산탄데르 고객 만족 예측 - 분류

학습 내용

- 캐글의 산탄데르 고객 만족 데이터 세트에 대해 고객 만족 여부를 XGBoost와 LightGBM을 활용하여 예측

데이터 설명

- 데이터 다운로드 : [https://www.kaggle.com/c/santander-customer-satisfaction/data (https://www.kaggle.com/c/santander-customer-satisfaction/data)](https://www.kaggle.com/c/santander-customer-satisfaction/data)
- 370개의 피처로 이루어진 데이터
- 피처 이름은 전부 익명처리되어 있음.
- 클래스 레이블 명은 TARGET
  - 값이 1이면 불만을 가지고 있음.
  - 값이 0이면 만족한 고객

평가 지표

- 성능 평가는 ROC-AUC(ROC 곡선 영역)으로 평가

데이터 로드 및 전처리

In [1]:

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

In [26]:

```
train = pd.read_csv("../../data/santander_customer/train.csv", encoding='latin-1')
test = pd.read_csv("../../data/santander_customer/test.csv", encoding='latin-1')
sub = pd.read_csv("../../data/santander_customer/sample_submission.csv")

train.shape, test.shape, sub.shape
```

Out[26]:

```
((76020, 371), (75818, 370), (75818, 2))
```

In [27]:

```
train.head()
```

Out[27]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	im
0	1	2	23	0.0	0.0	0.0	
1	3	2	34	0.0	0.0	0.0	
2	4	2	23	0.0	0.0	0.0	
3	8	2	37	0.0	195.0	195.0	
4	10	2	39	0.0	0.0	0.0	

5 rows × 371 columns

In [28]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76020 entries, 0 to 76019
Columns: 371 entries, ID to TARGET
dtypes: float64(111), int64(260)
memory usage: 215.2 MB
```

- 111개의 피처가 float형,
- 260개의 피처가 int형
- 모든 피처가 숫자형이며
- NUll값은 없다.

In [29]:

```
train.columns
```

Out[29]:

```
Index(['ID', 'var3', 'var15', 'imp_ent_var16_ult1', 'imp_op_var39_comer_ult1',
       'imp_op_var39_comer_ult3', 'imp_op_var40_comer_ult1',
       'imp_op_var40_comer_ult3', 'imp_op_var40_efect_ult1',
       'imp_op_var40_efect_ult3',
       ...
       'saldo_medio_var33_hace2', 'saldo_medio_var33_hace3',
       'saldo_medio_var33_ult1', 'saldo_medio_var33_ult3',
       'saldo_medio_var44_hace2', 'saldo_medio_var44_hace3',
       'saldo_medio_var44_ult1', 'saldo_medio_var44_ult3', 'var38', 'TARGET'],
      dtype='object', length=371)
```

전체 데이터의 만족(0), 불만족(1) 비율

In [30]:

```
train['TARGET'].value_counts()
```

Out[30]:

```
0    73012
1     3008
Name: TARGET, dtype: int64
```

In [31]:

```
unsatified = train['TARGET'].value_counts()[1]
unsatified / train['TARGET'].count()  # 비율
```

Out[31]:

```
0.03956853459615889
```

In [32]:

```
train.describe()
```

Out[32]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_u
count	76020.000000	76020.000000	76020.000000	76020.000000	76020.0000
mean	75964.050723	-1523.199277	33.212865	86.208265	72.3630
std	43781.947379	39033.462364	12.956486	1614.757313	339.3158
min	1.000000	-999999.000000	5.000000	0.000000	0.0000
25%	38104.750000	2.000000	23.000000	0.000000	0.0000
50%	76043.000000	2.000000	28.000000	0.000000	0.0000
75%	113748.750000	2.000000	40.000000	0.000000	0.0000
max	151838.000000	238.000000	105.000000	210000.000000	12888.0300

8 rows × 371 columns

- var3의 최소값이 -999999 - 이상치로 보임

In [33]:

```
train['var3'].value_counts()
```

Out[33]:

```
 2         74165
 8           138
-999999      116
 9           110
 3           108
           ...
 218           1
 215           1
 151           1
 87            1
 191           1
Name: var3, Length: 208, dtype: int64
```

In [34]:

```
# -999999를 가장 많은 값으로 변경
train['var3'].replace(-999999, 2, inplace=True)
```

In [35]:

```
## ID 열을 삭제
# train.drop('ID', axis=1, inplace=True)
train = train.loc[  :, "var3":  ]
train.head()
```

Out[35]:

	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_o
0	2	23	0.0	0.0	0.0	
1	2	34	0.0	0.0	0.0	
2	2	23	0.0	0.0	0.0	
3	2	37	0.0	195.0	195.0	
4	2	39	0.0	0.0	0.0	

5 rows × 370 columns

In [37]:

```
# 피처와 레이블를 지정.
X = train.iloc[:, :-1]
y = train['TARGET']

X.shape, y.shape
```

Out[37]:

```
((76020, 369), (76020,))
```

In [38]:

```
from sklearn.model_selection import train_test_split

X_train , X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size=0.2, random_state=0)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[38]:

```
((60816, 369), (15204, 369), (60816,), (15204,))
```

In [43]:

```
## 레이블 분포비율
print( "학습용 레이블 분포 비율 : ₩n" , y_train.value_counts() / y_train.count() )
print( "테스트용 레이블 분포 비율 : ₩n" , y_train.value_counts() / y_train.count() )
```

```
학습용 레이블 분포 비율 :
 0    0.960964
1    0.039036
Name: TARGET, dtype: float64
테스트용 레이블 분포 비율 :
 0    0.960964
1    0.039036
Name: TARGET, dtype: float64
```

모델 생성 및 학습, 그리고 평가해 보기

In [52]:

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score

xgb_model = XGBClassifier(n_estimators=500, random_state=156)
xgb_model.fit(X_train, y_train, early_stopping_rounds=100,
              eval_metric='auc', eval_set=[(X_train, y_train), (X_test, y_test)])
```

```
[31]	validation_0-auc:0.89916	validation_1-auc:0.83952
[32]	validation_0-auc:0.90106	validation_1-auc:0.83901
[33]	validation_0-auc:0.90253	validation_1-auc:0.83885
[34]	validation_0-auc:0.90278	validation_1-auc:0.83887
[35]	validation_0-auc:0.90293	validation_1-auc:0.83864
[36]	validation_0-auc:0.90463	validation_1-auc:0.83834
[37]	validation_0-auc:0.90500	validation_1-auc:0.83810
[38]	validation_0-auc:0.90519	validation_1-auc:0.83810
[39]	validation_0-auc:0.90533	validation_1-auc:0.83813
[40]	validation_0-auc:0.90575	validation_1-auc:0.83776
[41]	validation_0-auc:0.90691	validation_1-auc:0.83720
[42]	validation_0-auc:0.90716	validation_1-auc:0.83684
[43]	validation_0-auc:0.90737	validation_1-auc:0.83672
[44]	validation_0-auc:0.90759	validation_1-auc:0.83674
[45]	validation_0-auc:0.90769	validation_1-auc:0.83693
[46]	validation_0-auc:0.90779	validation_1-auc:0.83686
[47]	validation_0-auc:0.90793	validation_1-auc:0.83678
[48]	validation_0-auc:0.90831	validation_1-auc:0.83694
[49]	validation_0-auc:0.90871	validation_1-auc:0.83676
```

In [53]:

```
pred_prob = xgb_model.predict_proba(X_test)[:, 1]
pred_prob
```

Out[53]:

```
array([0.00643863, 0.02387667, 0.01260844, ..., 0.05883254, 0.01729385,
       0.01727541], dtype=float32)
```

In [54]:

```
xgb_roc_score = roc_auc_score(y_test, pred_prob, average='macro')
print("ROC AUC : {0:.4f}".format(xgb_roc_score))
```

```
ROC AUC : 0.8413
```

하이퍼 파라미터 튜닝

- max_depth, min_child_weight, colsample_bytree
- 먼저 2-3개 정도의 파라미터를 최적화 시킨 후,최적 파라미터를 기반으로 1-2개 파라미터를 결합하여 튜닝을 수행

In [55]:

```
%%time

from sklearn.model_selection import GridSearchCV

# 우선 하이퍼 파라미터 수행 속도를 향상을 위해 100으로
xgb_model1 = XGBClassifier(n_estimators=100, use_label_encoder=False)
params = {"max_depth":[5,7],
          "min_child_weight":[1,3],
          "colsample_bytree":[0.5, 0.75]}

gridcv = GridSearchCV(xgb_model1, param_grid=params, cv=3)
gridcv.fit(X_train, y_train, early_stopping_rounds=30,
           eval_metric='auc',
           eval_set = [(X_train, y_train), (X_test, y_test)])
```

```
[40]	validation_0-auc:0.87678	validation_1-auc:0.83859
[41]	validation_0-auc:0.87711	validation_1-auc:0.83830
[42]	validation_0-auc:0.87738	validation_1-auc:0.83823
[43]	validation_0-auc:0.87752	validation_1-auc:0.83796
[44]	validation_0-auc:0.87777	validation_1-auc:0.83765
[45]	validation_0-auc:0.87785	validation_1-auc:0.83786
[46]	validation_0-auc:0.87802	validation_1-auc:0.83761
[47]	validation_0-auc:0.87840	validation_1-auc:0.83698
[48]	validation_0-auc:0.87868	validation_1-auc:0.83699
[49]	validation_0-auc:0.87882	validation_1-auc:0.83708
[0]	validation_0-auc:0.80039	validation_1-auc:0.80013
[1]	validation_0-auc:0.82111	validation_1-auc:0.82026
[2]	validation_0-auc:0.82749	validation_1-auc:0.82627
[3]	validation_0-auc:0.83124	validation_1-auc:0.82830
[4]	validation_0-auc:0.83475	validation_1-auc:0.82881
[5]	validation_0-auc:0.83676	validation_1-auc:0.83385
[6]	validation_0-auc:0.83648	validation_1-auc:0.83085
[7]	validation_0-auc:0.84336	validation_1-auc:0.83472
[8]	validation_0-auc:0.84624	validation_1-auc:0.83404
[9]	validation_0-auc:0.84541	validation_1-auc:0.83287
```

In [64]:

```
print("GridSearchCV 최적 파라미터 : ", gridcv.best_params_ )

pred_prob = gridcv.predict_proba(X_test)[:, 1]
xgb_roc_score = roc_auc_score(y_test, pred_prob, average='macro')
print("ROC AUC : {0:4f}".format(xgb_roc_score))
```

```
GridSearchCV 최적 파라미터 : {'colsample_bytree': 0.5, 'max_depth': 5, 'min_child_
weight': 3}
ROC AUC : 0.844455
```

실습해 보기

- colsample_bytree : 0.5, max_depth : 5, min_child_weight : 3로 설정
- n_estimators = 1000으로 증가, learning_rate를 조정해보고, reg_alpha를 추가하여 ROC_AUC의 값을 구해보자.

In [59]:

```
%%time

xgb_model_l = XGBClassifier(n_estimators=1000,
                            random_state= 77,
                            learning_rate=0.02,
                            max_depth=5,
                            min_child_weight=3,
                            colsample_bytree=0.5,
                            reg_alpha=0.03)

# 성능 평가 지표를 auc로, 조기 중단 파라미터 값은 200으로 설정하고 학습 수행
xgb_model_l.fit(X_train, y_train, early_stopping_rounds=200,
              eval_metric='auc', eval_set=[(X_train, y_train), (X_test, y_test)])
```

```
[525]	validation_0-auc:0.88177	validation_1-auc:0.84484
[526]	validation_0-auc:0.88183	validation_1-auc:0.84480
[527]	validation_0-auc:0.88188	validation_1-auc:0.84481
[528]	validation_0-auc:0.88192	validation_1-auc:0.84479
[529]	validation_0-auc:0.88197	validation_1-auc:0.84479
[530]	validation_0-auc:0.88200	validation_1-auc:0.84481
[531]	validation_0-auc:0.88205	validation_1-auc:0.84484
[532]	validation_0-auc:0.88213	validation_1-auc:0.84478
Wall time: 1min 48s
```

Out[59]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.02, max_delta_step=0, max_depth=5,
              min_child_weight=3, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=8, num_parallel_tree=1, random_state=77,
              reg_alpha=0.03, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [60]:

```
pred_prob = xgb_model_l.predict_proba(X_test)[:, 1]
xgb_roc_score = roc_auc_score(y_test, pred_prob, average='macro')
print("ROC AUC : {0:4f}".format(xgb_roc_score))
```

```
ROC AUC : 0.845269
```

메모

- XGBoost는 GBM을 기반으로 하고 있기에, 수행시간이 어느정도 걸립니다.
- 앙상블 계열 알고리즘에서 하이퍼 파라미터 튜닝으로 성능 수치 개선이 급격하게 되는 경우는 많지 않습니다.

각 특징의 중요도 시각화

- xgboost 모듈의 plot_importance() 메서드를 이용

In [63]:

```
from xgboost import plot_importance
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1, figsize=(10,8))
plot_importance(xgb_model_l, ax=ax, max_num_features=20, height=0.4)
```

Out[63]:

```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Featur
es'>
```