IEEE-CIS Fraud Detection

(https://www.kaggle.com/c/ieee-fraud-detection)

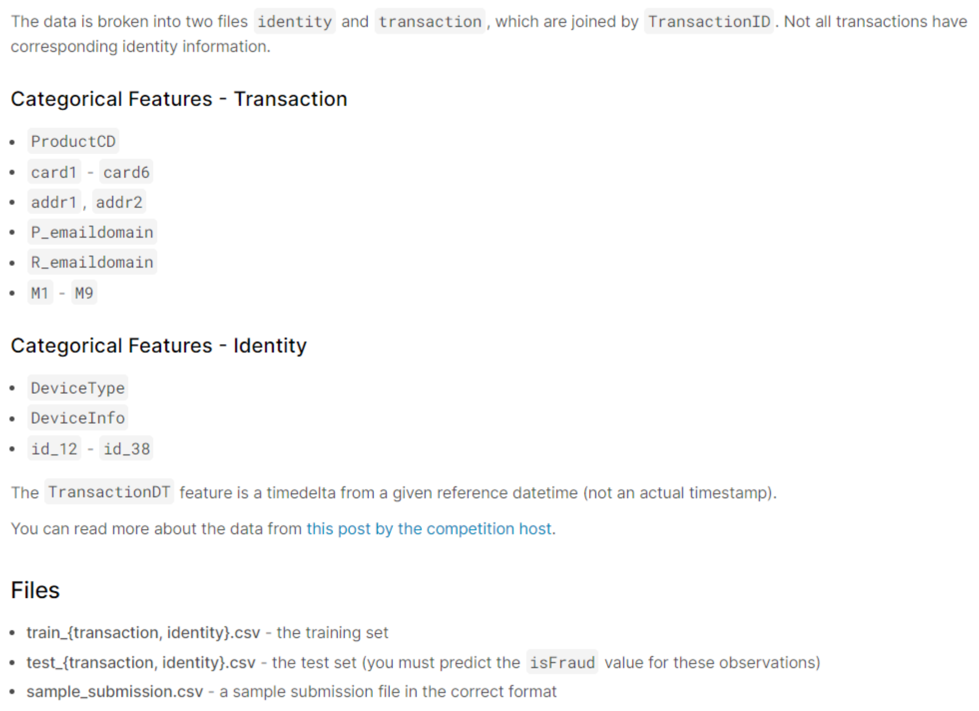
1. Problem Statement

1.1Business Framing

①Current Situation:
Fraud transaction happens every minute around the world. Although the fraud prevention system is actually saving consumers millions of dollars per year, it can be improved with more accuracy.
②Purpose of building the model:
This machine learning model can provide higher accuracy fraud detection for online transaction.
If successful, it can improve the efficacy of fraudulent transaction alerts for millions of people around the world, helping hundreds of thousands of businesses reduce their fraud loss and increase their revenue.

1.2Data Description

The data comes from a real-world e-commerce transactions company and contains a wide range of features from device type to product features.
We are predicting the probability that an online transaction is fraudulent, as denoted by the binary target *isFraud*.


The data is broken into two files `identity` and `transaction`, which are joined by `TransactionID`. Not all transactions have corresponding identity information.

Categorical Features - Transaction

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

Categorical Features - Identity

- `DeviceType`
- `DeviceInfo`
- `id_12` - `id_38`

The `TransactionDT` feature is a timedelta from a given reference datetime (not an actual timestamp).

You can read more about the data from this post by the competition host.

Files

- **train_{transaction, identity}.csv** - the training set
- **test_{transaction, identity}.csv** - the test set (you must predict the `isFraud` value for these observations)
- **sample_submission.csv** - a sample submission file in the correct format

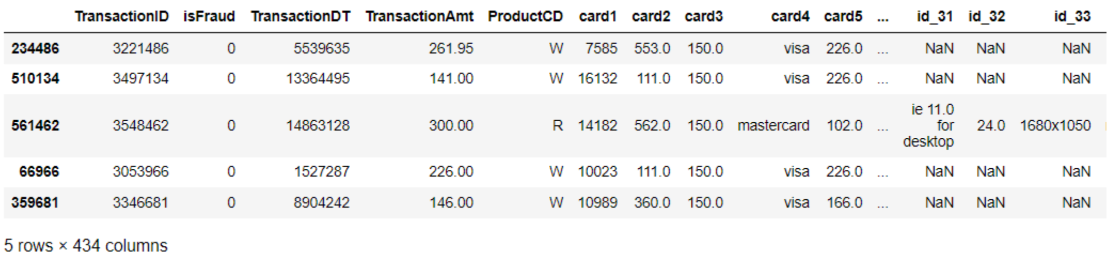
2. Technical Approach

There are two steps to build the model. First, I preprocess data, including handling missing data, encode categorical variables, PCA to reduce dimension. Then, I use five different approaches to predict fraud, including logistic regression, bagging trees, random forests, gradient boosting, XGBoost.

2.1 Data preprocessing

① **Import and merge the data:**

After I import the appropriate libraries and data, I merge *identity* and *transaction* to one data frame left joined by *TransactionID.* The computation on the whole data set is too expensive, so I randomly select a subset of size 5000 for demo.



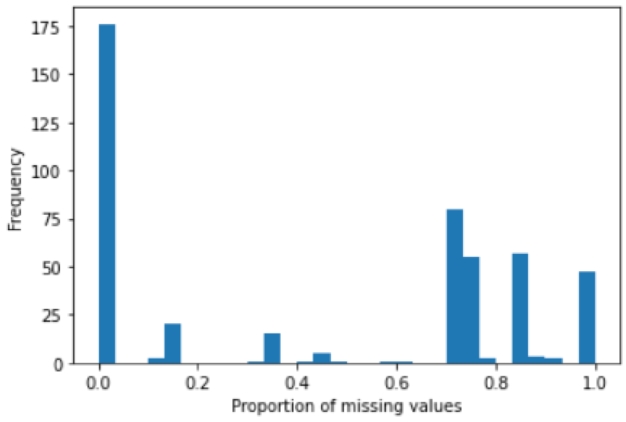
	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	...	id_31	id_32	id_33
234486	3221486	0	5539635	261.95	W	7585	553.0	150.0	visa	226.0	...	NaN	NaN	NaN
510134	3497134	0	13364495	141.00	W	16132	111.0	150.0	visa	226.0	...	NaN	NaN	NaN
561462	3548462	0	14863128	300.00	R	14182	562.0	150.0	mastercard	102.0	...	ie 11.0 for desktop	24.0	1680x1050
66966	3053966	0	1527287	226.00	W	10023	111.0	150.0	visa	226.0	...	NaN	NaN	NaN
359681	3346681	0	8904242	146.00	W	10989	360.0	150.0	visa	166.0	...	NaN	NaN	NaN

5 rows × 434 columns

Now, training dataset has 5000 observations and 434 features while test dataset has 506691 observations and 433 features.

② **Handle missing values:**

I drop features with high proportion (70%) of missing values. Then, I fill missing values in categorical variables with their mode and fill missing values in numerical variables with their mean.



After deleting features with high proportion of missing values, there are 223 features.

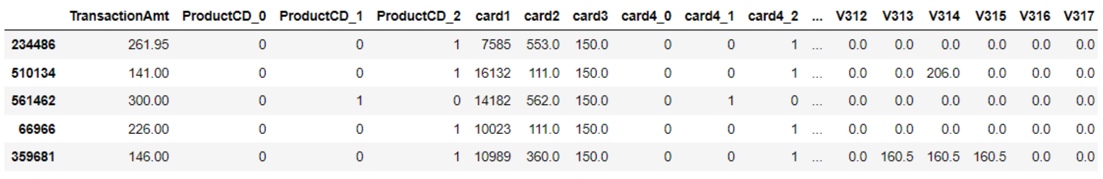
③ **Encode categorical variables:**

I try two different ways: numeric encoding and binary encoding. Numeric encoding (label encoding) simply assigns a value to each category. Binary encoding hashes the cardinalities into binary values. (Here is the reason why I do that: https://medium.com/data-design/visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931)

④ **PCA to reduce dimension:**

Although PCA can reduce dimension and computation time, the result showed that the prediction performance with PCA is worse. So I will not use the data after PCA to fit the model and make predictions in the modeling part.

⑤ **Save processed data**



	TransactionAmt	ProductCD_0	ProductCD_1	ProductCD_2	card1	card2	card3	card4_0	card4_1	card4_2	...	V312	V313	V314	V315	V316	V317
234486	261.95	0	0	1	7585	553.0	150.0	0	0	1	...	0.0	0.0	0.0	0.0	0.0	0.0
510134	141.00	0	0	1	16132	111.0	150.0	0	0	1	...	0.0	0.0	206.0	0.0	0.0	0.0
561462	300.00	0	1	0	14182	562.0	150.0	0	1	0	...	0.0	0.0	0.0	0.0	0.0	0.0
66966	226.00	0	0	1	10023	111.0	150.0	0	0	1	...	0.0	0.0	0.0	0.0	0.0	0.0
359681	146.00	0	0	1	10989	360.0	150.0	0	0	1	...	0.0	160.5	160.5	160.5	0.0	0.0

5 rows × 234 columns

The data is clean now.

2.2 Modeling

① **Logistic regression:**

The prediction score is 0.713617.

```python
lr = LogisticRegression(penalty='l2', max_iter=500, n_jobs=6, tol=1e-6, solver="sag")
lr.fit(Xtr, np.ravel(Ytr))
Yhat_lr = lr.predict_proba(Xte)
submission["isFraud"] = Yhat_lr[:, 1]
# submission.to_csv(f"{path}Y_hat_logistic.csv", index=False)
```

② **Bagging trees:**

I use RandomForestClassifier and set max_features="auto", which means max_features = n_features, so it is bagging.

```python
treeCount = 100

bagging = RandomForestClassifier(max_features="auto", min_samples_leaf=1, n_estimators=treeCount)
bagging.fit(Xtr, np.ravel(Ytr))
Yhat_bagging = bagging.predict_proba(Xte)
submission["isFraud"] = Yhat_bagging[:, 1]
# submission.to_csv(f"{path}Y_hat_bagging.csv", index=False)
```

③ **Random forests:**

Let's first do parameter tuning. Other than max_features (the number of variables considered at each split), the parameter n_estimators (the number of trees) is also important. However, the prediction performance will increase with the increase of n_estimators, so we should select the highest n_estimators as long as our machine can compute it. I try different parameters (max_features and n_estimators) and get the prediction scores (evaluated by AUC) as following.

max_features	n_estimators	PCA	prediction score
223	100	99%	0.871838
190	100	100%	0.892415
100	100	100%	0.894553
50	100	100%	0.895868
223	100	100%	0.896448
90	200	100%	0.898140
100	200	100%	0.899070
50	200	100%	0.900798
15	1000	100%	0.904874

④ **Gradient boosting:**

At first, let's try gradient boosting with default parameters. The prediction score on test data set is 0.891210.

```
# use default parameters
gbm0 = GradientBoostingClassifier()
gbm0.fit(Xtr, np.ravel(Ytr))
submission["isFraud"] = gbm0.predict_proba(Xte)[:, 1]
submission.to_csv(f"Y_hat_gbm_default.csv", index=False)
```

Now let's do parameter tuning. The parameters n_estimators and learning_rate are correlated, so we need to tune them together. Let's use grid search to find the best number of weak learners (n_estimators) and the best step size (learning_rate). After parameters tuning, the prediction score (by AUC) on test data set is 0.919523. It is better than that without parameters tuning.

⑤ **XGBoost:**

The prediction score on test data set by XGBoost with default parameters is 0.900976, and 0.931355 with the parameters tuned by GBT.

```
xgbc = xgb.XGBClassifier(n_jobs=4, max_depth=10, min_samples_leaf=0.001,
                         learning_rate=0.1, n_estimators=100, eval_metric="auc")
xgbc.fit(Xtr, np.ravel(Ytr))
submission["isFraud"] = xgbc.predict_proba(Xte)[:, 1]
```

3. Results

From the results, we can see the prediction score (evaluated by AUC) of XGBoost is the highest and most accurate, around 0.93.

4. Conclusion:

I think the model can be useful and help improving accuracy of fraud detection.

The superiority of my approach to relevant benchmark methods:

① I compare five different approaches and choose the most accurate one.

② I do parameter tuning before running the models.

③ I try to use PCA to reduce dimension and computation time. But as the result shows that the prediction performance with PCA is worse. So I don't use the data after PCA to fit the model and make predictions in the modeling part.

④ In data processing, I drop features with high proportion (70%) of missing values and encode categorical variables.