LoanDefaulter - Prediction

1. Overview

LoanWise is an intelligent web application designed to predict the probability of a loan applicant defaulting on their payment. It provides a simple, user-friendly interface for loan officers or financial analysts to input an applicant's financial and personal data and receive an instant risk assessment. The application leverages a machine learning model trained on historical credit data to generate a binary "High/Low Risk" classification, a precise default probability score, and a real-time visual risk gauge.

2. Introduction

In the financial industry, accurately assessing the risk associated with lending is crucial for maintaining profitability and stability. Manually evaluating loan applications can be time-consuming and subjective. The LoanWise project was initiated to automate and standardize this process by using a data-driven approach. By building a predictive model and embedding it in an accessible web tool, LoanWise aims to provide quick, consistent, and easy-to-understand credit risk assessments, enabling better decision-making for lenders.

3. Features

The application has been developed with a focus on usability and providing clear, actionable insights.

1. Predictive Form: An intuitive web form for entering 10 key applicant attributes, including personal information (age, dependents, income) and credit history (late payments, open credit lines).
2. Dual Risk Assessment: Upon submission, the tool provides two forms of output: A clear, binary classification (High Risk / Low Risk). B. precise Default Probability Score (e.g., 41.66%) for more nuanced analysis.
3. Real-Time Risk Gauge: An interactive speedometer-style gauge that provides instant, visual feedback on the default probability. It updates automatically when the "Calculate Risk" button is clicked, showing how the risk level changes based on the provided inputs.

4. System Architecture

The project follows a standard client-server architecture:-

1. Frontend (Client-Side): A single-page web interface built with HTML, CSS, and JavaScript. It's responsible for rendering the form, capturing user input, and displaying the results and real-time gauge without needing a full page reload (using AJAX).

2. Backend (Server-Side): A web server built using the Flask framework in Python. It serves the frontend page and exposes two main API endpoints:

a. * '/' renders the initial web page.

b. * '/predict' receives applicant data in JSON format, processes it using the machine learning model, and returns the prediction results as a JSON object.

3. Machine Learning Model: A pre-trained AdaBoost classifier (finalized_model.sav) is loaded into memory by the Flask application at startup. This ensures that prediction requests are handled quickly without the need to reload the model each time.

5. Technology Stack

Backend:

1. Language: Python 3.11.6

2. Framework: Flask

3. ML Model Serving: Pickle (for loading the .sav file)

- Machine Learning:

* Core Libraries: Scikit-learn, Pandas, NumPy

* Development Environment: Jupyter Notebook

- Frontend:

1. Structure: HTML5

2. Styling: CSS3

3. Interactivity: JavaScript (using the fetch API for AJAX calls)

Dependencies: Key libraries are listed in requirements.txt, including Flask, scikit-learn, pandas, and numpy.

6. Machine Learning Model

The predictive engine of the application is an AdaBoost Classifier model, which was selected after a thorough development process documented in the loan.ipynb notebook.

- Dataset: The model was trained on the cs-training.csv dataset, which contains historical loan data.

- Data Preprocessing: The initial data was cleaned by handling missing values (imputing with the median) and standardizing column names.

- Model Selection: Several models were evaluated, including K-Nearest Neighbors (KNN) and Logistic Regression. The AdaBoost Classifier was chosen for its superior performance, achieving an accuracy of 93.5% and an AUC score of 0.86 on the test set.

- Serialization: The final trained model was serialized into the finalized_model.sav file using pickle for easy loading in the Flask application.

7. Implementation Details

The core of the application is the app.py script. On startup, it loads the trained AdaBoost model. The main web page (predict.html) contains the form and the JavaScript logic. When the user clicks "Calculate Risk", an onsubmit event in the form triggers a JavaScript function. This function prevents the default page reload, gathers all form data, sends it to the /predict endpoint on the Flask server, and waits for a JSON response. Once the response is received, the JavaScript dynamically updates the content of the main result block and the real-time risk gauge.

8. Visualization

The primary visualization is the Real-Time Risk Gauge. This is a semi-circular gauge created with HTML and CSS, with a needle animated by JavaScript.

- Functionality: The needle's position and the percentage text are directly tied to the "probability" value returned by the /predict API endpoint.

- Design: The gauge features a color gradient from green (low risk) to yellow (moderate risk) to red (high risk), providing an immediate visual cue of the risk level. The percentage text below it changes color to match the corresponding risk zone.

9. Conclusion and Future Work

The LoanWise project successfully provides a functional and interactive tool for loan default prediction. It simplifies the risk assessment process and enhances it with real-time feedback.

Potential future enhancements could include:

- Explainable AI (XAI): Integrating libraries like SHAP or LIME to explain why a prediction was made by showing the top contributing factors.

- User Accounts: Adding a database and user authentication to allow loan officers to save and manage a history of their risk assessments.

- PDF Report Generation: Allowing users to download a formatted PDF summary of the prediction.

- API Endpoint: Creating a dedicated /api/predict endpoint for programmatic access, allowing other systems to use the model.

10. Help Taken From

This project was developed with the assistance of Gemini, an AI model from Google. Gemini provided help with:

- Analyzing and explaining the project's code and structure.

- Generating step-by-step instructions for setup and execution.

- Writing and refactoring Python (Flask) and JavaScript code for new features.

- Debugging issues, such as file path errors and JavaScript rendering problems.

- Brainstorming ideas for new features and visualizations.