Winning Model Documentation

Name: Shichao Luo, Zhongwei Yao, Yongyao Chao

Location: China, China, China

Email: 1070293362@qq.com, yaozhongwei0131@163.com, triplelift@foxmail.com

Competition: IEEE-CIS Fraud Detection

1. What is your academic/professional background?

a. Shichao Luo is currently an algorithm engineer at a bank.

b. Zhongwei Yao is currently an algorithm engineer at a logistics company

c. Yongyao Chao is currently an algorithm engineer at Tencent.

2. Did you have any prior experience that helped you succeed in this competition?

We have some experience about Fraud/ Credit Risk model. Also we have participated in various competitions before this one so we are all fairly experienced Kagglers.

Introduction:

In the IEEE-CIS Fraud Detection competition, many people define the problem as predicting the probability of fraud in each transaction. But we found that the essence of this competition is to pick out fraudulent client ("user"). If the user is fraudulent, then all the transactions corresponding to the user have a high probability of fraud. After we managed to decode the tricky property “DT-D1”, we spent some time carefully analyzing the logic to identify a client and use it to get a person’s transaction record. We noticed the fact that the client comes and goes over time (in particular, fraudulent client get banned). Most blacklisted client in train set are not seen in test set, which means merely tagging overlapping clients won’t work well. We came up with two solutions: the first is that we apply the user id for post-processing(i.e. data.groupby(‘userId’)[‘isFraud’].mean()); the second is that we combine “DT-D1” with other categories to create some user groups and then make more aggregation features. Facts have proved that the second solution works well and those user group behavior statistics show good generalization of unseen client in

test. Note that we abandoned those user id features as they lead to overfitting on train data because the model is trained to memorize seen clients in train set. We want our model to recognize behaviors instead of memorizing clients. At last we trained three models(Lgb、Ctb、NN) with 2 different feature sets. Our single Lgb got 0.961+ on LB and our single Ctb got 0.959+ on LB. One big advantage of our team is that we built a NN model with LB score over 0.956. Although it's not that high in comparison with the other two models, it has brought great diversity that gives a considerable boost to our blending result and eventually makes us top 3 in this competition.

Methodology:

890 features part

Data processing:

After taking a look at the data, we observe many anonymized features, such as V columns and C columns. We perform Recursive Feature Elimination (RFE) to drop more than 300 features with low ranking in the test, which not only give us boost in public leaderboard score but also save us a lot of time in model training.

When taking a look at categorical features, we find certain columns with large cardinality, which can make tree models go deep and overfit. Several public notebooks enlighten us to perform tagging and binning on categorical features. For example, we bin similar P_emaildomain values like 'yahoo.com.mx', 'yahoo.fr', 'yahoo.es' to 'yahoo', thus reducing cardinality of this feature. Another approach we adopt to reduce cardinality is low frequency filtering on card1, addr1, addr2. The logic we process is that we set values appear only once in dataset and values only appear in train or test to null. It helps to reduce cardinality in certain features and prevent overfitting as we see higher leaderboard score with a decrease in local cv score.

My final LGBM model uses 890 features. 124 of them come from original dataset and the other 766 are created through the feature engineering process. In order to detect potential features, we run a baseline LGBM model with only original features as input and print the top 50 feature importance columns, from which we select features for further construction. In summary, the categorical features we focus on includes hour of day, cardxx, addrxx, email domain, idxx (categorical part). The numeric features we focus on include transaction amount, time-delta, distxx, C columns and idxx (numeric part). For categorical features, we just perform frequency encoding to show the model if the variable takes a rare value. For numeric features, we try enormous aggregations (e.g. mean, median, std, min, max, rank, etc.) grouped by some particular categories (mainly different levels of user groups, strong solo categorical features and strong categorical interaction features). **One important point that should be mention is the extraction of user id. Given the mechanism of fraud labelling Lynn elaborated in the discussion thread, we realize the challenge of this competition lies in detecting fraudulent clients**. After we manage to decode the tricky **"DT-D1"**, we spend some time carefully analyzing the logic to identify a client and use it to get a person's transaction record.

We noticed the fact that the client comes and goes over time (in particular, fraudulent client get banned). Most blacklisted client in train set are not seen in test set, which means merely tagging overlapping clients won't work well.

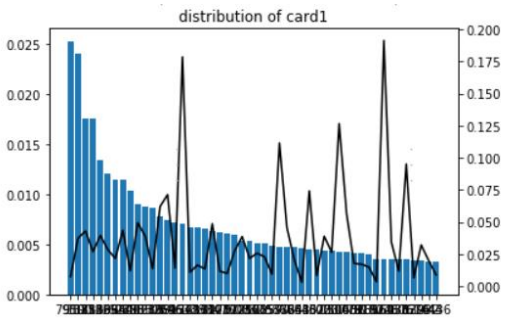
We come up with two solutions: the first is that we apply the user id for post-processing(i.e. data.groupby('userId')['isFraud'].mean()); the second is that we combine "DT-D1" with other categories to create some user groups and then make more aggregation features. For example, the code below elaborates our second solution in detail,

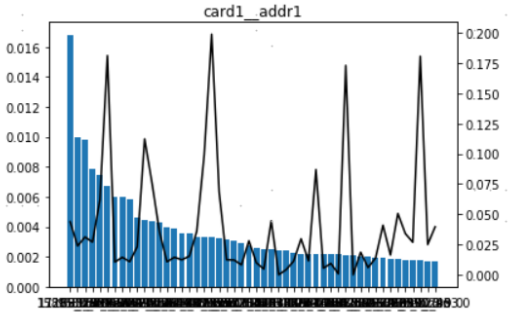
```
data['user_lvl_1'] = data['card1'].astype(str) +'_'+ data['card2'].astype(str)+\
                     '_'+data['card3'].astype(str)+'_'+data['card4'].astype(str)+'_'+data['card5'].astype(str)+'_'+\
                    data['card6'].astype(str)+'_'+data['addr1'].astype(str)+'_'+data['addr2'].astype(str)+\
                     '_'+data['P_emaildomain'].astype(str)

data['user_lvl_2'] = data['card1'].astype(str)+'_'+data['card2'].astype(str)+\
                     '_'+data['card3'].astype(str)+'_'+data['card4'].astype(str)+'_'+data['card5'].astype(str)+'_'+\
                    data['card6'].astype(str)+'_'+data['addr1'].astype(str)+'_'+data['addr2'].astype(str)

data['user_lvl_3'] = data['card1'].astype(str)+'_'+data['card2'].astype(str)+\
                     '_'+data['card3'].astype(str)+'_'+data['card4'].astype(str)+'_'+data['card5'].astype(str)+'_'+\
                    data['card6'].astype(str)+'_'+data['addr1'].astype(str)

data['user_lvl_4'] = data['card1'].astype(str)+'_'+data['card2'].astype(str)+\
                     '_'+data['card3'].astype(str)+'_'+data['card4'].astype(str)+'_'+data['card5'].astype(str)+'_'+\
                    data['card6'].astype(str)

data['days_created'] = data['Date'] - data['D1']
data['uid1'] = data['card1'].astype(str) + '_' + data['days_created'].astype(str)
data['uid2'] = data['card2'].astype(str)  + '_' + data['days_created'].astype(str)
data['uid3'] = data['addr1'].astype(str) + '_' + data['days_created'].astype(str)
#data['uid4'] = data['addr2'].astype(str) + '_' + data['days_created'].astype(str)
data['uid5'] = data['addr1'].astype(str) + '_' + data['days_created'].astype(str) + '_' + data['ProductCD'].astype(str)
data['uid6'] = data['P_emaildomain'].astype(str) + '_' + data['days_created'].astype(str)  + '_' + data['ProductCD'].astype(str)
data['uid7'] = data['card1'].astype(str) + '_' + data['days_created'].astype(str) + '_' + data['ProductCD'].astype(str)
data['uid8'] = data['user_lvl_1'].astype(str) + '_' + data['days_created'].astype(str)
data['uid9'] = data['user_lvl_2'].astype(str) + '_' + data['days_created'].astype(str)
data['uid10'] = data['user_lvl_3'].astype(str) + '_' + data['days_created'].astype(str)
data['uid11'] = data['user_lvl_4'].astype(str) + '_' + data['days_created'].astype(str)
```

We try both solutions and find that the benefit of first decreases and then vanishes as our public leaderboard score reaches 0.96. The reason may be that our user id is not accurate enough to truly separate different clients and averaging target value over more than one client will do damage to our submission. Fortunately, the second solution works well and those user group behavior statistics show good generalization of unseen client in test. Note that we abandon those user id features as they lead to overfitting on train data because the model is trained to memorize seen clients in train set. We want our model to recognize behaviors instead of memorizing clients.

The workflow of feature construction consists of exploratory data analysis, feature implementation, and adversarial validation. Adversarial validation, as last step, is applied as feature selection tool. Apart from adversarial validation, we also try permutation importance to select feature with strong predictive power and distribution shift metrics like Kolmogorov-Smirnov statistic and population stability index. Their performance is not significant.

In exploratory data analysis, we use dual axis to show the distribution of features and fraud rate in one figure, which helps to gain some visual instincts about whether this feature worth building and testing. By this mean we are able to find many nice solo features (e.g. card1, C13, id_02, etc.) and categorical interaction features.(e.g. card1_addr1, card1__card2, etc.). The figure below serves as examples.





In term of adversarial validation, we apply it for the purpose of dropping features with severe covariate shift because machine learning models are built under the assumption that all features are independently and identical distributed. Most of the features remained after the validation have validation AUC less than 0.6. Some exceptions of feature with validation AUC around 0.7 are included in training because we find the shift is mostly caused by the change of rate of nulls. In this case, we think knowledge learnt in train set can generalize to test set. By using multiple validation schemes, we are more confident about the performance of model on private leaderboard.

Model:

we mainly focus on building LGBM model about different parameters. The parameters are quite basic because of the lack of time and resource to tune parameters. Check the code for more details. We find it beneficial to assign different weights to training samples. What we do here is assigning more weights to samples with later transactionDT. We try various weighting methods and the best gives us a boost of 0.0005 in both public and private leaderboard. In the end our single LGBM finishes 0.9605 on public leaderboard and 0.935 on private leaderboard.

Validation:

IMHO, the biggest challenge of this competition is finding a stable validation scheme for feature selection. We have tried holdout validation, TimeSeriesSplit validation, KFold validation and GroupKFold validation. One big disadvantage of holdout validation is that it heavily relies on the holdout set and is likely to overfit on holdout set. As for TimeSeriesSplit, we observe inconsistency between local cv and public leaderboard score, which may be cause by different amount of training samples in each fold and the way we average AUC over folds. Since data are sorted by transaction time, KFold split works quite similar to GroupKFold split when group key is month. Our final choice is 5-fold cross validation, though not perfect but works just fine in comparison with other methods.

What can be perfected here is that we should try separate schemes for feature selections and model training. A simple holdout validation requires less computation and it may be good for feature selection. When it comes to training model and making predictions, cross validation is definitely a better choice.

692 features part

Data processing:

692 features=388 baseline_features + **301 uid_magic_features**+3 combine features

1、388 baseline_features—the result of feature selected

Type1: Resconstruction error

- Use an auto-encoder to encode the raw features into latent space and then decode to compact features. We use the reconstruction error as feature.
- It is a really strong feature that adding this reconstruction error improves local CV by 0.0017 and LB by 0.001 respectively.
- Name of selected features (in the dataframe): ['ae_reconstruction_error']

Type2: df[col_A] - df.groupby(col_B)[col_A].transform('mean')

- col_A is a numeric feature and col_B is a categorical feature
- Name of selected features (in the dataframe): [V13TransactionAmtminus_mean_all]

Type3: df[col_A] - df.groupby(col_B)[col_A].transform('mean')

- Similar to the operation above, except for that col_B represents a specific time window
- col_A is a numeric feature and col_B is a catrgorical feature representing a specific time window. For example, df['transactionAMT'] - df.groupby('Date')['transactionAMT'] represent the deviation to the mean transaction amount of that day.
- Name of selected features (in the dataframe): [DateC9minus_mean_all, Date+HourD10minus_mean_all, hourae_reconstruction_errorminus_mean_all, dayofweekae_reconstruction_errorminus_mean_all]

Type4: df.groupby(col_B)[col_A].transform('mean') / df.groupby(col_B)[col_A].transform('std')

- col_A is a numeric feature and col_B is a categorical feature
- Name of selected features (in the dataframe): [uid2ae_reconstruction_errorstd]

Type5: Concatenation: col_A+col_B

- where both col_A and col_B are categorical features represented by a string, the new feature concatenate two string
- Name of selected features (in the dataframe): None

Type6: Target Mean encoding: train_df.groupby([col])['isFraud'].agg(['mean'])

- First, I decreased the cardinality of all features(numeric & categorical) to 50 (Basically, just keep most frequent 49 values and represented all other values as 'others')
- Then I added the target mean of each feature
- Name of selected features (in the dataframe):

['addr1_fq_encnew_target_mean','card5__P_emaildomain_target_mean','P_emaildomain__C2_target_mean','card2_TransactionAmt_stdnew_target_mean','card2__id_20_target_mean','id_20_target_mean','card2_count_full_target_mean','DeviceInfo__P_emaildomain_target_mean','uid_TransactionAmt_stdnew_target_mean','dist1_target_mean','card2_TransactionAmt_meannew_target_mean','uid_TransactionAmt_meannew_target_mean','uid2_TransactionAmt_stdnew_target_mean','uid2_TransactionAmt_meannew_target_mean','card1_TransactionAmt_meannew_target_mean','P_emaildomain_target_mean','dist1_fq_encnew_target_mean','id_06_tar

get_mean','DeviceInfo_fq_encnew_target_mean','card1_TransactionAmt_stdnew_target_mean']

Type7: Four approaches to combine between numeric features (col_A + col_B, col_A - col_B, col_A * col_B, col_A / col_B)

- Name of selected features (in the dataframe):

[D1_minus_D15, P_emaildomain_fq_encnew_plus_card1_count_full]

Total nubmer of Features: 935, Also, I restricted number of features from 935 to 388 and got better local CV score

2、301 uid_magic_features

Magic features are about uid groupby features. The uid are defined as follow

```
#---uid4 Lb 23% ration PB 7% ration-----------
train['days_passed'] = train['TransactionDT'] // 86400
train['days_created'] = train['days_passed'] - train['D1']
train['uid4'] = train['card1'].astype(str) + '_' + train['ProductCD'].astype(str) +'_' + train['P_emaildomain'].astype(str) + '_' + \
                train['addr1'].astype(str) + '_' + train['days_created'].astype(str)
train.loc[train['addr1'].isnull(), 'uid4'] = np.nan

test['days_passed'] = test['TransactionDT'] // 86400
test['days_created'] = test['days_passed'] - test['D1']
test['uid4'] = test['card1'].astype(str) + '_' + test['ProductCD'].astype(str) +'_' + test['P_emaildomain'].astype(str) + '_' + \
                test['addr1'].astype(str) + '_' + test['days_created'].astype(str)
test.loc[test['addr1'].isnull(), 'uid4'] = np.nan

# groupby features about uid4
fea_by_list = ['uid4']
num_list  =['TransactionAmt','dist1','dist2','C1','C2','C3','C4','C5','C6','C7','C8','C9','C10','C11','C12','C13','C14','D1','D2','D3','D4','D5',
 'D6','D7','D8','D9','D10','D11','D12','D13','D14','D15','id_02','id_05','id_06','nulls1','V307','V310','V313','C5_fq_enc','P_emaildomain_bin',
 'card5_count_full','card5_TransactionAmt_std','V315','id_02_to_mean_card1','C11_fq_enc','C9_fq_enc','D15_to_mean_addr1','D15_to_mean_card4',
 'card5_TransactionAmt_mean','C2_fq_enc','D15_to_std_card1','C14_fq_enc','C6_fq_enc','id_31_count_dist','Transaction_day_of_week','C1_fq_enc',
 'uid_TransactionAmt_std','D15_to_mean_card1','TransactionAmt_decimal','card2_TransactionAmt_std','TransactionAmt_to_mean_card4','uid_TransactionAmt_mean',
 'C13_fq_enc','Transaction_hour_of_day','TransactionAmt_to_std_card1','TransactionAmt_to_std_card4','card1_TransactionAmt_std','card2_TransactionAmt_mean',
 'V13TransactionAmtminus_mean_all','TransactionAmt_to_mean_card1','card1_TransactionAmt_mean','uid2_TransactionAmt_std','card2_fq_enc',
 'uid2_TransactionAmt_mean','card1_count_full']
columns_name = []
added = []
for target in target_list:
    for fea_by in fea_by_list:
        print('Now Processing', target, fea_by)
        added.append(data.groupby(fea_by)[target].transform('mean'))
        added.append(data.groupby(fea_by)[target].transform('std'))
        added.append(  data[target] - data.groupby(fea_by)[target].transform('mean')  )
        added.append(  data[target] / data.groupby(fea_by)[target].transform('mean')  )
        added.append(  data[target] / data.groupby(fea_by)[target].transform('std')  )
        columns_name.append(fea_by + '_' + target +'_' + 'mean')
        columns_name.append(fea_by + '_' + target +'_' + 'std')
        columns_name.append(fea_by + '_' + target +'_' + '_minus_mean')
        columns_name.append(fea_by + '_' + target +'_' + '_divide_mean')
        columns_name.append(fea_by + '_' + target +'_' + '_divide_std')
```

3、3 combined features

['daypassed_minus_D15','daypassed_plus_D15','daypassed_minus_D2']

Model:

Single model about 692 features as follows

Models: Lgb、Ctb、NN are used

- Lgb_692_features single model
  ----------CV 9562 LB 9597
  ---------- tune the parameters the lgb LB can reach 9614
- Ctb_692_features single model
  ----------CV 9582 LB 9590
- NN model
  It's an ensemble of 2 NN (same architecture, same features, with different loss functions)
  ----------CV 9518 LB 9556

Validation: Still the KFold(n_splits=nfold, shuffle=False)

Model blending

step 01:
lgb_890features_blend_0.65=0.65*lgb_kfold_9614+0.35*lgb_kfold_9605
------LB 9646-----
step 02:
lgb_0930_0.95_v0=0.95*lgb_890features_blend_0.65+0.05*CV9518_NN_LB9556
------LB 9663-----
step 03:
lgb_0930_0.65_v1 =0.65*lgb_0930_0.95_v0.csv+0.35*pred_692_features_blend
while:
pred_692_features_blend=0.65*lgb_cv9562_692features_9597+0.35*CatBoost_cv9582_692features_9590
Finally：lgb_0930_0.65_v1
Public:0.967161 Private:0.943642

Summary:

The key to this competition is finding a way to accurately identify individual activities. Catching the “uid” by combing “DT-D1” with other categories to create some user groups and then making more aggregation features leads us to gold. LGB and NN are always good friends that ensembled model shows lower variance and better generalization. At last, nice teamwork is also Indispensable for winning the prize. We all appreciate those unforgettable days and nights when we discuss issues and run experiments and hold on to the last. Happy kaggling!