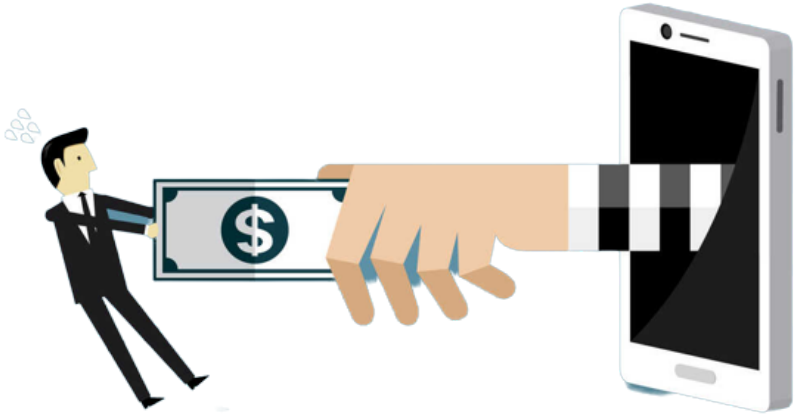
Predicting Mobile Transaction Fraud Using XGBoost Ensemble Model



Created by: Jason Ortiz, Kshitiz Badola, Sobnom Mostari

Motivation:

As technology advances, it benefits both those seeking to exploit others and those seeking to defend themselves. Machine Learning has enabled transactions to be deemed as fraudulent or not at the time of the transaction, potentially saving people a lot of money. You can't have a model that performs poorly in terms of both accuracy and time.

Data & Its Source:

We can obtain data similar to otherwise extremely sensitive data by using the PaySim financial money simulator Kaggle dataset. The dataset has 6,362,620 rows and 11 features (10 decisions and 1 target). Before we can do any modeling, we must first understand the data we are working with. We can obtain data similar to otherwise extremely sensitive data by using the PaySim financial money simulator Kaggle dataset. We can see how the models we've learned about operate in a production-like setting by using this synthetic dataset. This is far more useful than many of the "toy" datasets available on Kaggle.

Exploratory Data Analysis (EDA):

The first feature to grasp is the target feature: isFraud, of which there are only 8,213 instances in the whole dataset (6,000,000+ rows). This accounts for only about 0.129% of the data, making the dataset severely unbalanced.

Using the sorts of transactions that truly involve fraud can help with the data's imbalance, reducing the overall number of instances to 2,770,409 (43.5% of the original data) while

retaining all 8,213 fraud transactions. This takes the total number of fraudulent transactions to 0.296% of the dataset.

Data Preprocessing:

Some features were eliminated, such as isFlaggedFraud. Other features, such as type and nameDest, were encoded. Some features, such as isFlaggedFraud, have been deactivated: With only 16 instances in the dataset, it was a sparse feature, accounting for less than 0.0001% of the total. Other features, such as type and nameDest, were encoded, these features are made up of object data types that can be encoded (Label Encoding), in this case a String. This enables us to leverage those attributes with the various Machine Learning models we'll be employing. We didn't use One_Hot_Encoding since the three string characteristics had far too many distinct values. This would have resulted in a vast number of additional features and far too much storage space for sparse features. In preparation for some of the baseline models to be compared with the XGBoost model we utilize, separate scaled data had to be prepared in addition to encoding. As seen in HW 1, not all models require scaled data, with KNN preferring scaled data whereas Naive Bayes-based models do not. The "useful_data" is the data after we deleted the superfluous transaction 'types'. In order to prepare for some of the baseline models, separate scaled data had to be prepared.

Ensemble Learning

Ensemble Learning is a technique that combines multiple models to improve the overall performance and accuracy of a predictive model, with two main types being Bagging and Boosting. Bagging decreases variance and improves stability by training multiple models on different subsets of the training data. Boosting, on the other hand, reduces bias and improves accuracy by sequentially training models that focus on correcting the errors of the previous models. XGBoost is an ensemble learning technique that combines weak models, usually

decision trees, into a stronger model that can handle large datasets with high-dimensional features and achieve high prediction accuracy in various applications such as regression, classification, and ranking problems.

XGBoost or Extreme Gradient Boosting belongs to a group of gradient boosting algorithms like AdaBoost, Gradient tree boosting, etc. XGBoost sequentially builds a series of decision trees, where each following tree tries to correct the errors made by the previous tree. During the training, the algorithm calculates the gradient of the loss function with respect to the predictions of the previous tree, and uses it to update the weights of the data points. So that the subsequent tree can focus on the instances that were misclassified by the previous tree. So, in a nutshell XGBoost is simply an improved and refined form of gradient boosting. Since it is well optimized, it requires less CPU and can attain better results within a shorter span of time.

Auto Pruning, cache optimization, parallelization, and regularization are some of the features that contribute to XGBoost's reliable performance. Auto pruning controls the tree structure in order to avoid overfitting and improve model quality or robustness. Parallelization allows the training to use all CPU cores, making it faster and more efficient. Cache optimization caches intermediate calculations and statistics, allowing for more accurate predictions. Regularization reduces overfitting by reducing the coefficient estimates towards zero, preventing the formation of very complicated models.

Baseline Models

Before training our XGBoost model we first decided to train multiple other models on our data. We decided on training and testing some of the models learned about in this course: K-Nearest-Neighbors, Random Forest, Logistic Regression/SGDClassifier, as well as Gaussian & Bernoulli Naive Bayes classifiers. When deciding models there are 3 of us so we can afford to test more models. For we've decided on training and testing: K-Nearest-Neighbors, Random Forest, Logistic Regression/SGDClassifier, as well as Gaussian & Bernoulli Naive Bayes classifiers.

We tested each model using accuracy, confusion matrices, precision & recall. We streamlined this process when we noticed that we were each doing a similar process in model building for each model being used. We combined the model training and "rating" into a single feature that took in X_training, X_testing, y_training, y_testing data & a previously instantiated model (with the parameters wanted).

Results

In the chart below you can see the top three models which performed the best in each Metric, with Blue = **#1** performer & red = Top 3 performers. The True Positive and True Negative are both in relation to the total number of data points (with the actual total fraud amount being **0.29%**:

\ Metric Model \	Accuracy	True Positive	True Negative	Precision	Recall
K-Nearest Neighbors (K=7)	99.83%	0.13%	99.70%	91%	46%
Random Forest	99.93%	0.22%	99.71%	98%	78%
Logistic Regression	99.84%	0.14%	99.70%	91%	48%
SGDClassifier	99.81%	0.10%	99.71%	99%	34%
GaussianNB	98.49%	0.11%	98.38%	8%	40%
BernoulliNB	99.81%	0.14%	99.67%	77%	50%
XGBoost(Tree-based)	99.95%	0.24%	99.71%	98%	86%

As you can see the XGBoost outperformed all other models in every metric except for precision, in which it tied for an extremely close 2nd place at 98%.

References

- Seif, G. (2022, February 11). A beginner's guide to xgboost. Medium. https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7
- Gupta, P. (2017, November 16). Regularization in machine learning. Medium. https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a
- Hachcham, A. (2023, April 25). XGBoost: Everything you need to know. neptune.ai. https://neptune.ai/blog/xgboost-everything-you-need-to-know
- Lopez-Rojas, E. (2017, April 3). Synthetic financial datasets for fraud detection. Kaggle. https://www.kaggle.com/datasets/ealaxi/paysim1