**Comprehensive Project Report: Credit Card Default Prediction**

**Introduction**

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. A credit card default occurs when a cardholder fails to meet their repayment obligations, causing a crisis in consumer finance confidence.

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.

**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 month.

**Data Preprocessing**

**1. 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)

**2. Handling Missing Values:** The dataset was checked for missing values, and it was confirmed that there were no missing entries.

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

**3. 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)

**4. Feature Engineering:** Interaction terms and month-to-month changes in bill amounts were created to enhance the dataset with more meaningful features.

df['LIMIT\_AGE\_INTERACTION'] = df['LIMIT\_BAL'] \* df['AGE']

df['BILL\_CHANGE1'] = df['BILL\_AMT2'] - df['BILL\_AMT1']

df['BILL\_CHANGE2'] = df['BILL\_AMT3'] - df['BILL\_AMT2']

df['BILL\_CHANGE3'] = df['BILL\_AMT4'] - df['BILL\_AMT3']

df['BILL\_CHANGE4'] = df['BILL\_AMT5'] - df['BILL\_AMT4']

df['BILL\_CHANGE5'] = df['BILL\_AMT6'] - df['BILL\_AMT5']

df

**5. 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')

**Exploratory Data Analysis**

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

print(df['DEFAULT'].value\_counts())

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

**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

# Defining the features and target variable

y = df['DEFAULT']

X = df.drop('DEFAULT', axis=1)

# Splitting the dataset into training and test sets

X\_train\_raw, X\_test\_raw, y\_train, y\_test = train\_test\_split(X, y, random\_state=24, stratify=y)

# Print training and test set shapes and counts

print(f'Training set shape: {X\_train\_raw.shape}')

print(f'- Defaulters: {y\_train.sum()}')

print(f'- Non-defaulters: {len(y\_train) - y\_train.sum()}')

print(f'Test set shape: {X\_test\_raw.shape}')

print(f'- Defaulters: {y\_test.sum()}')

print(f'- Non-defaulters: {len(y\_test) - y\_test.sum()}')

**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.

**1. Min-Max Scaling:**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_train\_norm = X\_train\_raw.copy()

X\_test\_norm = X\_test\_raw.copy()

columns\_list = ['LIMIT\_BAL', 'AGE'] + [f'PAY\_{i}' for i in range(1, 7)] + [f'BILL\_AMT{i}' for i in range(1, 7)] + [f'PAY\_AMT{i}' for i in range(1, 7)]

for column in columns\_list:

X\_train\_norm[column] = scaler.fit\_transform(X\_train\_raw[column].values.reshape(-1, 1))

X\_test\_norm[column] = scaler.transform(X\_test\_raw[column].values.reshape(-1, 1))

**2. Robust Scaling:**

from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X\_train\_rob = X\_train\_raw.copy()

X\_test\_rob = X\_test\_raw.copy()

for column in columns\_list:

X\_train\_rob[column] = scaler.fit\_transform(X\_train\_raw[column].values.reshape(-1, 1))

X\_test\_rob[column] = scaler.transform(X\_test\_raw[column].values.reshape(-1, 1))

**3. Standard Scaling:**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_std = X\_train\_raw.copy()

X\_test\_std = X\_test\_raw.copy()

for column in columns\_list:

X\_train\_std[column] = scaler.fit\_transform(X\_train\_raw[column].values.reshape(-1, 1))

X\_test\_std[column] = scaler.transform(X\_test\_raw[column].values.reshape(-1, 1))

**Handling Class Imbalance**

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

1. **Cluster Centroids Undersampling:**

**Cluster Centroids Undersampling** is a technique used to balance the class distribution by reducing the number of majority class instances. The main idea is to identify clusters of majority class samples and replace each cluster with its centroid. This approach helps to retain the structure of the majority class while reducing its size, making the dataset more balanced. It is particularly useful when the dataset is heavily imbalanced, as it reduces the risk of overfitting by eliminating redundant majority class samples.

from imblearn.under\_sampling import ClusterCentroids

oversample = ClusterCentroids(random\_state=24)

X\_train\_cc, y\_train\_cc = oversample.fit\_resample(X\_train, y\_train)

1. **SMOTE:**

#### SMOTE (Synthetic Minority Over-sampling Technique)

**SMOTE** is an oversampling technique that generates synthetic samples for the minority class to balance the class distribution. It works by selecting a sample from the minority class and generating new samples along the line segments joining the selected sample and its nearest neighbors. This method helps in increasing the minority class instances without simply duplicating them, leading to a more generalized model.

from imblearn.over\_sampling import SMOTE

oversample = SMOTE(random\_state=24)

X\_train\_smote, y\_train\_smote = oversample.fit\_resample(X\_train, y\_train)

1. **KMeans SMOTE:**

#### KMeans SMOTE

**KMeans SMOTE** is a variation of the traditional SMOTE technique that incorporates clustering before generating synthetic samples. The idea is to apply KMeans clustering to the minority class samples, group them into clusters, and then apply SMOTE within each cluster. This approach aims to create more meaningful synthetic samples by considering the inherent structure within the minority class data.

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)

**Model Evaluation**

**Metrics Overview**

1. **Accuracy**: The proportion of correctly predicted instances (both true positives and true negatives) out of the total instances.
2. **Recall**: The ability of the model to identify all relevant instances (true positives). High recall indicates that the model is effective in capturing most of the defaulters.
3. **Precision**: The proportion of true positives out of all predicted positives. High precision means that when the model predicts a default, it is likely correct.
4. **F1-score**: The harmonic mean of precision and recall. It provides a balance between the two metrics and is useful when the class distribution is imbalanced.
5. **AUC (Area Under the ROC Curve)**: A measure of the model's ability to distinguish between classes. A higher AUC indicates better performance.

**Resampling Techniques**

1. **Raw Data**: The model was trained on the original dataset without any resampling.
2. **PCA**: Principal Component Analysis was applied to reduce the dimensionality of the data before training.
3. **PCA + SMOTE Oversampling**: SMOTE was applied after PCA to generate synthetic samples of the minority class.
4. **PCA + KMeansSMOTE Oversampling**: KMeansSMOTE, a variant of SMOTE, was used after PCA to create synthetic samples.
5. **PCA + ClusterCentroids Oversampling**: Cluster centroids undersampling was used after PCA to balance the classes by reducing the majority class.

Three classifiers were evaluated using different resampling techniques to determine the best model for predicting credit card defaults.

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.

**Oversampling Methods:**

* Raw data
* PCA
* PCA + SMOTE oversampling
* PCA + KMeansSMOTE oversampling
* PCA + ClusterCentroids oversampling

**Hyperparameters:**

params\_lr = {'C': [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2]}

**Results:**

* Performance metrics including Accuracy, Recall, Precision, F1-score, and AUC were collected for each method.
* The results were plotted to visualize the performance across different resampling techniques.

The choice of technique depends on the specific business requirement. If the goal is to capture as many defaulters as possible (high recall), PCA + ClusterCentroids may be preferred. However, if the goal is to maintain a balance between capturing defaulters and maintaining overall prediction accuracy (precision and F1-score), PCA + SMOTE may be a better choice.

1. **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.

**Oversampling Methods:**

* Raw data
* PCA
* PCA + SMOTE oversampling
* PCA + KMeansSMOTE oversampling
* PCA + ClusterCentroids oversampling

**Hyperparameters:**

params\_tree = {'max\_depth': [5, 10, 20, 30, 50]}

**Results:**

* Performance metrics including Accuracy, Recall, Precision, F1-score, and AUC were collected for each method.
* The results were plotted to visualize the performance across different resampling techniques.
* The choice of technique depends on the specific business requirement. If the goal is to capture as many defaulters as possible (high recall), PCA + ClusterCentroids may be preferred. However, if the goal is to maintain a balance between capturing defaulters and maintaining overall prediction accuracy (precision and F1-score), PCA + SMOTE may be a better choice.

1. **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).

**Oversampling Methods:**

* Raw data
* PCA
* PCA + SMOTE oversampling
* PCA + KMeansSMOTE oversampling
* PCA + ClusterCentroids oversampling

**Hyperparameters:**

params\_rf = {'n\_estimators': [10, 50, 100, 200], 'max\_features': [None, 'sqrt']}

**Results:**

* Performance metrics including Accuracy, Recall, Precision, F1-score, and AUC were collected for each method.
* The results were plotted to visualize the performance across different resampling techniques.

**Summary**

* **Raw Data**: Best for precision, accuracy, and overall balance between recall and precision (highest F1-score).
* **PCA**: Lower overall performance, indicating some loss of information.
* **PCA + SMOTE**: Significantly improved recall at the cost of precision and accuracy. Best balance between recall and precision among resampling methods.
* **PCA + KMeansSMOTE**: Similar to SMOTE but slightly less effective. Balanced but not as good as SMOTE.
* **PCA + ClusterCentroids**: Highest recall, moderate F1-score, but significantly reduced accuracy and precision.

The choice of technique depends on the specific business requirement. If the goal is to capture as many defaulters as possible (high recall), PCA + ClusterCentroids may be preferred. However, if the goal is to maintain a balance between capturing defaulters and maintaining overall prediction accuracy (precision and F1-score), PCA + SMOTE may be a better choice. Raw data also shows strong performance and may be suitable when a balance between precision and recall is desired.

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.

**Results and Analysis**

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:

**Logistic Regression**

The Logistic Regression model was evaluated with five different resampling methods. The hyperparameters were tuned using grid search, focusing on the regularization parameter **C**. The results showed that:

* **Raw data** and **PCA** generally performed well.
* **PCA + SMOTE** and **PCA + KMeansSMOTE** oversampling techniques improved the F1-score significantly.
* **PCA + ClusterCentroids** undersampling resulted in lower performance due to potential loss of valuable data during the undersampling process.

**Decision Tree**

The Decision Tree classifier was evaluated similarly, with the hyperparameters tuned for **max\_depth**. The results indicated that:

* **PCA + SMOTE** and **PCA + KMeansSMOTE** oversampling methods provided the best performance.
* **Raw data** and **PCA** had moderate performance, while **PCA + ClusterCentroids** undersampling did not perform as well due to reduced data availability.

**Random Forest**

The Random Forest classifier was evaluated with the number of estimators (**n\_estimators**) and maximum features (**max\_features**) as hyperparameters. The results showed:

* **PCA + SMOTE** and **PCA + KMeansSMOTE** consistently yielded the highest F1-scores, indicating their effectiveness in handling class imbalance.
* **Raw data** and **PCA** provided robust results, but were outperformed by the oversampling techniques.
* **PCA + ClusterCentroids** undersampling showed lower performance compared to the other methods.

**Combined Results**

The combined F1-scores of different classifiers and techniques are summarized in the following plot:

This plot provides a visual comparison of the performance of various classifiers using different oversampling techniques. It highlights the effectiveness of PCA combined with SMOTE and KMeansSMOTE oversampling methods across different classifiers.

**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.

**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.
* **Implementing cost-sensitive learning** to better handle the imbalanced dataset by assigning different misclassification costs for defaults and non-defaults.
* **Performing detailed hyperparameter tuning** using more sophisticated methods like Bayesian optimization or Random Search to optimize model performance.
* **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.

\documentclass{article}

\usepackage{graphicx}

\usepackage{float}

\usepackage{amsmath}

\title{Credit Card Default Prediction}

\author{Aayush Paudel \and Sony Shrestha}

\date{\today}

\begin{document}

\maketitle

\begin{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.

\end{abstract}

\section{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.

\section{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 \texttt{DEFAULT}, which indicates whether the customer defaulted on their payment (1) or not (0).

\textbf{Key features in the dataset include:}

\begin{itemize}

\item \texttt{LIMIT\\_BAL}: Amount of given credit (includes both the individual consumer credit and his/her family).

\item \texttt{SEX}: Gender (1 = male, 2 = female).

\item \texttt{EDUCATION}: Education level (1 = graduate school, 2 = university, 3 = high school, 4 = others).

\item \texttt{MARRIAGE}: Marital status (1 = married, 2 = single, 3 = others).

\item \texttt{AGE}: Age of the customer.

\item \texttt{PAY\\_1 to PAY\\_6}: History of past payment status.

\item \texttt{BILL\\_AMT1 to BILL\\_AMT6}: Amount of bill statement.

\item \texttt{PAY\\_AMT1 to PAY\\_AMT6}: Amount paid in previous months.

\end{itemize}

\section{Data Preprocessing}

\begin{enumerate}

\item \textbf{Column Renaming}: To ensure consistency and readability, several columns were renamed:

\begin{verbatim}

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)

\end{verbatim}

\item \textbf{Handling Missing Values}: The dataset was checked for missing values, confirming there were none:

\begin{verbatim}

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

\end{verbatim}

\item \textbf{Data Cleaning}: Rows with unspecified or irrelevant categories in the \texttt{MARRIAGE} and \texttt{EDUCATION} columns were removed to ensure data quality:

\begin{verbatim}

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)

\end{verbatim}

\item \textbf{Feature Engineering}: Interaction terms and month-to-month changes in bill amounts were created to enhance the dataset with more meaningful features:

\begin{verbatim}

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']

\end{verbatim}

\end{verbatim}

\item \textbf{Encoding Payment Features}: The payment status features were standardized to have consistent encoding:

\begin{verbatim}

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')

\end{verbatim}

\end{enumerate}

\section{Exploratory Data Analysis}

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

\begin{verbatim}

print(df['DEFAULT'].value\_counts())

\end{verbatim}

\begin{figure}[H]

\centering

\includegraphics[width=0.8\textwidth]{src/img/class\_imbalace.png}

\caption{Class Imbalance Dataset}

\end{figure}

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

\section{Data Splitting}

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

\begin{verbatim}

from sklearn.model\_selection import train\_test\_split

# Defining the features and target variable

y = df['DEFAULT']

X = df.drop('DEFAULT', axis=1)

# Splitting the dataset into training and test sets

X\_train\_raw, X\_test\_raw, y\_train, y\_test = train\_test\_split(X, y, random\_state=24, stratify=y)

\end{verbatim}

\section{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.

\subsection{Min-Max Scaling}

\begin{verbatim}

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_train\_norm = X\_train\_raw.copy()

X\_test\_norm = X\_test\_raw.copy()

columns\_list = ['LIMIT\_BAL', 'AGE'] + [f'PAY\_{i}' for i in range(1, 7)] + [f'BILL\_AMT{i}' for i in range(1, 7)] + [f'PAY\_AMT{i}' for i in range(1, 7)]

for column in columns\_list:

X\_train\_norm[column] = scaler.fit\_transform(X\_train\_raw[column].values.reshape(-1, 1))

X\_test\_norm[column] = scaler.transform(X\_test\_raw[column].values.reshape(-1, 1))

\end{verbatim}

\subsection{Robust Scaling}

\begin{verbatim}

from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X\_train\_rob = X\_train\_raw.copy()

X\_test\_rob = X\_test\_raw.copy()

for column in columns\_list:

X\_train\_rob[column] = scaler.fit\_transform(X\_train\_raw[column].values.reshape(-1, 1))

X\_test\_rob[column] = scaler.transform(X\_test\_raw[column].values.reshape(-1, 1))

\end{verbatim}

\subsection{Standard Scaling}

\begin{verbatim}

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_std = X\_train\_raw.copy()

X\_test\_std = X\_test\_raw.copy()

for column in columns\_list:

X\_train\_std[column] = scaler.fit\_transform(X\_train\_raw[column].values.reshape(-1, 1))

X\_test\_std[column] = scaler.transform(X\_test\_raw[column].values.reshape(-1, 1))

\end{verbatim}

\section{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.

\begin{figure}[H]

\centering

\includegraphics[width=0.8\textwidth]{src/img/pca.png}

\caption{Principal Component Analysis}

\end{figure}

\subsection{Number of Principal Components}

\begin{verbatim}

pc = 12

\end{verbatim}

\subsection{Fit PCA on the Normalized Training Data and Transform Both Training and Testing Datasets}

\begin{verbatim}

X\_train = pd.DataFrame(pca.fit\_transform(X\_train\_rob), columns=['PC' + str(i) for i in range(1, pc+1)])

X\_test = pd.DataFrame(pca.transform(X\_test\_rob), columns=['PC' + str(i) for i in range(1, pc+1)])

\end{verbatim}

\section{Handling Class Imbalance}

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

\subsection{Cluster Centroids Undersampling}

\begin{verbatim}

from imblearn.under\_sampling import ClusterCentroids

undersample = ClusterCentroids(random\_state=24)

X\_train\_cc, y\_train\_cc = undersample.fit\_resample(X\_train, y\_train)

\end{verbatim}

\subsection{SMOTE (Synthetic Minority Over-sampling Technique)}

\begin{verbatim}

from imblearn.over\_sampling import SMOTE

oversample = SMOTE(random\_state=24)

X\_train\_smote, y\_train\_smote = oversample.fit\_resample(X\_train, y\_train)

\end{verbatim}

\subsection{KMeans SMOTE}

\begin{verbatim}

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)

\end{verbatim}

\section{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).

\subsection{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.

\textbf{Mathematical Representation:}

\[ P(y=1) = \frac{1}{1 + e^{-(\beta\_0 + \beta\_1 x\_1 + \beta\_2 x\_2 + \ldots + \beta\_n x\_n)}} \]

where \( \beta\_0, \beta\_1, \ldots, \beta\_n \) are the coefficients learned from the data.

\textbf{Hyperparameters:}

\begin{verbatim}

params\_lr = {'C': [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2]}

\end{verbatim}

\subsection{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.

\textbf{Algorithm:}

\begin{enumerate}

\item Select the best feature to split the data (e.g., using Gini impurity or information gain).

\item Create a decision node based on the selected feature.

\item Recursively split the data at each node until a stopping criterion is met (e.g., maximum depth or minimum samples per leaf).

\end{enumerate}

\textbf{Hyperparameters:}

\begin{verbatim}

params\_tree = {'max\_depth': [5, 10, 20, 30, 50]}

\end{verbatim}

\subsection{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).

\textbf{Algorithm:}

\begin{enumerate}

\item Draw multiple bootstrap samples from the training data.

\item For each sample, grow a decision tree using a random subset of features.

\item Aggregate the predictions of all trees to make the final prediction.

\end{enumerate}

\textbf{Hyperparameters:}

\begin{verbatim}

params\_rf = {'n\_estimators': [10, 50, 100, 200], 'max\_features': [None, 'sqrt']}

\end{verbatim}

\subsection{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.

\textbf{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.

\textbf{Hyperparameters:}

\begin{verbatim}

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

\end{verbatim}

\subsection{Evaluation Results}

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.

\textbf{Support Vector Machine (SVM)}:

\begin{figure}[H]

\centering

\includegraphics[width=0.8\textwidth]{svm\_results.png}

\caption{SVM Results with Different Resampling Techniques}

\end{figure}

\textbf{Logistic Regression}:

\begin{figure}[H]

\centering

\includegraphics[width=0.8\textwidth]{src/img/logistic\_regression.png}

\caption{Logistic Regression Results with Different Resampling Techniques}

\end{figure}

\textbf{Decision Tree}:

\begin{figure}[H]

\centering

\includegraphics[width=0.8\textwidth]{src/img/decision\_tree.png}

\caption{Decision Tree Results with Different Resampling Techniques}

\end{figure}

\textbf{Random Forest}:

\begin{figure}[H]

\centering

\includegraphics[width=0.8\textwidth]{src/img/random\_forest.png}

\caption{Random Forest Results with Different Resampling Techniques}

\end{figure}

\section{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.

\section{Future Work}

Future work could explore the following:

\begin{itemize}

\item Incorporating additional features such as transaction history or external credit scores to enhance the predictive power of the models.

\item Using advanced ensemble techniques or deep learning models to further improve the accuracy and robustness of predictions.

\item Exploring other oversampling and undersampling techniques to see if there are better alternatives to the ones used in this project.

\end{itemize}

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.

\end{document}