Predicting Loan Defaults: A Machine Learning Approach

Introduction

Financial institutions face significant challenges in managing loan portfolios and minimizing default risks. In this project, we leverage machine learning techniques to predict loan defaults, enabling proactive risk management and improved decision-making processes. By analyzing borrower characteristics and loan attributes, we aim to identify patterns and indicators associated with default behavior.

Dataset Overview

The dataset used for this project consists of 255,347 loan applications collected from a financial institution. It encompasses a diverse range of features, including borrower demographics, financial metrics, and loan specifics. Each record in the dataset is labeled with a binary target variable indicating whether the loan defaulted (1) or not (0), providing valuable insights for predictive modeling.

Feature Analysis

Key Features:

- **Age**: Investigating the distribution of borrower ages and its impact on default rates.
- **Income**: Analyzing the relationship between borrower income levels and loan defaults.
- **CreditScore**: Assessing the significance of credit scores in predicting default behavior.
- **LoanAmount**: Exploring the distribution of loan amounts and their influence on default probabilities.
- **EmploymentType**: Examining the role of employment status in loan default prediction.
- **LoanPurpose**: Understanding the effect of loan purposes on default likelihood.

Target Variable:

- **Default**: A binary indicator representing loan defaults (1) or non-defaults (0).

Tools :

- Integrated Development and Learning Environment : Jupyter Notebook

Python Libraries :

- pandas
- numpy
- matplotlib

- seaborn
- scikit-learn (includes multiple submodules: preprocessing, model_selection, metrics, ensemble, linear_model, decomposition, neighbors)
- Scipy
- imbalanced-learn (imblearn)
- warnings

Now let's break-down project on the basis of code :

On this dataset we perform lots of different operations and methods for analysing, preprocessing and feature engineering , in that for model selection we need some libraries that's are :

Exploratory Data Analysis (EDA):

In this section, we perform exploratory data analysis (EDA) to gain insights into the dataset and understand its structure and characteristics.

Import Libraries: We import necessary libraries such as pandas, numpy, matplotlib, and seaborn for data manipulation, numerical operations, and data visualization.

```
import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

import seaborn as sns
```

Preprocessing Utilities: We import preprocessing utilities such as LabelEncoder, OrdinalEncoder, and Shapiro-Wilk test for normality assessment.

```
from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import OrdinalEncoder

from scipy.stats import Shapiro
```

Machine Learning Libraries: We import machine learning libraries such as SMOTETomek for dealing with imbalanced datasets, RandomForestClassifier, LogisticRegression, KNeighborsClassifier for classification tasks, train_test_split for data splitting, StandardScaler for feature scaling, PCA for dimensionality reduction, and evaluation metrics such as accuracy_score, classification_report, and confusion_matrix.

```
from imblearn.combine import SMOTETomek

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.decomposition import PCA

from sklearn.neighbors import KNeighborsClassifier
```

Ignore Warnings: We import warnings module and set it to ignore all warnings to suppress unnecessary warnings during code execution.

```
import warnings

warnings.filterwarnings("ignore")
```

Read CSV File: This line of code reads the CSV file named "Loan_default.csv" into a pandas DataFrame called 'data'. The DataFrame will contain the data from the CSV file, allowing us to perform further analysis and preprocessing on the dataset.

```
data = pd.read_csv("Loan_default.csv")
```

This line of code performs multiple operations on the DataFrame 'data' in a single line, including checking for duplicated rows, calculating the total number of missing values, displaying the first few rows, providing data information, checking for missing values using an alternative method, and returning the dataset's shape.

```
data.duplicated().sum()

data.isnull().sum()

 data.head()

data.info()

 data.isna().sum()

data.shape
```

Countplot Visualization: This code generates a countplot using seaborn's **countplot** function, displaying the distribution of the target variable 'Default'. In this column we understand the target column is unbalance and we need to apply some treatment on it like resampling.

```
sns.countplot(x='Default', data=data)

plt.show()
```

- **Change Data Types**: This code modifies the data types of selected columns in the DataFrame 'data' to categorical using the 'dtype' attribute. It converts columns such as 'Education', 'EmploymentType', 'MaritalStatus', 'HasMortgage', 'HasDependents', 'LoanPurpose', and 'HasCoSigner' to the 'category' data type.
- **Drop Missing Values**: This code removes rows with missing values from the DataFrame 'data' using the 'dropna' method with the 'inplace=True' parameter. It ensures data completeness by eliminating rows containing NaN values.
- **Drop Duplicates**:.
- **Sample Data**:

```
data.dtype = {'Education': 'category', 'EmploymentType': 'category', 'MaritalStatus': 'category', 'HasMortgage': 'category', 'HasDependents': 'category', 'LoanPurpose': 'category', 'HasCoSigner': 'category'}

data.dropna(inplace=True)

data.drop_duplicates(inplace=True)

data.sample(5)
```

converting the data set into DataFrame

```
df= pd.DataFrame(data)
```

Create subplots:

- **Create Subplots**: This code creates a 3x3 grid of subplots using matplotlib's 'subplots' function, with a total figure size of 15x15 inches.
- **Boxplots for Each Variable**: This code generates boxplots for each variable in the DataFrame 'df' and assigns them to the corresponding subplot axes. Each boxplot visualizes the distribution and variability of a specific numerical variable, such as 'Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed', 'NumCreditLines', 'InterestRate', 'LoanTerm', and 'DTIRatio'.
- **Set Title**: This code sets titles for each subplot based on the variable being visualized, providing clear labeling for better understanding.
- **Adjust Layout**: This code adjusts the layout of subplots for better visualization using matplotlib's 'tight_layout' function, ensuring proper spacing and alignment between subplots.

Correlation Heatmap:

 Set Figure Size: This code sets up the figure size for the correlation heatmap using matplotlib's 'figure' function, with dimensions 10x8 inches.

 Drop Non-Numeric Columns: This code removes non-numeric columns from the DataFrame 'df' to prepare the data for correlation analysis. Non-numeric columns such as 'LoanID', 'Education', 'EmploymentType', etc., are excluded from the correlation matrix calculation.

 Calculate Correlation Matrix: This code calculates the correlation matrix for the numerical variables in the DataFrame 'numerical_data' using pandas' 'corr' method. The correlation matrix provides insights into the relationships between different numerical variables.

 Create Heatmap: This code creates a heatmap visualization of the correlation matrix using seaborn's 'heatmap' function. It annotates the heatmap with correlation values, sets the color map to 'viridis', adds linewidths and line color for better visualization, includes a color bar with a label, and specifies other visualization parameters such as square layout and tick labels.

 Set Title: This code sets a title for the heatmap visualization, labeling it as 'Correlation Heatmap' for clarity.

The function plots a boxplot showing the distribution of the variable with target column :

- **Function Definition**: This code defines a function named 'pyplot_default' that takes two arguments: 'var_name' (the name of the variable to plot) and 'df' (the DataFrame containing the data).
- **Calculate Correlation**: Inside the function, the correlation coefficient between the variable 'var_name' and the target variable 'Default' is calculated using the 'corr' method.
- **Calculate Medians**: The function calculates the median values of the variable 'var_name' for loans that were repaid (Default == 0) and not repaid (Default == 1).
- **Plot Boxplot**: The function plots a boxplot showing the distribution of the variable 'var_name' colored by the value of the target variable 'Default'. The boxplot is created using Matplotlib's 'boxplot' function.
- **Label Plot**: The plot is labeled with appropriate axes labels and a title based on the variable name.
- **Print Information**: The function prints out the correlation coefficient between the variable and the target, as well as the median values for loans that were repaid and not repaid.
- **Show Plot**: Finally, the function displays the plot using Matplotlib's 'show' function.

Label Encoding:

label_encoder = LabelEncoder()

This code applies label encoding to the specified columns using the LabelEncoder from scikit-learn. Label encoding converts categorical variables into numerical labels, which can be understood by machine learning algorithms.

- **Apply Label Encoding**: For each specified column (EmploymentType, MaritalStatus, HasCoSigner, LoanPurpose, HasDependents, HasMortgage), label encoding is applied by fitting the label encoder to the column and transforming the values.
- **Display Value Counts**: After label encoding, the code prints the value counts for each column to show the distribution of encoded labels.

Ordinal Encoder:

- This code initializes an instance of the OrdinalEncoder from scikit-learn, specifying the categories for the ordinal encoding.
- **Apply Ordinal Encoding**: The 'Education' column in the DataFrame 'df' is transformed using the ordinal encoder by fitting it to the column and transforming the values. This converts the categorical education levels ('High School', 'Bachelor's', 'Master's', 'PhD') into numerical ordinal values.
- **Convert to Integer**: After encoding, the encoded values are converted to integer type using the 'astype' method, ensuring they are stored as integers in the DataFrame.
- **Display Encoded DataFrame**: The code displays the first few rows of the DataFrame 'df' after encoding to verify the transformation.

Here , Checking the Normal Distribution for Every Numeric Column:

- **Normality Check**: This code iterates through each column in the DataFrame 'df' and performs the Shapiro-Wilk test for normality using the **shapiro** function from scipy.stats. The test statistic and p-value are printed for each column to assess the normality of the data distribution.

- **Histogram Plot**: For each column, a histogram plot with Kernel Density Estimate (KDE) is generated using seaborn's **histplot** function. This plot visualizes the distribution of the data and allows for visual inspection of normality.
- **Title and Display**: The title of each histogram plot is set to indicate the column name, and the plot is displayed using matplotlib's **show** function.

Model Selection

- **Feature Matrix and Target Vector**: here , features are stored In to x variable and the target column is stored in to y variable
- **Train-Test Split**: The data is split into training and testing sets using the 'train_test_split' function from scikit-learn. The feature matrix 'X' and target vector 'y' are split into training and testing sets, with a test size of 20% (test_size=0.2). The random_state parameter is set to 42 for reproducibility.

Standard Scaler:

This initializes an instance of the StandardScaler from scikit-learn. StandardScaler standardizes features by removing the mean and scaling to unit variance.

Here applied the standardscaler () on both training and testing data

The treatment for unbalanced data #resampling :

the SMOTETomek resampling technique to the scaled training data to address class imbalance, which is a common issue in binary classification tasks where one class is significantly underrepresented compared to the other. Resampling techniques like SMOTETomek help improve the performance of machine learning models by creating a more balanced dataset.

this technique I applied on the target colume ,where this column show the unbalance value of data “0” and “1” of prediction.

Random Forest classifier:

Random Forest classifier on the resampled training data and evaluates its performance by making predictions on the scaled testing data and calculating the accuracy score. The accuracy score indicates the model's effectiveness in correctly classifying loan defaults.

Random Forest Classifier: Initialize and train the RandomForestClassifier on resampled training data, and calculate the accuracy score on the testing data.

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.97	0.93	45170
1	0.40	0.14	0.21	5900
accuracy			0.88	51070
macro avg	0.65	0.56	0.57	51070
weighted avg	0.84	0.88	0.85	51070

Logistic Regression Classifier:

Initialize and train the LogisticRegression classifier with balanced class weights on resampled training data. Then, predict on the scaled testing data and calculate the accuracy score and confusion matrix.

Classification report:

	precision	recall	f1-score	support
0	0.94	0.68	0.79	45170
1	0.22	0.69	0.33	5900
accuracy			0.68	51070
macro avg	0.58	0.68	0.56	51070
weighted avg	0.86	0.68	0.74	51070

Perform Principal Component Analysis (PCA)

PCA Transformation: Perform Principal Component Analysis (PCA) on the scaled training and testing data. Then, train a Logistic Regression classifier on the transformed training data and evaluate its accuracy on the testing data.

Classification Report:

	precision	recall	f1-score	support
0	0.89	1.00	0.94	45170
1	0.61	0.03	0.06	5900
accuracy			0.89	51070
macro avg	0.75	0.51	0.50	51070
weighted avg	0.86	0.89	0.84	51070

K-Nearest Neighbors (KNN) Classifier:

Initialize and train the KNeighborsClassifier with 7 neighbors on resampled training data. Then, predict on the scaled testing data and calculate the accuracy score.

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.65	0.76	45170
1	0.17	0.55	0.26	5900
accuracy			0.64	51070
macro avg	0.54	0.60	0.51	51070
weighted avg	0.83	0.64	0.70	51070

K-Nearest Neighbors (KNN) Classifier:

Initialize and train the KNeighborsClassifier with 5 neighbors on PCA-transformed training data. Then, predict on the PCA-transformed testing data and calculate the accuracy score.

Accuracy: 0.8749755237908753

Confusion Matrix:

[[44282 888]

 [5497 403]]