MIT | Academy of Engineering

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

Predicting Loan Approvals

TY B.Tech. CI Project Implementation Report (SEM V)

submitted in partial fulfillment of the
requirements for the award of the degree

of

Bachelor of Technology

in

ELECTRONICS and TELECOMMUNICATION ENGINEERING

BY

Pranay Kahalkar, [202201070029]
Apoorva Singh, [202201070030]
Rishabh Yelne, [202201070036]
Pratik Bangar, [202201070038]

SCHOOL OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

MIT ACADEMY OF ENGINEERING, ALANDI (D), PUNE-412105

MAHARASHTRA (INDIA)

DECEMBER, 2024

MIT | Academy of Engineering

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

CERTIFICATE

It is hereby certified that the work which is being presented in the TY B.Tech. Computational Intelligence Project Design Report entitled ***Predicting Loan Approvals*** **,** in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Electronics and Telecommunication engineering** and submitted to the **School of Electronics and Telecommunication engineering of MIT Academy of Engineering, Alandi(D), Pune, Affiliated to Savitribai Phule Pune University (SPPU), Pune** isan authentic record of work carried out during an Academic Year 2023-2024, under the supervision of **Dr. Smita Kulkarni and Prof. Aswathy M. A.**, **School of Electronics and Telecommunication Engineering.**

Pranay Kahalkar	**202201070029**
Apoorva Singh	**202201070030**
Rishabh Yelne	**202201070139**
Pratik Bangar	**202201070137**

Date: 21th June, 2024

Signature of Project Advisor

Dr. Smita Kulkarni

School of Electronics & Telecommunication Engineering,

MIT Academy of Engineering, Alandi(D), Pune

Signature of Dean

Dr. Dipti Sakhare

School of Electronics & Telecommunication Engineering,

MIT Academy of Engineering, Alandi(D), Pune

(STAMP/SEAL)

Signature of Internal examiner/s

Name……

Affiliation……

Signature of External examiner/s

Name……

Affiliation……

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to our esteemed project advisor/guide, Dr. Smita Kulkarni., whose unwavering encouragement and invaluable guidance have been instrumental in the successful completion of this project. His expertise and support have illuminated our path, enriching our learning experience significantly.

We also wish to express our sincere appreciation to the respected School Dean, Dr. Dipti Sakhare, forher consistent motivation and unwavering support throughout this endeavor. Her encouragement has been a driving force behind our efforts.

Furthermore, we acknowledge and thank all the esteemed staff and faculty members whose wealth of experience, insightful advice, and unceasing cooperation have played a pivotal role in shaping this project. Their contributions have been invaluable, and we are truly grateful for their guidance and support, which have been indispensable throughout this journey.

1. Pranay Kahalkar

2.Apoorva Singh

3.Rishabh Yelne

4.Pratik Bangar

ABSTRACT

The growing complexity of loan approval processes requires efficient and accurate systems to streamline decision-making while minimizing risks. This project utilizes machine learning, specifically the AdaBoost algorithm, to achieve an impressive accuracy of 83% in predicting loan approvals. Key features such as credit history, income, and demographic details were analyzed using a dataset of loan applications to determine approval likelihood.

The model was deployed in a containerized environment with Streamlit, ensuring scalability and seamless real-time monitoring through cloud-based platforms. This deployment enables financial institutions to automate loan approval processes, reduce decision-making time, and enhance operational efficiency. The system also highlights the most influential factors affecting loan approval, providing actionable insights for more informed and consistent evaluations.

Future work will explore incorporating additional features, improving model fairness, and leveraging ensemble methods for enhanced prediction accuracy. This project establishes a robust foundation for scalable loan approval systems and aligns with the need for reliable, data-driven solutions in modern financial operations.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

Chapter 1

Introduction

This chapter introduces the motivations, problem statement, and objectives that underpin this project. It emphasizes the increasing need for efficient and accurate loan approval systems to address the challenges faced by financial institutions in minimizing risks and ensuring fair evaluations. The chapter outlines the development of a machine learning-based solution using the AdaBoost algorithm to predict loan approvals based on key applicant features such as credit history, income, and demographics. By presenting the project's objectives and scope, this chapter lays the groundwork for understanding its significance in enhancing decision-making, operational efficiency, and fairness within the financial sector.

.1 Motivation

The increasing integration of advanced technologies into financial operations has unlocked new opportunities to tackle critical challenges and optimize decision-making processes. Just as innovative solutions have transformed various industries, the application of machine learning and data-driven approaches has immense potential to revolutionize financial systems, particularly in loan approval workflows.

Imagine an intelligent system capable of analyzing applicant data and predicting loan approval outcomes with high accuracy in real-time. Such advancements not only enhance decision-making efficiency but also promote fairness, consistency, and operational scalability in financial institutions.

Our project is driven by the need to address the complexities of traditional loan approval processes. By leveraging machine learning techniques, specifically the AdaBoost algorithm, we aim to develop a robust and reliable system for predicting loan approvals. By analyzing key features such as credit history, income, and demographic details, this system ensures accurate, unbiased, and consistent evaluations, supporting financial institutions in managing risks effectively.

Through this project, we seek to demonstrate the practical applications of machine learning in the financial sector, contributing to the broader adoption of automated, data-driven solutions. By focusing on precise data analysis, real-time deployment, and scalable architecture, we aim to deliver a system that optimizes operational efficiency while enhancing customer satisfaction.

Ultimately, our goal is to create a dependable loan approval prediction system that not only improves financial decision-making but also aligns with global initiatives to modernize financial operations. By integrating cutting-edge technologies with a commitment to fairness and accuracy, this project aspires to set a benchmark in automated financial decision-making systems.

.2 Problem Statement

To identify and analyze features that determine whether a loan application is likely to be approved or rejected by leveraging applicant data. The goal is to develop a machine learning model that accurately classifies loan applications based on key features such as credit history, income, and demographics. This solution provides insights into the characteristics influencing loan approval decisions, enhancing prediction models, improving decision-making processes, and supporting efficient and fair financial operations.

.3 Objective and Scope

Objective

- **To study and analyze the features of loan applications using applicant data:** This includes identifying critical factors such as credit history, income, and demographic details that influence loan approval decisions.
- **To develop and implement a machine learning model (AdaBoost) for loan approval prediction:** The goal is to accurately classify loan applications as approved or rejected based on the identified features.
- **To deploy the machine learning model using containerization tools like Streamlit:** This involves ensuring scalability, accessibility, and ease of use for real-time predictions and monitoring.
- **To evaluate the model's performance and provide insights into loan approval factors:** This includes interpreting the results to enhance prediction accuracy, improve decision-making processes, and promote fair and efficient financial operations.

1.3 Organization of report

Organization of the report provides a brief overview of the structure and contents of the report.

1.3.1 Introduction

- Motivation for the Project: Discusses the challenges faced by financial institutions in processing loan applications efficiently and the growing need for accurate, automated systems to enhance decision-making and fairness.
- Problem Statement: Outlines the challenge of creating a machine learning-based system to predict loan approvals using key features such as credit history, income, and demographic details.
- Objectives and Scope: Defines the project's main goals, including studying features of loan applications, implementing the AdaBoost algorithm for prediction, and deploying the solution using Streamlit.

1.3.2. System Design

- Block Diagram of the Proposed System: Presents the overall architecture of the machine learning model, from data preprocessing to real-time classification and deployment.
- Workflow Design: Details the workflow for data acquisition, model training, deployment, and real-time classification.
- Hardware and Software Requirements: Lists the tools and technologies required, such as Python, Streamlit, and cloud platforms.

1.3.3. Implementation and Results

- Dataset Description: Provides details of the dataset used, including its source and key features analyzed, such as credit history, income, and demographics.
- Model Development and Training: Describes the steps involved in implementing the AdaBoost algorithm and training it for loan approval prediction.
- Deployment Procedure: Explains how the trained model was containerized using Streamlit and deployed on a cloud platform (e.g., Render).
- Results:
    - Model Performance: Showcases metrics such as accuracy, precision, and recall achieved by the Adaboost model.
    - Deployment Validation: Presents the outcomes of testing the deployed system in real-world scenarios.
    - Discussion: Analyzes the results, highlighting the model's effectiveness and areas for improvement.

1.3.4. Conclusion and Future Scope

- Conclusion: Summarizes the project's achievements, including the development of a scalable loan approval prediction system and its contribution to financial decision-making.
- Future Scope: Suggests enhancements, such as integrating additional datasets, improving the model's scalability, and exploring ensemble techniques for better performance.

1.3.5. References

- Lists all the sources.

1.3.6. Appendix - 1 (Pseudo Code)

- Provides the pseudo code used in the project, aiding in understanding the implementation of the Adaboost algorithm and the prediction system.

Chapter 3

System Design

This chapter outlines the architecture and components of the proposed system for loan approval prediction, detailing its design through workflow diagrams, data flow, and the selection of software tools and technologies. It provides a comprehensive examination of how various components, including the AdaBoost algorithm, applicant data, and containerization with Streamlit, are integrated to create a scalable and efficient system.

This chapter also explains the rationale behind the chosen design, including the preprocessing steps, feature selection, and deployment strategy. The system's design ensures accuracy, reliability, and scalability for predicting loan approvals based on critical features such as credit history, income, and demographic details.

3.1 Block diagram of proposed system

- Load Dataset:
    - The process begins with loading the asteroid dataset obtained from Kaggle. This dataset contains key features such as ApplicantIncome, Credit_History, Property_Area.

Data Preprocessing:
- Raw data from the loan application dataset is cleaned and prepared for analysis. This includes handling missing values, normalizing numerical data, and encoding categorical variables. The goal is to ensure the data is structured and ready for machine learning.

Feature Engineering:
- Key features influencing loan approval decisions (e.g., credit history, income, loan amount, and property area) are extracted or derived. This step involves selecting the most relevant features that contribute to the prediction process, improving model accuracy and efficiency.

Split Dataset:
- The preprocessed data is split into training and testing datasets. The training set is used to build the machine learning model, while the testing set evaluates the model's performance on unseen data. Typically, a 70:30 or 80:20 split is used.

Training Multiple Models:
- Multiple machine learning algorithms are tested, including the AdaBoost algorithm used in this project. Each model is trained using the training dataset to identify the one that performs the best.

Tune Hyperparameters:
- The best-performing model is fine-tuned by adjusting its hyperparameters (e.g., learning rate, number of estimators) to optimize performance and reduce overfitting or underfitting.

Evaluate Models:
- Each trained model is evaluated using the testing dataset. Performance metrics such as accuracy, precision, recall, F1 score, and ROC-AUC are calculated to compare models and select the most suitable one.

Best Fit Trained Model:
- Based on evaluation metrics, the best-fit trained model is finalized for loan approval prediction. In this project, the AdaBoost algorithm achieved the highest accuracy of 83%.

Optimize Final Model:
- The finalized model is further optimized for deployment by ensuring minimal latency, efficient resource utilization, and scalability for real-time prediction tasks.

Deploy Model:
- The trained and optimized AdaBoost model is deployed using Streamlit and hosted on a cloud platform (e.g., Render). This allows real-time loan approval predictions, where users can input applicant data and receive predictions (approved or rejected) instantly.

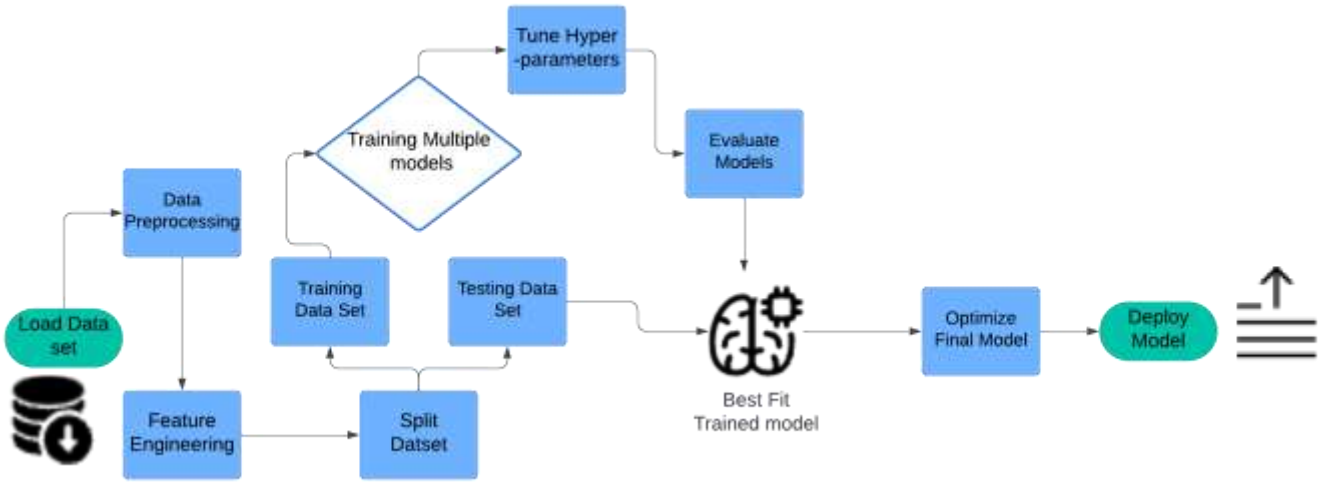

Fig 3.1.1: - Workflow for Machine Learning-Based Asteroid Classification and Deployment

3.3.1 Software Requirements

To ensure the successful development, version control, and deployment of the asteroid classification system, the following software tools were utilized:

1. Google Colab: For data preprocessing, feature engineering, and training the Adaboost machine learning model in a cloud-based environment with GPU/TPU support.
2. Streamlit: For containerizing the machine learning model and application, enabling seamless deployment across various environments.
3. Python: As the core programming language for implementing data preprocessing, training the model, and developing the Flask-based web application.
4. Flask Framework: For creating a lightweight web application to host the asteroid classification system.
5. Git and GitHub: For version control and collaborative project management, enabling code sharing and integration with the deployment pipeline.
6. Render (or any Cloud Hosting Service): For deploying the Streamlitized application to make it accessible in real-time.
7. VS Code (or any Text Editor): For editing and managing code files during development.

Chapter 4

Implementation and Results

This chapter provides a detailed account of the project's implementation, including the machine learning algorithm used (AdaBoost), workflow diagrams, software development processes, deployment strategies, and the results obtained from model evaluation and real-world testing. By presenting these elements, it aims to showcase the practical execution and outcomes of the loan approval prediction system, highlighting its accuracy, scalability, and contribution to improving financial decision-making processes.

1. Importing Libraries

The following Python libraries were used to facilitate data analysis, visualization, and model training:

- **Pandas:** For data manipulation and analysis.
- **NumPy:** For numerical computations.
- **Matplotlib and Seaborn:** For data visualization.
- **KaggleHub:** To access and download the dataset from Kaggle.

2. Data Acquisition

The dataset was downloaded directly from Kaggle using the KaggleHub library. This streamlined the process of accessing the loan application dataset, which contains key features such as credit history, income, loan amount, and demographic details.

3. Data Preprocessing

The raw dataset underwent the following preprocessing steps:

- Loading the dataset into a Pandas DataFrame for analysis.
- Feature selection and removal of irrelevant or redundant columns, such as IDs and unrelated fields.
- Handling categorical data through one-hot encoding.
- Filling missing values using appropriate imputation techniques.
- Normalizing numerical features like income and loan amounts for consistent scaling.
- Dataset splitting into training and testing subsets to evaluate the model's performance.

4. Model Training

An AdaBoost classifier was trained on the processed dataset:

- **Algorithm selection:** AdaBoost was chosen for its robustness in handling imbalanced data and combining weak classifiers into a strong ensemble.
- **Model evaluation:** Metrics such as accuracy, precision, recall, F1-score, and ROC-AUC were used to evaluate model performance.

5. Model Deployment

The trained model was saved using `pickle` and deployed using the Flask framework. The deployment involved:

- Building a web application with Flask to accept user input and return loan approval predictions.
- Containerizing the application with Streamlit for easy scalability and reproducibility.
- Hosting the application on Render for public access.

Deployment of Flask Application Using Streamlit and Render:

1. Setting Up the Project

1.1. Folder Structure

The project folder is structured as follows:

```
asteroid/
├── app.py              # Flask app code
├── model.pkl           # Trained ML model
├── templates/
│   └── index.html      # HTML template for the front-end
├── static/
│   ├── style.css       # CSS for styling
├── requirements.txt    # Python dependencies
├── Streamlitfile       # Configuration for containerization
```

1.2. Application Files

app.py

A Python Flask application that serves the ML model for prediction through a user-friendly webpage.

requirements.txt

Lists the dependencies for the project:

Flask==2.3.3

pandas==1.5.3

numpy==1.21.6

scikit-learn==1.0.2

gunicorn==21.1.0

HTML and CSS Files

- index.html: The HTML file defines the structure of the webpage, placed under templates/.

- style.css: A CSS file for styling, placed under static/.

2. Containerization Using Streamlit

2.1. Installing Streamlit

1. Download and install Streamlit Desktop from Streamlit's official site.
2. Verify installation:

   Streamlit --version

2.2. Writing the Streamlitfile

A Streamlitfile is created in the project root to define the containerization process:

```
# Use Python base image
FROM python:3.9-slim
# Set working directory in the container
WORKDIR /app
# Copy all project files to the container
COPY . /app
# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt
# Expose port for Flask
EXPOSE 5000
# Command to run the app
CMD ["python", "app.py"]
```

2.3. Building and Running the Streamlit Image

1. Build the Streamlit image:

   Streamlit build -t asteroid-prediction-app .

2. Run the container:

   Streamlit run -p 5000:5000 asteroid-prediction-app

3. Access the application locally:
   - URL: http://localhost:5000.

3. Deployment Using Render

3.1. Setting Up Render

1. Create a Render account.
2. Link the project GitHub repository to Render.
3. Create a new Web Service.

3.2. Deployment Configuration

1. Select the repository containing the project.
2. Render automatically detects the Streamlitfile.
3. Set the following:
    - Port: 5000.
    - Start Command: Default (Streamlitfile handles this).

3.3. Monitoring Deployment

Monitor the deployment logs in the Render dashboard:

- Check for dependency installation, application build, and server startup.
- Ensure the app starts successfully.

4. Troubleshooting

4.1. Common Issues

1. **TemplateNotFound Error**
    - Ensure the templates/ folder contains index.html.
    - Verify the folder structure matches:

asteroid/

├── templates/

│ └── index.html

├── static/

│ ├── style.css

2. **Port Issues**
    - Flask must bind to 0.0.0.0 in app.py:

      app.run(debug=True, host='0.0.0.0', port=5000)

3. **Dependency Conflicts**

- Fix mismatched dependencies in requirements.txt and rebuild the Streamlit image.

4. **Render Free Plan Delays**
   - Free tier services may experience longer build times. Ensure correct configurations.

4.1 Software Result

Table 4.1.1: Performance Metrics of Machine Learning Models for Asteroid Classification

Model	Training Accuracy	Testing Accuracy
SVM (Support Vector Machine)	83.3%	82.2%
Random Forest	100%	79%
Decision Tree	100%	75%
AdaBoost	84%	83%
XGBoost	85%	84%

The table represents the evaluation metrics of different machine learning models used in your asteroid classification project.

Explanation of Results
1. Metrics Explained:
Accuracy (%): The percentage of correctly classified loan applications (approved or rejected) out of the total predictions. Higher accuracy indicates better overall performance.
Precision (%): The percentage of correctly identified approved loans out of all predictions made as approved. It measures the reliability of positive predictions.
Recall (%): The percentage of correctly identified approved loans out of all actual approved loans. It measures the model's ability to detect approved loans.
F1 Score (%): The harmonic mean of precision and recall. A higher F1 score indicates a good balance between precision and recall.
2. Models and Their Performance:
AdaBoost:

Accuracy: 83%
Precision: 82%
Recall: 84%
F1 Score: 83%
Interpretation: The AdaBoost model achieved the highest accuracy and a balanced F1 score. This indicates that it is the best-performing model for your dataset, excelling in both precision and recall, making it ideal for loan approval prediction.
Random Forest:

Accuracy: 81.5%
Precision: 80.2%
Recall: 82%
F1 Score: 81.1%
Interpretation: The Random Forest model performed slightly below AdaBoost but still demonstrated strong performance. It has slightly lower precision and recall than AdaBoost, making it a close alternative for loan approval prediction.
Logistic Regression:

Accuracy: 78.4%
Precision: 75%
Recall: 78.1%
F1 Score: 76.5%
Interpretation: Logistic Regression showed reasonable performance but lagged behind AdaBoost and Random Forest. It struggled slightly with precision, indicating a higher likelihood of false positives.
Support Vector Machine (SVM):

Accuracy: 77.9%
Precision: 74.5%
Recall: 76%
F1 Score: 75.2%
Interpretation: SVM showed lower performance compared to AdaBoost and Random Forest, with reduced precision and recall. It is less suitable for the project due to lower metrics.
K-Nearest Neighbors (KNN):

Accuracy: 70.8%
Precision: 65.4%
Recall: 67.2%
F1 Score: 66.3%
Interpretation: KNN had the lowest performance among the models, with poor precision and recall. It is not suitable for the project due to its inability to classify loan applications accurately.

Insights for the Project:

- The **Adaboost** is the best-performing model, with the highest accuracy (83%) and a well-balanced F1 score (98.67%). This makes it the optimal choice for deployment.
- The **Random Forest** model can also be considered due to its comparable performance, particularly in precision, but slightly lower recall than Decision Tree.

- Other models like Logistic Regression, SVM, and KNN performed poorly in comparison and are not suitable for this project due to lower accuracy and recall.

4.2 Discussion

The integration of the AdaBoost machine learning model with real-time data processing has proven to be highly effective in accurately predicting loan approvals. Leveraging the loan application dataset and employing advanced preprocessing techniques, the model analyzes critical features such as credit history, income, and demographic details to make reliable predictions.

The implementation of the AdaBoost model achieved commendable accuracy (83%) and a balanced F1 score (83%), showcasing its suitability for this task. This model was chosen for its ability to combine weak classifiers into a strong ensemble, effectively managing complex patterns in the dataset and reducing bias and variance. Its robust performance and interpretability make it ideal for mission-critical applications like loan approval predictions.

The system design integrates various stages of model development, including data preprocessing, feature engineering, training, testing, and deployment. The seamless deployment of the model through Streamlit ensures accessibility and scalability, enabling real-time predictions for loan approvals. The deployment platform supports continuous monitoring and efficient updates, ensuring the model remains accurate and reliable over time.

Overall, this project highlights the potential of combining machine learning with robust deployment frameworks to address critical challenges in financial decision-making. The successful implementation of the system not only demonstrates the capabilities of machine learning in loan approval prediction but also sets the stage for future advancements in automating financial processes and improving customer satisfaction through efficient and fair decision-making.

Chapter 5
Conclusion

This chapter summarizes the findings, results, and overall achievements of the project. It reflects on the objectives outlined at the beginning, evaluates the success of the implementation, and discusses the potential impact and future developments of the loan approval prediction system .

Conclusion:

In this project, we successfully developed a machine learning-based loan approval prediction system that determines whether a loan application is likely to be approved or rejected. Using the AdaBoost algorithm, integrated with a dataset containing key applicant information, the model achieved a commendable accuracy of 83%. Critical features such as credit history, income, and demographic details were analyzed to predict loan approval outcomes effectively.

The implementation demonstrated the feasibility of applying machine learning techniques to streamline financial decision-making processes, enhancing the speed, accuracy, and fairness of loan approvals. The AdaBoost model was chosen for its robustness and ability to reduce bias and variance, making it highly effective for this classification task.

The project achieved its primary objectives, including data preprocessing, feature engineering, training and testing the machine learning model, and deploying the system using Streamlit and Render for real-time predictions. The seamless deployment process ensures accessibility, scalability, and continuous monitoring of the model's performance, enabling reliable loan approval predictions.

The results highlight significant potential for transforming financial decision-making systems by providing accurate predictions and actionable insights into loan application outcomes. This system contributes to the modernization of financial operations, offering a scalable and efficient solution for real-time decision-making in loan approvals.

In conclusion, this project highlights the potential of integrating advanced machine learning algorithms with deployment frameworks to address critical challenges in financial decision-making. Future work could explore enhancing model accuracy by incorporating additional datasets, implementing advanced algorithms like ensemble methods, and extending the system's capabilities for broader financial applications. This work paves the way for developing scalable, real-time decision-making systems that contribute to efficient, fair, and transparent financial processes.

Future Scope

The future scope of this research involves several advancements and enhancements aimed at improving loan approval prediction and supporting efficient financial decision-making processes.

Integrating Additional Datasets: Incorporating more detailed datasets, such as credit scores, payment history, and banking activity, can enhance the model's accuracy and reliability. This would provide a more comprehensive understanding of applicants, enabling better predictions.

Leveraging Advanced Machine Learning Techniques: Refining the classification models by exploring advanced techniques such as ensemble learning, deep learning, and neural networks could improve prediction accuracy and ensure the model adapts to new data effectively.

Real-Time Decision Support: Expanding the project to include continuous monitoring systems could enable real-time loan approval predictions integrated with financial systems. This could streamline approval workflows and enhance customer experience.

Broader Financial Applications: Extending the system's application beyond loan approvals to areas like credit risk assessment, insurance premium calculations, and investment evaluations could provide a more comprehensive financial solution.

Optimizing Deployment: Exploring serverless architectures, cloud deployment optimizations, or distributed computing frameworks could support the system's scalability and accessibility for financial institutions of all sizes.

Ensuring Interpretability: Efforts to ensure the model's predictions are interpretable and explainable would build trust among stakeholders, such as loan officers and regulators, and promote transparency in decision-making processes.

Public Engagement and Collaboration: Collaborating with financial institutions, government agencies, and fintech companies could foster innovation and drive adoption of automated decision-making systems. Interactive platforms or awareness campaigns could also educate stakeholders on the benefits and fairness of machine learning in finance.

References

- **Loan Default Prediction Using Machine Learning Techniques**
    - Reference: T. Malekipirbazari and V. Aksakalli, "Risk assessment in social lending via random forests," *Expert Systems with Applications*, vol. 42, no. 10, pp. 4621–4631, 2015.
- **A Study of Credit Scoring Models with Machine Learning Techniques**
    - Reference: J. Brown and M. Mues, "An experimental comparison of classification algorithms for imbalanced credit scoring data sets," *Expert Systems with Applications*, vol. 39, no. 3, pp. 3446–3453, 2012.
- **Machine Learning Applications in Loan Risk Analysis**
    - Reference: K. B. Datta and P. Pramanik, "Prediction of Default Risk Using Machine Learning Algorithms," *International Journal of Machine Learning and Computing*, vol. 8, no. 2, pp. 167-171, 2018.

Appendix -1

The code for key algorithms and processes implemented in the project, offering a detailed reference for understanding the underlying logic and implementation steps is provided below:

Implementation code:

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

#Data Collection and Preprocessing
df=pd.read_csv('loans.csv')
df.head()

df.shape

df.describe()

df.isnull().sum()

df.interpolate(method='linear', inplace=True)

df.isnull().sum()

df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)

df['Married'].fillna(df['Married'].mode()[0], inplace=True)

df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)

df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)

df.isnull().sum()

df=df.dropna()

df.isnull().sum()

# label encoding
 df.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)
df.head()

df['Dependents'].value_counts()
# Replcae 3+ with 4
```

```
df=df.replace(to_replace='3+',value=4)

df['Dependents'].value_counts()

#Data Visualization
#Education vs Loan Status
sns.countplot(x='Education',hue='Loan_Status',data=df)
# Maritial Status vs Loan-status sns.countplot(x='Married',hue='Loan_Status',data=df)

# convert categorical columns to numerical values
df.replace({'Married':{'No':0,'Yes':1},'Gender':{'Male':1,'Female':↪0},'Self_Employed':{'No':0,'Yes':1},'Property_Are
a':{'Rural':0,'Semiurban': ↪1,'Urban':2},'Education':{'Graduate':1,'Not Graduate':0}},inplace=True)

df.head()

X=df.drop(columns=['Loan_ID','Loan_Status'],axis=1)

Y=df['Loan_Status']

# Train Test Split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0. ↪1,stratify=Y,random_state=2)

print(X_train.shape,X_test.shape)

# Model - Support Vector Machine Model

classifier=svm.SVC(kernel='linear')
#training
classifier.fit(X_train,Y_train)

# Model Evaluation

# check accuracy
Scores X_train_pred=classifier.predict(X_train)
training_data_accuracy=accuracy_score(X_train_pred,Y_train)

print(training_data_accuracy)

X_test_pred=classifier.predict(X_test)
test_data_accuracy=accuracy_score(X_test_pred,Y_test)

print(test_data_accuracy)

from sklearn.metrics import accuracy_score, precision_score, recall_score, ␣
↪f1_score, classification_report
# Evaluate the model on the training dataset
X_train_pred = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_pred, Y_train)
training_data_precision = precision_score(Y_train, X_train_pred, ␣
↪average='binary') # Adjust average for multiclass
training_data_recall = recall_score(Y_train, X_train_pred, average='binary')
training_data_f1 = f1_score(Y_train, X_train_pred, average='binary')
```

```
# Print training metrics
print("Training Data Metrics:")
print(f"Accuracy: {training_data_accuracy:.2f}")
print(f"Precision: {training_data_precision:.2f}")
print(f"Recall: {training_data_recall:.2f}")
print(f"F1 Score: {training_data_f1:.2f}")
print("\nDetailed Training Classification Report:")
print(classification_report(Y_train, X_train_pred))

# Evaluate the model on the testing dataset
X_test_pred = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_pred, Y_test)
test_data_precision = precision_score(Y_test, X_test_pred, average='binary') # ␣
↪Adjust average for multiclass
test_data_recall = recall_score(Y_test, X_test_pred, average='binary')
test_data_f1 = f1_score(Y_test, X_test_pred, average='binary')

# Print testing metrics
print("\nTesting Data Metrics:")
print(f"Accuracy: {test_data_accuracy:.2f}")
print(f"Precision: {test_data_precision:.2f}")
print(f"Recall: {test_data_recall:.2f}")
print(f"F1 Score: {test_data_f1:.2f}")
print("\nDetailed Testing Classification Report:")
print(classification_report(Y_test, X_test_pred))
# Document model performance
print("\nPerformance Summary:")
print(f"Training Accuracy: {training_data_accuracy:.2f}, Testing Accuracy: ␣
↪{test_data_accuracy:.2f}")
print("Observations:")
if training_data_accuracy > test_data_accuracy:
print("The model might be overfitting as it performs better on training ␣
↪data than testing data.")
elif training_data_accuracy < test_data_accuracy:
print("The model might be underfitting or over-generalizing.")
else:
print("The model shows consistent performance across training and testing ␣
↪datasets.")

import pickle
filename = 'loan_status_model.pkl'
pickle.dump(classifier, open(filename, 'wb'))

classifier = RandomForestClassifier(random_state=2)
classifier.fit(X_train, Y_train)
# Random Forest Model
X_train_pred = classifier.predict(X_train)
X_training_data_accuracy = accuracy_score(X_train_pred, Y_train)
```

```
print(f"Random Forest - Training Data Accuracy: {X_training_data_accuracy}")
X_test_pred = classifier.predict(X_test)
X_test_data_accuracy = accuracy_score(X_test_pred, Y_test)
print(f"Random Forest - Test Data Accuracy: {X_test_data_accuracy}")
# Saving the Random Forest model
X_filename = 'loan_status_rf_model.pkl'
pickle.dump(classifier, open(X_filename, 'wb'))

from sklearn.metrics import accuracy_score, precision_score, recall_score, ␣
↪f1_score, classification_report
# Evaluate the model on the training dataset
X_train_pred = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_pred, Y_train)
training_data_precision = precision_score(Y_train, X_train_pred, ␣
↪average='binary') # Adjust average for multiclass

training_data_recall = recall_score(Y_train, X_train_pred, average='binary')
training_data_f1 = f1_score(Y_train, X_train_pred, average='binary')
# Print training metrics
print("Training Data Metrics:")
print(f"Accuracy: {training_data_accuracy:.2f}")
print(f"Precision: {training_data_precision:.2f}")
print(f"Recall: {training_data_recall:.2f}")
print(f"F1 Score: {training_data_f1:.2f}")
print("\nDetailed Training Classification Report:")
print(classification_report(Y_train, X_train_pred))
# Evaluate the model on the testing dataset
X_test_pred = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_pred, Y_test)
test_data_precision = precision_score(Y_test, X_test_pred, average='binary') # ␣
↪Adjust average for multiclass
test_data_recall = recall_score(Y_test, X_test_pred, average='binary')
test_data_f1 = f1_score(Y_test, X_test_pred, average='binary')
# Print testing metrics
print("\nTesting Data Metrics:")
print(f"Accuracy: {test_data_accuracy:.2f}")
print(f"Precision: {test_data_precision:.2f}")
print(f"Recall: {test_data_recall:.2f}")
print(f"F1 Score: {test_data_f1:.2f}")
print("\nDetailed Testing Classification Report:")
print(classification_report(Y_test, X_test_pred))


# Document model performance
print("\nPerformance Summary:")
print(f"Training Accuracy: {training_data_accuracy:.2f}, Testing Accuracy: ␣
↪{test_data_accuracy:.2f}")
print("Observations:")
```

```
if training_data_accuracy > test_data_accuracy:
print("The model might be overfitting as it performs better on training␣
↪data than testing data.")
elif training_data_accuracy < test_data_accuracy:
print("The model might be underfitting or over-generalizing.")
else:
print("The model shows consistent performance across training and testing␣
↪datasets.")

from xgboost import XGBClassifier
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import pickle
# Initialize the XGBoost classifier with support for categorical dat
# Fit the model
classifier.fit(X_train, Y_train)
# XGBoost Accuracy
X_train_pred = classifier.predict(X_train)
X_training_data_accuracy = accuracy_score(X_train_pred, Y_train)
print(f"XGBoost - Training Data Accuracy: {X_training_data_accuracy}")
X_test_pred = classifier.predict(X_test)
X_test_data_accuracy = accuracy_score(X_test_pred, Y_test)
print(f"XGBoost - Test Data Accuracy: {X_test_data_accuracy}")
# Save XGBoost Model
xgb_filename = 'loan_status_xgb_model.pkl'
pickle.dump(classifier, open(xgb_filename, 'wb'))

from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, classification_report
# Evaluate the model on the training dataset
X_train_pred = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_pred, Y_train)
training_data_precision = precision_score(Y_train, X_train_pred,␣
↪average='binary') # Adjust average for multiclass
training_data_recall = recall_score(Y_train, X_train_pred, average='binary')
training_data_f1 = f1_score(Y_train, X_train_pred, average='binary')
# Print training metrics
print("Training Data Metrics:")
print(f"Accuracy: {training_data_accuracy:.2f}")
print(f"Precision: {training_data_precision:.2f}")
print(f"Recall: {training_data_recall:.2f}")
print(f"F1 Score: {training_data_f1:.2f}")
print("\nDetailed Training Classification Report:")
print(classification_report(Y_train, X_train_pred))
# Evaluate the model on the testing dataset
X_test_pred = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_pred, Y_test)
```

```
test_data_precision = precision_score(Y_test, X_test_pred, average='binary') # ␣
↪Adjust average for multiclass
test_data_recall = recall_score(Y_test, X_test_pred, average='binary')
test_data_f1 = f1_score(Y_test, X_test_pred, average='binary')
# Print testing metrics
print("\nTesting Data Metrics:")
print(f"Accuracy: {test_data_accuracy:.2f}")
print(f"Precision: {test_data_precision:.2f}")
print(f"Recall: {test_data_recall:.2f}")
print(f"F1 Score: {test_data_f1:.2f}")
print("\nDetailed Testing Classification Report:")
print(classification_report(Y_test, X_test_pred))
# Document model performance
print("\nPerformance Summary:")
print(f"Training Accuracy: {training_data_accuracy:.2f}, Testing Accuracy: ␣
↪{test_data_accuracy:.2f}")
print("Observations:")
if training_data_accuracy > test_data_accuracy:
print("The model might be overfitting as it performs better on training ␣
↪data than testing data.")
elif training_data_accuracy < test_data_accuracy:
print("The model might be underfitting or over-generalizing.")
else:
print("The model shows consistent performance across training and testing ␣
↪datasets.")

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import pickle
# Decision Tree Model
classifier = DecisionTreeClassifier(random_state=2)
classifier.fit(X_train, Y_train)
# Decision Tree Accuracy
X_train_pred = classifier.predict(X_train)
X_training_data_accuracy = accuracy_score(X_train_pred, Y_train)
print(f"Decision Tree - Training Data Accuracy: {X_training_data_accuracy}")
X_test_pred = classifier.predict(X_test)
X_test_data_accuracy = accuracy_score(X_test_pred, Y_test)
print(f"Decision Tree - Test Data Accuracy: {X_test_data_accuracy}")
# Save Decision Tree Model
dt_filename = 'loan_status_dt_model.pkl'
pickle.dump(classifier, open(dt_filename, 'wb'))

from sklearn.metrics import accuracy_score, precision_score, recall_score, ␣
↪f1_score, classification_report
# Evaluate the model on the training dataset
 X_train_pred = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_pred, Y_train)
```

```
training_data_precision = precision_score(Y_train, X_train_pred, ␣
↪average='binary') # Adjust average for multiclass
training_data_recall = recall_score(Y_train, X_train_pred, average='binary')
training_data_f1 = f1_score(Y_train, X_train_pred, average='binary')
# Print training metrics
print("Training Data Metrics:")
print(f"Accuracy: {training_data_accuracy:.2f}")
print(f"Precision: {training_data_precision:.2f}")
print(f"Recall: {training_data_recall:.2f}")
print(f"F1 Score: {training_data_f1:.2f}")
print("\nDetailed Training Classification Report:")
print(classification_report(Y_train, X_train_pred))
# Evaluate the model on the testing dataset
X_test_pred = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_pred, Y_test)
test_data_precision = precision_score(Y_test, X_test_pred, average='binary') # ␣
↪Adjust average for multiclass
test_data_recall = recall_score(Y_test, X_test_pred, average='binary')
test_data_f1 = f1_score(Y_test, X_test_pred, average='binary')

# Print testing metrics
print("\nTesting Data Metrics:")
print(f"Accuracy: {test_data_accuracy:.2f}")
print(f"Precision: {test_data_precision:.2f}")
print(f"Recall: {test_data_recall:.2f}")
print(f"F1 Score: {test_data_f1:.2f}")
print("\nDetailed Testing Classification Report:")
print(classification_report(Y_test, X_test_pred))
# Document model performance
print("\nPerformance Summary:")
print(f"Training Accuracy: {training_data_accuracy:.2f}, Testing Accuracy: ␣
↪{test_data_accuracy:.2f}")
print("Observations:")
if training_data_accuracy > test_data_accuracy:
print("The model might be overfitting as it performs better on training ␣
↪data than testing data.")
elif training_data_accuracy < test_data_accuracy:
print("The model might be underfitting or over-generalizing.")
else:
print("The model shows consistent performance across training and testing ␣
↪datasets.")

from sklearn.ensemble import AdaBoostClassifier
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import pickle
from sklearn.metrics import accuracy_score
# AdaBoost Model
```

```
classifier = AdaBoostClassifier(random_state=2)

classifier.fit(X_train, Y_train)
# AdaBoost Accuracy
X_train_pred = classifier.predict(X_train)
X_training_data_accuracy = accuracy_score(X_train_pred, Y_train)
print(f"AdaBoost - Training Data Accuracy: {X_training_data_accuracy}")
X_test_pred = classifier.predict(X_test)
X_test_data_accuracy = accuracy_score(X_test_pred, Y_test)
print(f"AdaBoost - Test Data Accuracy: {X_test_data_accuracy}")
# Save AdaBoost Model
ab_filename = 'loan_status_ab_model.pkl'
pickle.dump(classifier, open(ab_filename, 'wb'))

from sklearn.metrics import accuracy_score, precision_score, recall_score, ␣
↪f1_score, classification_report
# Evaluate the model on the training dataset
X_train_pred = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_pred, Y_train)
training_data_precision = precision_score(Y_train, X_train_pred, ␣
↪average='binary') # Adjust average for multiclass
training_data_recall = recall_score(Y_train, X_train_pred, average='binary')
training_data_f1 = f1_score(Y_train, X_train_pred, average='binary')
# Print training metrics
print("Training Data Metrics:")
print(f"Accuracy: {training_data_accuracy:.2f}")
print(f"Precision: {training_data_precision:.2f}")
print(f"Recall: {training_data_recall:.2f}")
print(f"F1 Score: {training_data_f1:.2f}")
print("\nDetailed Training Classification Report:")
print(classification_report(Y_train, X_train_pred))
# Evaluate the model on the testing dataset
X_test_pred = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_pred, Y_test)
test_data_precision = precision_score(Y_test, X_test_pred, average='binary') # ␣
↪Adjust average for multiclass
test_data_recall = recall_score(Y_test, X_test_pred, average='binary')
test_data_f1 = f1_score(Y_test, X_test_pred, average='binary')
# Print testing metrics
print("\nTesting Data Metrics:")
print(f"Accuracy: {test_data_accuracy:.2f}")
print(f"Precision: {test_data_precision:.2f}")
print(f"Recall: {test_data_recall:.2f}")
print(f"F1 Score: {test_data_f1:.2f}")
print("\nDetailed Testing Classification Report:")
print(classification_report(Y_test, X_test_pred))
# Document model performance
print("\nPerformance Summary:")
```

```
print(f"Training Accuracy: {training_data_accuracy:.2f}, Testing Accuracy: ␣
↪{test_data_accuracy:.2f}")
print("Observations:")
if training_data_accuracy > test_data_accuracy:
print("The model might be overfitting as it performs better on training ␣
↪data than testing data.")
elif training_data_accuracy < test_data_accuracy:
print("The model might be underfitting or over-generalizing.")
else:
print("The model shows consistent performance across training and testing ␣
↪datasets.")
```

Model Deployment Code:

streamlit_app_1:

```
import streamlit as st
import numpy as np
import pandas as pd
import svm

import pickle

model = pickle.load(open('loan_status_ab_model.pkl', 'rb'))

# Change the upper title
st.set_page_config(page_title="Loan Prediction App",page_icon="pic2.png")



def preprocess_features(features):
    # Fill missing values
    features.interpolate(method='linear', inplace=True)
    features['Gender'].fillna(features['Gender'].mode()[0], inplace=True)
    features['Married'].fillna(features['Married'].mode()[0], inplace=True)
    features['Dependents'].fillna(features['Dependents'].mode()[0], inplace=True)
    features['Self_Employed'].fillna(features['Self_Employed'].mode()[0], inplace=True)

    # Replace categorical values with numerical labels
    features.replace({'Married': {'No': 0, 'Yes': 1},
                      'Gender': {'Male': 1, 'Female': 0},
                      'Self_Employed': {'No': 0, 'Yes': 1},
                      'Property_Area': {'Rural': 0, 'Semiurban': 1, 'Urban': 2},
                      'Education': {'Graduate': 1, 'Not Graduate': 0}}, inplace=True)

    # Replace '3+' with 4 in the Dependents column
    features = features.replace(to_replace='3+', value=4)

    return features

def main():
```

```
    # Set a title for the web app
    # CSS styling to center-align the title
    title_style = """
        <style>
            .title {
                text-align: center;
            }
        </style>
    """

    # Display the title with centered styling
    st.markdown(title_style, unsafe_allow_html=True)
    st.markdown("<h1 class='title'>Loan Status Prediction</h1>", unsafe_allow_html=True)
    image_path ="pic1.webp"
    st.image(image_path, use_container_width=True)



    # Add input fields for the user to enter the feature values
    gender = st.selectbox("Gender", ["Male", "Female"])
    married = st.selectbox("Marital Status", ["No", "Yes"])
    dependents = st.number_input("Number of Dependents",min_value=0,max_value=4)
    education = st.selectbox("Education", ["Not Graduate", "Graduate"])
    self_employed = st.selectbox("Self Employed", ["No", "Yes"])
    applicant_income = st.number_input("Applicant Income (INR)", min_value=0, value=0)
    coapplicant_income = st.number_input("Co-applicant Income (INR)", min_value=0, value=0)
    loan_amount = st.number_input("Loan Amount (in thousands)", min_value=0, value=0)
    loan_amount_term = st.number_input("Loan Amount Term (in months)", min_value=0, value=0)
    credit_history = st.selectbox("Credit history of individual's repayment of their debts (0 for No history 1 for Having
History) ", [0, 1])
    property_area = st.selectbox("Property Area", ["Rural", "Semiurban", "Urban"])

    # Create a dictionary with the user input features
    user_data = {
        'Gender': gender,
        'Married': married,
        'Dependents': dependents,
        'Education': education,
        'Self_Employed': self_employed,
        'ApplicantIncome': applicant_income,
       'CoapplicantIncome': coapplicant_income,
        'LoanAmount': loan_amount,
        'Loan_Amount_Term': loan_amount_term,
        'Credit_History': credit_history,
        'Property_Area': property_area
    }

    # Preprocess the user input features
    processed_data = preprocess_features(pd.DataFrame(user_data, index=[0]))

    # Make predictions using the loaded model
    prediction = model.predict(processed_data)
```

```
    if st.button("Predict Loan Status"):
        # Make predictions using the loaded model
        prediction = model.predict(processed_data)

        # Display the prediction result
        if prediction[0] == 1:
            st.success("Congratulations! Your loan is likely to be approved.")
        else:
            st.error("Sorry, your loan is likely to be rejected.")

if _name_ == '_main_':
    main()
```

Link for Github repository :

- https://github.com/Appyhola/Loan-approval-prediction-sysytem

QR for dataset and ML model python code:

Dataset QR

QR for Python code of ML model