Universitat Politècnica de Catalunya

Facultad de Informática de Barcelona

Project: Credit Card Default Prediction

Machine Learning

Spring 2024

Authors:
Aayush Paudel
Sony Shrestha

Professor:
Marta Arias Vicente

Contents

Abstract

In 1990, the Taiwanese government permitted the formation of new banks. Initially, these banks conducted business with real estate companies, but as the market became saturated, they shifted to the credit card business. To capture market share, banks over-issued credit cards to unqualified applicants. This led to many cardholders accumulating significant credit card debts, resulting in a high rate of defaults. This project aims to predict which customers are likely to default on their credit card payments. By identifying potential defaulters, banks can take proactive measures to mitigate losses.

1. Introduction

The purpose of this project is to predict which customers are likely to default on their credit card payments. By identifying potential defaulters, banks can take proactive measures to mitigate losses.

2. Data Description

The dataset contains information about 30,000 credit card clients from a Taiwanese bank. Each row represents a customer, with features including demographic information, credit limit, billing amounts, and payment history. The target variable is `DEFAULT`, which indicates whether the customer defaulted on their payment (1) or not (0).

Key features in the dataset include:

- `LIMIT_BAL`: Amount of given credit (includes both the individual consumer credit and his/her family)
- `SEX`: Gender (1 = male, 2 = female)
- `EDUCATION`: Education level (1 = graduate school, 2 = university, 3 = high school, 4 = others)
- `MARRIAGE`: Marital status (1 = married, 2 = single, 3 = others)
- `AGE`: Age of the customer
- `PAY_1 to PAY_6`: History of past payment status
- `BILL_AMT1 to BILL_AMT6`: Amount of bill statement
- `PAY_AMT1 to PAY_AMT6`: Amount paid in previous months

3. Data Preprocessing

Column Renaming
To ensure consistency and readability, several columns were renamed.

```
df.rename(columns={'default payment next month': 'DEFAULT'}, inplace=True)
df.rename(columns={'PAY_0': 'PAY_1'}, inplace=True)
df.rename(columns=lambda x: x.upper(), inplace=True)
```

Handling Missing Values
The dataset was checked for missing values, confirming there were none.

```
    print(df.isnull().sum())
```

Data Cleaning
Rows with unspecified or irrelevant categories in the MARRIAGE and EDUCATION columns were removed to ensure data quality.

```
    df = df.drop(df[df['MARRIAGE'] == 0].index)
    df = df.drop(df[df['EDUCATION'] == 0].index)
    df = df.drop(df[df['EDUCATION'] == 5].index)
    df = df.drop(df[df['EDUCATION'] == 6].index)
```

Feature Engineering
Interaction terms and new features were created to enhance the dataset with more meaningful features.

```
df['LIMIT_AGE_INTERACTION'] = df['LIMIT_BAL'] * df['AGE']
df['ON_TIME_PAYMENT_RATIO'] = (df[['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5',
    'PAY_6']] <= 0).sum(axis=1) / 6
df['MAX_UTILIZATION_RATIO'] = df[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', '
    BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].max(axis=1) / df['LIMIT_BAL']
```

New features, ON_TIME_PAYMENT_RATIO and MAX_UTILIZATION_RATIO, were created to enhance the dataset.

ON_TIME_PAYMENT_RATIO measures the proportion of on-time payments over six months. It is calculated by checking if the payment status columns (PAY_1 to PAY_6) are less than or equal to 0, summing these values, and dividing by 6.

MAX_UTILIZATION_RATIO captures the highest credit card utilization ratio over the same period. It is computed by taking the maximum value of the billing amount columns (BILL_AMT1 to BILL_AMT6) and dividing by the credit limit (LIMIT_BAL).

These features provide valuable insights into payment behavior and credit utilization, potentially improving the prediction of the DEFAULT variable.

Encoding Payment Features
The payment status features were standardized to have consistent encoding.

```
pay_features = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
for p in pay_features:
    df.loc[df[p] < 0, p] = -1
    df.loc[df[p] >= 0, p] = df.loc[df[p] >= 0, p] + 1
    df[p] = df[p].astype('int64')
```

4. Exploratory Data Analysis

Count Plot

The class distribution was examined to understand the imbalance in the dataset.

```
print(df['DEFAULT'].value_counts())
```

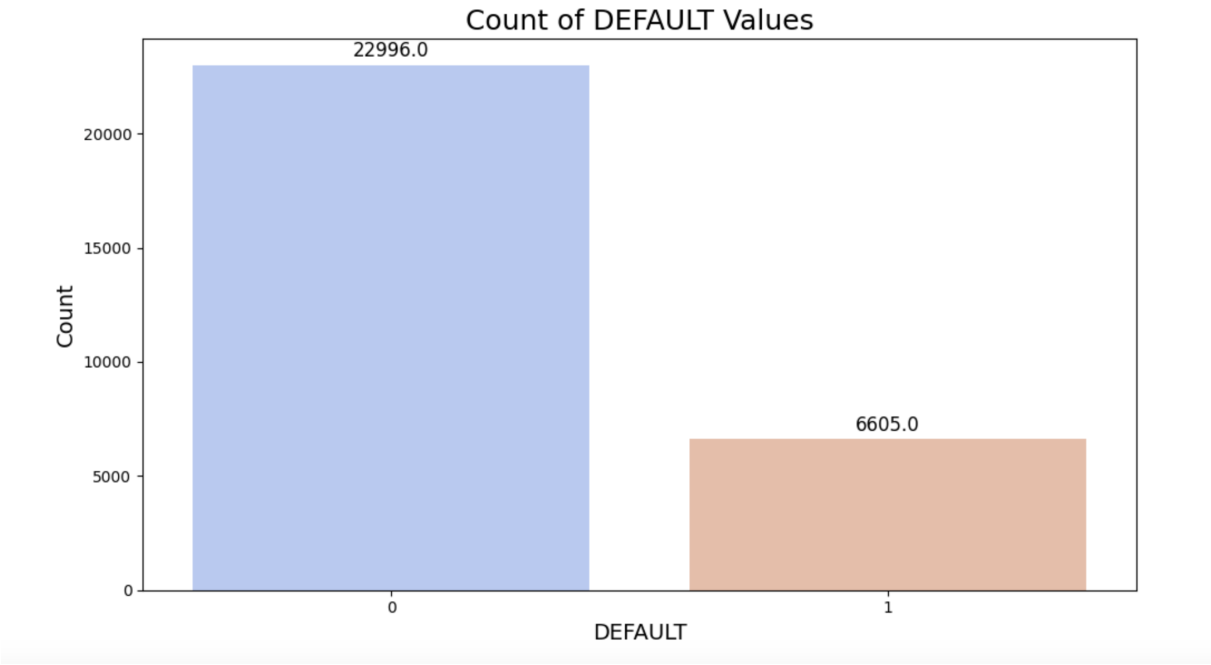

Figure 1: Class Imbalance in the Dataset

The dataset is imbalanced, with significantly more non-default instances (22,996) than default instances (6,605).

Distribution Based on Features

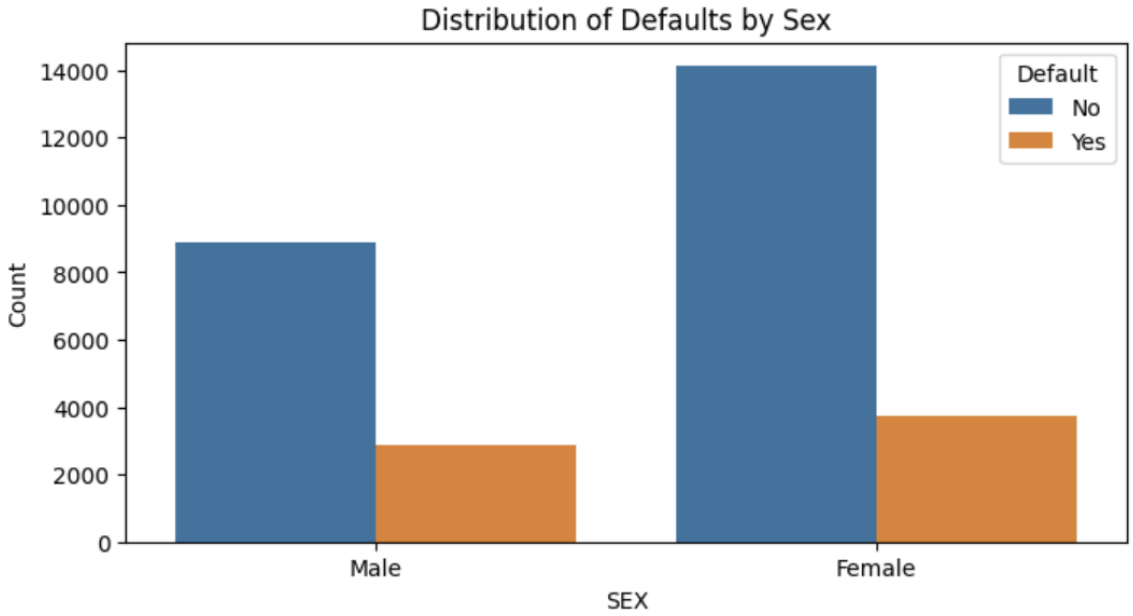

Figure 2: Distribution By Sex

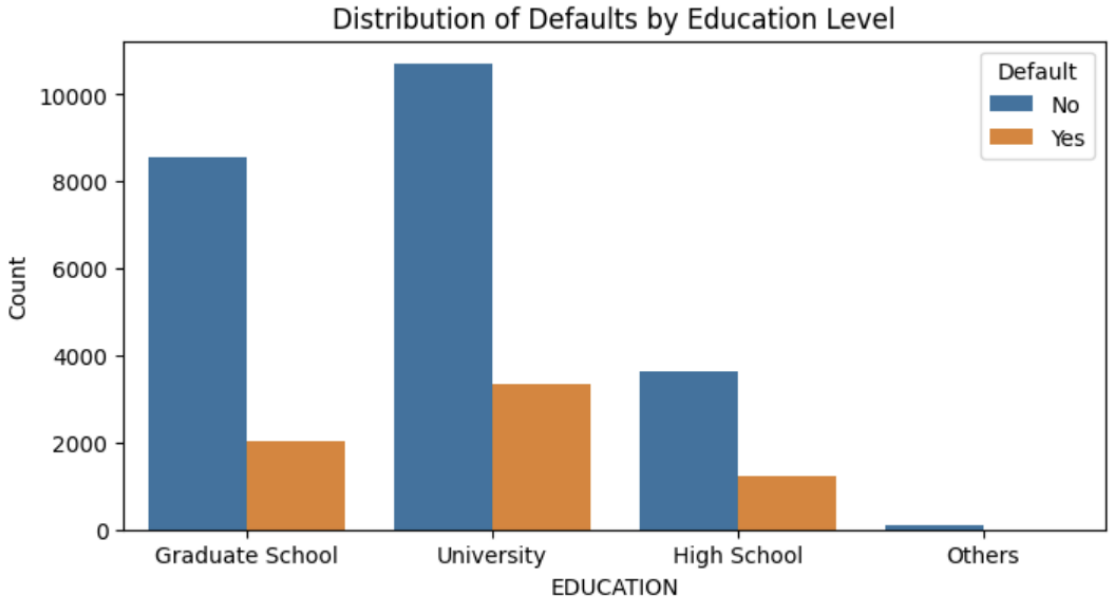

Figure 3: Distribution by Education

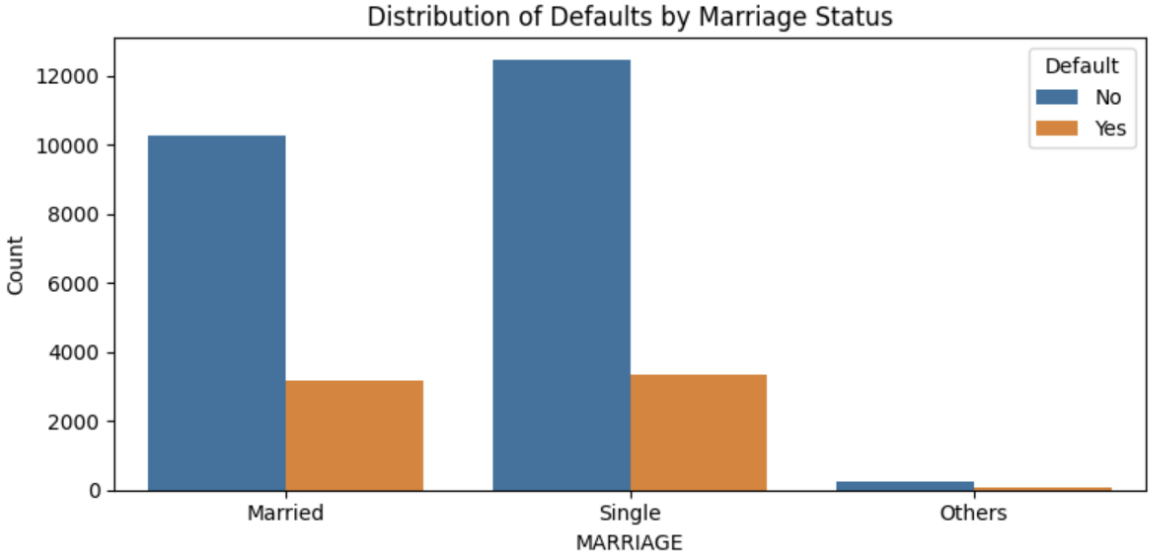

Figure 4: Distribution by Marital Status

The bar charts illustrate the distribution of loan defaults among different groups categorized by sex, education level, and marriage status. For both males and females, the majority did not default, with females having a higher count of non-defaulters compared to males. In terms of education, individuals with a university education had the highest number of non-defaulters, but also a noticeable number of defaulters, while those in the 'Others' category had the fewest defaults. Regarding marital status, single individuals had the highest number of non-defaulters, followed closely by married individuals, with both groups showing a significant difference between non-defaulters and defaulters. Overall, across all categories, non-defaulters were consistently in the majority.

Correlation Matrix Heatmap

The correlation matrix heatmap, shown in Figure 5, visualizes the relationships between different features in the dataset. The matrix includes features such as `LIMIT_BAL`, `AGE`, payment statuses (`PAY_1` to `PAY_6`), bill amounts (`BILL_AMT1` to `BILL_AMT6`), and the target variable `DEFAULT`.

Key observations include:

- Payment status features (`PAY_1` to `PAY_6`) show strong positive correlations with each other, indicating that the payment behavior of individuals tends to be consistent over time.
- Bill amounts (`BILL_AMT1` to `BILL_AMT6`) are also highly correlated with each other, suggesting that the billing amounts over the months are related.
- The target variable `DEFAULT` has notable positive correlations with the payment status features, especially `PAY_0` to `PAY_5`, and relatively weaker correlations with bill amounts.

  Overall, the heatmap provides insights into which features are closely related, helping to identify potential redundancies and inform feature selection and engineering for the modeling process.

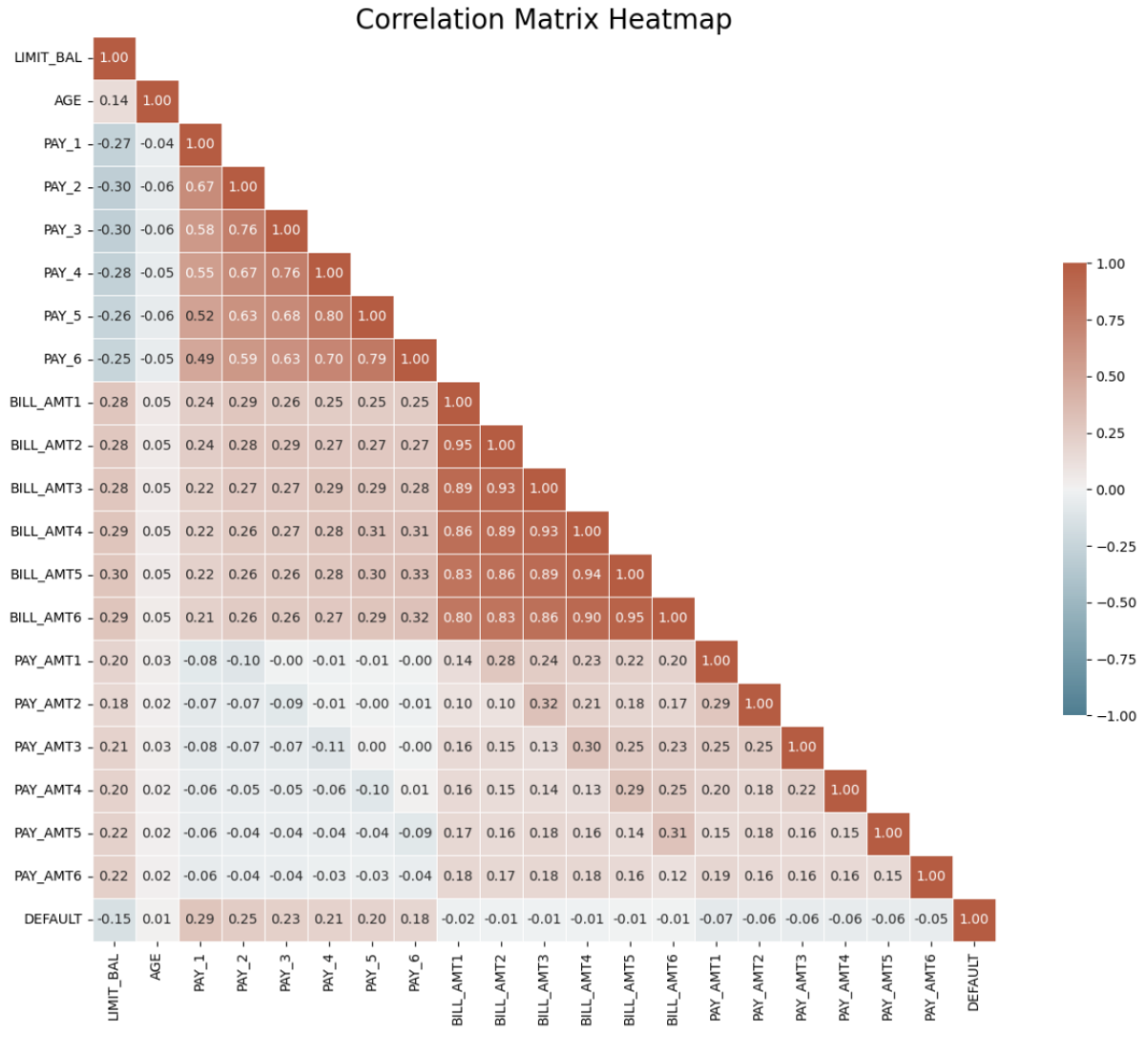

Figure 5: Correlation Matrix Heatmap

5. Data Splitting

The dataset was split into training and testing sets to evaluate the model performance effectively.

```
from sklearn.model_selection import train_test_split
y = df['DEFAULT']
X = df.drop('DEFAULT', axis=1)

X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y,
    random_state=24, stratify=y)
```

The `random_state` parameter is set to 24 to ensure reproducibility, and `stratify=y` is used to maintain the same proportion of the target classes in both the training and test sets, which is crucial for handling class imbalance effectively.

6. Feature Scaling

Different scaling techniques were applied to the training and testing datasets to ensure the features were on a similar scale, which is crucial for many machine learning algorithms.

6.1. Min-Max Scaling

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

6.2. Robust Scaling

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
```

6.3. Standard Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

We applied Robust Scaling to our dataset using `RobustScaler` to handle outliers effectively. Unlike other scaling methods, Robust Scaling uses the median and interquartile range, making it less sensitive to extreme values. This ensures that outliers do not disproportionately affect the scaling process, leading to more reliable and stable model performance. We specifically scaled `LIMIT_BAL`, `AGE`, `PAY_1` to `PAY_6`, `BILL_AMT1` to `BILL_AMT6`, and `PAY_AMT1` to `PAY_AMT6` due to their numerical nature and potential outliers.

7. Principal Component Analysis (PCA)

PCA was performed to reduce the dimensionality of the dataset while retaining most of the variance. This helps in improving model performance and reducing overfitting.

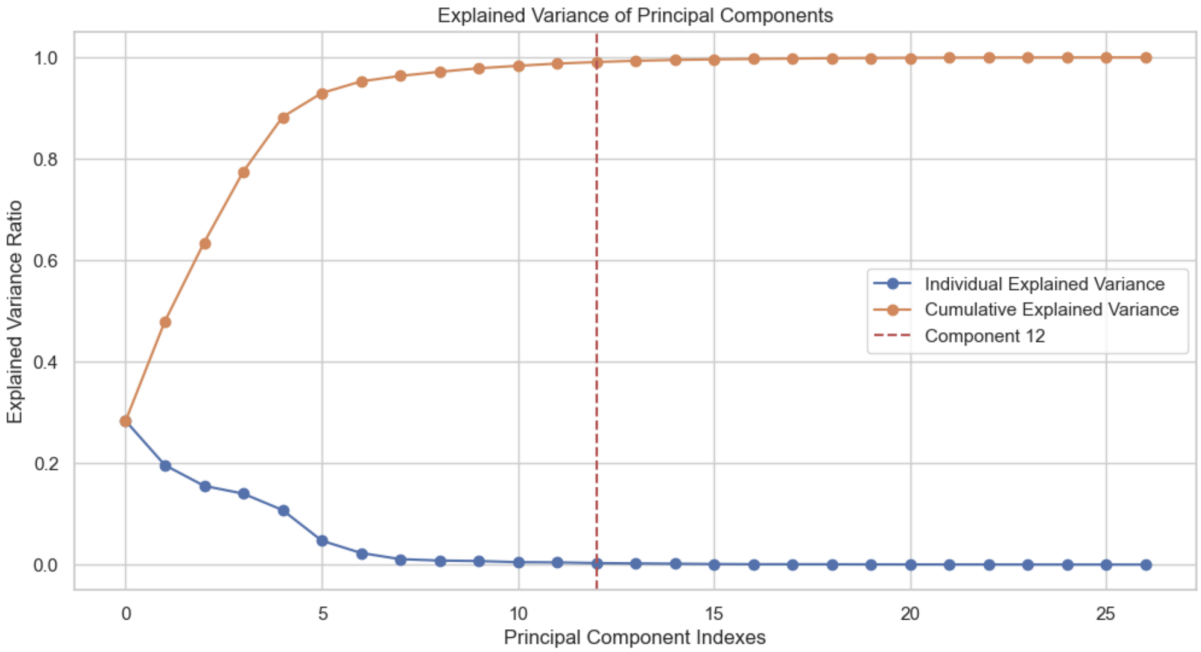

Figure 6: Principal Component Analysis

7.1. Number of Principal Components

```
pc = 12
```

In this case, we chose to retain 12 principal components (PCs), which cumulatively account for 98.79% of the total variance. This selection strikes a balance between reducing dimensionality and preserving the essential information in the dataset. By capturing nearly all of the variance with these 12 components, we ensure that the majority of the data's structure and important features are retained, while significantly simplifying the dataset to improve computational efficiency and potentially enhance the performance of subsequent machine learning models.

8. Handling Class Imbalance

Various resampling techniques were applied to handle the class imbalance and improve model performance.

8.1. Cluster Centroids Undersampling

Cluster Centroids is an undersampling technique that reduces the number of majority class samples by creating centroids of clusters formed by the majority class. In this process, the `ClusterCentroids` class from the `imblearn.under_sampling` module is initialized with a specified random state to ensure reproducibility. The method is applied to the training data (`X_train` and `y_train`), resulting in a balanced dataset (`X_train_cc` and `y_train_cc`). The class proportions are then calculated and displayed, demonstrating the effectiveness of the undersampling in achieving a more balanced dataset by reducing the instances of the majority class while retaining the minority class.

```
from imblearn.under_sampling import ClusterCentroids
```

```
undersample = ClusterCentroids(random_state=24)
X_train_cc, y_train_cc = undersample.fit_resample(X_train, y_train)
```

8.2. SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE (Synthetic Minority Over-sampling Technique) is an oversampling method used to balance class distribution by generating synthetic samples of the minority class. Using the `SMOTE` class from the `imblearn.over_sampling` module, the technique is applied to the training dataset. By initializing SMOTE with a random state, reproducibility is ensured. The synthetic samples are generated in the feature space, resulting in a new balanced training set (`X_train_smote` and `y_train_smote`). Class proportions are then computed and displayed, showing an equal representation of both classes, which helps in improving the performance of machine learning models by mitigating the bias towards the majority class.

```
from imblearn.over_sampling import SMOTE

oversample = SMOTE(random_state=24)
X_train_smote, y_train_smote = oversample.fit_resample(X_train, y_train)
```

8.3. KMeans SMOTE Over-sampling

KMeansSMOTE combines the principles of k-means clustering and SMOTE to perform more controlled oversampling of the minority class. The **KMeansSMOTE** class from `imblearn.over_sampling` is initialized with a very low `cluster_balance_threshold` and a random state for reproducibility. This technique involves clustering the dataset and then applying SMOTE within each cluster to generate synthetic samples. The result is a balanced dataset (`X_train_ksmote` and `y_train_ksmote`) that maintains the inherent structure of the data. The class proportions are calculated and displayed, indicating successful oversampling that provides an even distribution of the classes, thereby aiding in more robust model training.

```
from imblearn.over_sampling import KMeansSMOTE

oversample = KMeansSMOTE(cluster_balance_threshold=0.00001, random_state
    =24)
X_train_ksmote, y_train_ksmote = oversample.fit_resample(X_train, y_train)
```

Comparison of Sampling Methods

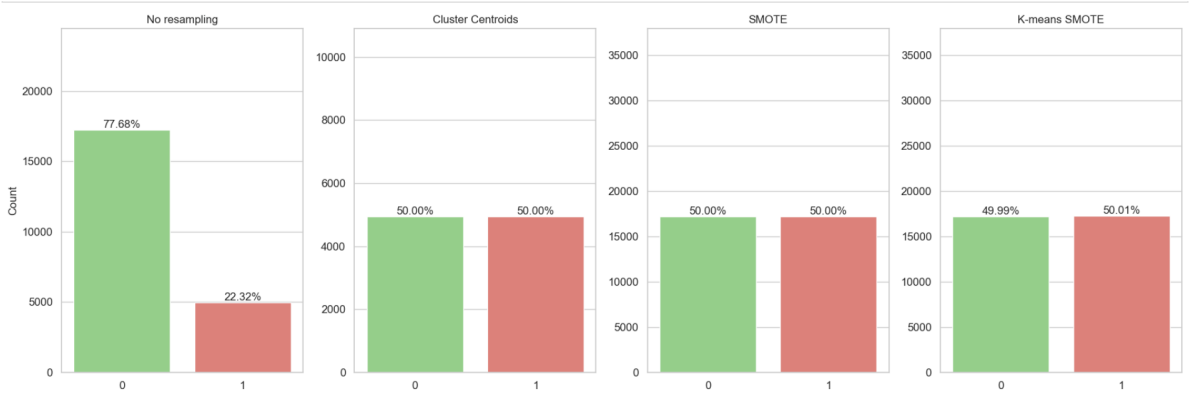

Figure 7: A Comparative Analysis of Undersampling and Oversampling Techniques

9. Model Evaluation

Four classifiers were evaluated using different resampling techniques to determine the best model for predicting credit card defaults. The classifiers used were Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine (SVM).

9.1. Logistic Regression

Logistic Regression is a statistical method used for binary classification that predicts the probability of an instance belonging to one of two classes. It is based on the logistic function, which maps any real-valued number into a value between 0 and 1. The key feature of logistic regression is its interpretability, as it provides coefficients that indicate the influence of each feature on the prediction.

Mathematical Representation:

$$P(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}}$$

where $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients learned from the data.

Hyperparameters:

```
params_lr = {'C': [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2]}
```

9.2. Decision Tree

Decision Tree is a non-linear classifier that splits the data into subsets based on feature values, forming a tree-like structure. Each internal node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome. Decision trees are intuitive and easy to interpret, making them a popular choice for classification tasks.

Hyperparameters

```
params_tree = {'max_depth': [5, 10, 20, 30, 50]}
```

9.3. Random Forest

Random Forest is an ensemble method that builds multiple decision trees and merges their outputs to get a more accurate and stable prediction. It combines the concept of "bagging" (Bootstrap Aggregating) and random feature selection to create a diverse set of trees. The final prediction is made by averaging the predictions of all trees (for regression) or taking the majority vote (for classification).

Hyperparameters

```
params_rf = {'n_estimators': [10, 50, 100, 200], 'max_features': [None, '
    sqrt']}
```

9.4. Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful classifier that aims to find the optimal hyperplane that best separates the data into different classes. The hyperplane is chosen to maximize the margin between the classes, which is the distance between the hyperplane

and the nearest data points from each class, known as support vectors. SVM can be used for both linear and non-linear classification by applying kernel functions.

Mathematical Representation
For a linear SVM, the decision function is:

$$f(x) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

where $\mathbf{w}$ is the weight vector and b is the bias term. For non-linear SVMs, kernel functions like the radial basis function (RBF) can be used to transform the data into a higher-dimensional space.

Hyperparameters

```
params_svm = {'C': [1e-1, 1e0, 1e1, 1e2], 'kernel': ['rbf', 'poly'], 'gamma
    ': [1e-1, 'scale']}
```

9.5 Gradient Boosting

Gradient Boosting is an ensemble learning technique that builds a strong predictive model by sequentially adding weak learners, typically decision trees, to correct the errors of the combined ensemble. Each new model is trained to minimize a loss function by fitting to the residual errors of the previous models. The algorithm iteratively updates the model by adding a new learner $h_m(x)$ that aims to reduce the prediction error $y - F_{m-1}(x)$, where $F_{m-1}(x)$ is the ensemble prediction from the previous iteration. Mathematically, the model at iteration m is updated as:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x)$$

where η is the learning rate that controls the contribution of each new learner. This iterative process continues until the maximum number of iterations or trees is reached, producing a highly accurate model that is particularly effective for both regression and classification tasks.

10. Results and Discussion

The performance of each classifier was evaluated using various resampling techniques to handle class imbalance. The F1-score, being a harmonic mean of precision and recall, was the primary metric used to assess model performance. The following sections summarize the results of each classifier.

10.1. Logistic Regression, Decision Tree, and Random Forest

The evaluation of Logistic Regression, Decision Tree, and Random Forest classifiers using different resampling techniques shows that using raw data generally provides the most balanced performance across all metrics, including accuracy, F1-score, and AUC. PCA alone tends to reduce model performance, while PCA combined with SMOTE or KMeansSMOTE improves recall significantly but often at the expense of precision and accuracy. The PCA + ClusterCentroids technique achieves the highest recall, indicating a strong ability to detect defaults, but it introduces many false positives, resulting in lower overall accuracy and precision. Overall, raw data without resampling maintains a better balance of predictive performance for credit card default prediction.

Logistic Regression

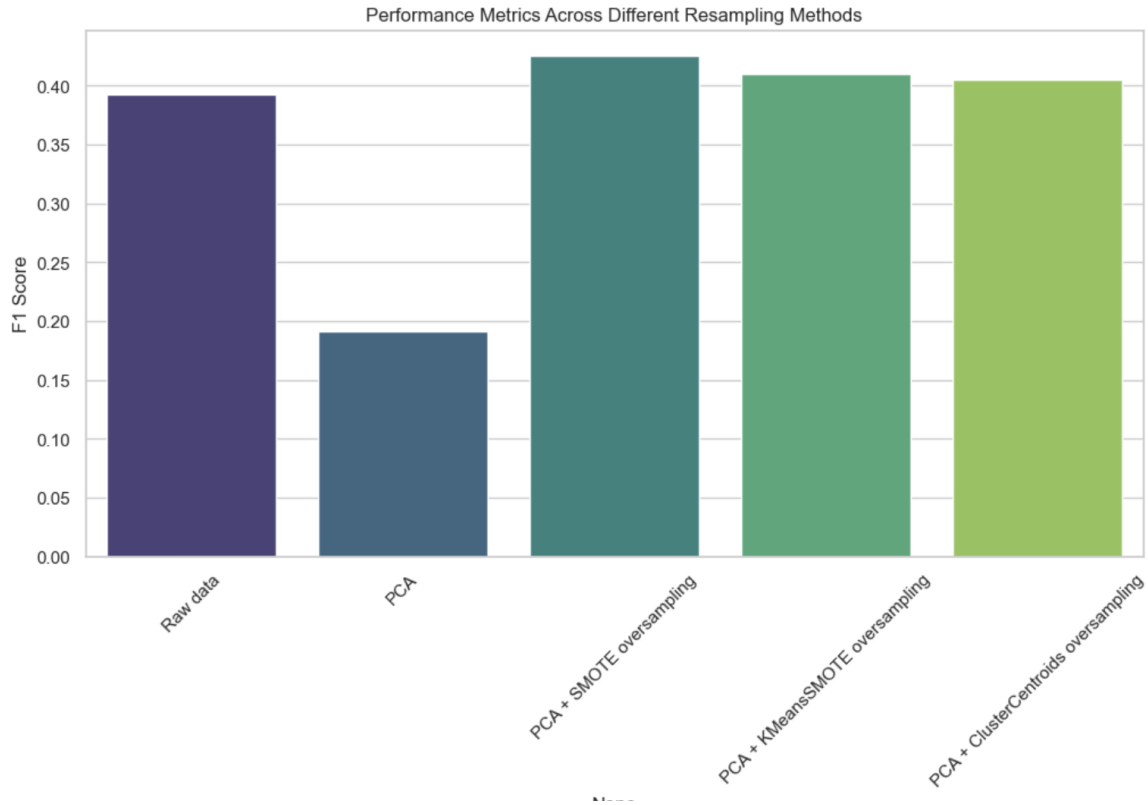

Figure 8: Logistic Regression Results with Different Resampling Techniques

Decision Tree

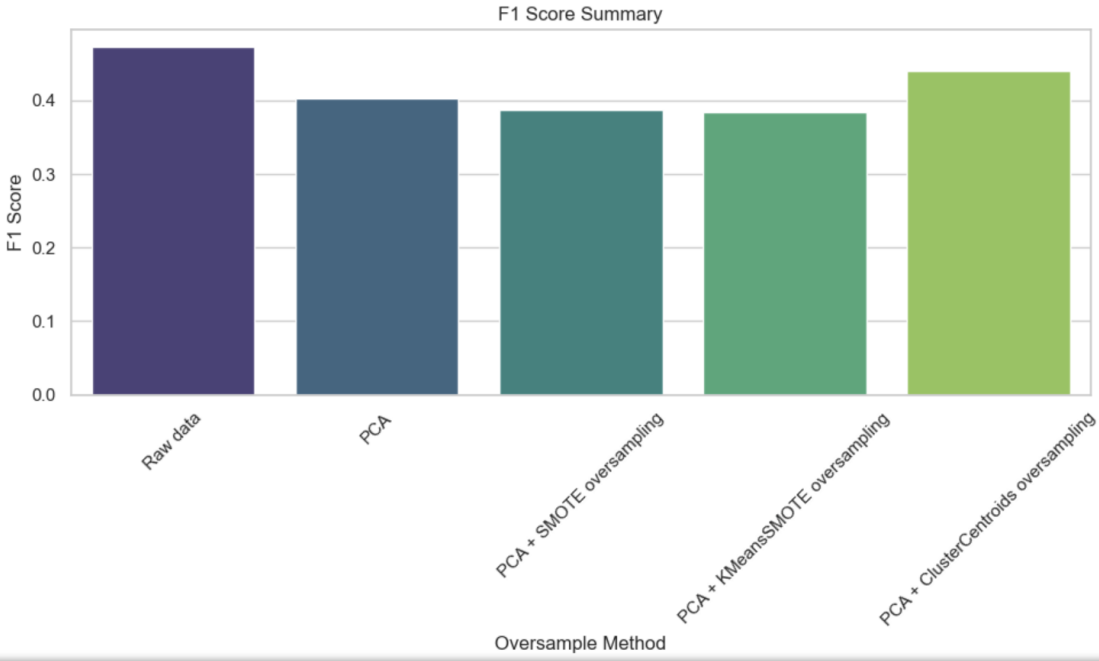

Figure 9: Decision Tree Results with Different Resampling Techniques

Random Forest

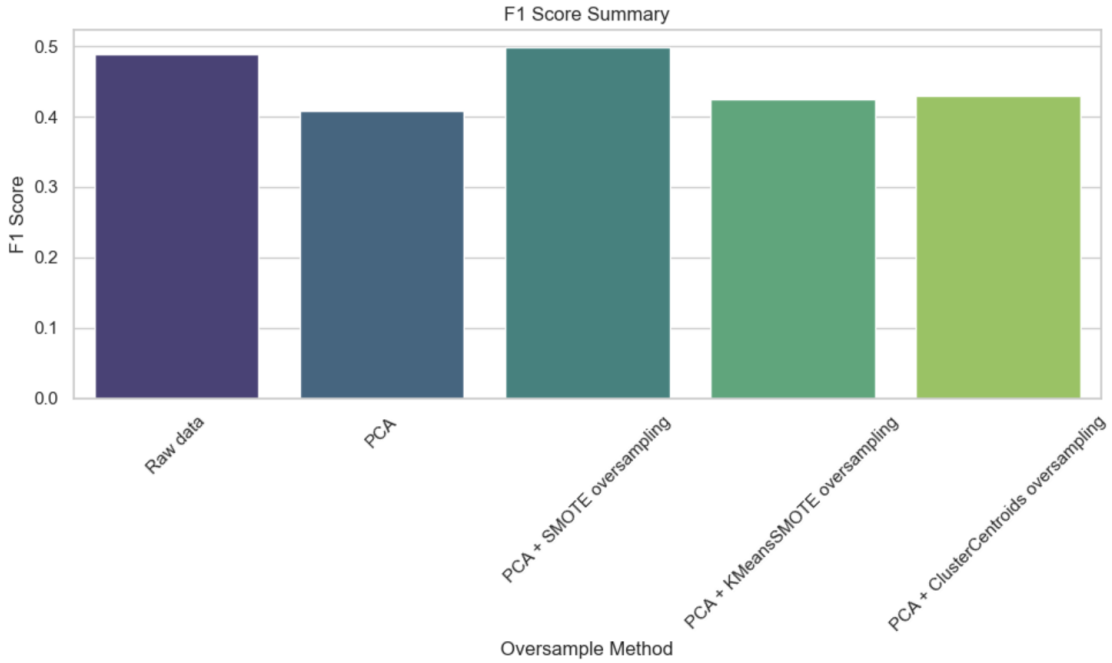

Figure 10: Random Forest Results with Different Resampling Techniques

Support Vector Machine

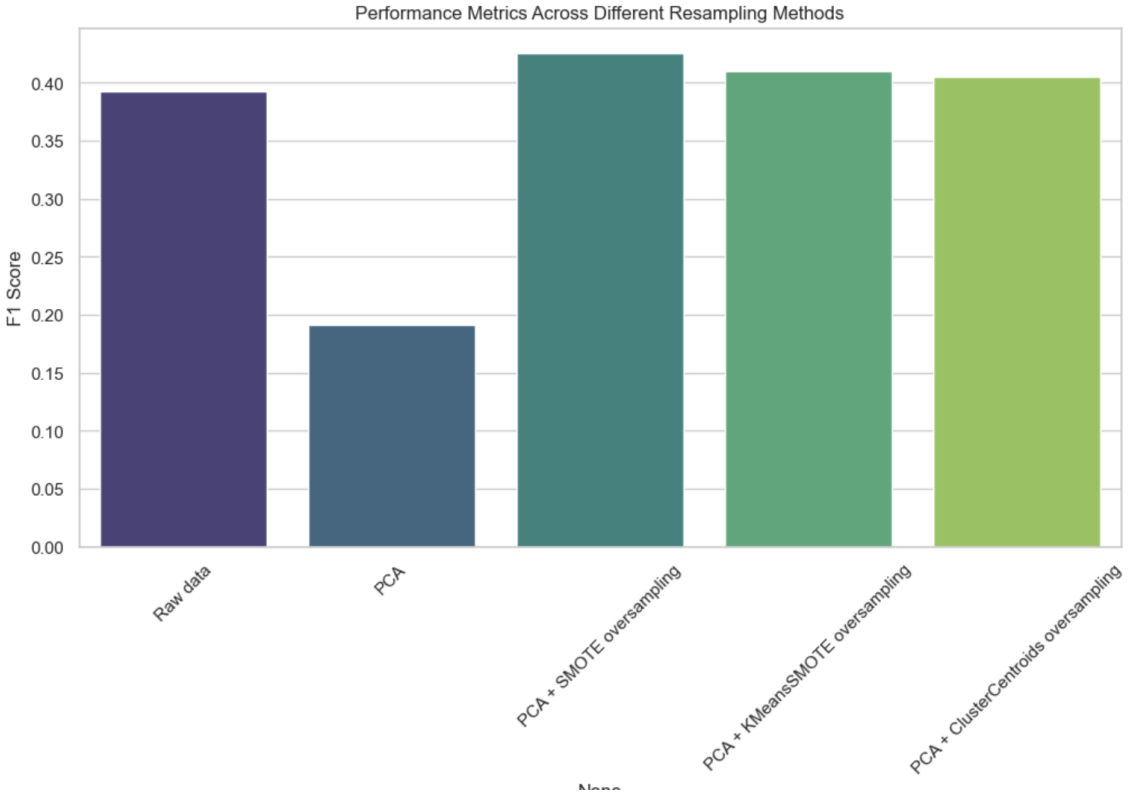

Figure 11: SVM Results with Different Resampling Techniques

Gradient Boosting

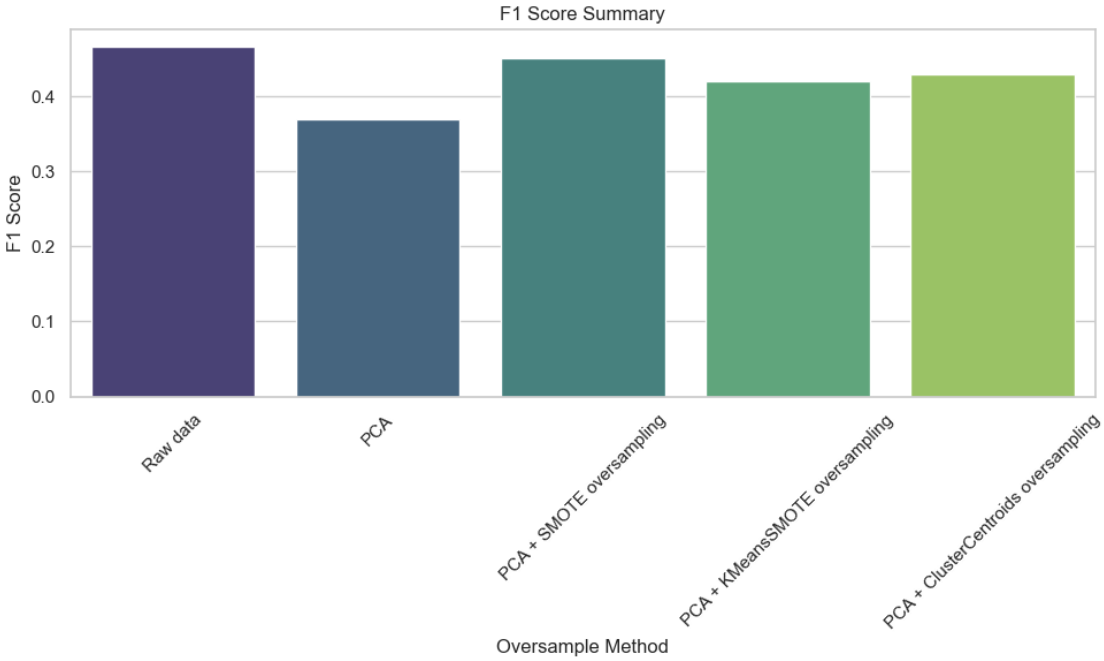

Figure 12: Gradient Boosting Results with Different Resampling Techniques

Due to computational limitations, the Support Vector Machine (SVM) and Gradient Boosting model took an excessively long time to process using GridSearchCV. Thus we made use of RandomSearchCV for model traning and validation. Analysis shows that for Gradient Boosting, using raw data achieves high accuracy (82%) and precision (69%) but low recall (35%). PCA with SMOTE improves recall (48%) but reduces precision (43%) and accuracy (74%). For SVM, PCA with SMOTE significantly improves performance, increasing the F1-score from 0.35 (raw data and PCA) to 0.50 and AUC from 0.60 to 0.69. PCA with KMeansSMOTE results in a poor balance for both models. Thus, PCA with SMOTE emerges as the most effective technique for both Gradient Boosting and SVM, enhancing their ability to balance minority class instances.

Model Comparison Chart

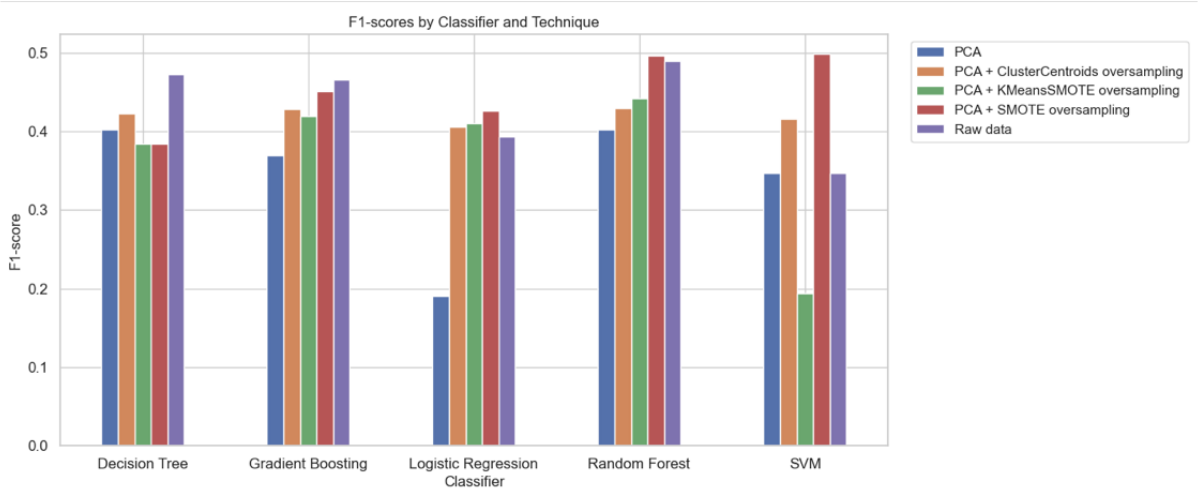

Figure 13: Model Comparisons

11. Conclusion

This project provided a comprehensive approach to predicting credit card defaults. By preprocessing the data, creating meaningful features, handling class imbalance, and visualizing key patterns, we developed robust predictive models. The findings help in identifying potential defaulters, allowing banks to take proactive measures to mitigate losses.

The Random Forest classifier with PCA + SMOTE oversampling demonstrated the highest F1-score, making it the most effective model for this task. The results indicate that applying PCA for dimensionality reduction followed by SMOTE oversampling to balance the classes significantly improves model performance.

12. Future Work

Future work could explore the following.

- Incorporating additional features such as transaction history or external credit scores to enhance the predictive power of the models.
- Using advanced ensemble techniques or deep learning models to further improve the accuracy and robustness of predictions.
- Exploring other oversampling and undersampling techniques to see if there are better alternatives to the ones used in this project.

By continuing to refine and enhance these models, banks can better manage credit risk and reduce the incidence of defaults, thereby maintaining financial stability and consumer confidence.

13. References

In this analysis, we have taken reference from the following source GitHub link.