Mastercard Data Science Assessment

July 2023

This report presents the approach and results for building an effective fraud detection model using machine learning algorithms containing numeric and categorical features.

Contents:

Gagandeep Kaur
VANCOUVER, BC

1. Analysis & Results

Key Insights:

- High-fraud accounts: Approximately 2.5% of the accounts are solely involved in fraudulent activities. The top account in terms of fraudulent transactions had 26 occurrences within the given time period. Such high-risk and high frequency accounts require close monitoring and investigation to mitigate losses.
- Accounts with both good and fraudulent transactions: 103 accounts (0.4% of total) have both good and fraudulent transactions. Assuming that there are no false positives in the data, this may imply that the good transactions associated with these accounts are mislabelled, representing a scenario where it might take a couple of transactions to detect fraudulent behavior.
- Device / Browser associations:
  - Browser16: This browser seems to be associated with the highest number of transactions and fraudulent activities. Analyzing this association can provide valuable insights to improve fraud detection mechanisms.
  - Device_feat_2 = 'device_4564': Around 32% of all fraudulent activities in the dataset are associated with this specific device. Focusing on monitoring and investigating transactions from this device can enhance fraud detection accuracy.
- Correlation and Feature Selection: Some features exhibit high correlation with values above 0.90. Removing redundant features with a correlation of 1 improves model performance.

Model Selection & Evaluation:

Given the dataset's characteristics, tree-based models *Random Forest* and *XGBoost* were selected due to their ability to handle mixed data and imbalanced classes effectively. To optimize model performance, random search cross validation and hyperparameter tuning (RandomizedSearchCV) is used ensuring robust estimates of the model's capabilities.
Due to the dataset's imbalance, Precision-Recall (PR) curves and F1 score are used for evaluation. The primary focus was achieving a *recall rate* within the desired range of 70-90%, while minimizing *false positives* (frictions to legitimate customers). These metrics are used to assess the model's ability to correctly detect fraud cases and balance precision to prevent inconvenience to genuine customers.

Results:

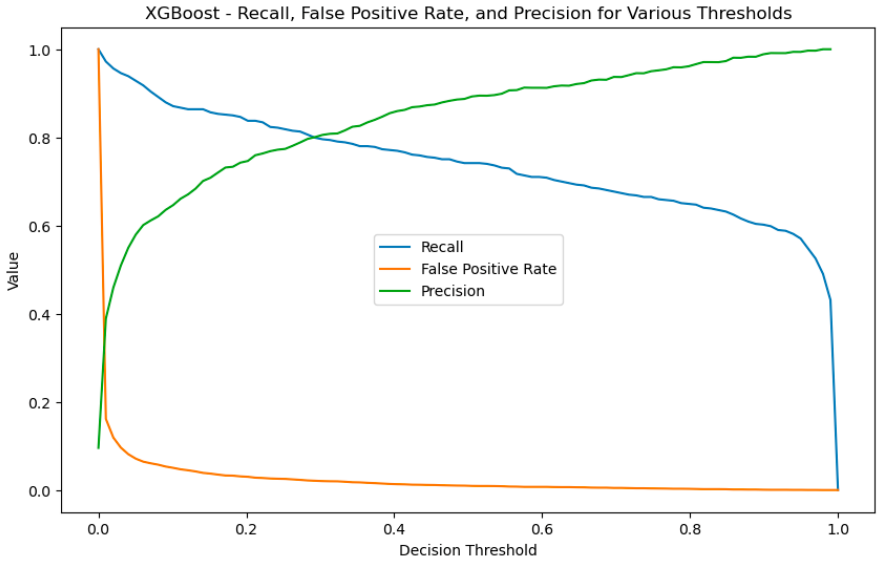
The Final model selected, XGBoost with a decision threshold of 0.55, can detect 73.1% fraud requests keeping the risk of incorrectly labelling good requests as fraud to as low as 0.8%.

Table 1: Key parameters and results

Model	Decision Threshold	Model Comparison (F1 score)	% of Fraud Detected (Recall)	% of Correct Predictions (Precision)	% of False Detections (False Positive Rate)
Random Forest	0.35 (optimal)	0.80	79.5%	80.7%	2.0%
Random Forest	0.50	0.78	70.0%	87.9%	1.0%
XGBoost	0.40 (optimal)	0.81	77.0%	86.0%	1.3%
XGBoost	**0.55**	**0.81**	**73.1%**	**90.3%**	**0.8%**

- Model Comparison: We use F1 score to compare model performance. (Higher the better). It measures the balance between precision and recall metrics. XGBoost has the highest F1 score and better performance over Random Forest.
- Tradeoffs: We need to consider a trade-off between Recall, Precision and False Positive Rate when choosing a model. For every model, we can adjust these metrics based on a decision threshold:
  - % of Fraud Detected (Recall): For the threshold of 0.1, we classify the vast majority of transactions as fraudulent and hence get really high recall of ~0.9. As the threshold increases the recall falls. Increase recall when you care about catching all fraudulent transactions even at a cost of false alerts.
    - Although Random Forest achieved the highest recall of 85.9%, it has the lowest precision of 54.4% and highest false positive rate of 6.6%. XGBoost performed well with a recall of 79.2% and a precision of 76.5%. The false positive rate was 2.6%, and the F1 score was 0.78.
  - % of Correct Predictions (Precision): The higher the threshold the better the precision. When raising false alerts is costly, and you want all the positive predictions to be worth looking at you should optimize for precision.
  - % of False Detections (False Positive Rate) - Fraction of false alerts given by the model. If the cost of dealing with friction is high, we optimize for this metric. If we increase the threshold observations with very high predictions will be classified as positive. In our example, we can see that to reach perfect FPR of 0 we need to increase the threshold to 0.85.

- By increasing the decision threshold from model-optimal (0.4) to 0.55, we can reduce the false positive rate from 1.3% to 0.8% by compromising on recall. Increasing the threshold any further, would decrease the friction rate but may not be worth the decrease in recall.



Model Limitations:

- Limited data quality - the approach involves strategically selecting specific data samples with potentially incorrect labels to train the manual audit team. These carefully chosen samples are then provided alongside similar data for labeling purposes.
- Alignment to real-world data - If the provided data is heavily biased towards the fraud class (10% fraud samples) and does not accurately represent real-world data, the model's performance may suffer.

Potential improvements:

- Real-world alignment can be overcome by collecting more diverse and balanced data; incorporating techniques such as neural networks can be explored once sufficient data is available.
- Model can be improved by using feature importance from XGBoost, using top N features and finding optimal value for K.
- Model can also be enhanced using GridSearchCV instead of randomizedSearchCV as GridSearchCV will find better parameters at the expense of speed.

2. Bonus Questions

Bonus Question 1: Your coworkers are impressed with your report and would like to have this model deployed in production in a real-time fashion, so the client can receive the prediction and make decision about the request in real time. What other considerations and/or code changes would you make prior to deploying your model to a production environment? Below are some considerations (but not limited to) you might address in your answer:

a. In what ways is/isn't this code ready for production deployment?

The current code is meant to explore the data and run experiments to develop an effective machine learning model. It requires major refactoring to be suitable for production deployment. The following steps should be taken to ensure a smooth deployment:

Optimization:

- Optimize the code for real-time environments using libraries like Tensorflow or cloud providers to implement parallel processing.

Packaging:

- Implement input validation checks to ensure data quality and handle inputs effectively.
- Refactor and modularize the code for preprocessing, modeling, and evaluation using functions, classes, configs, and appropriate error logging.
- Implement error handling mechanisms for prediction request errors.
- Add unit testing for each function to ensure reliability and accuracy.

Model Updates:

- Establish a process for regular model updates and retraining based on requirements.
- Save the trained model and establish a versioning system to track model versions in production.

Automated Pipelines and CI/CD:

- Implement an end-to-end automated pipeline for data collection, preprocessing, model training, and deployment.
- Utilize workflow orchestration tools like Airflow, Prefect, or AWS Step functions for integration and continuous deployment.

b. What approach and technology would you use for deployment?

Depending on whether the model is to be embedded in a product or deployed as a new service, we can use the following approach -

i. Embed in an application - The model code is integrated with the application code base and is executed within the application.

ii. Deployed as a service - We can deploy it as a micro service or an API.

For deploying the model, I would opt for a microservice architecture by implementing the model as an API using FastAPI. This approach allows seamless integration with other services and applications as required.

To ensure portability and consistency, I would package the model application and APIs in a Docker container. For infrastructure, I would consider cloud platforms like AWS Lambda, EC2, or Sagemaker for scalability and reliability.

c. When deployed, how do we know whether the solution is working effectively? What metrics might we want to track to confirm this?

We need to implement proper monitoring and performance tracking to ensure effectiveness. Following metrics should be tracked:

1. Model performance metrics - such as precision, recall, F1-score, and PR AUC, to assess the model's effectiveness in detecting fraud and reducing frictions.
2. API performance metrics - Response time, Latency, Throughput (number of requests per unit time) and Error rate (percentage of requests that fail)
3. Prediction performance metrics - Monitor the rate of false positives and false negatives to identify any issues with the model's predictions.

d. How would we update and maintain the model?

1. Define a retraining strategy: Establish a schedule or trigger mechanism for regular model retraining using new data
2. Handle model drift: Model performance can degrade over time in production as the data distribution changes from the training data. This requires regular data quality monitoring to detect any shifts or unexpected changes in data distribution.
3. Continuous Integration/Continuous Deployment (CI/CD): Implement CI/CD pipelines to automate the process of updating the model and deploying new versions.

Bonus Question 2: You found out from the client that the fraud dataset is incomplete, meaning that some of the normal traffic in fact includes fraudulent requests (but not reflected in the is_attack target column). However, the client is unable to identify which requests exactly are fraudulent.

a. **How would you modify your solution to address this additional information?**

- Identifying the labeling errors during pre-processing - Assuming the percentage of potentially fraudulent normal traffic is low, we can use anomaly detection techniques to identify unusual patterns that might indicate fraud. Unsupervised anomaly detection algorithms like Isolation Forest or One-Class SVM can be used.
- Analyze the model results - Investigate records where model prediction does not match the actual target but the prediction probability is high. Assuming that the model has captured sufficient pattern, we can analyze these records to get any hints of mis-labelling.
- Cluster analysis - Another approach involves clustering all the detected fraud cases and examining the normal traffic's proximity to this fraud cluster. Mislabeled samples that are erroneously categorized as “non-fraud" but are actually frauds would be closer to this fraud cluster in the feature space. We can modify the dataset by throwing away the labeled "non-fraud" samples which are very close to the fraud samples. That will improve purity of the dataset.
- Iterative supervised learning – In this approach we split the dataset into train and test and train various models on the train dataset and predict labels on the test set with 90% confidence level to exclude mislabeled data. Store predictions for each data point and retrain and predict for N epochs. For classification, use the most frequent predictions at the final label. We can tune the split ratio and number of epochs as a hyperparameter.

b. **How might this impact the results you presented previously?**

If the errors are random, it will have smaller impact since the tree-based models can handle that randomness. Otherwise, the previous results will not have a good predictive power if the contamination is very high (5-10% noise can be handled by tree based models and other techniques). Due to contamination, model’s predictive power to correctly identify the frauds is reduced which will show up in the metrics. That is the ROC AUC curve will not be that great.

c. **Will you still be able to use the dataset?**

Yes, the dataset can still be utilized effectively. It is common for real-world datasets to contain some labelling errors, and this dataset's richness in information makes it valuable for training a fraud detection model. We need to handle the mislabeled data appropriately, considering the sensitivity of the data and apply iterative supervised learning techniques to gradually improve model performance.

d. What machine learning approaches can help deal with the incomplete fraudulent information?

In addition to anomaly detection and iterative supervised learning, we can use semi-supervised machine learning to deal with incomplete datasets.

Semi-supervised learning - Treat the problem as a semi-supervised learning problem where the target column is partially labeled. Some normal traffic will now have labels indicating potential fraud. Split the dataset into labelled and unlabelled subsets. The labelled subset contains instances with reliable labels, while the unlabelled subset contains instances without labels or potentially mislabelled instances. Apply supervised learning to the labelled dataset and use it to predict labels for the unlabelled data. Repeat the process and merge the labels using a decision threshold.

3. **Appendix: Data Processing Approach**

1. Removing duplicates - reduces records from 30012 to 29973. This makes the dataset unique per timestamp and accountid except 4 records that are investigated further.

2. Handling outliers

   a. Describe() function and feature distribution box plots show that a particular record (accountid - f9fbc61b-66aa-4b32-a6ae-53d3a0eac19b) with extremely large feature values and target ‘is_attack’ = ’False’. The outlier values are replaced with NaN and later imputed with mean for numerical features and mode for boolean features.

   b. Columns 'count_feat_16', ‘count_feat_17' have only 11 non-null values and are dropped.*

3. Handling missing values - Check for missing values in the dataset and handle them appropriately, either by imputation or removal.

4. Data type cleaning - (‘anomaly_feat_0', 'anomaly_feat_1', 'anomaly_feat_2', ‘interaction_feat_0' are converted from object to bool)

5. Encoding Boolean features - boolean features are converted to numeric representation (e.g., 0 for False and 1 for True) so that they can be used in the model.

6. Encoding categorical variables - using ordinal encoder.