IEEE-CIS Fraud Detection Challenge

A Comparative Study of Binary Classification

Khashayar Zardoui
Dept. Computer Science & Software Engineering
Concordia University
Montreal, Canada
khashayar.zardoui@mail.concordia.ca
ID: 40052568

Paolo Junior Angeloni
Dept. Computer Science & Software Engineering
Concordia University
Montreal, Canada
p_ange@live.concordia.ca
ID: 25976944

***Abstract*—The objective of this project was to develop a machine learning pipeline capable of identifying fraudulent credit card transactions within the IEEE-CIS Fraud Detection dataset [1]. Given the extreme class imbalance, we utilized the *Area Under the Receiver Operating Characteristic Curve* (AUC-ROC) [2] as the primary metric to evaluate the models' ability to distinguish between classes. We compare LinearSVC, Decision Tree, and XGBoost classifiers. Our findings demonstrate that the XGBoost ensemble significantly outperformed the single-learner architectures.**

I. Exploratory Data Analysis (EDA)

A. Data Structure Inspection

The dataset consists of transaction and identity tables, which were merged by `TransactionID`.

1) **Data Types & Memory Optimization:** Initially, the datasets contained a mixture of `float64`, `int64`, and `object` types, consuming over 1.7 GB for the training transaction set alone. To improve computational efficiency, we applied a memory reduction function, downcasting numerical columns to `float32` and converting strings to `category`. This reduced memory usage by approximately 51.5%, resulting in a final training structure of 590,540 rows and 434 columns (399 `float32`, 31 `category`) (Fig. I).
2) **Missing Values:** The dataset is sparse. We observed that 414 out of 434 columns in the training set contained missing values, with 208 columns exceeding 60% missing data (Fig. I). A parallel inspection of the test set revealed a nearly identical structure, suggesting that feature reduction would be necessary to reduce nois across both datasets.
3) **Target Balance:** The target variable `isFraud` exhibits extreme class imbalance. Out of 590,540 samples, only 20,668 (3.50%) were labeled as fraud, while 569,871 (96.50%) were legitimate (Fig. 2). This necessitates the use of metrics like AUC rather than simple accuracy.

B. Statistical Summary & Visualizations

The `TransactionAmt` feature revealed a heavily right-skewed distribution. To address this, we applied a log-transformation, which normalized the distribution (Fig. 1), making it more suitable for linear classifiers like the SVM.

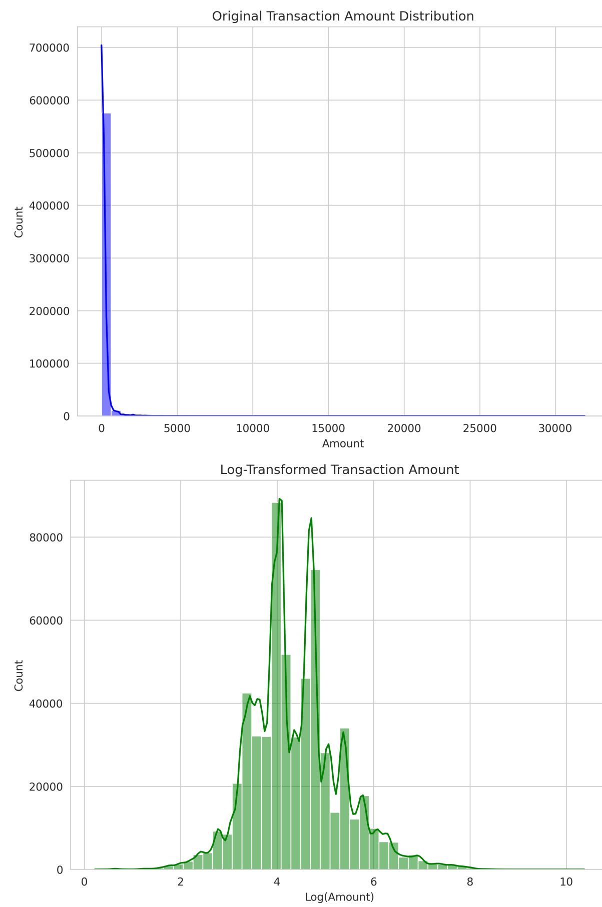

Fig. 1. TransactionAmt Distribution. The log-transformation reveals a normal-like structure, correcting the extreme skew observed in the raw data

TABLE I
Train Dataset Summary Statistics (after merge)

Metric	Value
Total Rows	590,540
Total Columns	434
Numerical Features	403
Categorical Features	31
Cols with $>$ 60% Missing	208

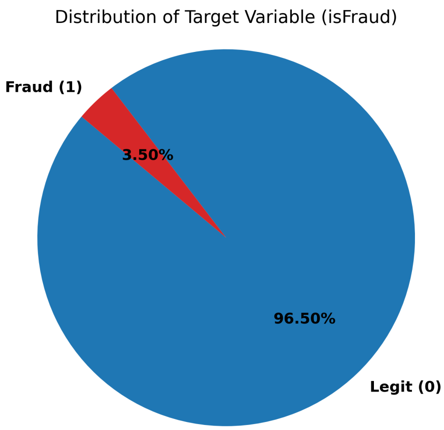

Fig. 2. isFraud Distribution

C. Findings & Hypotheses

Based on the EDA, we formed the following hypotheses to guide our modeling strategy:

- **Imbalance:** Due to the 96.5% vs 3.5% split, models will likely bias toward the majority class. As such, balancing training and validation sets will be important for model performance benchmarks.
- **Feature Selection:** With 208 columns mostly empty, removing features with $> 60\%$ missing values will likely improve model stability and training speed without losing significant information.
- **Normalization:** The extreme skew in `TransactionAmt` confirms that scaling/normalization is necessary for distance-based algorithms like SVM to function correctly.

II. Data Pre-Processing & Cleaning

A. Removal of Noisy and Empty Features [3]

We first calculated the percentage of missing values for every column in the training set. Columns exceeding the defined threshold (60%) of missing data were dropped from both the training and test sets to reduce noise. Additionally, we removed non-predictive features, specifically `TransactionID` (an index) and `TransactionDT` (a time-delta), to prevent the model from memorizing row identifiers or learning spurious correlations that would not generalize to test data.

B. Imputation of Missing Values [4]

The remaining missing values were adjusted using the `SimpleImputer` from Scikit-Learn. We adopted a split strategy based on feature type:

- **Numerical Features**: Imputed using the `median` value. This method was chosen over the mean to be more robust against the heavy outliers present in financial transaction amounts.
- **Categorical Features**: Imputed using the `most frequent` value to preserve the underlying category distribution.

The imputers were fitted only on the training set and applied to the test set to avoid data leakage.

C. Encoding Categorical Features [5]

To convert categorical variables (e.g., `ProductCD`) into a machine-readable numeric format, we applied label encoding.

A standard `LabelEncoder` was fitted on the training data. Known categories were mapped to their learned integers, while unknown categories in the test set were assigned a distinct value of `-1` to prevent errors during prediction.

D. Feature Normalization & Scaling [3]

We applied the (`StandardScaler`) to all features. This transformation centers the data such that each feature has a mean of 0 and a variance of 1.

This step is critical for the Support Vector Machine (SVM) model, which relies on Euclidean distance and can be heavily biased by features with large magnitudes (e.g., `TransactionAmt`) dominating those with small ranges (e.g., encoded categories).

III. Models

Each model required specific configuration and hyperparameter tuning to handle the dataset's size ($590,000+$ rows) and class imbalance (3.5% fraud / 96.5% legitimate). The labeled data was partitioned into a training set (80%), used for model fitting and hyperparameter tuning, and a validation set (20%), used for reporting final metrics. We utilized stratified sampling for this split to ensure that the minority class (fraud) remained represented equally (3.5%) in both subsets.

A. Linear Support Vector Machine (LinearSVC) [6]

To handle the large dataset, we applied **Principal Component Analysis (PCA)** [7] for feature reduction to 83 components that explain 95% of the data's variance. This resolved convergence issues and significantly improved training speed. We then utilized `GridSearchCV` [8] to compare two distinct strategies by tuning the following parameters:

- **Penalty**: Tested `l2`, which gently shrinks all feature weights to prevent overfitting, against `l1`, which aggressively sets weak feature weights to zero.
- **Regularization**: Low values ($[0.1, 0.01, 0.001]$) create a "wider margin" between classes, forcing the model to ignore noise and find a simpler, more generalizable boundary and improve convergence speed.
- **Tolerance**: We adjusted the stopping criteria precision using a standard tolerance ($1e^{-4}$) for the L2 models but a slightly looser tolerance ($1e^{-3}$) for the L1 models to ensure the convergence within a reasonable time.

B. Decision Tree [9]

We implemented a Decision Tree as a non-linear baseline, utilizing `GridSearchCV` [8] to evaluate 18 candidate structures evaluated via 3-fold cross-validation:

- **Max Depth**: We compared restricted depths $[10, 20]$ against `None`, which allows the tree to grow until all leaves are pure (maximum complexity).

- **Min Samples Split**: Tested $[20, 100, 500]$. Higher values force the tree to learn broader patterns by preventing it from creating specific rules for small groups of outliers.
- **Criterion**: `'gini'` vs. `'entropy'` to compare splitting strategies based on Gini Impurity versus Information Gain.

C. Extreme Gradient Boosting (XGBoost) [10]

We utilized the `XGBClassifier` with the following hyperparameters:

- **n_estimators**: Set to 500. This defines the ensemble size (number of trees), meaning the model corrects its errors sequentially 500 times to refine predictions.
- **Learning Rate**: Set to 0.05. A lower rate ensures that no single tree dominates the decision, preventing overfitting and leading to a more stable model.
- **Subsample & Colsample**: Both set to 0.9. This forces each tree to train on a random 90% of the rows and 90% of the features.

IV. Model Comparison

A. Performance Analysis

Table II summarize the performance of the three classifiers. Given the dataset's extreme class imbalance (3.5% fraud), standard **accuracy** was deemed an insufficient metric, as a naive classifier predicting "legitimate" for all transactions would achieve $\approx$ 96.5% accuracy while failing to detect any fraud.

Therefore, **precision** (to minimize customer friction from false positives), **recall** (to capture actual fraud losses), and the **F1-Score** (harmonic mean) are the primary indicators of success. **AUC-ROC** was used to evaluate the model's overall ability to rank fraudulent transactions higher than legitimate ones.

LinearSVC presented a trade-off: it achieved the highest recall (73%), detecting the most fraud cases, but suffered from extremely low precision (9%). This indicates the model generated a high volume of false alarms (False Positives).

Decision Tree showed clear signs of overfitting. Its precision dropped from 92% during training to 77% during validation, and it failed to generalize well, achieving a moderate F1-score of 0.57.

XGBoost emerged as the best model, achieving the highest AUC (0.963) and F1-Score (0.70). It prioritized precision (93%), minimizing false positives to ensure a smooth user experience, while capturing 56% of fraud cases (recall).

B. Computation and Interpretability

Computation: The LinearSVC required PCA dimensionality reduction to converge within a reasonable timeframe. The Decision Tree was the fastest to train but failed to generalize its fast learning to new data. XGBoost, while computationally intensive due to the ensemble size, implements histogram-based optimization to maintain training efficiency on the large dataset.

Interpretability: The Decision Tree offers the clearest "white-box" rules (e.g., "If Amount > 100, then Fraud"). LinearSVC provides feature weights, though these interpretability benefits were obscured by the PCA transformation of original features. XGBoost represents a "black-box" ensemble; while its internal logic is difficult to trace compared to a single tree, it delivers the highest predictive value.

TABLE II
METRICS COMPARISON

Training Metrics

Metric	LinearSVC	Decision Tree	XGBoost
accuracy	0.74	0.98	0.98
precision (0, 1)	0.99, 0.09	0.98, 0.92	0.99, 0.97
recall (0, 1)	0.74, 0.73	1.00, 0.53	1.00, 0.68
f1-score (0, 1)	0.85, 0.16	0.99, 0.68	0.99, 0.80

Validation Metrics

Metric	LinearSVC	Decision Tree	XGBoost
accuracy	0.74	0.97	0.98
precision (0, 1)	0.99, 0.09	0.98, 0.77	0.98, 0.93
recall (0, 1)	0.74, 0.73	1.00, 0.45	1.00, 0.56
f1-score (0, 1)	0.85, 0.16	0.99, 0.57	0.99, 0.70

AUC-ROC Metrics

Metric	LinearSVC	Decision Tree	XGBoost
auc-roc	0.815	0.848	0.962

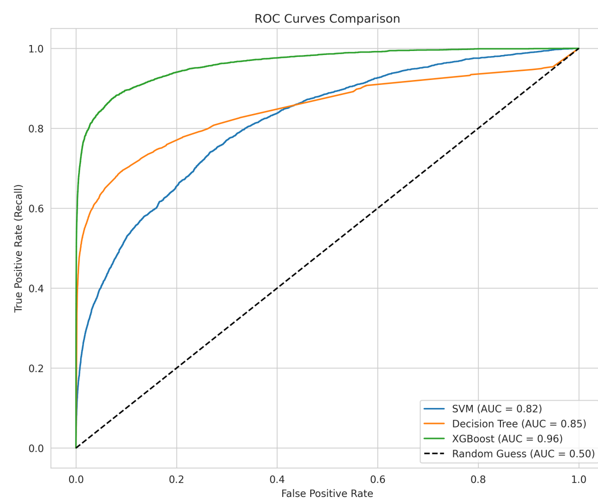

Fig. 3. AUC-ROC for each trained model

Figure 4 visualizes the distinct error patterns of each model. The **LinearSVC** detects the highest volume of fraud (High True Positives, bottom-right quadrant) but generates excessive false alarms (High False Positives, top-right), reflecting its "high recall, low precision" nature. Conversely, the **Decision Tree** misses the most fraud cases (Highest False Negatives, bottom-left), resulting in the lowest recall. **XGBoost** significantly minimizes False Positives (top-right) compared to the SVM while detecting more fraud than the Decision Tree, offering the most precise predictions.

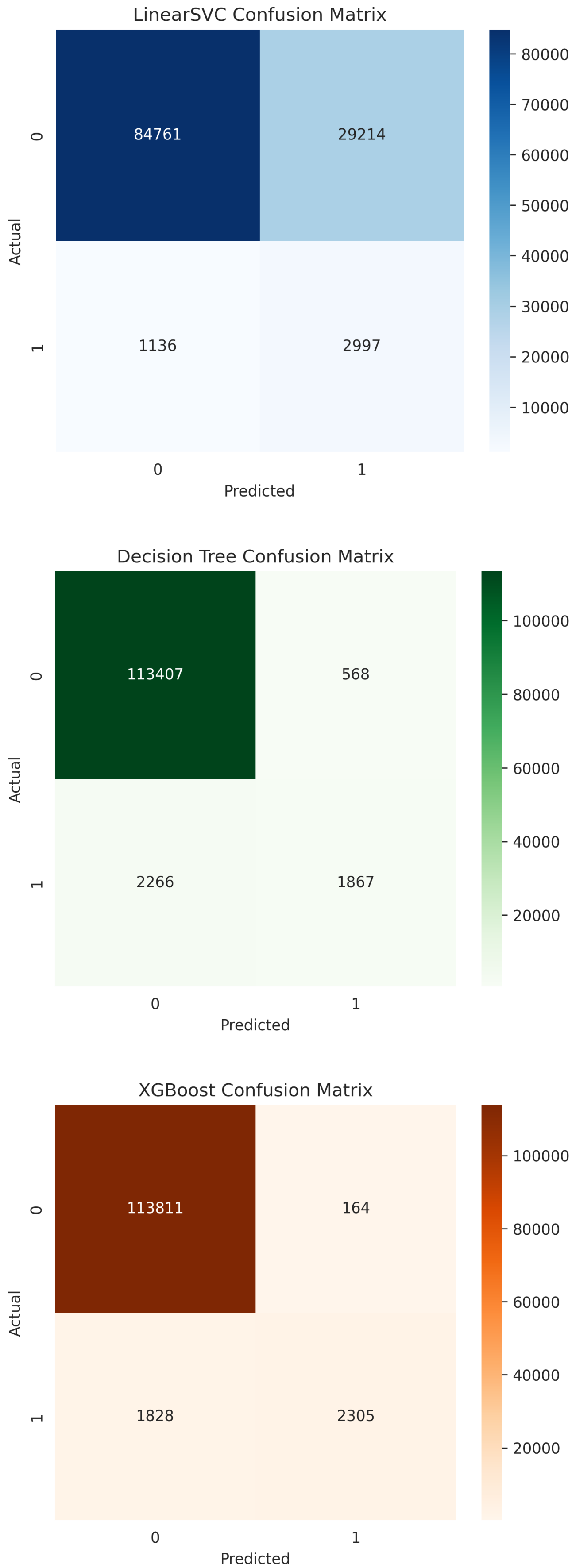

Fig. 4. Confusion Matrices across three evaluated models.

V. Kaggle Submission

Each model's *AUC-ROC* [2] results were submitted to the Kaggle competition [1] in `.csv` format. Each file is a two-column table with `TransactionID` and `isFraud` headers, indicating the confidence level that a transaction is fraudulent. Below are each model's score on the private and public test datasets.

Submission and Description	Private Score	Public Score
submission_xgboost_auc.csv Complete (after deadline) · 14h ago	**0.899808**	**0.930630**
submission_lsvc_auc.csv Complete (after deadline) · 14h ago	**0.821518**	**0.849011**
submission_decision_tree_auc.csv Complete (after deadline) · 14h ago	**0.762838**	**0.807223**

Fig. 5. Kaggle competition submission results

Acknowledgment

We would like to thank Professor Arash Azarfar and Firat Oncel for their guidance and support throughout this project. Large Language Models, like Google's Gemini were used in an educational context to further understand the resources for this research.

References

[1] IEEE-CIS Fraud Detection, "kaggle competition overview," [Online]. Available: https://www.kaggle.com/competitions/ieee-fraud-detection/overview. [Accessed: Nov. 20, 2025].

[2] metrics, "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/api/sklearn.metrics.html. [Accessed: Dec. 02, 2025].

[3] data preprocessing, "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html. [Accessed: Nov. 28, 2025].

[4] data imputation, "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/impute.html. [Accessed: Nov. 28, 2025].

[5] label encoding, "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing_targets.html#label-encoding. [Accessed: Nov. 28, 2025].

[6] Support Vector Machines, "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed: Nov. 30, 2025].

[7] Principal Component Analysis (PCA), "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/decomposition.html#pca. [Accessed: Dec. 02, 2025].

[8] hyperparameter tuning using GridSearchCV, "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html#grid-search. [Accessed: Nov. 30, 2025].

[9] Decision Trees , "sklearn API Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/tree.html. [Accessed: Nov. 30, 2025].

[10] XGBoostClassifier, "XGBoost API Documentation," [Online]. Available: https://xgboost.readthedocs.io/en/stable/. [Accessed: Nov. 30, 2025].