Credit Card Fraud Detection

Team Hotpot: Lanxin Kang, Xiwei Yang

May 7, 2022

-Executive Summary-

Decisions to be impacted:

Find features that may result in credit card fraud

Find and announce individuals and banks are more likely to be involved in credit card fraud

Suggest if the transaction is fraud or not by inputting certain information

Business value:

Reduce bad debts and loss

Improve customers' experience, protect customers' property and information safety

Protect bank's reputation

Enhance security system of transactions

Motivation:

We want to use machine learning techniques to build a model to sufficiently detect possible credit card fraud by inputting certain information like area and transaction amount. Our goal is to find customers and banks are more likely to be involved in credit card fraud.

-Data description/preprocessing

Data description:

Our dataset is from Kaggle, named "Credit Card Transactions Fraud Detection Dataset " which contains 21 feature variables and 1 binary target value. The dataset is imbalanced with 99% of target value = 0 and 1% of target value = 1 which means only 1% of transactions are actually a fraud.

Preprocessing:

We have a lot of categorical variables, to ensure better modeling, we did feature encoding. For "dob", we convert it into "age". For "trans_date_trans_time", we convert it into 3 new columns: month, broad time of day categories(like before_dawn), and weekend or not. We turn gender into 0/1 where 0 means female

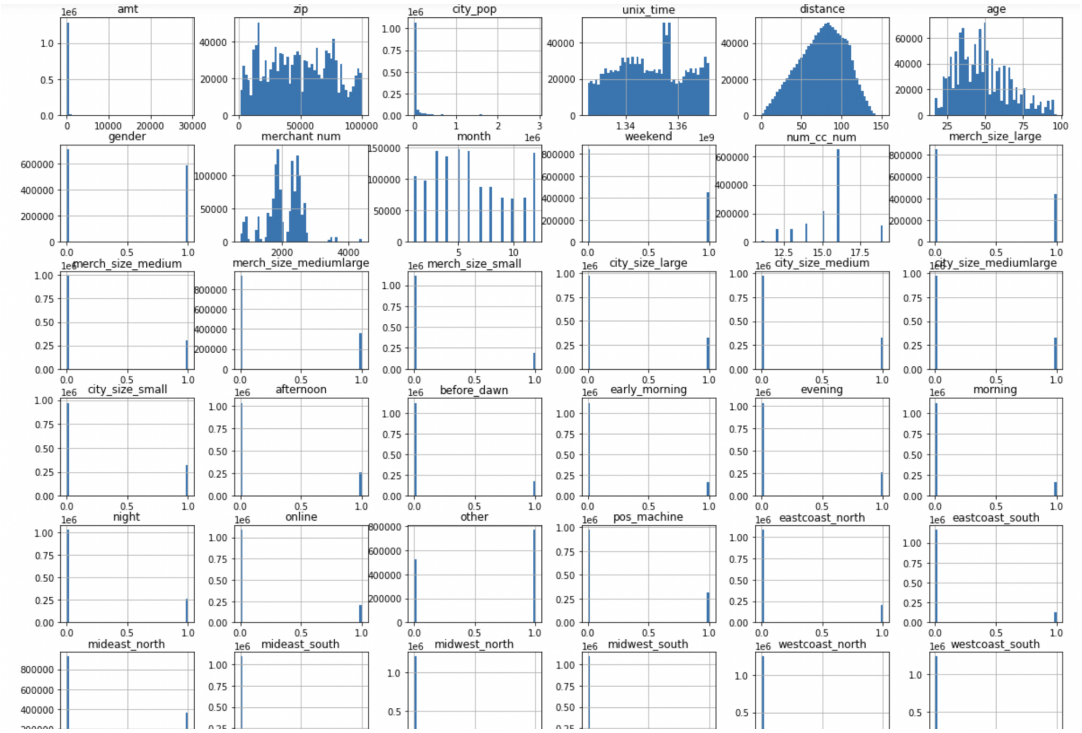
and 1 means male. We decide on city size by a four-quartile analysis of the city population. Cities with a city population less than the first quartile are classified as small. Cities with a city population equal to and larger than the first quartile but less than the medium are classified as medium. Cities with a city population equal to and larger than the medium but less than the third quartile are classified as medium-large. Cities with a city population equal to and larger than the third quartile are classified as large. The same strategy is used in deciding merchant size. We replace the credit card number with the number of digits it has. We encode “category” into “online”, “pos_machine”, and “other” based on the keywords found in merchant names. States are mapped into regions like west coast north and midwest south based on the geographical position. Also, we set user longitude and latitude as coordinate 1 and merchant longitude and latitude as coordinate 2 to derive distance. Then we do one-hot encoding to turn categorical features into binary variables so that we get an encoded dataset with all numerical values.

Then we split the train and test set with the test set size = 30%. Since the dataset is seriously imbalanced, we use the following 2 methods to resample the training set.

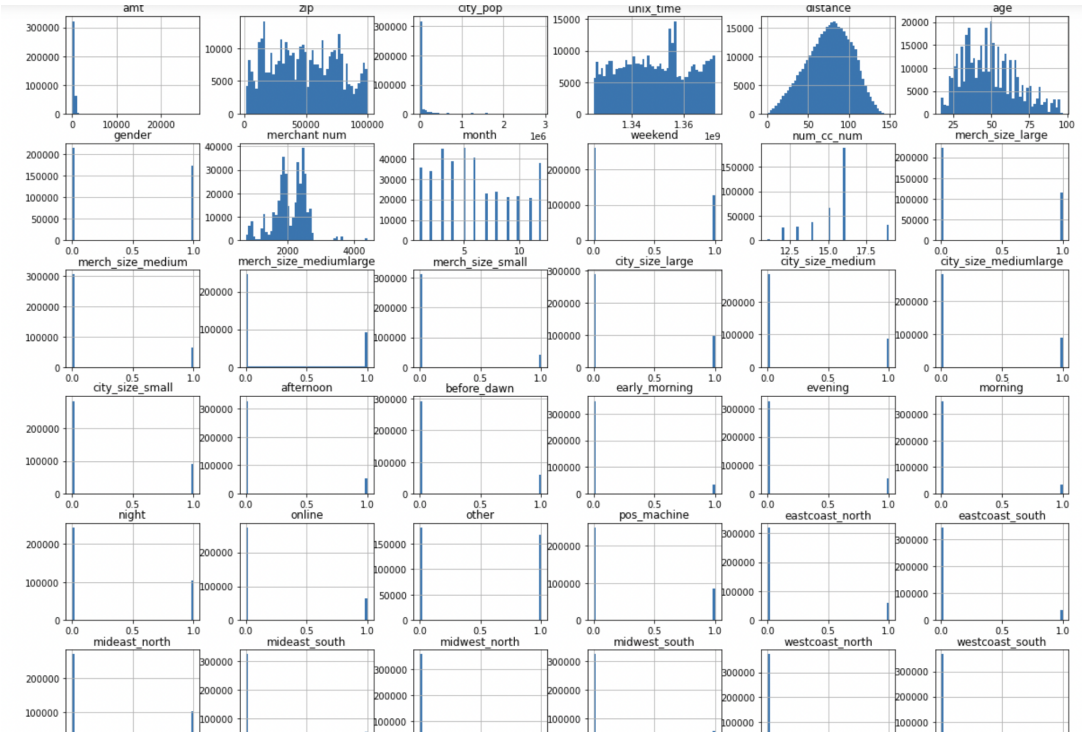
1. SMOTE oversampling minority + random undersampling the whole to ensure we have balanced data but are not losing too much information or not too hard to process. Then we get a balanced data with the target value:

```
0     257832
1     128916
Name: is_fraud
```

This is the histogram distribution of each attribute in the original encoded data frame



This is the histogram distribution of each attribute in the balanced data frame.



We can see that for each attribute, the shape and the ratio are maintained after balancing.

2. Undersampling

   as introduced below

-Modeling approach-

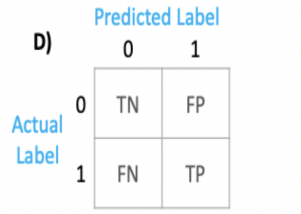
Undersample Part before doing feature engineering:

Random Forest Tree:

the training dataset was split into training and validation sets stratified, 2/3 and 1/3 respectively. Stratified splits maintain the original structure of the target. 89.28% is given by leave-one-out cross-validation. Leave-one-out cross-validation avoids bias in using a single validation set; this method fits a model that contains n -1 observations. Every n-1 observation will be tested against different single 1 observations. We tried a second method which is called stratified K-fold. As its name states, the test sample set contains approximately the same percentage of each target class as the complete set. The average cross-validation score is 89.56%. We satisfied the results and put features into a random forest model. The picture down below is the confusion matrix result:

```
: array([[2037,  108],
         [ 363, 1782]])
```

The default output of confusion matrix:



True Positive: 2037 and True Negative: 1782 show us the model performed approximately the same on two classes (0 or 1). Also, we tested on recall rate and accuracy scores: 83% and 89%, respectively.

After doing feature engineering:

One Hot encoding is the choice to fit feature engineering,

Repeated Edited Nearest Neighbours-Random Forest Tree:

The cross-Validation Average score improved by nearly 4% after comparing all features. Split training dataset stratified into a small training dataset and validation dataset, split ratio is ⅔, ⅓ respectively. And then throw them into two types of cross-validation methods; 92.77% generated by stratified cross-validation and 92.8% from leave one out of CV. Therefore, feature engineer characteristics of categorical values are successful and do impact building models. Move to the final step that tests on the entire encoded test dataset. The result is pretty satisfactory – 99.6% by an average cross validation score.

```
Cross Validation Mean Scores are [0.99614914 0.99614914 0.99614914 0.99640107 0.99620312 0.99613115
 0.99613115 0.99613115 0.99613115 0.99614907]
Average Validation Mean Scores are 0.9961725260006309
```

Although the score is high, looking at the confusion matrix and extracting information that performs best

```
array([[420064, 133510],
       [   606,   1539]])
```

True negative performs the best by default predicted probability threshold of 0.5, however, false alarm 133510 is high even if it exceeds true positive which is our detection probability. Precision rate and recall rate are easy to calculate from the confusion table. precision rate = True Positive / (True Positive + False Positive). Recall rate = True Positive / (True Positive + False Negative). To maximize precision and recall rate, threshold testing is needed. After several trials, 0.8 is the highest threshold to maximize predicted probability. The accuracy rate is 99.6% and the following updated confusion matrix table has a big change. Now, a random forest with a 0.8 probability threshold become a powerful tool to predict true negatives; it can predict safe transactions 99.6% correctly, and there's no miss to justify safe transactions being fraud transactions. However, there has a big problem with

detection probability, random forest with a 0.8 threshold doesn't work for detection probability.

```
array([[553574,      0],
       [  2145,      0]])
```

So here are more trials to find a balance between true negatives(0) and true positives (1). Finally, 0.43 is the best one to balance the results, the accuracy rate is 68% and the precision rate = is 84%.

```
0.6811805966684601

array([[376737, 176837],
       [   337,   1808]])
```

Therefore, tradeoff between detection probability and safe transactions show up in random forest tree. If other great models would be discovered for detection probability, then using ensemble learning to extract the great part would be the best combination function to detect credit card.

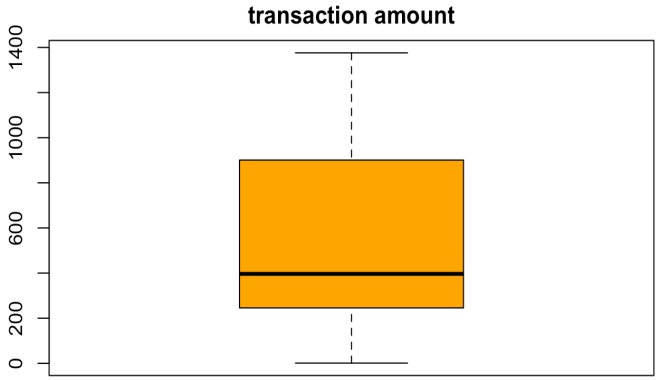
Let's move on to the importance of elements:

	feature	importance
0	amt	0.500
7	tran_Hour	0.162
20	night	0.082
17	before_dawn	0.045
32	merc_num	0.041
16	afternoon	0.023
19	evening	0.022
31	Top1_using_freq	0.021
15	morning	0.019
18	early_morning	0.016
3	birthyear	0.016
30	Top2_using_freq	0.007

As expected at the beginning, transaction amount is the important element to predict credit card fraud. Also, transaction amount is the the top node And here amt outliers

is helpful over here. The first numeric screenshot following is the fraud transactions outliers. The boxplot shows the range of fraud transactions. So if the pay attention on amt around $1300 that is likely to be maximized fraud transaction.

```
[1] 1334.07 1324.80 1312.98 1376.04
```



Random Undersample - Random Forest Tree:

Logistic Regression:

Logistic regression is typical binary regression, so we want to test it. Logistic regression performed well on testing true positives, but predicting the false negative is not as well as the random forest does. Logistic regression is more sensitive to predicting non-fraud transactions and less sensitive to detect fraud transactions, which we don't want it to.

```
array([[2054,   91],
       [ 533, 1612]])
```

Feature Engineering + SMOTE oversample + random undersample:

After we do feature encoding and apply SMOTE oversampling the minority and undersampling the majority, we get a balanced dataset: containing 267932 “0” and 128916 “1”.

```
0    257832
1    128916
Name: is_fraud
```

We create a model dictionary to compare their performance with the following models from scikit-learn:

- Logistic regression is a machine learning algorithm for classification based on the concept of probability. It is widely used when the problem is binary so we add it to our model dictionary.
- Decision Trees is a supervised learning method used for classification(our goal) and regression that predicts target values by learning simple decision rules inferred from the data features.
- Random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
- A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregates their individual predictions to form a final prediction.
- xgboost is an optimized distributed gradient boosting library that implements algorithms under the gradient boosting framework. It builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. For binary classification, only a single regression tree is induced. We add this model to our model dictionary because of its high efficiency and flexibility.

Hence, we have a set of classifiers to compare their training score which is the stratified 5-fold cross-validation from sklearn. Now we have a simple mind of how each classifier performs.

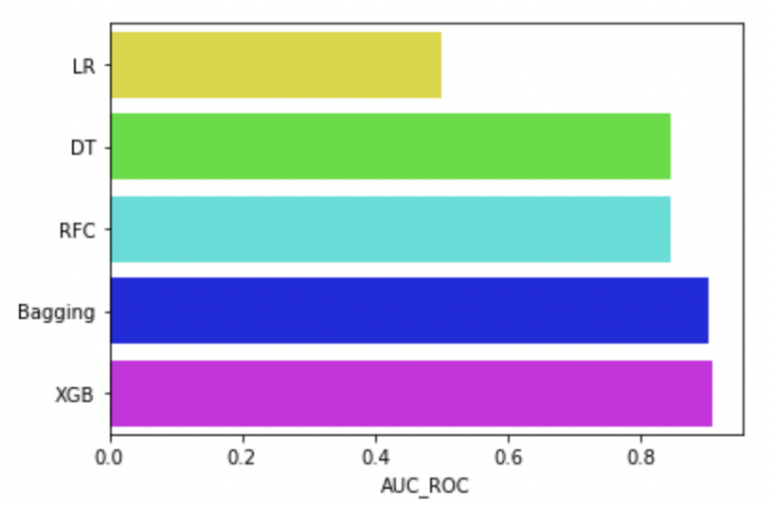
```
Classifier Name :  LogisticRegression  Training Score : 67.0 %
Classifier Name :  DecisionTreeClassifier  Training Score : 99.0 %
Classifier Name :  RandomForestClassifier  Training Score : 99.0 %
Classifier Name :  BaggingClassifier  Training Score : 99.0 %
Classifier Name :  GradientBoostingClassifier  Training Score : 99.0 %
Classifier Name :  XGBClassifier  Training Score : 99.0 %
```

Then we use GridSearchCV to find the best hyperparameter for each model by comparing accuracy since the training data is balanced.

```
DT_best_estimator: DecisionTreeClassifier(max_depth=4, min_samples_leaf=3)
RFC_best_estimator: RandomForestClassifier(max_depth=4, n_estimators=200)
BAG_best_estimator: BaggingClassifier(n_estimators=20)
XGB_best_estimator: XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
LG_best_estimator: LogisticRegression(C=0.001)
```

With these best hyperparameters, we get the best model for each classifier. Fit the test set into these best models to derive accuracy(by cross-validation), precision, recall, f1-score, and auc_roc.

	Accuracy	F1-score	Recall	Precision	AUC_ROC
LR	0.996140	[0.9980663359455076, 0.0]	[1.0, 0.0]	[0.9961401355721147, 0.0]	0.500000
DT	0.983168	[0.9914889822627988, 0.24515816655907036]	[0.9842333635611499, 0.7081585081585081]	[0.9988523700574915, 0.14823850883185322]	0.846196
RFC	0.990582	[0.9952556974233561, 0.36326034063260343]	[0.991722877158248, 0.696037296037296]	[0.9988137776451667, 0.24576131687242797]	0.843880
Bagging	0.995615	[0.9977956573454722, 0.5864585100967249]	[0.9963509846922002, 0.8055944055944056]	[0.9992445255276092, 0.4610458911419424]	0.900973
XGB	0.996944	[0.9984650458949469, 0.6744631901840491]	[0.9976299464931517, 0.8200466200466201]	[0.9993015445636282, 0.5727775968739824]	0.908838



We can see that all models work well on finding target value = 0 but are a little weak on finding target value = 1. XGB performs the best with the highest f1-score which is the harmonic mean of precision and recall. For example, the recall of XGB predicting target = 0 is 0.9976 means that 99.76% of non-fraud transactions are classified as non-fraud. The recall of XGB predicting target = 1 is 0.8200 means that 82.00% of fraud transactions are classified as fraud. The precision of XGB predicting target = 0 is 0.9993 means that 99.93% of transactions that are classified as non-fraud are actually non-fraud. The precision of XGB predicting target = 1 is 0.5728 means that 57.28% of transactions that are classified as fraud are actually a fraud.

To improve the prediction results, we do ensemble learning. We pick the best 3 models: RFC, Bagging, and XGB to create a supermodel by using a voting classifier from sklearn.

	RFC	Bagging	XGB
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
...	...	...	...
555714	0	0	0
555715	0	0	0
555716	0	0	0
555717	0	0	0
555718	0	0	0

The prediction result by each of the 3 best models for each transaction is assembled. Then by majority rule voting, we get the final prediction result. For example, if the result of 3 models is (0, 1, 1), then the final prediction result is 1.

Below is the classification report for the final prediction of the supermodel. We can see that the performance is highly improved.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	553574
1	1.00	0.92	0.96	2145
accuracy			1.00	555719
macro avg	1.00	0.96	0.98	555719
weighted avg	1.00	1.00	1.00	555719

-Result and insights-

Let's try out supermodel!

You can input your transaction information as below:

```
array([[  250, 63130,  5000, 38326,   400,    24,     0,  6468,     5,
             1,    16,     1,     0,     0,     0,     1,     0,     0,
             0,     0,     1,     0,     0,     0,     0,     1,     0,
             0,     0,     0,     1,     0,     0,     0,     0,     0]])
```

followed by these indices:

```
['amt', 'zip', 'city_pop', 'unix_time', 'distance', 'age', 'gender',
 'merchant num', 'month', 'weekend', 'num_cc_num', 'merch_size_large',
 'merch_size_medium', 'merch_size_mediumlarge', 'merch_size_small',
 'city_size_large', 'city_size_medium', 'city_size_mediumlarge',
 'city_size_small', 'afternoon', 'before_dawn', 'early_morning',
 'evening', 'morning', 'night', 'online', 'other', 'pos_machine',
 'eastcoast_north', 'eastcoast_south', 'mideast_north', 'mideast_south',
 'midwest_north', 'midwest_south', 'westcoast_north', 'westcoast_south'],
```

Then our supermodel will return you a result: `array([0])`

This means that your transaction is not a fraud!

-Conclusions-

Finally, we build a model to suggest whether the transaction on the credit card is fraud or not which is reliable based on the data we have. While doing credit card fraud detection model building, the biggest problem is that the training set we get is very imbalanced. This results that if our model simply passes every transaction, we get a very high accuracy which nearly equals the percentage of non-fraud transactions(99%). But this is not what we want. Our goal is to detect truly fraudulent transactions and pass truly legitimate transactions. To solve this, we rebalance the training set with a reasonable ratio of fraud and non-fraud transactions so that the machine learning algorithms we use can better learn the rules inside. Before this, since our raw dataset contains many categorical values which cannot be understood by computers well, we do feature encoding and then one-hot encoding to ensure we get the maximum dimension of information. While we splitting the training and testing set, the problem is: should we do split first or rebalance first? We actually do both of them on the given training set. And we have given testing set from Kaggle for the final prediction test. We do not rebalance the final testing set because once we have the new information that needs to be predicted by our model, the information we get

is not rebalanced. We use different rebalancing methods and also different classifiers. We actually run more models than we mentioned above like KNN and SVC. But they are either having bad performance or hard to converge. We choose candidate models by their feasibility and efficiency because we need to consider time and expense. When we get the test results of each model but not good enough precision for finding fraud transactions, we decide to apply ensemble learning. A group usually has more power than an individual, like team hotpot. We pick the three best models with their results and decide the final prediction result by the majority voting rule. Finally, we derive a good result with high recall and precision on both predicting fraud and non-fraud! In our code, we mostly define functions to preprocess data and build models. This means if we get new data, we can always efficiently update our models! Credit card fraud detection is a really interesting topic that not only allows us to learn, test, and practice machine learning methods but also relates data preprocessing with thinking about real situations. Hence, by our prediction result, we can suggest if the transaction is a fraud so that the bank can automatically give the customer a reasonable alert to protect his/her property and information safe on time.

-References-

dataset: https://www.kaggle.com/kartik2112/fraud-detection
sklearn: https://scikit-learn.org/stable/
smote oversample: https://www.jair.org/index.php/jair/article/view/10302
xgboost: https://xgboost.readthedocs.io/en/stable/
feature engineering:
https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/