[컴페티션] 머신러닝 성능 극대화

- 대회 주제 : Binary Classification with a Bank Churn Dataset
- 대회 링크 : https://www.kaggle.com/competitions/playground-series-s4e1
- 평가 항목 : ROC Curve
- 제출일 : 2024년 12월 10일 화요일 7교시
- 평가 항목 점수 : 0.90573/0.90249
- 수강생 성함 : 송현서

```
import numpy as np
import pandas as pd
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from catboost import CatBoostClassifier, Pool
from catboost.utils import eval_metric

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

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

```
# 성능을 올리는 가장 쉽고 빠른 방법 --> 학습을 많이 시킨다.

RAND_VAL=27
num_folds=6 ## Number of folds
n_est=8000 ## Number of estimators
```

```
df_train = pd.read_csv('/kaggle/input/playground-series-s4e1/train.csv')
df_train.head()
```

```
df_test = pd.read_csv('/kaggle/input/playground-series-s4e1/test.csv')
df_test_ov = df_test.copy()
df_test.head()
```

```
df_orig=pd.read_csv("/kaggle/input/bank-customer-churn-prediction/Churn_Modelling.csv")
df_orig=df_orig.rename(columns={'RowNumber':'id'})
df_orig.head()
```

```
scale_cols = ['Age','CreditScore', 'Balance','EstimatedSalary']
###
for c in scale_cols:
    min_value = df_train[c].min()
    max_value = df_train[c].max()
    df_train[c+"_scaled"] = (df_train[c] - min_value) / (max_value - min_value)
    df_test[c+"_scaled"] = (df_test[c] - min_value) / (max_value - min_value)
```

```
df_all = pd.concat([df_train,df_test]).reset_index(drop=True)
aggs = {
    'Age': ['min','max', 'mean'],
    'Balance': ['min','max', 'mean','sum'],
    'NumOfProducts': ['mean','sum'],
    'IsActiveMember': ['min','max', 'mean','sum'],
    'CreditScore': ['min','max', 'mean'],
    'EstimatedSalary': ['min','max', 'mean','sum'],
    'id': 'count',
}
df_grps=df_all.groupby(['CustomerId', 'Surname', 'Geography', 'Gender']).agg(aggs).reset_index()
df_grps.columns = list(map('_'.join, df_grps.columns.values))
print(len(df_grps))
df_grps.head()
```

```
aggs = {
    "Balance": ['min','max', 'mean','sum'],
    'NumOfProducts': ['mean','sum'],
    'IsActiveMember': ['min','max', 'mean','sum'],
    'CreditScore': ['min','max', 'mean'],
    'EstimatedSalary': ['min','max', 'mean','sum'],
    'id': 'count',
}
df_grps1=df_all.groupby(['CustomerId', 'Surname', 'Age', 'Gender']).agg(aggs).reset_index()
df_grps1.columns = list(map('grps1'.join, df_grps1.columns.values))
print(len(df_grps1))
df_grps1=df_grps1.rename(columns={'CustomerIdgrps1':'CustomerId','Surnamegrps1':'Surname',
                                  'Agegrps1':'Age','Gendergrps1':'Gender'})

df_grps1.head()
```

```
exitGrpBy=['CustomerId', 'Surname',  'Gender','Geography','EstimatedSalary']

exitSrtBy=['CustomerId', 'Surname',  'Gender','Geography',
           'Age', 'Tenure']
##
df_all_Exits=df_all.copy()
df_all_Exits['Exited']=df_all_Exits['Exited'].fillna(-1)
df_all_Exits=df_all_Exits.sort_values(exitSrtBy)
df_all_Exits['Exit_lag1']=df_all_Exits.groupby(exitGrpBy)['Exited'].shift(1)
df_all_Exits['Exit_lag2']=df_all_Exits.groupby(exitGrpBy)['Exited'].shift(2)
df_all_Exits['Exit_lag3']=df_all_Exits.groupby(exitGrpBy)['Exited'].shift(3)

df_all_Exits['Exit_lead1']=df_all_Exits.groupby(exitGrpBy)['Exited'].shift(-1)
df_all_Exits['Exit_lead2']=df_all_Exits.groupby(exitGrpBy)['Exited'].shift(-2)
df_all_Exits['Exit_lead3']=df_all_Exits.groupby(exitGrpBy)['Exited'].shift(-3)

df_all_Exits['Balance_lag_diff1']=df_all_Exits['Balance'].shift(1)
df_all_Exits['Balance_lead_diff1']=df_all_Exits['Balance'].shift(-1)

df_all_Exits=df_all_Exits[['id','Exit_lag1','Exit_lag2','Exit_lag3',
                           'Exit_lead1','Exit_lead2','Exit_lead3',
                           'Balance_lag_diff1','Balance_lead_diff1']]
df_all_Exits=df_all_Exits.fillna(-1).astype('int')
df_all_Exits
```

```
def getGrpsIndv(df_orig,df_train,df_test,grpCols,nm):
    grpBy=[]
    for c in grpCols:
        for i in grpCols:
            if c!=i:
                grpBy=[c,i]
                grpBy=[c,i]
                df_tmp=df_orig.groupby(grpBy).agg({'id':'count','Exited':'mean'}).reset_index()
                df_tmp.columns=list(map(''.join, list(df_tmp.columns.values)))
                sepCols=df_tmp.columns.drop(grpBy)+nm+'_ind_'+c+'-'+i
                df_tmp.columns=list(grpBy)+list(sepCols)
                #
                df_train=df_train.merge(df_tmp,how='left')
                df_test=df_test.merge(df_tmp,how='left')

                df_train[sepCols]=df_train[sepCols].fillna(0)
                df_test[sepCols]=df_test[sepCols].fillna(0)

                df_train[sepCols]=df_train[sepCols].astype('int')
                df_test[sepCols]=df_test[sepCols].astype('int')
    return df_train,df_test
```

```
def getGrps(df_orig,df_train,df_test,grpCols,nm):
    grpBy=[]
    for c in grpCols:
        grpBy.append(c)
        df_tmp=df_orig.groupby(grpBy).agg({'id':'count','Exited':['sum']}).reset_index()
        df_tmp.columns=list(map(''.join, list(df_tmp.columns.values)))
        sepCols=df_tmp.columns.drop(grpBy)+nm+'_grps_'+c
        df_tmp.columns=list(grpBy)+list(sepCols)
        #
        df_train=df_train.merge(df_tmp,how='left')
        df_test=df_test.merge(df_tmp,how='left')

        df_train[sepCols]=df_train[sepCols].fillna(0)
        df_test[sepCols]=df_test[sepCols].fillna(0)

        df_train[sepCols]=df_train[sepCols].astype('int')
        df_test[sepCols]=df_test[sepCols].astype('int')
    return df_train,df_test
```

```
grpCols=['CustomerId', 'Surname', 'Geography', 'Gender', 'Age', 'Tenure', 'CreditScore',
         'NumOfProducts','HasCrCard',
         'IsActiveMember','EstimatedSalary','Balance']
df_train,df_test=getGrps(df_orig,df_train,df_test,grpCols,'Orig_groups')
df_train,df_test=getGrpsIndv(df_orig,df_train,df_test,grpCols,'Orig_ind')
```

```
df_train.shape
```

```
df_train.head()
```

Feature Engineering

```
def getFeats(df):

    df['IsSenior'] = df['Age'].apply(lambda x: 1 if x >= 60 else 0)
    df['IsActive_by_CreditCard'] = df['HasCrCard'] * df['IsActiveMember']
    df['Products_Per_Tenure'] = df['Tenure'] / df['NumOfProducts']
    df['AgeCat'] = np.round(df.Age/20).astype('int').astype('category')
    df['Sur_Geo_Gend_Sal'] = df['Surname']+df['Geography']+df['Gender']+np.round(df.EstimatedSalary).astype('str')

    df = df.merge(df_grps,how='left',on=['CustomerId', 'Surname', 'Geography', 'Gender'])
    df = df.merge(df_grps1,how='left',on=['CustomerId', 'Surname', 'Age', 'Gender'])
    df = df.merge(df_all_Exits,how='left')
    return df
```

```
df_train = getFeats(df_train)
df_test = getFeats(df_test)
##
feat_cols=df_train.columns.drop(['id','Exited'])
feat_cols=feat_cols.drop(scale_cols)
print("Number of Features:",len(feat_cols))
print(feat_cols)
df_train.head()
```

```
X=df_train[feat_cols]
y=df_train['Exited']
##
cat_features = np.where(X.dtypes != np.float64)[0]
cat_features
```

Training

```
'''
folds = StratifiedKFold(n_splits=num_folds,random_state=RAND_VAL,shuffle=True)
test_preds = np.empty((num_folds, len(df_test)))
auc_vals=[]

for n_fold, (train_idx, valid_idx) in enumerate(folds.split(X, y)):

    X_train, y_train = X.iloc[train_idx], y.iloc[train_idx]
    X_val, y_val = X.iloc[valid_idx], y.iloc[valid_idx]

    train_pool = Pool(X_train, y_train,cat_features=cat_features)
    val_pool = Pool(X_val, y_val,cat_features=cat_features)

    clf = CatBoostClassifier(
    eval_metric='AUC',
    task_type='GPU',
    learning_rate=0.02,
    iterations=n_est)
    clf.fit(train_pool, eval_set=val_pool,verbose=300)

    y_pred_val = clf.predict_proba(X_val[feat_cols])[:,1]
    auc_val = roc_auc_score(y_val, y_pred_val)
    print("AUC for fold "+n_fold+": "+auc_val)
    auc_vals.append(auc_val)

    y_pred_test = clf.predict_proba(df_test[feat_cols])[:,1]
    test_preds[n_fold, :] = y_pred_test
    print("---------------")
    '''
```

```
from skopt import BayesSearchCV

param_space = {
    'learning_rate': (0.005, 0.1, 'log-uniform'),  # 범위
    'depth': (4, 12),                              # 트리 깊이 범위 확장
    'l2_leaf_reg': (1, 15),                        # 정규화 범위 확장
    'bagging_temperature': (0, 1.0),               # 샘플링 온도 추가
    'border_count': (32, 255),                     # 분할 기준 개수 추가
    'iterations': (500, 3000)                      # Iterations 범위 조정
}

folds = StratifiedKFold(n_splits=num_folds, random_state=42, shuffle=True)  # fold 수 그로 고정
test_preds = np.empty((folds.n_splits, len(df_test)))
auc_vals = []

for n_fold, (train_idx, valid_idx) in enumerate(folds.split(X, y)):
    X_train, y_train = X.iloc[train_idx], y.iloc[train_idx]
    X_val, y_val = X.iloc[valid_idx], y.iloc[valid_idx]

    train_pool = Pool(X_train, y_train, cat_features=cat_features)
    val_pool = Pool(X_val, y_val, cat_features=cat_features)

    clf = CatBoostClassifier(task_type='GPU', eval_metric='AUC', verbose=0)  # 기본 모델

    # Bayesian Optimization
    bayes_search = BayesSearchCV(
        estimator=clf,
        search_spaces=param_space,
        scoring='roc_auc',
        cv=3,
        n_iter=30,  # 탐색 횟수
        verbose=0,
        random_state=42
    )
    # 하이퍼파라미터 최적화 수행
    bayes_search.fit(X_train, y_train, cat_features=cat_features)

    # 최적 파라미터와 성능 출력
    best_params = bayes_search.best_params_
    print(f"Fold {n_fold + 1} - Best Parameters: {best_params}")

    # 최적 파라미터로 학습
    clf = CatBoostClassifier(
        **best_params, task_type='GPU', eval_metric='AUC', verbose=300
    )
    clf.fit(train_pool, eval_set=val_pool)

    # 성능 평가
    y_pred_val = clf.predict_proba(X_val[feat_cols])[:, 1]
    auc_val = roc_auc_score(y_val, y_pred_val)
    print(f"AUC for fold {n_fold + 1}: {auc_val}")
    auc_vals.append(auc_val)

    # 테스트 데이터 예측
    y_pred_test = clf.predict_proba(df_test[feat_cols])[:, 1]
    test_preds[n_fold, :] = y_pred_test

    # 현재까지 진행 상황 요약
    print(f"Completed fold {n_fold + 1}/{folds.n_splits}")
    print("---------------")

# 전체 결과 출력
print(f"Mean AUC across folds: {np.mean(auc_vals):.4f}")
```

Evaluation

```
"Mean AUC: ",np.mean(auc_vals)
```

Feature Importance

```
import shap
shap.initjs()
explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(train_pool)
shap.summary_plot(shap_values, X_train, plot_type="bar")
```

Prediction and Submission

```
join_cols=list(df_orig.columns.drop(['Exited']))
df_orig.rename(columns={'Exited':'Exited_Orig'},inplace=True)
df_orig['Exited_Orig']=df_orig['Exited_Orig'].map({0:1,1:0})
df_test_ov=df_test_ov.merge(df_orig,on=join_cols,how='left')[['id','Exited_Orig']].fillna(-1)
df_test_ov.head()
```

```
y_pred = test_preds.mean(axis=0)
df_sub = df_test_ov[['id','Exited_Orig']]
df_sub['Exited'] = np.where(df_sub.Exited_Orig==-1,y_pred,df_sub.Exited_Orig)
df_sub.drop('Exited_Orig',axis=1,inplace=True)
df_sub.head()
```

In [ ]:
```
df_sub.to_csv("submission.csv",index=False)
```

In [ ]:
```
df_sub.hist(column='Exited', bins=20, range=[0,1],figsize=(12,6))
plt.show()
```