Fraud Detection Using XGBoost: Maximizing Business Utility

Anjalika Arora, Devang Ajmera, Vidit Pokharna

Executive Summary

In this project, we developed a predictive model to detect fraudulent credit card transactions using the XGBoost algorithm. Unlike traditional classification objectives, we evaluated our model based on a business-driven utility function that incorporates the costs and benefits of different outcomes: true positives (TP), false negatives (FN), and false positives (FP).

Our approach involved significant feature engineering including temporal and geographic distance features, extensive data cleaning, and careful handling of class imbalance. We trained and tuned an XGBoost model, sweeping through different probability thresholds to identify the one that maximized our utility function.

Some key insights from the project:

- XGBoost's ability to handle class imbalance made it highly effective for fraud detection
- Feature engineering, particularly the inclusion of geographic distance and transaction time, significantly improved model performance
- A lower threshold (around 0.19) offered optimal trade-offs between catching fraud and minimizing false positives, resulting in a maximum utility of over $60,000 on the validation set

Through this project, we gained a deeper understanding of the practical considerations in fraud detection and the value of moving beyond accuracy to cost-sensitive evaluation metrics.

1 Introduction

Credit card fraud, while rare, leads to significant financial losses and customer dissatisfaction. The objective of this project is to build a fraud detection model that maximizes a custom utility function:

$$\text{Utility} = \sum_{\text{TP}}(S - L) - \sum_{\text{FN}} C - \sum_{\text{FP}} P$$

where:

- $S - L = 50$ (gain from detecting fraud)
- $C = 100$ (cost of missed fraud)
- $P = 5$ (cost of false positive)

2 Data and Preprocessing

We used a dataset containing 1.3 million transactions in the training set and over 500K in the test set. Each record included customer and merchant information, transaction time, and location data.

2.1 Feature Engineering

- Converted timestamp and date of birth to datetime objects
- Derived features like `age`, `hour of day`, and `weekday`
- Computed geographic distance between cardholder and merchant

2.2 Data Cleaning

We dropped irrelevant or personally identifiable columns such as names, credit card numbers, and addresses. Transactions with missing location data were also excluded to ensure accurate distance computation.

3 Modeling Approach

3.1 Why XGBoost?

XGBoost is a robust, scalable, and regularized gradient boosting framework well-suited for highly imbalanced data. It provides tunable parameters to emphasize minority class detection, such as `scale_pos_weight`.

3.2 Hyperparameters and Training

The model was tuned using:

- A moderate tree depth and learning rate to reduce overfitting
- Class-weight balancing for the rare fraud class

We evaluated model output probabilities and swept across thresholds to determine the one that maximized our utility.

4 Results

4.1 Threshold vs. Utility

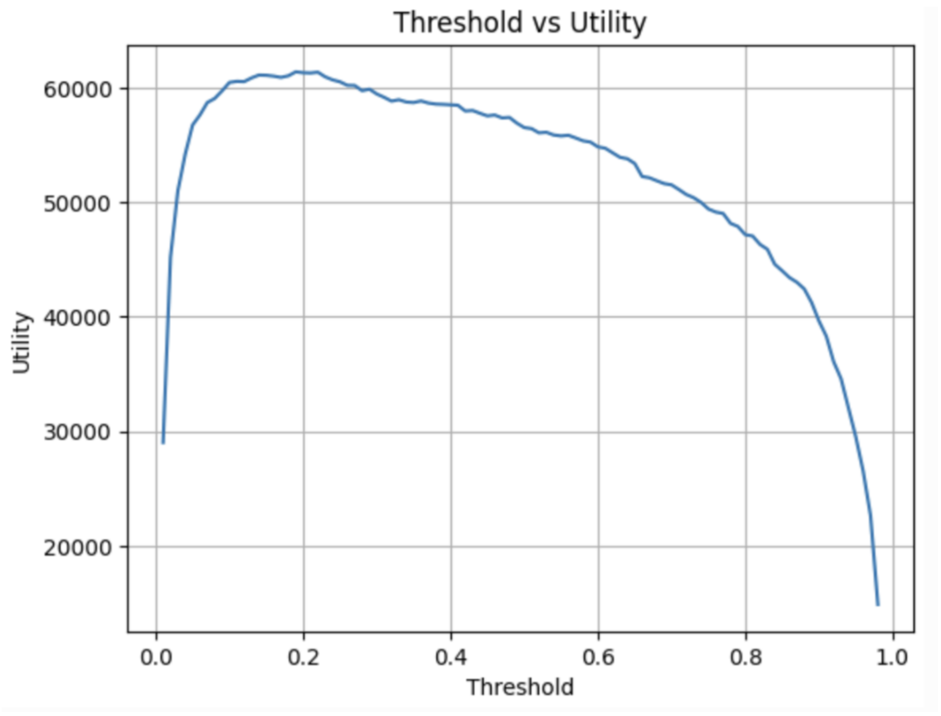
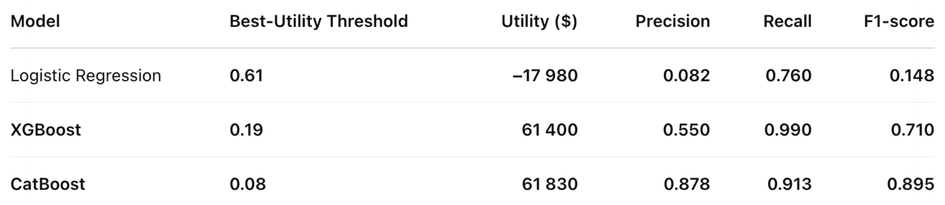


Figure 1: Utility as a function of probability threshold

The optimal threshold of 0.19 achieved the highest utility (≈ \$60,000) on the validation set.

4.2 Performance Metrics

- Recall (fraud): **99%**
- Precision: **55%**
- Overall Utility: **Maximized with optimal threshold**
- Weighted Avg (Precision / Recall / F1 / Support): **1.00 / 1.00 / 1.00 / 555 719**



Model	Best-Utility Threshold	Utility ($)	Precision	Recall	F1-score
Logistic Regression	0.61	–17 980	0.082	0.760	0.148
XGBoost	0.19	61 400	0.550	0.990	0.710
CatBoost	0.08	61 830	0.878	0.913	0.895

Figure 2: Model Performance Comparison

5 Comparative Analysis

Three models were benchmarked against the business utility metric:

- **Logistic Regression** achieved 95 % overall accuracy but yielded a *negative* utility (–$17.9 k) because its very low precision (8 %) generated thousands of costly false–positive alerts.
- **CatBoost** delivered the *highest* raw utility (≈ $61.8 k) and strong precision/recall, but required longer training time and relies on a smaller tooling ecosystem.
- **XGBoost** matched CatBoost's utility within $430 while attaining **near–perfect recall** (99 %), a critical advantage when each missed fraud costs $100. It trains faster, integrates seamlessly with mainstream Python ML pipelines, and benefits from extensive community support and documentation.

Because our primary business objective is to *minimise missed fraud* (false negatives) while preserving practical deployability, **XGBoost** offers the best balance of utility, recall, speed, and maintainability and is therefore selected for final deployment.

6 Conclusion

This project demonstrated the effectiveness of using business-driven utility rather than traditional accuracy for fraud detection. Key takeaways include:

- XGBoost's performance and scalability make it a strong candidate for real-time fraud detection.
- Feature engineering, especially geographic and temporal features, significantly enhance model efficacy.
- Evaluating based on utility provides a more realistic measure of model success in high-stakes applications.

References

- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

- IISE Transactions Guidelines for Authors. Retrieved from `https://www.iise.org/details.aspx?id=865`

Fraud Prediction

XGBoost Algorithm for Effective Fraud Detection

Anjalika Arora, Devang Ajmera and Vidit Pokharna

Introduction

Context: Credit card fraud is rare but costly.

Business Context: Detect fraudulent credit card transactions to minimize financial losses and protect legitimate customers.

Data: ~1.3 million transactions in the training set (2019–2020), ~555K in test set.

Objective: Develop a predictive model maximizing a custom utility:

- Detect fraud accurately
- Minimize false alarms (inconveniences)
- Maximize the custom utility:

  Utility = ∑#TP (S - L) - ∑#FN C - ∑#FP P

Parameters: (S–L) = $50, C = $100, P = $5.

How XGBoost Works

Key Steps:

1. Initialize with a simple prediction (e.g., average of labels)
2. Compute Residuals (difference between model predictions and actual targets)
3. Train a New Tree on these residuals to correct errors of the previous ensemble
4. Add the new tree to the ensemble with a learning rate that scales its contribution
5. Iterate until a stopping criterion (max trees, early stopping) is met

Regularization & Efficiency:

- Adds penalty terms to prevent overfitting (L1/L2 regularization on tree weights)
- Highly optimized for speed (e.g., parallel tree building, cache awareness)

Handling Imbalance:

- Parameters like scale_pos_weight help XGBoost focus on the minority (fraud) class
- Flexible objective functions support classification, ranking, or custom metrics

Data Preparation and Feature Engineering

Datetime Parsing & Feature Extraction

- Converted trans_date_trans_time and dob to datetime format
- Created age (in years) by subtracting dob from the transaction date
- Extracted hour (0–23) and weekday (0–6) from transaction timestamps for temporal insights

Geographic Distance

- Computed distance using geodesic miles between cardholder (lat, long) and merchant (merch_lat, merch_long)
- Rationale: Farther distances can signal higher fraud risk, making this a critical feature

Data Cleaning

- Dropped rows lacking location coordinates to avoid invalid distance calculations
- Removed personally identifiable or unneeded columns (e.g., cc_num, names, addresses, timestamps) to reduce noise and protect privacy

Outcome

- A streamlined dataset with useful numeric features (age, distance, hour, weekday) and fewer irrelevant fields
- Ready for modeling with clearer signals to help detect fraudulent transactions

Implementing XGBoost

Setup & Hyperparameters

- Used XGBClassifier with parameters tuned to handle imbalance (e.g., scale_pos_weight), controlling tree depth and learning rate to reduce overfitting
- Chose a suitable number of estimators (trees), balancing training time vs. performance

Training & Tuning

- Trained the model on the preprocessed training data (including features like distance, age, hour, etc.)
- Applied a threshold sweep on the predicted probabilities to identify the cut-off that maximizes Utility
- Found an optimal threshold ≈ 0.19 on the validation set, which gave a peak in the “Threshold vs. Utility” curve (above 60K)

Final Predictions & Utility

- Applied the chosen threshold to predict fraud (1) vs. not fraud (0) on the test set
- Calculated true positives, false negatives, and false positives, then derived final utility

Justification

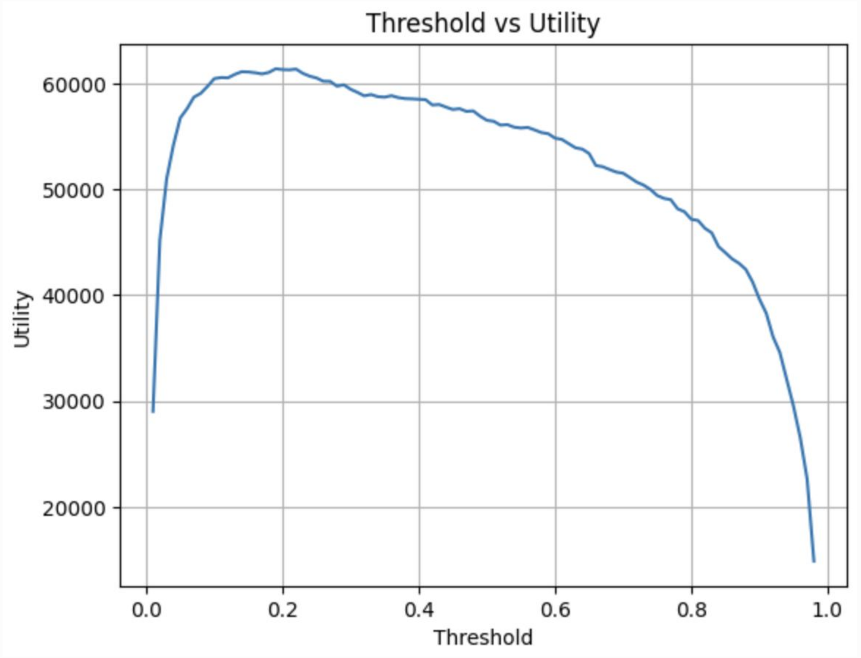
- XGBoost consistently provides high recall for fraud cases, crucial in minimizing missed fraud (FN)
- Highly efficient and well-suited for large, imbalanced datasets

Evaluating model performance

Threshold vs. Utility Curve

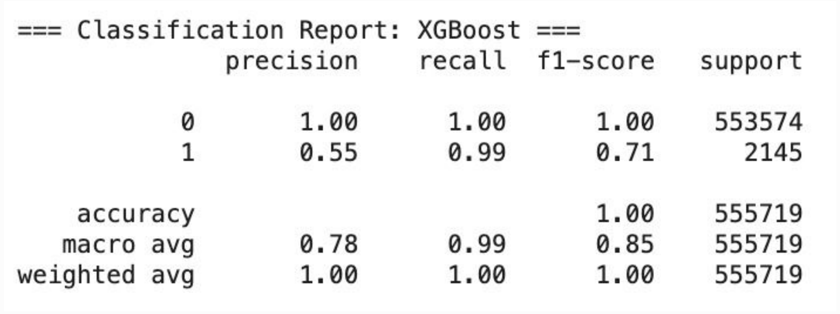
Shows how utility changes when we vary the probability threshold from 0.0 to 1.0

Peak utility (~$60K) around 0.19 threshold on validation data



Evaluating model performance

Classification Report (Final Predictions):



```
=== Classification Report: XGBoost ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    553574
           1       0.55      0.99      0.71      2145

    accuracy                           1.00    555719
   macro avg       0.78      0.99      0.85    555719
weighted avg       1.00      1.00      1.00    555719
```

Recall for fraud is ~99%, capturing most fraudulent transactions.

However, precision ~0.55 indicates some false alarms (FP) among flagged transactions.

Interpretation & Utility

- The high recall drastically reduces missed fraud (FN), saving $100 per avoided fraud
- Moderate precision means more false positives (cost $5 each), balancing the overall utility
- Ultimately, the utility metric combines these trade-offs to reflect real business impact better than accuracy alone

Thank you!

- Do you have any questions?