IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

Loan Approval Prediction Using Machine Learning

P. Girish[1], G. Vamsi Narayan[2], Aashish Kumar[3], R. Tejendra[4], Dr. S. Anitha Jebamani[5]

Department of Computer Science and Engineering (Emerging Technologies)

SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[1-4]

Guide, Department of Computer Science and Engineering (Emerging Technologies)

SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[5]

Abstract: Loan approval assessment depends on applicant information such as income, requested loan amount, repayment period, credit history and available assets. Manual assessment can be time-consuming, while fixed rules may not represent interactions among these attributes. This paper presents a Loan Approval Prediction prototype that integrates a machine-learning backend with a browser-based interface. The frontend is implemented using React and Vite, and the backend uses Python and scikit-learn. The visible training code includes Logistic Regression, Support Vector Classifier (SVC) and Random Forest as candidate classifiers, together with preprocessing components and feature engineering. Two derived attributes are calculated: loan-to-income ratio and total asset value. A trained pipeline is persisted using Joblib and is accessed through a locally running backend service. The interface accepts eleven applicant attributes and displays a decision, a probability value and the model label. In the supplied demonstration, the display shows "Not Approved," "Approval probability: 69.5%," and "random_forest." This single example demonstrates application flow, not overall predictive accuracy. Since the available project evidence does not include verified dataset statistics or numerical test metrics, no accuracy or comparative-performance claim is made.

Keywords: Loan approval, machine learning, Random Forest, Logistic Regression, Support Vector Classifier, feature engineering, React, Vite, scikit-learn, CIBIL score.

I. INTRODUCTION

Loan services support needs such as education, housing, vehicles, business expansion and personal expenses. Before granting credit, a lender considers an applicant's ability and willingness to repay. Income, existing commitments, credit score, employment status, requested principal and repayment term can all influence this assessment.

Traditional assessment may involve document verification and rule-based checks. These remain important, but manual review can become slow when application volumes increase. Machine-learning classifiers can learn statistical relationships from historical applications and estimate the likely approval class for a new applicant.

This work implements a prototype named Loan Approval Predictor. Applicant details are entered through a React web form and transmitted to a Python backend. The backend loads a persisted scikit-learn pipeline, prepares the features and returns a prediction response. The browser displays the predicted decision, a probability value and the model name.

The contributions of this project are: (1) a web interface for eleven applicant attributes; (2) feature engineering for loan-to-income ratio and total assets; (3) a Python inference service using a persisted machine-learning pipeline; and (4) an end-to-end local demonstration connecting React/Vite with Python. The system is an academic decision-support prototype and is not a substitute for lender verification or human underwriting.

II. RELATED WORK

Loan approval is commonly formulated as supervised binary classification using historical records labelled as approved or rejected. Traditional scorecards and rule-based systems apply fixed conditions or points. They are straightforward to explain and audit, but may fail to capture non-linear relationships among multiple applicant attributes.

Logistic Regression estimates class probability through a logistic function and is widely used as an interpretable baseline [3]. Support Vector Machines identify a maximum-margin separating boundary; kernel functions can represent non-linear decision boundaries, although feature scaling and parameter selection are important [2].

Random Forest combines predictions from multiple decision trees trained on bootstrap samples and random feature subsets. The ensemble can reduce the variance of an individual tree and is applicable to mixed tabular data [1]. Scikit-learn provides common preprocessing and model-pipeline components that support consistent transformations during training and inference [4].

 This work is licensed under a Creative Commons Attribution 4.0 International License

The gap addressed here is practical integration: an academic prediction model should not stop at offline training, but should accept validated inputs through an interface, apply consistent feature transformations and communicate its output responsibly. Unlike studies that report unsupported headline accuracy, this paper distinguishes the captured interface demonstration from a formal quantitative evaluation.

III. METHODOLOGY

A. Dataset and Applicant Attributes

The project report identifies a CSV file named loan_approval_dataset.csv. The available evidence confirms the file's presence in the project folder but does not provide its complete contents, row count, target-column name or class distribution. These dataset properties are therefore not assumed.

Input attribute	Expected entry	Purpose
Number of dependents	Integer	People financially dependent on applicant
Education	Graduate / Not Graduate	Education category
Self-employed	Yes / No	Employment status
Annual income	Numeric	Applicant income
Loan amount	Numeric	Requested principal
Loan term	Integer (years)	Repayment period
CIBIL score	Integer (300–900)	Credit score
Residential assets	Numeric	Residential asset value
Commercial assets	Numeric	Commercial asset value
Luxury assets	Numeric	Luxury asset value
Bank assets	Numeric	Bank asset value

B. Preprocessing and Feature Engineering

The interface validates required fields, parses numeric values and restricts the categorical entries to the options shown in the form. The CIBIL input hint indicates a range of 300 to 900. The training code imports ColumnTransformer, OneHotEncoder, StandardScaler and Pipeline, indicating a preprocessing pipeline intended to apply the transformations consistently.

The visible add_features function derives two attributes. The loan-to-income ratio is calculated as:

loan_to_income = loan_amount / income_annum

Zero income is replaced with a missing value before division to avoid division by zero. The total-assets feature is calculated as:

total_assets = residential + commercial + luxury + bank assets

These features represent the requested loan relative to annual income and the combined value of the applicant's listed assets. The exact missing-value strategy and final model input schema require confirmation from the complete training environment.

C. Candidate Models and Training

The supplied training-code screenshot shows imports for RandomForestClassifier, LogisticRegression and SVC. It also shows GridSearchCV, StratifiedKFold and train_test_split imports, a fixed RANDOM_STATE of 42, and the setting INCLUDE_RANDOM_FOREST = True. The model artifact is named model.joblib. The captured browser output labels its model as random_forest. These observations confirm the visible implementation components, but do not establish the final hyperparameters, validation split, search result or comparative model scores.

The intended workflow is to load the CSV, inspect the applicant columns, create engineered features, encode categorical fields, scale numerical fields where required, train candidate classifiers, persist the selected pipeline and load it for inference. The same transformations should be applied to incoming requests to reduce training-serving mismatch.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IV. SYSTEM DESIGN AND IMPLEMENTATION

A. Architecture

The system follows a client-server design. A browser hosts the React/Vite frontend, which collects applicant information and displays the returned result. The frontend is configured to communicate with a Python backend at localhost:8000. The backend prepares the request, loads the saved Joblib model pipeline and returns the predicted decision and probability information. The project report describes the Vite development server at localhost:5173 and a Uvicorn-based backend runtime.



Fig. 1. Conceptual system architecture of the Loan Approval Predictor.

B. Prediction Workflow

The end-to-end process consists of loading the dataset, preparing the columns, deriving loan-to-income and total-assets features, applying preprocessing, training and saving a model, receiving applicant input through the frontend, applying the same transformations at inference time and returning the result to the browser.



Fig. 2. End-to-end loan approval prediction workflow.

 This work is licensed under a Creative Commons Attribution 4.0 International License

C. Tools and Application Modules

Technology / module	Role in the project
Python, Pandas, NumPy	Backend language, tabular data handling and numerical operations
Scikit-learn	Preprocessing, candidate classifiers and evaluation utilities
Joblib	Persistence and loading of the trained model pipeline
React and Vite	Browser interface and local frontend development server
Uvicorn	ASGI server used for the local backend runtime
CSV dataset	Storage of historical loan application records

The frontend form contains eleven fields and Predict and Clear actions. The backend project evidence lists train.py, main.py, structure.py, check.py, metrics.json, model.joblib, requirements.txt and loan_approval_dataset.csv. The available report documents local development; it does not establish public cloud deployment or production hosting.



Fig. 3. Loan Approval Predictor input form captured from the project.



Fig. 4. Backend training script and visible feature-engineering code.



Fig. 5. Vite development server running locally.

V. RESULTS AND DISCUSSION

The supplied browser screenshot demonstrates that the application accepts applicant inputs and displays a prediction card. The captured example shows a loan amount of 1,500,000, a loan term of 24 years and a CIBIL score of 760. The four displayed asset fields contain zero values. Other input values are outside the visible crop and are not inferred.



Fig. 6. Captured application output: Not Approved, 69.5% and random_forest.

Displayed output	Observed value
Predicted decision	Not Approved
Approval probability (as labelled in UI)	69.5%
Model label	random_forest
Loan amount visible	1,500,000
Loan term visible	24 years
CIBIL score visible	760
Four asset values visible	0 each

This result confirms a working prediction display for the captured example. It is not a measure of overall model accuracy. In particular, the relationship between the displayed "Not Approved" class and the UI text "Approval probability: 69.5%" should be checked against the backend's class ordering and probability mapping. The screenshot alone cannot determine whether 69.5% represents approval probability, predicted-class probability or another confidence value.

The training script imports accuracy_score, precision_score, recall_score, f1_score, confusion_matrix and roc_auc_score. However, numerical test results, a confusion matrix, class distribution and validation outputs were not included in the supplied project evidence. Consequently, this paper does not report invented metric values or claim that one candidate classifier outperforms another.

VI. LIMITATIONS

The principal limitations are: (1) the complete CSV dataset and schema were not available for inspection; (2) the number of records and target-class distribution are undocumented in the supplied evidence; (3) the final model-selection settings and validation scores are not shown; (4) only one prediction example is available; (5) the displayed probability label requires verification; and (6) the screenshots demonstrate local development rather than a secured production deployment. The prototype does not perform formal identity verification, regulatory compliance checks or human underwriting.

A loan model can affect applicants differently depending on the error type. False approvals and false rejections have different consequences, so a future evaluation should report the confusion matrix alongside precision, recall, F1-score and ROC-AUC. Fairness analysis and probability calibration should also be considered before any real-world use.

VII. CONCLUSION AND FUTURE WORK

This paper presented a Loan Approval Prediction prototype that combines a React/Vite web interface with a Python and scikit-learn backend. The visible implementation includes Logistic Regression, SVC and Random Forest candidate imports, preprocessing components, two engineered financial features and Joblib model persistence. The browser demonstration displays a decision, a probability value and the model label for a sample application.

The project establishes an end-to-end academic application workflow, but the available evidence is insufficient to quantify predictive performance. Future work should document the complete dataset and target distribution, run a reproducible train/test evaluation, compare candidate models under the same validation protocol, verify the probability mapping, report a confusion matrix and standard metrics, and add explainability, fairness, security and privacy controls before considering practical deployment.

 This work is licensed under a Creative Commons Attribution 4.0 International License

ACKNOWLEDGMENT

The authors express their sincere gratitude to Dr. S. Anitha Jebamani, Assistant Professor, Department of Computer Science and Engineering (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, for her guidance and support. The authors also acknowledge the support of the department and the institution.

REFERENCES

[1] L. Breiman, “Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
[2] C. Cortes and V. Vapnik, “Support-Vector Networks,” Machine Learning, vol. 20, no. 3, pp. 273–297, 1995.
[3] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, Applied Logistic Regression, 3rd ed. Hoboken, NJ, USA: Wiley, 2013.
[4] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[5] React Team, “React: The library for web and native user interfaces.” [Online]. Available: https://react.dev
[6] Vite Team, “Vite: Next Generation Frontend Tooling.” [Online]. Available: https://vite.dev
[7] Credit Information Bureau (India) Limited, “CIBIL Score.” [Online]. Available: https://www.cibil.com
[8] Joblib Developers, “Joblib: running Python functions as pipeline jobs.” [Online]. Available: https://joblib.readthedocs.io
[9] Uvicorn Contributors, “Uvicorn: An ASGI web server.” [Online]. Available: https://www.uvicorn.org
[10] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning, 2nd ed. New York, NY, USA: Springer, 2009.

 This work is licensed under a Creative Commons Attribution 4.0 International License