Detect Fraud from Customer Transactions

Introduction

Lots of people use credit card nowadays, there may be millions of online transactions per day. At the same time, fraud happens from time to time, and it cause millions of dollars loss. So online transaction fraud detection is important for banks and customers.

We need a system to detect the credit card fraud automatically in time to prevent further damage. This will help the banks to save money, help the government to crack down economic crimes, and this protect customers benefits.

So, the problem is "can we automatically detect the online transaction fraud in time based on an ongoing transaction”? This is a binary classification problem, with the historical data of online transaction, we can use machine learning algorithms to solve this problem.

Data description

IEEE-CIS Fraud Detection is a completion in kaggle.com, IEEE-CIS works across a variety of AI and machine learning areas, including deep neural networks, fuzzy systems, evolutionary computation, and swarm intelligence. Today they’re partnering with the world’s leading payment service company, Vesta Corporation, seeking the best solutions for fraud prevention industry. In this report, I will use the dataset from this completion.

The dataset includes following csv files. The data comes from Vesta's real-world e-commerce transactions and contains a wide range of features from device type to product features.

- train_{transaction, identity}.csv - the training set
- test_{transaction, identity}.csv - the test set

The data is broken into two files `identity` and `transaction`, which are joined by `TransactionID`. Not all transactions have corresponding identity information. The target is predicting the probability that an online transaction is fraudulent, as denoted by the binary target `isFraud`.

```
train_transaction.head()
```

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	...	V330	V331	V332	V333
0	2987000	0	86400	68.5	W	13926	NaN	150.0	discover	142.0	...	NaN	NaN	NaN	NaN
1	2987001	0	86401	29.0	W	2755	404.0	150.0	mastercard	102.0	...	NaN	NaN	NaN	NaN
2	2987002	0	86469	59.0	W	4663	490.0	150.0	visa	166.0	...	NaN	NaN	NaN	NaN
3	2987003	0	86499	50.0	W	18132	567.0	150.0	mastercard	117.0	...	NaN	NaN	NaN	NaN
4	2987004	0	86506	50.0	H	4497	514.0	150.0	mastercard	102.0	...	0.0	0.0	0.0	0.0

5 rows × 394 columns

Data wrangling

As described above, the datasets include two csv files transaction.csv and identity.csv and they are joined by `TransactionID`. First thing first, lets merge these two datasets together into a new dataframe.

```
train = pd.merge(train_transaction, train_identity, on='TransactionID', how='left')
train.shape
```

```
(590540, 434)
```

```
train.filter(regex='V').head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V330	V331	V332	V333	V334	V335	V336	V337	V338	V339
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 339 columns

There are 590540 rows and 434 columns in this dataframe, V1~V339 columns have a lot of missing values so does id_01~id_38 columns. Take V330 for example, it has 86% of missing values. So I will just drop V1~V339, id_01~id_38 columns.

```
train['V330'].value_counts(dropna=False, normalize=True).head()
```

```
NaN      0.860550
 0.0     0.123275
 1.0     0.008804
 2.0     0.002945
 3.0     0.001097
Name: V330, dtype: float64
```

Of course there are missing values in other columns. Column card2, card3, card5 are numeric values and the missing value in these columns are just small proportion. For these missing values, it’s proper to replace the missing value with mean or the the common value. Take card3 for example, as show in the following figure it is obvious that the NaN value should be replaced with 150.0

```
train['card3'].value_counts(dropna=False, normalize=True).head()
```

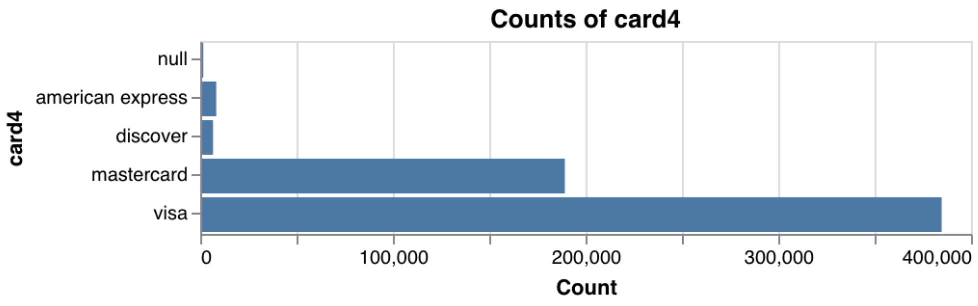
```
 150.0    0.882729
 185.0    0.095414
 106.0    0.002660
NaN       0.002650
 146.0    0.002120
Name: card3, dtype: float64
```

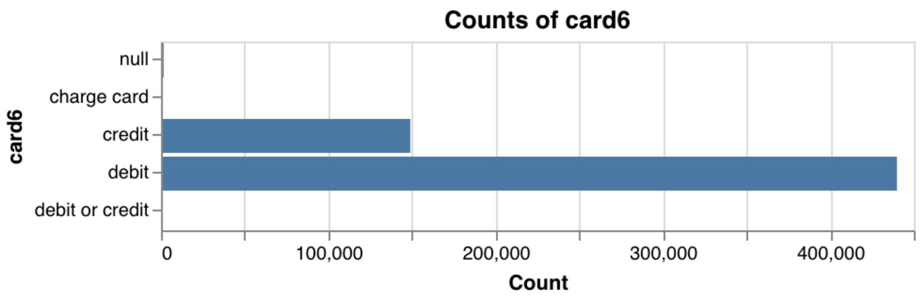
+ Code   + Markdown

```
train['card3'].describe()
```

```
count    588975.000000
mean        153.194925
std          11.336444
min         100.000000
25%         150.000000
50%         150.000000
75%         150.000000
max         231.000000
Name: card3, dtype: float64
```

Categorical data is data that takes only a limited number of values. For categorical columns like card4 and card6 as show in the following figures, I will deal them with the One-Hot Encoding method which is the most widespread approach for categorical data.





```
card4_dummies = pd.get_dummies(train['card4'], dummy_na=True, prefix='card4')
card6_dummies = pd.get_dummies(train['card6'], dummy_na=True, prefix='card6')
```

Train Model

Online transaction fraud detection is aimed to find out if a transaction is fraud or not, it is a binary classification problem. Besides, it would be grate for the model to give the probability of its prediction.

Logistic regression

There are many classification algorithms, I will try Logistic Regression first. Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Since Logistic Regression just use numerical values, after the data wrangling in the previous section, I selected the following features which are numerical values.

```
X = train[['TransactionDT', 'TransactionAmt', 'card1', 'card2', 'card3', 'card5', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8',
       'C12', 'C13', 'C14', 'card4_american express', 'card4_discover', 'card4_mastercard',
       'card4_visa', 'card4_nan', 'card6_charge card', 'card6_credit',
       'card6_debit', 'card6_debit or credit', 'card6_nan', 'ProductCD_C',
       'ProductCD_H', 'ProductCD_R', 'ProductCD_S', 'ProductCD_W']]
y = train['isFraud']
```

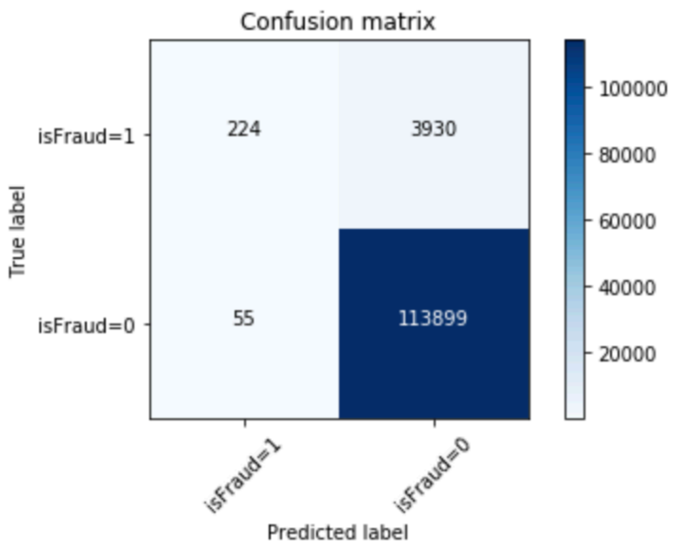
Then split the data into training data and testing data:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (472432, 35) (472432,)
Test set: (118108, 35) (118108,)
```

Import LogisticRegression from sklearn.linear_model and train the model with X_train and y_train. As a result, the confusion matrix is show as following which is a poor result.

I think LogisticRegression is easy over fitting for this dataset, I should try to use another algorithm like LGBM.

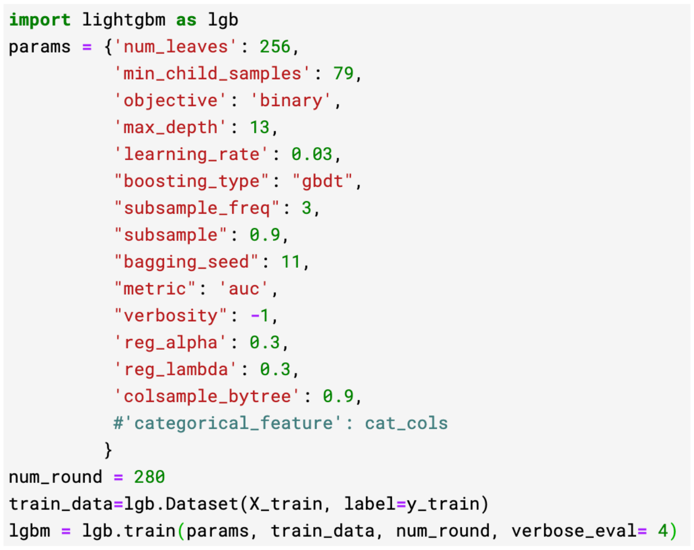


LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

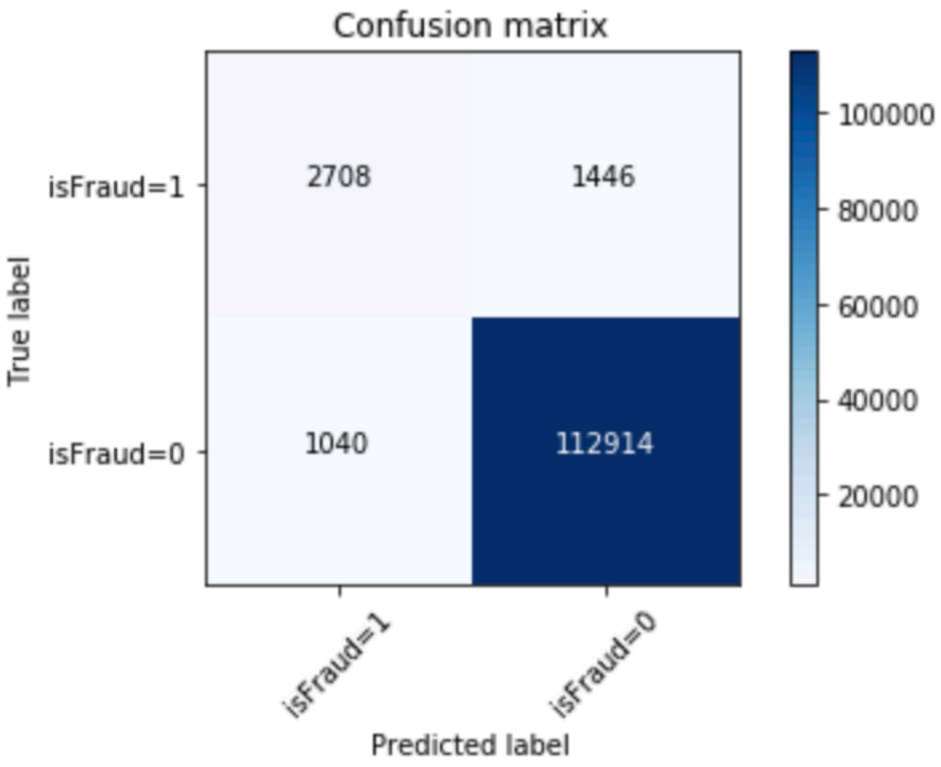
In this report we have a large dataset, and LightGBM also generates the probability as prediction.



```
import lightgbm as lgb
params = {'num_leaves': 256,
          'min_child_samples': 79,
          'objective': 'binary',
          'max_depth': 13,
          'learning_rate': 0.03,
          "boosting_type": "gbdt",
          "subsample_freq": 3,
          "subsample": 0.9,
          "bagging_seed": 11,
          "metric": 'auc',
          "verbosity": -1,
          'reg_alpha': 0.3,
          'reg_lambda': 0.3,
          'colsample_bytree': 0.9,
          #'categorical_feature': cat_cols
         }
num_round = 280
train_data=lgb.Dataset(X_train, label=y_train)
lgbm = lgb.train(params, train_data, num_round, verbose_eval= 4)
```

Results

Using the LightGBM model to predict the testing data I get the following confusion matrix. The results is much better than LogisticRegression, but its not good enough. There are many parameters in LightGBM, for example num_round. In this report I choose the num_round to 280, verbose_eval to 4. As the num_round goes up the results should be better, but it will also consume more compute resource.



Conclusion

In this report I build a model with machine learning algorithm to detect online transaction fraud. This is a binary classification problem, and I tried two algorithms Logistic Regression and LGBM. The data is from a completion in kaggle.com which is provided by Vesta's real-world e-commerce transactions and contains a wide range of features. As show in the results, LGBM achieved a better performance than Logistic Regression. This model cloud be helpful to detect online transaction fraud.