintaining a reasonable number of false positives. In contrast, Basic Logistic

Regression has higher precision (0.79) but lower recall (0.68) for the minority class, meaning it avoids more false positives but misses more true positives compared to the L2 regularized model.

Both models have similar macro and weighted average F1-scores (0.87 and 1.00, respectively), indicating overall balanced performance. However, the L2 regularized model has a slightly higher F1-score for the minority class, suggesting better performance in identifying minority class instances.

Figure 1.2



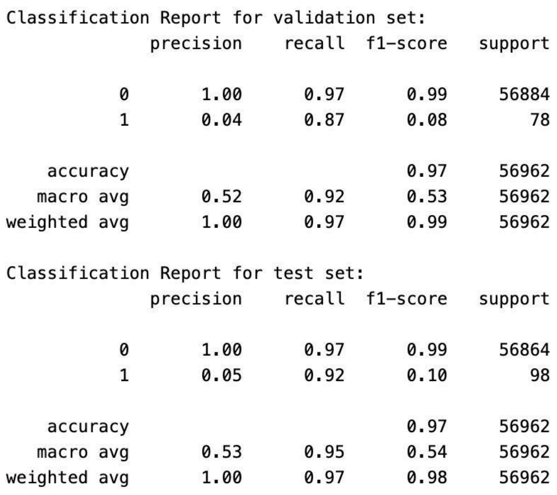
```
Basic Logistic Regression
Best threshold is 0.17899300102842425
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.79      0.68      0.73        78

    accuracy                           1.00     56962
   macro avg       0.90      0.84      0.87     56962
weighted avg       1.00      1.00      1.00     56962
```

```
Logistic Regression with L2 Penalty
Best threshold is 0.06691529978582363
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.72      0.76      0.74        78

    accuracy                           1.00     56962
   macro avg       0.86      0.88      0.87     56962
weighted avg       1.00      1.00      1.00     56962
```

1.2 Balancing Dataset Using SMOTE

The model's overall accuracy (shown in figure 1.3) is very high at 97% for both the validation and test sets, primarily due to its performance on the majority class (Class 0). However, this high accuracy can be misleading because of the significant class imbalance in the dataset.

Figure 1.3 Output of Logistic Regressiong Using SMOTE



```
Classification Report for validation set:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56884
           1       0.04      0.87      0.08        78

    accuracy                           0.97     56962
   macro avg       0.52      0.92      0.53     56962
weighted avg       1.00      0.97      0.99     56962

Classification Report for test set:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56864
           1       0.05      0.92      0.10        98

    accuracy                           0.97     56962
   macro avg       0.53      0.95      0.54     56962
weighted avg       1.00      0.97      0.98     56962
```

The dataset is highly imbalanced, with Class 0 having many more samples than Class 1. This imbalance is evident in the precision and recall scores for Class 1. Despite this, the use of SMOTE

(Synthetic Minority Over-sampling Technique) has significantly improved the recall for the minority class (Class 1), achieving 0.87 for the validation set and 0.92 for the test set. This shows that the model is now better at identifying the minority class.

However, the precision for Class 1 remains very low, at 0.04 for the validation set and 0.05 for the test set, meaning many instances predicted as Class 1 are false positives, shown as figure 1.4. Consequently, the F1-score for Class 1 is also very low, at 0.08 for the validation set and 0.10 for the test set, indicating that the model's performance on the minority class is still not optimal despite the use of SMOTE.

The extreme class imbalance makes it challenging for the model to learn the minority class effectively. While SMOTE improves recall, it may also introduce noise, leading to a high number of false positives, which affects precision. Additionally, logistic regression might not be complex enough to capture the nuances in the data, especially for the minority class.

Figure 1.4

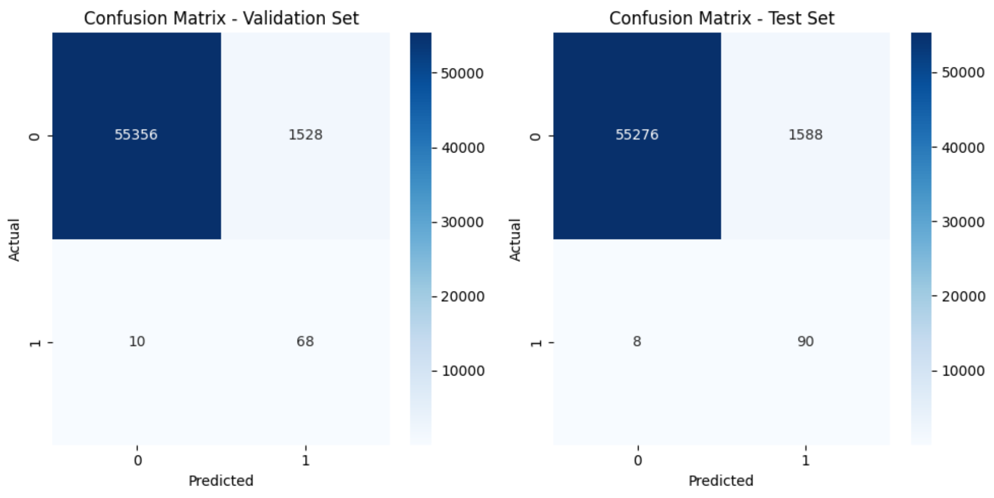
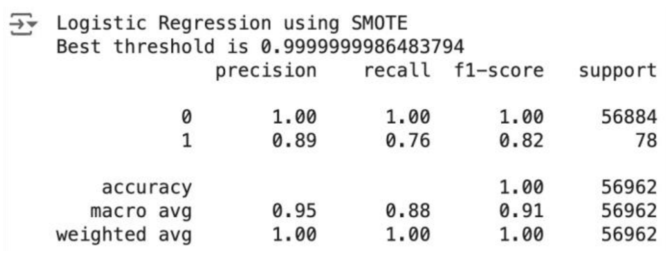


Figure 1.5 Best threshold for logistic regressiong using SMOTE is almost 1



```
Logistic Regression using SMOTE
Best threshold is 0.9999999986483794
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.89      0.76      0.82        78

    accuracy                           1.00     56962
   macro avg       0.95      0.88      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

A high threshold suggests that the model is very conservative in classifying instances as positive. It only classifies an instance as positive if the predicted probability is very high. This means the model might be too tuned to the validation set, detecting almost all positive cases but potentially missing some true positives in new data. The slight drop in recall from 0.87 on the validation set to 0.76 at the best threshold indicates potential overfitting.

Even with L2 regularization, overfitting can occur if the model is too sensitive to the training data, especially in highly imbalanced datasets. L2 regularization helps by penalizing large coefficients, but it may not be sufficient alone. The combination of SMOTE and class weighting can sometimes lead to overfitting if the synthetic samples do not accurately represent the minority class. This is why exploring more complex models is crucial to improve robustness.

2. Support Vector Machines (SVMs)

SVMs can be used for either classification or for the prediction of a continuous variable. In the optimization objective of SVMs, we ultimately want to find an optimal hyperplane that can separate the two classes, with the goal of maximizing the spacing between them. **In our case, we need to find an optimal decision boundary to classify fraudulent transactions from normal transactions of the Credit Card Fraud Detection.**

SVMs has the following significance and advantages in fraud detection:

The decision boundary of the SVM is determined by a small number of support vectors (**closest to the hyperplane** and the most difficult to classify), so it is particularly robust in the face of noise and outliers in the dataset. Besides, by introducing kernel functions, SVMs can improve the separation ability of the model that can map the data to a higher dimensional space although the original feature space is difficult to separate.

Maximizing the spacing (margin) of hyperplanes can help us to improve the generalization of the model, making it perform well on unseen data.

Choosing the right kernel function can significantly improve the effectiveness of SVM in fraud detection tasks. Data features (such as $V1$ to $V28$, Time, Amount) are subjected to dimensionality reduction or anonymization, the relationship between features may be linear or non-linear, and we do not know a priori whether a fraudulent transaction can be distinguished from a normal transaction by a linear relationship. Different kernel functions are suitable for different data distributions. So, it is important to try different kernel functions (e.g., linear kernel and RBF kernel). However, according to combine the simplicity of the linear kernel and the complexity of the RBF kernel, it is possible to test the fitness of the machine learning model in a more comprehensive way, effectively improve the ability to recognize a small number of classes. Ultimately it can obtain a highly efficient and accurate fraud detection model to help the enterprise or organization to respond to fraud more accurately.

2.1 Linear SVM Classification

The linear kernel assumes that the features of fraudulent and normal transactions can be separated by a linear hyperplane. It is computationally simple and fast to train.

(1) Firstly, we hope to **find a Linear hyperplane**: $\boldsymbol{w} \cdot \mathbf{X} + \boldsymbol{b} = \mathbf{0}$

$X = x_1, x_2, \dots, x_{28}, x_{\text{amount}}$: features, including 29 features ($V1$ to $V28$ and Amount) in the Credit Card Fraud Dataset

w: the weight vector defines the direction of the hyperplane

b: bias, defines the offset of a hyperplane

Decision rule: $f(x) = \text{sign}\,(\boldsymbol{w} \cdot \mathbf{X} + \boldsymbol{b})$

sign: sign function is to return the corresponding symbol based on the polarity of the input value. Only determining the classification result.

If $f(x) > 0$, predicted as fraudulent transaction (+1)

$f(x) < 0$, predicted as normal trading (-1).

(2) Constructing classification boundaries by **maximizing path width** P $P = \dfrac{b_u - b_d}{\sqrt{w_1^2 + w_2^2}}$

→ therefore set bu =b+1 and bd=b−1so that the width of the pathway is (Normalization of Interval)

$P = \dfrac{2}{\sqrt{w_1^2 + w_2^2}}$【A more specific derivation process can be found in textbook by John Hull using the triangle(sinθ/ cosθ) P106 - P108】

→ $\min \frac{1}{2}\|\mathbf{w}\|^2$ ($w_1^2 + w_2^2 + \cdots + w_{28}^2 + w_{\text{amount}}^2$) , subject to **r (*w* · X + *b*) ≥ 1**

r ∈{−1,+1}, indicate sample category (positive or negative)

(3) Introducing the **<u>Lagrange multiplier</u>** method to solve the problem allows us to optimize $\min \frac{1}{2}\|\mathbf{w}\|^2$ under constraints, thereby finding the optimal classification hyperplane. This method can help us integrate all constraints into the objective function, and the optimization problem is transformed into an unconstrained optimization problem by introducing Lagrange multipliers so that it can be solved using mathematical optimization tools (e.g., derivatives).

- Introduction of Lagrange multipliers $\alpha i \geqslant 0$ (each sample corresponds to one multiplier αi), the constrains y (***w*** · **X** + ***b***) ≥ 1 is integrated into the objective function to construct the Lagrangian function:
$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{t=1}^{N}\alpha^t\left[r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) - 1\right]$$
$$= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{t=1}^{N}\alpha^t r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) + \sum_{t=1}^{N}\alpha^t$$
- Optimizing the Lagrangian function: The partial derivatives with respect to w and b are 0 (i.e., with respect to the extreme points of L):
$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{t=1}^{N}\alpha^t r^t \mathbf{x}^t$$
$$\frac{\partial L_p}{\partial w_0} = 0 \Rightarrow \sum_{t=1}^{N}\alpha^t r^t = 0$$
- Optimizing the Lagrangian function: The dyadic condition:
$$L_d = \frac{1}{2}\left(\mathbf{w}^T\mathbf{w}\right) - \mathbf{w}^T\sum_t \alpha^t r^t \mathbf{x}^t - w_0\sum_t \alpha^t r^t + \sum_t \alpha^t$$
$$= -\frac{1}{2}\left(\mathbf{w}^T\mathbf{w}\right) + \sum_t \alpha^t$$
$$= -\frac{1}{2}\sum_t\sum_s \alpha^t\alpha^s r^t r^s \left(\mathbf{x}^t\right)^T\mathbf{x}^s + \sum_t \alpha^t$$
subject to $\sum \alpha^t r^t = 0$ and $\alpha^t \geq 0, \forall t$

  If $\alpha_i > 0$, corresponding to the sample **X** is the support vector;

  If $\alpha_i = 0$, corresponding to samples that do not contribute directly to the construction of the hyperplane.

The Primal Problem (Primal Problem) to directly optimize *w* and *b* is a constrained optimization problem with more variables and complexity. In contrast, by Lagrange multiplier method, the transformed dyadic problem only needs to optimize αi (a scalar variable), which makes the optimization process simpler and more efficient. After we know which support vectors contribute to the classification hyperplane and their corresponding Lagrange multipliers, we can compute *w* and *b* in the decision boundary equation by using the equation $\mathbf{w} = \sum_{t=1}^{N} \alpha^t r^t \mathbf{x}^t$ and r ($w \cdot X + b$) $= \pm 1$

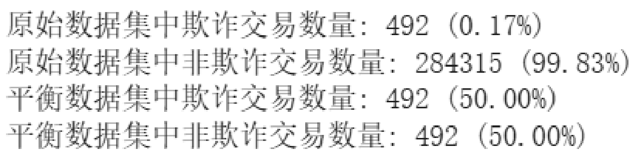
If r =1, $w \cdot X + b = 1$, positive support vector

If r = -1, $w \cdot X + b = -1$, negative support vector

(4) Credit Card Fraud Detection by using Linear SVM Classification

Linear SVM Classification doesn't work well for a seriously imbalanced data set, it can't compensate for imbalance through complex mapping. So firstly, we need to create a balanced data set from the original (284,807series). 【Fraud data==1: contains all fraudulent transactions in the original data, totaling 492. Non-fraudulent==0 data: 492 randomly selected from the non-fraudulent transactions in the original data to match the number of fraudulent data.】

Figure 2.1 The information of original and balanced data set of Credit Card Fraud Detection


```
原始数据集中欺诈交易数量: 492 (0.17%)
原始数据集中非欺诈交易数量: 284315 (99.83%)
平衡数据集中欺诈交易数量: 492 (50.00%)
平衡数据集中非欺诈交易数量: 492 (50.00%)
```

According to the model, we know that formulation of the decision function for the linear kernel is *f(x)* = -0.0253*Time + -92.6128*V1 + 74.8053*V2 + -201.2850*V3 + 223.4949*V4 + 41.3418*V5 + -45.6438*V6 + -99.2689*V7 + -39.9838*V8 + -94.0956*V9 + -202.5285*V10 + 151.5486*V11 + -227.1648*V12 + -28.6641*V13 + -389.3884*V14 + -7.6143*V15 + -88.2144*V16 + -81.8472*V17 + -4.0258*V18 + -10.8521*V19 + -2.0245*V20 + 31.3225*V21 + 9.5808*V22 + 1.6069*V23 + -18.6257*V24 + -1.9595*V25 + 9.8317*V26 + 11.4215*V27 + 5.0060*V28 + 2.8614*Amount + -1548.6275, the intercept is -1548.6275. In the decision function of a linear kernel, the weights reflect how much the linear relationship contributes to the classification result. The coefficients of V14 are -389.3884, which is the largest absolute value of all features, indicating that it has the strongest influence on the classification results. The weights of V12, V3, V4 are also very significant and are respectively −227.1648, -201.2850 and 223.4949. With the resolution of the formula and the intercept, the SVM model with linear kernel mainly relies on specific features (e.g., V14, V12, V3, V4) to distinguish between fraudulent and non-fraudulent samples.

Figure 2.2 Confusion matrix of the balanced data used Linear SVM Classification

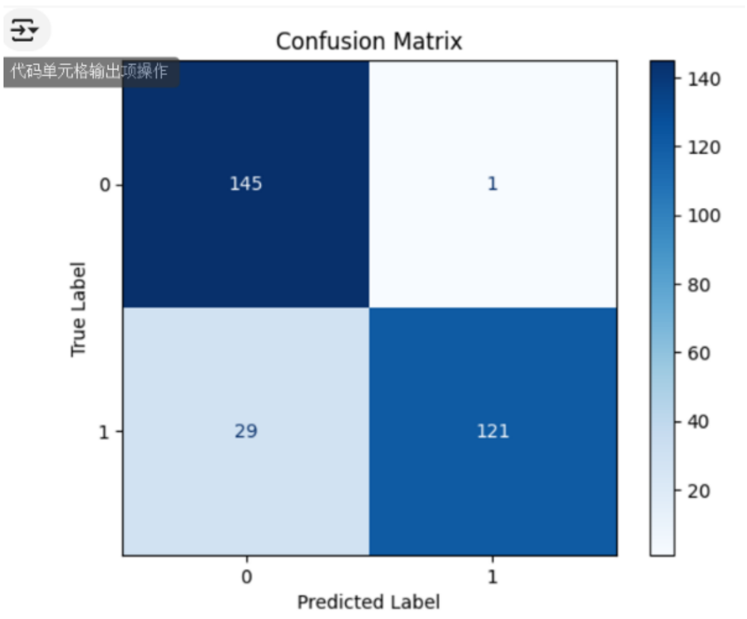
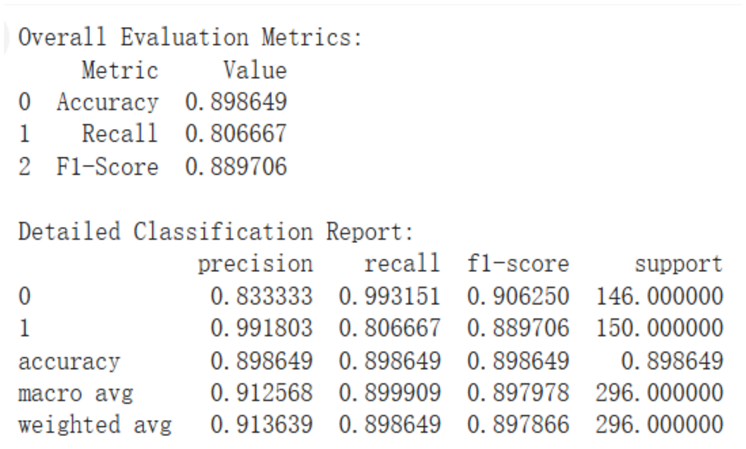


Figure 1.2 can be seen that the model performs better in detecting samples of category "0" (non-fraudulent), with a total of 145 samples correctly categorized and only 1 misclassified; while in detecting samples of category "1" (fraudulent), the model correctly categorizes 121samples, but 29 samples were misclassified as "0".This indicates that there is still room for improvement in the model's ability to recognize fraudulent samples.

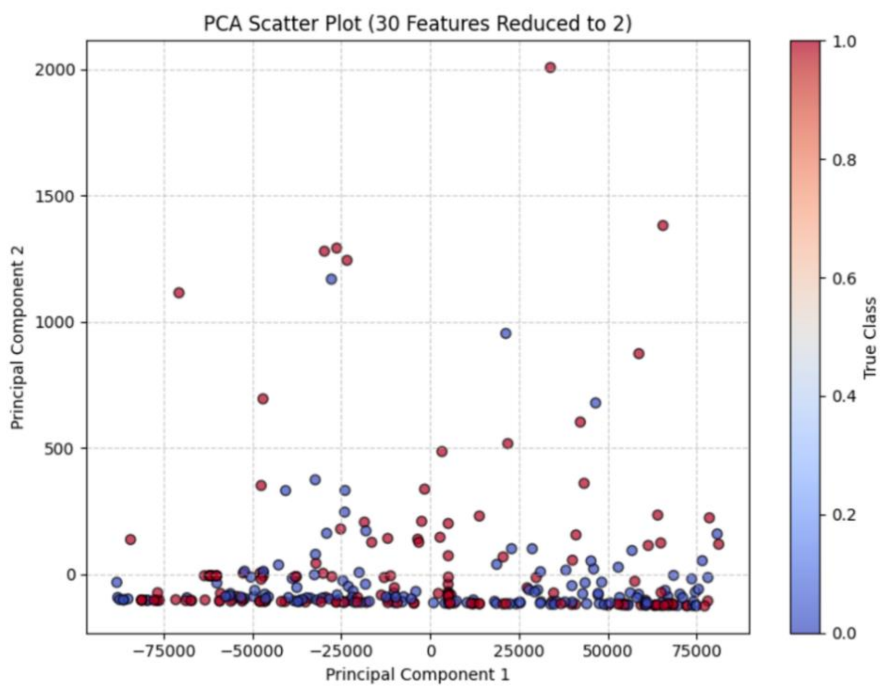
Table 2.1 Overall assessment indicators of the balanced data used Linear SVM Classification


```
Overall Evaluation Metrics:
     Metric     Value
0  Accuracy  0.898649
1    Recall  0.806667
2  F1-Score  0.889706

Detailed Classification Report:
              precision    recall  f1-score     support
0              0.833333  0.993151  0.906250  146.000000
1              0.991803  0.806667  0.889706  150.000000
accuracy       0.898649  0.898649  0.898649    0.898649
macro avg      0.912568  0.899909  0.897978  296.000000
weighted avg   0.913639  0.898649  0.897866  296.000000
```

In terms of the overall evaluation metrics, the model has an accuracy of 89.86%, reflecting a high overall categorization capability. On category 0 (non-fraud), the model has an accuracy rate of

83.33%, a recall rate of 99.32%, and an F1 score of 90.63%, indicating that the model covers most normal transactions well. However, **for category 1 (fraud),** despite the precision rate of 99.18%, **the recall rate is only 80.67%**, which means that there are still nearly 20% of fraudulent samples that are not recognized correctly, which may pose some risk in real scenarios.

Figure 2.3 PCA Scatter Plot of the balanced data used Linear SVM Classification



From Figure 1.3, the original 30-dimensional features are downscaled to 2 principal components, which reflect the real classification by different colors (blue for non-fraud, red for fraud).It can be observed that the distribution of non-fraudulent samples (blue dots) is more concentrated after dimensionality reduction, mainly clustered in low value regions, while the distribution of fraudulent samples (red dots) is more scattered. This suggests that the features of fraudulent samples may be more complex in high-dimensional space and difficult to be effectively separated by a linear kernel. Some of the fraudulent samples have some overlapping distribution with non-fraudulent samples, which further leads to the low recall of the linear model for category 1.

Therefore, the next step can be to try model learning based on the RBF (Radial Basis Function) kernel. RBF kernel has stronger nonlinear modeling capability and can effectively handle complex distribution patterns in high dimensional space, which will help to further improve the model's ability to identify fraudulent samples while optimizing the overall performance.

2.2 Gaussian radial bias function (RBF kernel)

Fraudulent credit card transactions often exhibit highly complex feature patterns hidden in normal transactions, there may be a nonlinear relationship between the amount and certain implicit features (e.g., $V1$ to $V28$ and Amount) The RBF kernel is designed by introducing the kernel function parameters γ, which maps the data to a higher dimensional space, can capture these complex non-linear patterns and thus more accurately distinguish between fraudulent and normal transactions.

(1) The formula for the RBF kernel can be: $\exp(-\gamma D^2)$ The range of output values is [0,1], is used to indicate the similarity between samples.

$$D = \sqrt{\sum_{j=1}^{m}(x_j - \ell_j)^2}$$

D: **The Euclidean distance of sample points x and reference point ℓ** (or "landmark"). Euclidean Distance is one of the most used distance measures for calculating the straight-line distance between two points in space. It is based on "straight-line distances" in two or more dimensions and reflects the "geometric distance" between two points.

j is the index, with a value range of 1 to m, Loop variable j will traverse all features

x_1, x_2 …, x_m, The eigenvectors of the sample points

ℓ_1, ℓ_2, …, ℓ_m, Eigenvectors of the reference points

(2) Optimizing the $\gamma = \frac{1}{2\sigma^2}$ by **Cross-Validation**

σ is the kernel width parameter that controls the perceptual range of the kernel function, it also is called "Scaling factor" to control how much the distance is attenuated.

Parameters γ determines the perceptual range of the RBF kernel and thus affects the model performance:

- γ Too large: the perceptual range of the kernel function is small and the model focuses too much on localized data, which may lead to overfitting (The kernel function has a narrow "field of view" and can only see very close sample points.);

- γ too small: the perceptual range of the kernel function is large and the model is unable to capture local structure, which may lead to underfitting. (The kernel function has a wider "field of view" and is able to see further away from the sample point.)

We have tried the results of the linear kernel for credit card fraud detection in 2.1. If the linear kernel already performs well on certain metrics (e.g., accuracy, recall, etc.), it may indicate that the data is close to a linear distribution. The γ parameter of the RBF kernel can be set smaller, as overly complex nonlinear mapping may lead to overfitting. If the performance of the linear kernel is poor (e.g., low accuracy or low recall), it may indicate that there is a strong nonlinear relationship in the data distribution. The RBF kernel requires a more complex feature mapping and may need a larger value of γ and an appropriate **regularization parameter C** to capture the nonlinear pattern complex relationships.

Therefore, we can specify the **systematic optimization of γ through Cross-Validation.** The specific steps include firstly defining a set of candidates γ, dividing the dataset into a training set and a validation set, and typically employing k-fold cross-validation, where in each fold, the model is fitted with the training set and the performance (e.g., accuracy or AUC) is evaluated on the validation set. All candidates γ are tested and their average validation performance is recorded, and finally the best performing γ values for training the final model. This approach finds an optimal balance between controlling model complexity and generalization ability, ensuring optimal model performance without over- or under-fitting.

(3) Introducing the regularization **hyperparameter C** (similarly used Cross-Validation to optimize) in loss function

The formula $C\sum_{i=1}^{n} z_i + w^2$ describes an equilibrium model complexity (w^2) and misclassification penalties ($\sum zi$) of the regularized objective function.

zi is usually the slack variable associated with the misclassification loss (as in SVMs ξi), essentially is the **error** term in the model.

C determines the weight of the misclassified sample's contribution to the overall loss function:

- When C is larger: penalizes misclassified samples with greater losses.

The model will focus more on **reducing misclassification errors**, which may lead to overfitting.

- When C is smaller: The effect of misclassified samples is weakened.

The model is more concerned with **maximizing the classification interval**, which may lead to underfitting.

In the RBF kernel, hyperparameter C determines the sensitivity of the model to misclassified samples and affects the shape of the decision boundary. γ values controls the "perceptual range" of the RBF kernel, which affects the similarity of the samples in the high-dimensional space. If γ is larger (narrower perceptual range) and C is large, the model generates very complex boundaries, leading to overfitting.

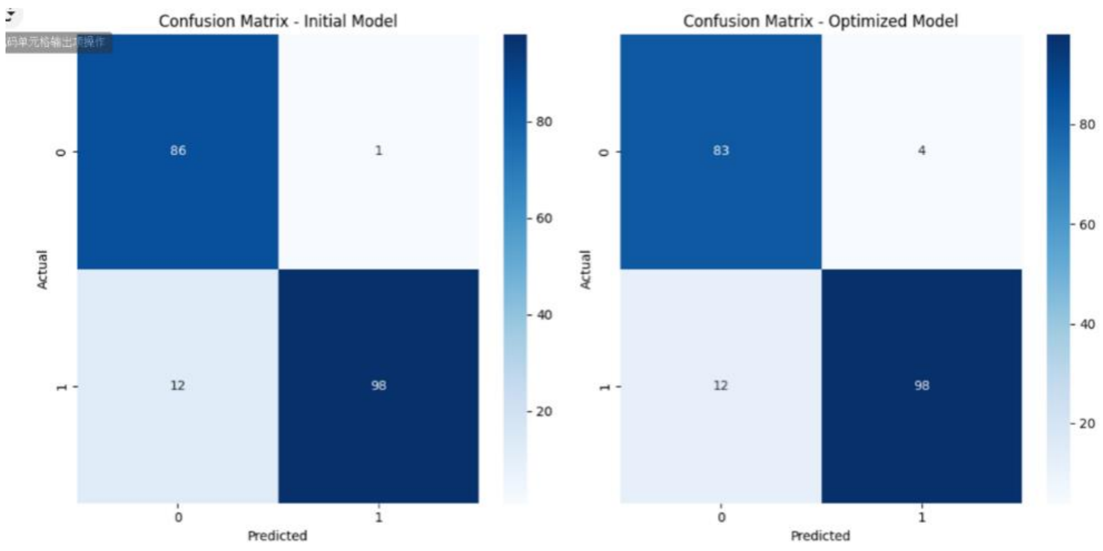
(4) Credit Card Fraud Detection by using RBF kernel

The complexity of the RBF kernel (Radial Basis Function Kernel) is of the square level, i.e.$O(n^2)$

This means that as the number of samples increases, the time and memory required for computation also shows a square increase. For the complete dataset, the number of samples is 284,807, which would result in the need to compute approximately $284{,}807^2$ distances, yielding a huge kernel matrix (about 8.1 billion elements), the time and memory requirements to compute this huge matrix would be overwhelming for most ordinary computers. We cannot be used directly to detect the complete 284,807 data of Credit Card Fraud Detection in 2013 by RBF kernel. So, we have used the following ways to detect Credit Card Fraud in balanced and unbalanced data sets.

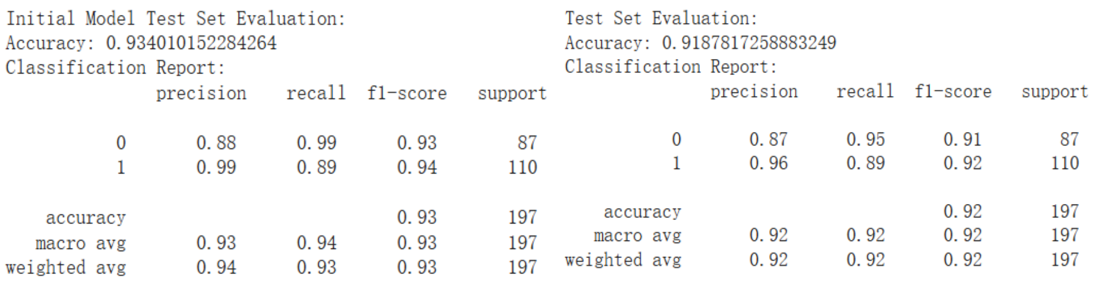
①In 50-50 a balanced data set of Non-fraudulent==0 & fraud==1 data【like 1.1(4)】

Figure 2.4 Confusion matrix of the balanced data used RBF



From Figure 1.4, RBF kernel initial model showed a significant decrease in false positives for category 1 (only 12 compared to Figure 1.2 used Linear SVM Classification), and the optimized model remained consistent. The false alarms for category 0 increased slightly, from 1 to 4. This change indicates that the RBF kernel trades an appropriate increase in false positives for category 0 for an increase in recall for category 1, a trade-off that is acceptable in real fraud detection applications.

Table 2.2 Overall assessment indicators of the balanced data used RBF



```
Initial Model Test Set Evaluation:
Accuracy: 0.934010152284264
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.93        87
           1       0.99      0.89      0.94       110

    accuracy                           0.93       197
   macro avg       0.93      0.94      0.93       197
weighted avg       0.94      0.93      0.93       197
```

```
Test Set Evaluation:
Accuracy: 0.9187817258883249
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91        87
           1       0.96      0.89      0.92       110

    accuracy                           0.92       197
   macro avg       0.92      0.92      0.92       197
weighted avg       0.92      0.92      0.92       197
```

With Table 1.2, comparing the two RBF kernel-based support vector machine (SVM) models on the balanced dataset (492-492), we can conclude that the hyper-parameter-tuned model still performs in a more balanced manner, even though it decreases from the initial model's 0.93 to 0.92 in terms of overall accuracy. Specifically, the tuned model maintains **a better-balanced performance** between non-fraudulent and fraudulent samples and avoids over-biasing towards one category.

The initial model has a high recall of 0.99 on category 0 (non-fraud), but this also means that it is too lenient in judging samples in this category, which may lead to a bias towards category 1 (fraud).The tuned model, on the other hand, has a recall of 0.95 and 0.89 on category 0 and category 1, respectively, showing a more balanced prediction of the two categories of samples. The difference in precision and F1 scores also suggests that the tuned model performs more consistently across categories. **Therefore, the cross-validated and hyper-parameter tuned RBF kernel SVM model is more suitable in the balanced dataset** as it achieves a more balanced performance in detecting fraudulent versus non-fraudulent samples, which helps to reduce the model bias and enhances the generalization ability.

Compared Table 1.1 and 1.2, overall accuracy of the linear kernel is 89.86%, while the initial model accuracy of the RBF kernel is 93.40%, and the accuracy of the optimized model slightly decreases to 91.88%. RBF kernel significantly improves the overall classification performance of the model. From the classification report, the macro-averaged F1 scores of the initial and optimized models of the RBF kernel are 0.93 and 0.92, respectively, which are both better than that of the linear kernel at 0.90, indicating that the RBF kernel has a better performance in dealing with complex data patterns. And RBF kernel significantly improves the recall of category 1 (from linear - 80.67% to 89%), which better addresses the problem of fraudulent sample underreporting. Although the slightly compromised performance of the RBF kernel in category 0 (a slight increase errors for non-fraudulent), the overall identification of non-fraudulent samples remained high (recall maintained above 95%). **Therefore, RBF kernel model outperforms the linear kernel on the balanced dataset, especially in terms of recall and overall classification performance for fraudulent sample detection.**

②In a imbalanced data set of Non-fraudulent==0 (random 10,000 entries) & fraud==1(492 entries) data

The RBF kernel can handle complex nonlinear distributions and learns better for minority class samples, but it still suffers from data imbalance. So, we will use imbalance data directly by using class weights in loss functions, which effectively improve model learning for minority classes (e.g., fraudulent samples). In our case, we will use class_weight='balanced' in Scikit-learn. This can automatically calculate weights based on the number of samples in each category (Fraud and non-fraud).

Figure 2.5 Confusion matrix of the imbalanced data used RBF

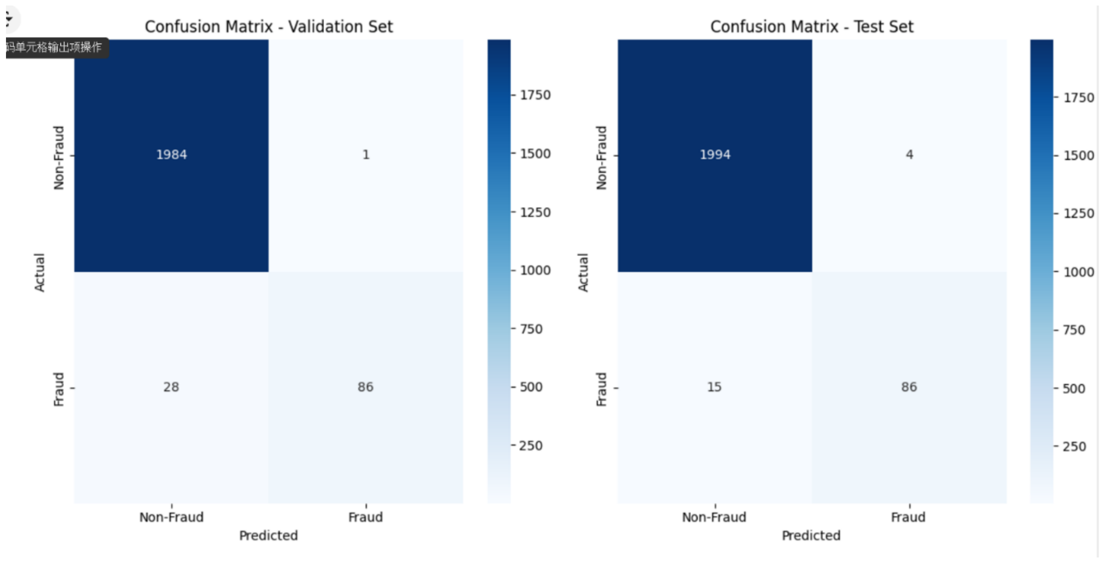
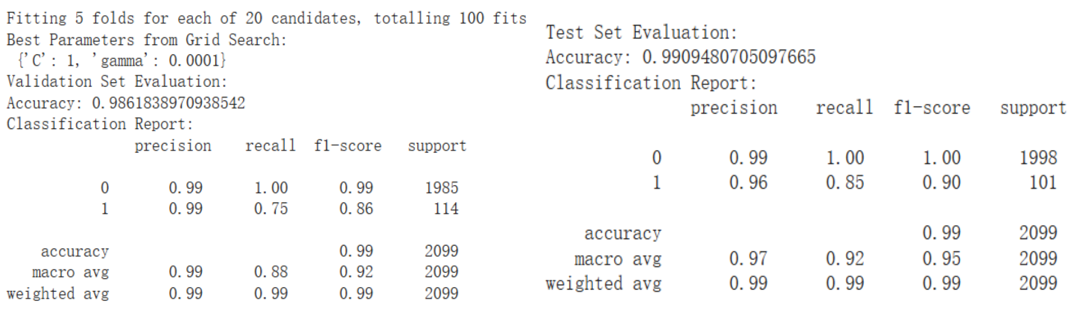


Table 2.3 Overall assessment indicators of the imbalanced data used RBF

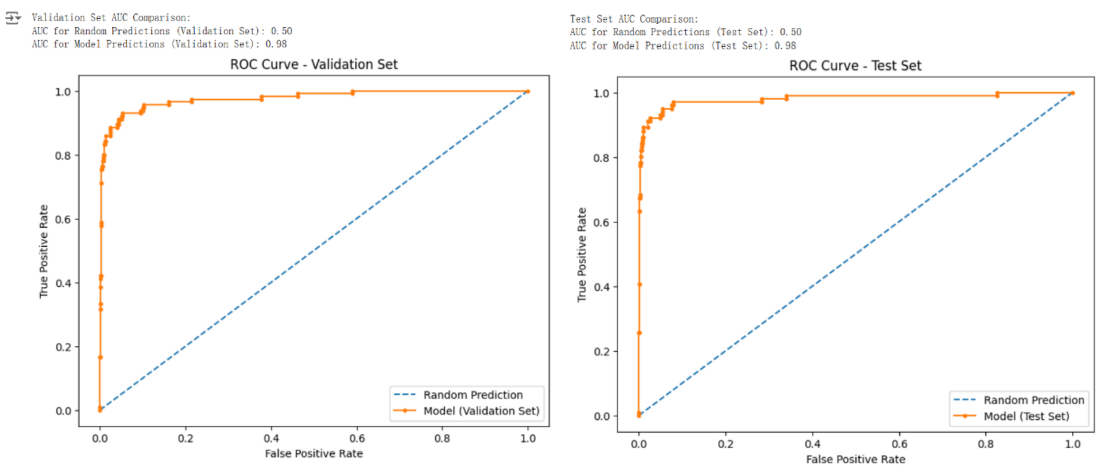


```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Parameters from Grid Search:
 {'C': 1, 'gamma': 0.0001}
Validation Set Evaluation:
Accuracy: 0.9861838970938542
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1985
           1       0.99      0.75      0.86       114

    accuracy                           0.99      2099
   macro avg       0.99      0.88      0.92      2099
weighted avg       0.99      0.99      0.99      2099
```

```
Test Set Evaluation:
Accuracy: 0.9909480705097665
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      1998
           1       0.96      0.85      0.90       101

    accuracy                           0.99      2099
   macro avg       0.97      0.92      0.95      2099
weighted avg       0.99      0.99      0.99      2099
```

From Figure 1.5 and Table 1.3, overall assessment of SVM model with RBF kernel on imbalanced fraudulent credit card datasets is quite outstanding, especially in classifying non-fraudulent samples, which is almost perfect. In the test set, the recall rate of non-fraudulent samples reached 1.00, the accuracy was 0.99, and the F1 score was 1.00, indicating that the model can accurately identify all non-fraudulent samples with almost no false positives. In the validation set, the recall rate of non-fraudulent samples also show very stable performance. This result reflects the strong ability of the model to handle majority class (non-fraudulent) samples, especially after adjusting the class weights through classwidth='balanced ', the adaptability of the model to imbalanced data has been significantly improved.

Figure 2.6 ROC curve and AUC value of the imbalanced data used RBF



```
Validation Set AUC Comparison:
AUC for Random Predictions (Validation Set): 0.50
AUC for Model Predictions (Validation Set): 0.98
```

```
Test Set AUC Comparison:
AUC for Random Predictions (Test Set): 0.50
AUC for Model Predictions (Test Set): 0.98
```

Figure 1.6 shows that SVM model with RBF kernel performs well on imbalanced fraudulent credit card datasets. The AUC values on both the validation and test sets reached 0.98, far higher than the benchmark AUC value of 0.50 for random prediction. This indicates that the model has strong discriminative ability and can effectively separate fraudulent transactions from non-fraudulent transactions. The shape of the ROC curve is close to the upper left corner, indicating that the model performs excellently in balancing both low False Positive Rate and high True Positive Rate.

Although the ROC curve and AUC values demonstrate the overall excellent performance of the model, they reflect the global capability of the model and are unable to fully reveal the local performance on minority classes (fraudulent samples). With Figure 1.5 and Table 1.3, there are still significant shortcomings in the classification performance of fraudulent samples. In the test set, the accuracy of fraudulent samples was 0.96, the recall rate was 0.85, and the F1 score was 0.90, which means that 15% of fraudulent samples were misclassified as non-fraudulent. In the validation set, the accuracy of fraudulent samples is 0.99, but the recall rate is only 0.75, and the F1 score is 0.86, indicating that the model has a more serious problem of false positives for fraudulent samples. Based on the confusion matrix, it can be inferred that these missed fraudulent samples will be misclassified as non-fraudulent, which contradicts the demand for high recall rates of fraudulent behavior in practical applications.

In 1.2②, we only sampled a portion of the data. In the complete dataset (284,807 transaction records, while only 492 fraudulent transactions, accounting for less than 0.2%), this imbalance will be more notable. It is great significant to introduce Random Forests for machine learning. As a powerful ensemble learning method, Random Forests can significantly improve the recognition ability of minority classes, especially recall rate by adjusting sampling strategies (such as balanced random forest or oversampling techniques) and optimizing model weights. Therefore, in such extremely imbalanced datasets, attempting to use it for modeling not only helps to further optimize recall rates, but also provides more comprehensive and reliable support for fraud detection.

3. Random Forest

3.1 Decision Trees

(1) Decision Trees are a type of machine learning algorithm that can be used for both classification and regression tasks. They infer the value of a target variable from data features by learning simple decision rules. Decision Trees are constructed by recursively partitioning the dataset into smaller subsets, with each partition based on the outcome of a test on a particular feature.

(2)The basic components of a Decision Tree are: ①Nodes: Root Node: The starting point of the decision tree, which contains the entire dataset. Internal Nodes: Represent a test on a feature, used to split the dataset into smaller subsets. Leaf Nodes: Represent the final decision outcome, typically containing a class label or predicted value. ②Branches: Paths that extend from an internal node, representing different decision outcomes. ③Paths: A series of connected branches from the root node to a leaf node, representing a sequence of decisions.

(3) The construction process of a Decision Tree:

①Selecting the Classification Feature:

Determine the threshold of the feature using methods such as entropy and the Gini index to identify the best classification point. Choose the feature with the greatest information gain as the splitting feature for the node, and use a certain value of this feature as the threshold to split the dataset.

Information Gain is one of the criteria used in decision trees to select the splitting feature. It is defined as: Information Gain = Original Information Entropy - Conditional Information Entropy. The greater the information gain, the better the feature A's effect on splitting the dataset, i.e., it reduces more impurity.

a)Entropy (Entropy): In machine learning (ML), entropy is often used to measure the uncertainty of a random variable. The formula for entropy is$-\sum_{i=1}^{n} p_i \log(p_i)$, where p_i is the probability of the i-th outcome, n is the total number of possible outcomes, and the sum of all probabilities is 1 ($\sum_{i=1}^{n} p_i = 1$). The initial entropy formula is: $H(Y)=-\sum p(y)\log_2(p(y))$, which represents the uncertainty without any feature splitting; the conditional information entropy formula is: H(Y|Degree)=∑p(Degree)H(Y|Degree=v); H(Y|Degree=v) is the entropy of the dataset given the feature Degree=v. This requires iterating over each possible value of the feature (p(Degree)) to calculate the entropy of the dataset under that feature value, then summing up weighted by the probability of the feature value occurring.

b)Gini Index (Gini Measure): Commonly used in decision tree algorithms, its calculation formula is: Gini=1-∑_(i=1)^n p_i^2. The closer the Gini index is to 1, the higher the uncertainty; the closer to 0, the lower the uncertainty. The Gini index is used to calculate the impurity of the dataset. When Gini(D)=0, the dataset is pure, with all samples belonging to the same class; when Gini(D) is close to 0.5, the impurity of the dataset is at its maximum, with samples evenly distributed across all classes. Decision trees use the Gini index as a splitting criterion, optimizing the classification effect by choosing the split that reduces the Gini index the most, with the reduction in Gini index equal to the information gain, which is the original Gini index minus the conditional Gini index. The Gini index is calculated faster than entropy because it does not involve logarithmic operations.

Mean Squared Error: A measure of the difference between predicted values and actual values, used for regression problems.

②Recursive Tree Construction: Split the dataset using the selected feature and threshold, then repeat the above process for each subset until a stopping condition is met (such as all samples in a node belonging to the same class, reaching a preset maximum depth, or the number of samples in a node being below a certain threshold).

③Handling Continuous and Categorical Features: For continuous features, the algorithm attempts to split at each unique value of the feature; for categorical features, the algorithm tries each category as a splitting point. Then, the algorithm compares the information gain of these splits and selects the best one.

④Optimization and Pruning: After the decision tree is constructed, it may need to be pruned to avoid overfitting. During the pruning process, cross-validation is often used to evaluate the performance of the tree. Evaluation criteria can include accuracy, precision, recall, F1 score, etc., and nodes that do not contribute significantly to the overall performance of the tree are removed.

(4) Advantages of Decision Trees in Classification Problems:

①Easy to understand and interpret: Decision trees are intuitive and close to human decision-making processes, and they can also be clearly displayed through visualization to show feature splits and classification rules.②No need for feature normalization or standardization for features of different scales, capable of handling both numerical and categorical features.③Applicable to small datasets.

Disadvantages:

①Prone to overfitting: The tree is too deep. ②Sensitive to fluctuations in small datasets.③Biased towards features with many values.④Sensitive to class imbalance, the decision tree may favor the dominant class.⑤Limited to univariate splits.⑥Computationally complex for very deep trees.⑦Greedy algorithm, choosing the best split point at each step, but it does not guarantee a global optimum.

Solutions to Disadvantages:

①Prevent overfitting: Regularization parameters such as limiting the maximum depth (max_depth); setting the minimum number of samples for a split (min_samples_split) or the minimum number of samples in a leaf node (min_samples_leaf); using pruning techniques (pre-pruning: limiting split conditions during training, or post-pruning: first growing a full tree and then simplifying it by removing unimportant branches).

②Sensitive to fluctuations in small datasets: Use ensemble learning (such as Random Forest: reducing overfitting and sensitivity to noise, or Gradient Boosting Trees: improving prediction accuracy).

③Biased towards features with many values: Use information gain ratio or penalty mechanisms.

④ When class distribution is imbalanced: Use balanced class weights (such as class_weight='balanced') or resampling (oversampling or undersampling).

3.2 Random Forest

(1)Since the credit card dataset contains 280,000 records, it is a massive dataset. Random Forest can efficiently handle large-scale datasets and has high computational efficiency by parallelizing the processing of multiple decision trees. Additionally, the dataset has 29 features, which is quite a few. Random Forest, by randomly selecting a subset of features for splitting, can prevent over-reliance on a single feature and improve generalization. Moreover, in anomaly detection tasks, Random Forest can identify samples that do not conform to the predictions of the majority of trees, making it suitable for anomaly detection tasks. Therefore, in the study of credit card fraud issues, we have chosen to use the Random Forest model for fraud detection.

(2)Random Forest is an ensemble model composed of multiple decision trees, whose main idea is to reduce the overfitting issues of a single decision tree through randomness and to improve the model's generalization ability. It employs the following two types of randomness:

①Random Sampling (Sample Randomness): Multiple subsets of samples are generated through bootstrap sampling with replacement, and each subset is used to train a decision tree.

②Feature Randomness: At each split node of a tree, a random subset of features (rather than all features) is selected to find the best split.

(2)The workflow of Random Forest:

① Randomly sample from the original dataset to generate k subsets of samples.

②Train a decision tree for each subset: At each split node, a random subset of features is selected for splitting (the size of the feature subset is typically $\sqrt{d}$, where d is the total number of features).

③Obtain the final result by integrating the prediction results of multiple trees: In classification problems, the classification result is determined by a majority vote (the minority follows the majority).

(3)Core parameters of Random Forest:

①n_estimators: The number of decision trees in the forest. Generally, the more trees, the better the model performance, but the greater the computational cost.

②max_features: The number of features randomly selected at each split. For classification problems, the default is √d.

③max_depth: The maximum depth of each tree. Limiting the depth can prevent overfitting.

④min_samples_split: The minimum number of samples required to split a node.

⑤min_samples_leaf: The minimum number of samples in a leaf node to avoid the leaf node being too small and causing the model to overfit.

⑥bootstrap: Whether to use bootstrap sampling with replacement (default is True).

⑦criterion: The criterion for the split, commonly "gini" or "entropy" for classification problems.

3.3 Dataset Processing:

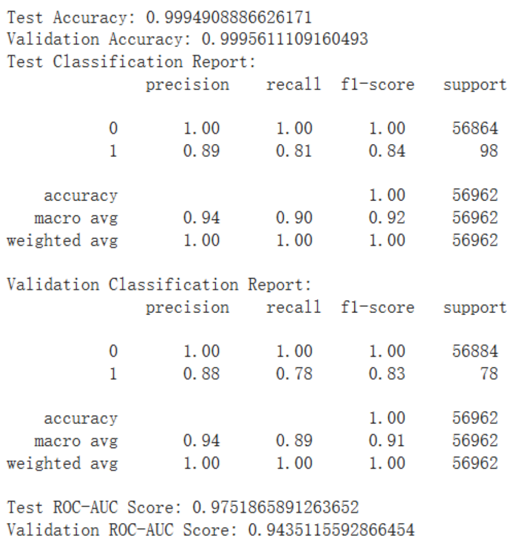
There are 284,807 transactions, with 492 flagged as fraudulent. The dataset clearly has a severe class imbalance issue (the number of normal transactions is much greater than the number of fraudulent transactions). To balance the data, class weight adjustment is incorporated.

In the code, SMOTE (Synthetic Minority Over-sampling Technique) is used to oversample the training set to balance the class distribution. Afterward, class weights are calculated, and additional weight is given to the minority class to further balance the dataset.

After processing the dataset, a Random Forest classifier is created, with the number of trees set to 200, and the calculated class weights are applied. The model is then trained, followed by model evaluation and result visualization.

3.4 Result Analysis

Figure 3.1 Overall assessment indicators of the total dataset used Random Forest



```
Test Accuracy: 0.9994908886626171
Validation Accuracy: 0.9995611109160493
Test Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.89      0.81      0.84        98

    accuracy                           1.00     56962
   macro avg       0.94      0.90      0.92     56962
weighted avg       1.00      1.00      1.00     56962

Validation Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.88      0.78      0.83        78

    accuracy                           1.00     56962
   macro avg       0.94      0.89      0.91     56962
weighted avg       1.00      1.00      1.00     56962

Test ROC-AUC Score: 0.9751865891263652
Validation ROC-AUC Score: 0.9435115592866454
```

The test set accuracy is 0.99949 and validation set accuracy is 0.99956. The accuracy is very high, close to 1, indicating that the model can correctly classify in most cases. However, high accuracy alone is not sufficient to judge the model's performance as the data is imbalanced.

A more detailed view of the model's performance metrics can be seen from the classification report:

In the test set: For class 0, which represents non-fraudulent credit card transactions, the precision, recall, and F1 score are all 1, indicating that the model can perfectly predict this class. For class 1, which represents fraudulent transactions (with a small number of samples), the precision is 0.89, suggesting that the model has a high accuracy in predicting this class but with some false positives. The recall is 0.81, indicating that the model fails to identify all samples of class 1. The F1 score is 0.84, a balanced measure between precision and recall, suggesting that the model's performance for class 1 is acceptable.

In the validation set: For class 0 (non-fraudulent), the metrics are consistent with the test set. For class 1 (fraudulent), the precision, recall, and F1 score are slightly lower than in the test set, indicating a slight decrease in the model's performance on the validation set.

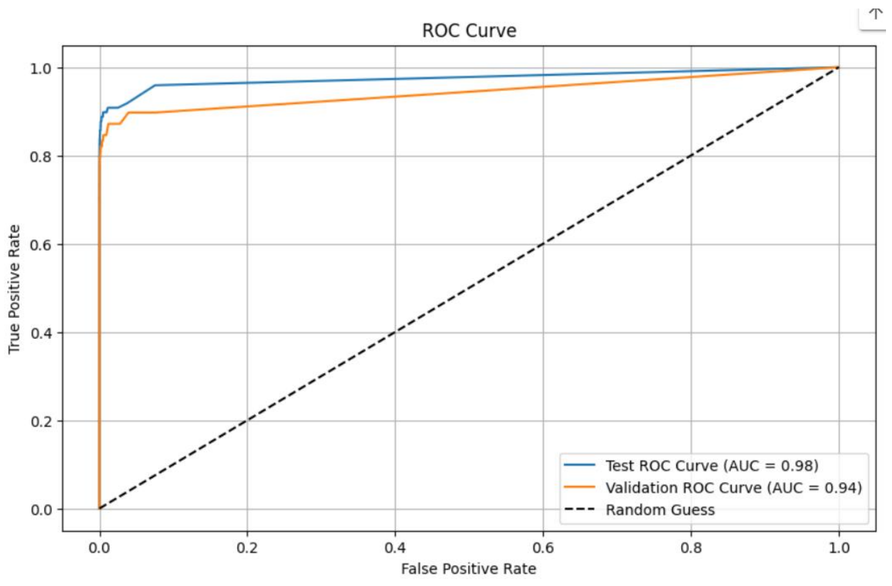
Macro average (macro avg): The arithmetic mean of the metrics for each class, regardless of sample distribution. It is approximately 0.9, slightly lower, indicating weaker performance for class 1.

Weighted average (weighted avg): A weighted mean that takes into account the number of samples. Since class 0 is overwhelmingly larger, it is close to the metrics of class 0.

The ROC-AUC score reflects the model's overall ability to distinguish between positive and negative classes. Both are above 0.9, indicating good performance in this regard.

The sample ratio of class 0 to class 1 is extremely imbalanced (very few class 1 samples: 98 vs. 56,864 in the test set, 78 vs. 56,884 in the validation set). This imbalance may lead to the model having weaker predictive power for class 1 (low recall) and being overly "confident" in its predictions for class 0.

Figure 3.2 ROC curve and AUC value



The ROC (Receiver Operating Characteristic) curve illustrates the model's performance at various classification thresholds. The vertical axis represents the True Positive Rate (TPR, also known as recall), and the horizontal axis represents the False Positive Rate (FPR).

On the test set: The AUC (Area Under the Curve) value is 0.98, close to 1. This indicates that the model has an extremely high ability to distinguish between positive and negative classes on the test set, almost perfect. The shape of the curve is close to the top left corner, suggesting that at most thresholds, the model can maintain a high recall rate and a low false positive rate.

On the validation set: The AUC value is 0.94, which is slightly lower than on the test set but still very high, indicating that the model also performs excellently on the validation set.

The black dashed line represents the performance of a random classifier, with an AUC of 0.5. The AUC values for both the test set and the validation set are significantly higher than 0.5, demonstrating that the model is substantially better than random guessing.

4. Neural Network

A neural network is a computational model inspired by the brain, capable of learning complex, non-linear patterns from data through interconnected layers of artificial neurons. **In the context of credit card fraud detection, neural networks are used to classify transactions as legitimate or fraudulent by analyzing features like transaction amount and time. By preprocessing data, addressing class imbalance, and employing advanced architectures with techniques like dropout and batch normalization, neural networks provide scalable, real-time fraud detection solutions. These models are particularly effective in adapting to evolving fraud patterns, ensuring high accuracy and reliability in identifying suspicious activities.**

Neural networks has the following significance and advantages in fraud detection:

Neural networks excel in credit card fraud detection due to their ability to model complex, non-linear relationships in the data, which are often inherent in fraudulent behavior. They can automatically learn and extract relevant features from raw data without extensive manual feature engineering, making them highly adaptable to diverse datasets. Neural networks handle large-scale datasets effectively, leveraging their capacity to uncover intricate patterns in high-dimensional data. When paired with techniques like SMOTE or class-weight adjustments, they can address imbalanced datasets, a common challenge in fraud detection. Furthermore, their high classification performance, including strong precision, recall, and AUC scores, ensures reliable detection of fraudulent transactions, even in challenging scenarios. Their flexibility also allows them to incorporate various data types, such as numerical, categorical, and sequential features, providing a holistic understanding of transaction behaviors.

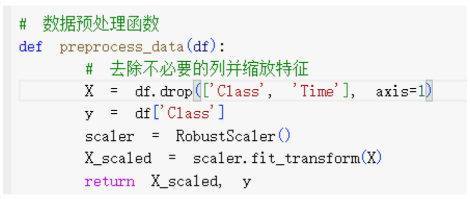
However，**there are some shortcoming:**

Despite their strengths, neural networks have several limitations in fraud detection. They require large amounts of high-quality data for effective training, which can be challenging to obtain, particularly for rare events like fraud. Training and deploying neural networks demand significant computational resources, which can be costly, especially for real-time systems. Their "black-box" nature makes them difficult to interpret, posing challenges in regulatory compliance and trust for financial institutions. Additionally, neural networks are sensitive to noisy data and outliers, potentially leading to performance degradation if data preprocessing is inadequate. They are prone to overfitting, especially when dealing with small or imbalanced datasets, unless carefully regularized. Furthermore, hyperparameter tuning is complex and time-consuming, requiring expertise and computational effort. Finally, their inability to adapt quickly to evolving fraud patterns without retraining can limit their effectiveness in dynamic fraud detection scenarios.

4.1 Data Preprocessing

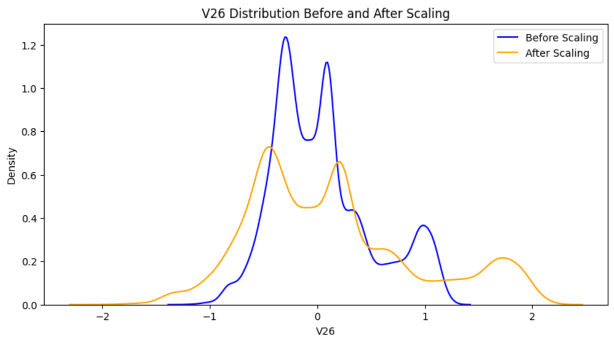
In the data preprocessing section, we first remove unnecessary columns, including "Class" and "Time", to ensure that only the data related to the feature is retained. Next, we extract the "Class" column separately as the target variable. To process outliers in the data and standardize the features, RobustScaler was used for feature scaling. Finally, the function returns the scaled feature data and target variables. This step ensures the quality and consistency of the data, laying the foundation for subsequent analysis and modeling. (Here, 26 feature processing results are shown as examples)

Figure 4.1



```
# 数据预处理函数
def preprocess_data(df):
    # 去除不必要的列并缩放特征
    X = df.drop(['Class', 'Time'], axis=1)
    y = df['Class']
    scaler = RobustScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, y
```

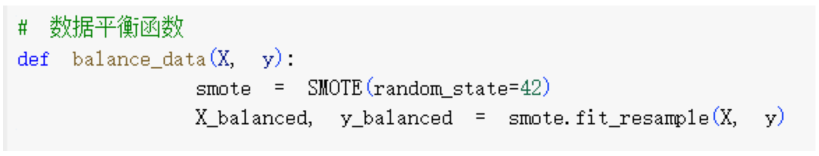
Figure 4.2



4.2 Class Balancing

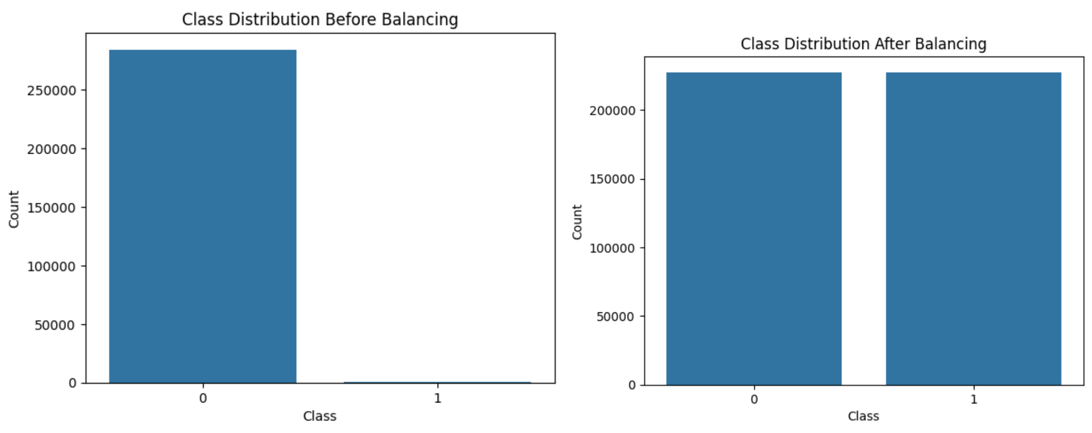
In the data balancing section, we define a function called 'balance_data' to handle category imbalance in the data set. This function uses the SMOTE (Synthetic Minority Oversampling Technique) method to balance the data set by generating new minority samples. The steps include: first instantiate SMOTE and set a random seed to ensure repeatability of the results; Then oversampling the feature data 'X' and the target variable 'y' with SMOTE method, finally returning the balanced feature data 'X_balanced' and the target variable 'y_balanced'. In this way, we can effectively solve the class imbalance problem and improve the performance and accuracy of the model.

Figure 4.3



```
# 数据平衡函数
def balance_data(X, y):
        smote = SMOTE(random_state=42)
        X_balanced, y_balanced = smote.fit_resample(X, y)
```

Figure 4.4



4.3 Neural Network Architecture

The neural network architecture is designed to model complex, non-linear relationships in the data:

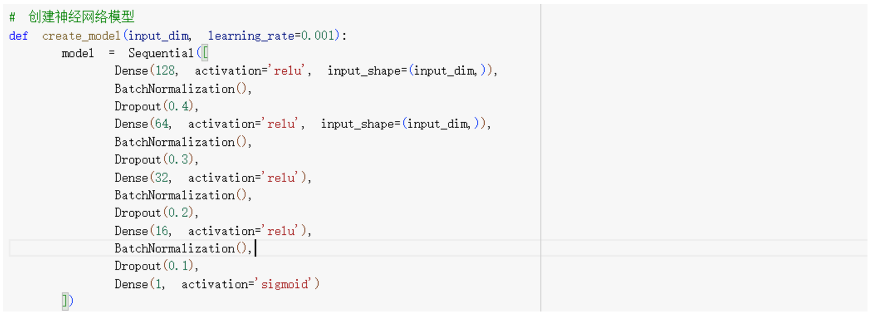
- **Input Layer**: Accepts preprocessed transaction features (dimensionality defined by the feature count).
- **Hidden Layers**: Four layers with decreasing neuron counts (128 → 64 → 32 → 16). Each

layer uses **ReLU activation functions** to model non-linear patterns while avoiding vanishing gradient problems.

  - **Batch Normalization** stabilizes and speeds up training by normalizing the input to each layer.
  - **Dropout** layers (with rates 0.4, 0.3, 0.2, and 0.1) prevent overfitting by randomly deactivating neurons during training.

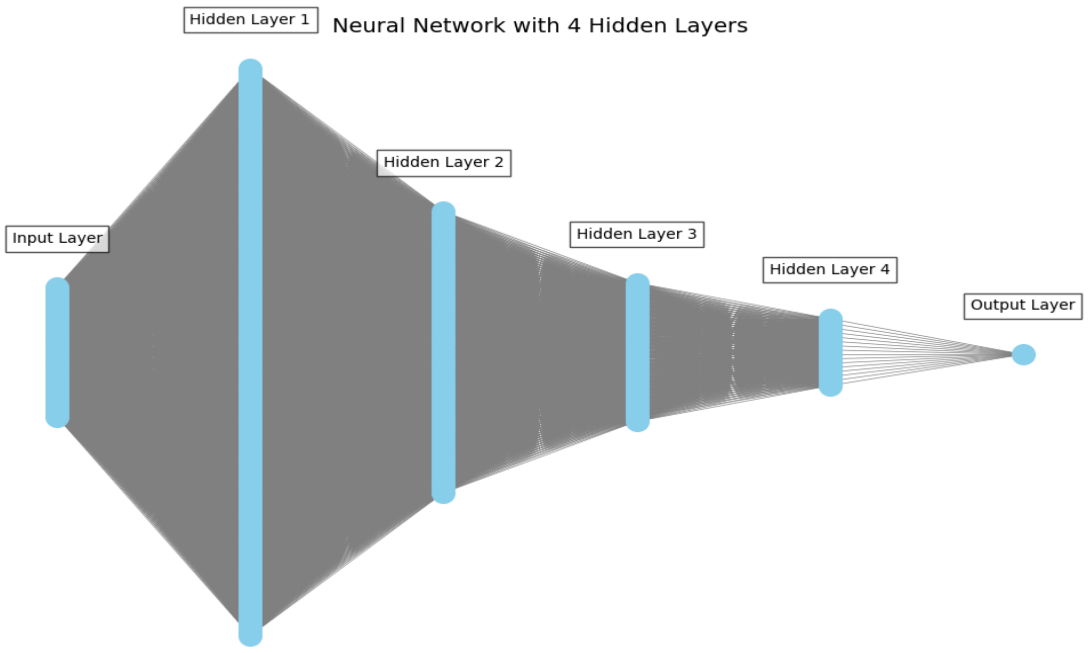
- **Output Layer**: A single neuron with a **sigmoid activation function**, producing a probability value between 0 and 1, which represents the likelihood of a transaction being fraudulent.

Figure 4.5



```
#  创建神经网络模型
def  create_model(input_dim,  learning_rate=0.001):
        model  =  Sequential([
                Dense(128,  activation='relu',  input_shape=(input_dim,)),
                BatchNormalization(),
                Dropout(0.4),
                Dense(64,  activation='relu',  input_shape=(input_dim,)),
                BatchNormalization(),
                Dropout(0.3),
                Dense(32,  activation='relu'),
                BatchNormalization(),
                Dropout(0.2),
                Dense(16,  activation='relu'),
                BatchNormalization(),
                Dropout(0.1),
                Dense(1,  activation='sigmoid')
        ])
```

Figure 4.6



4.4 Optimization and Regularization

The model uses **binary cross-entropy loss** as the objective function, which is well-suited for binary classification tasks. The loss is minimized using the **Adam optimizer**, which combines momentum and adaptive learning rates for efficient and stable convergence.

To prevent overfitting, the model incorporates:

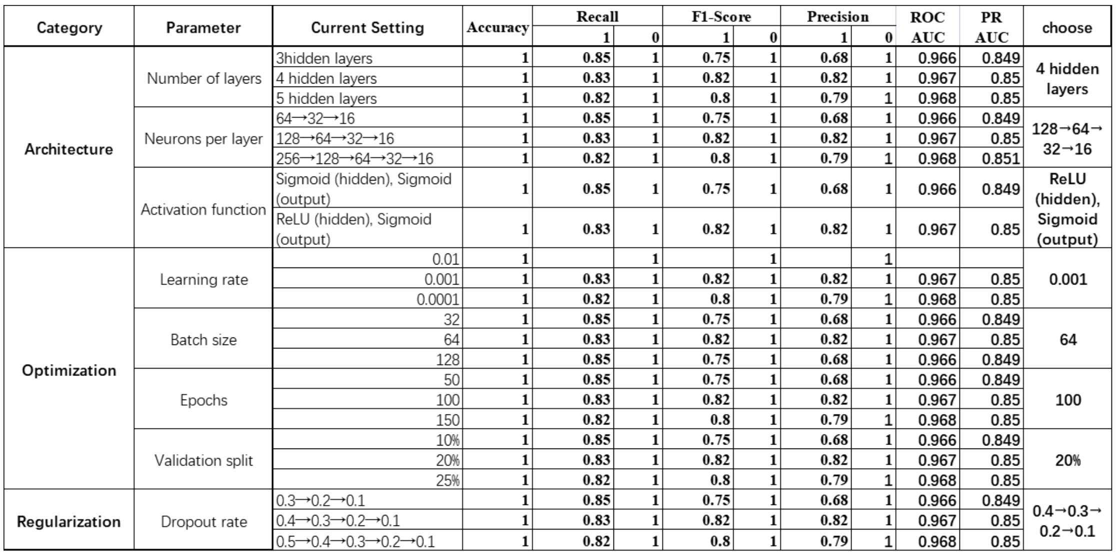
- **Early Stopping**: Monitors the validation AUC and halts training if the performance stops improving for 15 epochs.
- **Learning Rate Scheduling**: Uses the ReduceLROnPlateau scheduler to lower the learning rate when validation AUC plateaus, enabling finer adjustments near the minimum loss.

```
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy', 'AUC', 'Precision', 'Recall'])
    return model
```

4.5 Parameter Adjustment Process

To optimize the model's performance, several parameters were tuned. Below is an overview of the parameters adjusted, the tested values, and the results:

Table 4.1



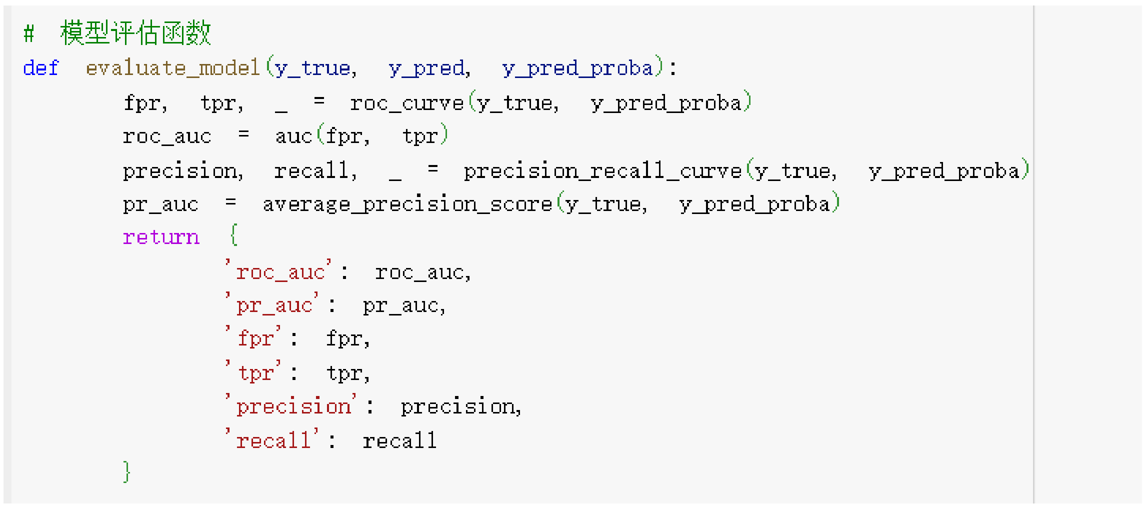
Category	Parameter	Current Setting	Accuracy	Recall		F1-Score		Precision		ROC AUC	PR AUC	choose
				1	0	1	0	1	0			
Architecture	Number of layers	3hidden layers	1	0.85	1	0.75	1	0.68	1	0.966	0.849	4 hidden layers
		4 hidden layers	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
		5 hidden layers	1	0.82	1	0.8	1	0.79	1	0.968	0.85	
	Neurons per layer	64→32→16	1	0.85	1	0.75	1	0.68	1	0.966	0.849	128→64→32→16
		128→64→32→16	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
		256→128→64→32→16	1	0.82	1	0.8	1	0.79	1	0.968	0.851	
	Activation function	Sigmoid (hidden), Sigmoid (output)	1	0.85	1	0.75	1	0.68	1	0.966	0.849	ReLU (hidden), Sigmoid (output)
		ReLU (hidden), Sigmoid (output)	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
Optimization	Learning rate	0.01	1		1		1		1			0.001
		0.001	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
		0.0001	1	0.82	1	0.8	1	0.79	1	0.968	0.85	
	Batch size	32	1	0.85	1	0.75	1	0.68	1	0.966	0.849	64
		64	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
		128	1	0.85	1	0.75	1	0.68	1	0.966	0.849	
	Epochs	50	1	0.85	1	0.75	1	0.68	1	0.966	0.849	100
		100	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
		150	1	0.82	1	0.8	1	0.79	1	0.968	0.85	
	Validation split	10%	1	0.85	1	0.75	1	0.68	1	0.966	0.849	20%
		20%	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
		25%	1	0.82	1	0.8	1	0.79	1	0.968	0.85	
Regularization	Dropout rate	0.3→0.2→0.1	1	0.85	1	0.75	1	0.68	1	0.966	0.849	0.4→0.3→0.2→0.1
		0.4→0.3→0.2→0.1	1	0.83	1	0.82	1	0.82	1	0.967	0.85	
		0.5→0.4→0.3→0.2→0.1	1	0.82	1	0.8	1	0.79	1	0.968	0.85	

Each parameter was systematically adjusted, and the best combination was chosen based on validation performance, focusing on AUC and PR AUC metrics. Considering the results were very similar, we took computation time into account and chose the most efficient optimal model.

4.6 Model Testing and Evaluation

After cross-validation, the final model is trained on the full balanced training dataset and evaluated on a separate test set to assess real-world performance. Key evaluation metrics include:

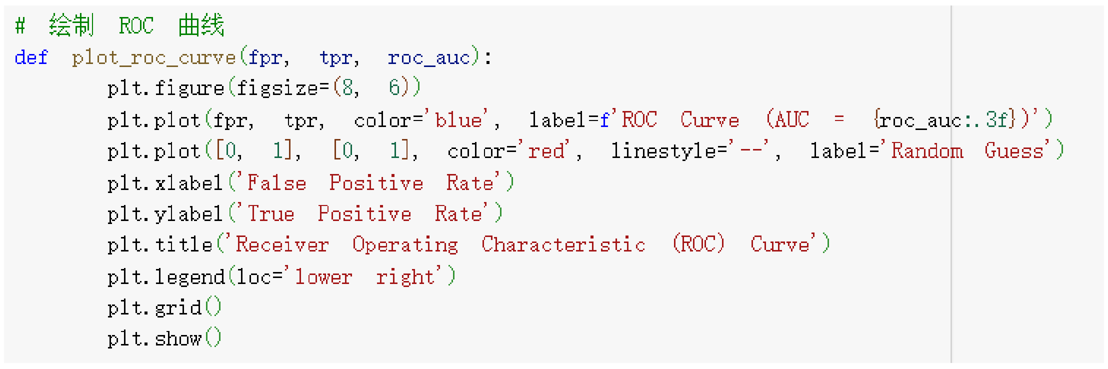
Figure 4.7



```python
# 模型评估函数
def evaluate_model(y_true, y_pred, y_pred_proba):
    fpr, tpr, _ = roc_curve(y_true, y_pred_proba)
    roc_auc = auc(fpr, tpr)
    precision, recall, _ = precision_recall_curve(y_true, y_pred_proba)
    pr_auc = average_precision_score(y_true, y_pred_proba)
    return {
        'roc_auc': roc_auc,
        'pr_auc': pr_auc,
        'fpr': fpr,
        'tpr': tpr,
        'precision': precision,
        'recall': recall
    }
```

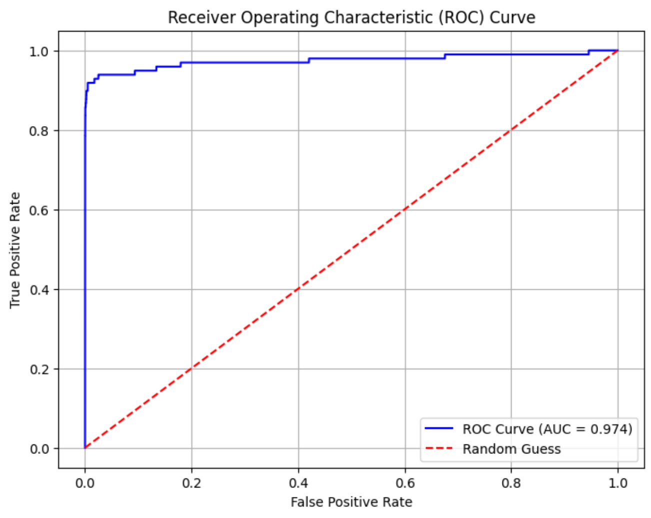
- **ROC AUC**: Measures the model's ability to distinguish between fraudulent and legitimate transactions.

Figure 4.8



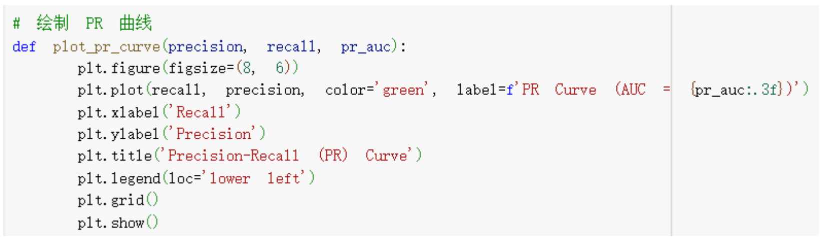
```python
# 绘制 ROC 曲线
def plot_roc_curve(fpr, tpr, roc_auc):
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random Guess')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.grid()
    plt.show()
```

Figure 4.9



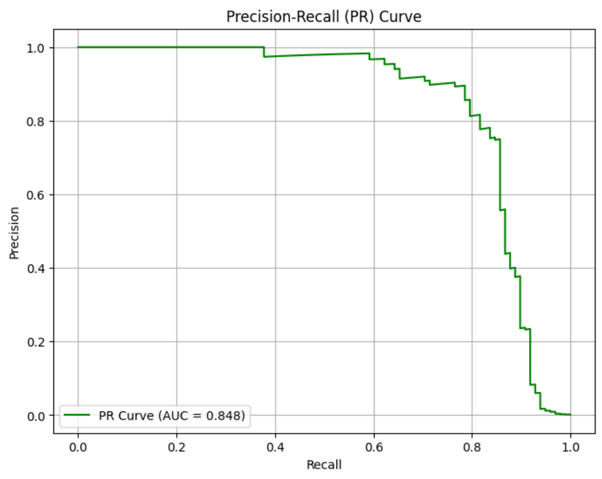
- **PR AUC**: Focuses on precision and recall trade-offs, particularly relevant for imbalanced datasets.

Figure 4.10



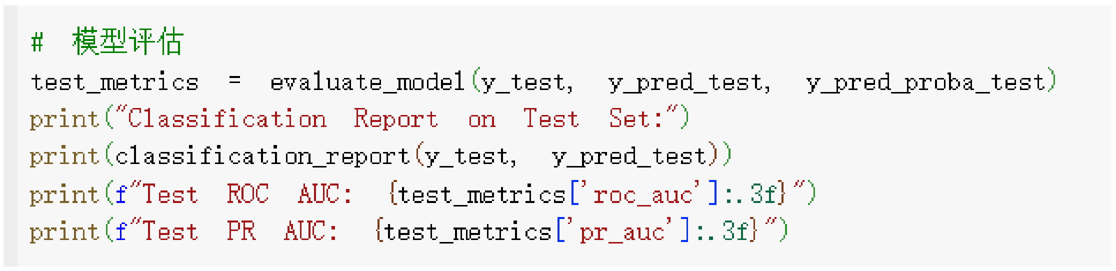
```
# 绘制 PR 曲线
def plot_pr_curve(precision, recall, pr_auc):
        plt.figure(figsize=(8, 6))
        plt.plot(recall, precision, color='green', label=f'PR Curve (AUC = {pr_auc:.3f})')
        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.title('Precision-Recall (PR) Curve')
        plt.legend(loc='lower left')
        plt.grid()
        plt.show()
```

Figure 4.11



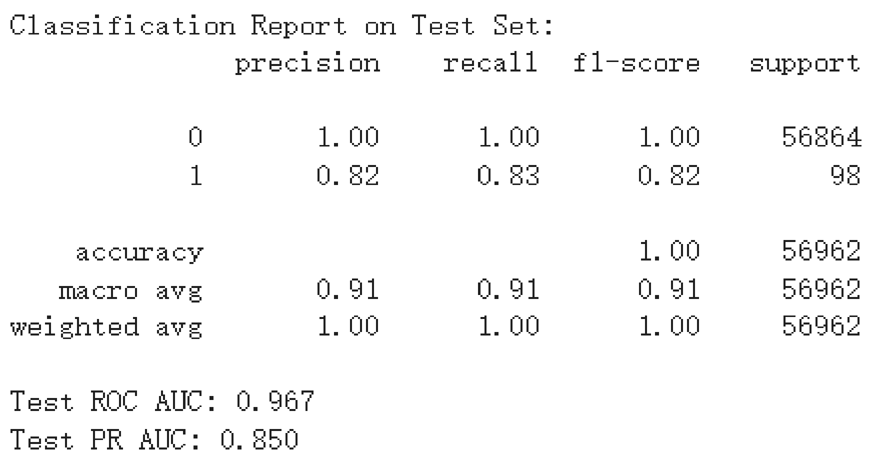
- **Classification Report**: Provides precision, recall, F1-score, and support for each class, highlighting the model's performance on minority (fraudulent) transactions.

Figure 4.12



```
# 模型评估
test_metrics = evaluate_model(y_test, y_pred_test, y_pred_proba_test)
print("Classification Report on Test Set:")
print(classification_report(y_test, y_pred_test))
print(f"Test ROC AUC: {test_metrics['roc_auc']:.3f}")
print(f"Test PR AUC: {test_metrics['pr_auc']:.3f}")
```

Figure 4.13



```
Classification Report on Test Set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.82      0.83      0.82        98

    accuracy                           1.00     56962
   macro avg       0.91      0.91      0.91     56962
weighted avg       1.00      1.00      1.00     56962

Test ROC AUC: 0.967
Test PR AUC: 0.850
```

4.7 Prediction and Thresholding

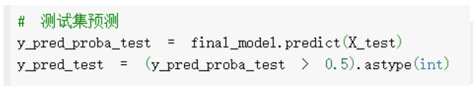
The model outputs probabilities for each transaction. A classification threshold (default: 0.5) is applied to determine whether a transaction is classified as fraudulent or legitimate. Threshold tuning can be performed to balance precision and recall, depending on the application requirements.

And this code converts probabilities into classification results by applying a default threshold of 0.5, here is the Logic Explanation:

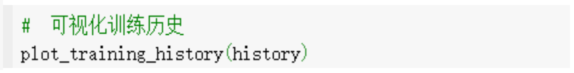
If y_pred_proba_test > 0.5, the sample is classified as 1 (fraudulent).

If y_pred_proba_test <= 0.5, the sample is classified as 0 (legitimate).

Figure 4.14

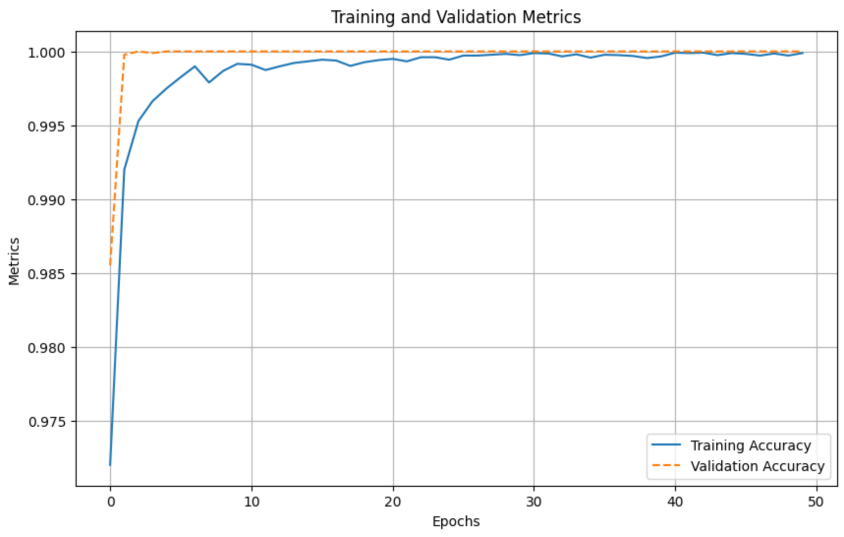


```
# 测试集预测
y_pred_proba_test = final_model.predict(X_test)
y_pred_test = (y_pred_proba_test > 0.5).astype(int)
```



```
# 可视化训练历史
plot_training_history(history)
```

Figure 4.15



4.8 Conclusion

This neural network model leverages effective preprocessing, class balancing, and regularization to address the challenges of credit card fraud detection. The architecture is designed to capture complex, non-linear relationships while minimizing overfitting, and the use of cross-validation ensures robust generalization. By focusing on key metrics like PR AUC, the model prioritizes performance on the minority class, making it well-suited for detecting fraudulent transactions in real-world scenarios. Future results from the parameter adjustment process will provide additional insights into the model’s effectiveness.

5. XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful machine learning algorithm based on gradient boosting. It is widely used for classification and regression tasks due to its efficiency, accuracy, and scalability. XGBoost builds an ensemble of decision trees, where each tree corrects the errors of the previous ones, leading to a strong predictive model.

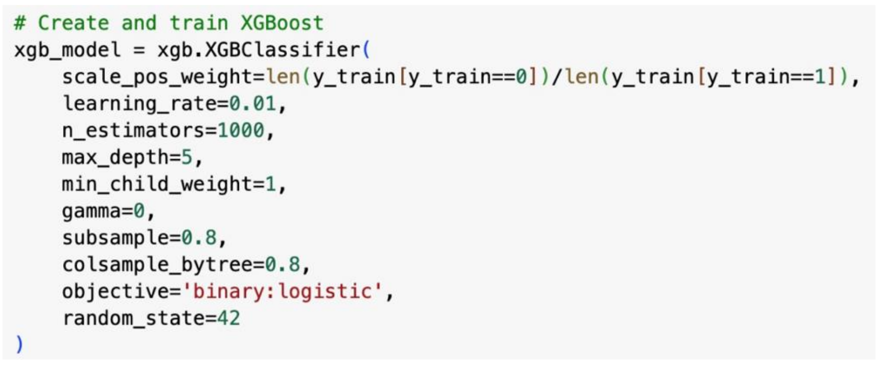
XGBoost can handle imbalance dataset like credit card fraud effectively by using techniques such as weighting classes and adjusting the decision threshold. It is known for its high predictive accuracy, often outperforming other algorithms in various competitions and practical applications, making it a reliable choice for detecting fraud. Additionally, XGBoost is optimized for speed and performance, utilizing parallel processing and efficient memory usage to handle large datasets quickly. The algorithm offers a wide range of hyperparameters that can be tuned to improve model performance, allowing for fine-tuning to better detect fraudulent transactions. Furthermore, XGBoost provides insights into feature importance, helping to understand which features contribute most to detecting fraud, which can be valuable for improving the model and for business insights.

Despite its strengths, XGBoost has some disadvantages. The numerous hyperparameters can make the tuning process complex and time-consuming, requiring extensive experimentation and expertise. XGBoost can also be prone to overfitting, especially with small or noisy datasets, necessitating proper regularization and cross-validation to mitigate this risk. While it provides feature importance, the overall model can be less interpretable compared to simpler models like logistic regression, which can be a drawback when transparency is crucial. Additionally, training XGBoost models can be resource-intensive, requiring significant computational power and memory, especially for large datasets.

5.1 Create and Train XGBoost

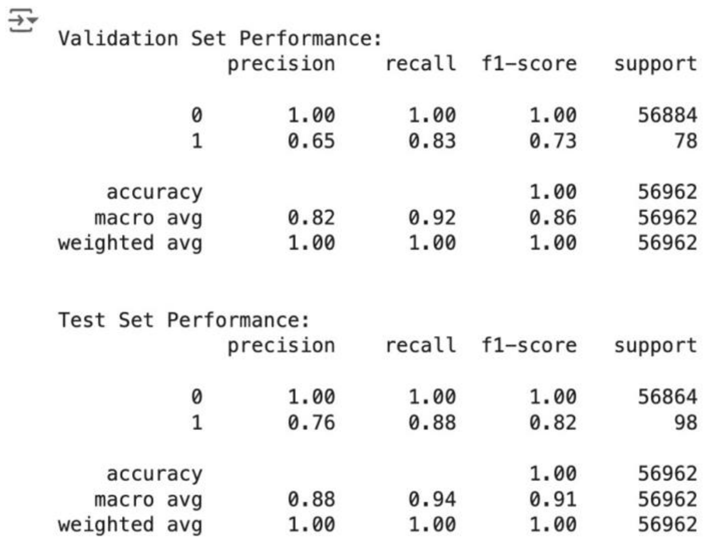
We first set the parameter as figure 4.1 shown and train xgboost model to fit the training set. Then we test it using validation set and test set to see the result, shown in figure 4.2. Among the parameter, scale_pos_weight is set to the ratio of the number of negative samples to positive samples in the training data, helping the model to pay more attention to the minority class (fraudulent transactions).; n_estimators is the number of trees in this model, which means there are total 1000 trees; subsample and colsample are two pareamters that are used to prevent overfitting.

Figure 5.1



```
# Create and train XGBoost
xgb_model = xgb.XGBClassifier(
    scale_pos_weight=len(y_train[y_train==0])/len(y_train[y_train==1]),
    learning_rate=0.01,
    n_estimators=1000,
    max_depth=5,
    min_child_weight=1,
    gamma=0,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='binary:logistic',
    random_state=42
)
```

Figure 5.2 Output from XGBoost Model



```
Validation Set Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.65      0.83      0.73        78

    accuracy                           1.00     56962
   macro avg       0.82      0.92      0.86     56962
weighted avg       1.00      1.00      1.00     56962

Test Set Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.76      0.88      0.82        98

    accuracy                           1.00     56962
   macro avg       0.88      0.94      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

The model performs exceptionally well for Class 0 (Non-Fraudulent), achieving perfect precision, recall, and F1-scores in both validation and test sets. For Class 1 (Fraudulent), the model shows good performance but with some room for improvement. Precision and recall are lower than for Class 0, indicating that while the model is effective at identifying fraudulent transactions, it still has some false positives and false negatives. Overall, the model demonstrates high accuracy and performs well in detecting fraudulent transactions, especially considering the imbalanced nature of the dataset. However, the performance metrics for Class 1 suggest that further tuning or additional techniques might be needed to improve the detection of fraudulent transactions.

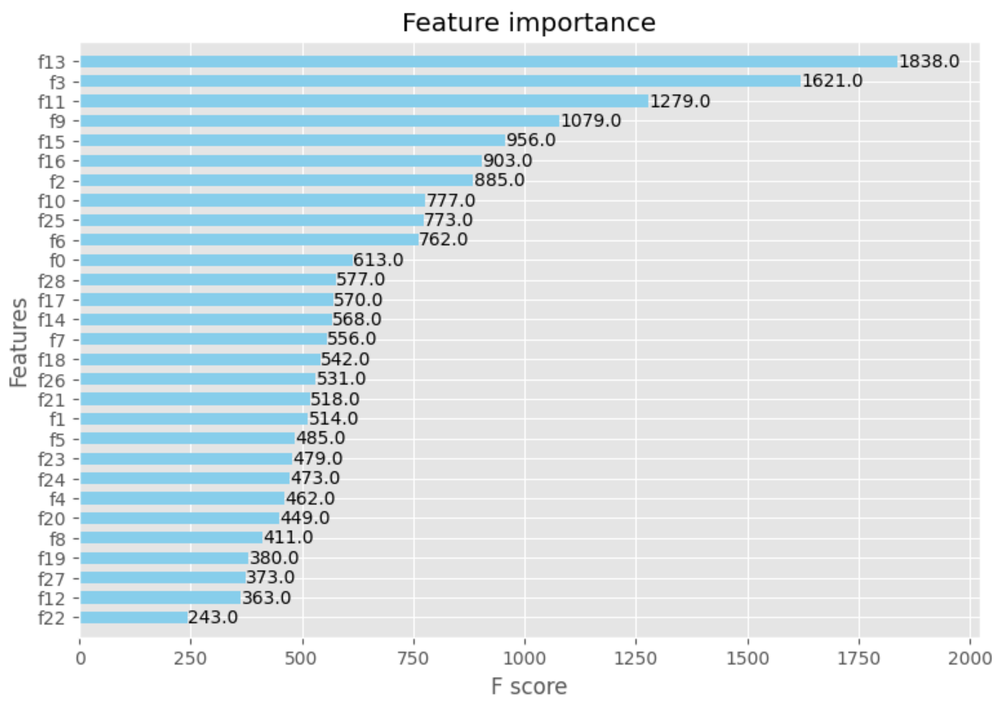
5.2 Model Visualization

5.2.1 Interpreting Feature Importance with plot_importance()

The plot_importance() function in XGBoost is used to visualize the importance of features in the model. There are three different criteria to be chosen to plot the importance scores, such as Weight (The number of times a feature is used to split the data across all trees), Gain (The average improvement in accuracy brought by a feature to the branches it is on), and Cover (The average number of samples affected by splits using this feature).

When using the above model to plot the feature importance, we get the most importance feature f13 with a score of 1838, as shown in figure 4.3. That means the feature f13 is used to split the data across all the 1000 trees 1838 times. With the importance plot, we can easily identify which features are most influential in predicting credit card fraud. By focusing on these key features, we can refine our model and gain a deeper understanding of the patterns in fraudulent transactions.

Figure 5.3 Feature Importance using Weight Criteria

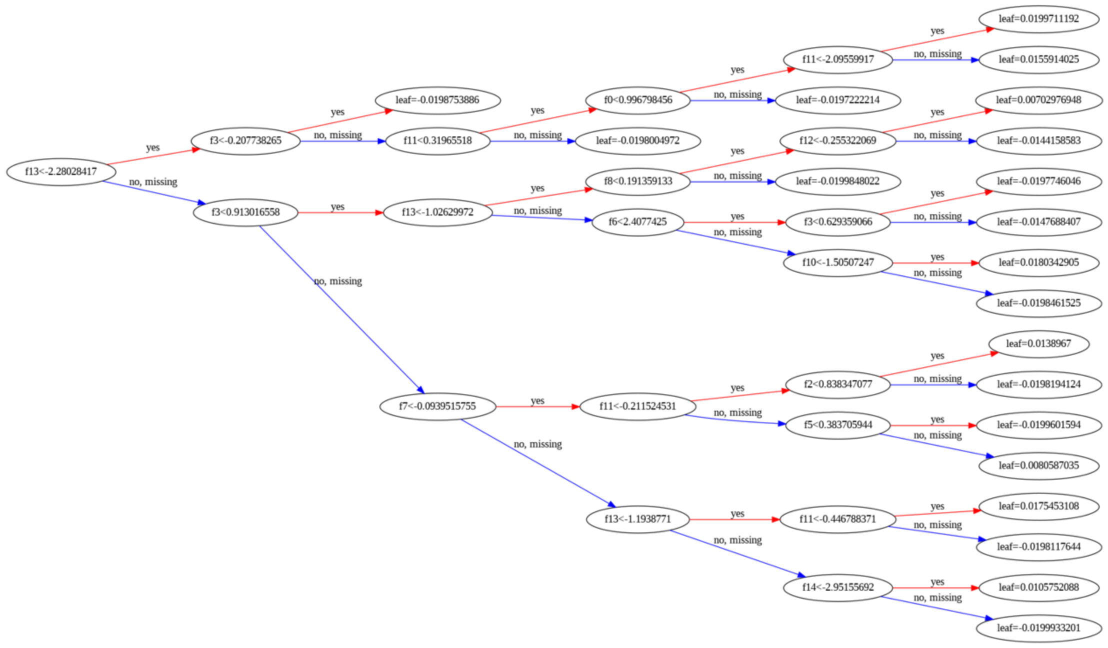


5.2.2 Visualizing Individual Boosted Trees with plot_tree()

The plot provides a detailed illustration of the structure of a single decision tree within the ensemble. Each node signifies a decision based on a specific feature and a corresponding threshold value. By tracing the paths from the root to the leaf nodes, the decision-making process can be comprehensively understood. Analyzing the splits within the tree reveals the features employed and their contributions to the final prediction. The use of plot_tree() facilitates the visualization of individual trees in the ensemble, thereby enhancing the understanding of the model's decision-making process.

Figure 4.4 illustrates the first tree out of the total 1,000 trees in the previous XGBoost model. The root node is identified as feature f13, which reappears in other nodes, totaling three occurrences within this tree. This is reasonable, as feature f13 is highlighted as the most important feature when using plot_importance(). While f13 appears three times, not all the features are represented in this initial tree. By analyzing the tree plot, we can gain a preliminary understanding of how the XGBoost model arrives at its final decision.

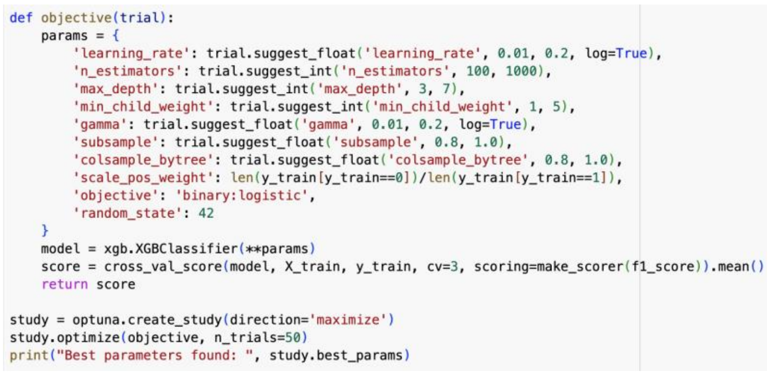
Figure 5.4 The first tree of n_estimators=1000



5.3 Hyperparameter Tuning

This approach uses Optuna, a hyperparameter optimization framework, to find the best hyperparameters for an XGBoost classifier. The objective function defines the hyperparameters to be optimized, including learning_rate, n_estimators, max_depth, min_child_weight, gamma, and etc, which are suggested by the trial object. This method automates the hyperparameter tuning process by running the optimization process for 50 trials, and focusing on the mean F1 score obtained from 3-fold cross-validation on the training set, which is crucial for imbalanced datasets. By efficiently finding the optimal parameters, this approach saves significant time and effort compared to manual tuning.

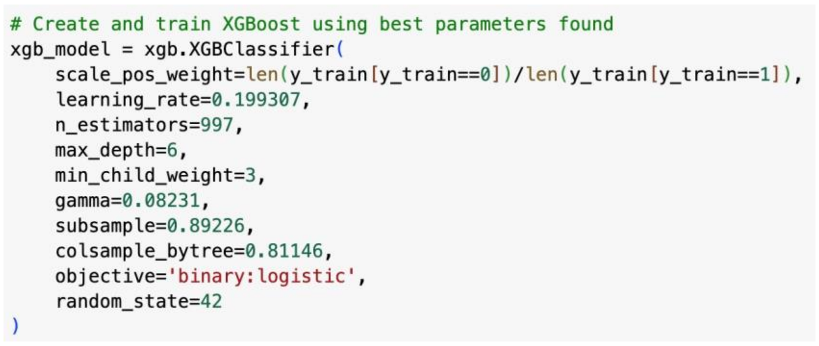
Figure 5.5 Hyperparameter Tuning using Optuna



```python
def objective(trial):
    params = {
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.2, log=True),
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'max_depth': trial.suggest_int('max_depth', 3, 7),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 5),
        'gamma': trial.suggest_float('gamma', 0.01, 0.2, log=True),
        'subsample': trial.suggest_float('subsample', 0.8, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.8, 1.0),
        'scale_pos_weight': len(y_train[y_train==0])/len(y_train[y_train==1]),
        'objective': 'binary:logistic',
        'random_state': 42
    }
    model = xgb.XGBClassifier(**params)
    score = cross_val_score(model, X_train, y_train, cv=3, scoring=make_scorer(f1_score)).mean()
    return score

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)
print("Best parameters found: ", study.best_params)
```

After 50 trials, we identified that the best trial is trial 10, which achieved a mean F1 score of 0.8427 from 3-fold cross-validation on the training set. We then used the optimal hyperparameters from trial 10 to train the model again, shown as figure 4.6.

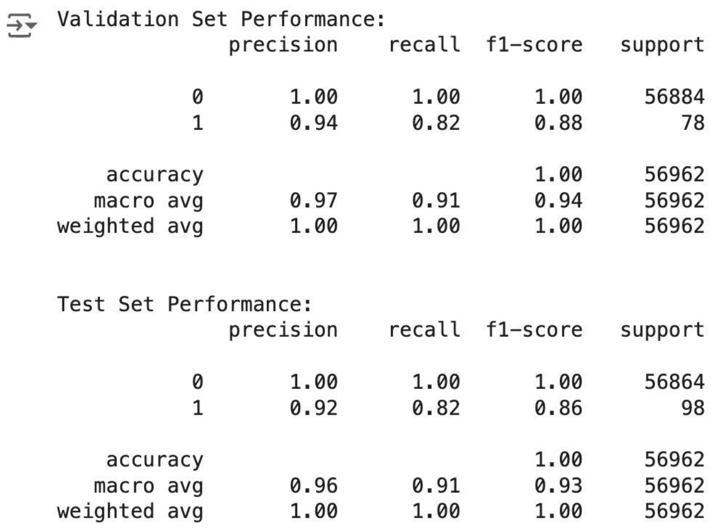
Figure 5.6 Train XGBoost using Optimal Parameters



```
# Create and train XGBoost using best parameters found
xgb_model = xgb.XGBClassifier(
    scale_pos_weight=len(y_train[y_train==0])/len(y_train[y_train==1]),
    learning_rate=0.199307,
    n_estimators=997,
    max_depth=6,
    min_child_weight=3,
    gamma=0.08231,
    subsample=0.89226,
    colsample_bytree=0.81146,
    objective='binary:logistic',
    random_state=42
)
```

Comparing the output before hyperparameter tuning (shown in Figure 4.2) and the output after hyperparameter tuning (shown in Figure 4.7), it is evident that the model's performance improved significantly, especially for class 1 (fraudulent transactions). The precision for class 1 increased from 0.65 to 0.94 on the validation set and from 0.76 to 0.92 on the test set. The F1-score for class 1 also improved from 0.73 to 0.88 on the validation set and from 0.82 to 0.86 on the test set. These improvements indicate that the model is better at correctly identifying fraudulent transactions while maintaining high accuracy for non-fraudulent transactions. The overall macro and weighted averages also show an increase, reflecting the enhanced performance of the tuned model.

Figure 5.7 Output of XGBoost after Hyperparamter Tuning



```
Validation Set Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56884
           1       0.94      0.82      0.88        78

    accuracy                           1.00     56962
   macro avg       0.97      0.91      0.94     56962
weighted avg       1.00      1.00      1.00     56962


Test Set Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.92      0.82      0.86        98

    accuracy                           1.00     56962
   macro avg       0.96      0.91      0.93     56962
weighted avg       1.00      1.00      1.00     56962
```

V. Conclusion

In balanced datasets or those with a small number of samples, using SVM as a classification model is very suitable. This report used a balanced dataset and selected 10,000 non-fraudulent transactions along with all fraudulent ones to apply the SVM model, which showed good performance. The SVM achieved an accuracy of 89.86% on the balanced dataset, demonstrating its strong generalization ability, especially in situations with fewer samples. It can effectively find a suitable decision boundary for good classification results. Additionally, the SVM with RBF kernel was used to handle data that is not linearly separable, and it performed better than the linear SVM on the balanced dataset, particularly for complex data patterns.

In identifying fraudulent transactions (Class 1), the neural network had a precision of 0.89, a recall of 0.81, and an F1 score of 0.84. This shows that the model performs well in predicting fraudulent transactions, but there is still room for improvement. However, the data processing analysis takes a long time, and there is a risk of overfitting.

For the imbalanced and large datasets in the context of this report, both Random Forest and XGBoost have shown very high accuracy and good classification performance in identifying credit card fraud. Both models achieved perfect precision, recall, and F1 scores for non-fraudulent transactions. For fraudulent transactions, the performance of both models was also very close, with precision and F1 scores slightly above 0.8, and recall slightly above 0.8, indicating that while there is room for improvement in identifying fraudulent transactions, the overall performance is quite good.

However, according to the ROC-AUC scores depicted in the images, the XGBoost model has a higher ROC-AUC score than the Random Forest model. The XGBoost model's ROC-AUC score on the test set is 0.98, which is higher than the Random Forest model's score of 0.97, indicating that XGBoost has a better ability to distinguish between classes.

Therefore, based on the analysis, XGBoost appears to be the better model for this task.

APPENDIX(COMPARISION)

IV. Approach and Results' Comparison of all model evaluation metrics:

Classification methods			**Accuracy**	**Recall**	**F1-Score**	**Precision**
2.1 Linear SVM Classification [Balanced data]		**0.898649**	0.806667	0.889706	/
	0== non-fraudulent data (support:146 test30%)		0.993151	0.906250	0.833333
	1== Fraud data (support:150 test30%)		0.806667	0.889706	0.991803
	Macro Avg(support:296)		0.899099	0.897976	0.912568
	Weighted Avg Weighted average by sample size		0.898649	0.897866	0.913639
2.2 RBF kernel SVM Classification		**0.93**	**Recall**	**F1-Score**	**Precision**
①RBF: Initial Model without hyper parameterization [Balanced data]	0== non-fraudulent data (support:87 test25%)		0.99	0.93	0.88
	1== Fraud data (support:110 test25%)		0.89	0.94	0.99
	Macro Avg(support:197)		0.94	0.93	0.93
	Weighted Avg		0.93	0.93	0.94
RBF: Optimizing the RBF model by Cross-Validation with the best hyperparameter C=100 & γ=0.01 [Balanced data]		**0.92**	**Recall**	**F1-Score**	**Precision**
	0== non-fraudulent data (support:87 test25%)		0.95	0.91	0.87
	1== Fraud data (support:110 test25%)		0.89	0.92	0.96
	Macro Avg(support:197)		0.92	0.92	0.92
	Weighted Avg		0.92	0.92	0.92
②RBF: Down-sampling	Validation set	**0.986**	**Recall**	**F1-Score**	**Precision**

non-fraudulent random samples (10,000 entries) merged with all fraud samples (492 entries) the best hyperparameter is C=1 & γ=0.0001 [Imbalanced data]	0== non-fraudulent data (support:1998 test25%)		1.0	1.0	0.99
	1== Fraud data (support:101 test25%)		0.75	0.9	0.96
	Macro Avg(support:2099)		0.92	0.95	0.97
	Weighted Avg		0.99	0.99	0.99
	Test set	**0.99**	**Recall**	**F1-Score**	**Precision**
	0== non-fraudulent data (support:1998 test25%)		1.0	1.0	0.99
	1== Fraud data (support:101 test25%)		0.85	0.9	0.96
	Macro Avg(support:2099)		0.92	0.95	0.97
	Weighted Avg		0.99	0.99	0.99
3.Random Forest(n_estimators=300, random_state=42)	Validation set	0.999	**Recall**	**F1-Score**	**Precision**
	0== non-fraudulent data (support:1998 test25%)		1.00	1.00	1.00
	1== Fraud data (support:101 test25%)		0.78	0.83	0.88
	Macro Avg(support:2099)		0.89	0.91	0.94
	Weighted Avg		1.00	1.00	1.00
	Test set	0.999	**Recall**	**F1-Score**	**Precision**
	0== non-fraudulent data (support:1998 test25%)		1.00	1.00	1.00
	1== Fraud data (support:101 test25%)		0.81	0.84	0.89
	Macro Avg(support:2099)		0.90	0.92	0.94
	Weighted Avg		1.00	1.00	1.00
4.Neural Network (4	Validation set	1.0	**Recall**	**F1-Score**	**Precision**

hidden layers, lr=0.001, Batch size=64, Epochs=100)	0== non-fraudulent data (support:1998 test25%)	1.00	1.00	1.00
	1== Fraud data (support:101 test25%)	0.86	0.85	0.82
	Macro Avg(support:2099)	0.93	0.93	0.94
	Weighted Avg	1.00	1.00	1.00
	Test set 1.0	**Recall**	**F1-Score**	**Precision**
	0== non-fraudulent data (support:1998 test25%)	1.00	1.00	1.00
	1== Fraud data (support:101 test25%)	0.83	0.83	0.82
	Macro Avg(support:2099)	0.91	0.91	0.91
	Weighted Avg	1.00	1.00	1.00
5. XGBoost (learning_rate=0.199307, n_estimators=997, max_depth=6, min_child_weight=3, gamma=0.08231, subsample=0.89226, colsample_bytree=0.81146, objective='binary:logistic')	Validation set 1.0	**Recall**	**F1-Score**	**Precision**
	0== non-fraudulent data (support:56884)	1.00	1.00	1.00
	1== Fraud data (support:78)	0.94	0.82	0.88
	Macro Avg(support:56962)	0.97	0.91	0.94
	Weighted Avg	1.00	1.00	1.00
	Test set	1.0	**Recall**	**F1-Score**
	0== non-fraudulent data (support:56864)	1.00	1.00	1.00
	1== Fraud data (support:98)	0.92	0.82	0.86
	Macro Avg(56962)	0.96	0.91	0.93
	Weighted Avg	1.00	1.00	1.00

REFERENCES:

Hull, John C. *Machine Learning in Business: An Introduction to the World of Data Science.* 2nd ed., 2019, 2020. ISBN: 9798644074372.

MLG-ULB. *Credit Card Fraud Detection Dataset.* Kaggle, n.d., https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud. Accessed 6 Dec. 2024.