1. **Exploratory Data Analysis (EDA)**

Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from wordcloud import WordCloud

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from xgboost import XGBClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

**Introduction**

This project analyzes a credit card customer dataset to predict the likelihood of a customer defaulting on their next month's payment. The dataset contains information on customers' demographics, credit history, and financial behavior, such as:

* **Demographics**: Gender, education level, marital status, and age.
* **Credit and Payment Details**: Credit limit, past payment behavior, and historical bill amounts.
* **Payment Status**: Whether the customer defaulted on their next payment (default.payment.next.month).

The objective of this project is to build a predictive model to classify customers into two categories: default or non-default, enabling financial institutions to mitigate credit risk effectively. The insights derived can help optimize credit policies, reduce losses, and improve customer relationship management.

**Loading Dataset**

data\_credit\_card = pd.read\_csv("C:/Users/user/Desktop/credit.csv")

data\_credit\_card.head(10) # Displaying first 10 records

**Dataset Overview**

print(data\_credit\_card.info())

print(data\_credit\_card.describe())

print(data\_credit\_card['default.payment.next.month'].value\_counts())

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ID 30000 non-null int64

1 LIMIT\_BAL 30000 non-null float64

2 SEX 30000 non-null int64

3 EDUCATION 30000 non-null int64

4 MARRIAGE 30000 non-null int64

5 AGE 30000 non-null int64

6 PAY\_0 30000 non-null int64

7 PAY\_2 30000 non-null int64

8 PAY\_3 30000 non-null int64

9 PAY\_4 30000 non-null int64

10 PAY\_5 30000 non-null int64

11 PAY\_6 30000 non-null int64

12 BILL\_AMT1 30000 non-null float64

13 BILL\_AMT2 30000 non-null float64

14 BILL\_AMT3 30000 non-null float64

15 BILL\_AMT4 30000 non-null float64

16 BILL\_AMT5 30000 non-null float64

17 BILL\_AMT6 30000 non-null float64

18 PAY\_AMT1 30000 non-null float64

19 PAY\_AMT2 30000 non-null float64

20 PAY\_AMT3 30000 non-null float64

21 PAY\_AMT4 30000 non-null float64

22 PAY\_AMT5 30000 non-null float64

23 PAY\_AMT6 30000 non-null float64

24 default.payment.next.month 30000 non-null int64

**Data Visualizations**

**Histograms**: To understand the distribution of LIMIT\_BAL (credit limit) and AGE.

**Scatter Plots**: To explore the relationship between important features (LIMIT\_BAL, PAY\_AMT1, and AGE) and the target variable (default.payment.next.month).

**Box Plots**: To convey the comparison of payment behavior and bill amounts across the two target categories (default or non-default).

# Calculate the correlation matrix

correlation\_matrix = data\_credit\_card.corr()

# Create a heatmap

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True, square=True)

plt.title("Correlation Heatmap", fontsize=16)

plt.xticks(rotation=45, ha='right', fontsize=10)

plt.yticks(fontsize=10)

plt.show()

**Data Preprocessing**

In this section, we shall consider the following:

* **Missing Values:** How they were handled.
* **Encoding:** Convert categorical variables to numeric.
* **Feature Engineering:** Add new features or remove redundant ones.
* **Scaling:** Apply normalization/standardization.

[31]:

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

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

# Handle Missing Values

data\_credit\_card.fillna(0, inplace=True) # Replace NaNs with 0 (modify as needed)

# Separate Features and Target

X = data\_credit\_card.drop('default.payment.next.month', axis=1)

y = data\_credit\_card['default.payment.next.month']

# Encode Categorical Variables

categorical\_features = ['SEX', 'EDUCATION', 'MARRIAGE']

numerical\_features = [col for col in X.columns if col not in categorical\_features]

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), numerical\_features),

('cat', OneHotEncoder(), categorical\_features)

]

)

# Encode Categorical Variables

categorical\_features = ['SEX', 'EDUCATION', 'MARRIAGE']

numerical\_features = [col for col in X.columns if col not in categorical\_features]

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), numerical\_features),

('cat', OneHotEncoder(), categorical\_features)

]

)

# Split Data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42)

# Feature Preprocessing

X\_train = preprocessor.fit\_transform(X\_train)

X\_test = preprocessor.transform(X\_test)

**Model Building and Evaluation**

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import GridSearchCV, cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve

import joblib

**Baseline Model**

**# Logistic Regression**

**logistic\_model = LogisticRegression(max\_iter=500)**

**logistic\_model.fit(X\_train, y\_train)**

**Advanced Model**

# Random Forest

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_train, y\_train)

XGBoost

xgb\_model = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

xgb\_model.fit(X\_train, y\_train)

**Hyperparameter Tuning**

# Logistic Regression Hyperparameter Tuning using GridSearchCV

logistic\_params = {'C': [0.01, 0.1, 1, 10], 'penalty': ['l2']}

logistic\_grid = GridSearchCV(LogisticRegression(max\_iter=1000), logistic\_params, cv=5, scoring='roc\_auc', n\_jobs=-1)

logistic\_grid.fit(X\_train, y\_train)

# Display the best parameters

print(f"Best Parameters: {logistic\_grid.best\_params\_}")

# Display the best score

print(f"Best ROC-AUC Score: {logistic\_grid.best\_score\_:.4f}")

# Logistic Regression Hyperparameter Tuning using RandomizedSearchCV

# Standardize the data

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Define hyperparameter search space

log\_reg\_params = [

{'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2'], 'solver': ['liblinear']}, # For liblinear

{'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2'], 'solver': ['saga']}, # For saga without elasticnet

{'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['elasticnet'], 'solver': ['saga'], 'l1\_ratio': [0.1, 0.5, 0.9]}, # For saga with elasticnet

]

# Use RandomizedSearchCV for faster tuning

log\_reg\_random = RandomizedSearchCV(

estimator=LogisticRegression(random\_state=42, max\_iter=5000, tol=1e-4), # Increased max\_iter and reduced tol

param\_distributions=log\_reg\_params,

n\_iter=15, # Number of random combinations to try

cv=3, # Cross-validation folds

scoring='roc\_auc',

random\_state=42,

n\_jobs=-1,

error\_score='raise' # Debug any remaining errors

)

# Fit the model

log\_reg\_random.fit(X\_train\_scaled, y\_train)

# Display best parameters and best score

print(f"Best Parameters: {log\_reg\_random.best\_params\_}")

print(f"Best ROC-AUC Score: {log\_reg\_random.best\_score\_:.4f}")

# RandomForest Hyperparameter Tuning using RandomizedSearchCV

# Define hyperparameter search space

rf\_params = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

# Use RandomizedSearchCV for faster tuning

rf\_random = RandomizedSearchCV(

estimator=RandomForestClassifier(random\_state=42),

param\_distributions=rf\_params,

n\_iter=20, # Number of random combinations to try

cv=3, # Reduce folds to speed up tuning

scoring='roc\_auc',

random\_state=42,

n\_jobs=-1

)

# Fit the model

rf\_random.fit(X\_train, y\_train)

# Display best parameters and best score

print(f"Best Parameters: {rf\_random.best\_params\_}")

print(f"Best ROC-AUC Score: {rf\_random.best\_score\_:.4f}")

# xbg Hyperparameter Tuning using RandomizedSearchCV

# Define hyperparameter search space

xgb\_params = {

'n\_estimators': [50, 100, 200, 300],

'learning\_rate': [0.01, 0.05, 0.1, 0.2],

'max\_depth': [3, 5, 7, 10],

'subsample': [0.6, 0.8, 1.0],

'colsample\_bytree': [0.6, 0.8, 1.0],

'gamma': [0, 1, 5],

'min\_child\_weight': [1, 3, 5]

}

# Initialize the XGBoost model

xgb\_model = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42)

# Use RandomizedSearchCV for faster hyperparameter tuning

xgb\_random = RandomizedSearchCV(

estimator=xgb\_model,

param\_distributions=xgb\_params,

n\_iter=20, # Number of random combinations to try

scoring='roc\_auc', # Evaluation metric

cv=5, # Cross-validation folds

n\_jobs=-1, # Use all available cores

random\_state=42,

verbose=2 # Optional: Shows progress during tuning

)

# Fit the model to the training data

xgb\_random.fit(X\_train, y\_train)

# Display the best parameters and best score

print(f"Best Parameters: {xgb\_random.best\_params\_}")

print(f"Best ROC-AUC Score: {xgb\_random.best\_score\_:.4f}")

**Cross-Validation**

# Evaluate with Cross-Validation

def evaluate\_with\_cross\_validation(model, X\_train, y\_train):

scores = cross\_val\_score(model, X\_train, y\_train, cv=5, scoring='roc\_auc')

print(f"Cross-Validation AUC-ROC Scores: {scores}")

print(f"Mean AUC-ROC Score: {scores.mean():.4f}\n")

# Cross-validate each tuned model

print("Logistic Regression Cross-Validation:")

evaluate\_with\_cross\_validation(logistic\_grid.best\_estimator\_, X\_train, y\_train)

print("Random Forest Cross-Validation:")

evaluate\_with\_cross\_validation(rf\_grid.best\_estimator\_, X\_train, y\_train)

print("XGBoost Cross-Validation:")

evaluate\_with\_cross\_validation(xgb\_grid.best\_estimator\_, X\_train, y\_train)

# Plot Confusion Matrix for Best Models on Test Set

for model, name in zip(

[logistic\_grid.best\_estimator\_, rf\_grid.best\_estimator\_, xgb\_grid.best\_estimator\_],

['Logistic Regression (Tuned)', 'Random Forest (Tuned)', 'XGBoost (Tuned)']

):

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

# Print classification metrics

print(f"{name} Classification Report:")

print(classification\_report(y\_test, y\_pred))

print(f"AUC-ROC Score: {roc\_auc\_score(y\_test, y\_proba):.4f}\n")

# Generate confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Plot confusion matrix

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Fraud', 'Fraud'], yticklabels=['Not Fraud', 'Fraud'])

plt.title(f"Confusion Matrix: {name}")

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.show()

# Save Confusion Matrices to CSV Files

for model, name in zip(

[logistic\_grid.best\_estimator\_, rf\_grid.best\_estimator\_, xgb\_grid.best\_estimator\_],

['logistic', 'rf', 'xgb']

):

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm\_df = pd.DataFrame(cm, index=["Not Fraud", "Fraud"], columns=["Not Fraud", "Fraud"])

cm\_df.to\_csv(f"confusion\_matrix\_{name}.csv", index=False)

print("Confusion matrices saved as CSV files!")

import pandas as pd

from sklearn.metrics import classification\_report, roc\_auc\_score

# Initialize an empty list to store metrics

metrics = []

# Evaluate each model and collect metrics

for model, name in zip(

[logistic\_grid.best\_estimator\_, rf\_grid.best\_estimator\_, xgb\_grid.best\_estimator\_],

['Logistic Regression (Tuned)', 'Random Forest (Tuned)', 'XGBoost (Tuned)']

):

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

# Calculate metrics

report = classification\_report(y\_test, y\_pred, output\_dict=True)

auc\_roc = roc\_auc\_score(y\_test, y\_proba)

# Append metrics to the list

metrics.append({

"Model": name,

"Accuracy": report['accuracy'],

"Precision": report['1']['precision'], # For the positive class (fraud)

"Recall": report['1']['recall'], # For the positive class (fraud)

"F1-Score": report['1']['f1-score'], # For the positive class (fraud)

"ROC-AUC": auc\_roc

})

# Convert metrics to a DataFrame

metrics\_df = pd.DataFrame(metrics)

# Save the metrics to a CSV file

metrics\_df.to\_csv("model\_metrics.csv", index=False)

print("Model metrics saved as model\_metrics.csv!")

**BUILDING INTERACTIVE DASHBOARD**

import pandas as pd

from sklearn.metrics import classification\_report, roc\_auc\_score

# Initialize an empty list to store metrics

metrics = []

# Evaluate each model and collect metrics

for model, name in zip(

[logistic\_grid.best\_estimator\_, rf\_grid.best\_estimator\_, xgb\_grid.best\_estimator\_],

['Logistic Regression (Tuned)', 'Random Forest (Tuned)', 'XGBoost (Tuned)']

):

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

# Calculate metrics

report = classification\_report(y\_test, y\_pred, output\_dict=True)

auc\_roc = roc\_auc\_score(y\_test, y\_proba)

# Append metrics to the list

metrics.append({

"Model": name,

"Accuracy": report['accuracy'],

"Precision": report['1']['precision'], # For the positive class (fraud)

"Recall": report['1']['recall'], # For the positive class (fraud)

"F1-Score": report['1']['f1-score'], # For the positive class (fraud)

"ROC-AUC": auc\_roc

})

# Convert metrics to a DataFrame

metrics\_df = pd.DataFrame(metrics)

# Save the metrics to a CSV file

metrics\_df.to\_csv("model\_metrics.csv", index=False)

print("Model metrics saved as model\_metrics.csv!")

**Conclusion**

The objective of predicting credit card payment defaults was analyzed using Logistic Regression, Random Forest, and XGBoost models. Below are the key takeaways:

1. **Logistic Regression:**
   * Strengths: High recall for non-defaults (0.97) and reasonable AUC-ROC score (0.7103).
   * Weaknesses: Poor recall for defaults (0.24), missing many defaulting customers.
2. **Random Forest:**
   * Strengths: Improved recall for defaults (0.35) and a better AUC-ROC score (0.7750).
   * Weaknesses: Still struggles with default recall and slightly more false positives.
3. **XGBoost:**
   * Strengths: Best AUC-ROC score (0.7795) and improved precision for defaults (0.61).
   * Weaknesses: Recall for defaults remains low (0.37), slightly higher than Random Forest.

**Key Insight:** All models effectively identify non-defaulting customers but struggle with recall for defaults, which is critical for mitigating credit risk. XGBoost shows the best overall performance but requires further optimization to improve default classification.

**Recommendations:** Enhance feature engineering, balance the dataset, and adjust classification thresholds to improve recall for defaults while maintaining overall accuracy.

**Credit Card Default Prediction**

**Overview**

This project aims to predict credit card payment defaults using machine learning models. The dataset includes customer payment history and financial attributes to assess the likelihood of default.

**Features**

* **Data Preprocessing**: Handling missing values, feature scaling, and encoding categorical variables.
* **Exploratory Data Analysis (EDA)**: Understanding data distribution and feature correlations.
* **Model Training**: Logistic Regression, Random Forest, and XGBoost.
* **Model Evaluation**: Performance assessment using precision, recall, and AUC-ROC score.
* **Model Deployment**: Saving the best model for future inference.

**Installation**

1. Clone the repository:

git clone https://github.com/your-repo/credit-card-default.git

1. Navigate to the project directory:

cd credit-card-default

1. Create a virtual environment and activate it:
2. python -m venv venv

source venv/bin/activate # On Windows: venv\\Scripts\\activate

1. Install dependencies:

pip install -r requirements.txt

**Dataset**

The dataset should be placed in the data/ directory. Ensure it contains customer transaction history and relevant financial indicators.

**Model Training**

Run the following script to train the models:

python train.py

This script performs data preprocessing, trains Logistic Regression, Random Forest, and XGBoost models, and evaluates their performance.

**Model Evaluation**

Performance metrics:

* **Logistic Regression**:
  + High recall for non-defaults (0.97)
  + AUC-ROC score: 0.7103
  + Weak recall for defaults (0.24)
* **Random Forest**:
  + Improved recall for defaults (0.35)
  + AUC-ROC score: 0.7750
* **XGBoost**:
  + Best AUC-ROC score: 0.7795
  + Improved precision for defaults (0.61)
  + Recall for defaults: 0.37

**Conclusion**

The objective of predicting credit card payment defaults was analyzed using Logistic Regression, Random Forest, and XGBoost models. Below are the key takeaways:

* **Logistic Regression**: High recall for non-defaults but poor recall for defaults.
* **Random Forest**: Better recall for defaults but more false positives.
* **XGBoost**: Best overall performance but still struggles with default recall.

**Key Insight**: While all models effectively identify non-defaulting customers, they struggle with recall for defaults, which is critical for mitigating credit risk. XGBoost shows the best overall performance but requires further optimization.

**Recommendations**:

* Enhance feature engineering
* Balance the dataset
* Adjust classification thresholds to improve recall for defaults while maintaining overall accuracy

**HOW TO UPLOAD IN GITHUB**

To upload this project to GitHub with proper formatting, follow these steps:

**1. Create a New GitHub Repository**

1. Go to [GitHub](https://github.com/) and log in.
2. Click the **"+"** icon in the top right and select **New repository**.
3. Enter a repository name like **credit-card-default**.
4. Add a description (optional).
5. Choose **Public** or **Private**.
6. **Do NOT** initialize with a README (we will add it manually).
7. Click **Create repository**.

**2. Set Up the Local Repository**

Run the following commands in your terminal:

bash

CopyEdit

# Navigate to the directory where you want to store the project

cd path/to/your/project

# Initialize Git

git init

# Add the remote GitHub repository (Replace "your-username" and "your-repo" with actual values)

git remote add origin https://github.com/your-username/credit-card-default.git

**3. Add Your Files**

Make sure your project directory contains:

* train.py (for training the models)
* requirements.txt (list of dependencies)
* data/ folder (containing the dataset)
* A properly formatted README.md file

**4. Create and Format README.md**

Save your README content into a README.md file with Markdown formatting:

markdown

CopyEdit

# Credit Card Default Prediction

## Overview

This project aims to predict credit card payment defaults using machine learning models. The dataset includes customer payment history and financial attributes to assess the likelihood of default.

## Features

- \*\*Data Preprocessing:\*\* Handling missing values, feature scaling, and encoding categorical variables.

- \*\*Exploratory Data Analysis (EDA):\*\* Understanding data distribution and feature correlations.

- \*\*Model Training:\*\* Logistic Regression, Random Forest, and XGBoost.

- \*\*Model Evaluation:\*\* Performance assessment using precision, recall, and AUC-ROC score.

- \*\*Model Deployment:\*\* Saving the best model for future inference.

## Installation

1. Clone the repository:

```bash

git clone https://github.com/your-username/credit-card-default.git

1. Navigate to the project directory:

bash

CopyEdit

cd credit-card-default

1. Create a virtual environment and activate it:

bash

CopyEdit

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

1. Install dependencies:

bash

CopyEdit

pip install -r requirements.txt

**Dataset**

The dataset should be placed in the data/ directory. Ensure it contains customer transaction history and relevant financial indicators.

**Model Training**

Run the following script to train the models:

bash

CopyEdit

python train.py

This script performs data preprocessing, trains Logistic Regression, Random Forest, and XGBoost models, and evaluates their performance.

**Model Evaluation**

Performance metrics:

**Logistic Regression:**

* High recall for non-defaults (0.97)
* AUC-ROC score: 0.7103
* Weak recall for defaults (0.24)

**Random Forest:**

* Improved recall for defaults (0.35)
* AUC-ROC score: 0.7750

**XGBoost:**

* Best AUC-ROC score: 0.7795
* Improved precision for defaults (0.61)
* Recall for defaults: 0.37

**Conclusion**

The objective of predicting credit card payment defaults was analyzed using Logistic Regression, Random Forest, and XGBoost models. Below are the key takeaways:

* **Logistic Regression:** High recall for non-defaults but poor recall for defaults.
* **Random Forest:** Better recall for defaults but more false positives.
* **XGBoost:** Best overall performance but still struggles with default recall.

**Key Insight:**

While all models effectively identify non-defaulting customers, they struggle with recall for defaults, which is critical for mitigating credit risk. XGBoost shows the best overall performance but requires further optimization.

**Recommendations:**

* Enhance feature engineering
* Balance the dataset
* Adjust classification thresholds to improve recall for defaults while maintaining overall accuracy

sql

CopyEdit