G5: Detecting Credit card fraudulent transactions

Pratikshit Singh (ps71), Praveen Kumar Murugaiah (pkm4), Aruna Parameswaran (aruna2)

Note: Our fourth teammate Safin (santon21@illinois.edu) has dropped this course. We have rescoped the work accordingly

Abstract: The objective of this project is to leverage various state-of-the-art data mining techniques to cluster and classify credit card transactions from the MLG-ULB Credit Card Fraud dataset [1] as either legitimate or fraudulent, and additionally extract useful association rules and patterns from the data to help combat fraud.

Introduction: A recent report by the FTC [2] claims that credit card fraud is the most frequent type of identity theft, with over 40% of Americans from the study claiming to have had their credit card information stolen. Banks rely on automated, real-time Fraud Detection Systems (FDS) to combat fraud and protect their customers. These systems screen each transaction, and flag potentially fraudulent ones. The specifics of each system vary based on the scale and amount of data processed, but generally speaking, sophisticated data mining approaches, particularly Knowledge Discovery [4], are used under the hood.

Motivation: There are two reasons why the problem of detecting fraudulent transactions has a high significance. Firstly, it is a high impact, real-world application area. Retail businesses and banks rely on highly accurate Machine Learning algorithms to predict when a fraud has transpired in near real-time to take immediate mitigation steps. Secondly, from a data mining perspective, this is a good case study. Transaction datasets are typically high-volume with a large number of dimensions, and are highly imbalanced; the number of fraudulent data points are disproportionate due to the relatively low probability of occurrence. This provides us with an opportunity to test sampling approaches, analyze implicit model bias, and report all findings.

Related work: Methods to analyze credit card fraud can be largely grouped into two categories [3]. The first category relies on statistical approaches to retrieve information from the data, using data visualization and metrics. The second category relies on AI algorithms to automatically uncover patterns. As the review literature [5] notes, historically, the most commonly used statistical data mining approaches relied on rule-based algorithms, such as association rules or sequential analysis [6], to extract useful patterns from the data. Apriori and FP-Growth are two such algorithms that have been used for Frequent Pattern Mining (FPM) for credit card fraud detection [7]. However, pattern matching algorithms analyze data in batches at a specific period. They do not automatically get re-tuned when new data is introduced, or new patterns emerge.

Over the last decade, the prominence of AI/ML has yielded much more robust approaches that use classification. These include Neural Nets, Bayesian approaches, k-Nearest Neighbors, and Support Vector Machines [8]. AI approaches generally yield much better rates of accuracy, high confidence, and adapt better to new or previously unseen data patterns. The current state of art approaches [9][10][11] combine tree-based models (which work well with structured tabular data) and neural nets (which perform better on unseen data), with an Area Under Curve (AUC) value of over 95%.

The most interesting problem that we want to explore is that of class imbalance. As the literature [12][13] points out, we can adopt a number of different tactics to handle imbalance, including resampling techniques, tuning the hyper parameters of the ML model, and applying boosting algorithms.

Methodology: We first started with data visualization and Exploratory Data Analysis (EDA) which helped us to understand the data, nature of the dataset, and provided insights on the extent of imbalance. Originally we had planned to pre-process the data using normalization to remove any null values and ensure uniform spread. However, our dataset was found to not have null values, so we skipped this step.

To handle class imbalance, we have evaluated different techniques such as random undersampling, oversampling, and SMOTE. We were particularly interested in testing SMOTE as random over and under sampling techniques balance the data by creating duplicate entries, which can introduce bias. SMOTE performs sampling not by duplicating data points but by a method synthesizing or selecting new minority data points which are in close proximity to the existing minority samples, thereby getting an objective evaluation of the model's performance. Some models resulted in better scores over SMOTE, which could be explained by the aforementioned bias.

We've also applied k-folds validation for training different models by generating different datasets (using the imbalance characteristic of our present dataset) and exploring various hyper-parameters. We have applied feature selection by using correlation to reduce the dimensionality and only select necessary features to use for model training. Finally, we trained different classification models, such as the Logistic Regression, KNN, Decision Tree and Random Forest and, against our resampled dataset. Due to the imbalance nature of our dataset, we have also evaluated cost-sensitive XGBoost, which outperformed other models as expected.

Empirical Results: The credit card transactional dataset consists of 30 features including Amount and Time Elapsed for each transaction. Out of the 30 features, 28 have been transformed by PCA for privacy and security reasons. Based on our Exploratory Data Analysis (EDA) using Python, we found that there are 284807 transactions in total, with no missing values in any feature, of which 284315 (99.83%) are non-fraud and 492 (0.17%) are fraud transactions. Our chosen dataset is highly imbalanced. We plotted the distplot for Amount and Time variables and found that the Amount variable is highly skewed, with the majority of transactions valued at less than $1500 and the maximum value spiking at $25691. Time features show a bimodal distribution with highest concentration around 75000 as shown in fig 1.

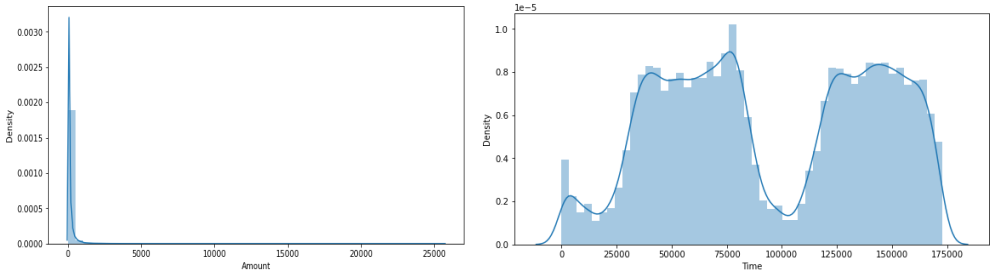

Figure 1: Distribution of Amount and Time variables in the raw dataset

In order for greater consistency, we normalize these features using StandardScaler. We tried out 4 different methods for our analysis - random undersampling, random oversampling, Cost-sensitive

XGBoost with cross validation, and SMOTE modeling. We first implemented a baseline, without preprocessing or outlier removal, and captured the metrics such as AUC score for various models, namely, Logistic Regression, KNN, SVM, Decision Trees, RandomForest, XGBoost.

Logistic Regression	**KNN**	**SVM**	**Decision Tree**	**Random Forest**	**XGBoost**	**Cost-Sensitive XGBoost**
Penalty: L1, L2	N_neighbors: 2, 3, 4, 5, 6, 7, 8, 9	C: 0.3, 0.5, 0.7, 0.9, 1	Criterion: gini, entropy	Tree count: 5, 20, 50, 100	Objective: binary logistic	scale_pos_weight: 577
C: 0.001, 0.01, 0.1, 1, 10, 100, 1000	Algorithm: auto, ball_tree, kd_tree, brute	Kernel: rbf, poly, sigmoid, linear	Max depth: 2, 3, 4, 5, 6	Max split features: auto, sqrt		Max depth: 3, 5
			Min leaf samples: 5, 6, 7, 8, 9	Max depth: 10, 20, … , 120		n_estimators: 30, 40
				Min split samples: 2, 6, 10		Learning_rate: 0.1, 0.01, 0.05
				Min leaf samples: 1, 3, 4		
				bootstrap: T, F		

Table 1: Hyperparameters chosen for each machine learning model

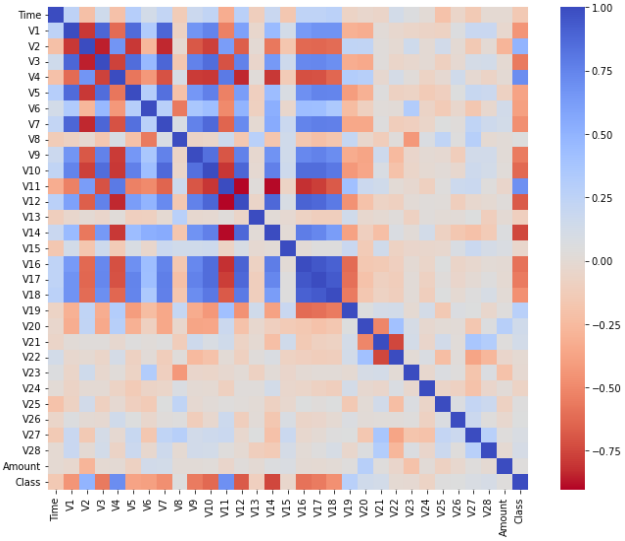

Figure 2: Heatmap to visualize the linear correlation between various features and target variables

Random Undersampling (RUS): The random undersampling (RUS) method to balance the dataset across both the classes was achieved by building a new dataset using all 492 fraud transactions, and randomly selecting 492 transactions from the non-fraud transactions. With the seaborn heatmap, we generate a correlation plot (Fig. 1) between features and the target, and found that the features 'V3','V7', 'V9', 'V10', 'V12', 'V14', 'V16','V17' have strong negative correlation while features 'V4', 'V11' have strong positive correlation with the target as shown in Fig. 2. Boxplots were plotted to visualize how well each of these variables separate the target independently, and found that the final features 'V3', 'V4', 'V10', 'V11', 'V12', 'V14', 'V16','V17' do a great job, and are strong predictors (Fig. 3). Based on [12], we removed the outliers in just these final features by using a basic validation check that removes any attribute value that falls either below the Interquartile range (IQR) from the 1st quartile or above 1.5 times from the third quartile. IQR is the difference between the 75th and 25th percentile values. Finally, the same models we chose for the baseline were tried out with GridSearch Cross Validation of 10 folds for a

variety of estimators, penalty, kernels, tree size (as shown in table 1). Results show that both random undersampled data models with and without outliers performed well compared to the baseline model. However, the RUS data model without outliers does not lead to any performance improvement and is similar to the RUS with no outlier treatment as shown in table 3.

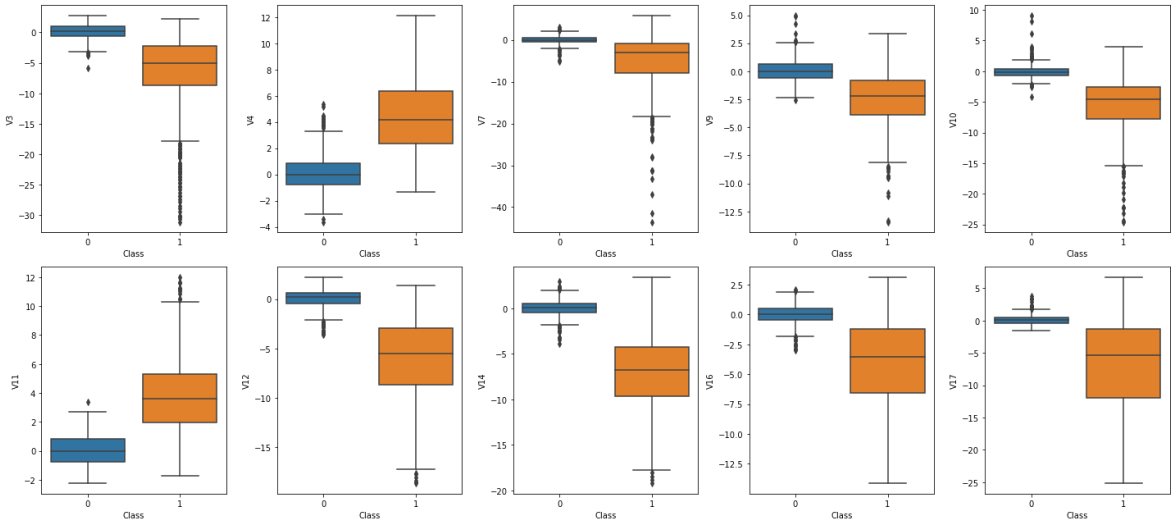

Figure 3: Boxplot to compare selected correlated features vs target variable

Random Oversampling (ROS): In the Random Oversampling (ROS) method, we first split the data into train and test and we get 227451 points in class 0 and 394 in class 1 for the training set. Then, we apply the random oversampler method from the imblearn package in Python3 to generate duplicate data points from class 1 to match the class 0 records. We found that the exact same features helpful for RUS were helpful for ROS. Again, here we implemented the two versions - with and without outlier removal. An interesting observation is that the ROS without outliers follows a distribution very close to the normal distribution compared to ROS with outliers as shown in Fig 4. We find a very significant performance drop in the ROS method with outlier removal as seen in table 3.

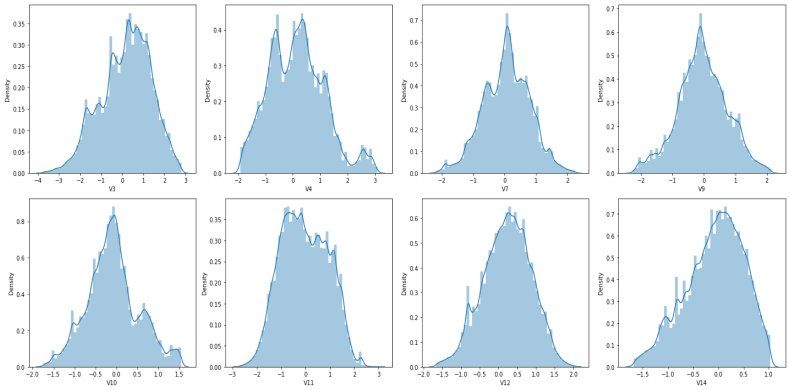

Figure 4: Distplot to visualize the density of feature distribution

SMOTE: SMOTE is another efficient technique for solving the Class Imbalance problems. It creates synthetic points from the minority class in order to reach an equal balance between the minority and majority class. It picks the distance between the closest neighbors of the minority class, and in between

these distances it creates synthetic points. SMOTE retains more information from the original dataset, since we do not delete any majority class data, unlike in random undersampling. Although it is likely that SMOTE will be more accurate than random under-sampling and oversampling, it takes much longer to build each model, since the amount of training data is more. The summary of the performance results of SMOTE can be found in table 3.

Observations from previous sampling methods & correction: In our previous report, we were undersampling and oversampling the data before cross-validation. This directly influenced the validation set before implementing cross-validation and caused a "data leakage" problem. **We observed amazing precision and recall but in reality our data was overfit!** So to correct the mistake, we split the data into train and test after normalizing the features and all the processing such as RUS, ROS, outlier treatment were implemented on the training set only. To avoid reducing the minority class data points, we didn't perform outlier treatment for SMOTE as this was degrading the performance of the overall technique.

	Original Dataset	Random undersampling		Random oversampling		SMOTE dataset
	w/o any data pre-processing	With outliers	Outliers removed	With outliers	Outliers removed	With outliers
Fraud	492	492	409	227451		45489
Non-fraud	284315	492	475	227451		56862

Table 2: Summary of the total number of fraud and non-fraud data in each of the modeling technique used

Cost-sensitive XGBoost learning: Cost-sensitive learning was only performed on the original dataset because we utilized the class-imbalance parameter (scale_pos_weight) of the XGBoost model rather than sampling the data to improve results. We used the ratio of positive & negative class to adjust our model to penalize the wrong classification of positive class more than that of negative class.

Algorithm: We define the data split using a stratified split on 'Y' to preserve the distribution of positive and negative classes in training, test data from the original dataset: *train:test = 80:20.* Using *scale_pos_weight* as the ratio of the number of data points of the negative class to the positive class, which for this case was 577.87. We get reasonable improvements over models previously used - LR, DT, RF. The mean ROC AUC training score was 0.97638. Further, tuning the hyperparameters of this classifier with gridsearchCV, we generate the best possible cost-sensitive XGBoost classifier for an imbalanced class dataset.
auc_roc training score = 0.976607
auc_roc testing score = 0.9506721371722577

Summarizing roc_auc scores of all the models for training and cross-validation:

Models	Logistic		KNN		SVM		DT		RF		Cost-sensitive XGBoost	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Baseline model (data without any pre-processing)	0.970	0.816	0.924	0.90	0.930	0.846	0.878	0.870	0.944	0.880	**0.977**	**0.951**
Undersampled data (without outliers removed)	0.971	0.940	0.964	0.93	0.976	0.930	0.916	0.890	0.979	0.938		
Undersampled data (with outliers removed)	0.975	0.930	0.960	0.93	0.970	0.930	0.890	0.890	0.970	0.930		
Oversampled data (without outliers removed)	0.990	0.920	0.990	0.90	0.990	0.850	0.990	0.850	1.000	0.890		
Oversampled data (with outliers removed)	0.920	0.920	0.990	0.50	0.990	0.500	0.990	0.500	1.000	0.500		
SMOTE (mean scores)	0.936	0.903	0.743	0.54	0.940	0.890	0.933	0.904	0.914	0.904		

Table 3: Final AUC scores across train and test data for each model and sampling method selected

Conclusion and Discussion:

We have compared the performance of different Machine learning models on our imbalance dataset using three resampling methods (random oversampling, undersampling and SMOTE). All the methods performed well compared to the baseline model in terms of AUC. Out of all the models, Cost-sensitive XGBoost was found to have the best performance of 95% on the test set. We believe that the precise tuning of the "scale_pos_weight" hyperparameter has helped improve its performance. The best parameters of the model were: 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 40, 'scale_pos_weight': 577. As the official documentation explains [14], scale_pos_weight controls the balance between positive and negative weights, and is particularly useful for unbalanced classes. This helps tune the loss function, adjusting whether to reward an incorrect prediction and give a large penalty on incorrect prediction for minority class. The weighted penalty is concluded as the reason for best performance of the cost-sensitive XGB model.

Another observation we made was that removing outliers actually reduced the overall performance. This may be attributed to the cut-off (threshold) value we had defined (removal of points that are less than 1.5 times the interquartile region from the 1st quartile and are more than 1.5 times the interquartile region from the 3rd quartile). We did not much experiment with the optimal level of threshold and fixed it to 1.5 as an input from our literature review. Because of this, we may have been excluding points that are not anomalies or noise, resulting in performance degradation. In future, we could explore other possibilities of outlier treatment.

References

1. Credit Card Fraud Detection https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
2. National Council of Identity Theft Protection https://identitytheft.org/credit-cards/statistics/
3. Data analysis techniques for fraud detection https://en.wikipedia.org/wiki/Data_analysis_techniques_for_fraud_detection
4. Akhilomen, John. "Data mining application for cyber credit-card fraud detection system." Industrial Conference on Data Mining. Springer, Berlin, Heidelberg, 2013.
5. Lim, Kha Shing, Lam Hong Lee, and Yee-Wai Sim. "A review of machine learning algorithms for fraud detection in credit card transactions." International Journal of Computer Science & Network Security 21.9 (2021): 31-40.
6. Saia, Roberto, and Salvatore Carta. "A frequency-domain-based pattern mining for credit card fraud detection." International Conference on Internet of Things, Big Data and Security. Vol. 2. SciTePress, 2017.
7. Lek, Monkol, et al. "Data mining prototype for detecting ecommerce fraud." ECIS 2001 Proceedings (2001): 60.
8. Chee, Chin-Hoong, et al. "Algorithms for frequent itemset mining: a literature review." Artificial Intelligence Review 52.4 (2019): 2603-2621.
9. Xu, Hongzuo, et al. "Deep Isolation Forest for Anomaly Detection." arXiv preprint arXiv:2206.06602 (2022).
10. Pang, Guansong, Chunhua Shen, and Anton van den Hengel. "Deep anomaly detection with deviation networks." Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019.
11. Sarkar, Tushar. "XBNet: An extremely boosted neural network." Intelligent Systems with Applications 15 (2022): 200097.
12. **Longadge, R., & Dongre, S. (2013). Class imbalance problem in data mining review. *arXiv preprint arXiv:1305.1707*.**
13. Krawczyk, B. Learning from imbalanced data: open challenges and future directions. Prog Artif Intell 5, 221–232 (2016).
14. XGBoost Parameters https://xgboost.readthedocs.io/en/latest/parameter.html