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)
- 코드 참조 링크 : https://www.kaggle.com/code/wanko5452/catboost-md-10-gpu-0-794-lb (https://www.kaggle.com/code/wanko5452/catboost-md-10-gpu-0-794-lb)
- 대회 평가 : M = 0.5 * (G + D)
    - G : Normalized Gini Coefficient
    - D : 4%에서의 기본 비율(default rate)
- 데이터 셋
    - train_data : 16.39 GB, test_data : 33.82 GB

학습 목표

- CatBoost 알고리즘을 활용한 데이터 EDA 부터, 기본 모델을 만들어 제출해봅니다.

목차

01. 라이브러리 불러오기

목차로 이동하기

In [1]:

```
import random

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

from sklearn.model_selection import KFold, StratifiedKFold, train_test_split
from sklearn.metrics import roc_auc_score

from catboost import CatBoostClassifier


import gc
import datetime
from IPython.display import display
import warnings
```

Vaex.ml

- vaex.ml의 API는 scikit-learn에 근접하면서, 데이터 처리를 수행하기 위한 더 나은 성능과 기능을 제공
- 사용 가능한 데이터보다 더 큰 RAM 에 대해

In [2]:

```
import vaex
vaex.multithreading.thread_count_default = 8
import vaex.ml

from cycler import cycler

from colorama import Fore, Back, Style
```

In [3]:

```
# config
MAX_DEPTH = 10
ITERATIONS = 28000

DATA_PATH = '../input/amex-data-integer-dtypes-parquet-format/'   # denoised data fr
LABELS_PATH = '../input/amex-default-prediction/train_labels.csv' # original data so

TEST_FEAT_PATH = '../input/amex-denoised-aggregated-features/test_feat.parquet' # ag
TRAIN_FEAT_PATH = '../input/amex-denoised-aggregated-features/train_feat.parquet'
```

02. 함수 정의

[목차로 이동하기](#)

- 데이터 불러오기
- 특성 중요도 그리기
- 평가 지표

특성 중요도 그리기

In [4]:

```
def plot_feature_importance(importance,names,model_type, n=50, figsize=(16,10)):

    # 특징명과 특징의 중요도로 배열을 만든다.
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    # 딕셔너리를 이용하여 데이터 프레임을 생성
    data={'feature_names':feature_names,'feature_importance':feature_importance}
    fi_df = pd.DataFrame(data)

    # 특징 중요도를 정렬
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)
    fi_df = fi_df[:n]

    # Define size of bar plot
    plt.figure(figsize=figsize)
    # Plot Searborn bar chart
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    # 그래프의 레이브를 추가
    plt.title(model_type + ' FEATURE IMPORTANCE')
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
    plt.tight_layout()
    plt.show()
    return fi_df.feature_names
```

amex 평가 지표

In [5]:

```
def amex_metric(y_true: np.array, y_pred: np.array) -> float:

    # count of positives and negatives
    n_pos = y_true.sum()
    n_neg = y_true.shape[0] - n_pos

    # sorting by describing prediction values
    indices = np.argsort(y_pred)[::-1]
    preds, target = y_pred[indices], y_true[indices]

    # filter the top 4% by cumulative row weights
    weight = 20.0 - target * 19.0
    cum_norm_weight = (weight / weight.sum()).cumsum()
    four_pct_filter = cum_norm_weight <= 0.04

    # default rate captured at 4%
    d = target[four_pct_filter].sum() / n_pos

    # weighted Gini coefficient
    lorentz = (target / n_pos).cumsum()
    gini = ((lorentz - cum_norm_weight) * weight).sum()

    # max weighted Gini coefficient
    gini_max = 10 * n_neg * (1 - 19 / (n_pos + 20 * n_neg))

    # normalized weighted Gini coefficient
    g = gini / gini_max

    return 0.5 * (g + d)
```

In [6]:

```
# TEST_FEAT_PATH = '../input/amex-denoised-aggregated-features/test_feat.parquet'
# TRAIN_FEAT_PATH = '../input/amex-denoised-aggregated-features/train_feat.parquet'

def get_data(read_from_cache=True):
    train = pd.read_parquet(TRAIN_FEAT_PATH)
    test = pd.read_parquet(TEST_FEAT_PATH)
    return train, test
```

03. 데이터 불러오기

[목차로 이동하기](#)

In [7]:

```
# LABELS_PATH = '../input/amex-default-prediction/train_labels.csv'
target = pd.read_csv(LABELS_PATH).target.values
train, test = get_data(read_from_cache=True)
print(f"target shape: {target.shape}, train shape: {train.shape}, test shape: {test.
```

```
target shape: (458913,), train shape: (458913, 469), test shape: (9246
21, 469)
```

- [B, D, P, R, S]_X_last
- [B, D, P, R, S]_X_min
- [B, D, P, R, S]_X_max
- [B, D, P, R, S]_X_avg

In [8]:

```
# train columns 확인
features = [f for f in train.columns if f != 'customer_ID' and f != 'target']
features
```

Out[8]:

```
['B_1_last',
 'B_2_last',
 'B_3_last',
 'B_4_last',
 'B_5_last',
 'B_6_last',
 'B_7_last',
 'B_8_last',
 'B_9_last',
 'B_10_last',
 'B_11_last',
 'B_12_last',
 'B_13_last',
 'B_14_last',
 'B_15_last',
 'B_16_last',
 'B_17_last',
 'B_18_last',
```

04. 모델 구축 및 학습

목차로 이동하기

In [9]:

```
param = {
    "objective": "Logloss",
    "learning_rate": 0.01,
    "n_estimators": ITERATIONS,
    #"eval_metric": "AUC", #AmexMetric(),
    #'colsample_bylevel': 0.10506469029379303,
    "max_depth": MAX_DEPTH,
    #"l2_leaf_reg": 15,
    "od_type": "Iter",
    "od_wait": 600,
    #"boosting_type": "Ordered",
    #"bootstrap_type": "MVS",
    "task_type":"GPU",
    #"devices":'0:1',
    #"auto_class_weights": "Balanced",
    #"grow_policy": "Lossguide",
    #"leaf_estimation_method": "Gradient",
    #"leaf_estimation_iterations": 15,
    #"leaf_estimation_backtracking": "Armijo",
    "use_best_model": True,
    #"scale_pos_weight": 20,
    #"score_function": "L2"
}
```

In [10]:

```
cv_folds = 5
ONLY_FIRST_FOLD = False
score_list, y_pred_list = [], []
kf = StratifiedKFold(n_splits=cv_folds)

for fold, (idx_tr, idx_va) in enumerate(kf.split(train, target)):
    X_tr, X_va, y_tr, y_va, model = None, None, None, None, None
    start_time = datetime.datetime.now()  # 시작 시간
    X_tr = train.iloc[idx_tr][features]
    X_va = train.iloc[idx_va][features]
    y_tr = target[idx_tr]
    y_va = target[idx_va]

    with warnings.catch_warnings():
        warnings.filterwarnings('ignore', category=UserWarning)
        model = CatBoostClassifier(**param)
        model.fit(X_tr, y_tr,
                  eval_set = [(X_va, y_va)],
                  metric_period=100
                 )
    X_tr, y_tr = None, None
    y_va_pred = model.predict_proba(X_va)[:,1]
    score = amex_metric(y_va, y_va_pred)
    n_trees = model.best_iteration_
    if n_trees is None: n_trees = model.n_estimators
    print(f"{Fore.GREEN}{Style.BRIGHT}Fold {fold} | {str(datetime.datetime.now() - s
          f" {n_trees:5} trees |"
          f"                Score = {score:.5f}{Style.RESET_ALL}")

    score_list.append(score)
    y_pred_list.append(model.predict_proba(test[features])[:,1])
    gc.collect()
    if ONLY_FIRST_FOLD: break # we only want the first fold

print(f"{Fore.GREEN}{Style.BRIGHT}OOF Score:                       {np.mean(score_li
gc.collect()
```

```
0:	learn: 0.6791702	test: 0.6792409 best: 0.6792409 (0)
total: 65.8ms	remaining: 30m 43s
100:	learn: 0.2710022	test: 0.2752627 best: 0.2752627 (10
0)	total: 6.2s	remaining: 28m 31s
200:	learn: 0.2370489	test: 0.2438037 best: 0.2438037 (20
0)	total: 12.8s	remaining: 29m 27s
300:	learn: 0.2269402	test: 0.2357646 best: 0.2357646 (30
0)	total: 18.4s	remaining: 28m 15s
400:	learn: 0.2214221	test: 0.2321372 best: 0.2321372 (40
0)	total: 24.4s	remaining: 27m 59s
500:	learn: 0.2173831	test: 0.2299194 best: 0.2299194 (50
0)	total: 30s	remaining: 27m 29s
600:	learn: 0.2140748	test: 0.2284084 best: 0.2284084 (60
0)	total: 36s	remaining: 27m 23s
700:	learn: 0.2111725	test: 0.2272325 best: 0.2272325 (70
0)	total: 41.6s	remaining: 27m 1s
800:	learn: 0.2086000	test: 0.2263073 best: 0.2263073 (80
0)	total: 48.2s	remaining: 27m 17s
900:	learn: 0.2062820	test: 0.2255653 best: 0.2255653 (90
```

05. 모델 학습 후, 정보 확인

목차로 이동하기

학습 파라미터 확인

In [11]:

```
model.get_all_params()
```

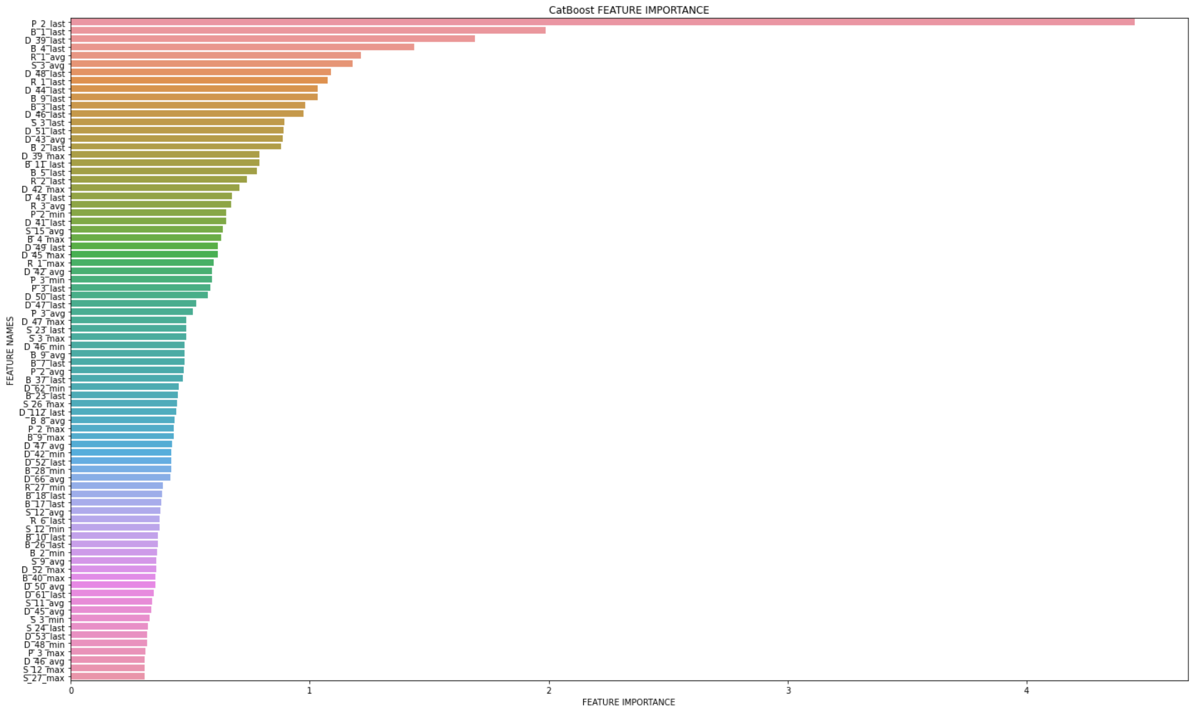
Out[11]:

```
{'nan_mode': 'Min',
 'gpu_ram_part': 0.95,
 'eval_metric': 'Logloss',
 'iterations': 28000,
 'leaf_estimation_method': 'Newton',
 'observations_to_bootstrap': 'TestOnly',
 'od_pval': 0,
 'grow_policy': 'SymmetricTree',
 'boosting_type': 'Plain',
 'feature_border_type': 'GreedyLogSum',
 'bayesian_matrix_reg': 0.10000000149011612,
 'devices': '-1',
 'pinned_memory_bytes': '104857600',
 'force_unit_auto_pair_weights': False,
 'l2_leaf_reg': 3,
 'random_strength': 1,
 'od_type': 'Iter',
 'rsm': 1,
 'boost_from_average': False,
 'gpu_cat_features_storage': 'GpuRam',
 'fold_size_loss_normalization': False,
 'model_size_reg': 0.5,
 'pool_metainfo_options': {'tags': {}},
 'use_best_model': True,
 'meta_l2_frequency': 0,
 'od_wait': 600,
 'class_names': [0, 1],
 'random_seed': 0,
 'depth': 10,
 'border_count': 128,
 'min_fold_size': 100,
 'data_partition': 'DocParallel',
 'bagging_temperature': 1,
 'classes_count': 0,
 'auto_class_weights': 'None',
 'leaf_estimation_backtracking': 'AnyImprovement',
 'best_model_min_trees': 1,
 'min_data_in_leaf': 1,
 'add_ridge_penalty_to_loss_function': False,
 'loss_function': 'Logloss',
 'learning_rate': 0.009999999776482582,
 'meta_l2_exponent': 1,
 'score_function': 'Cosine',
 'task_type': 'GPU',
 'leaf_estimation_iterations': 10,
 'bootstrap_type': 'Bayesian',
 'max_leaves': 1024}
```

특성 중요도 확인

In [12]:

```
top_n = plot_feature_importance(model.feature_importances_, features,'CatBoost', n=8
print(f'Mean CV score for {MAX_DEPTH} max_depth: {np.mean(score_list)}')
```




```
Mean CV score for 10 max_depth: 0.7933820749798006
```

In [13]:

```
# 5개 데이터 셋에 대한 평
print(f'LGBM baseline: {np.mean([0.79605, 0.79674, 0.79362, 0.79283, 0.79442])}')
```

```
LGBM baseline: 0.794732
```

예측값의 확인

In [14]:

```
np.exp((np.log(y_pred_list[0])+np.log(y_pred_list[1])+np.log(y_pred_list[2])+np.log(
```

Out[14]:

```
array([0.01859136, 0.00176784, 0.03999009, ..., 0.3929696 , 0.4208918
,
       0.04277044])
```

In [15]:

```
(y_pred_list[0]+y_pred_list[1]+y_pred_list[2]+y_pred_list[3]+y_pred_list[4])/5
```

Out[15]:

```
array([0.01866999, 0.00177495, 0.04032808, ..., 0.39607875, 0.4218845
1,
       0.04302383])
```

In [16]:

```
np.mean(y_pred_list[0])
```

Out[16]:

```
0.25215407968337267
```

In [17]:

```
list(top_n)
```

Out[17]:

```
['P_2_last',
 'B_1_last',
 'D_39_last',
 'B_4_last',
 'R_1_avg',
 'S_3_avg',
 'D_48_last',
 'R_1_last',
 'D_44_last',
 'B_9_last',
 'B_3_last',
 'D_46_last',
 'S_3_last',
 'D_51_last',
 'D_43_avg',
 'B_2_last',
 'D_39_max',
 'B_11_last',
 'B_5_last',
 'R_2_last',
 'D_42_max',
 'D_43_last',
 'R_3_avg',
 'P_2_min',
 'D_41_last',
 'S_15_avg',
 'B_4_max',
 'D_49_last',
 'D_45_max',
 'R_1_max',
 'D_42_avg',
 'P_3_min',
 'P_3_last',
 'D_50_last',
 'D_47_last',
 'P_3_avg',
 'D_47_max',
 'S_23_last',
 'S_3_max',
 'D_46_min',
 'B_9_avg',
 'B_7_last',
 'P_2_avg',
 'B_37_last',
 'D_62_min',
 'B_23_last',
 'S_26_max',
 'D_112_last',
 'B_8_avg',
 'P_2_max',
 'B_9_max',
 'D_47_avg',
 'D_42_min',
 'D_52_last',
```

```
 'B_28_min',
 'D_66_avg',
 'R_27_min',
 'B_18_last',
 'B_17_last',
 'S_12_avg',
 'R_6_last',
 'S_12_min',
 'B_10_last',
 'B_26_last',
 'B_2_min',
 'S_9_avg',
 'D_52_max',
 'B_40_max',
 'D_50_avg',
 'D_61_last',
 'S_11_avg',
 'D_45_avg',
 'S_3_min',
 'S_24_last',
 'D_53_last',
 'D_48_min',
 'P_3_max',
 'D_46_avg',
 'S_12_max',
 'S_27_max']
```

06. 제출

[목차로 이동하기](#)

In [18]:

```
sub = pd.DataFrame({'customer_ID': test.index,
                    'prediction':  np.exp((np.log(y_pred_list[0])+np.log(y_pred_list
                                   np.log(y_pred_list[3])+np.log(y_pred_list[4]))/5)
sub.to_csv('./submission.csv', index=False)
```

In [19]:

```
sub
```

Out[19]:

	customer_ID	prediction
0	00000469ba478561f23a92a868bd366de6f6527a684c9a...	0.018591
1	00001bf2e77ff879fab36aa4fac689b9ba411dae63ae39...	0.001768
2	0000210045da4f81e5f122c6bde5c2a617d03eef67f82c...	0.039990
3	00003b41e58ede33b8daf61ab56d9952f17c9ad1c3976c...	0.221594
4	00004b22eaeeeb0ec976890c1d9bfc14fd9427e98c4ee9...	0.828064
...	...	...
924616	ffff952c631f2c911b8a2a8ca56ea6e656309a83d2f64c...	0.010519
924617	ffffcf5df59e5e0bba2a5ac4578a34e2b5aa64a1546cd3...	0.799683
924618	ffffd61f098cc056dbd7d2a21380c4804bbfe60856f475...	0.392970
924619	ffffddef1fc3643ea179c93245b68dca0f36941cd83977...	0.420892
924620	fffffa7cf7e453e1acc6a1426475d5cb9400859f82ff61...	0.042770

924621 rows × 2 columns