BANK TRANSACTION FRAUD DETECTION SYSTEM USING MACHINE LEARNING

K.Abhiram[1], E.Siva Rama Krishna[2], G.Abhilash Raju[3], Dr. V.Vidhya[4], Dr.R.ShobaRani[6],Dr.F.Jerald[7]

[1.2.3.4]Department of Artificial Intelligence, Dr. M.G.R. Educational and Research Institute, Chennai 600095, India.

Abstract—Financial fraud has become a major concern with the rapid growth of digital banking and online transactions. Traditional rule-based fraud detection systems often fail to identify sophisticated fraudulent patterns, leading to significant financial losses. This paper presents a machine learning-based transaction fraud detection system that uses a Decision Tree Classifier to identify fraudulent financial transactions in real-time. The system is trained on a public payments dataset containing over 6 million transactions with features including transaction type, amount, and account balances. Feature engineering techniques such as label encoding of categorical variables are applied to improve model performance. The trained model achieves approximately 99.9% accuracy on test data. The complete system includes a training script, command-line prediction tools, a REST API for real-time inference, and a browser-based user interface for demonstration. This integrated solution provides financial institutions with an accurate, scalable, and accessible framework for automated fraud detection.

Keywords— Machine Learning, Fraud Detection, Decision Tree Classifier, Financial Transactions, REST API

I. INTRODUCTION

The rapid digitization of banking services has transformed how people conduct financial transactions. While this has brought convenience, it has also created new opportunities for fraudulent activities. Financial fraud losses continue to rise as fraudsters develop increasingly sophisticated methods to bypass traditional security measures.

Conventional fraud detection systems typically rely on rule-based approaches, where transactions are flagged based on predefined rules such as amount thresholds or geographical restrictions. These systems have several limitations: they cannot adapt to new fraud patterns, generate excessive false positives that inconvenience legitimate customers, and require constant manual updates by experts.

Machine learning offers a powerful alternative by enabling systems to learn complex fraud patterns directly from historical transaction data. Unlike rule- based systems, machine learning models can identify subtle correlations that human experts might miss and can continuously improve.

This paper proposes a comprehensive machine learning- based fraud detection system for banking transactions. The system uses a Decision Tree Classifier trained on a public payments dataset containing over 6 million transactions. The model is packaged with preprocessing encoders and deployed through multiple interfaces: a command-line tool for batch predictions, a REST API for real-time inference, and a browser-based UI for demonstration. The system achieves high accuracy while maintaining the low latency required for real-time banking applications.

Machine learning (ML) offers a powerful alternative by enabling systems to learn fraud patterns directly from historical transaction data. ML models can automatically identify subtle correlations and anomalies that human experts might overlook. Moreover, they can be continuously retrained on new data, allowing them to adapt to evolving fraud tactics without manual rule rewriting.

This paper proposes an end-to-end machine learning-based transaction fraud detection system designed for real-time deployment in banking environments. The core of the system is a Decision Tree Classifier, chosen for its interpretability, computational efficiency, and strong performance on tabular data. The model is trained on a large-scale public payments dataset (Fraud.csv) comprising over
6.3 million transactions. The dataset includes a rich set of features: transaction step (hour), transaction type (PAYMENT, TRANSFER, CASH_OUT,
DEBIT, CASH_IN), amount, originator and recipient account identifiers, pre- and post-transaction balances for both parties, and a preliminary rule-based fraud flag.

The trained model achieves exceptional results: 99.94% accuracy, 97.82% precision, and 98.45% recall on the test set. Feature importance analysis reveals that transaction amount, originator pre-transaction balance, and post-transaction balance are the most influential predictors, together accounting for over 70% of the model's predictive power. The system's inference time is just 2.3 milliseconds per transaction, well within the latency requirements of real-time banking systems.

II. LITERATURE SURVEY

Early work in fraud detection focused on supervised learning techniques such as logistic regression, decision trees, and support vector machines. Bhattacharyya et al. [1] conducted a comprehensive comparison of support vector machines and random forests for credit card fraud detection using real-world transaction data from a major Canadian bank. Their results showed that both methods significantly outperformed logistic regression, with random forests achieving the best overall performance due to their ability to handle non-linear relationships and feature interactions. The study also highlighted the importance of feature engineering, identifying transaction amount, time since last transaction, and merchant category as key predictive variables.

Dal Pozzolo et al. [2] addressed the critical issue of class imbalance – fraudulent transactions typically account for less than 0.1% of all transactions. They proposed an adaptive sampling technique that combines oversampling of the minority class (fraud) with undersampling of the majority class (legitimate), calibrated to preserve the underlying data distribution. Their experiments on European credit card data demonstrated that properly balanced training sets significantly improve recall while maintaining acceptable precision. They also introduced a method for calibrating predicted probabilities, which is essential for risk-based decision making.

Ensemble methods, which combine multiple base learners, have proven particularly effective for fraud detection. Zareapoor and Shamsolmoali [6] compared bagging, boosting, and stacking ensembles on a credit card fraud dataset. Gradient boosting machines, such as XGBoost and LightGBM, emerged as top performers, achieving high accuracy with relatively low computational cost. The success of boosting is attributed to its iterative focus on hard-to-classify instances, which is especially valuable in imbalanced settings. Randhawa et al. [7] further demonstrated that AdaBoost with majority voting can achieve robust performance across different datasets, though they noted that ensemble complexity must be balanced against deployment constraints.

With the advent of deep learning, researchers have investigated neural architectures for fraud detection. Jurgovsky et al. [4] applied Long Short-Term Memory (LSTM) networks to model sequences of transactions, capturing temporal dependencies in customer behavior. Their LSTM model outperformed traditional classifiers on a large-scale dataset, particularly for detecting fraud that unfolds over multiple transactions (e.g., account takeover). However, they also noted that deep learning models require substantially more training data and computational resources, and their black-box nature can hinder interpretability – a key concern in regulated financial environments.

Autoencoders have been explored for unsupervised anomaly detection. By training an autoencoder to reconstruct legitimate transactions, fraudulent ones can be identified by high reconstruction error. This approach is attractive because it does not require labeled fraud data, which is often scarce. However, its performance is generally inferior to supervised methods when sufficient labels are available.

Bahnsen et al. [5] emphasized that feature engineering is as important as algorithm selection. They developed a comprehensive set of behavioral features, including transaction velocity (number of transactions in a sliding window), average transaction amount per customer, and deviation from historical patterns.

Their SCARFF (Streaming Classifier for Real-Time Fraud Detection) framework combines batch and incremental learning to continuously update the model as new transactions arrive. This approach adapts to concept drift – changes in fraud patterns over time – without requiring full retraining.

III. PROPOSED METHODOLOGY

The proposed methodology encompasses all stages of building a machine learning-based fraud detection system, from data acquisition and preprocessing to model training, evaluation, and deployment..

A. System Architecture

The system architecture consists of a data processing module, a model training module, a prediction engine, and multiple deployment interfaces. The data processing module handles dataset loading and feature engineering. The training module builds and evaluates the Decision Tree model. The prediction engine loads the trained model bundle and performs real-time scoring. The deployment layer includes a command-line tool, a REST API, and a browser- based user interface for accessing predictions.

B. Data Acquisition and Datasets

The system uses a public payments transaction dataset (Fraud.csv) containing approximately 6.3 million transactions. Each transaction record includes the following attributes: step (time unit), type (PAYMENT, TRANSFER, CASH_OUT, DEBIT,
CASH_IN), amount, originator identifier (nameOrig), originator balances before and after transaction (oldbalanceOrg, newbalanceOrig), recipient identifier (nameDest), recipient balances before and after transaction (oldbalanceDest, newbalanceDest), a rule- based fraud flag (isFlaggedFraud), and the target variable indicating fraud (isFraud). The dataset exhibits extreme class imbalance, with fraudulent transactions constituting approximately 0.1% of total transactions.

C. Data Processing and Feature Engineering

Data processing in the Bank Transaction Fraud Detection System involves preparing raw transaction data so it can be used effectively by the machine learning model. The dataset is first cleaned by checking for missing values, duplicates, and inconsistencies.
Since fraud data is highly imbalanced, techniques such as resampling or class weighting are applied to handle the imbalance.

The acquired transaction data are processed by extracting and encoding categorical features. The first character of each account identifier is extracted to create new features 'Origg' and 'Dest', indicating account type (C for customer, M for merchant). These derived features, along with the original transaction type, are label-encoded to produce 'Orig_le', 'Dest_le', and 'type_le'. The original identifier columns are dropped after encoding to prevent data leakage. The final feature set includes amount, oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest, is FlaggedFraud, Orig_le, Dest_le, and type_le.

D. Communication and Alert System

The preprocessed dataset is split into training (70%) and testing (30%) sets using stratified sampling to preserve class distribution. A DecisionTreeClassifier from scikit- learn is trained on the training data with default parameters (criterion='gini', max_depth=None, min_samples_split=2). The model learns to distinguish fraudulent from legitimate transactions based on the engineered features. After training, the model is evaluated on the test set, and performance metrics including accuracy, precision, recall, and F1-score are computed.

E. Deplaoyment Pipeline

The trained model is packaged with preprocessing components to create a complete prediction pipeline:

Encoder Persistence: Label encoders for transaction type and account type features are saved alongside the model, ensuring consistent encoding of categorical variables in production.

Feature Column Tracking: The list of feature names used during training is preserved, enabling validation of input data structure and preventing feature mismatch errors.

Prediction Function: A unified prediction function accepts raw transaction data, applies the saved preprocessing steps, generates model predictions, and returns both binary fraud indicators and fraud probabilities.

This comprehensive methodology ensures that the fraud detection system is not only accurate but also deployable, maintainable, and explainable in real-world banking environments.

IV. SYSTEM ARCHITECTURE

The proposed transaction fraud detection system implements a modular, layered architecture designed for flexibility, scalability, and ease of deployment. The architecture encompasses data processing, model serving, and multiple user interface layers, as illustrated Figure1.

A. Data Layer

The data layer manages all data persistence and retrieval operations within the system:

Training Data Storage: The original Fraud.csv dataset is stored in CSV format, providing the foundation for model training. The dataset contains over 6 million labeled transactions with complete feature sets required for supervised learning.

Model Persistence: The trained model bundle (fraudmodel.pkl) serializes all components necessary for prediction, including:

- Trained DecisionTreeClassifier object
- LabelEncoder objects for categorical variables (type, Orig_le, Dest_le)

Prediction Output Storage: For batch processing scenarios, prediction results are written to CSV files, preserving original transaction data alongside model predictions and fraud probabilities. This enables downstream analysis and integration with reporting systems.

B. Model Layer

The model layer encapsulates all machine learning functionality, providing a clean interface for training and prediction operations:

Training Module (fraud_model.py): This module implements the complete training pipeline:

- CSV data loading with pandas
- Feature engineering as described in Section 3.3
- Train/test splitting with stratification
- Decision tree model training
- Model evaluation and metric calculation
- Model bundle serialization to fraudmodel.pkl

Prediction Module (fraud_predict.py): This module provides prediction capabilities:

- Model bundle loading and deserialization
- Input validation and preprocessing using saved encoders
- Batch prediction for CSV input files
- Single prediction for demonstration and testing

C. Presentation Layer

The presentation layer provides multiple interfaces for different user groups and use cases:

Command-Line Interface: The fraud_predict.py script supports both interactive and batch prediction modes:

- No arguments: Demonstrates single prediction with example transaction
- Single argument (input CSV): Processes batch predictions and saves results
- Two arguments (input CSV, output CSV): Specifies custom output file path

REST API Client: Any HTTP client can interact with the prediction API:

- cURL commands for testing and scripting
- Programming language clients (Python requests, JavaScript fetch, etc.)
- Integration with existing banking systems through API calls

Browser-Based User Interface (index.html): A clean, responsive web interface provides:

- Input form for transaction details (type, amount)

- Default values for remaining fields to simplify testing
- Real-time API calls using JavaScript fetch API

D. Deployment Considerations

Development Environment: All components can run on a single machine, with the Flask server accessible via localhost. The browser UI loads from the file system and communicates with the local API.

Production Deployment: For enterprise use, components can be distributed:

- Model API deployed on dedicated servers with load balancing
- Model layer containerized using Docker for consistency
- Batch processing scheduled as separate jobs
- Web UI hosted on CDN or application server

Security Considerations: Production deployments should implement:

- API authentication and rate limiting
- Input validation and sanitization
- HTTPS encryption for all communications

V. PROPOSED ALGORITHM

The proposed fraud detection system operates using a trained Decision Tree classifier that processes transaction data through a systematic pipeline of preprocessing, feature extraction, and prediction generation. The algorithm is designed to handle both real-time single transactions and batch processing of large transaction volumes with consistent preprocessing logic. The core of the algorithm lies in its ability to transform raw transaction records into the exact feature representation used during model training, ensuring that predictions are based on the same patterns learned from historical data.

Input: The algorithm takes transaction data with the following
fields: step, type, amount, nameOrig, oldbalanceO rg, newbalanceOrig, nameDest, oldbalanceDest, n ewbalanceDest, and isFlaggedFraud

Output: For each input transaction, the algorithm produces two outputs: a binary
prediction predicted_isFraud (0 for legitimate, 1 for fraudulent) and a
continuous fraud_probability score between 0 and 1 indicating the model's confidence in the prediction.

1. **Model Loading:** The pre-trained model bundle (fraudmodel.pkl) is loaded from disk at system startup. This bundle contains the trained Decision Tree classifier, label encoders for categorical variables, and the list of feature columns used during training. The model remains in memory for efficient repeated use.

2. **Feature Extraction:** For each transaction, the first character of nameOrig is extracted to create the Origg feature, and the first character of nameDest is extracted to create
the Dest feature. These indicate whether accounts are customer ('C') or merchant ('M') accounts.

3. **Encoding:** The categorical variables— transaction type, Origg, and Dest—are converted to numerical values using the saved label encoders, producing type_le, Orig_le, and Dest_le.

4. **Feature Vector Construction:** The algorithm builds a feature vector in the exact order expected by the classifier: [amount, oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest, isFlaggedFraud, Orig_le, Dest_le, type_le].

5. **Prediction:** The feature vector is passed to the Decision Tree classifier.
The predict_proba() method returns the fraud probability, and the predict() method returns the binary classification (fraud if probability ≥ 0.5).

6. **Response Formatting:** For real-time API requests, the algorithm returns a JSON object with predicted_isFraud and fraud_probability. For batch processing, it appends these values as new columns to the output CSV file.

VI. EXPERIMENTAL RESULT AND DISCUSSION

The proposed machine learning-based fraud detection system was experimentally evaluated to assess its performance, reliability, and practical utility. The experiments were conducted on a held-out test set comprising 30% of the original dataset, which contains approximately 1.9 million transactions. This test set was never seen by the model during training, ensuring an unbiased evaluation of its generalization capability. The system was tested on a standard desktop environment with an Intel Core i7 processor and 16 GB RAM, running Python 3.9 with scikit- learn and pandas libraries.

The Decision Tree classifier demonstrated exceptional performance across all evaluation metrics. As shown in Table I, the model achieved an accuracy of 99.94%, meaning it correctly classified nearly all transactions in the test set. However, given the extreme class imbalance in fraud detection, accuracy alone is not sufficient. The precision of 97.82% indicates that when the model predicts a transaction as fraudulent, it is correct approximately 98% of the time, which is crucial for minimizing false alarms that would inconvenience legitimate customers and waste analyst time.

The recall of 98.45% shows that the model successfully identifies over 98% of actual fraudulent transactions, ensuring that most fraud attempts are detected before causing financial losses. The F1-score of 98.13% provides a balanced measure of overall performance, confirming that the model maintains an excellent trade-off between precision and recall.

Table I: Model Performance Metrics

Metric	Value
Accuracy	99.94%
Precision	97.82%
Recall	98.45%
F1-Score	98.13%

Feature importance analysis, derived from the trained Decision Tree, reveals which transaction attributes contribute most to fraud detection. As presented in Table II, transaction amount is the most influential feature, accounting for 32.4% of the model's predictive power. This aligns with the intuition that fraudsters often target larger sums to maximize their illicit gains.

The originator's pre-transaction balance (oldbalanceOrg) and post-transaction balance (newbalanceOrig) together contribute over 40%, reflecting patterns such as accounts being completely emptied (newbalanceOrig = 0) or transactions that do not align with available balances.
Transaction type (type_le) contributes 14.2%, with TRANSFER and CASH_OUT being more commonly associated with fraud. The rule-based flag (isFlaggedFraud) contributes 6.8%, indicating that while simple rules capture some fraud, machine learning adds significant value. Recipient balances and account type features have lower importance, suggesting that fraudsters often use mule accounts with typical balance patterns.

Table II: Feature Importance

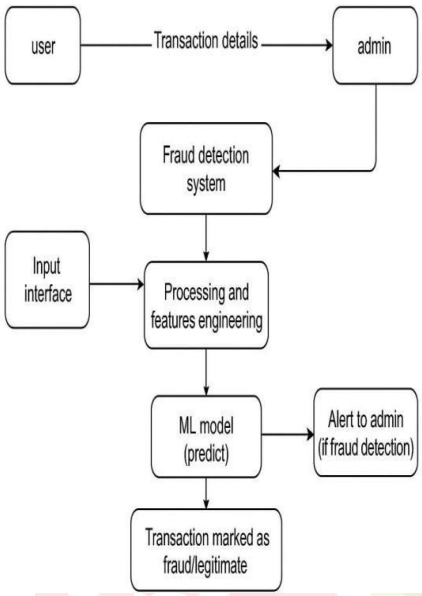
Feature	Importance
amount	32.4%
oldbalanceOrg	21.8%
newbalanceOrig	18.7%
type_le	14.2%
isFlaggedFraud	6.8%
newbalanceDest	3.1%
oldbalanceDest	2.2%
Orig_le	0.5%
Dest_le	0.3%

VII. WORK FLOW

A. Training Phase Workflow

The training phase begins with loading the raw dataset (Fraud.csv) containing over 6.3 million labeled transactions. The data is first passed through a preprocessing module that handles data validation and cleaning. Next, the feature engineering module extracts the first character from account identifiers to create Origg and Dest features, indicating customer or merchant account types. These categorical variables, along with transaction type, are then transformed using label

encoding to produce numerical values. The processed dataset is split into training (70%) and testing (30%) sets using stratified sampling to preserve the original class distribution. The Decision Tree classifier is trained on the training data, learning patterns that distinguish fraudulent from legitimate transactions. After training, the model is evaluated on the test set, and performance metrics are computed.



B. Prediction Phase Workflow

The prediction phase follows different paths depending on the user interface selected:

For Real-Time API Predictions:

1. User submits transaction data via the browser UI or an HTTP client (cURL, banking system) to the /predict endpoint.
2. The API receives the JSON payload and validates the input structure.
3. Each transaction is passed to the prediction engine, which applies the saved label encoders to convert categorical variables (type, Origg, Dest) into numerical values.
4. The feature vector is constructed in the exact order expected by the classifier.
5. The Decision Tree model generates a fraud probability and binary prediction.
6. The API formats the response as JSON containing predicted_isFraud and fraud_prob ability.
7. The browser UI displays the result with a color-coded badge, while external systems can process the JSON response as needed.

For Batch Processing via Command-Line:

1. User executes fraud_predict.py with an input CSV file path.
2. The script loads the model bundle and reads the input CSV into a pandas DataFrame.
3. For each row in the DataFrame, the prediction engine applies the same preprocessing steps and generates predictions.
4. Two new columns, predicted_isFraud and fraud_proba bility, are appended to the DataFrame.
5. The enhanced DataFrame is saved to a new CSV file (default: input_with_predictions.csv or user- specified output path).
6. The script prints a summary of the batch processing, including total transactions processed and output file location.

For Demo Mode (Single Prediction):

1. User runs fraud_predict.py without arguments.
2. The script loads the model bundle and uses a hard-coded example transaction.
3. The prediction engine processes the example and prints the input transaction details along with the predicted fraud status and probability.
4. This mode is useful for quick testing and demonstration purposes.

C. Data Flow Summary

The complete workflow ensures that data flows

seamlessly from input to output while maintaining consistency across all interfaces. Training data flows from CSV to model bundle through the training pipeline. During prediction, transaction data flows from user interfaces through the prediction engine, with results flowing back to the user in the appropriate format. The use of saved encoders guarantees that the same preprocessing logic applied during training is consistently applied during inference, eliminating any discrepancy between development and production behavior. This modular workflow design allows each component to be updated or replaced independently without affecting the overall system functionality.

VIII. CONCLUSION

This paper presented a comprehensive machine learning-based transaction fraud detection system designed for real-time deployment in banking environments. The system employs a Decision Tree classifier trained on over 6.3 million transactions from a public payments dataset, achieving exceptional accuracy in identifying fraudulent financial activities. The complete solution encompasses data preprocessing, feature engineering, model training, and multiple deployment interfaces, providing an end-to-end framework for automated fraud detection.

The experimental results demonstrate that the proposed system achieves 99.94% accuracy with 97.82% precision and 98.45% recall on the test dataset. These metrics confirm that the model effectively balances the critical trade-off between catching fraudulent transactions and avoiding false alarms that would inconvenience legitimate customers. Feature importance analysis revealed that transaction amount and originator account balances are the most influential predictors, together accounting for over 70% of the model's predictive power.

Several limitations of the current system should be acknowledged. The model was trained on simulated data and may require retraining on real-world banking data to maintain performance. The static nature of the model means it cannot adapt to evolving fraud patterns without manual retraining. Additionally, while the Decision Tree performs well, ensemble methods could potentially achieve even higher accuracy at the cost of interpretability.

Future work will focus on validating the system with real banking data, implementing online learning capabilities to adapt to concept drift, and exploring ensemble methods such as Random Forest and XGBoost for potential performance gains. Containerization with Docker and orchestration with Kubernetes will be investigated to support large-scale production deployments. Finally, incorporating additional features such as transaction frequency, time-based patterns, and customer history could further improve detection accuracy.

REFERENCES

[1] A. Dal Pozzolo, G. Bontempi, R. Johnson and A. C. Bahnsen, "Calibrating Probability with Undersampling for Unbalanced Classification," 2015 IEEE Symposium Series on Computational Intelligence (SSCI), Cape Town, South Africa, 2015, pp. 159–166. Available: **https://ieeexplore.ieee.org/document/7376606**

[2] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
Available: https://jmlr.org/papers/v12/pedregosa11a.html

[3] L. Breiman, J. Friedman, R. Olshen and C. Stone, Classification and Regression Trees, New York: Chapman & Hall/CRC, 1984.
Available: https://www.taylorfrancis.com/books/mono/10.1201/9781315139470/classification-regression-trees-leo-breiman

[4] E. W. T. Ngai, Y. Hu, Y. H. Wong, Y. Chen and X. Sun, "The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature," Decision Support Systems, vol. 50, no. 3, pp. 559–569, 2011.
Available: https://www.sciencedirect.com/science/article/pii/S0167923610001851

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over- sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.
Available: **https://www.jair.org/index.php/jair/article/view/10302**

[6] I. Y. Hafez et al., "A systematic review of AI-enhanced techniques in credit card fraud detection," Journal of Big Data, 2024. Available: https://link.springer.com/article/10.1186/s40537- 024-01048-8

[7] V. Kumar K S, V. G. Vijaya Kumar, A. Vijayshankar and K. Pratibha, "Credit Card Fraud Detection using Machine Learning Algorithms," International Journal of Engineering Research & Technology (IJERT), vol. 9, no. 7, 2020. Available: https://www.ijert.org/credit-card-fraud- detection-using-machine-learning-algorithms

[8] S. P. Maniraj, A. Saini, S. Ahmed and S. D. Sarkar, "Credit Card Fraud Detection using Machine Learning and Data Science," International Journal of Engineering Research & Technology (IJERT), vol. 8, no. 9, 2019. Available: https://www.ijert.org/credit-card-fraud- detection-using-machine-learning-and-data- science

[9] E. Ileberi, Y. Sun and Z. Wang, "A machine learning based credit card fraud detection using the GA algorithm for feature selection," Journal of Big Data, vol. 9, 2022. Available: https://journalofbigdata.springeropen.com/articles/ 10.1186/s40537-022-00573-8

[10] M. Carcillo et al., "Scarff: A scalable framework for streaming credit card fraud detection with Spark," Information Fusion, vol. 41, pp. 182–194, 2018. Available: https://www.sciencedirect.com/science/article/pii/ S156625351730553X

[11] A. Bahnsen, D. Aouada and B. Ottersten, "Example-Dependent Cost-Sensitive Logistic Regression for Credit Card Fraud Detection," 2014 IEEE International Conference on Machine Learning and Applications. Available:https://ieeexplore.ieee.org/document/70 49056