Fraud Detection - Classification Model Evaluation Report

Date: 18th June, 2025

Asmita Ojha

Table of Contents

1. Introduction

This task is about performing different classification algorithms in given transaction dataset to predict fraudulent transactions . The transaction dataset contains a column “isFraud” that can have value either ‘True’ or ‘False’ based on which we will train model and predict if given transaction is fraudulent or non fraudulent. The problem here is binary classification problem.

2. Objective

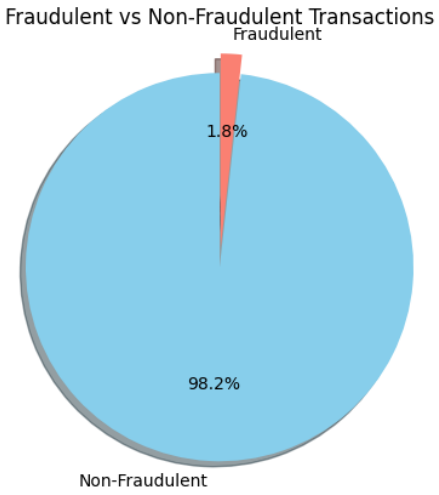
The objective of this task is as follows:

- To perform different classification model for fraud detection in the transaction data.
- Evaluate and compare their prediction and performance
- Generate key insights.

3. Dataset Summary and basic steps

Original dataset contains:

1. 23 columns and 641914 rows
2. After EDA we have extracted some significant columns so we have: 32 columns and 641914 rows
3. Fraud transactions count: 11302
4. Non fraud transactions count: 630612



5. We use 15 columns as features and 1 as target

```
# ✅ Choose meaningful features only
features = [
    'transactionAmount', 'availableMoney', 'creditLimit', 'currentBalance',
    'posEntryMode', 'posConditionCode', 'cardPresent','cvv_mismatch',
    'txn_hour', 'txn_month', 'weekday', 'is_weekend',
    'account_age_days',
    'transactionType',
    # 'acqCountry', 'merchantCountryCode', 'merchantCategoryCode',
    # , 'txn_day','expirationDateKeyInMatch', 'age_bin'
]

X = df2[features]
y = df2['isFraud']
```

6. Encoded categorial features

```
# Encode categorical features
X = pd.get_dummies(X, drop_first=True)

# Optional: scale if needed
```

7. Train/Test split

   We use 70% data for training and 30% for testing.

```python
from sklearn.model_selection import train_test_split

# Split the data: 70% training, 30% testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,        # 30% for testing
    random_state=42,      # Ensures reproducibility
    stratify=y            # Preserves fraud/non-fraud ratio
)
```

4. Models Implemented

In this section, we discuss the machine learning models applied for fraud detection, covering preprocessing techniques, handling of class imbalances, and model evaluation. The goal was to improve fraud detection performance in a highly imbalanced dataset (~1.8% fraud)

4.1. K-Nearest Neighbors (KNN) (Initial Attempt)

A simple instance based algorithm that classifies a transaction based on majority vote of its k-nearest neighbors.

Configurations:

- Tried n_neighbors = 3 and 5
- Used scaled data due to distance-based nature of KNN

Challenges:

- Extremely high computation time
- Hardware performance issues (memory and heating)

Decision:

- Dropped due to poor scalability on large datasets (~640k records)

4.2. Logistic Regression

A linear model that estimates the probability of fraud using a logistic function.

Preprocessing:

- StandardScaler applied to numeric features
- One-hot encoding for categorial features using pd.get_dummies
- class_weight = ‘balanced’ to address data imbalance

Training:

- Used train_test_split(stratify=y) to maintain class ratio in both sets

Initial Evaluation (Threshold = 0.5)

- Accuracy : ~72% (misleading due to imbalance)
- Precision (Fraud) : ~3 %
- Recall (Fraud) : ~ 54%
- F1-score (Fraud) : ~6%
- AUC : ~0.68

Threshold Tuning:

- Lowered threshold to 0.45 to improve recall
- Recall improved to ~68%, but precision remained at ~3%.

Conclusion:

- Good recall, poor precision
- High false positive rate unsuitable for production

4.3. Random Forest Classifier

An ensemble of decision trees, robust to overfitting and capable of modeling complex patterns.

Initial Setup:

- RandomForestClassifier(n_estimators = 100, class_weight = 'balanced', n_jobs = -1),
- n_estimators specifies number of decision trees in the forest. A higher number improves performance and reduces overfitting risk.
- class_weight = 'balanced' : Since fraud class is underrepresented, this helps the model pay more attention to fraudulent cases and not get biased towards the majority of (non fraud) class.
- n_jobs = -1 : utilizes all available CPY cores during training for faster computation.
- Used without SMOTE initially

Variants Explored:

a) Based Model (NO SMOTE, Threshold – 0.5)
   - Confusion matrix : $\begin{matrix}[[189183 & 1] \\ [3375 & 16]]\end{matrix}$

- Recall (Fraud) : 0.004 -> Model failed to detect fraud
- AUC : 0.671

b) With Threshold Tuning (Threshold : 0.08)

- Confusion Matrix [[184779 4405]
  [3034 357]]
- Recall (Fraud) : ~11%
- Precision (Fraud) : ~7%
- F1-Score: ~0.09
- Insight: Slight improvement, but many false positives

c) With SMOTE (Synthetic Minority Oversampling Technique)

- Applied only to training set to balance classes
- Confusion Matrix: [[189010 174]
  [3337 54]]
- Precision: ~24%, Recall: ~2%
- Slight AUC gain: 0.693

d) SMOTE + Threshold Tuning (0.08)

- Confusion Matrix: [[159251 29933]
  [2083 1308]]
- Recall (Fraud) : ~39% (Best recall so far)
- Precision (Fraud) : ~4%
- F1-Score (Fraud) : ~8%
- Conclusion:
  - Effective for high-sensitivity use cases
  - High false positive rate (29k+), suitable only when human verification is possible

4.4. XGBoost Classifier

A high-performance gradient boosting algorithm known for its predictive power.

Imbalance Handling:

- Used scale_pos_weight = ratio(non-fraud/fraud) (~27.89)
- Also tested with SMOTE and undersampling (not as effective)

1. Baseline XGBoost (Default Threshold)

- **Class Ratio (Non-Fraud : Fraud)** = 630,612 : 11,302
- **Issue**: The model is heavily biased toward the majority class.

Metric	**Fraud (True)**	**Non-Fraud (False)**
Precision	0.04	0.99
Recall	0.64	0.71
F1-Score	0.07	0.83
AUC Score	**0.738**	

Observation: High recall for fraud (64%), but extremely low precision (4%), meaning many false positives.

2. Threshold Adjustment (Threshold = 0.6)

- Default threshold (0.5) was replaced with 0.6 based on Precision-Recall curve tuning.

Metric	**Fraud (True)**	**Non-Fraud (False)**
Precision	0.05	0.99
Recall	0.47	0.83
F1-Score	0.09	0.90
AUC Score	**0.738**	

Observation: Reduced false positives, and fraud precision slightly improved. A better trade-off between false positives and false negatives.

3. SMOTE Oversampling

- SMOTE (Synthetic Minority Oversampling Technique) was applied to balance the dataset.

Metric	**Fraud (True)**	**Non-Fraud (False)**
Precision	0.03	0.99
Recall	0.47	0.76
F1-Score	0.06	0.86
AUC Score	**0.676**	

Observation: Slight recall gain, but **precision dropped**, leading to many false positives. AUC dropped as well.

4. Scale Pos Weight Tuning (Final Model)

- The optimal scale_pos_weight was found to be **~27.899**, computed from class distribution and fine-tuned using cross-validation.

Metric	**Fraud (True)**	**Non-Fraud (False)**
Precision	0.06	0.99
Recall	0.36	0.89
F1-Score	0.10	0.94
AUC Score	**0.742**	

Observation: This model had the **best AUC score**, showing improved balance between fraud recall and overall precision, without applying SMOTE.

5. **Scale Pos Weight Tuning (with threshold best based on F1 score: 0.607)**

Threshold chosen based on maximizing F1-score, which balances precision and recall.

This version offers the best trade-off for production consideration.

Metric	Fraud (True)	Non-Fraud (False)
Precision	0.08	0.99
Recall	0.19	0.96
F1-Score	0.11	0.97
AUC Score	**0.742**	

Confusion Matrix:

[[181341 7843]
[2753 638]]

Observation: Significant improvement in both fraud detection (recall = 19%) and false positive reduction. Though precision remains low, this is common in fraud detection where even small gains in recall can be impactful. The model is now far more balanced and reliable for deployment.

6. Performance Metrics Used

I used the following performance metrics to understand how well the models performed.

6.1 Confusion Matrix

The confusion matrix is a 2x2 table that helps visualize the performance of a classification model. It compare the actual class labels with the predicted lables.

	Predicted: False	Predicted: True
Actual : False	True Negative (TN)	False Positive (FP)
Actual : True	False Negative (FN)	True Positive (TP)

- True Positive (TP): Model correctly predicts a fraud case.

- True Negative (TN): Model correctly predicts a non-fraud case.
- False Positive (FP): Model incorrectly flags a non-fraud as fraud.
- False Negative (FN): Model fails to detect a fraud case.

This matrix is the foundation for many performance metrics.

6.2 Accuracy

- The ratio of correctly predicted instances out of total predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Limitation: In imbalanced datasets , accuracy is misleading because the model can predict “non-fraud” for everything and still get high accuracy

6.3 Precision

- Out of all the transactions that the model predicted as fraud, how many were actually fraud.

$$Precision = \frac{TP}{TP + FP}$$

- High precision means fewer false alaram.
- Important when the cost of a false positive is high (eg. wrongly blocking customer transactons).

6.4 Recall

- Out of all the actual fraud cases, how many did the model correctly identify ?

$$Recall = \frac{TP}{TP + FN}$$

- High recall means fewer frauds are missed.
- Crucial in fraud detection, where missing fraud is riskier than flagging in a few extra ones.

6.5 F1 Score

- The harmonic mean of precision and recall.

$$F1\ Score = 2\ \times \frac{Precision\ \times Recall}{Precision + Recall}$$

- A good measure when you need to balance precision and recall.

- Useful in imbalanced datasets like this one.

6.6 ROC Curve and AUC Score

- ROC Curve (Receiver Operating Characteristic) : A graph showing the trade-off between True Positive Rate and False Positive Rate at different thresholds.
- AUC (Area Under Curve): A single value summarizing the ROC curve. Ranges from 0 to 1.
- AUC Score close to 1 = better model performance.

Interpretation:

- AUC = 0.5: Model is guessing randomly.
- AUC ≥ 0.7: Acceptable.
- AUC ≥ 0.8: Good.
- AUC ≥ 0.9: Excellent.

6.7 Classification Report

The classification report is a standard output provided by many machine learning libraries (like scikit-learn). It gives a detailed breakdown of how the model performed on each class, showing:

Metric	Meaning
Precision	Out of predicted positives, how many were actually correct.
Recall	Out of actual positives, how many did the model catch.
F1-Score	Harmonic mean of precision and recall — balances both.
Support	The number of actual occurrences of each class in the test set.

Example

	Precision	Recall	F1-score	Support

False	0.99	0.96	0.97	189184
True	0.08	0.19	0.11	3391

- For the **non-fraud (False)** class:
  - **Precision = 0.99** → 99% of the predicted non-fraud cases were correct.
  - **Recall = 0.96** → 96% of actual non-fraud cases were correctly identified.
  - **F1-score = 0.97** → A high F1-score means the model was both precise and sensitive.
  - **Support = 189184** → There were 189184 non-fraud cases in the test set.
- For the **fraud (True)** class:
  - **Precision = 0.08** → Only 8% of predicted frauds were actually frauds.
  - **Recall = 0.19** → The model caught 19% of all real fraud cases.
  - **F1-score = 0.11** → Low F1-score suggests room for improvement.
  - **Support = 3391** → There were 3391 fraud cases in the test set.

7. Key Insights

Model Performance Comparison (Fraud Class)

Model	Best Precision	Best Recall	Best F1-Score	Best AUC	Remarks
Logistic Regression	~3%	~68%	~6%	~0.68	High recall, but extremely low precision

Random Forest	~4%	~39%	~8%	~0.693	Good recall, low precision, high false positives
XGBoost	~8%	~19%	~11%	0.742	Best overall balance of metrics, suitable for deployment

- XGBoost is the best performing model overall, offering the highest AUC (0.742) and a reasonable trade-off between precision and recall. Ideal for real-world fraud detection when moderate false positives are tolerable.
- Random Forest achieves the highest recall (39%), making it useful when missing a fraud is costlier than flagging non-fraud. But its very low precision (4%) leads to excessive false alerts, limiting production usability without human verification.
- Logistic Regression provides the simplest model, but poor fraud precision (~3%) and modest AUC (0.68) make it unsuitable on its own for highly imbalanced datasets like fraud detection.

8. Conclusion

After comparing the performance of Logistic Regression, Random Forest, and XGBoost on a highly imbalanced fraud detection dataset, XGBoost emerged as the most balanced and reliable model. It achieved the highest AUC score (0.742) and offered a reasonable trade-off between detecting fraudulent transactions (recall = 19%) and minimizing false positives (precision = 8%) after threshold tuning and class weight adjustment. This makes it a suitable choice for deployment, especially in environments where some level of manual review is acceptable.

Random Forest showed the highest fraud detection ability (recall = 39%), but its very low precision (4%) led to a high number of false alarms, which may overwhelm human reviewers. It is more suitable for high-sensitivity settings where missing a fraud is riskier than flagging an innocent transaction.

Logistic Regression, while simple and interpretable, performed poorly in terms of precision and F1-score, making it unsuitable for real-world fraud detection tasks where accuracy in predictions is crucial.

In summary, XGBoost with scale_pos_weight tuning and threshold adjustment provides the best balance between performance metrics and practical usability, making it the recommended model for fraud detection in this context.

9. References (if any)

ChatGPT. (2025). Interactive AI Assistance for Machine Learning Explanation and Analysis. OpenAI. https://openai.com/chatgpt

10. Appendix (Optional)

The full source code, and additional resources used in this project are available in the GitHub repository below:

🔗 **GitHub Repository:** https://github.com/AsmitaOjha/Internship_ExtensoData_Work