잡케어 추천 알고리즘 경진대회

김도연, 김혜원, 장서연, 정세은

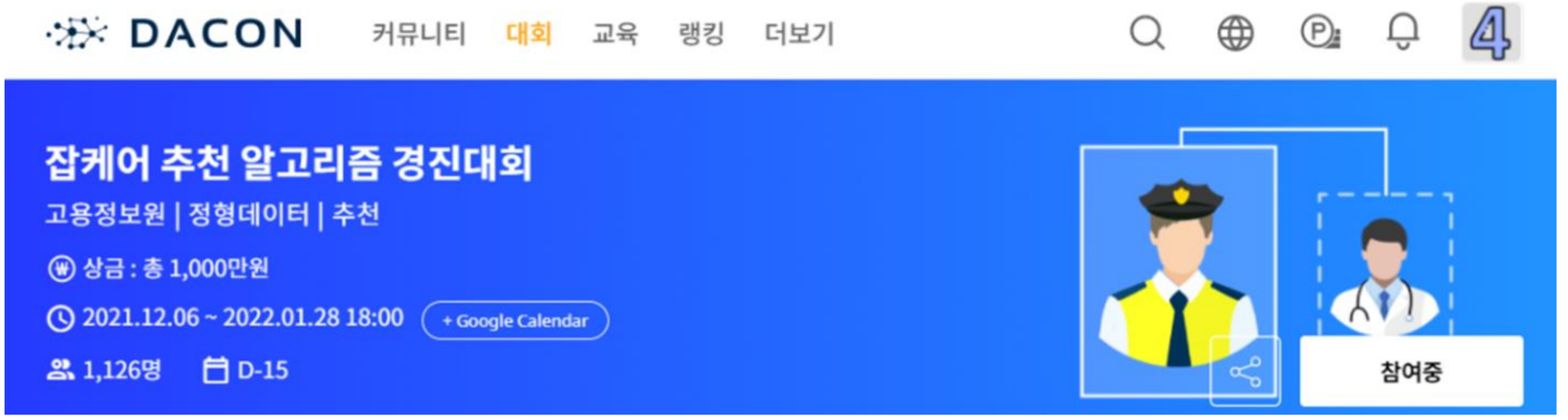
CONTENTS

01

대회 소개

잡케어 추천 알고리즘 경진대회

잡케어 추천 알고리즘 경진대회



● 목적: 잡케어 서비스에 적용 가능한 추천 알고리즘 개발.

● 성능 지표: F1 - score

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

• 리더보드

a. 평가 산식 : F1-score

- precision(정밀도): 모델이 True라고 분류한 것 중에서 실제 True인 것의 비율
- recall(재현율): 실제 True인 것 중에서 모델이 True라고 예측한 것의 비율
- F1: precision과 recall의 조화평균

=〉 F1 값은 0 ~ 1사이이며, 클 수록 예측 성능이 좋음을 의미.

잡케어 추천 알고리즘 경진대회

● 변수 설명

id: 회원의 아이디	person_prefer_c : 회원 선호 속성 C	contents_attribute_i : 컨텐츠 속성 I	contents_attribute_e : 컨텐츠 속성 E
d_l_match_yn : 속성 D 대분류 매칭 여부	person_prefer_d_1 : 회원 선호 속성 D 1번	contents_attribute_a : 컨텐츠 속성 A	contents_attribute_h : 컨텐츠 속성 H
d_m_match_yn : 속성 D 세분류 매칭 여부	person_prefer_d_2 : 회원 선호 속성 D 2번	contents_attribute_j_1 : 컨텐츠 속성 J 하위 속성 1	person_rn : 사용자번호
d_s_match_yn : 속성 D 코드 매칭 여부	person_prefer_d_3 : 회원 선호 속성 D 3번	contents_attribute_j : 컨텐츠 속성 J	contents_rn : 컨텐츠번호
h_l_match_yn : 속성 H 대분류 매칭 여부	person_prefer_e : 회원 선호 속성 E	contents_attribute_c : 컨텐츠 속성 C	contents_open_dt : 컨텐츠 열람 일시
h_m_match_yn : 속성 H 중분류 매칭 여부	person_prefer_f : 회원 선호 속성 F	contents_attribute_k : 컨텐츠 속성 K	target : 컨텐츠 사용 여부 (라벨)
h_s_match_yn : 속성 H 코드 매칭 여부	person_prefer_g : 회원 선호 속성 G	contents_attribute_l : 컨텐츠 속성 L	
person_attribute_a : 회원 속성 A	person_prefer_h_1 : 회원 선호 속성 H 1번	contents_attribute_d : 컨텐츠 속성 D	
person_attribute_a_1 : 회원 속성 A 하위 속성 1	person_prefer_h_2 : 회원 선호 속성 H 2번	contents_attribute_m : 컨텐츠 속성 M	
person_attribute_b : 회원 속성 B	person_prefer_h_3 : 회원 선호 속성 H 3번		

개인 속성 및 선호 속성 관련 변수

컨텐츠 속성 및 정보 관련 변수

02

전처리

잡케어 추천 알고리즘 경진대회

데이터 전처리

● 데이터 불러오기

: train, test, 속성_D_코드, 속성_H_코드, 속성_L_코드

```
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")

d_code = pd.read_csv('속성_D_코드.csv', index_col=0).T.to_dict()
h_code = pd.read_csv('속성_H_코드.csv', index_col=0).T.to_dict()
l_code = pd.read_csv('속성_L_코드.csv', index_col=0).T.to_dict()

print("train_data.shape: ", train_data.shape)
print("test_data.shape: ", test_data.shape)
```

```
train_data.shape:  (501951, 35)
test_data.shape:  (46404, 34)
```

데이터 전처리

● d_code, h_code, l_code 풀어서 변수 추가

```
def add_code(df, d_code, h_code, l_code):
    df = df.copy()

    # D Code
    df['person_prefer_d_1_n'] = df['person_prefer_d_1'].apply(lambda x: d_code[x]['속성 D 세분류코드'])
    df['person_prefer_d_1_s'] = df['person_prefer_d_1'].apply(lambda x: d_code[x]['속성 D 소분류코드'])
    df['person_prefer_d_1_m'] = df['person_prefer_d_1'].apply(lambda x: d_code[x]['속성 D 중분류코드'])
    df['person_prefer_d_1_l'] = df['person_prefer_d_1'].apply(lambda x: d_code[x]['속성 D 대분류코드'])
                                                    .
                                                    .
    # H Code
    df['person_prefer_h_1_l'] = df['person_prefer_h_1'].apply(lambda x: h_code[x]['속성 H 대분류코드'])
    df['person_prefer_h_1_m'] = df['person_prefer_h_1'].apply(lambda x: h_code[x]['속성 H 중분류코드'])
                                                    .
                                                    .
    # L Code
    df['contents_attribute_l_s'] = df['contents_attribute_l'].apply(lambda x: l_code[x]['속성 L 소분류코드'])
    df['contents_attribute_l_m'] = df['contents_attribute_l'].apply(lambda x: l_code[x]['속성 L 중분류코드'])
    df['contents_attribute_l_l'] = df['contents_attribute_l'].apply(lambda x: l_code[x]['속성 L 대분류코드'])
    return df
```

데이터 전처리

● 날짜 변환

: YYYY-MM-DD 형태를 년, 월, 일로 각각 분할

```
def preprocessing_contents_open_dt(data):
    data['contents_open_dt'] = data['contents_open_dt'].astype('str')
    DATE = data['contents_open_dt'].apply(lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))

    DATE = pd.DataFrame(DATE)
    DATE = DATE.rename(columns = {'contents_open_dt': 'date'})

    DATE['year'] = DATE['date'].apply(lambda x: x.timetuple()[0])
    DATE['month'] = DATE['date'].apply(lambda x: x.timetuple()[1])
    DATE['day'] = DATE['date'].apply(lambda x: x.timetuple()[2])
    DATE['id'] = data['id']

    data = data.merge(DATE, on = 'id', how = 'left')
    data = data.drop(columns = ['date', 'contents_open_dt'])
    return data

train_data = preprocessing_contents_open_dt(train_data)
test_data = preprocessing_contents_open_dt(test_data)
```

데이터 전처리

● 변수 drop

1. ID

```
train_data.drop('id', axis=1, inplace=True)
test_data.drop('id', axis=1, inplace=True)
```

2. unique value가 1개인 변수

```
train_data.drop(['person_prefer_f', 'person_prefer_g', 'year'], axis = 1, inplace = True)
test_data.drop(['person_prefer_f', 'person_prefer_g', 'year'], axis = 1, inplace = True)
```

3. unique value가 너무 많은 변수

```
train_data.drop(['person_rn', 'contents_rn'], axis = 1, inplace = True)
test_data.drop(['person_rn', 'contents_rn'], axis = 1, inplace = True)
```

● boolean형을 int형으로 변환

```
cols = x_train2.select_dtypes(bool).columns.tolist()
x_train2[cols] = x_train2[cols].astype(int)
```

```
cols = x_test2.select_dtypes(bool).columns.tolist()
x_test2[cols] = x_test2[cols].astype(int)
```

03

모델링

잡케어 추천 알고리즘 경진대회

Pycaret을 이용한 모델 선택

```
[60] top3 = compare_models(n_select = 3, sort = 'F1')
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.6253	0.6757	0.6722	0.6144	0.6420	0.2507	0.2518	160.216
xgboost	Extreme Gradient Boosting	0.6221	0.6708	0.6708	0.6111	0.6396	0.2443	0.2455	182.522
lightgbm	Light Gradient Boosting Machine	0.6149	0.6614	0.6699	0.6034	0.6349	0.2299	0.2313	14.056
rf	Random Forest Classifier	0.6199	0.6683	0.6330	0.6166	0.6247	0.2397	0.2398	241.633
gbc	Gradient Boosting Classifier	0.6006	0.6434	0.6377	0.5935	0.6148	0.2013	0.2019	294.476
et	Extra Trees Classifier	0.6138	0.6596	0.6152	0.6132	0.6142	0.2275	0.2275	200.122
ada	Ada Boost Classifier	0.5924	0.6317	0.6182	0.5877	0.6025	0.1848	0.1851	69.339
ridge	Ridge Classifier	0.5846	0.0000	0.6063	0.5809	0.5933	0.1693	0.1694	2.149
lda	Linear Discriminant Analysis	0.5846	0.6186	0.6063	0.5809	0.5933	0.1693	0.1694	6.242
knn	K Neighbors Classifier	0.5614	0.5817	0.5790	0.5590	0.5688	0.1227	0.1228	74.038
lr	Logistic Regression	0.5662	0.5932	0.5709	0.5653	0.5681	0.1324	0.1324	96.391
nb	Naive Bayes	0.5686	0.5893	0.5468	0.5714	0.5589	0.1371	0.1372	0.595
dt	Decision Tree Classifier	0.5488	0.5488	0.5505	0.5484	0.5495	0.0977	0.0977	15.058
qda	Quadratic Discriminant Analysis	0.5650	0.5962	0.5458	0.5770	0.5389	0.1300	0.1381	3.828
svm	SVM - Linear Kernel	0.5129	0.0000	0.5817	0.5026	0.4398	0.0258	0.0358	157.977
dummy	Dummy Classifier	0.5002	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.315

1) Gradient Boosting Classifier

● 파라미터 튜닝 시도

```
gb_clf = GradientBoostingClassifier()
params = {"learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
          "min_samples_split": np.linspace(0.1, 0.5, 12),
          "min_samples_leaf": np.linspace(0.1, 0.5, 12),
          "max_depth":[3,5,6,7,8],
          "max_features":["log2","sqrt"],
          "criterion": ["friedman_mse",  "mae"],
          "subsample":[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],
          "n_estimators":[10,50,100,300,500]
         }
scorer = make_scorer(f1_score)
clf_grid = GridSearchCV(estimator=gb_clf, param_grid=params, scoring=scorer, cv=5)
```

```
clf_grid.fit(x_train2, y_train)
best_param = clf_grid.best_params_
best_param
```

11시간 돌아간 후 런타임이 끊겨
결과를 도출하지 못함

1) Gradient Boosting Classifier

● default 값으로 모델링 진행

```
scores = []
models = []

for tri, vai in cv.split(x_train2):
    print("="*50)
    preds = []

    model = GradientBoostingClassifier(random_state=42)
    model.fit(x_train2.iloc[tri], y_train[tri])
    gb_pred = model.predict(x_train2.iloc[vai])
    score = f1_score(y_train[vai], gb_pred)

    models.append(model)
    scores.append(score)

    if is_holdout:
        break
```

CrossValidation 이용

1. train 데이터로 모델링
2. validation 데이터로 예측
3. f1 score 계산

```
print(scores)
print(np.mean(scores))
```

```
[0.6132768905137579, 0.6185311494308084, 0.613427752227375, 0.6122970015740777, 0.615257048092869]
0.6145579683677777
```

1) Gradient Boosting Classifier

● threshold를 조정하여 예측력 높이기

```
threshold = 0.35
```

```
pred_list = []
scores = []
for i,(tri, vai) in enumerate( cv.split(x_train2) ):
    pred = models[i].predict_proba(x_train2.iloc[vai])[:, 1]
    pred = np.where(pred >= threshold , 1, 0)
    score = f1_score(y_train[vai],pred)
    scores.append(score)
    pred = models[i].predict_proba(x_test2)[:, 1]
    pred_list.append(pred)
print(scores)
print(np.mean(scores))
```

```
[0.6803539823008851, 0.6825540330555162, 0.6789544948639241, 0.6819163807345279, 0.6811952878618601]
0.6809948357633427
```

→ predict를 이용하여 target을 예측할 때보다 predict_proba를 이용해 확률을 구한 후 threshold를 조정하면서 target을 예측할 때 f1 score가 더 높음.

● 결과 제출

```
pred = np.mean( pred_list , axis = 0 )
pred = np.where(pred >= threshold , 1, 0)
```

```
sample_submission = pd.read_csv('sample_submission.csv')
sample_submission['target'] = pred
sample_submission.to_csv('gradient_boost.csv', index=False)
```

=〉 0.6825586746

2) Random Forest Classifier

● GridSearchCV를 이용한 하이퍼 파라미터 튜닝

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV

#하이퍼 파라미터 튜닝
params = {
    'n_estimators':[50,100,200],
    'max_depth': [5,10,15,20],
    'min_samples_leaf': [8,12,18],
    'min_samples_split': [10,15,20,25],

}
#RandomForestClassifier 객체 생성 후 GridSearchCV 수행
model_rf = RandomForestClassifier(n_jobs=-1, random_state=1234)
grid_cv = GridSearchCV(model_rf, param_grid = params, cv=2, n_jobs=-1)

X_train = train_data.drop(['target'], axis = 1)
y_train = train_data['target']

grid_cv.fit(X_train,y_train)

print('최적 하이퍼 파라미터:₩n', grid_cv.best_params_)
print('최고 예측 F1_score {0:4f}'.format(grid_cv.best_score_))
```

- 최적 하이퍼 파라미터:

{'max_depth': 15,
'min_samples_leaf': 12,
'min_samples_split': 25,
'n_estimators': 200}

- 최고 예측 F1_score: 0.592709

2) Random Forest Classifier

● Random Foreset Classifier 최종 모형

```
#최적의 하이퍼 파라미터로 다시 RandomForestClassifier 학습
model_rf1 = RandomForestClassifier(max_depth=15, min_samples_leaf=12, min_samples_split=25, n_estimators=200, n_jobs=-1, random_state=1234)
model_rf1.fit(X_train,y_train)

#예측성능
preds = model_rf1.predict(test_data)
```

최적의 하이퍼 파라미터로 학습한

Random Forest Classifier의 F1 score: 0.62792267

3) LightGBM

● trial함수 생성

```
cat_feature = X.columns[X.nunique() > 2].tolist()
```

```
def objective(trial):
    train_x, valid_x, train_y, valid_y = train_test_split(X,y, test_size=0.3)

    param = {
        "objective": trial.suggest_categorical("objective", ["binary"]),
        "depth": trial.suggest_int("depth", 1, 12),
        "learning_rate" : 0.01,
        "used_ram_limit": "3gb",
        "cat_features" : cat_feature,
        'eval_metric': 'F1',
        'random_seed' : 42
    }

    gbm = lgbm.LGBMClassifier(**param)

    gbm.fit(train_x, train_y, eval_set=[(valid_x, valid_y)], verbose=0, early_stopping_rounds=100)

    preds = gbm.predict(valid_x)
    pred_labels = np.rint(preds)
    f1 = f1_score(valid_y, pred_labels)
    return f1
```

파라미터 지정

3) LightGBM

● optuna 이용

```
[ ]  study = optuna.create_study(direction="maximize")
     study.optimize(objective, n_trials=20, timeout=600)
```

```
[I 2022-01-25 06:19:30,516] A new study created in memory with name: no-name-
[I 2022-01-25 06:19:51,756] Trial 0 finished with value: 0.6218981507055308 a
[I 2022-01-25 06:20:13,045] Trial 1 finished with value: 0.6240453662896416 a
[I 2022-01-25 06:20:33,984] Trial 2 finished with value: 0.6214808498107374 a
[I 2022-01-25 06:20:54,941] Trial 3 finished with value: 0.6212758077955746 a
[I 2022-01-25 06:21:15,862] Trial 4 finished with value: 0.6220074541178782 a
[I 2022-01-25 06:21:36,803] Trial 5 finished with value: 0.6258334095830664 a
[I 2022-01-25 06:21:57,745] Trial 6 finished with value: 0.6232307456744756 a
[I 2022-01-25 06:22:18,831] Trial 7 finished with value: 0.625811147196075[illegible] a
[I 2022-01-25 06:22:39,925] Trial 8 finished with value: 0.6206585531599432 a
[I 2022-01-25 06:23:01,122] Trial 9 finished with value: 0.6249770183975388 a
[I 2022-01-25 06:23:22,337] Trial 10 finished with value: 0.6242103444872175
[I 2022-01-25 06:23:43,485] Trial 11 finished with value: 0.622122396233962 a
[I 2022-01-25 06:24:04,682] Trial 12 finished with value: 0.6229667245921168
[I 2022-01-25 06:24:25,938] Trial 13 finished with value: 0.6224703161141094
[I 2022-01-25 06:24:46,731] Trial 14 finished with value: 0.6228219137625517
[I 2022-01-25 06:25:07,158] Trial 15 finished with value: 0.6219583546322925
[I 2022-01-25 06:25:27,885] Trial 16 finished with value: 0.6232023128498494
[I 2022-01-25 06:25:48,351] Trial 17 finished with value: 0.6217770335163888
[I 2022-01-25 06:26:08,754] Trial 18 finished with value: 0.6213296313435138
[I 2022-01-25 06:26:29,471] Trial 19 finished with value: 0.6220398434764447
```

best trial

```
Number of finished trials: 20
Best trial:
  Value: 0.6258334095830664
  Params:
    objective: binary
    depth: 2
```

3) LightGBM

● CrossValidation 이용

```
cat_params = study.best_trial.params
```

```
NFOLDS = 5
folds = StratifiedKFold(n_splits=NFOLDS, random_state=42, shuffle=True)
predictions = np.zeros(len(X_test))
for fold, (train_index, test_index) in enumerate(folds.split(X, y)):
    print("--> Fold {}".format(fold + 1))

    X_train, X_valid = X.iloc[train_index], X.iloc[test_index]
    y_train, y_valid = y.iloc[train_index], y.iloc[test_index]

    lgbm_model = lgbm.LGBMClassifier(**cat_params).fit(X_train, y_train,
                                                       eval_set=[(X_valid, y_valid)],
                                                       early_stopping_rounds=300, verbose=0)

    y_preds = lgbm_model.predict_proba(X_valid)[:,1]
    predictions += lgbm_model.predict_proba(X_test)[:,1] / folds.n_splits
```

```
--> Fold 1
--> Fold 2
--> Fold 3
--> Fold 4
--> Fold 5
```

● threshold 지정

```
threshold = 0.4
```

```
predictions= np.where(predictions >= threshold , 1, 0)
```

```
sample = pd.read_csv('/content/gdrive/MyDrive/sample_submission (1).csv')
```

```
sample['target'] = predictions
sample.to_csv('lgbm(optuna).csv',index=False)
```

=〉 0.6850464489

4) XGBoost

● 모델링

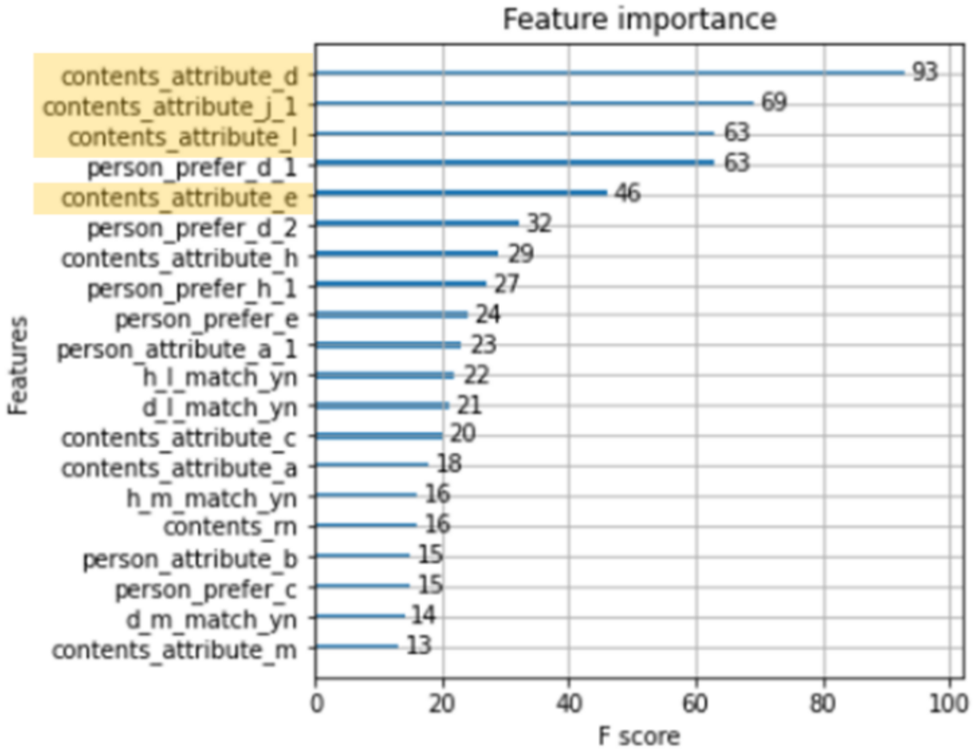
```
1 from xgboost import XGBClassifier
2
3 xgb = XGBClassifier()
4 xgb.fit(X, y)
5 pred = xgb.predict(test_data)
6 pred
```

```
array([0, 0, 0, ..., 1, 1, 1])
```

=> 0.6188207635

```
1 from xgboost import plot_importance
2
3 fig,ax = plt.subplots(1,1,figsize=(5,5))
4 plot_importance(xgb, max_num_features=20, ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7f7da1b910>
```



4) XGBoost

● 파라미터 튜닝

```python
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

xgb = XGBClassifier()

xgb_param = {
    'min_child_weight': [1, 5, 10],
    'gamma': [0.5, 1, 1.5, 2, 5],
    'max_depth': [3, 4, 5],
    'subsample':[0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 500, 1000]
}

xgb_grid = GridSearchCV(xgb, param_grid=xgb_param, scoring='f1', cv=5)
xgb_grid.fit(X, y)
```

[단계적으로 파라미터 튜닝 진행]

```
xgboost best param: {'gamma': 5, 'min_child_weight': 5}
xgboost best f1-score: 0.60637257718265
```

```
xgb = XGBClassifier(gamma = 5, min_child_weight = 5)
```

```
xgboost best param: {'max_depth': 5}
xgboost best f1-score: 0.6141511020964833
```

```
xgb = XGBClassifier(gamma = 5, min_child_weight = 5, max_depth = 5)
```

```
xgboost best param: {'colsample_bytree': 0.6, 'subsample': 1.0}
xgboost best f1-score: 0.6154869148690 84
```

```
xgb = XGBClassifier(gamma = 5, min_child_weight = 5, max_depth = 5,
                    colsample_bytree = 0.6, subsample = 1.0)
```

```
xgboost best param: {'learning_rate': 0.05, 'n_estimators': 1000}
xgboost best f1-score: 0.6158611125311999
```

=> 성능이 미세하게 향상.

4) XGBoost

● XGBoost 최종 모형

```
xgb_fin = XGBClassifier(gamma = 5, min_child_weight = 5, max_depth = 5,
                        colsample_bytree = 0.6, subsample = 1.0,
                        learning_rate = 0.05, n_estimator = 1000)
xgb_fin.fit(X, y)
pred_fin = xgb_grid.predict(test_data)
pred_fin
```

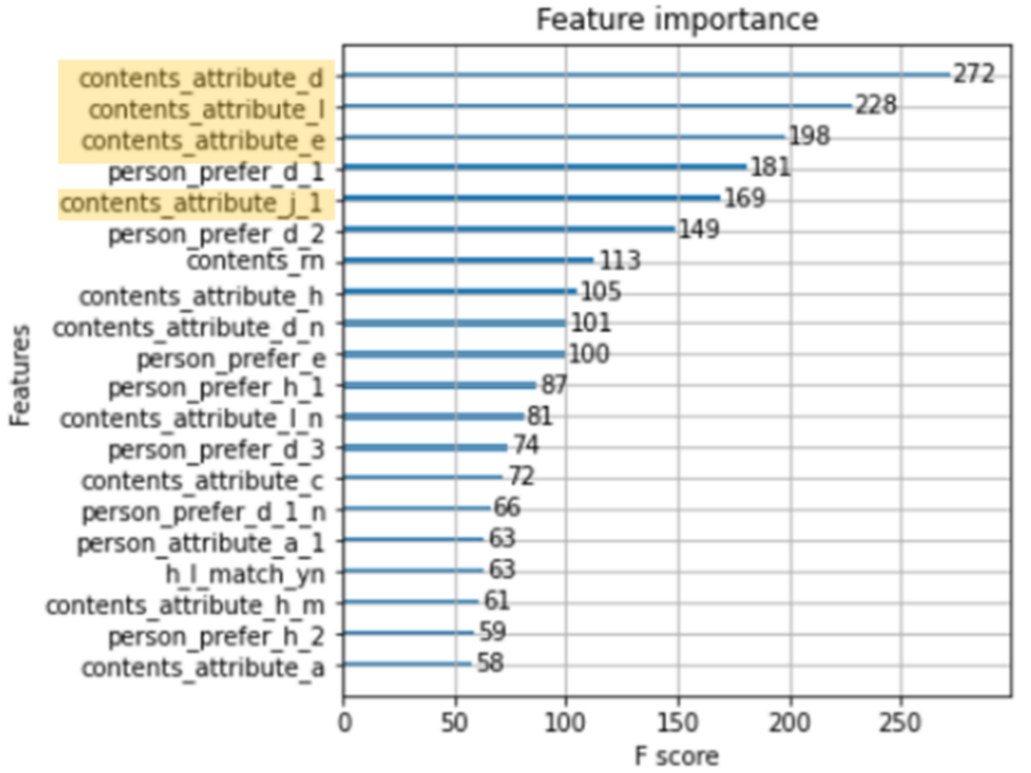
```
array([0, 0, 0, ..., 1, 1, 1])
```

=〉 0.6298912006

```
from xgboost import plot_importance

fig,ax = plt.subplots(1,1,figsize=(5,5))
plot_importance(xgb_fin, max_num_features=20, ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7f7d6b1c90>
```



5) CatBoost Classifier

● default 값으로 모델링 진행

```
cat_features = x_test.columns[x_test.nunique() > 2].tolist()
```

```
is_holdout = False
n_splits = 5
iterations = 3000
patience = 50

cv = KFold(n_splits=n_splits, shuffle=True, random_state = 0)
```

```
scores = []
models = []


models = []
for tri, vai in cv.split(x_train):
    print("="*50)
    preds = []

    model = CatBoostClassifier(iterations=iterations,random_state=0,task_type="GPU", eval_metric="F1",cat_features=cat_features,one_hot_max_size=4)
    model.fit(x_train.iloc[tri], y_train[tri],
            eval_set=[(x_train.iloc[vai], y_train[vai])],
            early_stopping_rounds=patience ,
            verbose = 100
        )

    models.append(model)
    scores.append(model.get_best_score()["validation"]["F1"])
    if is_holdout:
        break
```

```
print(scores)
print(np.mean(scores))
```

```
[0.6722609277041889, 0.6733883344339754, 0.6423987302463599, 0.6329791935512383, 0.6785384165550546]
0.6599131204981634
```

5) CatBoost Classifier

● default 값으로 모델링 진행

```
pred_list = []
scores = []
for i,(tri, vai) in enumerate( cv.split(x_train) ):
    pred = models[i].predict_proba(x_train.iloc[vai])[:, 1]
    pred = np.where(pred >= threshold , 1, 0)
    score = f1_score(y_train[vai],pred)
    scores.append(score)
    pred = models[i].predict_proba(x_test)[:, 1]
    pred_list.append(pred)
print(scores)
print(np.mean(scores))
```

```
[0.7087538367096378, 0.7030424982646607, 0.6640228363065336, 0.6684528507099419, 0.7063731722822633]
0.6901290388546074
```

```
pred = np.mean( pred_list , axis = 0 )
pred = np.where(pred >= threshold , 1, 0)
```

● 결과 제출

```
sample_submission = pd.read_csv('sample_submission.csv')
sample_submission['target'] = pred
sample_submission.to_csv('catboost_CV 5Fold.csv', index=False)
```

=〉 0.698701425

5) CatBoost Classifier

● month rank 생성

```
train_data['month'].value_counts()
```

```
2     50424
3     49029
6     47444
7     47129
1     45683
10    45605
11    44979
8     44458
4     42790
9     42393
5     42017
Name: month, dtype: int64
```

```
test_data['month'].value_counts()
```

```
12    46404
Name: month, dtype: int64
```

```
month_rank = {2:1,3:2,6:3,7:4,12:5,1:6,10:7,11:8,8:9,4:10,9:11,5:12}
```

```
train_data['month_rank'] = train_data['month'].map(month_rank)
test_data['month_rank'] = test_data['month'].map(month_rank)
```

5) CatBoost Classifier

● x_train,y_train,x_test 및 cat_features 지정

```
x_train = train_data.drop(['target'], axis = 1)
x_train.shape
```

```
(501951, 60)
```

```
y_train = train_data['target']
y_train.shape
```

```
(501951,)
```

```
x_test = test_data.copy()
```

```
cat_features = x_train.columns[x_train.nunique() > 2].tolist()
```

```
model = CatBoostClassifier(iterations=3000,random_state=1234,task_type="GPU",
                           eval_metric="F1",cat_features=cat_features,one_hot_max_size=4)
model.fit(x_train, y_train,
          eval_set=[(x_train, y_train)],
          early_stopping_rounds=50,
          verbose = 100
         )
```

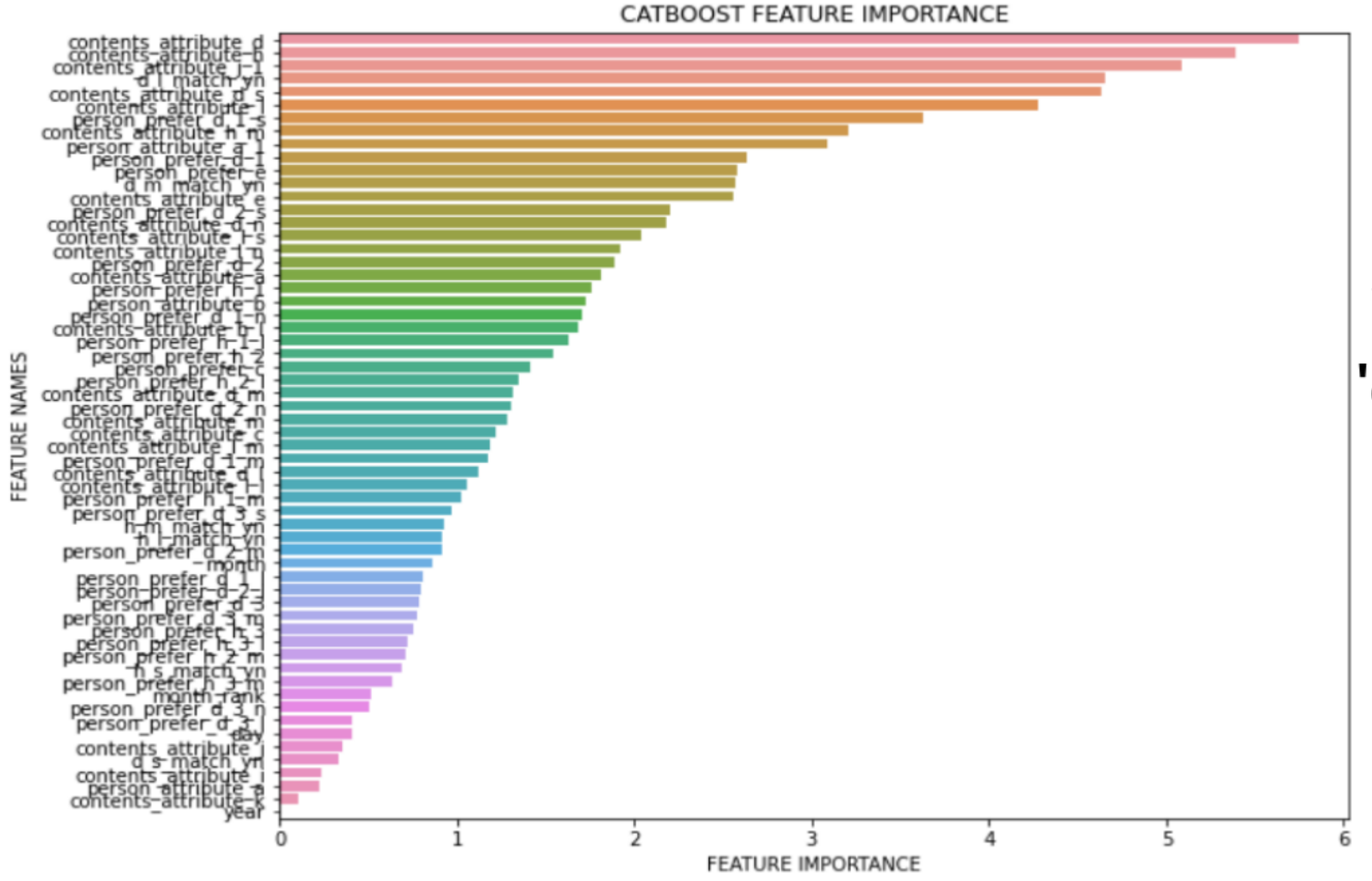
→ 변수 drop

```
bestTest = 0.8278177386
bestIteration = 2996
Shrink model to first 2997 iterations.
```

5) CatBoost Classifier

● CatBoost Feature Importance

```
plot_feature_importance(model.get_feature_importance(),x_train.columns,'CATBOOST')
```



importance이 0.5보다 작은 변수 drop

'd_s_match_yn', 'person_attribute_a', 'contents_attribute_i','contents_attribute_j', 'contents_attribute_k', 'person_prefer_d_3_l', 'year', 'day'

5) CatBoost Classifier

● 학습 파라미터

```
is_holdout = False
n_splits = 5
iterations = 1000
patience = 50

cv = KFold(n_splits=n_splits, shuffle=True, random_state=1234)
```

● CV 결과 확인

np.mean(scores)
=> 0.6794525221030916

● 학습시키기

```
scores = []
models = []

for tri, vai in cv.split(x_train):
    print("="*50)
    preds = []

    model = CatBoostClassifier(iterations=iterations,random_state=1234,task_type="GPU",
                               eval_metric="F1",cat_features=cat_features,one_hot_max_size=4)
    model.fit(x_train.iloc[tri], y_train[tri],
            eval_set=[(x_train.iloc[vai], y_train[vai])],
            early_stopping_rounds=patience ,
            verbose = 100
        )

    models.append(model)
    scores.append(model.get_best_score()["validation"]["F1"])
    if is_holdout:
        break
```

5) CatBoost Classifier

● threshold 값에 따른 검증점수 예측

```
pred_list = []
scores = []

for i,(tri, vai) in enumerate( cv.split(x_train) ):
    pred = models[i].predict_proba(x_train.iloc[vai])[:, 1]
    pred = np.where(pred >= threshold , 1, 0)
    score = f1_score(y_train[vai],pred)
    scores.append(score)
    pred = models[i].predict_proba(x_test)[:, 1]
    pred_list.append(pred)
```

np.mean(scores)
=〉 0.7074832526201054

● 산술평균 앙상블

```
pred = np.mean( pred_list , axis = 0 )
```

```
pred = np.where(pred >= threshold , 1, 0)
```

● 제출

```
sample_submission = pd.read_csv('sample_submission.csv')
sample_submission['target'] = pred
sample_submission.to_csv('catboost.csv', index=False)
```

=〉 최종 LB Score : 0.7019695836

04

결론

집에어 추천 알고리즘 경진대회

성능 비교

● 5개 모델링 결과 비교

	f1 score 기준 LB 점수
Gradient Boosting Classifier	0.68255867
Random Forest Classifier	0.62792267
LightGBM	0.68504645
XGBoost	0.62989120
Catboost Classifier	0.69870142

최종 점수

● catboost 성능 비교

	f1 score 기준 LB 점수
군집화변수 추가	0.6999074841
threshold 0.35	0.6982639574
month_rank, day_rank 추가	0.6993971595
optuna 사용	0.691233626
feature importance	0.7019695836

→ data leakage 발생

#	팀	팀 멤버	점수	제출수
78	ENjoy!		0.70196	34

감사합니다.

김도연, 김혜원, 장서연, 정세은