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]. The primary challenge was the extreme class imbalance (approx. 3.5% fraud vs. 96.5% legitimate), requiring models that prioritize Precision (minimizing customer friction via false alarms) while maintaining high Recall (capturing actual fraud). Given this imbalanced dataset, we used the more robust *Area Under the Receiver Operating Characteristic Curve* [2] metric to evaluate a model's ability to rank based on confidence. In experimenting with various machine-learning models, our findings show...**

I. Exploratory Data Analysis (EDA)

A. Data Structure Inspection

1) Before any data transformation, we observed the `train` and `test` datasets had a mixture of `float64`, `int64` and `object` types
2) missing values
   description goes here...
3) target balance
   description goes here...

B. Statistical Summary & Visualizations

Fig. 1. some image here

TABLE I
SOME STATS...

Metric	Value
one	...
two (%)	...
three	...
four	...

C. Findings & Hypotheses

...

...

II. Data Pre-Processing & Cleaning

A. Imputation & Removal

1) ...
2) ...
3) ...

B. Normalize & Scale Features

1) ...
2) ...
3) ...

C. Encoding Categorical Features

1) ...
2) ...
3) ...

III. Models

We experimented with three types of machine-learning classifiers to compare linear, non-linear, and ensemble approaches: 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).

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

To handle the large dataset, we applied **Principal Component Analysis (PCA)** [9] for feature reduction to 83 components that explain 95% of the data's variance. We then utilized `GridSearchCV` [10] 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 (feature selection).
- **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

We implemented a Decision Tree as a non-linear baseline, utilizing `GridSearchCV` [10] to evaluate 54 candidate tree structures, comparing restricted (pruned) trees against fully grown trees:

- **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) [8]

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

The LinearSVC provided a baseline AUC of 0.815, but struggled with convergence times and lacked the complexity to model non-linear fraud patterns. Due to the size of the dataset (590,000+ samples), a standard SVM with a non-linear kernel ($O(n^3)$) was computationally infeasible. We opted for a LinearSVC ($O(n)$) to utilize the entire training set. To satisfy the hyperparameter tuning requirement, we tuned the Regularization parameter (C), penalty (L1 vs. L2) and tolerance, instead of the kernel. Additionally, we applied Principal Component Analysis (PCA) to the SVM input to reduce dimensionality, which resolved convergence issues and significantly improved training speed.

The Decision Tree achieved a higher AUC of 0.848, but exhibited signs of overfitting (high training accuracy vs. lower validation precision), confirming that a single tree has high variance.While it achieved a high F1-score of 0.68 on the training set, this dropped to 0.57 on the validation set. Specifically, the Precision for fraud detection fell from 92% (training) to 77% (validation), indicating that some of the decision rules learned were specific to the training noise and did not generalize well. Furthermore, the Recall remained low in both sets (0.53 training vs. 0.45 validation), suggesting that a single decision tree lacks the complexity required to capture the full variety of fraudulent patterns in this dataset.

XGBoost emerged as the superior model, achieving an AUC-ROC of 0.963 and an F1-Score of 0.70. It successfully balanced a high Precision (93%) with a Recall of 56%, significantly outperforming the other models in identifying fraud without disrupting legitimate users. While the Decision Tree suffered from overfitting (high variance), XGBoost demonstrated robust generalization. On the Validation set, XGBoost achieved a Precision of 0.93, meaning it generated very few false positives (false alarms), which is critical for maintaining user trust. Moreover, it achieved a Recall of 0.56, capturing the majority of fraud instances. The F1-Score of 0.70 (Validation) significantly outperforms the Decision Tree (0.57) and indicates that the Gradient Boosting method successfully captured complex, non-linear relationships that the simpler models missed.

TABLE II
SOME STATS...

Metric	SVM	Decision Tree
one	...	...
two (%)	...	...
three	...	...
four	...	...

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.

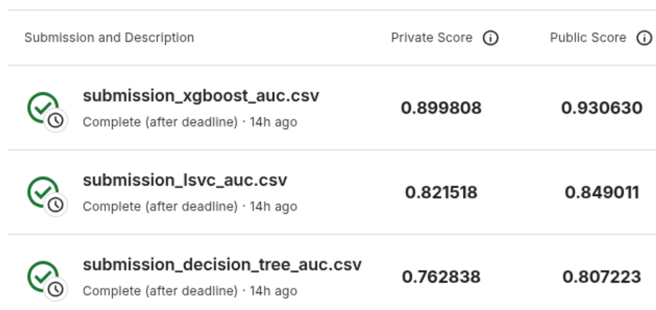

Fig. 2. Kaggle competition submission results

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] Decision Trees , “sklearn API Documentation,” [Online]. Available: https://scikit-learn.org/stable/modules/tree.html. [Accessed: Nov. 30, 2025].
[8] XGBoostClassifier, “XGBoost API Documentation,” [Online]. Available: https://xgboost.readthedocs.io/en/stable/. [Accessed: Nov. 30, 2025].
[9] Principal Component Analysis (PCA), “sklearn API Documentation,” [Online]. Available: https://scikit-learn.org/stable/modules/decomposition.html#pca. [Accessed: Dec. 02, 2025].
[10] hyperparameter tuning using GridSearchCV, “sklearn API Documentation,” [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html#grid-search. [Accessed: Nov. 30, 2025].
[11] Numpy, “Numpy API Documentation,” [Online]. Available: https://numpy.org/doc/stable/. [Accessed: Nov. 25, 2025].
[12] matplotlib, “Matplotlib API Documentation,” [Online]. Available: https://matplotlib.org/stable/index.html. [Accessed: Nov. 25, 2025].
[13] pandas, “pandas API Documentation,” [Online]. Available: https://pandas.pydata.org/docs/. [Accessed: Nov. 26, 2025].
[14] seaborn, “seaborn API Documentation,” [Online]. Available: https://seaborn.pydata.org/api.html. [Accessed: Nov. 26, 2025].