Dr. V. Subha, 2026, 14:6
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Credit Card Default Prediction Using Random Forest Classification

Dr. V. Subha[1]
Assistant Professor, School of Computing Sciences
Department of Applied Computing and Emerging Technologies
VISTAS, Pallavaram, Chennai[1]

K Soma Surya Kalyan[2], **MB Rahul Krish**[3]
BCA Data Science, VISTAS, Pallavaram, Chennai[2,3]

Abstract- Credit card default prediction is an important machine learning problem in financial risk assessment because identifying customers who may default can support informed credit-management decisions. This study presents a Random Forest-based binary classification approach using the UCI Default of Credit Card Clients dataset. The implementation was carried out in Python using pandas, NumPy, ucimlrepo, and scikit-learn. The data were combined into a single DataFrame, inspected, and cleaned by removing duplicate rows. After preprocessing, 29,965 records and 23 input features were used. The dataset was divided using an 80:20 train-test split with random_state=42 and stratified sampling. A Random Forest Classifier containing 100 trees was trained without hyperparameter tuning. Performance was evaluated using accuracy, weighted precision, weighted recall, weighted F1-score, classification report, and confusion matrix. The reported model results are 81.31% accuracy, 79.39% weighted precision, 81.31% weighted recall, and 79.30% weighted F1-score. Class-wise results show stronger performance for the No Default class than for the Default class, particularly in recall. The study therefore provides a baseline Random Forest implementation while demonstrating the difficulty of detecting the minority default class.

Keywords- Credit Card Default; Random Forest; Machine Learning; Binary Classification; Credit Risk; Predictive Analytics

I. INTRODUCTION

Credit cards are widely used financial instruments that provide customers with access to revolving credit. When customers fail to meet repayment obligations, they may enter default, creating financial risk for lending institutions. Reliable prediction of default status can therefore contribute to credit-risk assessment and early risk identification. Machine learning methods can learn patterns from historical customer and repayment information and use those patterns to classify previously unseen observations. Ensemble methods such as Random Forest are particularly suitable for tabular classification because they combine the predictions of multiple decision trees.

This study applies a Random Forest classifier to the UCI Default of Credit Card Clients dataset. The implementation follows a simple, reproducible workflow: dataset acquisition, inspection, duplicate removal, exploratory analysis, feature-target separation, stratified train-test splitting, model training, prediction, and evaluation. The uploaded Python implementation uses a fixed configuration of 100 trees and does not perform hyperparameter optimization.

II. PROBLEM STATEMENT

The objective is to predict whether a credit card customer will default on the next payment based on demographic and financial information contained in historical records. This is formulated as a binary classification task with the following target labels:

Target Value	Meaning
0	No Default
1	Default

© 2026 Dr. V. Subha. his is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

Objectives

- Use the UCI Default of Credit Card Clients dataset for credit-default prediction.
- Inspect and preprocess the dataset.
- Identify and remove duplicate records.
- Perform exploratory data analysis, including correlation and target-distribution analysis.
- Separate input features and target values.
- Create an 80:20 stratified train-test split.
- Train a Random Forest Classifier with 100 trees.
- Generate predictions for the test data.
- Evaluate the classifier using weighted accuracy-related metrics, a classification report, and a confusion matrix.

III. DATASET DESCRIPTION

The study uses the Default of Credit Card Clients dataset from the UCI Machine Learning Repository. The repository describes the dataset as a multivariate classification dataset containing 30,000 instances and 23 features, with integer and real-valued feature types. The variables describe credit limits, demographic characteristics, repayment history, bill statements, and previous payment amounts. The repository also reports that the dataset has no missing values. [1]

The uploaded code obtains the feature matrix and target matrix using fetch_ucirepo(id=350), concatenates them into a single DataFrame, checks duplicates, and removes duplicate rows before model development. The resulting dataset used in the paper contains 29,965 records after removal of 35 duplicate rows.

Class	Records	Proportion
No Default	23,335	77.87%
Default	6,630	22.13%
Total	29,965	100.00%

The class distribution is therefore imbalanced, with the No Default class substantially larger than the Default class. The imbalance is an important consideration when interpreting the classification results.

IV. METHODOLOGY

The workflow implemented in the uploaded Python script is:

Dataset Acquisition → Data Inspection → Duplicate Removal → Exploratory Data Analysis → Feature/Target Separation → Stratified Train-Test Split → Random Forest Training → Prediction → Metric Evaluation

1. Dataset Acquisition

The code imports fetch_ucirepo from ucimlrepo and retrieves dataset ID 350 from the UCI Machine Learning Repository. The features and targets are then combined with pandas.concat(). The repository provides the same Python import pattern for this dataset. [1]

2. Data Inspection and Cleaning

The implementation checks the head and tail of the DataFrame, dataset information, duplicate count, shape, and target distribution. Duplicate rows are counted using df.duplicated().sum() and removed using df.drop_duplicates(). The paper reports 35 duplicated rows removed, leaving 29,965 records.

3. Exploratory Data Analysis

The code generates a numerical correlation heatmap using df.corr(numeric_only=True) and a bar chart showing the target-class distribution. The target distribution confirms that the No Default class is the majority class.

4. Feature and Target Separation

The target column is identified programmatically using target_column = y.columns[0]. The feature matrix is then created using X = df.drop(target_column, axis=1), while y contains the target values. The resulting feature matrix contains 29,965 records and 23 input features.

5. Train-Test Split

The code uses train_test_split() with test_size=0.2, random_state=42, and stratify=y. This produces an 80:20 split while preserving approximately the same class distribution in the training and test subsets.

Subset	Records	Features
Training set	23,972	23
Testing set	5,993	23

6. Random Forest Model

The classifier is implemented using sklearn.ensemble.RandomForestClassifier with n_estimators=100 and random_state=42. The model is trained with rf_model.fit(X_train, y_train). No GridSearchCV, RandomizedSearchCV, SMOTE, scaling, feature selection, or alternative-model comparison is present in the uploaded script.

Model configuration:

RandomForestClassifier(n_estimators=100, random_state=42)

7. Prediction and Evaluation

Predictions are generated using rf_model.predict(X_test). The code calculates accuracy_score, precision_score, recall_score, and f1_score. Importantly, precision, recall, and F1-score are calculated with average="weighted" and zero_division=0. A classification report and confusion matrix are also generated.

V. PERFORMANCE EVALUATION

Metric	Reported Result
Accuracy	81.31%
Weighted Precision	79.39%
Weighted Recall	81.31%
Weighted F1-Score	79.30%

The weighted metrics above correspond to the metric definitions implemented in the uploaded script. Because the supplied .py file contains the metric-calculation statements but does not embed the printed numerical outputs, the reported values are retained from the supplied research paper rather than presented as a newly reproduced run.

Classification Report

Class	Precision	Recall	F1-Score
No Default (0)	0.84	0.94	0.89
Default (1)	0.64	0.36	0.46

The No Default class has substantially stronger recall than the Default class. In particular, default-class recall is 0.36, indicating that a significant fraction of actual default cases are not identified by the classifier. This is an important finding because default prediction is generally more concerned with identifying risky cases than simply maximizing overall accuracy.

Confusion Matrix

The uploaded implementation generates a confusion matrix using confusion_matrix(y_test, y_pred) and visualizes it as a heatmap. The four outcomes are true negative, false positive, false negative, and true positive. The source .py file does not contain the numerical confusion-matrix output, so numerical cell counts are not stated here.

VI. RESULTS AND DISCUSSION

The Random Forest classifier achieved an accuracy of 81.31% on the test set. Weighted precision was 79.39%, weighted recall was 81.31%, and weighted F1-score was 79.30%. These figures indicate useful overall classification performance for the selected baseline configuration.

However, the class-wise evaluation is more revealing than overall accuracy alone. No Default recall reached 0.94, whereas Default recall was only 0.36. The corresponding F1-scores were 0.89 and 0.46, respectively. The lower performance on the Default class is consistent with the observed class imbalance, where 6,630 default observations are present compared with 23,335 non-default observations.

The results should therefore be interpreted as a baseline experiment rather than as evidence that the selected Random Forest configuration is optimal. The uploaded implementation intentionally uses a fixed model configuration and a single stratified train-test split without hyperparameter search or model comparison.

Advantages of the Approach

- Simple and reproducible implementation in Python.

- Suitable for nonlinear relationships in tabular data.
- Combines multiple decision trees to produce an ensemble prediction.
- Works directly with the numerical feature representation used in the dataset.
- Provides standard classification metrics and a confusion matrix for evaluation.

Limitations

- The target classes are imbalanced, which contributes to unequal class-wise performance.
- Only one Random Forest configuration is evaluated.
- No hyperparameter optimization is performed.
- The evaluation uses one train-test split rather than cross-validation.
- The relatively low recall of the Default class limits the model's usefulness for applications where missed defaults are costly.
- The uploaded code does not include threshold tuning, probability calibration, class weighting, or imbalance-handling techniques.

Future Scope

- Apply class-imbalance techniques such as class weighting or resampling and evaluate their effect on Default recall.
- Investigate Random Forest feature importance and other explainability methods.
- Use cross-validation to obtain more robust estimates of generalization performance.
- Tune Random Forest hyperparameters and compare multiple configurations.
- Evaluate additional metrics such as ROC-AUC, precision-recall AUC, and class-specific error rates.
- Compare Random Forest with other classification algorithms to establish a stronger benchmark.

VII. CONCLUSION

This study presented a Random Forest-based approach for predicting credit card default using the UCI Default of Credit Card Clients dataset. Following the workflow implemented in the uploaded Python code, the data were acquired through ucimlrepo, combined into a DataFrame, inspected, and cleaned by removing 35 duplicate rows. The resulting dataset contained 29,965 records and 23 input features. Exploratory analysis included correlation analysis and examination of the target distribution.

The data were divided using an 80:20 stratified split, resulting in 23,972 training records and 5,993 testing records. A Random Forest Classifier with 100 trees and random_state=42 was trained without hyperparameter optimization. The reported evaluation results were 81.31% accuracy, 79.39% weighted precision, 81.31% weighted recall, and 79.30% weighted F1-score.

Although the model performed well on the No Default class, the Default class had a recall of only 0.36. This finding highlights the importance of class-specific evaluation in credit-default prediction. Overall, the experiment provides a clear baseline implementation and identifies class-imbalance handling, model tuning, and more robust validation as important directions for future work.

REFERENCES

1. UCI Machine Learning Repository, "Default of Credit Card Clients," Dataset ID 350, I-Cheng Yeh. DOI: 10.24432/C55S3H. [1]
2. L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
3. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.