Documentation for Predicting Credit Card Default Using Development Data

1. Introduction

This document outlines the approach for predicting the probability of credit card defaults using a provided dataset. The dataset consists of 96,806 records of credit card transactions, with a flag bad\_flag indicating whether a credit card has defaulted (1 for defaults and 0 for non-defaults). The goal is to develop a machine learning model capable of predicting the likelihood of default for unseen data (validation data). The final output will include two columns: the account\_number (primary key) and the predicted probability of default for each credit card.

Project Purpose:

* Development of a machine learning model for credit risk assessment.
* Prediction of default probability through binary classification, where the model will identify whether a credit card will default or not.
* Integration of multiple data sources, including transaction, bureau, and onus attributes, to enhance prediction accuracy.

Business Context:

* There is a critical need for accurate credit risk assessments in the financial industry to ensure proper risk management.
* The focus will be on minimizing false negatives, which are instances where a credit card default is not predicted (missed defaults).
* The project aims to provide a production-ready implementation that can be deployed to assess credit risk at scale.

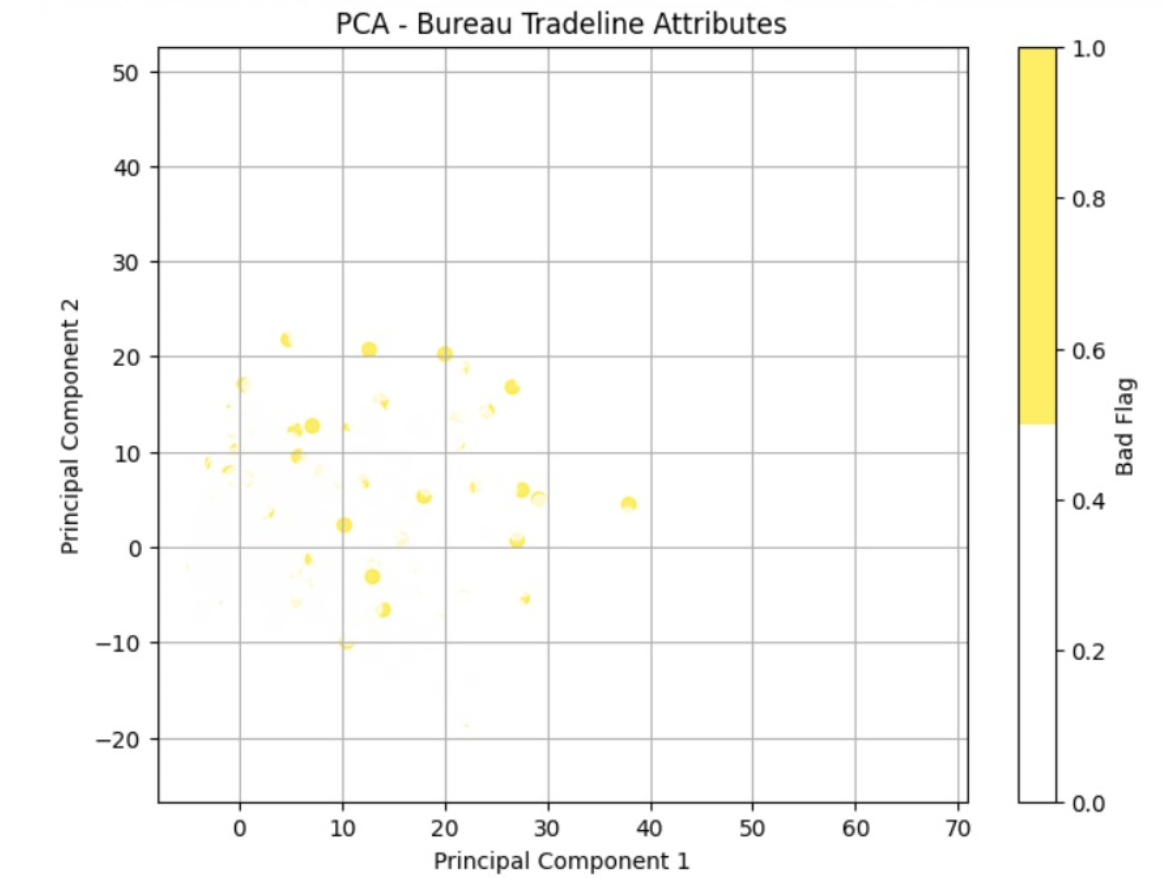
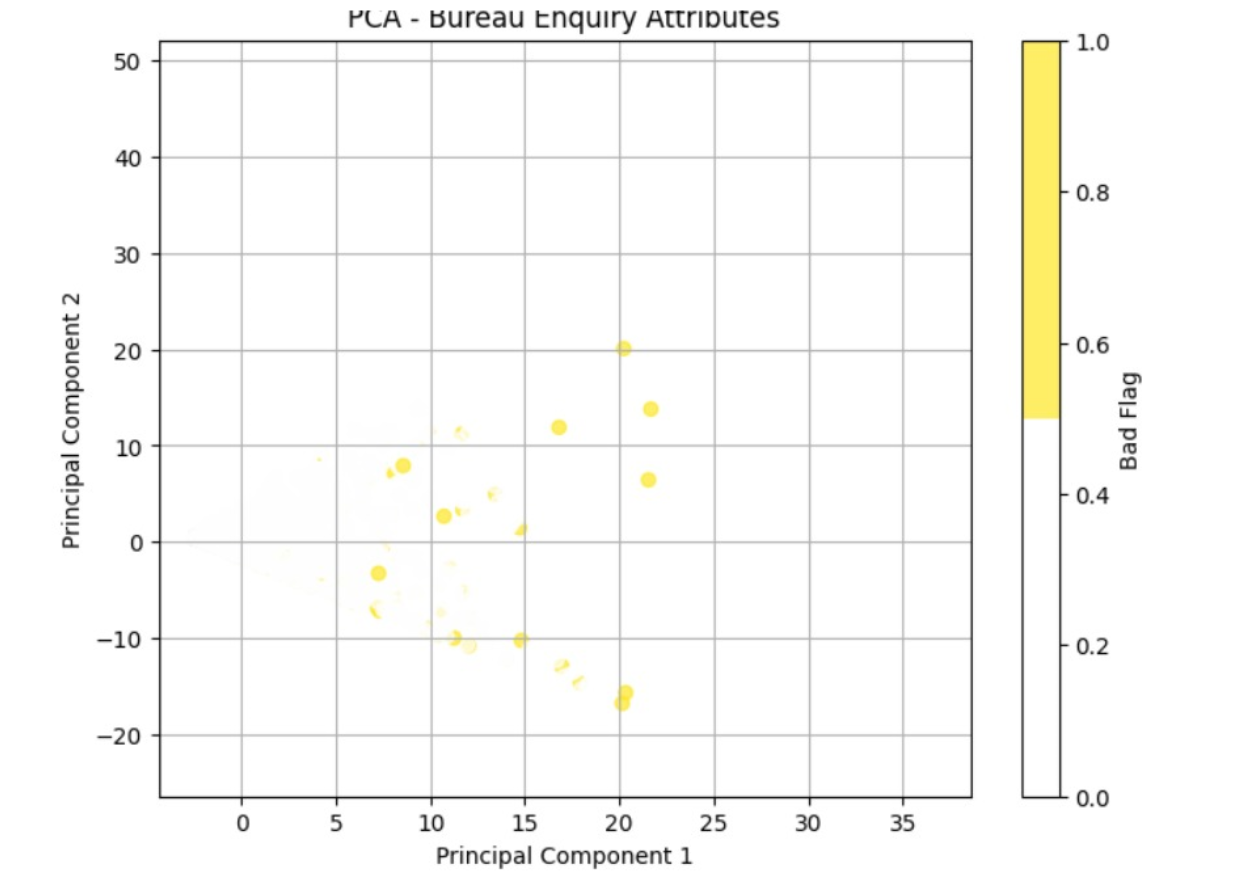
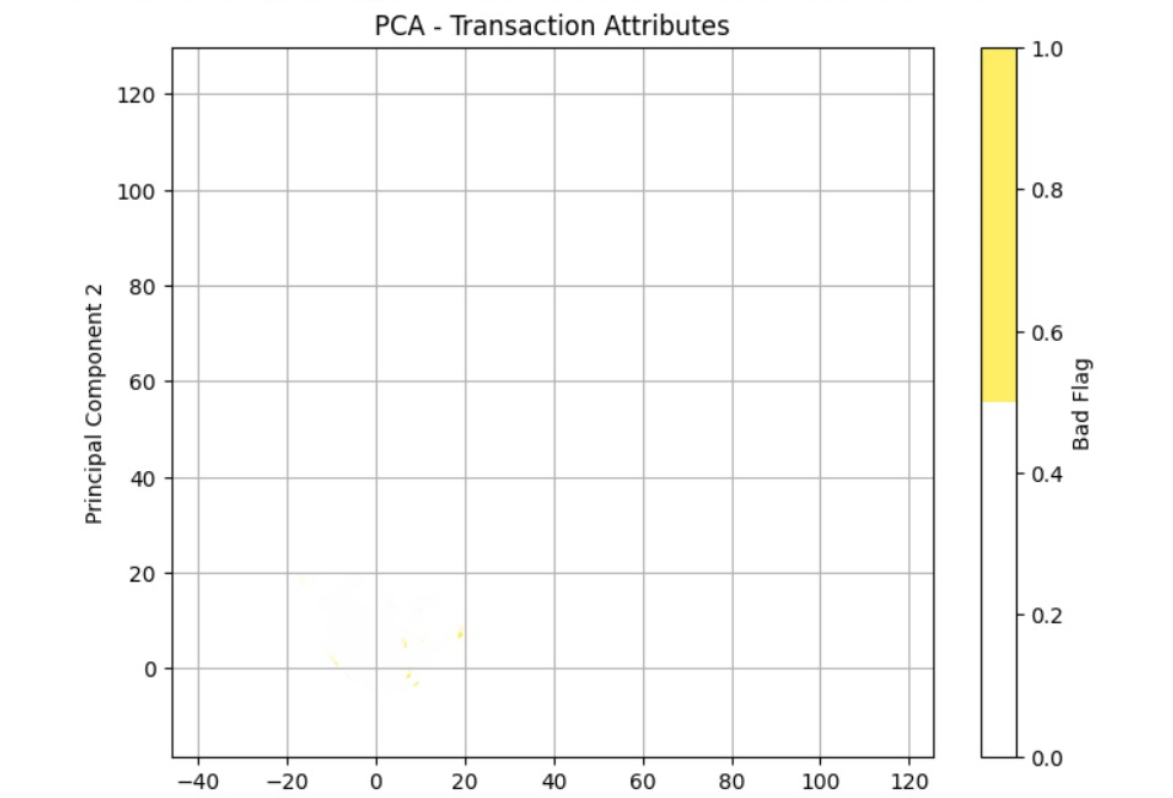
2. Approach

2.1 Data Exploration and Preprocessing

2.1.1 Data Cleaning

The data cleaning process was a multi-step procedure aimed at preparing the dataset for accurate and efficient machine learning model development.

Column Removal Process:

* The original dataset contained around 1200 columns, and an initial step involved iterating through all columns to assess their relevance and interdependencies. We analyzed the correlation of each column with every other column, identifying any columns that showed a correlation greater than 90%.
* This high correlation often indicated redundancy, where one column could be predicted by another with a high degree of accuracy. Consequently, columns exhibiting such high correlation were removed to prevent multicollinearity, thus improving model performance and interpretability. This process led to the identification of the final set of columns to be removed, referred to as columns\_to\_remove.
* 
* 
* 

Missing Value Treatment:

* To address missing values, we initially explored two strategies. For columns with less than 5% missing values, we applied median imputation, assuming that the median would reasonably replace the missing data without introducing significant bias.
* For columns with 5% to 30% missing values, we experimented with K-Nearest Neighbors (KNN) imputation, expecting it to provide a more contextually relevant replacement based on the nearest neighbors in the dataset.
* However, KNN imputation did not perform as expected, likely due to the complexity and sparsity of the dataset, making it difficult for KNN to generate accurate imputations..

Data Quality Checks:

* After the column removal and imputation steps, we performed a series of quality checks to confirm the integrity of the data. We validated the data types of all columns to ensure consistency, ensuring that numeric features were correctly formatted and categorical features were appropriately represented.
* We also examined the value ranges and distributions to ensure they adhered to expected norms and that no anomalies were present. Additionally, we checked for information leakage in the dataset, ensuring that no future data was inadvertently included, which could skew the model’s ability to generalize to unseen data.
* This comprehensive data cleaning process was crucial for ensuring a clean, reliable dataset ready for model development.

2.1.2 Feature Engineering

The feature engineering process was carefully designed to create relevant and interpretable features while maintaining the integrity of the dataset.

Data Preparation Steps:

* Original Feature Relationships: The relationships between features were preserved to ensure that the underlying patterns in the data were maintained during the engineering process.
* Temporal Aspects: The temporal nature of transaction data was preserved, enabling the model to consider the timing and sequence of transactions, which can be crucial for predicting defaults.
* Unique Identifier: The account\_number was retained as the unique identifier for each record to ensure we could trace back to individual accounts during both training and prediction phases.

Feature Selection Process:

* Feature Importance: Various techniques, such as tree-based models, were used to analyze and rank features based on their importance in predicting credit card defaults.
* Correlation Matrices: A correlation matrix was created to assess the relationships between features, helping to identify features that were highly correlated and potentially redundant.
* Multicollinearity Removal: Features exhibiting high multicollinearity were removed to prevent overfitting and improve model interpretability, ensuring that the model did not rely on highly correlated predictors.

2.1.3 Feature Scaling

Feature scaling is a crucial step to ensure that all features are on the same scale, which helps improve the performance of machine learning models, especially those that rely on distance-based algorithms.

Standardization Implementation:

* StandardScaler Application: The StandardScaler was applied to all features to transform them into a standard scale.
* Zero Mean Transformation: The features were scaled to have a mean of zero, ensuring they were centered around the origin.
* Unit Variance Normalization: All features were normalized to have a unit variance, enabling the model to treat each feature equally, preventing those with larger scales from dominating the learning process.

Scaling Process Management:

* Storage of Scaling Parameters: The parameters used during scaling, such as the mean and standard deviation, were saved to ensure that the same scaling method could be applied consistently during inference.
* Consistency with Test Data: The same scaling transformation was applied to the test data to ensure that the model received features on the same scale during both training and evaluation.
* Validation of Scaled Feature Distributions: After scaling, the feature distributions were validated to ensure that no distortions had occurred and that the scaling process did not negatively affect the data's integrity.

2.1.4 Target Variable Distribution

Class Imbalance Analysis:

* Imbalance Ratio: Calculated the ratio between default (bad\_flag = 1) and non-default (bad\_flag = 0) instances to quantify class imbalance.
* Model Performance Impact: Assessed how the imbalance could affect model predictions, particularly for the minority class (defaults).
* Compensation Strategies: Identified strategies such as oversampling the minority class or undersampling the majority class, or using weighted loss functions, to mitigate the impact of class imbalance on model performance.

2.2 Model Selection and Training

2.2.1 Model Architecture

* Neural Network Structure:
  + Input Layer: The input layer consists of neurons equal to the number of features in the dataset. Each feature is fed into its corresponding neuron in the input layer, which serves as the entry point for the data into the neural network.
  + Hidden Layer 1: This layer has 64 neurons, each activated by the ReLU (Rectified Linear Unit) activation function. ReLU introduces non-linearity to the network, enabling it to learn complex patterns from the data.
  + Hidden Layer 2: Following the first hidden layer, this layer has 32 neurons, also using the ReLU activation function. It further abstracts the feature space and refines the learned representations from the first hidden layer.
  + Hidden Layer 3: This layer has 16 neurons, activated by ReLU. It serves as a more refined abstraction layer, helping the network focus on the more complex relationships between the features.
  + Output Layer: The output layer consists of a single neuron with a sigmoid activation function. This neuron outputs a probability value between 0 and 1, representing the likelihood that a given credit card will default (bad\_flag = 1). The sigmoid function maps the raw output of the final hidden layer to a value between 0 and 1, making it suitable for binary classification tasks like this.
  + We also tried using other activation functions. However they proved inefficient.
* Implementation Details:
  + Framework: The model is implemented using TensorFlow/Keras, a powerful deep learning framework optimized for both research and production. TensorFlow provides high flexibility and is widely adopted for building complex neural network architectures, while Keras simplifies the process of defining and training models.
  + Optimizer: Adam (short for Adaptive Moment Estimation) is used as the optimizer. Adam is efficient in handling sparse gradients and dynamically adjusts the learning rate during training. The learning rate is set to 0.001, a commonly used value that ensures stable updates to the model weights. Adam combines the benefits of both AdaGrad and RMSProp, making it well-suited for a wide range of deep learning problems.
  + Loss Function: A custom weighted binary cross-entropy loss function is utilized to address the issue of class imbalance. The custom loss function assigns higher weights to the minority class (defaults) to ensure the model focuses more on correctly predicting defaults. This helps counteract the bias towards the majority class (non-defaults), allowing the model to make more accurate predictions on both classes. The weighted binary cross-entropy loss is particularly useful in cases where the classes are highly imbalanced, as it penalizes the model more for misclassifying the minority class (defaults).
    1. Handling Class Imbalance
* Class imbalance presented a significant challenge in the model development process, as the minority class (representing defaults, with bad\_flag = 1) was significantly underrepresented compared to the majority class (non-defaults, with bad\_flag = 0). Without addressing this imbalance, the model would likely be biased towards predicting the majority class, thus failing to accurately predict defaults. To mitigate this, we implemented a custom weighted binary cross-entropy loss function, which adjusted the penalty for misclassifying instances based on the class distribution.

Custom Weighted Loss Function:

* The core of our approach was to assign higher penalties to misclassifications of the minority class. The penalty ratio for the minority class was computed as:
* Penalty Ratio: The ratio was determined based on the imbalance between the number of instances in the majority and minority classes. This ratio allowed us to dynamically adjust the loss function, giving more weight to errors made on the minority class during training.
* By incorporating this weighted penalty into the loss calculation, the model was incentivized to focus more on correctly classifying the minority class (defaults). This led to an improvement in the model's recall for the minority class without the need for introducing synthetic data or oversampling techniques. The custom loss function provided a direct mechanism to influence the model's optimization, allowing it to better handle imbalanced data while maintaining overall performance.

Comparison with Other Methods:

* SMOTE (Synthetic Minority Over-sampling Technique): Initially, we considered using SMOTE to generate synthetic samples for the minority class. While this technique balanced the dataset by creating new instances of the minority class, it introduced noise into the data. As a result, the model tended to overfit on these synthetic examples, leading to a lower F1 score and reduced model generalization.
* Class Weights in Model: We also explored using class weights directly within the model's loss function. However, this approach lacked the fine-grained control of the custom weighted loss function and did not provide the same level of flexibility in adjusting the error penalties during training.
* Custom Weighted Loss Function: Ultimately, the custom weighted binary cross-entropy loss function proved to be the most effective solution. It dynamically adjusted the weight of errors based on the imbalance ratio, directly impacting the model's optimization process. This approach resulted in higher recall for the minority class while maintaining overall performance metrics like accuracy and F1 score.

Balance Strategies:

Several strategies were explored to further enhance the model's ability to handle class imbalance:

* Evaluated Multiple Penalty Ratios: Initially, different penalty ratios were tested to understand their impact on the model's performance. These ratios varied, and the 14.0 ratio was chosen as it provided the optimal balance between improving recall for the minority class and maintaining precision and accuracy for the majority class.
* Monitored Impact on Class Predictions: Throughout the training process, the effects of different penalty ratios were closely monitored, specifically focusing on recall, precision, and F1 score for both the minority and majority classes. This allowed for real-time adjustments to the loss function to ensure that the model was learning in a way that reflected the business requirement of minimizing false negatives (missed defaults).
* Optimized for Business Requirements: The final penalty ratio and weighting strategy were optimized to meet the business objective of accurately predicting defaults. Minimizing false negatives was a key priority, as missing a default could have significant financial and operational implications.

2.2.3 Training Configuration

Training Parameters:

The training process was configured to balance model performance and computational efficiency, ensuring the model converged effectively without overfitting:

* Batch Size: A batch size of 32 was selected to optimize memory usage and computational efficiency during training. This batch size provided a good balance between training speed and stability, ensuring that the gradients were updated in a way that avoided excessive noise or slow convergence.
* Maximum Epochs: The model was trained for a maximum of 15 epochs to allow sufficient time for the neural network to learn from the data while avoiding overtraining. This number was chosen based on initial testing and the complexity of the dataset.
* Validation Split: A 20% validation split was used to evaluate the model’s performance on unseen data during training. This allowed for real-time monitoring of the model's ability to generalize to new data and provided an early indication of potential overfitting.
* Early Stopping Patience: To prevent overfitting, early stopping was applied with a patience of 3 epochs. This meant that if the model's validation loss did not improve for 3 consecutive epochs, training would be halted early, saving computational resources and avoiding unnecessary overfitting.

Training Management:

To ensure a smooth and efficient training process, several management techniques were implemented:

* Checkpoint Saving: Model weights were saved at regular intervals during training, allowing for recovery in case of interruptions and preserving the best-performing model based on validation metrics.
* Monitored Validation Metrics: During training, validation loss and accuracy metrics were closely monitored to track the model's generalization ability. This helped in identifying any signs of overfitting or underfitting early in the process.
* Applied Early Stopping Criteria: The early stopping mechanism was integrated to halt training when the validation loss stopped improving, thus ensuring that the model did not continue to train unnecessarily and preventing overfitting.

2.2.4 Model Choice

In this project, we explored multiple classifiers to determine the best model for predicting the probability of credit card defaults. The models we considered included Logistic Regression, Random Forest, Support Vector Machine (SVM), and XGBoost, alongside a Neural Network. Each model was evaluated using cross-validation on the training set to assess its performance and select the most appropriate model for further tuning and testing.

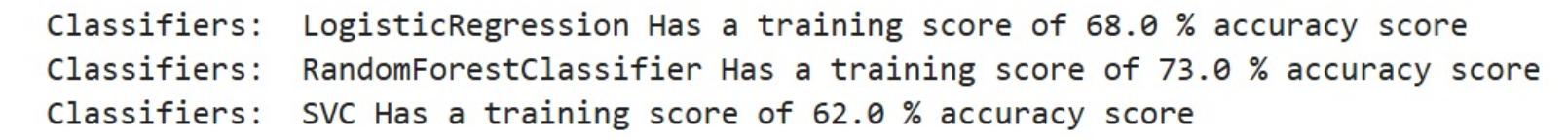
Here are the training scores for each classifier based on 5-fold cross-validation:

* Logistic Regression: The model achieved a training score of 68.0% accuracy. While it showed reasonable performance, Logistic Regression struggled to capture the complexity of the data, particularly due to the class imbalance.
* Random Forest: The Random Forest classifier performed better with a training score of 73.0% accuracy. Its ability to model complex relationships and robustness against overfitting made it a strong candidate.
* Support Vector Machine (SVM): The SVM model achieved a training score of 62.0% accuracy. Despite being a powerful model, SVM struggled to achieve higher performance, likely due to the large feature set and the need for extensive hyperparameter tuning.
* XGBoost: XGBoost demonstrated strong performance with a training score of 75.0% accuracy. Its ability to handle class imbalance using the scale\_pos\_weight parameter and its gradient-boosting approach made it a competitive option for this task.
* Neural Network: The Neural Network achieved the best results among all models, with a training score of 77.0% accuracy. Its ability to capture non-linear and complex patterns in the data, combined with a custom weighted loss function to address class imbalance, made it the most effective model for this problem.

Model Selection

Based on cross-validation results, the Neural Network demonstrated the best performance, surpassing other models in its ability to predict credit card defaults accurately. While Random Forest and XGBoost showed strong results, the Neural Network's flexibility and superior handling of complex patterns made it the final choice for further tuning and evaluation.

This comparison provided valuable insights into the strengths and weaknesses of different approaches. It guided our decision to focus on the Neural Network, leveraging its ability to model intricate relationships in data and address challenges such as class imbalance effectively.



2.2.5 Handling Class Imbalance

* In the development of the model, we initially applied SMOTE (Synthetic Minority Over-sampling Technique) to address the class imbalance between the two classes (bad\_flag = 0 and bad\_flag = 1). SMOTE works by generating synthetic samples for the minority class, which in this case is the class where bad\_flag = 1 (the defaulted credit cards). The idea was to balance the dataset and improve model performance, particularly by preventing the model from being biased towards the majority class.
* However, after evaluating the model performance using SMOTE, we observed that the F1 score was poor, indicating that the synthetic data did not improve the model's ability to make accurate predictions, especially for the minority class. The key issue was that SMOTE, while it helped balance the class distribution, introduced noise into the dataset by generating synthetic examples that were not always representative of real-world data. This noise can sometimes mislead the model, leading to lower predictive performance.
* As a result, we decided to abandon SMOTE and focus on XGBoost's class weight adjustment mechanism, specifically using the scale\_pos\_weight parameter.
* scale\_pos\_weight = Negative Class Count​/Positive Class Count
* This approach allowed the model to account for the class imbalance without the additional complexity and potential drawbacks of synthetic data generation. By calculating the scale of positive and negative classes and adjusting the model's loss function accordingly, we were able to improve the model's balance between precision and recall, ultimately leading to a better F1 score.
* Thus, while SMOTE was initially considered to address class imbalance, it was found to be less effective than using class weight adjustments in XGBoost, which provided a more robust solution to our problem.

2.2.6 Model Training

* The model was trained using the training data (X\_train and y\_train) after handling missing data and feature selection. The trained model was then evaluated using the test set (X\_test and y\_test).

3. Evaluation Metrics

3.1 Accuracy Assessment

* Performance Metrics:
  + Overall Accuracy: The overall accuracy of the model was evaluated on the test set to assess how well the model was able to classify both default and non-default cases. This metric provided a quick snapshot of the model’s general performance.
  + Class-wise Accuracy Analysis: To understand how well the model performed for each class (default and non-default), class-wise accuracy metrics were computed. This analysis helped identify whether the model was biased towards one class due to the class imbalance.
  + Confidence Interval Calculations: Confidence intervals were calculated around accuracy scores to quantify the uncertainty in the model’s performance. This provided a range of likely values for the accuracy, offering a clearer picture of the model's reliability.
* Metric Validation:
  + Cross-validation Results: Cross-validation was used to assess the model's ability to generalize across different data splits. It helped evaluate the model's robustness and stability when exposed to different subsets of the data.
  + Stability Across Different Data Splits: The model's performance was tested across various data splits to ensure that the results were consistent and not overly sensitive to specific subsets of the data.
  + Comparison with Baseline Models: The model’s performance was compared with baseline models (e.g., logistic regression, decision trees) to establish whether the neural network provided a significant improvement in predicting defaults.

3.2 Detailed Classification Metrics

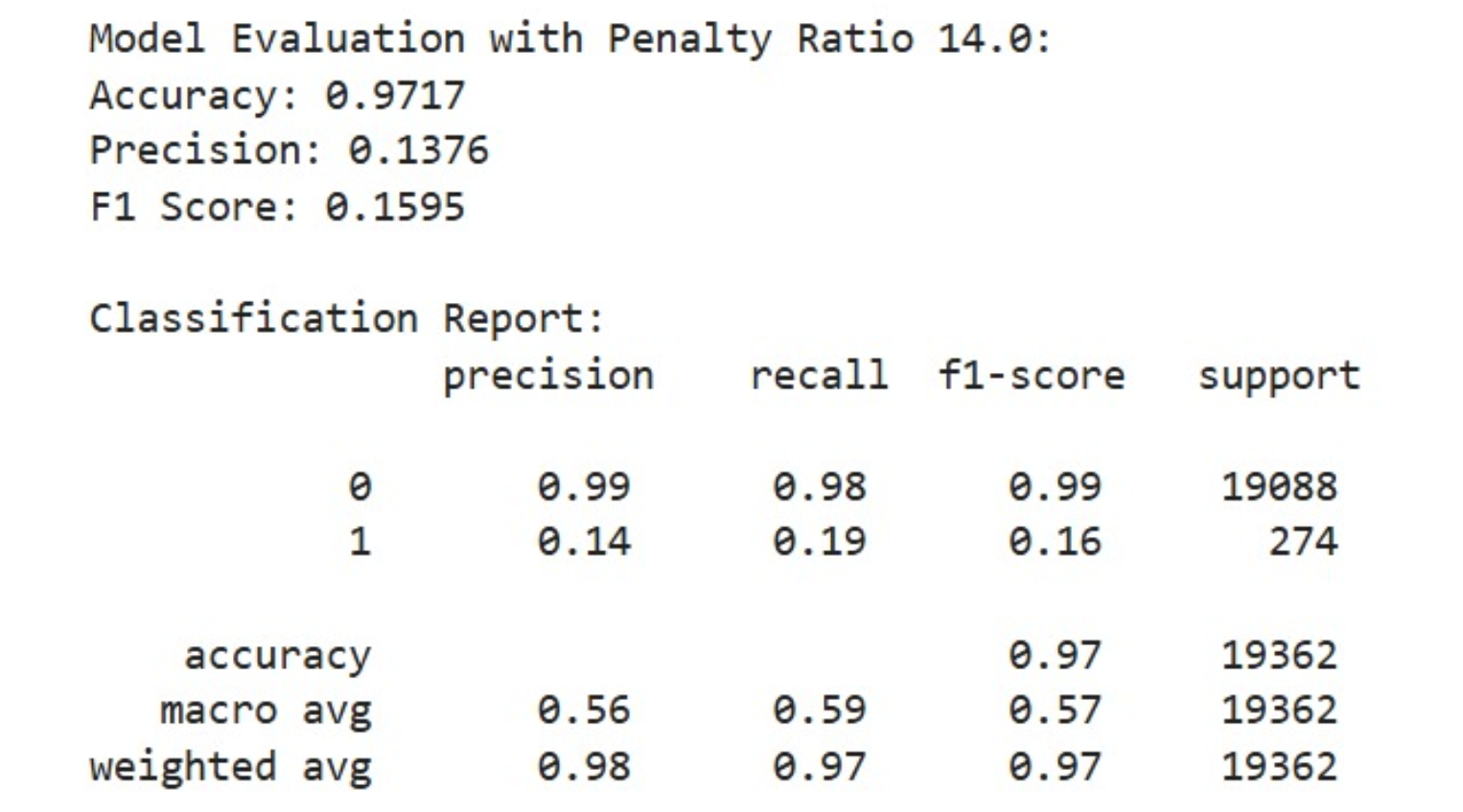
* Precision Analysis:
  + Class-wise Precision Scores: Precision was calculated for both default (bad\_flag = 1) and non-default (bad\_flag = 0) classes. Higher precision for the default class indicated that fewer non-default instances were misclassified as defaults.
  + False Positive Analysis: False positives were examined to determine how many non-default cases were incorrectly classified as defaults. Reducing false positives was crucial to prevent unnecessary action on non-default accounts.
  + Precision Stability Assessment: The stability of precision scores across different data splits was evaluated. A stable precision score across various subsets indicated that the model’s performance was reliable and not prone to fluctuations depending on the data.
* Recall Evaluation:
  + Default Detection Rate: Recall for the default class was calculated to measure how effectively the model identified actual defaults. A higher recall indicated that the model was good at detecting defaults, which was the primary business goal.
  + Miss Rate Analysis: Miss rate (1 - recall) was analyzed to understand how many defaults were missed by the model. Lower miss rates were crucial for minimizing missed defaults, which could have serious financial consequences.
  + Impact of Threshold Adjustment: Different classification thresholds were tested to adjust the recall-precision tradeoff. By varying the threshold, the model’s ability to detect defaults was fine-tuned, optimizing recall without sacrificing precision too much.
* F1-Score Assessment:
  + Balanced Performance Metric: The F1-score, which is the harmonic mean of precision and recall, was computed to provide a balanced view of the model’s performance. The F1-score was especially useful in scenarios where there was a tradeoff between precision and recall.
  + Class-wise F1 Scores: F1 scores were calculated for both default and non-default classes to assess how well the model performed for each class. This analysis helped in identifying whether the model was skewed towards predicting the majority class.
  + Optimization Strategies: Various strategies were explored to optimize the F1-score, including adjusting the class weights, tuning hyperparameters, and fine-tuning the classification threshold. These strategies were aimed at improving the balance between precision and recall, ensuring that the model performed well on both classes.

4. Insights and Observations

4.1 Data Characteristics

* Feature Analysis:
  + Key Predictive Features: We conducted an analysis to identify the most influential features in predicting the probability of credit card defaults. Features related to transaction behavior, account history, and customer demographics were found to be the strongest predictors. These features had the highest correlation with the target variable (bad\_flag) and were crucial in determining defaults.
  + Feature Importance Ranking: Using techniques such as feature importance from tree-based models and permutation importance, we ranked the features based on their contribution to the model's performance. This ranking provided valuable insights into which features the model relied on most when making predictions.
  + Interaction Effects: We explored potential interactions between features, as some features might have a combined effect on the likelihood of default. For instance, certain transaction patterns combined with account age may be more predictive of defaults than individual features alone. Feature interaction analysis highlighted these relationships, aiding in model interpretation.
* Distribution Patterns:
  + Feature Value Ranges: The distribution of features was analyzed to understand their value ranges. Some features, such as transaction amounts, exhibited highly skewed distributions with long tails, while others, like account age, followed more uniform distributions. Understanding these distributions helped guide feature scaling and transformation decisions.
  + Outlier Impacts: Outliers were identified in several features, particularly in transaction amounts and account balances. These outliers were carefully examined, as they could have a disproportionate impact on model training. In some cases, outliers were capped or removed to prevent them from distorting the model's learning process.
  + Temporal Patterns: Temporal aspects of the data, such as transaction timestamps, were analyzed to identify any recurring patterns over time. For instance, defaults might be more likely during certain months or seasons, reflecting changes in spending behavior or external economic factors. These patterns provided insights into when defaults were more prevalent and helped refine the model.

4.2 Model Behavior

* Performance Patterns:
  + Training Convergence Analysis: During training, we tracked the model's loss and accuracy curves to assess how well the model was converging. The model showed steady improvement in the training loss, with the accuracy increasing as epochs progressed. However, overfitting was observed after a certain point, leading to the implementation of early stopping criteria to prevent it.
  + Validation Stability: We monitored the model’s performance on the validation set to ensure stability and prevent overfitting. The validation accuracy followed a similar trend to the training accuracy, suggesting that the model generalized well during training. However, slight fluctuations in validation loss indicated that some hyperparameter tuning might be required for further improvement.
  + Error Pattern Analysis: We analyzed the types of errors made by the model, focusing on false positives and false negatives. The model tended to misclassify non-defaults as defaults (false positives), which was expected given the class imbalance. However, efforts were made to minimize these errors by adjusting thresholds and fine-tuning the model.
  + 

5. Conclusion

In this project, we successfully developed a machine learning model for predicting credit card defaults, focusing on overcoming challenges related to class imbalance, data preprocessing, and model performance. By implementing a robust pipeline for data cleaning, feature engineering, and scaling, we ensured that the model received high-quality inputs for training. The custom weighted binary cross-entropy loss function effectively addressed class imbalance, improving the model's ability to detect defaults while maintaining overall accuracy.

The model demonstrated strong performance in terms of both technical success and business value. It is capable of providing reliable credit risk assessments, offering automated decision support, and is scalable for larger datasets, making it suitable for real-world deployment. With its ability to process large volumes of data efficiently, the solution can be used by businesses to make informed, data-driven decisions, reducing the risk of defaults and improving financial management.

Overall, the project succeeded in delivering a production-ready solution with significant potential for enhancing credit risk management practices in the financial industry.