Credit Card Fraud Detection

Team 5: Lucas Embrey, Simon Grishin, Bea Lee, Tony Lin, Gabriel Montes

1 Executive Summary

This project focused on finding the best model for predicting fraudulent credit card transactions given a sparse multi-dimensional training and testing data sets.

We assessed Logistic Regression, Random Forest, Gaussian Naive Bayes, SVM and XGBoost models using the training data, checking how the trained models performed on unseen validation that was made up from 20% of the training data. All the baseline models gave us negative utility, using our custom utility function except for XGBoost. So, we decided to move forward with the boosting algorithm.

We thought boosting would be particularly helpful in our model due to the nature of the underlying data. Since we knew that we would be using the model to detect credit card fraud, we knew the number of true frauds would be relatively low compared to non-fraudulent transactions. The utility function reminded the team of the impact of not detecting one of these fraudulent transactions. Boosting works well with this kind of data because it is particularly adept at handling imbalanced datasets. By focusing on the instances that are misclassified, boosting ensures that the model gives more attention to the rare fraudulent transactions, which are often overshadowed by the large number of non-fraudulent ones. This helps the model learn to better identify the patterns and characteristics of fraud, even when it occurs infrequently in the data. We chose XGBoost over other boosting algorithms because of its superior performance, flexibility, ability to handle imbalanced data, and efficiency.

We learned that Logistics Regression, Linear SVM, and Gaussian Naive Bayes models were ineffective in predicting fraud, likely due to their inability to capture complex, nonlinear interactions between predictors well. The Random Forest was able to do this to a certain extent because it trains many nonlinear trees on random subsets of the data with random feature selection to de-correlate trees and prevent overfitting, but XGBoost proved to be the best due to the aforementioned reasons.

Lastly, we discuss possible limitations of our approach, and describe areas for improvement.

2 Problem Overview

The goal of this project was to detect fraudulent credit card transactions. This problem is particularly challenging due to the highly imbalanced dataset — fraudulent transactions are rare compared to legitimate ones. Our aim was to minimize both false positives and false negatives, using a custom utility function that incorporates the real-world cost of incorrect predictions. We then used this utility function to evaluate and train the performance of our models.

The utility function is defined as:

$$\sum_{\#TP}(S-L) - \sum_{\#FN} C - \sum_{\#FP} P$$

where:

- $S = \$100$ is the savings from stopping fraud,
- $L = \$50$ is the cost of blocking or investigating a flagged transaction,
- $C = \$100$ is the cost of undetected fraud,
- $P = \$5$ is the penalty for false alarms.

3 Data Description

We worked with a dataset of 1,296,675 credit card transactions, with 23 predictive features and a binary response indicating fraud or not. The class distribution was heavily imbalanced: 1 fraudulent transactions to 172 non-fraudulent transactions.

When preprocessing the data, we decided to drop certain columns that did not add utility to our models, sometimes including cc_num, trans_num, job, first, last, steet, lat, long, merch_lat, merch_long, and the repetitive time unix_time, dependent on the model. We did not drop all of these columns in the logistic regression model since we used forward selection.

We engineered several new features, including a distance variable calculated from latitude and longitude, and performed encoding:

- **Encoding Frequency:** normalized counts for categorical features,
- **Label Encoding:** for ordinal categorical variables like gender,

- **Log Transform:** for skewed numerical features like amount and population.

We split the data into 80% training and 20% validation. Due to the imbalance, we applied a positive class weight of 172 in models like XGBoost and SVM. A bisection search was used to find the best class weight for logistic regression, which was 84.

4 Modeling Approaches

Several models were tested, including:

- Logistic Regression,
- Random Forest
- Gaussian Naive Bayes
- Support Vector Machine (SVM),
- XGBoost.

The models were evaluated on the validation set using the custom utility function. While many models had poor or even negative utility, Random Forest and XGBoost performed the best, with XGBoost significantly outperforming the other models and becoming our chosen model described in more detail outside of this section.

4.1 Logistic Regression

Logistic regression is a supervised algorithm widely used in binary classification tasks, such as identifying whether a transaction is fraudulent or not. It applies a logit transformation on a linear combination of the predicting variables to give a value between 0 and 1 representing the probability that the response is positive, which corresponds to a fraudulent transaction. We chose the predicting variables for the model using forward subset selection with BIC scores. Variable selection stopped when the BIC scores of the remaining unselected variables were below the z-score threshold with an alpha of 0.05.

There was also the issue of class imbalance, with many more legitimate transactions than fraudulent ones. Having no class weighting adjustments led to 0 true positives for the validation set, so we conducted an initial assessment where the fraudulent data points were weighted from 10 to 5000 times as much as the non-fraudulent data points and determined bounds to conduct a bisection search for the best class weight. Although we had utilities on the scale of -1,200,000 for large class weights like 1000 and 5000, we had utilities better than -60,000 for weights between 50 and 500, and the bisection search gave us an optimal class weight of 84 after optimizing to maximize the utility function.

The final logistic regression model used the following variables: amt, dob, hour, month, distance, gender, day, first, state, city_pop, unix_time, city, and category. It had a utility of -13205 on the validation set.

4.2 Random Forest

A Random Forest for binary classification is an ensemble machine learning method that builds many decision trees during training and outputs a positive (fraud) if the proportion of trees predicting positive is above some threshold (default is 0.5) (Tibshirani et al., 2009). Each tree is trained on a random subset of the data and a random subset of features — this introduces randomness to help reduce overfitting and increases model robustness. We considered this model for fraud detection because of its ability to handle complex nonlinear patterns, its resistance to noise and outliers, its ability to interpret which variables are most predictive of fraud, and because it tends to work well with imbalanced datasets.

Then the model was trained on 80% of the data, with 20% left for validation. 100 was used for the number of tree estimators (more than this became very computationally expensive and offered diminishing returns), no max-depth (overfitting didn't seem to be a problem), and balanced class weight to account for the class imbalance in the dataset, i.e. the much greater number of non-frauds than frauds. Next, an optimal decision threshold was acquired by testing the classifications for a number of possible thresholds, and selecting the threshold that maximizes the given utility function on the training data.

In testing the model on the validation set, a utility of -1325 (based on 990 TPs, 257743 TNs, 91 FPs, and 511 FNs) was obtained. Although it still had a negative utility, this model did quite well compared to other models, likely due to its ability to handle complex nonlinear interactions between variables, and its resistance to overfitting and noise.

4.3 Gaussian Naive Bayes

Gaussian Naive Bayes (GNB) focuses on creating probability distributions by class and for every feature. This process creates a probabilistic measure for classification based on a learned threshold for every feature individually. Similar to Logistic Regression, GNB is a base model to test the accuracy of probability reasoning on fraud detection. The largest challenges were feature selection and lack of model complexity. This required a much more hands-on approach to finding features and parameter estimation. In turn, this model was able to return predictions quickly.

During preprocessing, handling feature selection and feature engineering required an iterative approach based on the predictive strength of features, as GNB doesn't estimate parameters directly. GNB learns the mean and variance of features which doesn't provide much separation unless we transform features to numerical values with clear separability. The combination of features that may signify greater chance of fraud can be engineered for better predictability.

When tested on the validation set, GNB returned a maximum utility score of -52675 with a large number of FN and FP values due to the class imbalance. This score signifies that GNB, by itself, is not the best model for fraud detection. Nevertheless, adding boosting methods to learn from false predictions and focus on the smaller subset of fraud cases can be a beneficial addition to increase predictive power.

4.4 Support Vector Machine (SVM)

Support Vector Machines are generally well-regarded for binary classification problems, especially when the classes are separable with a clear margin. Since fraud detection is inherently a two-class problem (fraud vs. non-fraud), SVM initially appeared to be a strong candidate.

SVMs are known to perform well in high-dimensional spaces, which made them appealing for our dataset with 23 features. With proper regularization and tuning, SVMs can also handle noisy data and avoid overfitting, which are both relevant in the context of real-world transaction data. Additionally, the margin-maximizing property of SVMs can help with generalization—crucial for a problem like fraud detection, where the cost of false negatives is high.

However, the success of SVMs is often contingent on the right kernel and the ability to scale efficiently with data size, both of which became limiting factors in our case.

We used the `scikit-learn` library to implement a linear Support Vector Machine (SVM) for fraud classification. Nonlinear kernels, such as the RBF or polynomial kernels, were not used due to computational intractability on our system given the size of the dataset (more than 1.2 million transactions). Linear SVM was more scalable and allowed for experimentation within reasonable runtime.

To address the extreme class imbalance (approximately 1 fraud per 172 legitimate transactions), we assigned a class weight of 172 to fraudulent transactions. This weight ensures that the minority class received sufficient influence during model training.

We also ran a script to search for the optimal decision threshold rather than relying on the default (0.5) to better align predictions with our custom utility function.

Despite these efforts, the linear SVM did not perform well. On the validation dataset (derived from an 80%/20% training split), the SVM produced a utility that was significantly lower than other models. When evaluated on the validation data, the utility score was **-68,860** using an optimized decision threshold. This large negative value indicates that the model performed poorly in terms of cost sensitivity—largely due to false negatives and false positives contributing high penalties in the utility function. As such, SVM was not selected as a viable final model.

4.5 Summary Table

Table 1: Summary Table comparing the results of baseline models. This table includes the number of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) for each of the baseline models as well as the Validation Utility Score.

	Validation Utility Score	TP	TN	FP	FN
Logistic Regression	-13025	1410	252879	8489	735
Gaussian Naive Bayes	-52675	811	252989	4845	690
Random Forest	-1325	990	257743	91	511
Linear SVM	-68860	585	256893	922	935

5 XGBoost Details and Performance

Boosting builds an ensemble of decision trees, improving accuracy by iteratively correcting errors from previous trees. This technique focuses on hard-to-classify instances, giving them more weight in subsequent rounds of learning. Each new tree corrects the residuals of the previous trees. This iterative correction process helps boost the model's performance (Chen and Guestrin, 2016). When implementing our XGBoost model, we set the objective to binary:logistic, which outputs probabilities between 0 and 1, indicating the likelihood of a sample belonging to the positive class. The other important hyperparameters in the model were:

- `n_estimators` – number of trees,

- `learning_rate` – step size,
- `max_depth` – depth of trees,
- `subsample` – fraction of samples used per tree,
- `colsample_bytree` – fraction of features used per tree.

We selected values for these model parameters after running 5 fold cross validation, selecting the ensemble of parameter values with the highest AUC score. After we ran the model utilizing XGBoost on a validation set created from our training data, it performed best by a large margin. The utility was 60,070. When running the model on the test data as a final step, the utility was 63,785.

This large improvement showed both generalization and minimal overfitting. XGBoost also allowed for minimal false negatives, which aligned well with our utility function's penalties.

6 Why XGBoost Was Chosen

We chose XGBoost over other boosting algorithms because of its superior performance and efficiency. XGBoost typically does very well, especially with tabular data, due to its optimization techniques. Its algorithm includes both L1 and L2 regularization techniques which make it less prone to overfitting compared to other boosting algorithms like Adaboost that can overfit, resulting in a worse model. XGBoost provided us with the opportunity to fine tune and tweak the model due to its many hyperparameters. It also supports early stopping (Chen and Guestrin, 2016). XGBoost performs well when handling imbalanced datasets (as stated above), specifically with its inclusion of the **scale_pos_weight** feature that allows for adjustment for class imbalances. We also learned that XGBoost performs quickly, which made cross validation practical.

7 Conclusion

As shown in figure 1, the baseline model ranking was Random Forest, Logistic Regression, Gaussian Naive Bayes and then linear SVM from best to worst. We can conclude that the data is likely to be nonlinear as the linear models underperformed compared to Random Forest which handles nonlinearity better. XGBoost emerged as the most effective model for our fraud detection task, achieving the highest utility on both validation and test sets. The distance feature and encoding schemes contributed to its predictive strength. Through this project, we gained insights into dealing with imbalanced data, cost-sensitive evaluation metrics, and model selection and tuning strategies in a real-world setting.

It is also important to mention some of limitations of our approach, as well as areas for improvement. Firstly, we encountered computational limits in parameter and hyperparameter tuning (e.g. inability to use nonlinear kernel in SVM, inability to test larger number of trees in RF). Also, further data inspection would have revealed the testing data used the same individuals from the training data. Incorporating a measure of personal fraud history could have been a valuable thing to consider in each of our models, and possibly led to improved performance.

References

[1] Chen, T. and Guestrin, C. (2016) XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* `https://doi.org/10.1145/2939672.2939785`

[2] Hastie, T., Tibshirani, R., and Friedman, J. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd edn. Springer-Verlag, pp. 587–603.

[3] Kashishdafe (2024) Gaussian naive bayes: Understanding the basics and applications. *Medium.* `https://medium.com/@kashishdafe0410/gaussian-naive-bayes-understanding-the-basics-and-applications-52098087b963`