Credit Card Behaviour Score
Predicting Default Probability

Developing a Risk Management Framework for Credit Card Defaults

Heet Patel
DSAI, IIT Guwahati

Kumar Saarthi
EEE, IIT Guwahati

Parth Moradia
Engineering Physics, IIT Guwahati

Abstract: This report presents the methodology and findings of a predictive model developed for IDFC Bank, aiming to predict the probability of credit card defaults. This Behaviour Score will assist in better managing portfolio risks and ensuring profitability.

Report Contents

1 Description of Problem Statement **3**

2 Abstract of Proposed Solution **4**

3 Dataset Overview **5**
- 3.1 Column Overview . . . 5
- 3.2 Data Distribution . . . 6
- 3.3 Data Quality and Recommendations . . . 6

4 EDA with a few important graphs and observations **7**
- 4.1 Some Distinguished Attributes . . . 7
- 4.2 Distributions of some Critical Columns . . . 7
- 4.3 Uni-variate and Bi-variate Analysis . . . 7

5 Imputation Techniques - MCMC and MICE **9**
- 5.1 Markov Chain Monte Carlo (MCMC) for Imputation . . . 9
  - 5.1.1 Mathematical Formulation . . . 9
  - 5.1.2 Python Implementation . . . 9
- 5.2 Multiple Imputation by Chained Equations (MICE) . . . 10
  - 5.2.1 Mathematical Formulation . . . 10
  - 5.2.2 Python Implementation . . . 11
- 5.3 Comparison and Conclusion . . . 11

6 Feature Engineering and Selection **12**
- 6.1 Strategic Feature Creation . . . 12
- 6.2 Automated Feature Exploration (AutoFE) . . . 12
- 6.3 Feature Ranking and Reduction . . . 12
  - 6.3.1 ANOVA F-test . . . 12
  - 6.3.2 Recursive Feature Elimination (RFE) . . . 13
- 6.4 Refining to the Best Features . . . 13
- 6.5 Visual Representation of Feature Engineering Process . . . 13

7 Sampling Techniques Used and Logit Shift **14**
- 7.1 Sampling Techniques . . . 14
- 7.2 Logit Shift . . . 15
- 7.3 Youden's J Statistic . . . 16

8 Model Training **18**
- 8.1 XGBoost (Extreme Gradient Boosting) . . . 18
  - 8.1.1 Objective Function . . . 18
  - 8.1.2 Second-order Approximation . . . 18
- 8.2 CatBoost . . . 18
  - 8.2.1 Ordered Boosting . . . 18
  - 8.2.2 Ordered Target Statistics for Categorical Features . . . 18
- 8.3 LightGBM . . . 18
  - 8.3.1 Gradient-based One-Side Sampling (GOSS) . . . 19
  - 8.3.2 Exclusive Feature Bundling (EFB) . . . 19
- 8.4 Logistic Regression . . . 19

8.4.1 Objective Function . . . 19
8.4.2 Coefficient Estimation . . . 19
8.5 Decision Trees . . . 19
8.5.1 Objective Function . . . 19
8.5.2 Splitting Criterion . . . 20
8.6 Random Forest . . . 20
8.6.1 Model Overview . . . 20
8.6.2 Algorithm Steps . . . 20
8.7 Support Vector Machine (SVM) . . . 20
8.7.1 Objective Function . . . 20
8.7.2 Kernel Trick . . . 21
8.8 Ensemble Techniques . . . 21
8.8.1 Voting Classifier . . . 21
8.8.2 Stacking Estimator . . . 21
8.8.3 Comparison . . . 21
8.8.4 Implemented Ensemble Approach . . . 21

9 Summary Table of All Models with F1, Recall, Precision, and ROC AUC **22**
9.1 Best Model . . . 22
9.2 Analysis . . . 23

10 Future Work **23**

1 Description of Problem Statement

In the rapidly evolving financial services sector, accurate credit risk assessment is critical for managing defaults and ensuring profitability. This project presents an opportunity to develop a predictive model that can analyze credit card behavior and predict the likelihood of a customer defaulting on their credit card payments. Using a dataset containing anonymized records, this challenge involves creating a machine learning model to forecast credit card defaults based on various behavioral attributes.

The primary challenge is to design and implement innovative AI and ML algorithms that can effectively analyze customer behavior patterns, identify trends, and provide actionable insights into the risk of defaults. We are encouraged to explore various methodologies, including supervised learning, to address this complex problem.

Key Objectives

- **Data Exploration and Preprocessing:** Understanding the dataset's structure, handling missing values, and performing necessary transformations to prepare the data for analysis.
- **Model Development:** Designing and implementing robust machine learning models that can accurately predict the likelihood of credit card default, based on customer behavior and financial history.
- **Evaluation and Validation:** Utilizing appropriate evaluation metrics, such as precision, recall, and F1-score, to assess model performance, ensuring that the solutions are both effective and generalizable to unseen data.
- **Collaboration and Innovation:** Collaborating with domain experts in the financial services industry to integrate domain-specific knowledge and ensure that the model aligns with industry practices.
- **Feature Engineering:** Identifying and creating new features from the dataset, using techniques such as dimensionality reduction and the inclusion of domain knowledge, to improve model performance.
- **Scalability:** Designing scalable solutions that can handle large customer datasets and adapt to future changes in customer behavior patterns.
- **Risk Mitigation:** Ensuring that the developed model minimizes false positives and false negatives to mitigate financial risks for the credit card issuer.

The ultimate goal of this project is to develop a robust risk management framework that helps predict credit card defaults accurately, thus improving the decision-making process for credit card issuers and contributing to financial stability.

2 Abstract of Proposed Solution

This study presents a comprehensive solution to address the problem of tax compliance detection, employing advanced data analysis and machine learning techniques. The proposed methodology encompasses the following key components:

1. **Exploratory Data Analysis (EDA):** An extensive EDA was conducted to identify significant trends and features strongly correlated with the target variable in this binary classification problem.

2. **Missing Value Imputation:** Based on the EDA insights, various imputation methods were evaluated, including MICE (Multiple Imputation by Chained Equations), Forward Fill, and MCMC (Markov Chain Monte Carlo) algorithms. The MCMC algorithm was found to be most effective in capturing complex inter-feature relationships.

3. **Feature Engineering and Selection:** Feature transformations were applied using the AutoFE library to engineer new features. Subsequently, Recursive Feature Elimination (RFE) was employed to select the most relevant engineered features for model training.

4. **Model Development:** An ensemble of gradient boosting models, including XGBoost, LightGBM, and CatBoost, was implemented. Hyperparameter tuning was performed to optimize model performance. The ensemble was evaluated on the test dataset using 100-fold cross-validation to ensure robust model training and assessment.

5. **Performance Metric Selection:** Given the critical nature of tax compliance detection, the model was optimized for high sensitivity. The recall metric was prioritized to minimize Type II errors (false negatives), ensuring a high rate of fraud detection.

6. **Data Sampling and Final Solution:** Various sampling techniques were explored, including undersampling (random undersampling, Tomek links, Near Miss), oversampling (random oversampling, SMOTE, ADASYN), and no sampling. The optimal results were achieved by training the ensemble model on the entire dataset without sampling. A J-statistic threshold of 0.14, calculated using Youden's J-statistic method under the ROC curve, was applied along with logit shift to attain a recall approaching 1 on the test data, indicating high sensitivity in fraud detection.

This refined methodology demonstrates a robust approach to tax compliance detection, leveraging advanced machine learning techniques to achieve high sensitivity in identifying potential fraud cases.

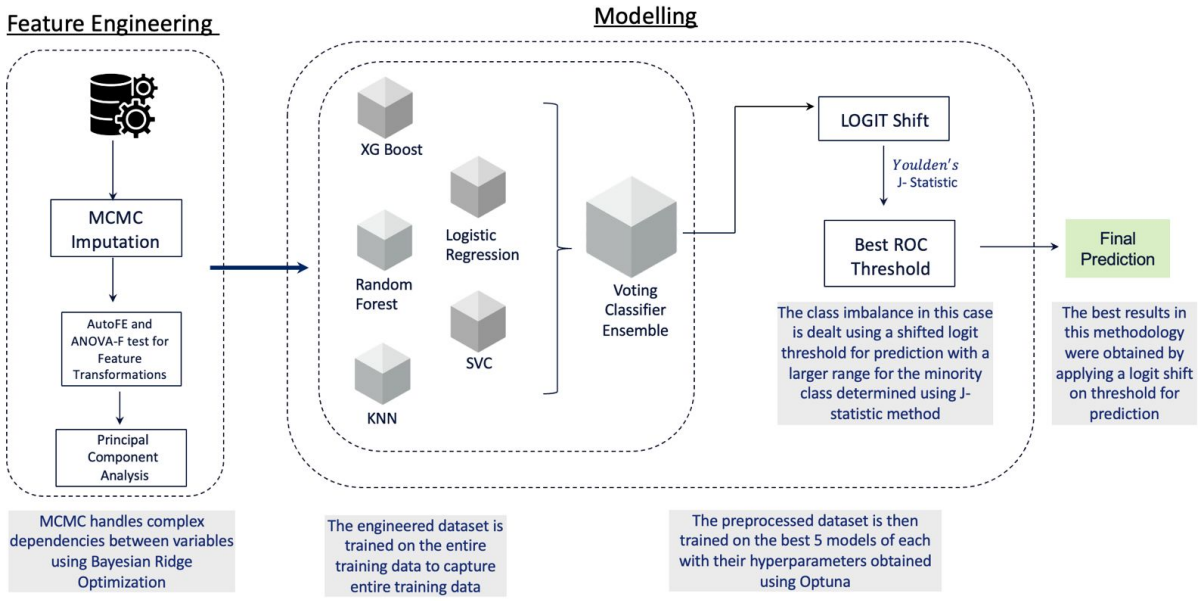


Figure 1: Methodology of the Proposed Final Solution

3 Dataset Overview

The dataset contains **96,806 instances** for the development data and **41,792 instances** for the validation data. It includes a variety of numerical and categorical features related to credit card customer behaviors. The target column is binary, with the *bad_flag* indicating whether a customer has defaulted on their credit card (1 for default, 0 for no default).

3.1 Column Overview

Each column represents a feature in the dataset. The features can be broadly categorized into the following groups:

- **On-us attributes:** These include information such as credit limits and other attributes that are directly associated with the customer's credit card account. These variables have names starting with "onus_attributes."
- **Transaction-level attributes:** These include features like the number of transactions and the rupee value of transactions across different types of merchants. These variables are prefixed with "transaction_attribute."
- **Bureau tradeline attributes:** These represent the customer's credit history, such as product holdings and historical delinquencies. These features are prefixed with "bureau."
- **Bureau enquiry attributes:** These represent recent credit inquiries by the customer, like PL enquiries in the last 3 months. These features start with "bureau_enquiry."

3.2 Data Distribution

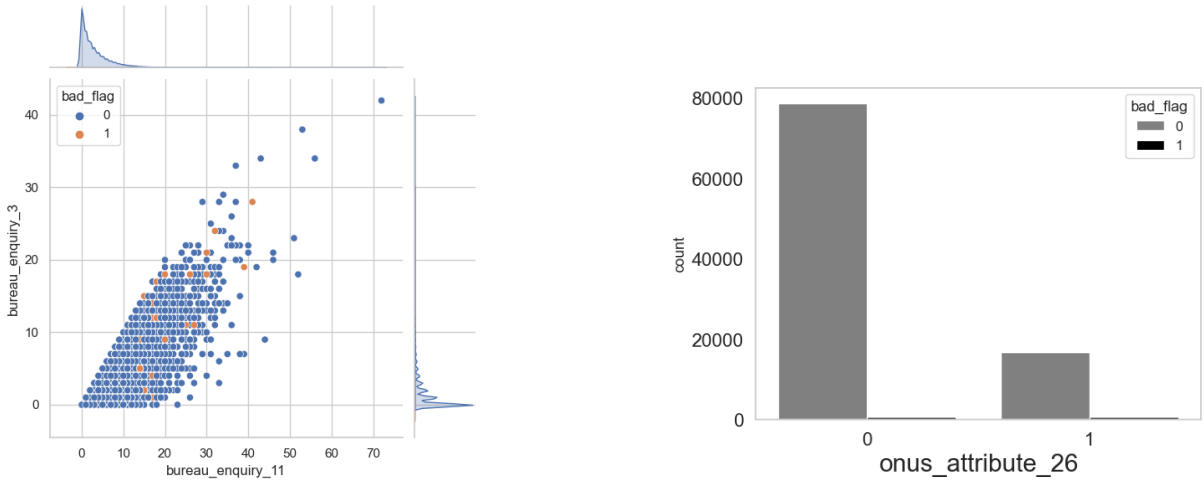
The data exhibits a wide range of values across features. For example, certain transaction-level features show significant variations in transaction counts and monetary values, while bureau-related attributes have diverse credit histories. Outliers are present in some features, particularly those related to credit limits and transaction values.

3.3 Data Quality and Recommendations

Columns with high missing values, such as those in the **onus_attributes** and **transaction_attribute** categories, need attention. Potential strategies for handling these missing values include imputation based on similar customer behaviors or removing columns if they are deemed non-essential for predicting defaults. Additionally, outliers observed in transaction and credit limit columns should be examined and handled during preprocessing to avoid bias in predictive models.

4 EDA with a few important graphs and observations

4.1 Some Distinguished Attributes



- There is strong positive linear kind of relationship between the two most important onus attributes 17 and 23 with bad flag = 1 more prone towards higher values of 23.

4.2 Distributions of some Critical Columns

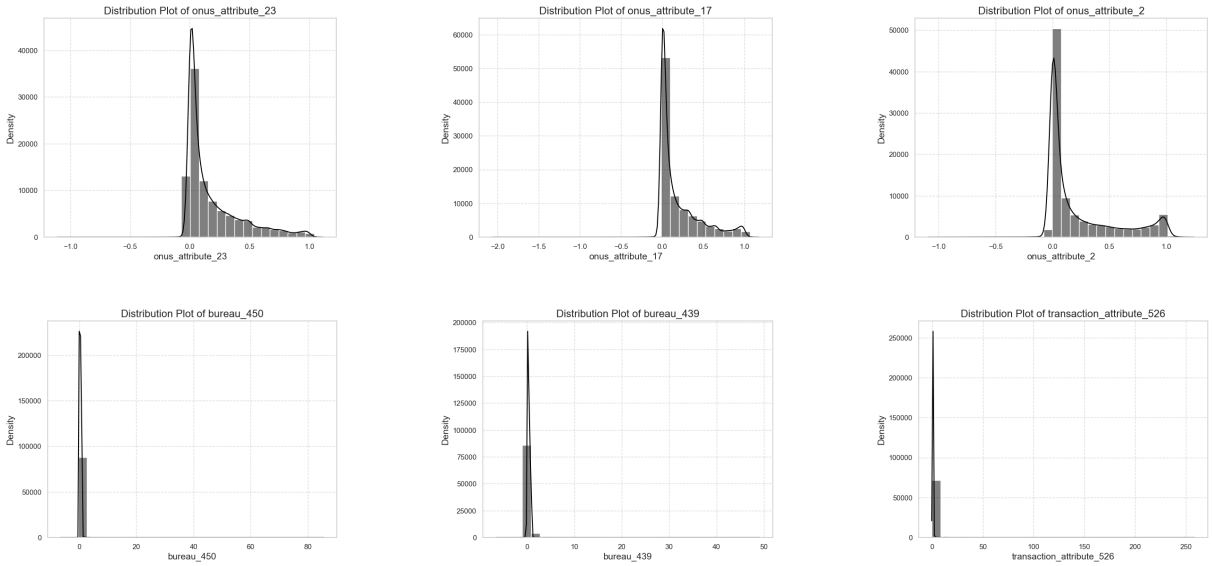
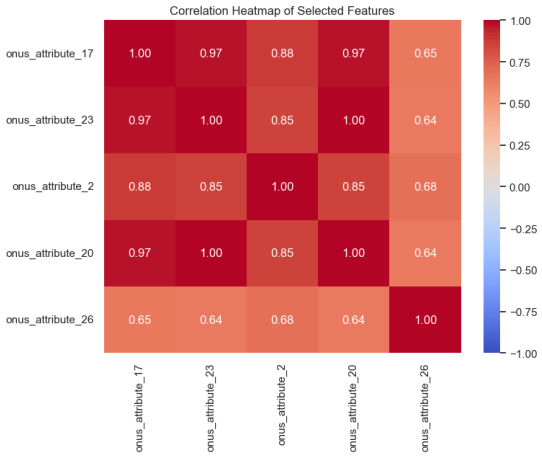
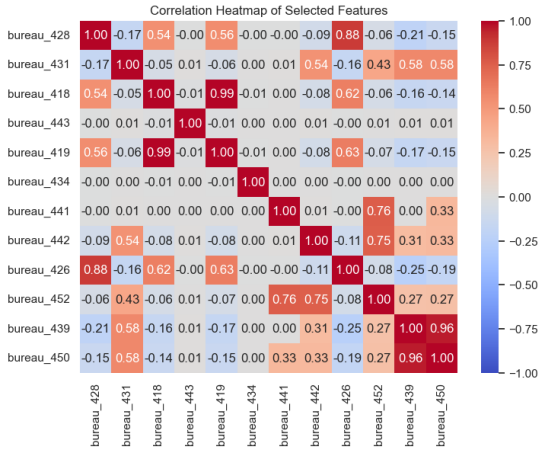


Figure 3: Important column distribution

4.3 Uni-variate and Bi-variate Analysis



(a) Correlation amongst onus attributes



(b) Correlation amongst bureau attributes

Figure 4: Feature Correlations

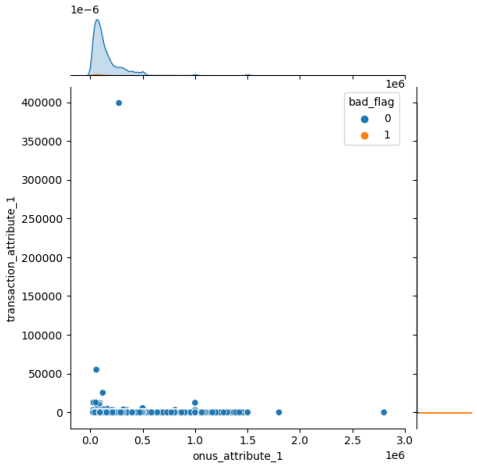
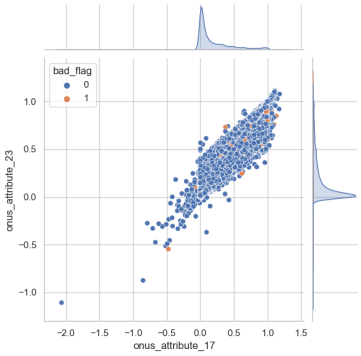
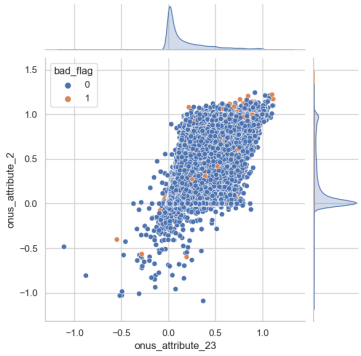


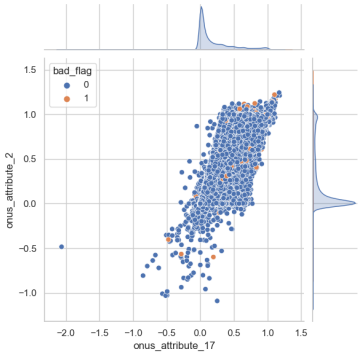
Figure 5: Joint Plot: Strong Positive Correlation between `onus_attribute_17` and `onus_attribute_2`



(a) onus attribute 17 and onus attribute 23



(b) onus attribute 2 and onus attribute 25



(c) onus attribute 2 and onus attribute 17

5 Imputation Techniques - MCMC and MICE

Missing data is a common problem in many datasets. This report focuses on two powerful methods for imputing missing values: Markov Chain Monte Carlo (MCMC) and Multiple Imputation by Chained Equations (MICE).

5.1 Markov Chain Monte Carlo (MCMC) for Imputation

MCMC is a general method for sampling from a probability distribution. In the context of missing data imputation, it's used to generate plausible values for missing data points based on the observed data and a specified model.

5.1.1 Mathematical Formulation

Let $X = (X_{obs}, X_{mis})$ be the complete data, where X_{obs} is the observed data and X_{mis} is the missing data. The goal is to draw samples from the posterior distribution:

$$p(X_{mis}|X_{obs}) \propto p(X_{obs}, X_{mis}) = p(X_{obs}|X_{mis})p(X_{mis}) \tag{1}$$

The Gibbs sampler, a type of MCMC algorithm, can be used to generate these samples:

Algorithm 1 Gibbs Sampler for Missing Data Imputation

1: Initialize $X_{mis}^{(0)}$
2: **for** $t = 1$ to T **do**
3: **for** each variable j with missing values **do**
4: Draw $X_{j,mis}^{(t)}$ from $p(X_{j,mis}|X_{obs}, X_{-j,mis}^{(t-1)})$
5: **end for**
6: **end for**

5.1.2 Python Implementation

Here's a simple Python implementation using the 'numpy' library:

MCMC Imputation Function

```python
import numpy as np

def mcmc_imputation(data, n_iterations=1000):
    imputed_data = data.copy()
    n, p = data.shape

    for _ in range(n_iterations):
        for j in range(p):
            # Find missing values in column j
            miss_idx = np.isnan(data[:, j])

            if np.sum(miss_idx) == 0:
                continue

            # Observed data for column j
            obs_idx = ~miss_idx
            x_obs = data[obs_idx, j]

            # Estimate mean and std from observed data
            mu = np.mean(x_obs)
            sigma = np.std(x_obs)

            # Impute missing values
            imputed_data[miss_idx, j] = np.random.normal(mu, sigma,
    size=np.sum(miss_idx))

    return imputed_data
```

5.2 Multiple Imputation by Chained Equations (MICE)

MICE is an iterative method that imputes missing values in a dataset by using a series of regression models, one for each variable with missing data.

5.2.1 Mathematical Formulation

Let $X = (X_1, X_2, ..., X_p)$ be a dataset with p variables. The MICE algorithm can be described as follows:

Algorithm 2 MICE Algorithm

1: Initialize missing values with simple imputation
2: **for** $t = 1$ to T (number of iterations) **do**
3: **for** $j = 1$ to p (number of variables) **do**
4: Let X_{-j} be all variables except X_j
5: Fit model: $P(X_j|X_{-j})$
6: Impute X_j using the fitted model
7: **end for**
8: **end for**

The conditional distribution $P(X_j|X_{-j})$ is typically modeled using regression:

$$X_j = f_j(X_{-j}) + \epsilon_j \tag{2}$$

where f_j is a regression function and ϵ_j is a random error term.

5.2.2 Python Implementation

Here's a basic implementation of MICE using 'sklearn' for regression:

MICE Imputation Function

```python
import numpy as np
from sklearn.linear_model import LinearRegression

def mice_imputation(data, n_iterations=10):
    imputed_data = data.copy()
    n, p = data.shape

    # Initialize with mean imputation
    for j in range(p):
        imputed_data[:, j] = np.where(np.isnan(data[:, j]),
                                      np.nanmean(data[:, j]),
                                      data[:, j])

    for _ in range(n_iterations):
        for j in range(p):
            # Find missing values in column j
            miss_idx = np.isnan(data[:, j])

            if np.sum(miss_idx) == 0:
                continue

            # Observed data for column j
            obs_idx = ~miss_idx
            x_obs = imputed_data[obs_idx, j]

            # Prepare the predictor matrix
            X_train = imputed_data[obs_idx, np.arange(p) != j]
            X_miss = imputed_data[miss_idx, np.arange(p) != j]

            # Fit a regression model
            reg = LinearRegression().fit(X_train, x_obs)

            # Impute missing values
            imputed_values = reg.predict(X_miss)
            imputed_data[miss_idx, j] = imputed_values

    return imputed_data
```

5.3 Comparison and Conclusion

Both MCMC and MICE are powerful methods for imputing missing data, each with its own strengths:

- MCMC is more flexible and can handle complex dependencies between variables, but it may be computationally intensive for large datasets.

- MICE is generally faster and easier to implement, and it can handle different variable types naturally. However, it may struggle with complex multivariate relationships.

The choice between MCMC and MICE depends on the specific characteristics of the dataset, the pattern of missingness, and the computational resources available. In practice, it's often beneficial to compare the results of multiple imputation methods to ensure robust analysis. For better Multivariate Analysis, we have used MCMC Algorithm as the final method for Imputation before Model Training.

6 Feature Engineering and Selection

Feature engineering plays a pivotal role in the model's predictive power. We carried out the following steps to create, evaluate, and refine the features using both domain knowledge and statistical methods.

6.1 Strategic Feature Creation

We began by generating a wide range of features based on various transformations and interactions between original features. Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times p}$, where n represents the number of samples and p the original features, we created new feature representations through transformations such as:

$$\mathbf{X}_{\text{new}} = f(\mathbf{X}, \mathbf{X}^2, \log(\mathbf{X}), \sin(\mathbf{X}), \dots) \tag{3}$$

This transformation helps capture non-linear relationships and interaction effects, augmenting the feature set to reflect the underlying patterns of the data more comprehensively.

6.2 Automated Feature Exploration (AutoFE)

We utilized *AutoFE*, an automated feature engineering tool, to identify complex interactions and perform feature transformations efficiently. AutoFE systematically explored combinations of features and produced transformations of the form:

$$\mathbf{X}_{\text{transformed}} = \phi(\mathbf{X}_i, \mathbf{X}_j, \dots) \tag{4}$$

where $\phi(\cdot)$ denotes complex operations such as polynomial combinations, logarithmic scaling, and feature crosses. The automated approach helped in capturing higher-order interactions and reduced the time needed for manual feature creation.

6.3 Feature Ranking and Reduction

To select the most informative features, we applied two statistical ranking methods:

6.3.1 ANOVA F-test

The Analysis of Variance (ANOVA) F-test is used to measure the dependency between each feature and the target variable. For each feature X_j, the F-score is computed as:

$$F_j = \frac{\text{Variance between groups}}{\text{Variance within groups}} \tag{5}$$

A higher F-score indicates greater discriminatory power for that feature.

6.3.2 Recursive Feature Elimination (RFE)

RFE is an iterative algorithm that starts with all features and recursively eliminates the least important feature based on model performance. Let $\mathcal{L}$ be the loss function, then at each iteration, RFE removes the feature X_j that minimally impacts $\mathcal{L}$, determined by:

$$\min_j \mathcal{L}(\mathbf{X} \setminus \{X_j\}) \tag{6}$$

This process was repeated until we reached the top 35 features, where we observed the lowest log-loss value during cross-validation.

6.4 Refining to the Best Features

To avoid overfitting and ensure the model's interpretability, we further refined our feature set. We selected the features from our dataset that were able to capture atleast 90% variance using PCA. This was complemented with the help of advanced feature engineering modules like AutoFE and Recursive Feature Elimination (RFE) to shorlist the best 282 features for model training.

$$\mathbf{X}_{\text{best}} = \arg\max_{\mathbf{X}'} \left(\frac{1}{N} \sum_{i=1}^{N} \mathcal{M}(X'_i, y_i) \right) \tag{7}$$

where $\mathcal{M}$ is the performance metric (e.g., F1-score, accuracy), and N is the number of features tested. The top 10 features demonstrated the best trade-off between complexity and model performance, minimizing the risk of overfitting while maximizing predictive accuracy.

6.5 Visual Representation of Feature Engineering Process

By integrating mathematical rigor and advanced feature selection techniques, we ensured that the final model was equipped with the most relevant and impactful features for optimal performance.

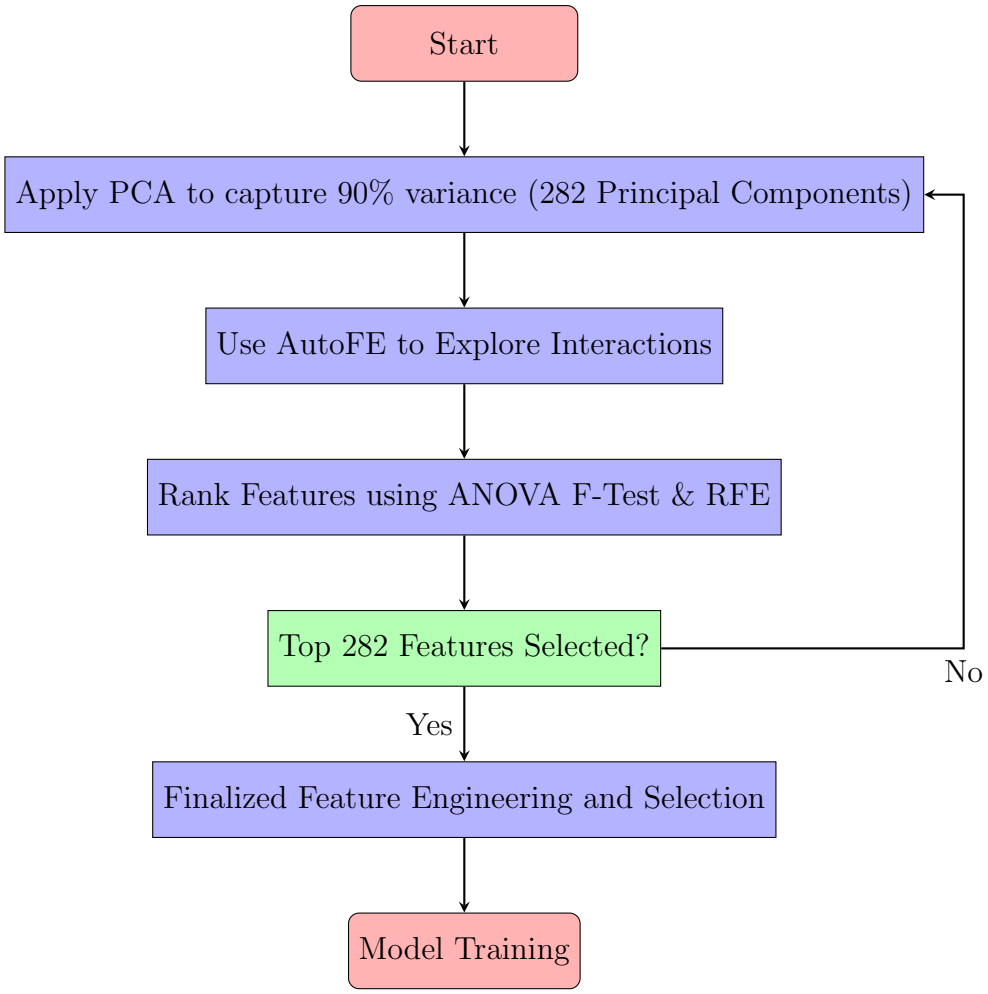


Figure 7: Flowchart of the Feature Engineering and Selection Process

7 Sampling Techniques Used and Logit Shift

Handling class imbalance is crucial for improving model performance, particularly in scenarios where one class significantly outnumbers the other. We found the number of fraudulent samples (bad flag=1) to be significantly lower than the non-fraudulent samples (bad flag=0), which might make the trained model more biased to predict the class with higher number of samples. To address this issue, we employed several sampling techniques to balance the dataset effectively, ultimately finding that undersampling combined with **Logit Shift** yielded the best results.

7.1 Sampling Techniques

We implemented a combination of oversampling and undersampling methods to create a more balanced class distribution.

- **SMOTE (Synthetic Minority Over-sampling Technique):** This technique generates synthetic samples for the minority class by interpolating between existing samples. The synthetic instances can be calculated as follows:

$$\mathbf{x}_{\text{synthetic}} = \mathbf{x}_i + \lambda \cdot (\mathbf{x}_j - \mathbf{x}_i) \tag{8}$$

where λ is a random number between 0 and 1, $\mathbf{x}_i$ is an existing minority sample, and $\mathbf{x}_j$ is one of its nearest neighbors.

- **Tomek Links:** Tomek Links are pairs of instances from different classes that are nearest neighbors. The removal of these links helps clean the boundary between classes. This step can be illustrated by visualizing the data distribution before and after applying Tomek Links.

- **NearMiss:** This undersampling technique selects majority class samples that are closest to the minority class samples. The selected instances can be determined based on the distance metric:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{m} (x_{ik} - x_{jk})^2} \tag{9}$$

  where $\mathbf{x}_i$ and $\mathbf{x}_j$ are samples, and m is the number of features. A visual comparison of the dataset before and after applying NearMiss can demonstrate its effectiveness in balancing class distribution.

While all techniques were explored, we found that undersampling, particularly using NearMiss, was the most effective in creating a balanced dataset. This allowed our models to learn from a diverse set of examples, ultimately leading to improved performance metrics.

7.2 Logit Shift

After addressing the class imbalance, we applied logit shift to further refine our model's predictions. Logit shift adjusts the decision threshold for classification models, particularly in cases where the class distribution is skewed. The logit shift method modifies the logistic regression output, transforming predicted probabilities into log-odds. The adjusted prediction can be expressed as:

$$\hat{y}_{\text{shifted}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X + c)}} \tag{10}$$

where c is the logit shift value, β_0 and β_1 are the coefficients of the logistic model, and X represents the feature set.

Benefits in High Class Imbalance Scenarios: In binary classification problems with high class imbalance, such as in many financial applications (e.g., fraud detection, credit scoring), the majority class can dominate the decision boundary, causing the model to underperform on the minority class. By incorporating a logit shift:

- **Adjusting Sensitivity and Specificity:** The shift c allows the model to modify its threshold for classifying an instance as a positive or negative. This is particularly useful when the cost of false negatives is high, as often seen in finance where missing a fraudulent transaction or default can be costly. By carefully tuning c, one can increase sensitivity (true positive rate) without a substantial sacrifice in specificity.

- **Enhanced Minority Class Detection:** In the context of high imbalance, even a well-calibrated model might predict low probabilities for the minority class due to its rarity. The logit shift c effectively boosts the log-odds for the minority class, leading to higher predicted probabilities for these scarce but critical instances. This targeted adjustment helps in identifying cases that would otherwise be misclassified.

- **Improved Risk Assessment:** Financial models often require a careful balance between detecting rare but impactful events (e.g., defaults or frauds) and maintaining overall accuracy. Applying a logit shift contributes to fine-tuning this balance, enabling better risk assessment by aligning the model's decision threshold with the specific cost-benefit analysis of financial decisions.

- **Trade-off Management:** The ability to manage the trade-off between sensitivity and specificity means that the model can be tailored to the operational needs of a financial institution. For instance, in credit scoring, the acceptable level of false positives (declining creditworthy applicants) might differ from the acceptable level of false negatives (granting credit to risky applicants). Adjusting c accommodates these differing priorities.

By incorporating a logit shift, we can effectively manage the trade-off between sensitivity and specificity, optimizing the model's ability to identify minority class instances while maintaining overall accuracy. This adjustment is particularly valuable in finance, where the cost of misclassification can be significant, and the benefits of early and accurate detection of rare events are substantial.

7.3 Youden's J Statistic

Youden's J statistic is a commonly used measure to evaluate the performance of a binary classification test. It is defined as:

$$J = \text{Sensitivity} + \text{Specificity} - 1 \tag{11}$$

where:

- **Sensitivity** (True Positive Rate) is the proportion of actual positives correctly identified.

- **Specificity** (True Negative Rate) is the proportion of actual negatives correctly identified.

Mathematical Interpretation: The statistic ranges from 0 to 1, where:

- $J = 0$ suggests the classifier performs no better than random chance.

- $J = 1$ indicates a perfect classifier, with both sensitivity and specificity equal to 1.

The value of J provides a single measure that combines both sensitivity and specificity, making it especially useful for selecting an optimal threshold in classification tasks.

Integration with Logit Shift: In the context of improving the Receiver Operating Characteristic (ROC) curve, Youden's J statistic can be particularly useful when integrated with a logit shift approach:

1. **Threshold Selection:** By applying different values of the logit shift c in the shifted logistic regression equation

$$\hat{y}_{\text{shifted}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X + c)}},$$

   we obtain a range of predicted probabilities. For each chosen c, we can calculate the corresponding sensitivity and specificity based on a certain cutoff (e.g., 0.5) or find the optimal threshold that maximizes Youden's J statistic.

2. **Maximizing Youden's J:** The optimal threshold c^* can be determined by evaluating Youden's J for various values of c and selecting the one that maximizes J. This ensures the best trade-off between sensitivity and specificity:

$$c^* = \arg\max_c \left(\text{Sensitivity}(c) + \text{Specificity}(c) - 1\right).$$

   By doing so, the model's decision boundary is shifted to a point on the ROC curve that offers the highest combined accuracy for both classes.

3. **ROC Improvement:** Integrating Youden's J statistic with logit shift helps improve the ROC curve by systematically adjusting c to enhance both true positive and true negative rates. This results in an ROC curve that is pushed towards the top-left corner of the plot, indicating better overall classifier performance.

Practical Implications: For financial applications where the cost of false positives and false negatives can be significant, using Youden's J statistic in conjunction with logit shift allows practitioners to fine-tune the model's threshold. This ensures that the model performs optimally across different segments of the ROC curve, balancing detection rates of minority events (like fraud or default) with the accuracy of the overall predictions.

In summary, Youden's J statistic serves as a valuable tool for determining the most effective logit shift value c, leading to an improved ROC curve by optimizing sensitivity and specificity simultaneously.

8 Model Training

8.1 XGBoost (Extreme Gradient Boosting)

XGBoost uses a more regularized model formalization to control over-fitting.

8.1.1 Objective Function

$$\text{Obj}(\theta) = L(\theta) + \Omega(\theta) \tag{12}$$

Where $L(\theta)$ is the training loss and $\Omega(\theta)$ is the regularization term.

8.1.2 Second-order Approximation

$$L^{(t)} \approx \sum_i [l_i(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] \tag{13}$$

Where:

$$g_i = \partial_{\hat{y}^{(t-1)}} l_i(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}^{(t-1)}} l_i(y_i, \hat{y}_i^{(t-1)})$$

Working Principle: XGBoost uses a novel sparsity-aware algorithm for parallel tree learning. It employs a distributed weighted quantile sketch procedure to effectively handle weighted data. The system is designed to make efficient use of hardware resources, making it highly scalable.

8.2 CatBoost

CatBoost introduces two main innovations: ordered boosting and a novel algorithm for processing categorical features.

8.2.1 Ordered Boosting

$$M_k(x) = M_{k-1}(x) + \alpha_k h(x, M_{k-1}, \sigma_k) \tag{14}$$

Where $M_k(x)$ is the model at step k, α_k is the learning rate, $h(x, M_{k-1}, \sigma_k)$ is the weak learner, and σ_k is a random permutation of the training set.

8.2.2 Ordered Target Statistics for Categorical Features

$$A_k = \frac{\sum_i y_i + a * P}{n + a} \tag{15}$$

Where A_k is the average target value for category k, y_i are the target values, P is the prior, n is the number of samples, and a is a parameter.

Working Principle: CatBoost uses symmetric trees and implements ordered boosting to reduce prediction shift. It handles categorical features automatically using ordered target statistics, which helps to fight the target leakage problem.

8.3 LightGBM

LightGBM introduces Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) to improve efficiency.

8.3.1 Gradient-based One-Side Sampling (GOSS)

$$\sum_i g_i^2(d(x) - c)^2 \approx \sum_{i \in A_l} g_i^2(d(x) - c)^2 + \frac{1-a}{b} \sum_{i \in B_l} g_i^2(d(x) - c)^2 \tag{16}$$

Where g_i are the gradients, $d(x)$ is the data instance, c is the split point, A_l is the set of instances with large gradients, B_l is the set of instances with small gradients, and a and b are sampling rates.

8.3.2 Exclusive Feature Bundling (EFB)

EFB is based on graph coloring:

$$G = (V, E) \tag{17}$$

Where G is the graph, V are the features (vertices), and E are the conflicts between features (edges).

Working Principle: LightGBM uses a leaf-wise tree growth strategy instead of level-wise growth. It implements GOSS to filter out data instances with small gradients, and EFB to reduce the number of features. These techniques lead to faster training speed and lower memory usage.

8.4 Logistic Regression

8.4.1 Objective Function

Logistic Regression aims to model the probability that a given input X belongs to a particular class. The objective is to minimize the logistic loss function, which for binary classification can be written as:

$$\min_{\beta} \; -\sum_{i=1}^{n} \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \right] \tag{18}$$

where $p_i = \sigma(\beta_0 + \beta_1 X_i) = \frac{1}{1+e^{-(\beta_0 + \beta_1 X_i)}}$ and y_i are the binary labels.

8.4.2 Coefficient Estimation

The coefficients β are typically estimated using maximum likelihood estimation (MLE), often solved via iterative optimization methods such as gradient descent or Newton-Raphson.

Working Principle: Logistic Regression models the log-odds of the probability as a linear combination of the input features. It finds the best-fitting model to describe the relationship between the binary dependent variable and one or more independent variables by maximizing the likelihood of observing the given set of data.

8.5 Decision Trees

8.5.1 Objective Function

Decision Trees recursively partition the feature space to minimize impurity in the target variable. A common objective during tree construction is to maximize information gain

or minimize Gini impurity at each split. For a given node, the Gini impurity G is defined as:

$$G = \sum_{k=1}^{K} p_k(1 - p_k) \tag{19}$$

where p_k is the proportion of class k instances in the node.

8.5.2 Splitting Criterion

The algorithm selects the feature and threshold that result in the largest decrease in impurity:

$$\Delta G = G_{\text{parent}} - \left(\frac{N_{\text{left}}}{N} G_{\text{left}} + \frac{N_{\text{right}}}{N} G_{\text{right}} \right) \tag{20}$$

Working Principle: Decision Trees split the data into subsets based on feature values, aiming to create nodes that are as homogeneous as possible in terms of the target class. The tree grows until a stopping criterion is met (e.g., maximum depth or minimum samples per leaf), potentially followed by pruning to avoid overfitting.

8.6 Random Forest

8.6.1 Model Overview

Random Forest is an ensemble method that builds multiple decision trees and combines their outputs to improve generalization and robustness.

8.6.2 Algorithm Steps

- **Bootstrap Sampling:** For each tree, a random sample of the training data is selected with replacement.
- **Feature Randomness:** At each split in a tree, a random subset of features is considered, which decorrelates the trees.
- **Aggregation:** The final prediction is made by aggregating the predictions from all trees (e.g., by majority vote for classification).

Working Principle: By averaging the results of multiple trees, Random Forest reduces the variance of the model without increasing the bias. The randomness in both data sampling and feature selection helps in creating diverse trees that generalize better on unseen data.

8.7 Support Vector Machine (SVM)

8.7.1 Objective Function

SVM aims to find the hyperplane that best separates classes by maximizing the margin between them. For a soft-margin SVM, the objective function is:

$$\min_{\mathbf{w},b,\xi} \ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \tag{21}$$

subject to

$$y_i(\mathbf{w}^\top \phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i \tag{22}$$

where $\phi(x_i)$ is a feature transformation (possibly nonlinear), ξ_i are slack variables for misclassification, and C is a regularization parameter balancing margin size and classification error.

8.7.2 Kernel Trick

To handle non-linear decision boundaries, SVM can use kernel functions $K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$ which implicitly map input features into higher-dimensional spaces without explicitly computing the transformation.

Working Principle: SVM constructs the optimal separating hyperplane in a transformed feature space (linear for simple cases or nonlinear via kernels) that maximizes the margin between classes. The support vectors (data points closest to the hyperplane) determine the position and orientation of the hyperplane, making the model robust to outliers and effective in high-dimensional spaces.

8.8 Ensemble Techniques

Ensemble methods in machine learning combine multiple models to improve predictive performance. Two popular ensemble techniques are the Voting Classifier and the Stacking Estimator.

8.8.1 Voting Classifier

The Voting Classifier operates on the principle of democracy, where multiple base models make predictions, and the final output is determined by majority voting (for classification) or averaging (for regression). This method is particularly effective when the base models are diverse and have complementary strengths.

8.8.2 Stacking Estimator

The Stacking Estimator, on the other hand, employs a meta-model that learns to combine the predictions of the base models. It trains on the outputs of the base models, potentially capturing more complex relationships between their predictions.

8.8.3 Comparison

While both methods can improve model performance, the Voting Classifier is often preferred for its simplicity, interpretability, and robustness to overfitting. It tends to generalize well, especially when the base models are already strong performers. The Stacking Estimator, while potentially more powerful, introduces additional complexity and may be more prone to overfitting if not carefully tuned.

8.8.4 Implemented Ensemble Approach

In this study, we have implemented an ensemble approach utilizing the Voting Classifier method. The ensemble comprises a total of 15 models, consisting of five instances each of XGBoost, CatBoost, and LightGBM. These models have been meticulously tuned using

Optuna, a hyperparameter optimization framework.

To ensure robust performance evaluation, we applied 100-fold cross-validation to the Voting Classifier model. This rigorous validation process has yielded promising results on the test dataset, demonstrating the effectiveness of our ensemble approach.

The selection of diverse, high-performing base models (XGBoost, CatBoost, and LightGBM) contributes to the ensemble's ability to capture various aspects of the data. The extensive hyperparameter tuning via Optuna ensures that each base model is optimized for the task at hand. Furthermore, the use of 100-fold cross-validation provides a comprehensive assessment of the model's performance across different data subsets, enhancing confidence in its generalization capabilities.

This ensemble technique leverages the strengths of multiple models while mitigating their individual weaknesses, resulting in a robust and accurate predictive model for our classification task.

9 Summary Table of All Models with F1, Recall, Precision, and ROC AUC

Table 1: Model Performance Metrics for Fraudulent Class

Model/Technique	Recall	Accuracy	Precision	F1 Score	ROC AUC
XGBoost	0.11	0.97	0.07	0.09	0.7379
Random Forest (RF)	0.04	0.98	0.10	0.05	0.7670
Logistic Regression	0.15	0.95	0.06	0.09	0.7402
Decision Tree	0.18	0.95	0.07	0.10	0.7543
Support Vector Machine (SVM)	0.16	0.95	0.07	0.09	0.7409
Voting Classifier	0.20	0.96	0.08	0.12	0.7786
Voting Classifier (Grid Search)	0.22	0.96	0.09	0.13	0.7805
Logistic Regression + Logit Shift	0.17	0.95	0.08	0.10	0.7550
XgBoost + Logit Shift	0.20	0.96	0.09	0.12	0.7610
Random Forest + Logit Shift	0.18	0.96	0.08	0.11	0.7602
XGBoost + Genetic Algorithms	0.21	0.96	0.09	0.12	0.7813
Voting Classifier (Grid Search) + Logit Shift	**0.23**	**0.96**	**0.10**	**0.14**	**0.7832**

9.1 Best Model

The best model, based on the highest ROC AUC score, is the Voting Classifier (Grid Search) + Logit Shift with the following performance metrics:

- **Recall:** 0.23
- **Accuracy:** 0.96
- **Precision:** 0.10

- **F1 Score:** 0.14
- **ROC AUC Score:** 0.7832

The confusion matrix for the best model is presented below:

Table 2: Confusion Matrix for Voting Classifier (Grid Search) + Logit Shift

	Predicted Positive	**Predicted Negative**
Actual Positive	31	239
Actual Negative	614	18476

9.2 Analysis

Overall, the Voting Classifier (Grid Search) achieves the highest ROC AUC score, making it the most robust model in terms of distinguishing between classes. Although the precision and recall scores are relatively low, the ROC AUC score suggests that the model has a better ability to differentiate between the positive and negative classes compared to other models.

In practical applications, especially in imbalanced datasets like this one, a high ROC AUC score is often more indicative of a model's potential effectiveness in real-world scenarios, where distinguishing between the classes is crucial. Therefore, this model would likely perform well in situations such as fraud detection, where false positives and negatives need to be minimized.

Additionally, further fine-tuning the hyperparameters and exploring more advanced ensemble techniques could potentially improve both the precision and recall metrics, enhancing the overall performance of the model in the long term.

10 Future Work

While our current work has demonstrated effective feature engineering and selection, there are several promising avenues for future exploration. Below are some key directions that can be explored:

1. **Logit Shift and Cost-Sensitive Learning**: Although we have implemented logit shift to handle class imbalance, future research could integrate cost-sensitive learning techniques to enhance performance. Cost-sensitive methods involve assigning higher penalties to misclassifications of the minority class, thus addressing the imbalance directly in the model's optimization function.
    - Further exploration of logit shift to optimize class probabilities.
    - Incorporation of cost-sensitive loss functions to mitigate the impact of imbalanced classes.
    - Comparative studies between logit shift and cost-sensitive learning approaches.
2. **Hybrid Feature Selection Approaches**: Implementing more advanced feature selection techniques could further refine the process. For instance, using genetic

algorithms (GA) to evolve optimal subsets of features by combining filter and embedded methods (e.g., Mutual Information and Lasso) can yield highly informative feature sets:

- Using genetic algorithms to optimize feature selection.
- Combining filter and embedded methods for more comprehensive feature selection.

3. **Dimensionality Reduction Techniques**: Exploring non-linear dimensionality reduction methods such as t-SNE or UMAP could help reveal hidden patterns in high-dimensional spaces, which traditional methods like PCA might miss. Additionally, autoencoders could uncover complex latent representations of features.

   - t-SNE and UMAP for non-linear dimensionality reduction.
   - Autoencoders for discovering latent feature representations.

4. **Explainability and Interpretability**: Introducing model-agnostic methods such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to explain the importance of the selected features would offer better insights into why certain features drive predictions.

   - SHAP and LIME for interpreting feature importance.
   - Detection of complex feature interactions using methods like H-statistics.