CC Fraud Detection - A Supervised Learning Model

CSCA-5622 Introduction to Machine Learning - Supervised Learning

The focus of this project is the prediction of fraud credit card transactions, given a set a features collected during the time of the transaction. For card-not-present transactions, such as online e-commerce, the amount of fraudulent transactions has a direct impact on a company's profitability. Without the customer present, it is much easier for a bad actor to utilize false or stolen credit card data to purchase products. Given the current trends towards fast delivery, the product can be in the bad actor's hands before a bank or credit card holder is aware that the credit card or bank account was used fraudulently.

The dataset is provided through a Kaggle competition, hosted by the IEEE Computational Intelligence Society (IEEE-CIS Fraud Detection, n.d.). The dataset has two components. A transaction file is provided that has the historical credit card and transaction related attributes, which are mostly obfuscated or anonymized, with the associated IsFraud results. An identity file is provided for some transactions, where additional attributes were collected during the transaction by a fraud protection system and digital security partners. Much of this data is also obfuscated or anonymized.

The goal is to build a machine learning model, based on the historical data, that can predict a transaction to be fraudulent (IsFraud = 1) or not (IsFraud = 0). There is also a balance that should be achieved between catching fraudulent transactions and bad customer experiences, when a valid transactions is identified as fraud. Properly identifying fraudulent transactions directly impacts revenue loss, but a bad customer experience can result in revenue loss from a different perspective.

Since we have historical data and a known set of labels, we can focuses on a supervised learning model approach. A good model also needs to balance its effectiveness against performance, because it would ultimately be placed in-line with the credit card transactions, to predict fraud during the checkout. A model that takes too long to run would result in customer abandonment during the transaction.

Table of Contents

Installation

Follow these steps to setup the project locally.

1. Clone the Repository

```
git clone https://github.com/jfroggatt/cc_fraud_detect.git
cd cc_fraud_detect
```

2. Create a Virtual Environment (Optional but Recommended)

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Install Dependencies

Install all required packages from `requirements.txt` :

```
pip install -r requirements.txt
```

4. Launch Jupyter Notebook

```
Jupyter Notebook
```

Then open the `cc_fraud_detect.ipynb` file from the Jupyter interface.

Requirements

- Python 3.12
- Jupyter Notebook
- See `requirements.txt` for the full list

Usage

- Run each cell in sequence

- Modify parameters or data to explore custom scenarios
- Refer to comments for explanation of each step

Project Notebook

The project code and process is maintained and documented in the Jupyter Notebook file `cc_fraud_detect.ipynb` .

Data Source

The dataset used for this project was provided by the IEEE Computational Intelligence Society (IEEE-CIS Fraud Detection, n.d.) competition on Kaggle. The dataset files are too large to maintain within my cc_fraud_detect GitHub project respository, so the competition project archive is downloaded to the local working directory, and the `train_transaction.csv` and `train_identity.csv` files are extracted. No other files are needed from the competition project archive. The download, extraction, and loading of the dataset files is accomplished in the `cc_fraud_detect.ipynb` project notebook, under the *Download Dataset* section.

Data Description

A detailed description of the dataset is provided by the competition host, Vesta Corporation, on a dedicated post on the IEEE-CIS Fraud Detection Discussion Board. Per the post:

Transaction Table: `train_transaction.csv`

- TransactionDT: timedelta from a given reference datetime (not an actual timestamp)
- TransactionAMT: transaction payment amount in USD
- ProductCD: product code, the product for each transaction
- card1 - card6: payment card information, such as card type, card category, issue bank, country, etc.
- addr: address
- dist: distance
- P_ and (R__) emaildomain: purchaser and recipient email domain
- C1-C14: counting, such as how many addresses are found to be associated with the payment card, etc. The actual meaning is masked.
- D1-D15: timedelta, such as days between previous transaction, etc.
- M1-M9: match, such as names on card and address, etc.
- Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.

Categorical Features:

- ProductCD
- card1 - card6
- addr1, addr2
- P_emaildomain
- R_emaildomain
- M1 - M9

Identity Table: `train_identity.csv`

Variables in this table are identity information – network connection information (IP, ISP, Proxy, etc) and digital signature (UA/browser/os/version, etc) associated with transactions. They're collected by Vesta's fraud protection system and digital security partners. (The field names are masked and pairwise dictionary will not be provided for privacy protection and contract agreement)

Categorical Features:

- DeviceType
- DeviceInfo
- id_12 - id_38

Data Details

The `train_transaction.csv` dataset consists of 590,540 historical credit card transactions, with the target label 'isFraud', and 393 features. ProductCD, card4, card6, P_emaildomain, R_emaildomain, and M1-9 are nominal data types. The remaining 4XX features are numerical, with most containing low order integer or binary values.

The `train_identity.csv` dataset consists of 144,233 transactions with 40 identity related features. The link between the transaction and idenitity files is the TransactionID attribute. The features are a mixture of numerical and nominal features.

Exploratory Data Analysis

The following steps were taken in the Exploratory Data Analysis:

1. Import .csv files, describe the dataset
2. Evaluate missing data for the features
3. Impute missing data
4. Perform correlation anaysis
5. Perform Principal Component Analysis (PCA)
6. Review feature label categorization
7. EDA Conclusion

The output of steps 1 through 6 can be reviewed in the `cc_fraud_detect.ipynb` notebook, under the *Exploratory Data Analysis* section.

EDA Conclusion

Based on the evaluation performed, the following was identified:

Primary Difficulties

- Between the Transaction and Identity datasets, there are 433 features to determine the single IsFraud label.
  - This is a large number of features, which will preclude using some models or approaches
- 186 features contain data for less than 50% of the transactions.
  - over 43% of the features are missing data for a significant portion of the transactions.

    - This will affect decisions for using these attributes and imputing missing data.
- Target label imbalance
    - Of the 590,540 transactions, only 20,663 are labeled as isFraud = 1. This is just under 3.5%
    - Having such a large imbalance on the isFraud is False will affect the meaning around evaluations of model accuracy
- Identity data only available for 144,233 transactions (24.2%)

Decisions

- Features missing data for more than 50% of the transactions
    - Evaluating the correlation of those features to the IsFraud outcome, showed no significance for any of the features.
    - As such, these features will be targeted for feature reduction.
- Principal component analysis
    - Reducing the feature set over iterations of 100, 200, and 400 showed no identified improvement with feature correlation.
    - No features identified for additional feature reduction.
- Identity data
    - while the identity data was available for only 24% of the transactions, the additional features my provide additional value to the models.
    - identity data will be utilized for the model evaluation
- Data set scaling
    - standardized scaling of the dataset features only improved the results of the Logistic Regression Classifier.
    - The dataset values will be used as is and not scaled.
- Categorical features
    - review of the top 10 categories for each feature showed a reasonable classification balance, while not over inflating the 'other' categories.
    - encoding of each categorical feature will utilize the top 10 categories by membership, with the remaining values in a single category.

Feature Engineering

Given the size of the data sets, the number of features missing values, and the number of transactions that do not have identity related data, the following feature reduction and engineering steps were taken:

Remove features with low data population

Based on the evaluations of the features that are populated for less than 50% of the data, these features will be not be evaluated as part of the initial modeling and will be removed from the data set.

Impute missing values

For the initial pass, we will utilize a default value for the missing data attributes. Without knowledge of the purpose or associated values in the current dataset, imputing mean, median, or mode defaults could have an unkown impact on the accuracy, and would likely require focused testing and evaluation.

Based on the data set evaluations, the following will be used:

1. For integer numerical features, the default will be 1 less than the minimum value in the feature range.
2. For float numerical features, the default will be 1.0 less than the minimum value in the feature range.
3. For nominal features, the default will be 'NotDefined'

Define categorical features and encode top 10 for each

The data provider noted the expected categorical features within the dataset, so we will have these applied appropriately. Based on the initial evaluations, categorical features will be OneHotEncoded for the top 10 categories initially. This may be adjusted based on the performance and accuracy.

Reduce data footprint

To improve the memory footprint and processing time, the numeric data types will be updated from 64-bit to 32-bit numerical values, i.e, int32 and float32.

Split dataset for Training and Test

Split the dataset to provide 80% for training and 20% for testing. All data split operations, including the k-fold statistical training is stratified with the IsFraud label to ensure that a consistent ratio is maintained between IsFraud = 0 and IsFraud = 1 transactions, since it is a highly unbalanced set, with most transactions being IsFraud = 0.

Feature scaling

For one aspect of the model evaluations, the features will be scaled to determine how it affects each model being tested.

Models

For the modeling, we will use evaluate multiple models to determine which provides the best accuracy. Once the best model has been identified, we will then examine tuning performance to achieve the highest accuracy for that model.

The models that will be evaluated for this dataset:

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. LightGBM (mention in lecture but not covered in course)

I decided not to use AdaBoost, Gradient Boost, or SVM (other models covered in lectures), because the performance is a problem with such a high number of parameters and observations, where the models selected would work better.

Evaluation Criteria

For the evaluation criteria, the metrics that are important for fraud detection on credit card transactions are recall, precision, and F1 score.

1. Recall: maximizes catching fraud cases, but possibly with a higher risk of false positives.

2. Precision: minimizes false positives, which are a bad experience for valid transactions, but may have a financial impact for missed true positives.
3. F1 Score: provides a balance between the two

Accuracy is not necessarily as good a measure for these models, given that the fraud rate can be pretty low over the total number of transaction.

Process

Each of the models were run against the non-standardize and standardized datasets. The results were evaluated based on the Precision, Recall, F1 Score, and Accuracy for the predictions against the test data set. A confusion matrix was generated for each of the results, along with a PR-ROC AUC Curve. The PR-ROC AUC Curve was chosen over the standard ROC AUC Curve, because the focus was on the Precision and Recall balance of the models, over the Accuracy score.

The process and results can be found in the *Model* section of the `cc_fraud_detect.ipynb` notebook.

Model Selection

Based on the PR-ROC AUC curve, LightGBM out-performed the other models in results as well as speed, with Logistic Regression performing the worst across the board. The Logistic Regression model that was run using standardized features approached the accuracy of the other models more significantly than the one running against the standard dataset; however, the speed of the model was the slowest overall by far. For the LightGBM models, the version running on the non-standardized dataset performed slightly better, so we will use that for the final modeling.

Hyperparameter Tuning

After selecting the LightGBM model for the final predictions, tuning was performed using the Sklearn GridSearchCV function. An arrange of parameters was provided to determine the Hyperparameter configuration that resulted in the best prediction performance.

GridSearchCV was configured to perform a 3-CV K-Fold, stratified, and shuffled comparison across the hyperparameter options.

Final Results

Based on the GridSearchCV process, the hyperparameter tuned model was run against the test data set. The tuned model prediction results outperformed all model runs performed during the evalation phase.

The final predictions also provided a decent balance between maintaining a good customer experience with less false positives, at 0.14%, while catching ~54% of the fraud transactions.

At this point, the results would be presented to the business to determine if the model performed adequately or if the balance between Precision and Recall should be adjusted to further reduce the loss due to misclassified fraud transactions.

Prediction Results

The result from the final LightGBM Classifier model was:

- True Negatives = 113,810 : transactions correctly classified as Not Fraud
- True Positives = 2,225 : transactions correctly classified as Fraud
- False Positives = 165 : valid transactions misclassified as Fraud (0.14% of valid transactions = good customer experience overall)
- False Negatives = 1,908 : fraud transactions misclassified as Not Fraud (46.17%)

Next Steps

If the balance between the customer experience, for customer transactions misclassified as fraud, is too high for the loss generated by the misclassified Fraud transactions, then additional tuning exercises could be taken to try and improve the outcome. Based on the initial EDA, feature engineering, and model selection, I would take the next course of actions:

1. Approach to impute missing values
    - The current approach was setting a constant, based on the minimum value in the current range, for numerical features, and 'NotDefined' for nomial features.
    - A couple alternative approaches could be:
        - mean, median, or mode for the values
        - ML model to impute the values based on other feature values, which could be time consuming without more domain knowledge about the specific features
2. Feature reduction
    - Perform additional methods for identifying and reducing features or run models against iterations of PCA reductions
    - Remove features based on attribute fill rates other than 50%
3. Deeper dive on tuning hyper-parameters
    - Iterate through additional combinations of hyper-parameters for the LightGBM Classifier to determine if there is additional benefit to be achieved
4. Try additional models
    - Other models, such as XBoost, CatBoost, or ExtraTree, Neural Networks
    - Run the Logistical Regression Classifier with a different solver, e.g., 'saga'

License

Acknowledgements

- IEEE-CIS Fraud Detection competition
- IEEE-CIS Fraud Detection Data Description
- Vesta Corporation

    - provided the competition project dataset files
- NT Project Summary: Part 3
    - provided some starting parameters for evaluating LightGBM
- readme.so Editor
    - a handy online README.md editor by Katherine Oelsner