**Detecting Fraudulent Credit Card Transactions**

**(Using Machine Learning and Anomaly Detection)**

**Final Report**

**Project Details / Requirements**

**Goals:** The goal of this project is to build anomaly detection models to identify and flag fraudulent credit card transactions. Fraudulent activities are rare and must be accurately detected to minimize financial losses while reducing false positives to maintain user trust.

**About The Dataset:** [**Kaggle's Credit Card Fraud Detection Dataset**](https://www.kaggle.com/datasets/kartik2112/fraud-detection)

This dataset contains anonymized credit card transactions, the data is simulator generated, each entry labeled as either fraudulent or non-fraudulent. The data includes numerical features derived from the source’s information to protect privacy. These features help identify patterns and relationships for fraud detection. The dataset is pre-divided into training and testing files, making it suitable for machine learning tasks like anomaly detection. It is commonly used to test models that handle imbalanced data, as fraud cases are much rarer than normal transactions.This project aims to detect fraudulent transactions by building an anomaly detection model using credit card transaction data. The goal is to experiment with techniques such as autoencoders and isolation forests to optimize fraud detection accuracy. Utilizing Jupyter Notebooks to analyze transaction patterns and identifying anomalies, the model will flag suspicious activity effectively. This work will also address challenges like class imbalance and ensure the model's performance is evaluated using metrics such as precision, recall, and F1-score.

**Features:** [‘trans\_date\_trans\_time', 'cc\_num', 'merchant', 'category', 'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip', 'lat', 'long', 'city\_pop', 'job', 'dob', 'merch\_lat', 'merch\_long', 'is\_fraud'

**Class Imbalance**: Only 0.524% of the transactions are labeled as fraudulent.]

**Requirements**

* Use machine learning algorithms to analyze transaction data and detect fraud (in credit card transactions)
* **Selected Algorithms** (8):
  + Isolation Forest, Random Forest, Naive Bayes, K-Means Clustering, Decision Tree, Linear Regression, Logistic Regression, K-nearest-neighbors (KNN)
* Address class imbalance inherent to fraud detection datasets.
* Experiment with dimensionality reduction techniques like PCA (Primary Component Analysis) to improve model performance or scalability (in the case of KNN).
* Evaluate model performance using metrics like ROC curves, AUC, precision, recall, scatter plots (for K-means clustering), and F1-score.
* **Tools:**
  + Jupyter Notebooks
  + Python Libraries: Numpy, Pandas, Scikit-learn, Matplotlib
  + For convenience, we opted to use sea born over matplotlib for a few graphs in our Data Exploration.ipynb notebook

**Developed Methods & Algorithms**

**Dataset Pre-Processing**

Because the dataset is already split for us, we decided to join it back into the df variable, and add a source dummy column denoting if the row comes from the test/train dataset. This is done so that when we perform any modifications, it will apply to both of the datasets. After pre-processing is complete, we remove the source column and separate them back into their respective sets via our *resplit\_train\_test* method.

**Feature Engineering:**

To start, we looked for any features we could immediately drop that provided redundant information. Taking a glance, we noticed two different forms of dates *unix\_time* and *trans\_date\_trans\_time*. We opted to drop unix\_time since it wasn’t as easily interpretable. Plus we found an easier way to decompose the data into separate columns (year, month, day, hour, minute, and seconds). Other than the date, the trans\_num was unique in each row, so served as an id, and the *‘Unnamed: 0’* feature was the same as the row’s index. We continued by looking for any simple features we could manually encode, but we only had one binary feature *gender*.

Next we used one-hot encoding for the features with few unique values such as the transaction *state* & *category* feature. The rest of the categorical features (bar the transaction date) would end up being label encoded, due to all having over 600 unique values (high cardinality features).

**Handling Class Imbalance:**

Now that all features are numerical, we move on to handling the dramatic imbalance in our data. To do this we used SMOTE (Synthetic Minority Oversampling Technique) to resampling the minority class (fraudulent transactions) to 20% of the majority class size. The Resulting class distribution is 1,289,169 non-fraud vs. 257,833 fraud instances (from an original 7,506 fraud cases from the training set).

**Scaling / Normalization:**

An issue came up when we were trying to do normalization before handling the class imbalance. SMOTE was complaining that it couldn’t use continuous values, and to keep to integer data instead. This happened due to normalization changing our values into ratios. However, changing the order around quickly resolved this. Now the models are ready for training.

**Model Evaluations:**

1. **Isolation Forest**: Anomaly detection focused on rare events.

* Standard implementation:
* precision recall f1-score support
* 0 0.83 1.00 0.91 1289169
* 1 1.00 0.00 0.00 257833
* accuracy 0.83 1547002
* macro avg 0.92 0.50 0.45 1547002
* weighted avg 0.86 0.83 0.76 1547002
* AUC: 0.79

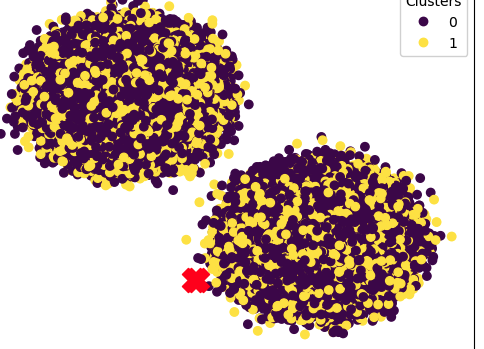
1. **Random Forest**: Ensemble model effective for classification tasks.

* Standard implementation: ('Accuracy Score: ', 0.9768293270388747)
* Area Under Curve: 0.9888227217915413
* ROC: 0.9888227217915413
* Feature Importance Analysis:
* 2 amt 0.431871
* 25 category\_grocery\_pos 0.089056
* 32 category\_shopping\_net 0.077170
* 19 hour 0.062851
* 29 category\_misc\_net 0.040408
* PCA variant:
  + ('Accuracy Score for pca variant: ', 0.9316606652536249)
  + Area Under Curve for pca variant:
  + 0.913017302660426

1. **Naive Bayes**: Probabilistic model assuming feature independence.

* Standard implementation:
  + ('Accuracy Score: ', 0.8840277108723927)
  + Area Under Curve: 0.7781566255746715
* PCA variant: predictions are **very** fast
  + ('Accuracy Score for pca variant: ', 0.9161794281998169)
  + Area Under Curve for pca variant: 0.8698878587263572

1. **K-means Clustering**: Unsupervised learning for grouping similar transactions.

* Standard implementation:
  + ('Accuracy Score: ', 0.5855201358446939)
  +  yellow is fraud, purple is non-fraud

1. **Decision Tree**: A simple, interpretable classification model.

* Standard implementation:
  + ('Accuracy Score: ', 0.9849899441206224)
  + Area Under Curve: 0.9610356523403298
  + Feature importance analysis:
    - 2 amt 0.708768
    - 25 category\_grocery\_pos 0.068402
    - 23 category\_gas\_transport 0.023988
    - 19 hour 0.023969
    - 13 dob 0.022158
* PCA variant:
  + ('Accuracy Score for pca variant: ', 0.9106788019654126)
  + Area Under Curve for pca variant: 0.7746889149587899

1. **Logistic Regression**: Baseline linear classifier for binary outcomes.

* Standard implementation:
  + ('Accuracy Score: ', 0.9691218266775856)
  + Area Under Curve: 0.9739202148054873
* PCA variant:
  + ('Accuracy Score for pca variant: ', 0.915121152271304)
  + Area Under Curve for pca variant: 0.8649035262149508

1. **K-Nearest Neighbors (KNN)**: Instance-based learning for classification.

* PCA variant only, as the dataset was too large to run without dimensionality reduction:
  + Accuracy Score: 0.8953360590587215
  + Area Under Curve: 0.7308814624124019

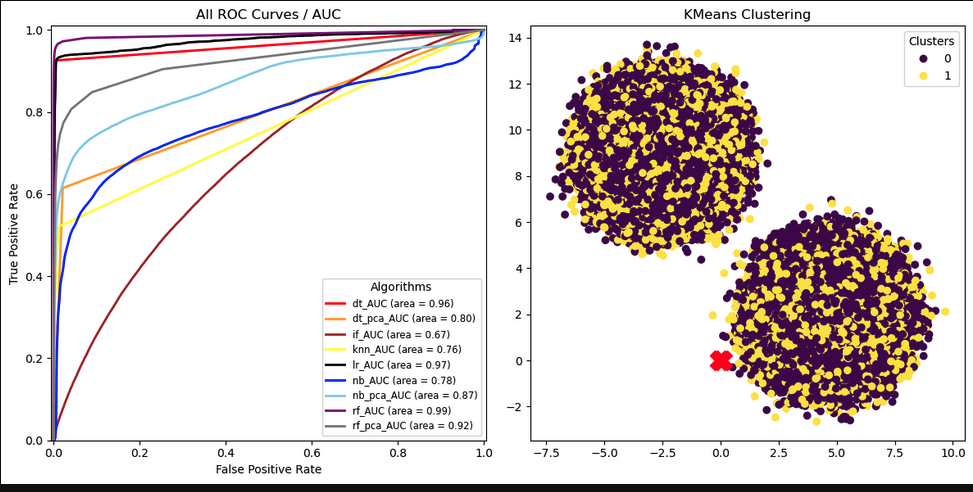
1. **Linear Regression**: Initially tested but unsuitable for binary classification.

* Standard implementation:
  + Mean Squared Error (MSE): 0.04908494807844559
  + R-squared: 0.64658667137428
  + Feature importance analysis:
    - 25 category\_grocery\_pos 0.193659
    - 2 amt 0.150396
    - 23 category\_gas\_transport 0.146813
    - 32 category\_shopping\_net 0.136448
    - 33 category\_shopping\_pos 0.116231

**Dimensionality Reduction via PCA (Principal Component Analysis):**

We applied PCA to reduce feature dimensions and assess its impact on algorithm performance, and observed whether or not PCA improves computational efficiency or if it could improve model accuracy (in one of the cases it did, whilst being more performant).

**Final Results**

****

As it turns out, random forest was the best one (non-pca) for detecting fraudulent credit card transactions. And based on random forests’ feature importance analysis, the most important features, in order:

2 amt 0.431871

25 category\_grocery\_pos 0.089056

32 category\_shopping\_net 0.077170

19 hour 0.062851

29 category\_misc\_net 0.040408

So it would appear you can use the transaction amount with a degree of accuracy of 43% to classify whether or not a transaction is fraudulent, as well as to a less reliable extent, if the transaction was an internet or grocery purchase. Further exploration of feature engineering and model refinement could improve classification accuracy and strengthen fraud prevention systems.

**Team Member Contributions**

| **Steven Gonzales** | Matplotlib visualizations, Project Organization, KNN Algorithm |
| --- | --- |
| **Evan Peraza** | Preprocessing (imbalance handling) and Data Exploration |
| **Weihao Liu** | (Did not contribute to overall project or try to keep in contact with group) |
| **Yongkang Liu** | Evaluation: Isolation Forest + K-means Clustering + Linear Regression |
| **Ramon Ulloa** | Evaluation: Decision Tree + Random Forest + Naive Bayes + Logistic Regression |