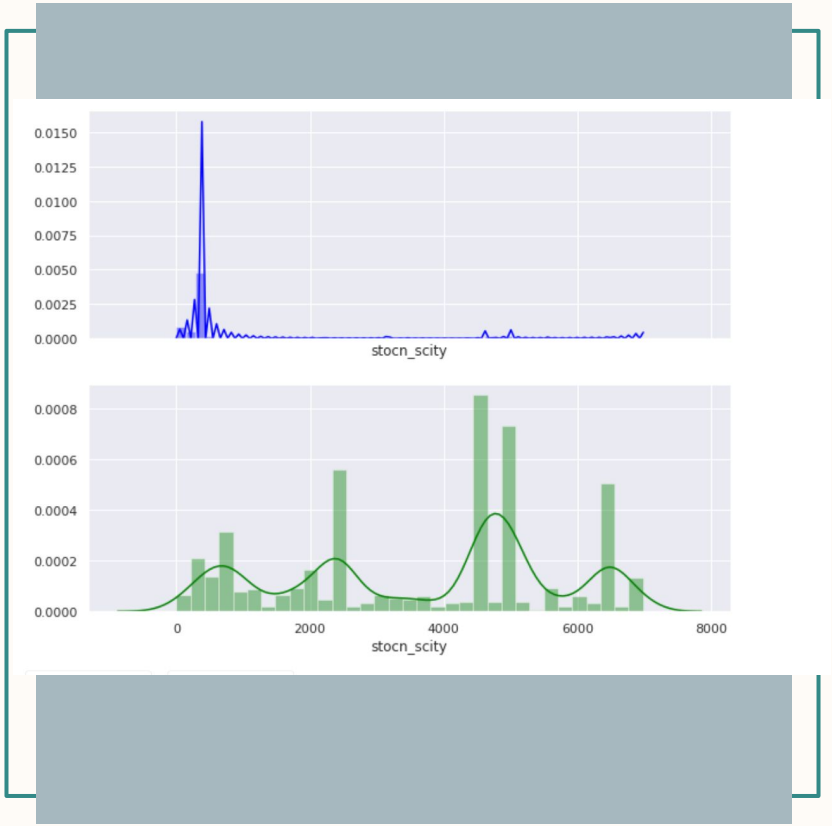
E-SUN Fraud Detection Competition

Team: 松下問童子
Author : Benson, Arvis, KW, Dean, Lish

Table of Contents
2

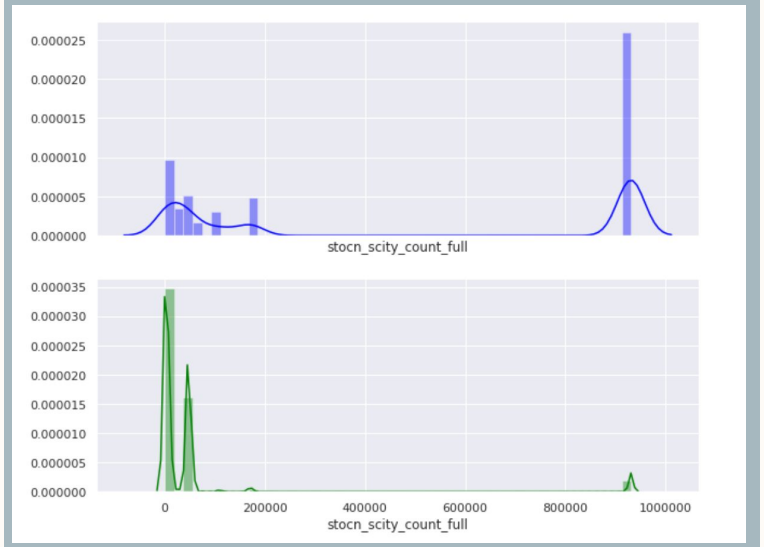
Distribution of feature

1. the distribution of conam (fraud and is not fraud)
2. the distribution of stscd and csmcu (fraud and is not fraud)
3. etc.



Label Encoding

Plot is Fraud and is not Fraud distribution



Frequency Encoding

Label encoding
Plot is Fraud and is not Fraud distribution

Ex: Feature Groupby conam

```
for df in [train, test]:
    for agg_type in ['mean', 'std']:
        # group 歸卡帳戶及卡號，計算交易金額平均及標準差，及加入每日時間的交易金額平均及標準差
        df['bacno_cano_group_to_conam_{}'.format(agg_type)] = df['conam'] / df.groupby(['bacno', 'cano'])['conam'].transform(agg_type)
        df['bacno_cano_locdt_group_to_conam_{}'.format(agg_type)] = df['conam'] / df.groupby(['bacno', 'cano', 'locdt'])['conam'].transform

        df['csmcu_group_to_conam_{}'.format(agg_type)] = df['conam'] / df.groupby(['csmcu'])['conam'].transform(agg_type)
        df['csmcu_locdt_group_to_conam_{}'.format(agg_type)] = df['conam'] / df.groupby(['csmcu', 'locdt'])['conam'].transform(agg_type)


for df in [train, test]:
    df['weekly_transaction_conam_mean'] = df['conam'] / df.groupby(['date_weekly'])['conam'].transform('mean')
    df['weekly_transaction_conam_std'] = df['conam'] / df.groupby(['date_weekly'])['conam'].transform('std')

    df['cano_weekly_transaction_conam_mean'] = df['conam'] / df.groupby(['cano', 'date_weekly'])['conam'].transform('mean')
    df['cano_weekly_transaction_conam_std'] = df['conam'] / df.groupby(['cano', 'date_weekly'])['conam'].transform('std')

    df['bacno_weekly_transaction_conam_mean'] = df['conam'] / df.groupby(['bacno','date_weekly'])['conam'].transform('mean')
    df['bacno_weekly_transaction_conam_std'] = df['conam'] / df.groupby(['bacno','date_weekly'])['conam'].transform('std')

    df['mcc_weekly_transaction_conam_mean'] = df['conam'] / df.groupby(['mcc', 'date_weekly'])['conam'].transform('mean')
    df['mcc_weekly_transaction_conam_std'] = df['conam'] / df.groupby(['mcc', 'date_weekly'])['conam'].transform('std')
```

Ex: Label Encoding

```
for df in [train, test]:

    # 消費地國別＋消費城市＋狀態碼
    df['stocn_scity_stscd'] = df['scity'].astype(str)+'_'+df['stocn'].astype(str)+'_'+df['stscd'].astype(str)

    # 消費地國別＋消費地幣別＋消費城市
    df['stocn_csmcu_scity'] = df['stocn'].astype(str)+'_'+df['csmcu'].astype(str)+'_'+df['scity'].astype(str)

    # 歸卡帳戶＋卡號＋交易類別
    df['bacno_cano_contp'] = df['bacno'].astype(str)+'_'+df['cano'].astype(str)+'_'+df['contp'].astype(str)

    # 卡號＋特店代號＋消費地國別
    df['bacno_mchno_stocn'] = df['bacno'].astype(str)+'_'+df['mchno'].astype(str)+'_'+df['stocn'].astype(str)

    # 收單行代碼認證授權 + 特店代號 : reference : https://progressbar.tw/posts/75
    df['acqicn_mchno'] = df['acqic'].astype(str)+'_'+df['mchno'].astype(str)

   # 收單行 + 地區 + MCC : reference https://zi.media/@yidianzixun/post/zyDyH3
   df['acqicn_scity_mcc'] = df['acqic'].astype(str)+'_'+df['scity'].astype(str)+'_'+df['mcc'].astype(str)

   df['contp_stocn_scity'] = df['contp'].astype(str)+'_'+df['stocn'].astype(str)+'_'+df['scity'].astype(str)
```

Ex: Label Encoding and Frequency Encoding

```
## label encoding

label_encoder_list = ['contp', 'etymd', 'ecfg', 'iterm', 'flbmk',
                      'flg_3dsmk', 'insfg', 'stscd', 'hcefg','stocn', 'csmcu',
                      'mcc','acqic', 'cano', 'scity',
                      'bacno_mchno_stocn', 'stocn_csmcu_scity',
                      'stocn_scity_stscd', 'acqicn_mchno', 'acqicn_mcc',
                      'bacno_cano_locdt_etymd', 'bacno_cano_contp','acqicn_scity_mcc',
                      'contp_stocn_scity', 'contp_etymd_stocn',
                      'hcefg_acqic_csmcu', 'hcefg_ecfg_mcc', 'etymd_mchno_csmcu', 'contp_flg_3dsmk_mcc'
                     ]
for col in label_encoder_list:
    le = LabelEncoder()
    le.fit(np.concatenate([train[col].values.reshape(-1, 1).astype('str'), test[col].values.reshape(-1, 1).astype('str')]))
    train[col] = le.transform(train[col].values.reshape(-1, 1).astype('str'))
    test[col] = le.transform(test[col].values.reshape(-1, 1).astype('str'))



#Frequency encoding
for feature in ['bacno_mchno_stocn', 'stocn_scity_stscd','stocn_csmcu_scity', 'acqicn_mchno','acqicn_mcc',\
                'bacno_cano_locdt_etymd', 'bacno_cano_contp', 'acqicn_scity_mcc', 'contp_stocn_scity', 'contp_etymd_stocn',\
                'hcefg_acqic_csmcu', 'hcefg_ecfg_mcc', 'etymd_mchno_csmcu', 'contp_flg_3dsmk_mcc']:
    train[feature + '_count_full'] = train[feature].map(pd.concat([train[feature], test[feature]], ignore_index=True).value_counts(dropna=Fals
    test[feature + '_count_full'] = test[feature].map(pd.concat([train[feature], test[feature]], ignore_index=True).value_counts(dropna=False)
```

Ex: bacno or cano fieldFraud ?

```python
# 卡號或歸戶是否曾經盜刷
def fieldFraud(train, test):

    bacno_fraud = set(train[train['fraud_ind']==1]['bacno'])
    cano_fraud = set(train[train['fraud_ind']==1]['cano'])

    train['bacno_fieldFraud'] = 0
    train['cano_fieldFraud'] = 0

    banco_list = list()
    cano_list = list()

    ## training data section
    for i in range(1,91):
        print(i)

        train.loc[train['locdt'] == i,'bacno_fieldFraud'] = train[train['locdt'] == i].apply(lambda x : 1 if x['bacno'] in banco_list else 0,
        banco_list.extend(list(set(train[(train['locdt'] == i)&(train['fraud_ind']==1)]['bacno'])))

        train.loc[train['locdt'] == i,'cano_fieldFraud'] = train[train['locdt'] == i].apply(lambda x : 1 if x['bacno'] in cano_list else 0, a
        cano_list.extend(list(set(train[(train['locdt'] == i)&(train['fraud_ind']==1)]['cano'])))

    ## testing data section
    test['bacno_fieldFraud'] = test.apply(lambda x : 1 if x['bacno'] in bacno_fraud else 0, axis=1)
    test['cano_fieldFraud'] = test.apply(lambda x : 1 if x['cano'] in cano_fraud else 0, axis=1)

    return train, test
```

Ex: Nunique Feature

```
## nunique feature
for col in ['acqic', 'mcc', 'mchno', 'scity', 'conam']:
    comb = pd.concat([train[['bacno']+[col]],test[['bacno']+[col]]],axis=0)
    mp = comb.groupby('bacno')[col].agg(['nunique'])['nunique'].to_dict()
    train['bacno_'+col+'_ct'] = train['bacno'].map(mp).astype('float32')
    test['bacno_'+col+'_ct'] = test['bacno'].map(mp).astype('float32')
    print('bacno_'+col+'_ct, ',end='')


## nunique feature
for col in ['acqic', 'mcc', 'csmcu']:
    comb = pd.concat([train[['cano']+[col]],test[['cano']+[col]]],axis=0)
    mp = comb.groupby('cano')[col].agg(['nunique'])['nunique'].to_dict()
    train['cano_'+col+'_ct'] = train['cano'].map(mp).astype('float32')
    test['cano_'+col+'_ct'] = test['cano'].map(mp).astype('float32')
    print('cano_'+col+'_ct, ',end='')
```

```
for df in [train, test]:
    # 當天金額重複的筆數 & 比例
    df['cano_locdt_conam_nunique'] = df.groupby(['cano', 'locdt'])['conam'].transform('nunique')

    ind = list(df.loc[df['cano_locdt_conam_count'] <= 1].index.values)
    df.loc[ind, 'cano_locdt_conam_nunique'] = -1

    df['cano_locdt_conam_nunique_percent'] = df['cano_locdt_conam_nunique'] / df['cano_locdt_conam_count']

    df['cano_locdt_conam_nunique_percent_to_mean'] = df['cano_locdt_conam_nunique_percent'] / df.groupby(['cano'])['cano_locdt_conam_nunique_pe
```

Ex: Time interval Feature

```
start_time = time()

df_sorted_by_time = train.sort_values(['locdt', 'loctm'], ascending=[True, True])
df_sorted_by_time = df_sorted_by_time.reset_index()

df_cano_day_first = df_sorted_by_time.groupby(['cano', 'locdt']).first()

multi_index = df_cano_day_first.loc[df_cano_day_first.conam == 0].index

train['dayfirst_conam_is_zero'] = train.apply(if_first_conam_zero, axis=1)


### 與當天上一筆消費時間間隔
# train['sec_time'] = train['loctm'].apply(lambda x : change_time_to_sec(x))

df_sorted_by_time2 = train.sort_values(['cano', 'locdt', 'loctm'], ascending=[True, True, True])
df_sorted_by_time2 = df_sorted_by_time2.reset_index()

df_sorted_by_time2['last_time_interval'] = df_sorted_by_time2['sec_time'].rolling(window=2).apply(lambda x: x[1] - x[0])

# ind = list(df_sorted_by_time2.loc[df_sorted_by_time2['cano_locdt_conam_count']<=1].index.values)

# df_sorted_by_time2.loc[ind, 'last_time_interval'] = -999

txkey_ind = df_cano_day_first.txkey.values

df_sorted_by_time2.loc[df_sorted_by_time2['txkey'].isin(txkey_ind), 'last_time_interval'] = -1

df_sorted_by_time2 = df_sorted_by_time2.loc[:, ['last_time_interval', 'txkey']]

train = pd.merge(left=train, right=df_sorted_by_time2, on='txkey')

print("--- %s seconds ---" % (time() - start_time))

del df_sorted_by_time, df_cano_day_first, df_sorted_by_time2
gc.collect()
```

Ex: Time interval Feature

```
# 11/19
# 跟下一筆時間間隔
# train
df_sorted_by_time = train.sort_values(['locdt', 'loctm'], ascending=[True, True])
df_sorted_by_time = df_sorted_by_time.reset_index()

df_cano_day_last = df_sorted_by_time.groupby(['cano', 'locdt']).tail(1)


# train['sec_time'] = train['loctm'].apply(lambda x : change_time_to_sec(x))

df_sorted_by_time2 = train.sort_values(['cano', 'locdt', 'loctm'], ascending=[True, True, True])
df_sorted_by_time2 = df_sorted_by_time2.reset_index()

df_sorted_by_time2['next_time_interval'] = df_sorted_by_time2['sec_time'].shift(-1).rolling(window=2).apply(lambda x: x[0] - x[1])
df_sorted_by_time2['next_time_interval'] = df_sorted_by_time2['next_time_interval']*-1
txkey_ind = df_cano_day_last.txkey.values

df_sorted_by_time2.loc[df_sorted_by_time2['txkey'].isin(txkey_ind), 'next_time_interval'] = -1

df_sorted_by_time2 = df_sorted_by_time2.loc[:, ['next_time_interval', 'txkey']]

train = pd.merge(left=train, right=df_sorted_by_time2, on='txkey')

# train['next_time_interval'] = train['next_time_interval'].apply(lambda x: (-1)*x if x != -999 else x)
# train.loc[train['txkey'].isin(txkey_ind), 'next_time_interval'] = -1

# test
df_sorted_by_time = test.sort_values(['locdt', 'loctm'], ascending=[True, True])
df_sorted_by_time = df_sorted_by_time.reset_index()

df_cano_day_last = df_sorted_by_time.groupby(['cano', 'locdt']).tail(1)


# train['sec_time'] = train['loctm'].apply(lambda x : change_time_to_sec(x))

df_sorted_by_time2 = test.sort_values(['cano', 'locdt', 'loctm'], ascending=[True, True, True])
df_sorted_by_time2 = df_sorted_by_time2.reset_index()

df_sorted_by_time2['next_time_interval'] = df_sorted_by_time2['sec_time'].shift(-1).rolling(window=2).apply(lambda x: x[0] - x[1])
df_sorted_by_time2['next_time_interval'] = df_sorted_by_time2['next_time_interval']*-1

txkey_ind = df_cano_day_last.txkey.values

df_sorted_by_time2.loc[df_sorted_by_time2['txkey'].isin(txkey_ind), 'next_time_interval'] = -1

df_sorted_by_time2 = df_sorted_by_time2.loc[:, ['next_time_interval', 'txkey']]

test = pd.merge(left=test, right=df_sorted_by_time2, on='txkey')
```

Ex: moving average feature : rolling window : 3

```
def compute_moving_average_3_noshift(arg):

    grp, lst = arg
    res = temp_df.loc[temp_df['cano'] == grp, :]
    res['cano_locdt_move_average_3day_noshift'] = res['mean'].rolling(window=3).mean()
    res = res.reset_index()
    # print(res)
    return res


def main():
    global temp_df

    start_time = time.time()

    grp_lst_args = list(temp_df.groupby('cano').groups.items())

    pool = mp.Pool(processes=N_CORES)
    results = pool.map(compute_moving_average_3_noshift, grp_lst_args)
    pool.close()
    pool.join()

    results_df = pd.concat(results)
    results_df.to_csv('cano_locdt_move_average_3day_noshift.csv', index = False)

    print("--- %s seconds ---" % (time.time() - start_time))


if __name__ == '__main__':
    main()
```

Total Feature Used : around 200 features

Feature accumulate : ecfg

Fraud ratio : scity stocn

Number of same conam

Day, weekly, month feature

…

```
Python 3.6.4 (default, Nov 24 2018, 15:00:44)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> train = pickle.load(open('train_686.pkl', 'rb'))
>>> print(list(train.columns))
['acqic', 'bacno', 'cano', 'conam', 'contp', 'csmcu', 'ecfg', 'etymd', 'flbmk', 'flg_3dsmk', 'fraud_ind', 'hcefg', 'insfg', 'iterm', 'locdt',
'loctm', 'mcc', 'mchno', 'ovrlt', 'scity', 'stocn', 'stscd', 'txkey', 'delta_conam_mean_etymd', 'hr_highrisk', 'prop_hr_highrisk',
'prop_contp_stocn', 'prop_contp_stscd', 'ecfg_acc', 'ecfg_inc_acc', 'counts_ex1d', 'cost_mean_ex1d', 'cost_med_ex1d', 'counts_ex1hr',
'cost_mean_ex1hr', 'cost_med_ex1hr', 'hr_scity_prop', 'hr_stocn_prop', 'acqic_bacno_mean', 'cano_bacno_mean', 'contp_bacno_mean',
'csmcu_bacno_mean', 'etymd_bacno_mean', 'hcefg_bacno_mean', 'iterm_bacno_mean', 'locdt_bacno_mean', 'mcc_bacno_mean', 'mchno_bacno_mean',
'scity_bacno_mean', 'stocn_bacno_mean', 'stscd_bacno_mean', 'bacno_acqic_ct', 'bacno_mcc_ct', 'bacno_mchno_ct', 'bacno_scity_ct',
'bacno_conam_ct', 'cano_acqic_ct', 'cano_mcc_ct', 'cano_csmcu_ct', 'index_x', 'cano_locdt_move_average_3day_noshift', 'index_y',
'bacno_locdt_move_average_3day_noshift', 'cano_rolling_3day_divide_conam', 'bacno_rolling_3day_divide_conam', 'stocn_fraud_ratio_risk',
'locdt_move_average_7day_conam_mean', 'locdt_move_average_7day_conam_std', 'date_weekly', 'weekly_transaction_conam_mean',
'weekly_transaction_conam_std', 'cano_weekly_transaction_conam_mean', 'cano_weekly_transaction_conam_std',
'bacno_weekly_transaction_conam_mean', 'bacno_weekly_transaction_conam_std', 'mcc_weekly_transaction_conam_mean',
'mcc_weekly_transaction_conam_std', 'bacno_date_weekly_ct', 'date_weekly_bacno_mean', 'bacno_fieldFraud', 'cano_fieldFraud',
'cano_locdt_conam_count', 'sec_time', 'cano_locdt_conam_nunique', 'cano_locdt_conam_nunique_percent',
'cano_locdt_conam_nunique_percent_to_mean', 'dayfirst_conam_is_zero', 'last_time_interval', 'todat_last_time_binary_feature',
'todat_first_time_binary_feature', 'last_time_interval_mean', 'last_time_interval_mean_to_mean', 'last_time_interval_to_mean',
'next_time_interval', 'next_time_interval_mean', 'next_time_interval_mean_to_mean', 'next_time_interval_to_mean', 'short_last_time_interval',
'short_next_time_interval', 'date_15_day', 'mcc_date_month_transaction_conam_mean', 'etymd_count_full', 'csmcu_count_full',
'stocn_count_full', 'mcc_count_full', 'acqic_count_full', 'mchno_count_full', 'contp_count_full', 'scity_count_full', 'loctm_minus_loctm_max',
'loctm_minus_loctm_min', 'bacno_cano_group_to_conam_mean', 'bacno_cano_locdt_group_to_conam_mean', 'csmcu_group_to_conam_mean',
'csmcu_locdt_group_to_conam_mean', 'bacno_cano_group_to_conam_std', 'bacno_cano_locdt_group_to_conam_std', 'csmcu_group_to_conam_std',
'csmcu_locdt_group_to_conam_std', 'first_time_money_mean', 'last_time_money_mean', 'first_time_money_std', 'last_time_money_std',
'stscd_conam_mean', 'stocn_conam__mean', 'mchno_conam_mean', 'mcc_conam_mean', 'stscd_conam_std', 'stocn_conam__std', 'mchno_conam_std',
'mcc_conam_std', 'stocn_scity_stscd', 'stocn_csmcu_scity', 'bacno_cano_contp', 'bacno_cano_locdt_etymd', 'bacno_mchno_stocn', 'acqicn_mchno',
'acqicn_mcc', 'acqicn_scity_mcc', 'contp_stocn_scity', 'contp_etymd_stocn', 'hcefg_acqic_csmcu', 'hcefg_ecfg_mcc', 'etymd_mchno_csmcu',
'contp_flg_3dsmk_mcc', 'acqic_target_encoding', 'mcc_target_encoding', 'csmcu_target_encoding', 'bacno_mchno_stocn_count_full',
'stocn_scity_stscd_count_full', 'stocn_csmcu_scity_count_full', 'acqicn_mchno_count_full', 'acqicn_mcc_count_full',
'bacno_cano_locdt_etymd_count_full', 'bacno_cano_contp_count_full', 'acqicn_scity_mcc_count_full', 'contp_stocn_scity_count_full',
'contp_etymd_stocn_count_full', 'hcefg_acqic_csmcu_count_full', 'hcefg_ecfg_mcc_count_full', 'etymd_mchno_csmcu_count_full',
'contp_flg_3dsmk_mcc_count_full', 'bacno_mchno_stocn_conam_mean', 'bacno_mchno_stocn_conam_std', 'stocn_csmcu_scity_conam_mean',
'stocn_csmcu_scity_conam_std', 'stocn_scity_stscd_conam_mean', 'stocn_scity_stscd_conam_std', 'acqicn_mchno_conam_mean',
'acqicn_mchno_conam_std', 'acqicn_mcc_conam_mean', 'acqicn_mcc_conam_std', 'bacno_cano_locdt_etymd_conam_mean',
'bacno_cano_locdt_etymd_conam_std', 'bacno_cano_contp_conam_mean', 'bacno_cano_contp_conam_std', 'acqicn_scity_mcc_conam_mean',
'acqicn_scity_mcc_conam_std', 'contp_stocn_scity_conam_mean', 'contp_stocn_scity_conam_std', 'contp_etymd_stocn_conam_mean',
'contp_etymd_stocn_conam_std', 'hcefg_acqic_csmcu_conam_mean', 'hcefg_acqic_csmcu_conam_std', 'hcefg_ecfg_mcc_conam_mean',
'hcefg_ecfg_mcc_conam_std', 'etymd_mchno_csmcu_conam_mean', 'etymd_mchno_csmcu_conam_std', 'contp_flg_3dsmk_mcc_conam_mean',
'contp_flg_3dsmk_mcc_conam_std', 'num_same_conam_sameday', 'same_conam_sameday_ratio', 'num_sameday_bef_has_0_conam', 'if_bef_has_zero_conam',
'bacno_cano_etymd_ecfg_stscd_to_conam_mean', 'bacno_cano_etymd_ecfg_stscd_to_conam_std', 'cents', 'conam_acqic_mean', 'conam_acqic_std',
'conam_cano_acqic_mean', 'conam_cano_acqic_std', 'ecfg_mcc', 'stscd_ecfg_acqic', 'ecfg_stocn_scity', 'ecfg_mcc_count_full',
'stscd_ecfg_acqic_count_full', 'ecfg_stocn_scity_count_full', 'ecfg_mcc_conam_mean', 'stscd_ecfg_acqic_conam_mean',
'ecfg_stocn_scity_conam_mean']
>>> print(len(list(train.columns))
... )
210
```

XGBoost, LGBM and Catboost

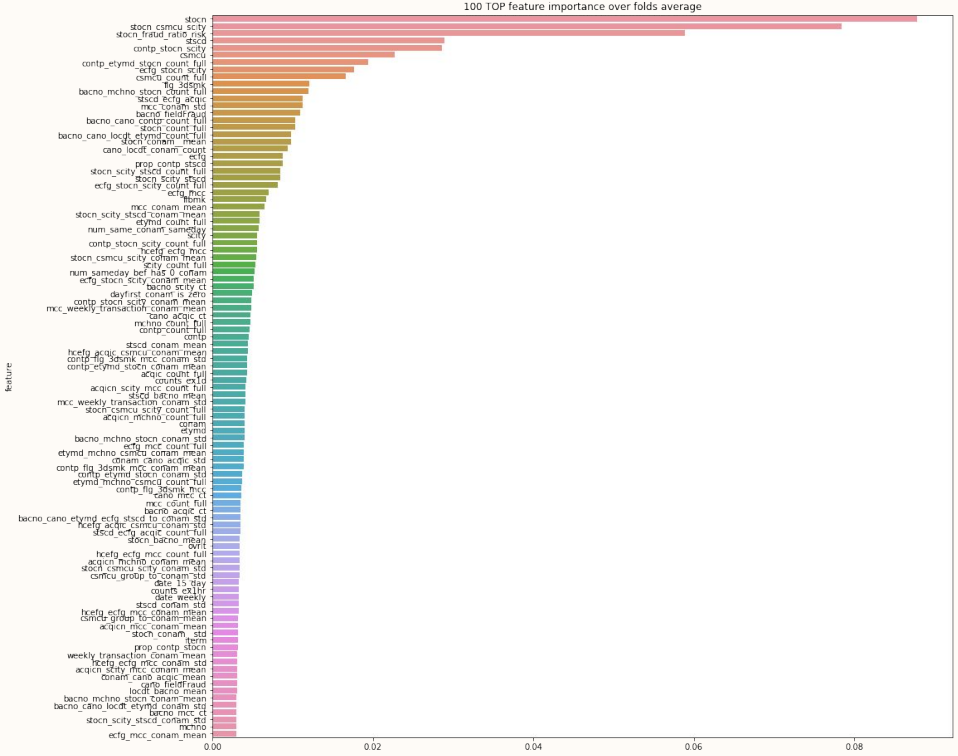
```
train = pickle.load(open('./train_686.pkl', 'rb')).drop(columns=['bacno_conam_ct', 'hr_highrisk', 'prop_hr_highrisk', 'acqic_target_encoding',
test = pickle.load(open('./test_686.pkl', 'rb')).drop(columns=['bacno_conam_ct', 'hr_highrisk', 'prop_hr_highrisk', 'acqic_target_encoding',
sub = pd.read_csv('submission_test.csv')

# testing code
# train = train.sample(n=20000).reset_index(drop=True)


train = train.fillna(-999)
test = test.fillna(-999)


features_columns = list(train.columns)
not_use_feature_columns = [ 'fraud_ind', 'txkey','sec_time',
                            'todat_first_time_binary_feature', 'todat_last_time_binary_feature',
                           'loctm_minus_loctm_max', 'loctm_minus_loctm_min',
                            'cano_locdt_conam_nunique_percent', 'cano_locdt_conam_nunique',
                           'last_time_interval_mean', 'last_time_interval',
                           'next_time_interval_mean', 'next_time_interval',
                           'bacno_locdt_move_average_3day_noshift', 'cano_locdt_move_average_3day_noshift',
                           'bacno', 'cano',
                          ]

for i in not_use_feature_columns:
    print(i)
    features_columns.remove(i)




SEED = 777
seed_everything(SEED)
LOCAL_TEST = False
TARGET = 'fraud_ind'


print(f'The total of features : {len(features_columns)}')

NFOLDS = 6
feature_importances = pd.DataFrame()
feature_importances['feature'] = train[features_columns].columns

folds = GroupKFold(n_splits=NFOLDS)

```

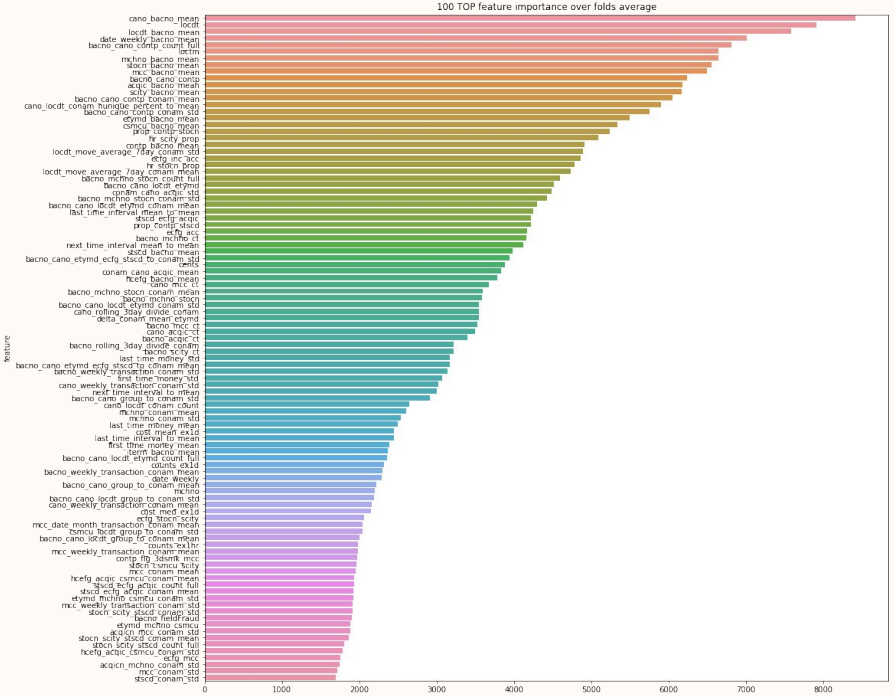
XGBoost, LGBM and Catboost

```
for fold_, (trn_idx, val_idx) in enumerate(folds.split(X, y, groups=split_groups)):
    print('Fold:',fold_)

    tr_x, tr_y = X.iloc[trn_idx,:], y[trn_idx]
    vl_x, vl_y = X.iloc[val_idx,:], y[val_idx]

    print(len(tr_x),len(vl_x))
    print('train fraud total : {}'.format(len(np.where(tr_y>0)[0])))
    print('validation fraud total : {}'.format(len(np.where(vl_y>0)[0])))

    clf = xgb.XGBClassifier(
            n_estimators=100000,
            max_depth=12,
            learning_rate=0.01,
            subsample=0.8,
            colsample_bytree=0.4,
            missing=-999,
            eval_metric='auc',
            # need gpu
            tree_method='gpu_hist'
        )
    estimator = clf.fit(tr_x, tr_y,
                eval_set=[(vl_x, vl_y)],
                verbose=500, early_stopping_rounds=500)

    gc.collect()
    pp_p = estimator.predict_proba(P)[:,1]
    predictions += pp_p/NFOLDS

    oof[val_idx] += estimator.predict_proba(vl_x)[:,1]
    feature_importances['fold_{}'.format(fold_ + 1)] = clf.feature_importances_


    print(threshold_search_fold(vl_y, oof[val_idx]))
    del tr_x, tr_y, vl_x, vl_y
    del estimator, clf
    gc.collect()

print('OOF AUC:', roc_auc_score(y, oof))
```

Model
17

Feature Importance Plot





Blending, Vote and Stacking record

9	11/05/2019			0.684015	X	0.68654	0.15	7721	0.680906	
10	11/06/2019			0.684015	0.681076	0.68654	0.17	7559	0.683969	
11	11/06/2019		v40	0.678944	0.676606	0.68654	0.18	7416	0.689046	
12	11/06/2019			0.678944	0.681076	0.68654	0.19	7120	0.692152	跟689301差103/108
13	11/06/2019			0.679535	0.681076	0.68654	0.19	7172	0.690551	
14	11/06/2019			0.678944	0.681076	xgb 2 686	0.2	6974	0.693618	跟692152差0/146
15	11/07/2019			0.678944	0.681076	xgb 2 686 + 682 xgb	0.21	6827	0.688532	跟693618差1/148
16	11/07/2019			0.684164	0.681076	xgb 2 686	0.18	7329	0.688815	跟693618差376/21
17	11/07/2019			0.684164	0.681076	xgb 2 686	0.2	6961	0.691644	跟693618差108/121
18	11/08/2019			0.678944	0.68117	xgb 686 and 685	0.2	6894	0.692725	
19	11/08/2019			678+681+684	0.681076+catv26	xgb 2 686 and 2 685	0.2	6878	0.693467	跟693618差20/116
20	11/09/2019			684	681	686	0.16	7840	0.682237	跟693618差870/4
21	11/09/2019			684	6811	xgb 2 686	0.2		0.691838	跟693618差104/49
22	11/10/2019		v1	678	676+681	xgb 2 686	0.21	6904	0.691538	跟693618差104/49
23	11/10/2019		v4	678	676+681	xgb 2 686	0.21	6784	0.69029	跟693618差104/49
24										
25	11/12/2019		Vote	693467	692725	693618	692152	6930	693515	
26				678	676+681+681	686+687+689	0.21	6937	0.69	77/114
						687+689	0.16	7249	0.690804	跟693618差445/170

A	B	C	D
ExtraTreesClassifier12models_100_8_5_5_36.csv			0.687528
ExtraTreesClassifier12models_10_5_5_5_36.csv			0.689686
Lasso10models_001_50000.csv			0.687727

Submit record

xgb 6fold 686 v16			v1	0.16	0.726901427	6934	0.686873	na	686 features	6 gkfold			693 -> 327/287
xgb 6fold 686 v16			v3	0.15	0.7286627528	7029	0.685495		686 features	6 gkfold	bacno nest		
xgb 6fold 686 v16			v4	0.15	0.729475932	7322	0.685156		686 features	6 gkfold	bacno nest	prop_hr_highris	693 -> 482/288
xgb 6fold 686 v16			v5	0.16	0.7251814111	7168			686 features	6 gkfold	no bacno nest	prop_hr_highris	693 -> 587/229
xgb 6fold 686 v16			v9	0.15	0.7523579962	6600			686 features	6 gkfold	mcc acqic scity mean encoding		
			v13	0.16	0.7289995342	7086	0.681542		686 features	6 gkfold	mcc acqic csmcu target encoding		
xgb-686-feature_selection-model			v6	0.16	0.732818027	6813	0.680138		686 features	6 gkfold	remove bacno_conam_ct	693 -> 345/506	a
xgb-686-feature_selection-model			v11	0.18	0.6898588949	7408	0.677384						
xgb-686-feature_selection-model			v20	0.17	0.7280897681	6635	0.689158		686 features	6 gkfold	remove bacno_conam_ct	693 -> 204/543	add can
											early_stopping=500, seed=777	cents	conam_
xgb-686-feature_selection-model			v26	0.2	0.8748681898	6238	0.680658						
xgb-686-feature_selection-model			v34	0.15	0.7250089482	7580	0.678201						
xgb-686-feature_selection-model			v42	0.16	0.7275528054	6958			samp v1			693 -> 294/310	
xgb-686-feature_selection-model			v44	0.17	0.7267908727	6758	0.687198					693 -> 217/433	conam_
xgb-686-feature_selection-model			v45	0.25	0.8736191345	5837	0.668796						
colab			v1	0.17	0.7285276074	6680	0.681471		target encoding			693 -> 374/668	

Final results 7 / 1366?? , some of people who cheat a little bit.

7	松下問童子 佳作	5	260	0.685949	11/15/2019 11:56:12 PM

????

Thanks for listening !