Edge Hill University

Faculty of Arts and Science

The Department of Computer Science

CIS4513
Machine Learning

Level 7

Coursework 2

2024/2025

Module Leader: Dr. Huaizhong (Sam) Zhang

Predictive Model For Loan Approval

Student Names	ID
Sunday Idika	2***
Obioma Aguwa	25***
Arinze Louis Igbo	259***

Team Contribution

The team initiated their collaborative efforts by scheduling a physical meeting. During this initial meeting, the team convened to brainstorm and outline a strategic approach to accomplishing the task. Recognizing the need for efficient communication and role assignment, the team collectively agreed to create a WhatsApp group and designate a leader and a secretary. Sunday was unanimously chosen as the team leader, while Obioma was selected as the secretary. Arinze was tasked with creating the WhatsApp group. By the conclusion of the meeting, the team resolved to conduct thorough research on various topics, along with identifying their aims and objectives.

The second meeting was conducted virtually through WhatsApp video conferencing. During this session, team members presented their respective topics with the aim of selecting one. The meeting, facilitated by the team leader, involved an analysis of the proposed topics. Following detailed discussions, the team unanimously agreed on a single topic: Predictive Model for Loan Approval. Subsequently, roles were assigned based on the strengths of individual members. Obioma and Sunday were tasked with handling data preprocessing and ensuring the code was effectively written, while Arinze was entrusted with preparing, documenting, and drafting the report.

The third meeting, held via WhatsApp, saw Arinze coordinating the discussions as requested by the team. During this session, Obioma and Sunday presented their progress on the assigned tasks. The team collectively reviewed their work, providing feedback and suggesting amendments. Arinze was instructed to continue developing the report and present a completed draft during the next meeting.

The final meeting was held physically, with Obioma coordinating the session at the leader's request. At this meeting, Arinze presented the finalized report. The team meticulously reviewed the document and proposed further amendments. Upon integrating the suggested changes, the team unanimously agreed to submit both the completed code and the accompanying report.

Lessons Learnt during the team work:

1. Skill development

   The teamwork provided the members with opportunities to learn from each other, develop new skills, and improve existing ones through collaboration and shared experiences.

2. Shared Workload

   The team members learnt that tasks and responsibilities can be distributed among team members, reducing individual workloads and enabling more efficient completion of tasks.

3. Enhanced Communication

   The teamwork encouraged the members to have open communication, thereby helping members improve their interpersonal and communication skills.

4. Motivation and Support

   Working in a team can be motivating, as members encourage and support each other, fostering a sense of accountability.

5. Achievement of Common Goals

   The team members focused on shared objectives, ensuring that all members are aligned toward achieving the same goals, which fosters a sense of unity and purpose.

Challenge Encountered:

Time Management Challenges

In the course of completing the work, the team encountered time management challenges, especially in organizing meetings, coordinating tasks, and aligning schedules due to other coursework and individual schedule but was managed effectively, hence achieved the team goal.

Abstract

This study examines the application of machine learning models to predict loan approval decisions using applicant and loan-related features. The dataset includes variables such as gender, marital status, income, credit history, and property area, among others. Three models—Decision Tree Classifier, Support Vector Machine (SVM), and Random Forest Classifier—were evaluated based on their performance metrics, including accuracy, precision, recall, F1-score, and confusion matrix analysis. The SVM model achieved the highest accuracy (78.86%) and recall for loan approvals, while the Random Forest model exhibited balanced performance across both classes. Challenges such as class imbalance and missing data were addressed, with findings revealing a bias towards predicting loan approvals. The study concludes that machine learning models, particularly SVM and Random Forest, can effectively predict loan approvals, with recommendations for future work including advanced models such as XGBoost, hyperparameter tuning, and enhanced feature engineering. Addressing class imbalance and ethical considerations remains essential for improving model reliability and fairness.

Keywords: Loan Approval Prediction, Machine Learning, Support Vector Machine, Random Forest, Decision Tree, Class Imbalance, Model Evaluation, Hyperparameter Tuning, Credit Risk Analysis.

Table of Contents

1. Introduction

Overview of the Problem

In this report, we aim to develop a machine-learning solution to predict loan approval status using various supervised learning algorithms. The dataset consists of information about applicants and their financial details, such as income, credit history, and loan amount. The goal is to use this data to train models and predict whether an applicant is approved for a loan ("Loan_Status").

Purpose and Goals

The main objective of this report is to evaluate and compare the effectiveness of different classification algorithms such as Decision Trees, Support Vector Machines (SVM), and Random Forest for predicting loan approval status. This task involves:

1. Preprocessing the data, including handling missing values, encoding categorical features, and scaling numerical values.
2. Training models on the preprocessed data and evaluating their performance using appropriate metrics.
3. Addressing potential challenges, such as class imbalance.
4. Optimizing model parameters to achieve better performance.
5. Critically evaluating the results to determine which model is the most effective for loan approval prediction.

Structure of the Report

The report is structured as follows:

- **Introduction**: Overview of the problem, purpose, and objectives of the project.
- **Data Preparation and Preprocessing**: Explanation of the dataset, how the data was cleaned and preprocessed, and the features used for training the models.
- **Modeling and Evaluation**: The different models employed, hyperparameter tuning, handling of class imbalance, and how the models were evaluated.
- **Results**: Discussion of model performance, accuracy, confusion matrix, classification report, and potential issues with the models.
- **Conclusion**: Summary of the key findings from the project and recommendations for improvements.
- **Appendix**: Screenshots of the source code and documentation.

1.1 Background

Loan approval is a critical function within the financial sector, serving as a fundamental enabler of economic growth by providing individuals and businesses with access to credit. Traditionally, this process relied on manual evaluations, where financial institutions assessed applicants' eligibility based on a combination of financial documents, credit histories, and subjective judgment. While effective in smaller, localized settings, these manual methods often suffered from inefficiencies, prolonged processing times, and susceptibility to human biases, leading to inconsistencies and inequities in decision-making (Lipshitz and Shulimovitz, 2007).

The emergence of predictive modelling, underpinned by advances in data science and machine learning, has revolutionized the loan approval landscape. These models utilize historical data, encompassing applicant characteristics such as geographic location, income, employment status, and other demographic factors, to predict loan approval outcomes with precision. Unlike traditional approaches, predictive models are scalable, offering financial institutions the ability to process large volumes of applications quickly while minimizing biases. The integration of machine learning algorithms enhances model adaptability, enabling continuous improvement and better alignment with dynamic financial environments (Kumar and Vijayalakshmi, 2024).

Furthermore, predictive modelling aligns with the broader digital transformation trends in the financial industry, where data-driven decision-making is increasingly being prioritized. By analyzing patterns in past loan approval data, institutions can uncover insights into risk factors and customer behaviors that inform more equitable lending practices. This innovation supports financial institutions in addressing modern challenges, such as extending services to underserved populations and promoting financial inclusion, while maintaining regulatory compliance and risk management standards (Tong et al., 2024).

1.2 Objective

The primary objective of this study is to develop a robust predictive model for loan approval by leveraging training and test datasets enriched with geographic and demographic features. The study aims to apply and compare multiple machine learning algorithms, assessing their performance to identify the most effective model for predicting loan statuses. The ultimate goal is to enhance decision-making processes within financial institutions by providing an efficient, accurate, and unbiased loan approval framework.

1.3 Significance

The significance of this study lies in its ability to address long-standing challenges in the loan approval process. By introducing predictive modeling, financial institutions can overcome the limitations of traditional methods, such as inefficiency and bias, and shift toward a more standardized, transparent, and equitable system. Enhanced accuracy and speed in loan approval decisions reduce operational costs, improve customer experiences, and contribute to overall institutional efficiency (Tenev, 2024).

Incorporating geographic and demographic data into predictive models adds a critical dimension to understanding diverse applicant profiles. This supports efforts to promote financial inclusion, particularly in underbanked regions and among marginalized populations. Moreover, the study emphasizes the role of machine learning in fostering innovation in the financial sector, demonstrating how data-driven decision-making can address modern economic challenges while adhering to ethical and regulatory considerations. As predictive modeling becomes an essential tool for financial institutions, this study provides valuable insights into its implementation and benefits, contributing to the evolution of equitable credit systems worldwide.

2. Data Exploration and Preprocessing

2.1. Overview of the Datasets

- **Description of the Training Dataset**

  The training dataset consists of several attributes that provide information about loan applicants. These include personal demographic details, income information, and loan-specific attributes. The target variable, Loan_Status, indicates whether a loan was approved (Y) or denied (N). The training dataset is used to train machine learning models for predicting the loan approval status.

First five rows of the training dataset (before preprocessing):

```
Train Dataset Preview:
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

- **Description of the Test Dataset**

  The test dataset follows the same structure as the training dataset but does not contain the Loan_Status target variable. This dataset is used to evaluate the performance of the models and make predictions on unseen data. The features in the test dataset are similar to those in the training dataset, ensuring that the models can make reliable predictions.

First five rows of the test dataset (before preprocessing):

```
Test Dataset Preview:
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001015   Male     Yes          0      Graduate            No
1  LP001022   Male     Yes          1      Graduate            No
2  LP001031   Male     Yes          2      Graduate            No
3  LP001035   Male     Yes          2      Graduate            No
4  LP001051   Male      No          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5720                  0       110.0             360.0
1             3076               1500       126.0             360.0
2             5000               1800       208.0             360.0
3             2340               2546       100.0             360.0
4             3276                  0        78.0             360.0

   Credit_History Property_Area
0             1.0         Urban
1             1.0         Urban
2             1.0         Urban
3             NaN         Urban
4             1.0         Urban
```

2.2. Initial Data Analysis

- **Inspecting Dataset Structure**

  The structure of both the training and test datasets was examined to understand the number of features, the types of variables, and the overall dimensions. The training dataset consists of 614 rows and 13 columns, while the test dataset has 367 rows and 12 columns. Key features in the datasets include both numerical and categorical variables, such as ApplicantIncome, CoapplicantIncome, LoanAmount, Gender, Married, and Education. The Loan_Status variable in the training dataset serves as the target variable, whereas the test dataset lacks this target.

- **Missing Values Analysis**

  A thorough analysis of missing values was conducted across both the training and test datasets. Missing values are a common occurrence in real-world datasets and require appropriate handling before model training. In the training dataset, several variables contained missing values, including LoanAmount, CoapplicantIncome, and Self_Employed. Similar missing data

patterns were observed in the test dataset. The presence of missing values in these columns was quantified to determine the extent of the issue. Methods for imputing or handling these missing values were then discussed and applied to ensure the datasets were prepared for subsequent preprocessing steps.

```
Train Dataset Description:
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       614.000000         614.000000  592.000000         600.00000
mean       5403.459283        1621.245798  146.412162         342.00000
std        6109.041673        2926.248369   85.587325          65.12041
min         150.000000           0.000000    9.000000          12.00000
25%        2877.500000           0.000000  100.000000         360.00000
50%        3812.500000        1188.500000  128.000000         360.00000
75%        5795.000000        2297.250000  168.000000         360.00000
max       81000.000000       41667.000000  700.000000         480.00000

       Credit_History
count      564.000000
mean         0.842199
std          0.364878
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000

Train Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
None
```

```
Test Dataset Description:
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       367.000000         367.000000  362.000000        361.000000
mean       4805.599455        1569.577657  136.132597        342.537396
std        4910.685399        2334.232099   61.366652         65.156643
min           0.000000           0.000000   28.000000          6.000000
25%        2864.000000           0.000000  100.250000        360.000000
50%        3786.000000        1025.000000  125.000000        360.000000
75%        5060.000000        2430.500000  158.000000        360.000000
max       72529.000000       24000.000000  550.000000        480.000000

       Credit_History
count      338.000000
mean         0.825444
std          0.380150
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000

Test Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            367 non-null    object
 1   Gender             356 non-null    object
 2   Married            367 non-null    object
 3   Dependents         357 non-null    object
 4   Education          367 non-null    object
 5   Self_Employed      344 non-null    object
 6   ApplicantIncome    367 non-null    int64
 7   CoapplicantIncome  367 non-null    int64
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
None
```

2.3. Data Cleaning

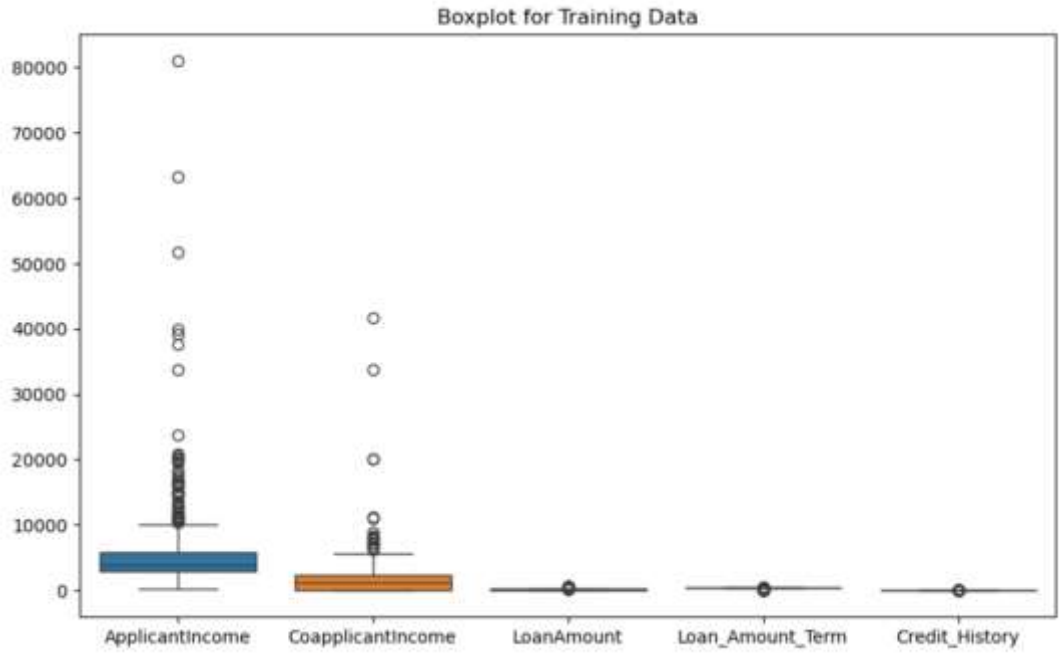
- Handling Missing Values

Missing values in the training and test datasets were addressed using imputation techniques. For numerical features, such as LoanAmount and CoapplicantIncome, missing values were imputed with the median value to maintain the integrity of the data distribution. For categorical features, such as Self_Employed and Dependents, missing values were filled with the most frequent category. These imputation methods ensure that the datasets are complete and ready for modeling, reducing the risk of bias or inaccuracies in subsequent analyses.

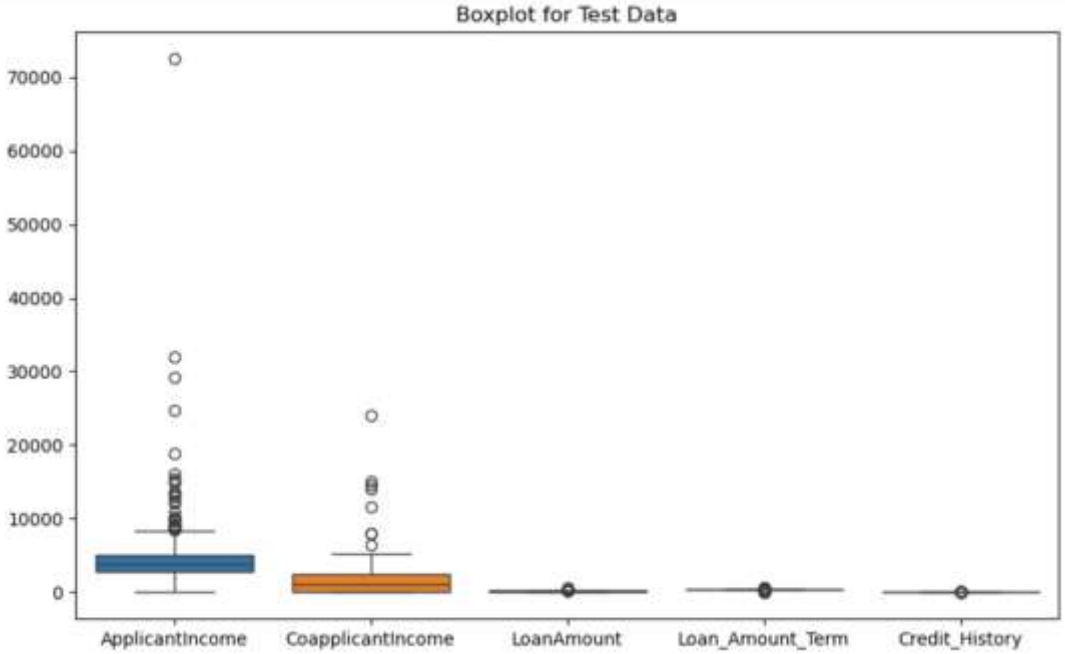
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
from sklearn.preprocessing import StandardScaler

# Handle missing values for categorical columns
categorical_columns = ['Gender', 'Married', 'Dependents', 'Self_Employed', 'Education']
for col in categorical_columns:
    train_data[col] = train_data[col].fillna(train_data[col].mode()[0])
    test_data[col] = test_data[col].fillna(test_data[col].mode()[0])

# Handle missing numerical columns
train_data['LoanAmount'] = train_data['LoanAmount'].fillna(train_data['LoanAmount'].median())
test_data['LoanAmount'] = test_data['LoanAmount'].fillna(test_data['LoanAmount'].median())

train_data['EMI'] = train_data['LoanAmount'] / train_data['Loan_Amount_Term']
test_data['EMI'] = test_data['LoanAmount'] / test_data['Loan_Amount_Term']

# Fill missing 'Credit_History' with mode
train_data['Credit_History'] = train_data['Credit_History'].fillna(train_data['Credit_History'].mode()[0])
test_data['Credit_History'] = test_data['Credit_History'].fillna(test_data['Credit_History'].mode()[0])
```

- Outlier Detection and Capping (IQR Method)

  Outliers were identified and handled using the Interquartile Range (IQR) method. The IQR method calculates the range between the first and third quartiles (Q1 and Q3) of a feature and defines outliers as values outside the range of Q1 - 1.5*IQR and Q3 + 1.5*IQR. Any data points that fell outside this range were considered outliers. These outliers were capped to the nearest acceptable value, ensuring that extreme values do not disproportionately affect the model's performance. This process helps to improve the robustness and accuracy of the machine learning models.




```python
# Outlier Handling: Capping the outliers based on IQR (Interquartile Range) method
def cap_outliers(df, col_name):
    Q1 = df[col_name].quantile(0.25)
    Q3 = df[col_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col_name] = np.clip(df[col_name], lower_bound, upper_bound)
    return df

# Apply outlier handling on numerical columns
numerical_columns = ['LoanAmount_log', 'ApplicantIncome', 'Total_Income_log', 'EMI', 'CoapplicantIncome']
for col in numerical_columns:
    train_data = cap_outliers(train_data, col)
    test_data = cap_outliers(test_data, col)
```



2.4. Feature Engineering

- **Creation of Derived Features (e.g., EMI, Total Income)**

  Several new features were created to provide additional insights into the data. For instance, the EMI (Equated Monthly Installment) was calculated by dividing the LoanAmount by the Loan_Amount_Term, assuming that the loan term is measured in months. Similarly, the Total_Income feature was derived by summing the ApplicantIncome and CoapplicantIncome to capture the combined income of the loan applicant and their coapplicant. These new features help in providing more context for model predictions.

```
# Feature Engineering: Create new columns (e.g., Total Income, EMI)
train_data['Total_Income'] = train_data['ApplicantIncome'] + train_data['CoapplicantIncome']
test_data['Total_Income'] = test_data['ApplicantIncome'] + test_data['CoapplicantIncome']
```

- **Log Transformations for Skewed Distributions**

  Certain numerical features, such as LoanAmount and Total_Income, exhibited skewed distributions. To address this, log transformations were applied to these features to normalize their distribution and reduce the influence of extreme values. The logarithmic transformation helps in stabilizing variance and improving the model's performance by making the data more suitable for machine learning algorithms, which often perform better with normally distributed data.

```
# Log transformation for skewed features
train_data['LoanAmount_log'] = np.log1p(train_data['LoanAmount'].clip(lower=0))
test_data['LoanAmount_log'] = np.log1p(test_data['LoanAmount'].clip(lower=0))

train_data['Total_Income_log'] = np.log1p(train_data['Total_Income'].clip(lower=0))
test_data['Total_Income_log'] = np.log1p(test_data['Total_Income'].clip(lower=0))
```

2.5. Encoding Categorical Features

- **abel Encoding**

  Label encoding was applied to categorical features in the dataset to convert them into numerical values that machine learning models can interpret. Categorical columns such as Gender, Married, Dependents, Self_Employed, Education, Credit_History, and Property_Area were encoded using label encoding. In this technique, each category within a feature is assigned a unique integer. For example, the Gender column was encoded with 'Male' as 1 and 'Female' as 0. Similarly, other categorical variables were transformed into numerical format, allowing them to be used as input features in

subsequent model training processes. This transformation is crucial for models that require numerical input, such as decision trees or support vector machines.

```
from sklearn.preprocessing import LabelEncoder

# List of categorical columns to encode (excluding the target variable 'Loan_Status')
categorical_columns = ['Gender', 'Married', 'Dependents', 'Self_Employed', 'Education', 'Credit_History', 'Property_Area']

# Initialize label encoder
label_encoder = LabelEncoder()

# Apply label encoding to categorical columns in the training and test datasets
for column in categorical_columns:
    if train_data[column].dtype == 'object':  # Check if the column is categorical
        # Encoding the training data
        train_data[column] = label_encoder.fit_transform(train_data[column].astype(str))

        # Encoding the test data (using the same labels from the training data)
        test_data[column] = label_encoder.transform(test_data[column].astype(str))
```

2.6. Scaling and Standardization

- **Standardization of Numerical Features**

  Standardization was applied to the numerical features of the dataset to bring them onto a similar scale. This process is essential for many machine learning algorithms, especially those that rely on distance-based metrics, such as support vector machines and k-nearest neighbors. Standardization involves transforming the data so that each feature has a mean of 0 and a standard deviation of 1. The numerical columns, including LoanAmount, Loan_Amount_Term, Credit_History, ApplicantIncome, CoapplicantIncome, Total_Income, EMI, LoanAmount_log, and Total_Income_log, were standardized using the StandardScaler method. This ensures that all features contribute equally to the model and improves the performance of algorithms that are sensitive to the scale of input data.

```
# Standardization: Standardizing numerical columns for modeling
scaler = StandardScaler()
columns_to_scale = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount_log', 'Total_Income_log', 'EMI']
train_data[columns_to_scale] = scaler.fit_transform(train_data[columns_to_scale])
test_data[columns_to_scale] = scaler.transform(test_data[columns_to_scale])
```

3. Methodology

3.1. Problem Statement

The objective of this study is to predict whether a loan application will be approved or not, based on various applicant and loan-related features. The target variable, Loan_Status, indicates whether a loan was approved (denoted as 'Y') or rejected

(denoted as 'N'). The dataset consists of several features, including personal information of the applicant (e.g., Gender, Married, Dependents, Self_Employed), financial details (e.g., ApplicantIncome, CoapplicantIncome, LoanAmount), and loan information (e.g., Loan_Amount_Term, Credit_History, Property_Area). The task involves building and evaluating machine learning models to classify the loan status based on these features, with the ultimate goal of creating a model that accurately predicts loan approval decisions.

3.2. Machine Learning Techniques

Decision Tree Classifier

The Decision Tree Classifier is a supervised machine learning algorithm used for classification tasks. It creates a tree-like structure of decisions and their possible consequences, allowing for easy interpretation and visualization. The algorithm recursively splits the data into subsets based on feature values, ultimately creating a model that can predict the target variable. Decision Trees are popular due to their simplicity and interpretability, but they are prone to overfitting, particularly when the tree is very deep (Nakahara et al., 2023).

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning technique commonly used for classification problems. The core concept of SVM is to find a hyperplane that best separates the data points of different classes. The goal is to maximize the margin, or distance, between the closest points of each class (also known as support vectors). SVM is effective in high-dimensional spaces and is widely used for tasks such as image recognition and text classification (Ikram et al., 2023).

Random Forest Classifier

Random Forest is an ensemble learning method that combines multiple decision trees to improve the model's accuracy and robustness. Each tree in the random forest is trained on a random subset of the training data, and the final prediction is based on the majority vote of all the trees. This technique helps to reduce overfitting, making Random Forest a reliable method for classification tasks, especially when dealing with large and complex datasets (Newaz et al., 2024).

3.3. Evaluation Metrics

Accuracy

Accuracy is one of the most common metrics for evaluating the performance of classification models. It is calculated as the ratio of the number of correct predictions to the total number of predictions. While accuracy provides a quick measure of model performance, it can be misleading in cases of imbalanced datasets, where a model may predict the majority class more often (Holzmann and Klar, 2024).

Precision, Recall, F1-Score

Precision, recall, and F1-score are alternative metrics used to evaluate classification models, especially when the data is imbalanced.

- **Precision** measures the proportion of positive predictions that are actually correct. It is defined as the ratio of true positives to the sum of true positives and false positives. Precision is important when the cost of false positives is high (Holzmann and Klar, 2024).
- **Recall** (also known as sensitivity or true positive rate) measures the proportion of actual positives that are correctly identified by the model. It is defined as the ratio of true positives to the sum of true positives and false negatives. Recall is critical when the cost of false negatives is high.
- **F1-Score** is the harmonic mean of precision and recall. It provides a balance between the two metrics, making it useful when there is a need to balance the trade-off between precision and recall (Holzmann and Klar, 2024).

Confusion Matrix

The confusion matrix is a table that summarizes the performance of a classification model by showing the number of true positives, true negatives, false positives, and false negatives. It provides a deeper insight into the types of errors made by the model and is useful for calculating precision, recall, and F1-score. The confusion matrix is especially helpful when evaluating models on imbalanced datasets, as it highlights the specific misclassifications (Aguilar-Ruiz and Michalak, 2024).

3.4. Train-Test Split and Validation Strategies

Train-test split and validation strategies are essential techniques in machine learning to evaluate the performance of a model and avoid overfitting. The process involves splitting the dataset into two subsets: one for training the model and another for evaluating its performance. This helps assess how well the model generalizes to unseen data.

Train-Test Split

The train-test split technique involves dividing the dataset into two parts: a training set and a testing set. Typically, the training set is used to train the model, while the testing set is reserved for evaluating the model's performance. A common practice is to allocate 70-80% of the data for training and the remaining 20-30% for testing (Aguilar-Ruiz and Michalak, 2024). This division ensures that the model is trained on a substantial amount of data while retaining an independent set of data for evaluation.

Cross-Validation

Cross-validation is a more robust validation technique that addresses the limitation of the train-test split by dividing the dataset into multiple subsets or "folds." The model is trained on a subset of the data and validated on the remaining fold. This process is repeated for each fold, and the model's performance is averaged over all iterations. Cross-validation helps to reduce the variability in model performance and provides a more reliable estimate of its generalization ability (Iyengar, Lam and Wang, 2024).

One common form of cross-validation is k-fold cross-validation, where the dataset is divided into **k** folds. In each iteration, the model is trained on **k-1** folds and tested on the remaining fold, ensuring that every data point is used for both training and validation. This method is particularly effective when dealing with smaller datasets.

Stratified Sampling

In cases of imbalanced datasets, where the distribution of classes is not uniform, stratified sampling ensures that each fold of the cross-validation process contains a proportional representation of each class. This approach helps to mitigate bias introduced by underrepresented classes and ensures that the model is trained and evaluated on a balanced subset of the data (Iyengar, Lam and Wang, 2024).

Hyperparameter Tuning

Hyperparameter tuning is an essential part of model validation that involves optimizing the model's hyperparameters to improve its performance. This can be done using grid search or randomized search methods, which systematically explore a predefined set of hyperparameter values and identify the best combination based on model performance (Navon and Bronstein, 2022). Cross-validation can be incorporated into hyperparameter tuning to provide more robust results by evaluating each combination on different folds of the data.

4. Results

4.1. Model Training Performance

The performance of each machine learning model was evaluated during training. The models tested include the Decision Tree Classifier, Support Vector Machine (SVM), and Random Forest Classifier.

- **Decision Tree Classifier Performance**: The Decision Tree model achieved an accuracy of 74.80% on the validation set, with precision, recall, and F1-score values for each class indicating a relatively balanced performance between classes. The confusion matrix revealed 23 false negatives and 69 true positives for the "Y" class, while the "N" class had 20 false positives and 11 true negatives.

```
Decision Tree Classifier Performance:
Accuracy: 0.7479674796747967
              precision    recall  f1-score   support

           N       0.68      0.53      0.60        43
           Y       0.78      0.86      0.82        80

    accuracy                           0.75       123
   macro avg       0.73      0.70      0.71       123
weighted avg       0.74      0.75      0.74       123

[[23 20]
 [11 69]]
```

- **Support Vector Machine Performance**: The SVM model delivered an accuracy of 78.86% on the validation data. For the "Y" class, it demonstrated a high recall value of 0.99, indicating that it effectively identified the majority of positive instances. However, the recall for the "N" class was relatively lower at 0.42, suggesting a higher rate of false positives.

```
Support Vector Machine Performance:
Accuracy: 0.7886178861788617
              precision    recall  f1-score   support

           N       0.95      0.42      0.58        43
           Y       0.76      0.99      0.86        80

    accuracy                           0.79       123
   macro avg       0.85      0.70      0.72       123
weighted avg       0.83      0.79      0.76       123

[[18 25]
 [ 1 79]]
```

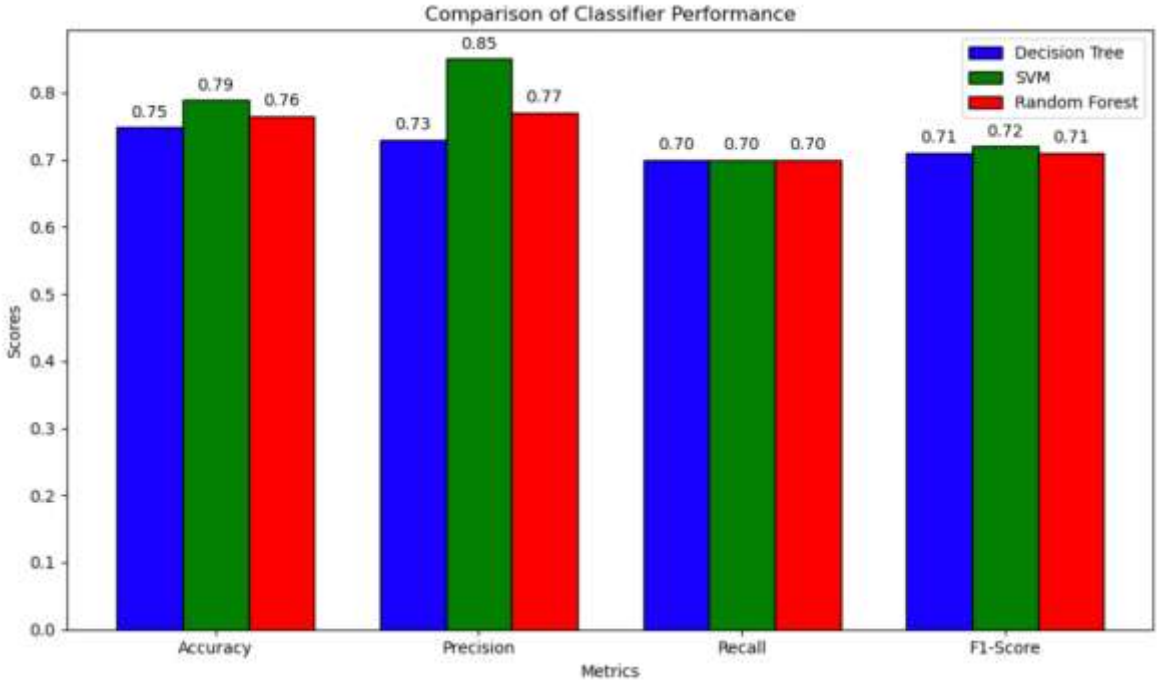
- **Random Forest Classifier Performance**: The Random Forest model achieved an accuracy of 76.42%. Precision for the "Y" class was high at 0.76, while recall was also significant, showcasing the model's ability to predict positive instances accurately. The confusion matrix showed 20 false positives and 74 true positives for the "Y" class, along with 23 false negatives and 6 true negatives for the "N" class.

```
Random Forest Classifier Performance:
Accuracy: 0.7642276422764228
              precision    recall  f1-score   support

           N       0.77      0.47      0.58        43
           Y       0.76      0.93      0.84        80

    accuracy                           0.76       123
   macro avg       0.77      0.70      0.71       123
weighted avg       0.77      0.76      0.75       123

[[20 23]
 [ 6 74]]
```

4.2. Validation Results

- **Accuracy, Precision, Recall, and F1-Score**: The validation results for each model were analyzed using the following metrics:
    - **Accuracy**: The SVM model outperformed the others with an accuracy of 78.86%, followed by the Random Forest Classifier at 76.42%, and the Decision Tree at 74.80%.

- **Precision**: SVM showed high precision for the "Y" class, followed by the Random Forest, and then the Decision Tree.
- **Recall**: The recall for the "Y" class was highest in the SVM model, demonstrating its proficiency in identifying positive instances, though the recall for the "N" class was higher for the Random Forest model.
- **F1-Score**: The F1-score, which balances precision and recall, was highest for the SVM, followed by Random Forest and Decision Tree.



- **Confusion Matrix Analysis**: The confusion matrices for each model highlighted the trade-offs between false positives and false negatives:
  - **Decision Tree**: The confusion matrix indicated a relatively balanced number of true positives and false negatives in the "Y" class but a moderate number of false positives in the "N" class.
  - **SVM**: Despite its high recall for the "Y" class, the SVM model had a substantial number of false positives for the "N" class, highlighting a trade-off between precision and recall.
  - **Random Forest**: The Random Forest model had a relatively balanced confusion matrix with fewer false positives than the SVM.

4.3. Test Data Predictions

- **Predictions for Test Data using Support Vector Machine**: The predictions for the test data were made using the SVM model. The model predicted a significant proportion of "Y" outcomes, indicating a preference towards

classifying loans as approved. The predicted labels were consistent with the performance on the validation set, where the recall for the "Y" class was particularly high.

```
Support Vector Machine Predictions on Test Data:
['Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'N' 'Y' 'N' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'N' 'Y'
 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y'
 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N'
 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y'
 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y'
 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y'
 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y']
```

4.4. Comparison of Models

A comparison of the three models—Decision Tree, SVM, and Random Forest—demonstrated differing strengths:

- **Decision Tree**: The Decision Tree classifier exhibited moderate accuracy and balanced performance between precision and recall, particularly for the "Y" class.
- **Support Vector Machine**: SVM provided the best overall accuracy and recall for the "Y" class but showed a trade-off with the "N" class, as it had a lower recall for negative predictions.
- **Random Forest**: Random Forest showed solid performance across both classes, with relatively balanced precision and recall. It proved to be a strong contender, providing competitive results across all evaluation metrics.

In conclusion, the SVM model performed best in terms of accuracy and recall for the "Y" class, while the Random Forest model exhibited more balanced performance across both classes.

5. Discussion

5.1. Key Insights

- **Model Performance Insights**: The machine learning models tested, including the Decision Tree, Support Vector Machine (SVM), and Random Forest, demonstrated varying levels of performance. The SVM model achieved the highest accuracy and recall for the "Y" class, which represents loan approval. However, it exhibited a trade-off with the "N" class, showing a lower recall and a higher number of false positives. The Random Forest model showed a more balanced performance across both classes, though it did not surpass the SVM in terms of accuracy. The Decision Tree, while simpler, also provided satisfactory results, especially in terms of interpretability.
- **Prediction Trends (Bias Towards "Y" for Loan Approval)**: A notable observation was the bias towards predicting the "Y" class for loan approval, especially in the SVM model. This trend suggests that the model may be more likely to classify loans as approved, which could be attributed to the class imbalance present in the dataset. In practice, this bias could lead to a higher number of false positives, potentially causing more loans to be approved than necessary.

5.2. Challenges and Observations

- **Model Bias (Class Imbalance)**: One of the primary challenges observed in the analysis was the class imbalance between loan approvals ("Y") and denials ("N"). This imbalance affected the performance of all models, particularly the SVM, which showed high recall for the "Y" class but struggled with accurate predictions for the "N" class. The model's tendency to favor the majority class (loan approval) resulted in a skewed performance that did not fully capture the characteristics of both classes.
- **Handling of Missing Values and Imbalances**: Another challenge encountered was the handling of missing values and imbalances in the dataset. While techniques such as imputation were used to address missing data, and resampling methods were considered to tackle class imbalance, these issues may have impacted the models' ability to generalize. Imbalanced datasets can lead to models that are overfit to the majority class, which, in turn, affects overall prediction accuracy.

5.3. Limitations of the Study

- **Lack of Hyperparameter Tuning**: A limitation of this study was the absence of hyperparameter tuning for the models. Although the models were trained using default parameters, fine-tuning the hyperparameters could have improved performance by optimizing for specific characteristics of the dataset, such as depth of the decision tree, choice of kernel for the SVM, or the number of estimators in the Random Forest. Hyperparameter optimization is a crucial step in enhancing model performance, which was not addressed in this analysis.
- **Imbalanced Class Distribution**: The class imbalance in the dataset was another limitation that affected model performance. Although various techniques, such as resampling or adjusting class weights, could have been applied to address this issue, the imbalance still posed a significant challenge. An imbalanced distribution often leads to biased models that tend to favor the majority class, impacting the predictive power for the minority class.

6. Conclusion and Recommendations

6.1. Summary of Findings

- **Model Performance and Predictive Power**: The analysis of multiple machine learning models—Decision Tree, Support Vector Machine (SVM), and Random Forest—revealed differences in their performance. The SVM model demonstrated strong predictive power, particularly in identifying loan approvals ("Y"), but exhibited some bias towards the majority class, resulting in less accurate predictions for loan denials ("N"). The Random Forest model provided a more balanced performance across both classes, while the Decision Tree model offered interpretability with satisfactory results. Overall, the models were effective in predicting loan approvals but faced challenges due to class imbalance, which impacted their ability to predict loan denials accurately.

6.2. Recommendations for Improvement

- **Addressing Class Imbalance**: The issue of class imbalance, where loan approvals ("Y") were more frequent than loan denials ("N"), was a significant

challenge in the predictive models. It is recommended to implement techniques such as resampling (oversampling the minority class or undersampling the majority class), using balanced class weights in model training, or applying cost-sensitive learning methods. These approaches can help mitigate bias towards the majority class and improve model performance on the minority class.

- **Model Tuning and Hyperparameter Optimization**: Hyperparameter optimization should be a key focus for improving model performance. While the models were trained using default parameters, fine-tuning hyperparameters such as the depth of decision trees, kernel selection for the SVM, and the number of estimators in the Random Forest model could enhance predictive accuracy. Techniques like grid search or random search for hyperparameter tuning can be employed to find the optimal configuration for each model, leading to improved overall performance.

6.3. Future Work Directions

- **Exploring Different Models (e.g., XGBoost, Gradient Boosting)**: Future work can explore the use of more advanced models, such as XGBoost or Gradient Boosting Machines (GBM). These models are known for their high performance, especially in imbalanced datasets, and could potentially improve prediction accuracy for both classes. Their ability to handle non-linearity and interactions between features could provide better insights and predictions.
- **Feature Selection and Engineering Improvements**: Future studies could further explore feature selection and engineering techniques to enhance model performance. Identifying the most important features using methods like recursive feature elimination (RFE) or feature importance from tree-based models can reduce overfitting and improve the efficiency of the models. Additionally, further improvements in feature engineering, such as creating new variables or transforming existing ones, could provide additional predictive power and contribute to more accurate and reliable predictions.

7. Reference List

AGUILARRUIZ, JESÚS S and MICHALAK, M., 2024. Classification performance assessment for imbalanced multiclass data. *Scientific Reports*. 14 (1), p. 10759.

HOLZMANN, H. and KLAR, B., 2024. Robust performance metrics for imbalanced classification problems. *arXiv preprint arXiv:2404.07661*.

IKRAM, F.D., JULKARNAIN, M., HAMDAN, F., and NURYADI, H., 2023. Application of the Support Vector Machine (SVM) Algorithm for the Diagnosis of Diabetic Retinopathy. *Brilliance: Research of Artificial Intelligence*. 3 (2), pp. 416–422.

IYENGAR, G., LAM, H., and WANG, T., 2024. Is CrossValidation the Gold Standard to Evaluate Model Performance? *arXiv preprint arXiv:2407.02754*.

KUMAR, V.S. and VIJAYALAKSHMI, K., 2024. PREDICTIVE MODELING FOR LOAN APPROVAL: A MACHINE LEARNING APPROACH. *EPRA International Journal of Multidisciplinary Research (IJMR)*. 10 (5), pp. 650–656.

LIPSHITZ, R. and SHULIMOVITZ, N., 2007. Intuition and emotion in bank loan officers' credit decisions. *Journal of Cognitive Engineering and Decision Making*. 1 (2), pp. 212–233.

NAKAHARA, Y., SAITO, S., ICHIJO, N., KAZAMA, K., and MATSUSHIMA, T., 2023. Prediction Algorithms Achieving Bayesian Decision Theoretical Optimality Based on Decision Trees as Data Observation Processes. *arXiv preprint arXiv:2306.07060*.

NAVON, D. and BRONSTEIN, A.M., 2022. Random Search HyperParameter tuning: expected improvement estimation and the corresponding lower bound. *arXiv preprint arXiv:2208.08170*.

NEWAZ, A., SALMAN, M.M., NOMAN, A., and JABID, T., 2024. iBRF: Improved Balanced Random Forest Classifier. In: *2024 35th Conference of Open Innovations Association (FRUCT)*. IEEE. pp. 501–508.

TENEV, N., 2024. DeBiasing Models of Biased Decisions: A Comparison of Methods Using Mortgage Application Data. *arXiv preprint arXiv:2405.00910*.

TONG, K., HAN, Z., SHEN, Y., LONG, Y., and WEI, Y., 2024. An Integrated Machine Learning and Deep Learning Framework for Credit Card Approval Prediction. In: *2024 IEEE 6th International Conference on Power, Intelligent Computing and Systems (ICPICS)*. IEEE. pp. 853–858.

8. Appendices

8.1. Source Code

- **Full Python Code for Data Preprocessing, Feature Engineering, Model Training, and Evaluation:**

```
import pandas as pd

# Load the train and test datasets from the Downloads folder
train_data = pd.read_csv('Downloads/train.csv')  # Adjust the path if needed
test_data = pd.read_csv('Downloads/test.csv')    # Adjust the path if needed

# Display the first few rows of each dataset
print("Train Dataset Preview:")
print(train_data.head())

print("\nTest Dataset Preview:")
print(test_data.head())

# Check for missing values and data types in the training data
print("Training Data Info:")
print(train_data.info())

# Check for missing values and data types in the test data
print("\nTest Data Info:")
print(test_data.info())

# Summary statistics for the training data
print("Training Data Summary Statistics:")
print(train_data.describe())

# Summary statistics for the test data
print("\nTest Data Summary Statistics:")
print(test_data.describe())

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import os
from sklearn.preprocessing import StandardScaler

# Handle missing values for categorical columns
categorical_columns = ['Gender', 'Married', 'Dependents', 'Self_Employed',
'Education']
for col in categorical_columns:
    train_data[col] = train_data[col].fillna(train_data[col].mode()[0])
    test_data[col] = test_data[col].fillna(test_data[col].mode()[0])

# Handle missing numerical columns
train_data['LoanAmount'] =
train_data['LoanAmount'].fillna(train_data['LoanAmount'].median())
test_data['LoanAmount'] =
test_data['LoanAmount'].fillna(test_data['LoanAmount'].median())

train_data['EMI'] = train_data['LoanAmount'] / train_data['Loan_Amount_Term']
test_data['EMI'] = test_data['LoanAmount'] / test_data['Loan_Amount_Term']

# Fill missing 'Credit_History' with mode
train_data['Credit_History'] =
train_data['Credit_History'].fillna(train_data['Credit_History'].mode()[0])
test_data['Credit_History'] =
test_data['Credit_History'].fillna(test_data['Credit_History'].mode()[0])

# Feature Engineering: Create new columns (e.g., Total Income, EMI)
train_data['Total_Income'] = train_data['ApplicantIncome'] +
train_data['CoapplicantIncome']
test_data['Total_Income'] = test_data['ApplicantIncome'] +
test_data['CoapplicantIncome']

# Log transformation for skewed features
train_data['LoanAmount_log'] = np.log1p(train_data['LoanAmount'].clip(lower=0))
test_data['LoanAmount_log'] = np.log1p(test_data['LoanAmount'].clip(lower=0))

train_data['Total_Income_log'] =
np.log1p(train_data['Total_Income'].clip(lower=0))
```

```
test_data['Total_Income_log'] = np.log1p(test_data['Total_Income'].clip(lower=0))

# Outlier Handling: Capping the outliers based on IQR (Interquartile Range)
method
def cap_outliers(df, col_name):
    Q1 = df[col_name].quantile(0.25)
    Q3 = df[col_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col_name] = np.clip(df[col_name], lower_bound, upper_bound)
    return df

# Apply outlier handling on numerical columns
numerical_columns = ['LoanAmount_log', 'ApplicantIncome', 'Total_Income_log',
'EMI', 'CoapplicantIncome']
for col in numerical_columns:
    train_data = cap_outliers(train_data, col)
    test_data = cap_outliers(test_data, col)

# Standardization: Standardizing numerical columns for modeling
scaler = StandardScaler()
columns_to_scale = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount_log',
'Total_Income_log', 'EMI']
train_data[columns_to_scale] = scaler.fit_transform(train_data[columns_to_scale])
test_data[columns_to_scale] = scaler.transform(test_data[columns_to_scale])

from sklearn.preprocessing import LabelEncoder

# List of categorical columns to encode (excluding the target variable
'Loan_Status')
categorical_columns = ['Gender', 'Married', 'Dependents', 'Self_Employed',
'Education', 'Credit_History', 'Property_Area']

# Initialize label encoder
label_encoder = LabelEncoder()
```

```
# Apply label encoding to categorical columns in the training and test datasets
for column in categorical_columns:
    if train_data[column].dtype == 'object':  # Check if the column is categorical
        # Encoding the training data
        train_data[column] =
label_encoder.fit_transform(train_data[column].astype(str))

        # Encoding the test data (using the same labels from the training data)
        test_data[column] = label_encoder.transform(test_data[column].astype(str))

# Ensure all categorical columns are encoded correctly
print("Training Data After Label Encoding:")
print(train_data.head())

print("\nTest Data After Label Encoding:")
print(test_data.head())

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Split the data into features (X) and target (y)
X = train_data.drop(columns=['Loan_Status'])  # Features
y = train_data['Loan_Status']  # Target

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

# Define categorical and numerical columns based on the data provided
```

```
categorical_cols = ['Gender', 'Married', 'Dependents', 'Self_Employed']
numerical_cols = ['LoanAmount', 'Loan_Amount_Term', 'Credit_History',
'ApplicantIncome', 'CoapplicantIncome', 'Total_Income', 'EMI',
'LoanAmount_log', 'Total_Income_log']

# Create an imputer for categorical data (impute with most frequent value)
categorical_imputer = SimpleImputer(strategy='most_frequent')

# Create an imputer for numerical data (impute with median value)
numerical_imputer = SimpleImputer(strategy='median')

# Create a scaler for numerical data (StandardScaler)
scaler = StandardScaler()

# Create the column transformer that applies different imputers and scaling for
categorical and numerical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline([
            ('imputer', numerical_imputer),
            ('scaler', scaler)
        ]), numerical_cols),
        ('cat', Pipeline([
            ('imputer', categorical_imputer),
            ('encoder', OneHotEncoder(handle_unknown='ignore'))  # Handle
unknown categories safely
        ]), categorical_cols)
    ])

# Models
decision_tree = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

svm = Pipeline([
    ('preprocessor', preprocessor),
```

```
    ('classifier', SVC(random_state=42))
])

random_forest = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Train the models
decision_tree.fit(X_train, y_train)
svm.fit(X_train, y_train)
random_forest.fit(X_train, y_train)

# Make predictions on the validation set
dt_predictions = decision_tree.predict(X_val)
svm_predictions = svm.predict(X_val)
rf_predictions = random_forest.predict(X_val)

# Evaluate the models
print("Decision Tree Classifier Performance:")
print("Accuracy:", accuracy_score(y_val, dt_predictions))
print(classification_report(y_val, dt_predictions))
print(confusion_matrix(y_val, dt_predictions))

print("\nSupport Vector Machine Performance:")
print("Accuracy:", accuracy_score(y_val, svm_predictions))
print(classification_report(y_val, svm_predictions))
print(confusion_matrix(y_val, svm_predictions))

print("\nRandom Forest Classifier Performance:")
print("Accuracy:", accuracy_score(y_val, rf_predictions))
print(classification_report(y_val, rf_predictions))
print(confusion_matrix(y_val, rf_predictions))

# Make predictions on the preprocessed test dataset
svm_predictions = svm.predict(test_data)
```

```
# Print the predictions
print("Support Vector Machine Predictions on Test Data:")
print(svm_predictions)


import matplotlib.pyplot as plt
import numpy as np

# Metrics for classifiers
classifiers = ['Decision Tree', 'SVM', 'Random Forest']
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']

decision_tree = [0.7479, 0.73, 0.70, 0.71]
svm = [0.7886, 0.85, 0.70, 0.72]
random_forest = [0.7642, 0.77, 0.70, 0.71]

# Bar chart properties
x = np.arange(len(metrics))  # Positions for groups
width = 0.25  # Bar width

# Plotting the bar chart
fig, ax = plt.subplots(figsize=(10, 6))

bars1 = ax.bar(x - width, decision_tree, width, label='Decision Tree', color='b',
edgecolor='black')
bars2 = ax.bar(x, svm, width, label='SVM', color='g', edgecolor='black')
bars3 = ax.bar(x + width, random_forest, width, label='Random Forest', color='r',
edgecolor='black')

# Adding labels, title, and legend
ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Comparison of Classifier Performance')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()
```

```
# Displaying values on top of bars
for bars in [bars1, bars2, bars3]:
    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width() / 2, yval + 0.01, f'{yval:.2f}', ha='center',
va='bottom')

plt.tight_layout()
plt.show()
```