

Credit Card Fraud Detection Using Machine Learning

An academic demonstration on a class-imbalanced credit card dataset

Kanisya E¹, Harini S², Indhumathi S³, Keerthika L⁴, Dr. K. Pradeepa⁵

#1,2,3,4 Students, Department of Computer Science with Artificial Intelligence and Data Science, KPR College, Coimbatore, Tamil Nadu, India

5 Dr. K. Pradeepa

Head of the Department

Department of Computer Science with Artificial Intelligence and Data Science, KPR College, Coimbatore, Tamil Nadu, India

Abstract

Credit card fraud is a significant challenge in digital financial transactions, causing financial losses to customers, banks, and financial institutions. Traditional fraud detection systems rely on predefined rules and manual verification, which may fail to identify complex and continuously changing fraudulent patterns. This project proposes an artificial-intelligence-based credit card fraud detection system using machine learning to classify financial transactions as legitimate or potentially fraudulent. The system uses the publicly available European credit card fraud detection dataset from Kaggle. Data preprocessing, feature scaling, and class-imbalance handling with class weighting are applied. SMOTE is not used. Logistic Regression is the baseline classifier, and Random Forest is the primary classifier. The system integrates a Python backend with a MySQL database and a web interface. The application is an academic demonstration and does not connect to a live banking or payment network.

Keywords—credit card fraud detection; machine learning; Random Forest; Logistic Regression; class imbalance; class weighting; Flask; MySQL; web application.

I. INTRODUCTION

The rapid growth of online shopping, mobile wallets, and contactless cards has made digital payments a normal part of everyday life. Each card transaction travels through several parties, including the cardholder, merchant, acquiring bank, card network, and issuing bank. The same speed and convenience that help genuine customers also create opportunities for misuse of stolen card details. The resulting losses can affect customers, merchants, and financial institutions.

Fraud detection has traditionally relied on rules written by human analysts. A typical rule may flag a payment above a fixed amount or a purchase made in an unusual location. Rules are easy to understand, but they are fixed when written. Fraud patterns can change quickly, so a rule that worked previously may fail later. A large rule set can also be costly to maintain and can create false alarms that affect legitimate customers.

Machine learning provides a complementary approach. Instead of writing every condition by hand, a model is trained on historical transactions whose labels are known as legitimate or fraudulent. The model learns statistical patterns that separate the two groups and can then score a new transaction automatically. This project follows that supervised-learning approach and presents it through a simple web application so that the complete path from input to prediction can be demonstrated.

A. Motivation

The work is motivated by the continued growth of digital payments, which makes it impractical to review every

transaction manually. Rule-based systems can also lose value when fraud patterns change. Because fraud is rare, a model can appear highly accurate while detecting no fraud if only accuracy is reported. An integrated demonstration connecting a machine-learning model, an application programming interface, a database, and a simple interface is therefore useful for academic learning.

B. Scope and Objectives

The scope is an academic demonstration. Logistic Regression and Random Forest are trained and compared on a public research dataset. Class imbalance is handled with class weighting. Synthetic Minority Oversampling Technique (SMOTE) is not used in the proposed implementation. The selected model is wrapped in a Flask application programming interface, predictions are stored in MySQL, and the result is shown in a browser interface. The system does not connect to a live payment network and does not approve or decline real payments.

The objective is to preprocess the public transaction dataset, train Logistic Regression as a baseline and Random Forest as the primary classifier, and evaluate both with accuracy, precision, recall, F1-score, and a confusion matrix. The selected model is then integrated with a Python Flask backend and a MySQL database so that a user can submit transaction features and view the predicted category.

II. PROBLEM STATEMENT

The increasing use of digital payments has created more opportunities for fraudulent credit card transactions. Traditional rule-based systems may not identify complex and changing fraud patterns, and they can generate false alarms. At the same time, fraudulent transactions form only a very small share of all payments. This class imbalance makes accurate classification difficult because a model that labels every transaction as legitimate can report high accuracy while detecting no fraud. Manual verification of every payment requires time and effort, so there is a need for an automated system that classifies transactions using machine learning. Such a system should reduce false alarms while still identifying fraudulent activity. Performance must be judged with suitable metrics, especially precision, recall, and F1-score, rather than accuracy alone.

The dataset contains 492 fraud cases among 284,807 transactions. A program that always answers legitimate would be correct for 284,315 records, about 99.83 percent accuracy, while detecting none of the fraud. This illustrates the accuracy paradox and motivates the use of multiple evaluation measures.

III. LITERATURE REVIEW

A. Efficient Credit Card Fraud Detection Using Meta-Heuristic Techniques

Published in Mathematics in 2024, the CCFD study investigates credit card fraud detection using machine-learning algorithms and meta-heuristic optimization techniques. It provides background for algorithm selection and parameter optimization. The present project adopts a manageable classification approach using Random Forest [2].

B. Data Leakage and Deceptive Performance

Hayat and Magnier (2025) examine data leakage, temporal validation, and misleading performance measurements. The study emphasizes splitting data before preprocessing and keeping the test set untouched. These principles are followed in the proposed workflow [1].

C. Investigating Credit Card Payment Fraud with Advanced Machine Learning

Published in Information in 2024, this study uses the European credit card transaction dataset and evaluates machine-learning approaches. It discusses class imbalance and the importance of suitable classification metrics [3].

D. Gated Attention Generative Adversarial Networks

This study explores deep learning and generative techniques for class imbalance. It provides background for representing minority fraud examples. The current project uses class weighting as a simpler alternative suitable for an academic demonstration [4].

E. Research Gap

Existing studies investigate optimized algorithms, deep learning, and imbalance-handling methods. An academic implementation also needs a simple interface that connects model predictions, transaction storage, and user interaction. This project integrates a classification pipeline with a web application and a MySQL database.

IV. DATASET DESCRIPTION

A. Dataset Source

The project uses the European Credit Card Fraud Detection dataset from Kaggle. The dataset contains transactions made by European cardholders during September 2013. It contains 284,807 transactions over two days and 31 columns: 30 input features and one target variable named Class. The input features are Time, Amount, and 28 anonymized principal components labelled V1 to V28. Class is 0 for legitimate transactions and 1 for fraudulent transactions. Only 492 records are fraud, approximately 0.172 percent of the dataset. The original file does not contain customer names or card numbers [5]. The dataset source is <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>.

B. Column Summary

Before training, the dataset should be inspected to confirm its dimensions, missing values, class distribution, and duplicate records. The Amount values of legitimate and fraudulent transactions can also be compared to identify simple value-based patterns before modelling. Table I summarizes the columns. Fig. 2 shows why classification is difficult: 492 fraud

cases are plotted against 284,315 legitimate transactions on a logarithmic scale.

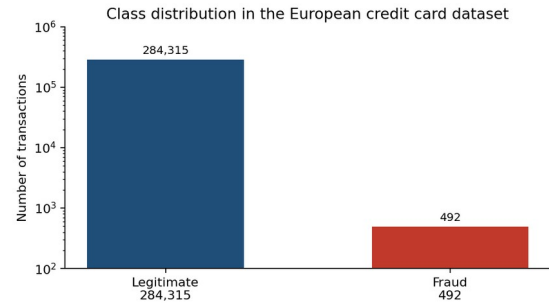


Fig. 2. Class distribution. Fraud is about 0.172 percent of the 284,807 transactions.

TABLE I. DATASET COLUMN SUMMARY

Feature	Type	Description
Time	Numeric	Seconds since the first transaction
V1-V28	Numeric	Anonymized PCA components
Amount	Numeric	Transaction value
Class	Binary	0 = legitimate, 1 = fraud

V. METHODOLOGY

A. Data Processing

The CSV file is loaded with Pandas. Dimensions and missing values are inspected, and input features are separated from the target. The data are split into 80 percent training and 20 percent testing using stratified sampling so that both sets preserve the class proportions. StandardScaler is fitted on the training data only and then applied to both training and test data. This avoids leaking information from the test set into preprocessing.

The standardization operation is represented as

$$z = (x - \mu) / \sigma \quad (1)$$

where x is the original feature value, μ is the training-set mean, σ is the training-set standard deviation, and z is the scaled value. The mean and standard deviation are calculated from the training data only. Scaling is particularly useful for Logistic Regression, while Random Forest is generally less sensitive to feature scaling.

B. Handling Class Imbalance

Fraud is a small minority class. Although SMOTE can generate synthetic minority examples by interpolating between minority samples, the proposed implementation does not generate synthetic transactions. Instead, both classifiers use class weighting. This increases the influence of fraud cases during training while retaining real training examples. The test set remains in its original imbalanced form so that performance is measured on realistic class proportions.

C. Logistic Regression

Logistic Regression estimates the probability that a transaction belongs to the fraud class. It calculates a weighted combination of input features and applies the sigmoid function to produce a probability between zero and one. A threshold then converts the probability into a class label. It is used as an efficient and interpretable baseline classifier.

D. Random Forest Classifier

Random Forest is the primary proposed algorithm. It combines multiple decision trees, with each tree trained using sampled data and feature subsets. The forest combines the outputs of the individual trees. Random Forest can learn nonlinear relationships and is relatively insensitive to feature scaling. The implementation uses 100 trees, a fixed random state, class weighting, and parallel processing [7].

E. Model Evaluation

Both models are evaluated on the held-out test set. Accuracy is the proportion of all predictions that are correct. Precision measures the reliability of fraud alerts, recall measures the proportion of actual fraud cases detected, and F1-score is the harmonic mean of precision and recall. Because the dataset is highly imbalanced, the project also considers the precision-recall curve and the area under the precision-recall curve. A confusion matrix separates true negatives, false positives, false negatives, and true positives [10], [11].

VI. SYSTEM DESIGN AND IMPLEMENTATION

A. Training and Model Storage

The training implementation uses Python libraries including Pandas, NumPy, Scikit-learn, and Joblib. A pipeline combines preprocessing and the classifier. Logistic Regression and Random Forest are trained on the training data, and the selected Random Forest pipeline is saved as `fraud_model.joblib` for use by the backend.

B. Web Application

HTML creates the transaction form and the result display. CSS provides the user-interface layout and styling. JavaScript validates input and sends transaction data to the Python backend through an HTTP POST request. The JSON response is read and the predicted category is displayed on the page. The interface displays either Legitimate Transaction or Potentially Fraudulent.

Python with Flask receives and validates requests, loads the saved preprocessing and model pipeline, and expects the transaction fields Time, V1 through V28, and Amount. The incoming values are converted into a data frame with the required feature order. The model produces a fraud probability and a category. A probability of at least 0.5 is mapped to Potentially Fraudulent; otherwise, the result is mapped to Legitimate Transaction. The returned response contains the prediction and the fraud probability. The prediction is stored in MySQL and returned as JSON.

MySQL stores transaction records, predictions, and timestamps. Suggested fields include transaction identifier, amount, prediction, fraud probability, and created time. The full model input can be stored separately if required for reproducibility. Unnecessary sensitive financial information should not be collected. Real card numbers, card verification values, and personal identification numbers must never be stored. Parameterized SQL queries should be used to reduce SQL injection risk, and database credentials should be kept outside source code. Input validation should occur in both the browser and the server.

C. System Architecture

The system follows a complete three-tier architecture. Fig. 1 shows the path from user input to a stored prediction and a displayed result. The presentation layer, the application layer, and the data layer are separated so that the interface, the model, and the stored records can be maintained independently.

The presentation layer is the browser interface. HTML creates the transaction form, CSS sets the layout, and JavaScript checks that Time, V1 through V28, and Amount are present and numeric before the request is sent. A valid form is submitted as a JSON HTTP POST request.

The application layer is the Flask API and the saved machine-learning pipeline. Flask loads `fraud_model.joblib`, checks the feature order, and converts the values into a data frame. `StandardScaler`, fitted on the training data only, scales the input. Random Forest then returns a fraud probability. A probability of at least 0.5 is mapped to Potentially Fraudulent; otherwise, the result is mapped to Legitimate Transaction.

The data layer is MySQL. After a successful prediction, the backend stores a transaction identifier, the amount, the predicted category, the fraud probability, and the created timestamp. Card numbers, card verification values, and personal identification numbers are not stored. Missing or non-numeric fields are rejected with an error message and are not written to the database.

The response path returns JSON containing the category and the fraud probability. The browser reads that response and displays either Legitimate Transaction or Potentially Fraudulent. The application is an academic demonstration. It does not connect to a live payment network and does not approve or decline real payments. Fig. 1 shows this flow.

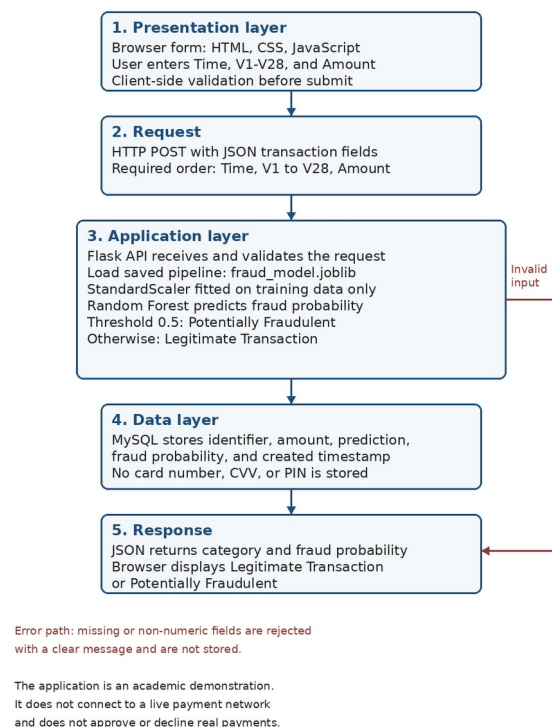


Fig. 1. Complete three-tier architecture: browser input and validation, Flask pipeline and Random Forest prediction, MySQL storage, and JSON response.

D. Technologies Used

Table II lists the tools used to build the demonstration.

TABLE II. TECHNOLOGIES USED IN THE PROJECT

Technology	Purpose
HTML	Web page structure
CSS	User-interface styling
JavaScript	Validation and API requests
Python	Backend and machine learning
Flask	Web API
MySQL	Transaction and prediction storage
Pandas, NumPy	Data processing and numbers
Scikit-learn	Training and evaluation
Joblib	Saving and loading the model
Matplotlib	Performance graphs

E. Testing, Security, and Ethics

Testing covers both the machine-learning model and the web application. The model is tested on the untouched 20 percent test set. The application is tested with valid and invalid inputs to confirm that errors are handled and that records are stored correctly. A complete request should return a prediction and a probability with HTTP 200. Missing or non-numeric fields should be rejected with a clear error message. Database records should be checked after successful predictions.

Predictions are advisory only. A false positive can inconvenience a genuine customer, while a false negative can allow fraud to pass. The project also has dataset limitations because the data come from European transactions in 2013 and do not represent all current transaction behaviour. Only the fields needed for the academic demonstration should be stored.

VII. EXPERIMENTAL RESULTS AND DISCUSSION

Table III reports the measured test-set scores supplied for both classifiers. Logistic Regression scores were supplied earlier, and the Random Forest scores are those printed by the project script. The Logistic Regression area under the precision-recall curve is 0.7124, and the Random Forest area under the precision-recall curve is 0.8629. Accuracy is not interpreted alone, because a model that labels every transaction as legitimate can exceed 99 percent accuracy on this dataset while detecting no fraud.

TABLE III. MEASURED TEST-SET RESULTS

Metric	Logistic Regression	Random Forest
Accuracy	97.55%	99.95%
Precision	6.08%	90.59%
Recall	91.84%	78.57%
F1-score	11.41%	84.15%
AUPRC	0.7124	0.8629

On the supplied Logistic Regression results, accuracy is 97.55 percent, precision is 6.08 percent, recall is 91.84 percent, and F1-score is 11.41 percent. Fig. 3 compares these four scores. The high recall means most fraud cases in the test set are detected. The low precision means most fraud alerts are false positives, which pulls the F1-score down. Fig. 4 shows that split directly: of the transactions flagged as fraud, 6.08 percent are true fraud and 93.92 percent are false alarms. This pattern

is consistent with class weighting, which increases the influence of the rare fraud class and can raise more alerts.

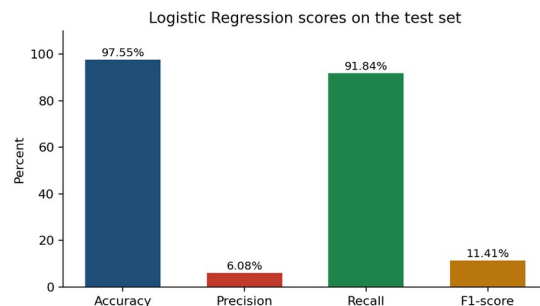


Fig. 3. Logistic Regression test-set scores. Recall is high and precision is low.

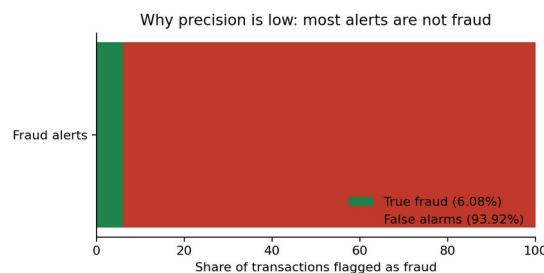


Fig. 4. Composition of Logistic Regression fraud alerts, from the reported precision of 6.08 percent.

The Random Forest script printed accuracy of 99.95 percent, precision of 90.59 percent, recall of 78.57 percent, F1-score of 84.15 percent, and an area under the precision-recall curve of 0.8629. Fig. 5 compares the two supplied result sets. Random Forest has much higher precision, F1-score, and AUPRC, so its fraud alerts are more reliable and its ranking of fraud cases is stronger. Logistic Regression has higher recall, so it catches a larger share of fraud cases but raises many more false alarms. Its area under the precision-recall curve is 0.7124, well above the fraud share of about 0.172 percent, so its ranking is better than a random alert. On the reported scores, Random Forest is the stronger primary classifier because its F1-score of 84.15 percent is far above the Logistic Regression F1-score of 11.41 percent.

Table IV gives the Random Forest confusion matrix printed by the script. Of 98 fraud cases, 77 were detected and 21 were missed. Only 8 legitimate transactions were flagged as fraud.

TABLE IV. RANDOM FOREST CONFUSION MATRIX ON THE TEST SET

Actual class	Predicted legitimate	Predicted fraud
Legitimate	56,856	8
Fraud	21	77

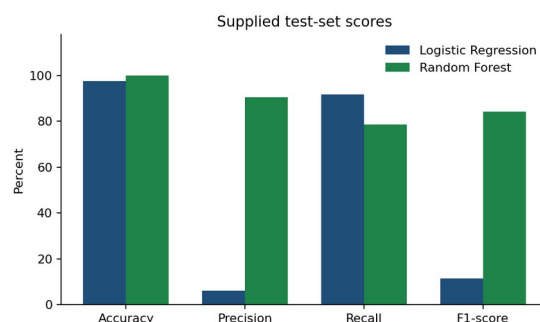


Fig. 5. Comparison of the supplied Logistic Regression and Random Forest test-set scores.

The system can automatically classify transactions as legitimate or potentially fraudulent. Random Forest can capture nonlinear relationships among transaction features, and class weighting addresses class imbalance without creating synthetic samples. The web interface makes the prediction process easy to demonstrate, and MySQL provides a simple history of transaction predictions. Multiple evaluation metrics support a more meaningful assessment than accuracy alone. The use of a public research dataset also supports academic reproducibility.

The dataset contains European transactions from 2013 and may not represent current Indian spending patterns. Most input features are anonymized principal-component values, so their direct business meaning is limited. Class weighting can still miss some fraud or raise false alerts, and fraud patterns change over time, which can reduce model performance. Predictions can include false positives and false negatives. The model has not been validated on live banking traffic, and the application is an academic demonstration that does not connect to a real payment system.

VIII. CONCLUSION

This project presents an academic credit card fraud detection system based on supervised machine learning. The goal is to classify a transaction as legitimate or potentially fraudulent and to demonstrate the complete path from data preparation to a usable web application. The project uses the publicly available European Credit Card Fraud Detection dataset from Kaggle, which is anonymized and highly imbalanced.

The data are split into training and test sets before scaling so that the test set remains untouched. StandardScaler is fitted on training data only. Class imbalance is handled with class weighting in Logistic Regression and Random Forest, and no synthetic fraud samples are created. Logistic Regression is used as an interpretable baseline, while Random Forest is the primary classifier because it can learn nonlinear relationships and is less sensitive to feature scaling.

Model quality is assessed with accuracy, precision, recall, F1-score, a confusion matrix, and related curves. Accuracy alone is not treated as sufficient because a model that never predicts fraud can still appear accurate on this dataset. The trained pipeline is stored with Joblib and served through a Flask API. A browser interface built with HTML, CSS, and JavaScript allows users to submit transaction features and view the predicted class together with a fraud probability. MySQL stores the submitted record and prediction.

The work has clear limitations. The dataset comes from European cards in 2013 and may not match present-day Indian spending patterns. Most features are principal-component values, so their business meaning is limited. Logistic Regression has high recall but low precision. The printed Random Forest scores have higher precision and F1-score, with lower recall. The system has not been validated on live banking traffic.

Future work can use larger and more recent transaction datasets and can explore secure real-time monitoring through authorized payment-gateway integrations. Stronger tree-boosting methods such as XGBoost and LightGBM can be compared with Random Forest, and deep-learning methods can be studied for more complex fraud patterns. Federated

learning can be explored to reduce the need to centralize raw transaction data. Explainable methods such as SHAP can support interpretation of model predictions, and adaptive learning can allow the model to respond to changing fraud behaviour. A secure mobile application with authorized alerts can also be developed.

Overall, the project demonstrates a complete and understandable pipeline consisting of public data, careful data splitting, class-weighted model training, metric-based evaluation, a Flask backend, a MySQL database, and a web interface. The effectiveness of the final model should always be confirmed on a held-out test set before stronger performance claims are made.

REFERENCES

- [1] K. Hayat and B. Magnier, "Data leakage and deceptive performance: A critical examination of credit card fraud detection methodologies," *Mathematics*, vol. 13, no. 16, Art. 2563, 2025.
- [2] "CCFD: Efficient credit card fraud detection using meta-heuristic techniques and machine learning algorithms," *Mathematics*, vol. 12, no. 14, Art. 2250, 2024.
- [3] "Investigating credit card payment fraud with detection methods using advanced machine learning," *Information*, vol. 15, no. 8, Art. 478, 2024.
- [4] "Gated attention based generative adversarial networks for imbalanced credit card fraud detection," *PubMed Central*.
- [5] Machine Learning Group, Universite Libre de Bruxelles, "Credit Card Fraud Detection Dataset," Kaggle.
- [6] C. Elkan, "The foundations of cost-sensitive learning," in *Proc. 17th Int. Joint Conf. Artificial Intelligence*, 2001, pp. 973–978.
- [7] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.
- [8] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B*, vol. 20, no. 2, pp. 215–242, 1958.
- [9] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.
- [10] J. Davis and M. Goodrich, "The relationship between precision-recall and ROC curves," in *Proc. ICML*, 2006, pp. 233–240.
- [11] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, no. 3, Art. e0118432, 2015.
- [12] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification," in *Proc. IEEE SSCI*, 2015.
- [13] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi, "Credit card fraud detection: A realistic modeling and a novel learning strategy," *IEEE Trans. Neural Netw. Learn. Syst.*, 2015.
- [14] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009.
- [15] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, 2002.
- [16] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [17] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [18] G. King and L. Zeng, "Logistic regression in rare events data," *Political Analysis*, vol. 9, no. 2, pp. 137–163, 2001.

- [19] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. KDD, 2016, pp. 785–794.
- [20] Flask Documentation, Flask Web Development Framework.
- [21] MySQL Documentation, MySQL Reference Manual.
- [22] Imbalanced-learn Developers, “Imbalanced-learn: A Python toolbox to tackle imbalanced datasets.”
- [23] C. R. Harris et al., “Array programming with NumPy,” Nature, vol. 585, pp. 357–362, 2020.
- [24] W. McKinney, “Data structures for statistical computing in Python,” in Proc. 9th Python in Science Conf., 2010, pp. 56–61.
- [25] Joblib Developers, “Joblib: Running Python functions as pipeline jobs.”
- [26] Matplotlib Development Team, “Matplotlib: Visualization with Python.”
- [27] Scikit-learn Developers, “sklearn class_weight and compute_class_weight documentation.”