American Express - Default Prediction

- 대회 내용 : 고객이 미래의 채무 불이행 여부를 예측
- 대회 링크 : https://www.kaggle.com/competitions/amex-default-prediction (https://www.kaggle.com/competitions/amex-default-prediction)
- 대회 평가 : M = 0.5 * (G + D)
  - G : Normalized Gini Coefficient
  - D : 4%에서의 기본 비율(default rate)
- 평가 파일
  - customer_ID, prediction
  - 각 고객 ID별 채무 불이행을 예측 후, 제출

학습 목표

- xgboost를 활용한 기본 모델을 만들어 제출해봅니다.

목차

참조 URL 링크

- URL : https://www.kaggle.com/code/drrajkulkarni/576-tuned-xgbm (https://www.kaggle.com/code/drrajkulkarni/576-tuned-xgbm)

01. 라이브러리 불러오기

목차로 이동하기

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix

import xgboost as xgb
from xgboost import XGBClassifier

import warnings, gc
warnings.filterwarnings("ignore")
```

02. 데이터 불러오기

목차로 이동하기

In [2]:

```
%%time
train = pd.read_parquet("../input/amex-data-integer-dtypes-parquet-format/train.parquet")
label = pd.read_csv("../input/amex-default-prediction/train_labels.csv")
train = train.merge(label,how='inner',on="customer_ID")
```

```
CPU times: user 2min 21s, sys: 30 s, total: 2min 51s
Wall time: 2min 49s
```

In [3]:

```
print(train.shape)
train.head(3)
```

```
(5531451, 191)
```

Out[3]:

	customer_ID	S_2	P_2	D_39	B_1	B_2
0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-03-09	0.938469	0	0.008724	1.006838
1	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-04-07	0.936665	0	0.004923	1.000653
2	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-05-28	0.954180	3	0.021655	1.009672

3 rows × 191 columns

03. customer_ID의 컬럼의 라벨 인코딩을 수행 및 인덱스 지정

목차로 이동하기

In [4]:

```
lab = LabelEncoder()
train['customer_ID']= lab.fit_transform(train['customer_ID'])
```

In [5]:

```
%%time
train = train.groupby(['customer_ID']).tail(1).set_index('customer_ID')
```

```
CPU times: user 1.81 s, sys: 1.47 s, total: 3.29 s
Wall time: 3.29 s
```

In [6]:

```
print( train.shape )
train.head()
gc.collect()
```

```
(458913, 190)
```

Out[6]:

```
21
```

04. 테스트 데이터 불러오기

목차로 이동하기

In [7]:

```
%%time
test = pd.read_parquet("../input/amex-data-integer-dtypes-parquet-format/test.parquet")
```

```
CPU times: user 20.3 s, sys: 13.6 s, total: 33.9 s
Wall time: 36.6 s
```

In [8]:

```
print(test.shape)
test.head()
gc.collect()
```

```
(11363762, 190)
```

Out[8]:

```
42
```

05. test 데이터 셋도 customer_ID로 라벨 인코딩

목차로 이동하기

In [9]:

```
test['customer_ID']= lab.fit_transform(test['customer_ID'])
test = test.groupby(['customer_ID']).tail(1).set_index('customer_ID')
```

06. 데이터 나누기 및 결측치 처리

목차로 이동하기

In [10]:

```
y = train.target
X = train.drop(["target","S_2"],axis=1)
test = test.drop(['S_2'],axis=1)

X = X.fillna(-123)
test = test.fillna(-123)
```

In [11]:

```
y.value_counts()
```

Out[11]:

```
0    340085
1    118828
Name: target, dtype: int64
```

In [12]:

```
print(X.shape, y.shape, test.shape)
gc.collect()
```

```
(458913, 188) (458913,) (924621, 188)
```

Out[12]:

```
42
```

07. 범주형과 수치형 컬럼을 나누기

목차로 이동하기

In [13]:

```
cat_cols = ['B_30', 'B_38', 'D_63', 'D_64', 'D_66',
            'D_68', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126']

num_cols = [col for col in X.columns if col not in cat_cols]

all_cols = [cat_cols,num_cols]
```

In [14]:

```
print(all_cols)
```

```
[['B_30', 'B_38', 'D_63', 'D_64', 'D_66', 'D_68', 'D_114', 'D_116', 'D_117', 'D_12
0', 'D_126'], ['P_2', 'D_39', 'B_1', 'B_2', 'R_1', 'S_3', 'D_41', 'B_3', 'D_42', 'D_
43', 'D_44', 'B_4', 'D_45', 'B_5', 'R_2', 'D_46', 'D_47', 'D_48', 'D_49', 'B_6', 'B_
7', 'B_8', 'D_50', 'D_51', 'B_9', 'R_3', 'D_52', 'P_3', 'B_10', 'D_53', 'S_5', 'B_1
1', 'S_6', 'D_54', 'R_4', 'S_7', 'B_12', 'S_8', 'D_55', 'D_56', 'B_13', 'R_5', 'D_5
8', 'S_9', 'B_14', 'D_59', 'D_60', 'D_61', 'B_15', 'S_11', 'D_62', 'D_65', 'B_16',
'B_17', 'B_18', 'B_19', 'B_20', 'S_12', 'R_6', 'S_13', 'B_21', 'D_69', 'B_22', 'D_7
0', 'D_71', 'D_72', 'S_15', 'B_23', 'D_73', 'P_4', 'D_74', 'D_75', 'D_76', 'B_24',
'R_7', 'D_77', 'B_25', 'B_26', 'D_78', 'D_79', 'R_8', 'R_9', 'S_16', 'D_80', 'R_10',
'R_11', 'B_27', 'D_81', 'D_82', 'S_17', 'R_12', 'B_28', 'R_13', 'D_83', 'R_14', 'R_1
5', 'D_84', 'R_16', 'B_29', 'S_18', 'D_86', 'D_87', 'R_17', 'R_18', 'D_88', 'B_31',
'S_19', 'R_19', 'B_32', 'S_20', 'R_20', 'R_21', 'B_33', 'D_89', 'R_22', 'R_23', 'D_9
1', 'D_92', 'D_93', 'D_94', 'R_24', 'R_25', 'D_96', 'S_22', 'S_23', 'S_24', 'S_25',
'S_26', 'D_102', 'D_103', 'D_104', 'D_105', 'D_106', 'D_107', 'B_36', 'B_37', 'R_2
6', 'R_27', 'D_108', 'D_109', 'D_110', 'D_111', 'B_39', 'D_112', 'B_40', 'S_27', 'D_
113', 'D_115', 'D_118', 'D_119', 'D_121', 'D_122', 'D_123', 'D_124', 'D_125', 'D_12
7', 'D_128', 'D_129', 'B_41', 'B_42', 'D_130', 'D_131', 'D_132', 'D_133', 'R_28', 'D
_134', 'D_135', 'D_136', 'D_137', 'D_138', 'D_139', 'D_140', 'D_141', 'D_142', 'D_14
3', 'D_144', 'D_145']]
```

08. 변수 구분

목차로 이동하기

- D_* = Delinquency variables(연체 변수)
- S_* = Spend variables(지출 변수)
- P_* = Payment variables(지불 변수)
- B_* = Balance variables(균형 변수)
- R_* = Risk variables(위험 변수)

In [15]:

```
D_n_cols = [col for col in num_cols if col.startswith("D")]
S_n_cols = [col for col in num_cols if col.startswith("S")]
P_n_cols = [col for col in num_cols if col.startswith("P")]
B_n_cols = [col for col in num_cols if col.startswith("B")]
R_n_cols = [col for col in num_cols if col.startswith("R")]
D_c_cols = [col for col in cat_cols if col.startswith("D")]
B_c_cols = [col for col in cat_cols if col.startswith("B")]
```

In [16]:

```
print( len(D_n_cols),  len(S_n_cols), len(P_n_cols)  )
print( len(B_n_cols),len(R_n_cols), len(D_c_cols),len(B_c_cols) )
```

```
87 21 3
38 28 9 2
```

09. 변수별 컬럼명 확인 및 새로운 변수 생성

[목차로 이동하기](#)

In [17]:

```
X_num_agg_D = X.groupby("customer_ID")[D_n_cols].agg(['mean','min', 'last'])
X_num_agg_D.columns = ['_'.join(x) for x in X_num_agg_D.columns]
print( X_num_agg_D.columns)

del X_num_agg_D
gc.collect()
```

```
Index(['D_39_mean', 'D_39_min', 'D_39_last', 'D_41_mean', 'D_41_min',
       'D_41_last', 'D_42_mean', 'D_42_min', 'D_42_last', 'D_43_mean',
       ...
       'D_142_last', 'D_143_mean', 'D_143_min', 'D_143_last', 'D_144_mean',
       'D_144_min', 'D_144_last', 'D_145_mean', 'D_145_min', 'D_145_last'],
      dtype='object', length=261)
```

Out[17]:

```
0
```

In [18]:

```
%%time
X_num_agg_D = X.groupby("customer_ID")[D_n_cols].agg(['mean','min', 'last'])
X_num_agg_D.columns = ['_'.join(x) for x in X_num_agg_D.columns]

X_num_agg_S = X.groupby("customer_ID")[S_n_cols].agg(['mean','min', 'last'])
X_num_agg_S.columns = ['_'.join(x) for x in X_num_agg_S.columns]

X_num_agg_P = X.groupby("customer_ID")[P_n_cols].agg(['mean','min','max' ,'last'])
X_num_agg_P.columns = ['_'.join(x) for x in X_num_agg_P.columns]

X_num_agg_B = X.groupby("customer_ID")[B_n_cols].agg(['mean','min', 'last'])
X_num_agg_B.columns = ['_'.join(x) for x in X_num_agg_B.columns]

X_num_agg_R = X.groupby("customer_ID")[R_n_cols].agg(['mean','min','last'])
X_num_agg_R.columns = ['_'.join(x) for x in X_num_agg_R.columns]

X_cat_agg_D = X.groupby("customer_ID")[D_c_cols].agg([ 'count','last','first','nunique'])
X_cat_agg_D.columns = ['_'.join(x) for x in X_cat_agg_D.columns]

X_cat_agg_B = X.groupby("customer_ID")[B_c_cols].agg([ 'count','last','nunique'])
X_cat_agg_B.columns = ['_'.join(x) for x in X_cat_agg_B.columns]

X = pd.concat([X_num_agg_D, X_num_agg_S,X_num_agg_P,X_num_agg_B,X_num_agg_R,X_cat_agg_D,X_cat_agg_B]
del X_num_agg_D, X_num_agg_S,X_num_agg_P,X_num_agg_B,X_num_agg_R,X_cat_agg_D,X_cat_agg_B
_ = gc.collect()

print('X shape after engineering', X.shape)
```

```
X shape after engineering (458913, 576)
CPU times: user 7.91 s, sys: 941 ms, total: 8.85 s
Wall time: 8.85 s
```

In [19]:

```
X.head()
```

Out[19]:

customer_ID	D_39_mean	D_39_min	D_39_last	D_41_mean	D_41_min	D_41_last	D_42_mean
0	0.0	0	0	0.0	0.0	0.0	-123.0
1	6.0	6	6	0.0	0.0	0.0	-123.0
2	0.0	0	0	0.0	0.0	0.0	-123.0
3	0.0	0	0	0.0	0.0	0.0	-123.0
4	0.0	0	0	0.0	0.0	0.0	-123.0

5 rows × 576 columns

In [20]:

```
%%time
test_num_agg_D = test.groupby("customer_ID")[D_n_cols].agg(['mean','min', 'last'])
test_num_agg_D.columns = ['_'.join(x) for x in test_num_agg_D.columns]

test_num_agg_S = test.groupby("customer_ID")[S_n_cols].agg(['mean','min', 'last'])
test_num_agg_S.columns = ['_'.join(x) for x in test_num_agg_S.columns]

test_num_agg_P = test.groupby("customer_ID")[P_n_cols].agg(['mean','min','max', 'last'])
test_num_agg_P.columns = ['_'.join(x) for x in test_num_agg_P.columns]

test_num_agg_B = test.groupby("customer_ID")[B_n_cols].agg(['mean','min', 'last'])
test_num_agg_B.columns = ['_'.join(x) for x in test_num_agg_B.columns]

test_num_agg_R = test.groupby("customer_ID")[R_n_cols].agg(['mean','min', 'last'])
test_num_agg_R.columns = ['_'.join(x) for x in test_num_agg_R.columns]

test_cat_agg_D = test.groupby("customer_ID")[D_c_cols].agg(['count','first', 'last','nunique'])
test_cat_agg_D.columns = ['_'.join(x) for x in test_cat_agg_D.columns]

test_cat_agg_B = test.groupby("customer_ID")[B_c_cols].agg([ 'count','last','nunique'])
test_cat_agg_B.columns = ['_'.join(x) for x in test_cat_agg_B.columns]

test = pd.concat([test_num_agg_D, test_num_agg_S,test_num_agg_P,test_num_agg_B,test_num_agg_R,test_c
del test_num_agg_D, test_num_agg_S,test_num_agg_P,test_num_agg_B,test_num_agg_R,test_cat_agg_D,test_
_ = gc.collect()

print('Test shape after engineering', test.shape)
```

```
Test shape after engineering (924621, 576)
CPU times: user 16 s, sys: 1.86 s, total: 17.9 s
Wall time: 17.9 s
```

10. xgboost 모델 파라미터 설정

[목차로 이동하기](#)

- 참조 : [https://xgboost.readthedocs.io/en/stable/parameter.html (https://xgboost.readthedocs.io/en/stable/parameter.html)](https://xgboost.readthedocs.io/en/stable/parameter.html)

파라미터 이름	상세 설명	기타
booster	사용할 Booster (gblinear, dart, gbtree, dart 등)	ooo
n_estimators	사용할 트리의 개	ooo
subsample	학습 인스턴스의 하위 샘플 비율.0.5로 설정시, XGBoost가 나무를 성장시키기 전에 학습 데이터의 절반을 무작위로 샘플링.	default=1
max_depth	나무의 최대 깊이. 깊은 트리는 메모리 소비 크다.	default=6
min_child_weight	자식에게 필요한 인스턴스 가중치의 최소 합계. min_child_weight가 클수록 알고리즘이 더 보수적	default=1
eta	과적합을 방지하기 위해 업데이트에 사용되는 단계 크기 축소	default=0.3 - learning_rate
lambda	가중치에 대한 L2 정규화 항. 커지면 보수적.	default=1 (reg_lambda)
alpha	가중치에 대한 L1 정규화 항. 커지면 보수적.	default=0 (reg_alpha)
gamma	트리의 리프 노드에서 추가 파티션을 만드는데 필요한 최소 손실 감소. 클수록 더 보수적.	default=0, alias: min_split_loss
grow_policy	depthwise : 루틍 가장 가까운 노드에서 분할. lossguide : 손실 변화가 가장 큰 노드에서 분할.	default=depthwise
sample_type	샘플링 알고리즘 타	default='uniform'
normalize_type	정규화 알고리즘의 유형	default='tree'
rate_drop	드롭아웃 비율(드롭아웃 동안 드롭할 이전 트리의 일부)	default=0.0

In [21]:

```
xgb_parms ={
    'booster': 'dart',
     'n_jobs':4,
     'n_estimators':1000,
    'lambda': 4.091409953463271e-08,
    'alpha': 3.6353429991712695e-08,
    'subsample': 0.6423675532438815,
    'colsample_bytree': 0.7830450413657872,
    'max_depth': 9,
    'min_child_weight': 5,
    'eta': 0.3749337530972536,
    'gamma': 0.0745370910451703,
    'grow_policy': 'depthwise',
    'sample_type': 'uniform',
    'normalize_type': 'tree',
    'rate_drop': 0.0723975209176045,
    'skip_drop': 0.9026367296518939}
```

11. 데이터 나누기 및 학습, 평가

[목차로 이동하기](#)

Type *Markdown* and LaTeX: α^2

In [22]:

```
X_train,X_valid,y_train,y_valid = train_test_split(X, y, test_size=0.25,stratify=y)
```

In [23]:

```
my_model = XGBClassifier(**xgb_parms)
my_model.fit(X_train, y_train,
             early_stopping_rounds=10,
             eval_set=[(X_valid, y_valid)],
             verbose=1)
```

```
[0]	validation_0-logloss:0.48681
[1]	validation_0-logloss:0.38871
[2]	validation_0-logloss:0.33252
[3]	validation_0-logloss:0.29827
[4]	validation_0-logloss:0.27661
[5]	validation_0-logloss:0.26264
[6]	validation_0-logloss:0.25410
[7]	validation_0-logloss:0.24806
[8]	validation_0-logloss:0.24436
[9]	validation_0-logloss:0.24168
[10]	validation_0-logloss:0.23984
[11]	validation_0-logloss:0.23855
[12]	validation_0-logloss:0.23804
[13]	validation_0-logloss:0.23750
[14]	validation_0-logloss:0.23714
[15]	validation_0-logloss:0.23691
[16]	validation_0-logloss:0.23679
[17]	validation_0-logloss:0.23679
[18]	validation_0-logloss:0.23685
[19]	validation_0-logloss:0.23692
[20]	validation_0-logloss:0.23684
[21]	validation_0-logloss:0.23684
[22]	validation_0-logloss:0.23717
[23]	validation_0-logloss:0.23739
[24]	validation_0-logloss:0.23768
[25]	validation_0-logloss:0.23800
[26]	validation_0-logloss:0.23808
```

Out[23]:

```
XGBClassifier(alpha=3.6353429991712695e-08, base_score=0.5, booster='dart',
              callbacks=None, colsample_bylevel=1, colsample_bynode=1,
              colsample_bytree=0.7830450413657872, early_stopping_rounds=None,
              enable_categorical=False, eta=0.3749337530972536,
              eval_metric=None, gamma=0.0745370910451703, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', lambda=4.091409953463271e-08,
              learning_rate=0.374933749, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=9, max_leaves=0, min_child_weight=5,
              missing=nan, monotone_constraints='()', n_estimators=1000,
              n_jobs=4, normalize_type='tree', num_parallel_tree=1, ...)
```

In [24]:

```
pred_val = my_model.predict(X_valid)
```

In [25]:

```
cf = classification_report(y_valid,pred_val)
print(cf)
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93     85022
           1       0.80      0.79      0.80     29707

    accuracy                           0.89    114729
   macro avg       0.86      0.86      0.86    114729
weighted avg       0.89      0.89      0.89    114729
```

12. 혼동 행렬를 이용한 시각화

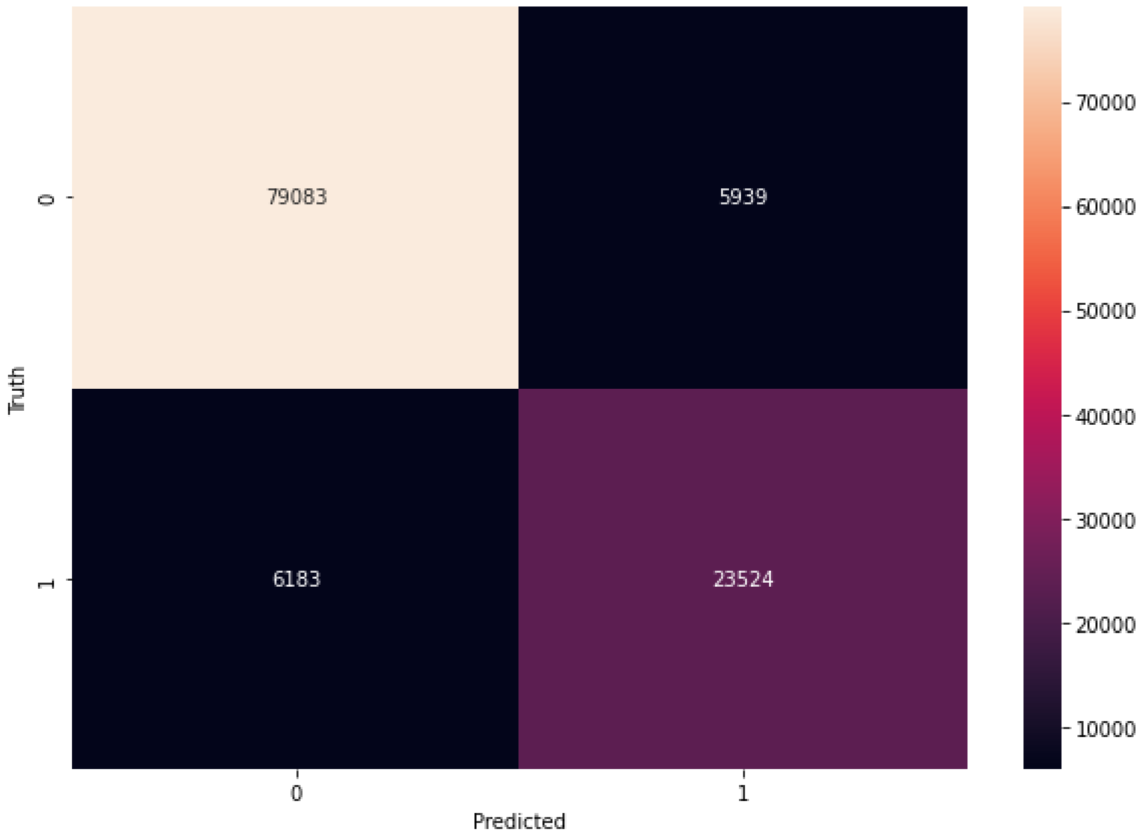
목차로 이동하기

In [26]:

```
cm = confusion_matrix(y_valid,pred_val)

plt.figure(figsize=(10,7))

sns.heatmap(cm,annot=True,fmt='d')

plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[26]:

```
Text(69.0, 0.5, 'Truth')
```



13. 예측을 수행(1을 예측할 확률), 제출

[목차로 이동하기](#)

In [27]:

```
pred_test = my_model.predict_proba(test)
```

In [28]:

```
preds = pd.DataFrame(pred_test)
pred_final = np.array(preds[1])
pred_final
```

Out[28]:

```
array([0.03345069, 0.00519914, 0.02148229, ..., 0.24811423, 0.42536047,
       0.19586097], dtype=float32)
```

In [29]:

```
submission = pd.read_csv("../input/amex-default-prediction/sample_submission.csv")
```

In [30]:

```
submission['prediction']=pred_final
submission
```

Out[30]:

	customer_ID	prediction
0	00000469ba478561f23a92a868bd366de6f6527a684c9a...	0.033451
1	00001bf2e77ff879fab36aa4fac689b9ba411dae63ae39...	0.005199
2	0000210045da4f81e5f122c6bde5c2a617d03eef67f82c...	0.021482
3	00003b41e58ede33b8daf61ab56d9952f17c9ad1c3976c...	0.282343
4	00004b22eaeeeb0ec976890c1d9bfc14fd9427e98c4ee9...	0.857462
...	...	...
924616	ffff952c631f2c911b8a2a8ca56ea6e656309a83d2f64c...	0.037001
924617	ffffcf5df59e5e0bba2a5ac4578a34e2b5aa64a1546cd3...	0.730337
924618	ffffd61f098cc056dbd7d2a21380c4804bbfe60856f475...	0.248114
924619	ffffddef1fc3643ea179c93245b68dca0f36941cd83977...	0.425360
924620	fffffa7cf7e453e1acc6a1426475d5cb9400859f82ff61...	0.195861

924621 rows × 2 columns

In [31]:

```
submission.to_csv("submission.csv",index=False)
```