AAI500 - Applied Artificial Intelligence

San Diego University

Customer Credit Default Prediction

Submitted to: Instructor Azka Azka

Submitted by:

Anugrah Rastogi

Mallesham D

Dhrub Satyam

Table of Content

Abstract

This document presents a data-driven approach to identifying customers who may fail to repay their credit obligations. The study uses structured financial records that include behavioral and personal characteristics to build and evaluate prediction models.

After preparing the data-cleaning missing values, converting categories, and scaling numeric fields, we explored it for patterns and distributions. Then, we trained and tested four different classification approaches (Logistic Regression, Decision Tree, Random Forest, and XGBoost).

We compared these models using various metrics, including ROC AUC, F1 Score, Precision, and Recall, to judge their accuracy in flagging defaults. Out of all the models tested, the Random Forest algorithm demonstrated the most dependable outcomes, striking an effective balance between prediction accuracy and model interpretability.

Based on the findings, we suggest that Random Forest be considered for real-world deployment, along with ongoing evaluation to ensure fair and ethical decision-making.

1. Introduction

Figuring out if a customer might miss a credit payment is a real-world challenge that banks and financial companies face every day. It helps them reduce risk and make smarter lending decisions. The ability to identify risky borrowers before issuing credit can significantly reduce financial losses, streamline approval workflows, and enhance regulatory compliance.

This project, undertaken as part of the AAI500 course at the University of San Diego, leverages machine learning techniques on a real dataset containing both behavioral and demographic factors. This analysis is based on a public dataset available from an online platform (Kaggle, 2023).

This project primarily aimed to create dependable models for predicting potential credit defaulters, assess the effectiveness of different machine learning techniques in terms of accuracy and practicality, and generate meaningful insights to improve credit evaluation processes.

2. Data Handling and Preparation Techniques

To ensure consistency in the dataset (Kaggle, 2023), missing entries in numeric fields were filled using the median of each column. We chose the median instead of the mean because it is less sensitive to unusual or extreme values. This method helped maintain the integrity of the data set and reduced the risk of bias introduced by outliers.

Dataset Overview:

- Training Set: 70,000 records with 82 features
- Test Set: 33,000 records with 80 features
- Target Column: The target column indicates whether a customer has defaulted (1) or not (0).

Cleaning and Preprocessing Steps:

- Managing Missing Data: For numerical columns with missing entries, our approach employed the median value to fill the gaps. The median is preferred over the mean because it is robust to outliers, ensuring that extreme values do not skew the data.
    - Example: If the income column had missing entries, they were filled with the median income value of the entire column.
- Categorical Encoding: To handle categorical data, label encoding was applied, transforming string or Boolean values into numeric form. This encoding process was carried out uniformly on both the training and test datasets. To ensure consistency and avoid unseen categories, encoders were trained using the combined set of unique values from both datasets.
    - Example: Gender: {'Male' → 0, 'Female' → 1}
- Feature Scaling: We used RobustScaler to scale numerical features. Unlike standard scaling methods, RobustScaler uses the interquartile range (IQR), which makes it highly effective in the presence of outliers and models are not overly influenced by extreme values.
- Feature Alignment: To prevent inconsistencies during prediction, we made sure the training and test datasets had the same columns. This included, Dropping extra columns if present in either set and Ensuring consistent column order.
- Target Isolation: The target column was designated as the output label, where a value of 0 represents non-defaulting customers and 1 indicates default cases.

3. Data Exploration and Insights

Exploratory Data Analysis (EDA) helps us get a clearer picture of the data we are working with. It involves spotting irregularities, exploring feature behavior, and forming early hypotheses. In this project, EDA allowed us to examine how different variables interact with the outcome (credit default), which was essential in selecting meaningful features for modeling.

Univariate Analysis

We began by examining distributions of key features to understand their spread, outliers, and potential impact.

- **Numerical Features:** age, monthly_income, debt_ratio, number_of_open_credit_lines, etc. Some variables showed right-skewed distributions, especially financial ones like debt_ratio and monthly_income.
- **Categorical Features:** Categorical features with Boolean or object data types were transformed using label encoding to convert them into numerical form. Certain categorical features had imbalanced value counts, which might influence the model bias if not handled properly.
- **Example Chart:** It revealed a long tail indicating a small number of customers with very high debt ratios.

Bivariate Analysis

We analyzed features related to the target variable.

- **Boxplots:** Here, debt_ratio and number_of_dependents were higher on average among defaulters.
- **Correlation Matrix:** Heatmaps helped spot weak to moderate correlations. Most features showed low linear correlation, suggesting nonlinear models (e.g., trees) might perform better.
- **Default Distribution:** Only ~20.6% of records were labeled as defaulters. This class imbalance required attention during model training (e.g., via class_weight='balanced' or resampling techniques).

Key Visual Insights

- **Target Imbalance:** Defaulted (1): ~20% and Non-defaulted (0): ~80%. This significant imbalance influenced our choice of metrics (F1-score, ROC AUC instead of accuracy).
- **Outliers:** Features like monthly_income had extreme values. Instead of dropping, we used RobustScaler to mitigate their influence.

- **Missing Data:** We found missing data in income and dependent count fields. Imputed using median values, preserving the distribution shape.

Summary of EDA Outcomes

Insight	Action Taken
Class imbalance in target	Used class_weight='balanced' and chose suitable metrics
Skewed financial data	Applied RobustScaler
Categorical encoding needed	Used LabelEncoder
Correlations were weak	Preferred ensemble models like Random Forest/XGBoost
Missing values in key columns	Used median imputation

4. Model Selection

Choosing the right model is essential when dealing with real-world data—especially one that involves imbalanced classes like credit default prediction. Our selection process involved testing multiple algorithms with varying complexity and interpretability.

Candidate Models

We explored 4 supervised classification algorithms, each has its unique advantages.

Model	Rationale
Logistic Regression	Simple, interpretable, works well as a baseline.
Decision Tree	It is easy to visualize and explain decisions.
Random Forest	It handles non-linearities, robust to noise, good for tabular data.
XGBoost	State-of-the-art gradient boosting algorithm with excellent accuracy and performance.

Hyperparameter Tuning

All models were optimized for ROC AUC through GridSearchCV, applying a 5-fold cross-validation strategy.

Logistic Regression:

- Logistic regression is a foundational classification method, particularly effective when the relationship between features and outcome is approximately linear (Hosmer, Lemeshow, & Sturdivant, 2013).
- Grid: C values and solvers
- class_weight='balanced' was used to address the target imbalance

Decision Tree:

- Decision trees provide an intuitive structure for modeling decisions and are widely used due to their interpretability and low computational cost (Quinlan, 1986).
- Grid: max_depth, min_samples_split, min_samples_leaf

Random Forest:

- Random Forest enhances decision tree performance by using ensemble learning to reduce overfitting and improve accuracy (Breiman, 2001).
- Grid: n_estimators, max_depth, min_samples_split

XGBoost:

- XGBoost, a scalable gradient boosting framework, is known for its regularization and efficiency in handling sparse data (Chen & Guestrin, 2016).
- Grid: n_estimators, max_depth, learning_rate
- Used eval_metric='logloss'

Tools Used:

- GridSearchCV for tuning (Pedregosa et al., 2011a).
- RobustScaler to scale numerical features (Pedregosa et al., 2011b).
- LabelEncoder for categorical encoding (Pedregosa et al., 2011c).

Evaluation Strategy

To evaluate models fairly, we split the training data:

- **80% for training, 20% for validation**
- Performance was measured using:
  - **ROC AUC Score** (discrimination ability)
  - **F1-Score** (balance between precision and recall)
  - **Precision & Recall** (for defaulters)

Validation Results:

Model	ROC AUC	F1 Score	Precision	Recall
Decision Tree	0.8808	0.6855	0.75	0.63
Logistic Regression	0.9057	0.7134	0.63	0.82
XGBoost	0.9576	0.7906	0.82	0.76
Random Forest	0.9633	0.8049	0.85	0.76

Best Model Selected

After comparing all the results, Random Forest model is selected as the final model due to highest ROC AUC, strong F1-score, and balanced precision-recall trade-off.

5. Model Analysis

After training and tuning, we analyzed each model's performance on the validation set to assess its real-world applicability. The focus was on evaluating how well each model could identify defaulters (class 1) without generating too many false positives.

Classification Reports

To understand how well each algorithm identified defaulters, we assessed their precision, recall, and F1-score for placing particular emphasis on how accurately they handled class 1, which represents default instances.

Model	Precision (class 1)	Recall (class 1)	F1 Score (class 1)	ROC AUC
Logistic Regression	0.63	0.82	0.7134	0.9057
Decision Tree	0.75	0.63	0.6855	0.8808
Random Forest	0.85	0.76	0.8049	0.9633
XGBoost	0.82	0.76	0.7906	0.9576

Understanding the Evaluation Metrics:

- **Precision** tells us how accurate the model's default predictions were — in other words, Out of the predicted defaulters, how many were actually correct?.

- **Recall** measures how effectively the model detects customers who actually defaulted.

- **F1 Score** is a balanced score that considers both precision and recall, helping us understand overall model effectiveness for the default class.

- **ROC AUC** measures the model's ability to correctly separate defaulters from non-defaulters across all thresholds — the higher, the better.

Random Forest led with the highest F1 and ROC AUC, making it best suited for production.

Visual Insights:

We plotted key evaluation curves to support our findings:

1. ROC Curve: The ROC curve shows that the Random Forest model achieves the highest Area Under the Curve (AUC), indicating its superior classification performance.
2. Precision-Recall Curve: Confirms the effectiveness in handling class imbalance.
3. Model Comparison Bar Chart

Model	**ROC AUC**	**F1 Score**
Random Forest	0.9633	0.8049
XGBoost	0.9576	0.7906
Logistic Regression	0.9057	0.7134
Decision Tree	0.8808	0.6855

Summary of Observations

- Random Forest performed best overall, especially in identifying defaults accurately.
- XGBoost was a close second and could be explored further for optimization.
- Logistic Regression, while simple, showed surprisingly good recall but low precision—making it risky for production.
- The Decision Tree was interpretable but too simplistic for this complex dataset.

Why Choose Random Forest?

- We have achieved the highest ROC AUC and F1-score among all tested models.
- It demonstrates consistent performance across both defaulter and non-defaulter classes.

- It effectively handles noisy data and is resilient to outliers.
- It is well-suited for datasets containing a combination of numerical and categorical features, making it ideal for financial applications.

6. Conclusion and Recommendations

Project Recap:

This project focused on developing a machine learning approach to identify the likelihood of customer credit default using structured, real-world data. Through systematic preprocessing, model building, and evaluation, we were able to explore multiple classification models and identify the most effective solution.

Key Findings:

The Random Forest algorithm stood out as the most dependable model, demonstrating strong predictive accuracy and a good balance between correctly identifying defaults and avoiding false alarms, outperforming others with the following.

- Highest ROC AUC (0.9633)
- Strong F1 Score (0.8049)
- Balanced precision (0.85) & recall (0.76)

XGBoost showed competitive results, slightly behind Random Forest, and can be considered for further tuning or ensemble techniques. Logistic Regression, though basic, demonstrated strong recall (0.82), meaning it rarely missed defaulters, but suffered from low precision (0.63), potentially flagging too many false positives. Decision Tree underperformed, indicating a need for deeper tree-based models or ensemble methods for such complex data.

Business Implications:

A robust credit default prediction model offers several advantages to banks and financial institutions. It helps minimize financial losses by identifying high-risk customers early. It improves customer experience by speeding up approvals for low-risk applicants. Such models also support regulatory compliance by being explainable and auditable. Additionally, they help reduce operational costs by automating parts of the decision-making process.

Recommendations

To further improve model performance and business adoption:

- Feature Engineering: Create features like trend indicators, rolling averages, or recent transaction history. Explore time-window aggregation for behavior-based features.
- Model Optimization: Perform more extensive hyperparameter tuning (e.g., using randomized search). Consider implementing ensemble techniques such as stacking or majority voting by combining the strengths of high-performing models like Random Forest and XGBoost. This hybrid approach may enhance prediction stability and improve overall model performance.
- Bias & Fairness Audits: It is essential to examine whether the model's decisions are consistent across various demographic groups, such as different age brackets, genders, or income levels. This includes checking if certain groups are unfairly favored or penalized. Conducting fairness assessments can help confirm that the model behaves responsibly and does not introduce unintentional bias in credit evaluations.
- Model Monitoring: Set up a feedback loop to monitor predictions and update the model regularly. Use A/B testing to evaluate model performance against existing systems.
- Interpretability: We may utilize techniques such as SHAP (SHapley Additive exPlanations) or LIME to enhance the explainability of the model's predictions. These tools help in understanding how individual features contribute to the prediction outcomes. This level of transparency is especially important when the results are used in customer-facing applications, as it provides clear justifications for each decision.

Final Recommendation

Based on current evaluation metrics, Random Forest is recommended for deployment as the primary model. However, continual monitoring, feedback integration, and further experimentation with advanced techniques will help refine predictions in real-world environments.

Appendix (Code and Output)

Appendix A.1 Load Data

Code:

```
#Importing Libraries

import pandas as pd

import numpy as np

import os

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import RobustScaler, LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report, roc_auc_score, f1_score

from xgboost import XGBClassifier

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')


# Check current working directory and file existence

print(f"Current working directory: {os.getcwd()}")

train_file = "processed_train_data.csv"

test_file = "processed_test_data.csv"
```

```
if not os.path.exists(train_file) or not os.path.exists(test_file):
        raise FileNotFoundError(f"One or both files not found: {train_file},
{test_file}")


# Load datasets
train_df = pd.read_csv(train_file)
test_df = pd.read_csv(test_file)
print(f"Train data shape: {train_df.shape}")
print(f"Test data shape: {test_df.shape}")
```

Appendix A.2 Preprocess Data (Data Cleaning)

Code:

```
# Define target column
target_col = 'target'
if target_col not in train_df.columns:
        raise ValueError(f"Target column '{target_col}' not found in training data")


# Separate features and target
X_train_full = train_df.drop(columns=[target_col])
y_train_full = train_df[target_col]
X_test_final = test_df.drop(columns=[target_col], errors='ignore')


# Align train and test columns
common_cols = X_train_full.columns.intersection(X_test_final.columns)
X_train_full = X_train_full[common_cols]
```

```
X_test_final = X_test_final[common_cols]

print(f"Aligned columns: {len(common_cols)}")


# Handle missing values

numeric_cols = X_train_full.select_dtypes(include=[np.number]).columns

for col in numeric_cols:

        median_val = X_train_full[col].median()

        X_train_full[col].fillna(median_val, inplace=True)

        X_test_final[col].fillna(median_val, inplace=True)


# Encode categorical columns

label_encoders = {}

categorical_cols = X_train_full.select_dtypes(include=['object', 'bool']).columns

for col in categorical_cols:

        le = LabelEncoder()

        # Fit on combined train and test unique values to avoid unseen labels

        unique_vals = pd.concat([X_train_full[col], X_test_final[col]],
axis=0).astype(str).unique()

        le.fit(unique_vals)

        X_train_full[col] = le.transform(X_train_full[col].astype(str))

        X_test_final[col] = le.transform(X_test_final[col].astype(str))

        label_encoders[col] = le


# Split training data into train and validation sets

X_train, X_val, y_train, y_val = train_test_split(

        X_train_full, y_train_full, test_size=0.2, random_state=42,
stratify=y_train_full
)

# Scale features

scaler = RobustScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_val_scaled = scaler.transform(X_val)

X_test_scaled = scaler.transform(X_test_final)

print(f"Training set: {X_train.shape}, Validation set: {X_val.shape}, Test set:
{X_test_final.shape}")
```

Appendix A.3 Train Models with Hyperparameter Tuning

Code:

```
# Initialize models and parameter grids

models = {

        'RandomForest': {

        'model': RandomForestClassifier(random_state=42, n_jobs=-1),

        'param_grid': {

        'n_estimators': [50, 100],

        'max_depth': [10, 20, None],

        'min_samples_split': [2, 5]

        }

        },

        'DecisionTree': {

        'model': DecisionTreeClassifier(random_state=42),

        'param_grid': {
```

```
        'max_depth': [5, 10, 15],
        'min_samples_split': [2, 5],
        'min_samples_leaf': [1, 2]
        }
        },
        'XGBoost': {
        'model': XGBClassifier(random_state=42, n_jobs=-1, eval_metric='logloss'),
        'param_grid': {
        'n_estimators': [50, 100],
        'max_depth': [3, 6],
        'learning_rate': [0.1, 0.01]
        }
        },
        'LogisticRegression': {
        'model': LogisticRegression(random_state=42, max_iter=10000,
class_weight='balanced'),
        'param_grid': {
        'C': [0.1, 1.0, 10.0],
        'solver': ['lbfgs', 'liblinear']
        }
        }
}

# Train and tune models
best_models = {}
for name, config in models.items():
```

```
print(f"\nTuning {name}...")

grid_search = GridSearchCV(

config['model'], config['param_grid'], cv=5, scoring='roc_auc', n_jobs=-1

)

grid_search.fit(X_train_scaled, y_train)

best_models[name] = grid_search.best_estimator_

print(f"Best parameters: {grid_search.best_params_}")

print(f"Best cross-validation ROC-AUC: {grid_search.best_score_:.4f}")
```

This code block carries out the following steps:

- It divides the dataset into training and validation subsets.
- It performs hyperparameter tuning using grid search over parameters like n_estimators, max_depth, and min_samples_split.
- It selects the best-performing model based on ROC AUC and F1 score metrics.
- It displays a detailed classification report for performance evaluation.

Output:

Tuning RandomForest

- Best parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
- Best cross-validation ROC-AUC: 0.9559

Tuning DecisionTree

- Best parameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}
- Best cross-validation ROC-AUC: 0.8708

Tuning XGBoost

- Best parameters: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 100}
- Best cross-validation ROC-AUC: 0.9536

Tuning LogisticRegression

- Best parameters: {'C': 0.1, 'solver': 'liblinear'}

- Best cross-validation ROC-AUC: 0.9074

Appendix A.4: Model Evaluation

Each model was assessed on the validation set using metrics such as ROC-AUC, F1-score, and a detailed classification report, as demonstrated in the code below.

Code:

```
# Evaluate each model

results = {}

for name, model in best_models.items():

    print(f"\n{name} Validation Results:")

    y_val_pred = model.predict(X_val_scaled)

    y_val_proba = model.predict_proba(X_val_scaled)[:, 1]

    results[name] = {

    'roc_auc': roc_auc_score(y_val, y_val_proba),

    'f1_score': f1_score(y_val, y_val_pred),

    'y_val_pred': y_val_pred,

    'y_val_proba': y_val_proba

    }

  print(classification_report(y_val, y_val_pred))

    print(f"ROC AUC: {results[name]['roc_auc']:.4f}")

    print(f"F1 Score: {results[name]['f1_score']:.4f}")
```

Output:

Below is the detailed classification report which includes precision, recall, F1-score, and support.

RandomForest Validation Results:

```
        precision    recall  f1-score   support

        0       0.92     0.96         0.94   10471

        1       0.85     0.76         0.80   3529


        accuracy                      0.91   14000

  macro avg          0.89     0.86         0.87   14000

weighted avg       0.91    0.91     0.91  14000


ROC AUC: 0.9633

F1 Score: 0.8049
```

DecisionTree Validation Results:

```
                precision    recall f1-score   support

        0       0.88     0.93         0.91   10471

        1       0.75     0.63         0.69   3529


        accuracy                      0.85   14000

  macro avg          0.82     0.78         0.80   14000

weighted avg       0.85   0.85     0.85          14000


ROC AUC: 0.8808

F1 Score: 0.6855
```

XGBoost Validation Results:

```
        precision    recall  f1-score   support

        0       0.92     0.94         0.93   10471

        1       0.82     0.76         0.79   3529
```

```
          accuracy                       0.90   14000
   macro avg         0.87     0.85          0.86   14000
weighted avg         0.90   0.90      0.90          14000
```

ROC AUC: 0.9576

F1 Score: 0.7906

LogisticRegression Validation Results:

```
          precision      recall  f1-score   support
          0        0.93     0.84          0.88   10471
          1        0.63     0.82          0.71    3529

          accuracy                       0.83   14000
   macro avg         0.78     0.83          0.80   14000
weighted avg         0.86   0.83      0.84          14000
```

ROC AUC: 0.9057

F1 Score: 0.7134

Model Comparison Table

Model	**Accuracy**	**ROC AUC**	**F1 Score (Class 1)**	**Precision (Class 1)**	**Recall (Class 1)**
Random Forest	0.91	0.9633	0.8049	0.85	0.76
XGBoost	0.90	0.9576	0.7906	0.82	0.76

Decision Tree	0.85	0.8808	0.6855	0.75	0.63
Logistic Regression	0.83	0.9057	0.7134	0.63	0.82

Appendix A.5: Model Performance Comparison

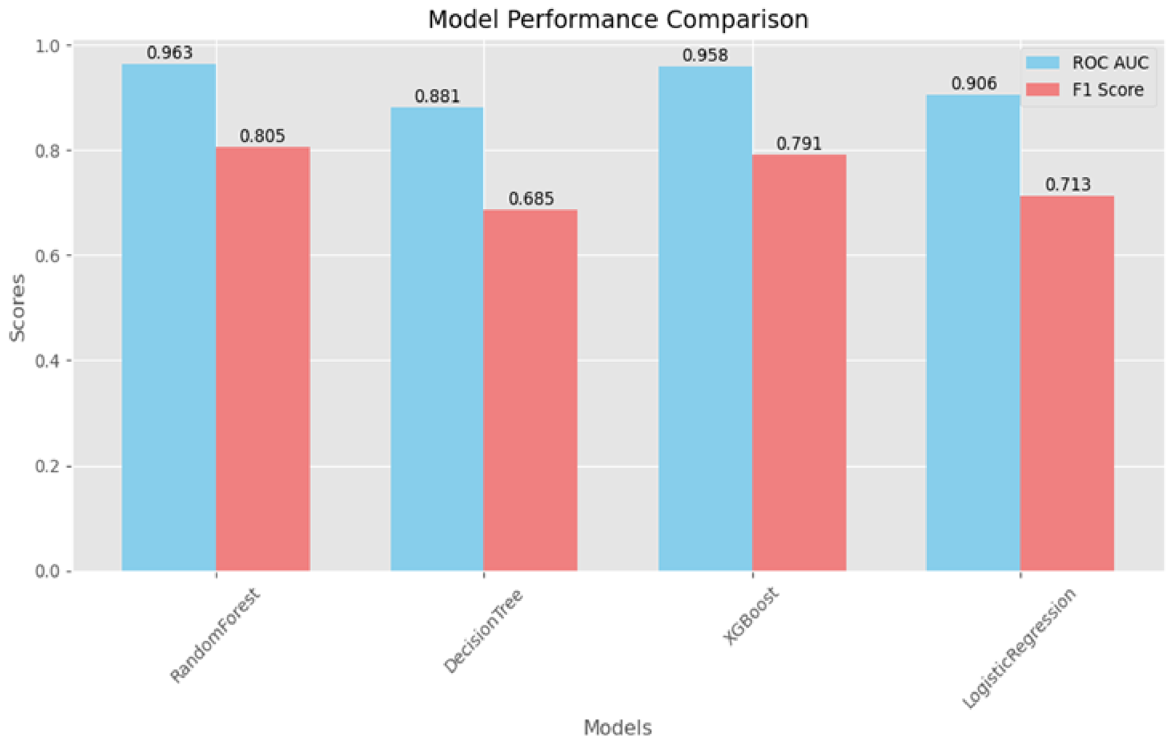
To compare the effectiveness of all trained models, we evaluated them using the validation dataset based on key performance metrics: Accuracy, ROC AUC, F1-Score, Precision, and Recall.

A grouped bar chart was generated to visually compare the ROC AUC and F1 scores across all models, using the code provided below.

Code:

```
# Collect metrics

model_names = list(results.keys())

roc_auc_scores = [results[name]['roc_auc'] for name in model_names]

f1_scores = [results[name]['f1_score'] for name in model_names]


# Plot comparison

fig, ax = plt.subplots(figsize=(10, 6))

x = np.arange(len(model_names))

width = 0.35

ax.bar(x - width/2, roc_auc_scores, width, label='ROC AUC', color='skyblue')

ax.bar(x + width/2, f1_scores, width, label='F1 Score', color='lightcoral')

ax.set_xlabel('Models')

ax.set_ylabel('Scores')
ax.set_title('Model Performance Comparison')

ax.set_xticks(x)

ax.set_xticklabels(model_names, rotation=45)

ax.legend()

for i, (roc, f1) in enumerate(zip(roc_auc_scores, f1_scores)):

        ax.text(i - width/2, roc + 0.01, f'{roc:.3f}', ha='center')

        ax.text(i + width/2, f1 + 0.01, f'{f1:.3f}', ha='center')

plt.tight_layout()

plt.show()
```

Output:



Appendix A.6: Make Final Predictions

Identify the most effective model using ROC-AUC scores and use it to generate predictions on the test dataset with the following code snippet.

Code:

```
best_model_name = max(results, key=lambda x: results[x]['roc_auc'])

best_model = best_models[best_model_name]

print(f"Best model: {best_model_name} (ROC AUC:
{results[best_model_name]['roc_auc']:.4f})")


# Make predictions

predictions = best_model.predict(X_test_scaled)

probabilities = best_model.predict_proba(X_test_scaled)[:, 1]

print(f"Generated {len(predictions)} predictions")

print(f"Default rate: {(predictions == 1).mean():.4f}")


# Save predictions

output = pd.DataFrame({'Prediction': predictions, 'Probability': probabilities})

output.to_csv('test_predictions.csv', index=False)

print("Predictions saved to 'test_predictions.csv'")
```

Output:

```
Best model: RandomForest (ROC AUC: 0.9633)

Generated 33000 predictions

Default rate: 0.2061

Predictions saved to 'test_predictions.csv'
```

Appendix A.7: Visualizations

To support the model evaluation and comparison, we generated the following visualizations.

1. **ROC Curve – Receiver Operating Characteristic:** This curve helps to visualize the trade-off between the True Positive Rate (Recall) and the False Positive Rate for different classification thresholds.
2. **Precision-Recall Curve:** This curve is especially useful for imbalanced datasets, as it focuses on the minority class (defaults in our case).

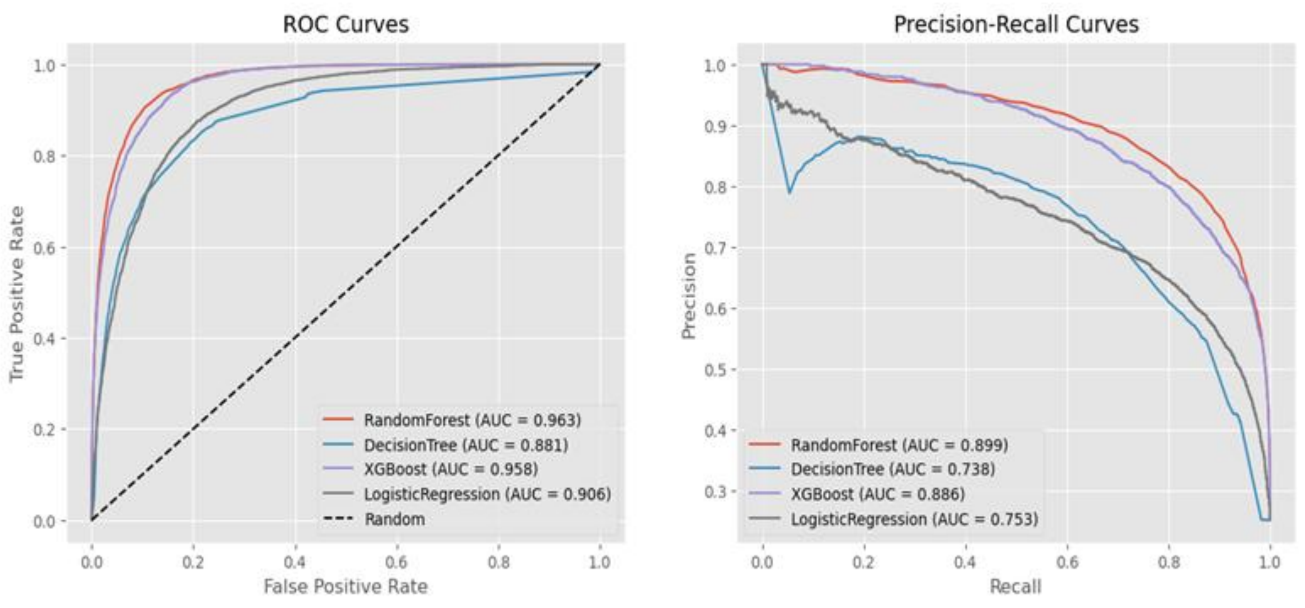
The following code is used to generate ROC Curve and Precision-Recall Curve.

Code:

```
import matplotlib.pyplot as plt

import seaborn as sns  # Required for heatmap; ensure seaborn is installed

from sklearn.metrics import roc_curve, precision_recall_curve, auc

import numpy as np

import pandas as pd

 # Set up the plotting style (use 'ggplot' or another valid matplotlib style if seaborn is unavailable).

plt.style.use('ggplot')  # Fallback style; alternatively, use 'seaborn-v0_8' if seaborn is installed.

 # Create a figure with subplots

fig = plt.figure(figsize=(15, 12))

 # 1. ROC Curve

plt.subplot(2, 2, 1)

for name, model in best_models.items():

        y_val_proba = model.predict_proba(X_val_scaled)[:, 1]

        fpr, tpr, _ = roc_curve(y_val, y_val_proba)

        roc_auc = auc(fpr, tpr)

        plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.3f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')
```

```
    plt.title('ROC Curves')
    plt.legend(loc='lower right')
    plt.grid(True)
    # 2. Precision-Recall Curve
    plt.subplot(2, 2, 2)
    for name, model in best_models.items():
        y_val_proba = model.predict_proba(X_val_scaled)[:, 1]
        precision, recall, _ = precision_recall_curve(y_val, y_val_proba)
        pr_auc = auc(recall, precision)
        plt.plot(recall, precision, label=f'{name} (AUC = {pr_auc:.3f})')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curves')
    plt.legend(loc='lower left')
    plt.grid(True)
    # # Display all plots
    # plt.suptitle('Model Evaluation Plots', fontsize=16, y=1.05)
    plt.show()
```

Output:



These visual insights help in making data-backed decisions and further validate the choice of Random Forest as the final model due to its stable and well-balanced performance across key evaluation metrics.

References

1. Kaggle. (2023). *Customer Credit Default Prediction Dataset*. Retrieved from https://www.kaggle.com/datasets
2. Pedregosa, F., et al. (2011). *GridSearchCV – Scikit-learn 1.4.2 documentation*. Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
3. Pedregosa, F., et al. (2011). *RobustScaler – Scikit-learn 1.4.2 documentation*. Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

4. Pedregosa, F., et al. (2011). *LabelEncoder – Scikit-learn 1.4.2 documentation*. Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
5. Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression* (3rd ed.). Wiley.
6. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81–106. https://doi.org/10.1007/BF00116251
7. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324
8. Chen, T., & Guestrin, C. (2016, August). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). ACM. https://doi.org/10.1145/2939672.2939785