🎮

게임 플레이 이력 바탕_이상유저 분류

💡 **1차 채용형 코딩테스트_넥슨**

✏️ 활용 데이터

Data List

Aa 이름	데이터 유형	파일	링크
테스트 데이터	csv	test.csv	https://drive.google.com/file/d/1zPkSiWdbb7qIqFbiW3sxpaaHzCnUvOTn/view?usp=share_link
학습용 데이터	csv	train.csv	https://drive.google.com/file/d/1-Dh1uvV7i2b94fe87VVzCB7egvhTYNGK/view?usp=share_link
검증용 데이터	csv	valid.csv	https://drive.google.com/file/d/1hr4EmFpeKO2QC-I36XLsu-yiuGNkJpqd/view?usp=share_link
과제 설명	pdf	채용과제_설명.pdf	https://drive.google.com/file/d/1WIUfc2c4vagadyN-UOwmLkJskpL4gxP4/view?usp=share_link

✏️문제정의

게임 플레이 이력을 바탕으로 이상유저를 분류하는 문제입니다.

대부분의 변수는 비식별화가 적용되어 구체적인 의미를 알 수 없는 상황을 가정합니다.

✏️ 목차

0. Import

- Import Library
- Load Data

1. Data Preprocessing

- Checking Column
- Missing Value
- Outlier
- Onehot Encoding

2. Feature Generation

- Heatmap
- Histogram
- 파생변수 생성

3. Scaling

- Standard Scale
- Log Scale

4. Feature Importance

- Feataue Importance 시각화

5. Modeling

- Data Split
- DecisionTreeClassifier
- RandomForestClassifier
- ExtraTreeClassifier
- LightBGM
- AdaBoostClassifier
- KNN
- HyperParameter Tuning

6. test파일 생성 및 성능 확인

- test파일 생성
- test파일 성능 확인

✏️ 분석

- 1. Data PreProcessing, 2. Feature Generation, 3.Scaling과정은 Train Data, Test Data, Valid Data모두 동일하게 진행해주었습니다.
- 1,2,3과정은 Train Data과정으로만 작성했습니다. 자세한 사항은 코드 참고 부탁드립니다.

1. Data PreProcessing

Checking Column

```
#feature확인
print(len(train.columns))
print(len(test.columns))
print(len(val.columns))

print(train.columns)
print(test.columns)
print(val.columns)
```

```
16
15
15
Index(['account_id', 'sequence', 'char_level', 'char_type', 'job_1',
       'social_status_1', 'social_status_2', 'social_status_3',
       'social_status_4', 'activity_cum_score_1', 'activity_cum_score_2',
       'activity_cum_score_3', 'activity_score_1', 'activity_score_2',
       'activity_score_3', 'is_bot'],
      dtype='object')
Index(['account_id', 'char_level', 'char_type', 'job_1', 'social_status_1',
       'social_status_2', 'social_status_3', 'social_status_4',
       'activity_cum_score_1', 'activity_cum_score_2', 'activity_cum_score_3',
       'activity_score_1', 'activity_score_2', 'activity_score_3', 'is_bot'],
      dtype='object')
Index(['account_id', 'char_level', 'char_type', 'job_1', 'social_status_1',
       'social_status_2', 'social_status_3', 'social_status_4',
       'activity_cum_score_1', 'activity_cum_score_2', 'activity_cum_score_3',
       'activity_score_1', 'activity_score_2', 'activity_score_3', 'is_bot'],
      dtype='object')
```

→ 'sequence' column은 train에만 존재하고 test, val에는 존재하지 않습니다.

Missing Value

```
[ ] train.isnull().sum()

account_id              0
char_level              0
char_type               0
job_1                   0
social_status_1         0
social_status_2         0
social_status_3         0
social_status_4         0
activity_cum_score_1    0
activity_cum_score_2    0
activity_cum_score_3    0
activity_score_1        0
activity_score_2        0
activity_score_3        0
is_bot                  0
sequence                0
dtype: int64
```

→ 결측값은 없는 것으로 확인되었습니다.

Outlier

- 이상치 처리 기준 : 1사분위값 - IQR*1.5미만 or 3사분위값 + IQR*1.5초과

```
columns = ['char_level', 'activity_score_1', 'activity_score_3']
for col in columns:
    q1=train1[col].quantile(0.25)
    q3=train1[col].quantile(0.75)
    IQR=q3-q1
    # 소수점(1.5, 2.5) 형태로 나와서 올림 처리
    line_down = math.ceil(q1 - IQR * 1.5)
    line_up = math.ceil(q3 + IQR * 1.5)

    train1[col] = train1[col].clip(line_down, line_up)
```

- 이상치 처리 feature : char_level, activity_score_1, activity_score_3

- 이상치 없는 feature : char_type, job_1 ⇒ 이상치 처리를 진행하지 않았습니다.
- 이상치가 매우 높은 feature : activity_cum_score_1,activity_cum_score_2, activity_cum_score_3,activity_score_2 => 이상치 높은 feature는 scaling진행했습니다.

Onehot Encoding

```
train1 = train1.replace({True:1, False:0})
```

True는 1로, False는 0으로 원핫인코딩을 진행하였습니다.

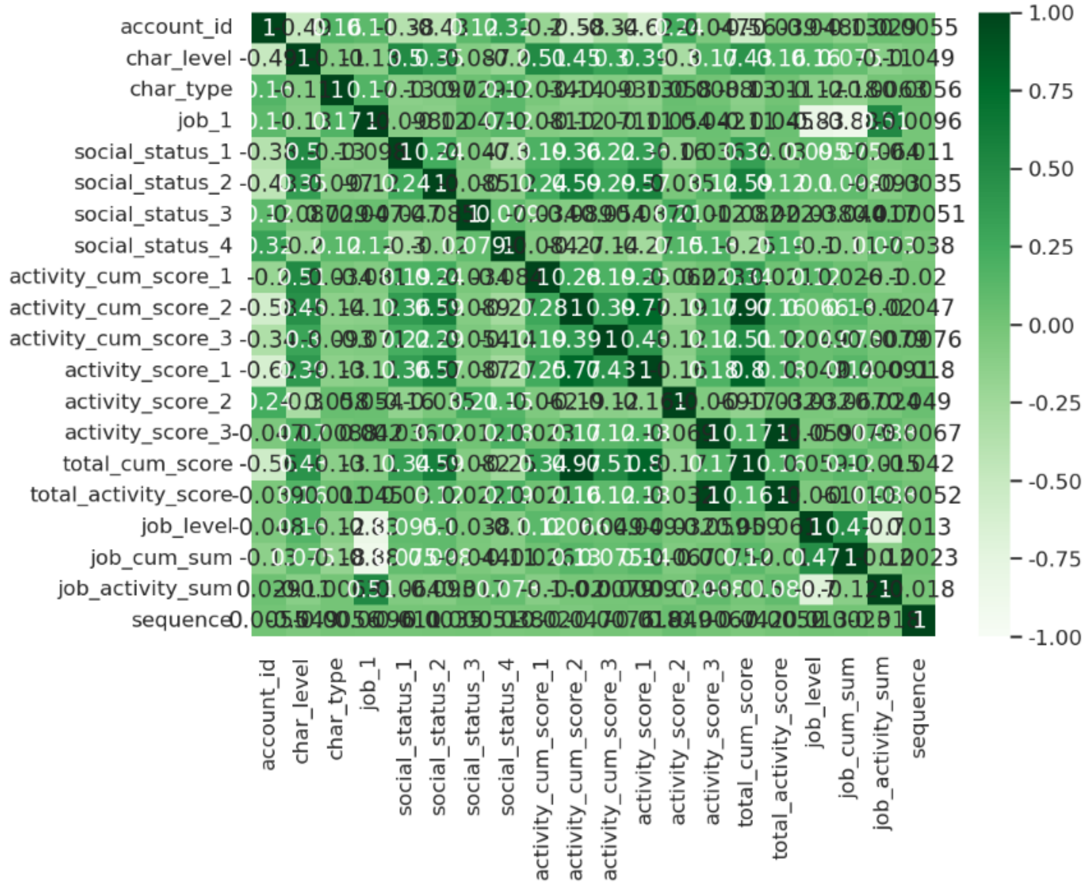
2. Feature Generation

Heatmap

- 상관관계를 확인하기 위해 heatmap을 형성했습니다.

```
plt.rcParams['figure.figsize'] = (8,6)
sns.heatmap(train1.corr(),annot=True, cmap='Greens',vmin=-1,vmax=1)
```

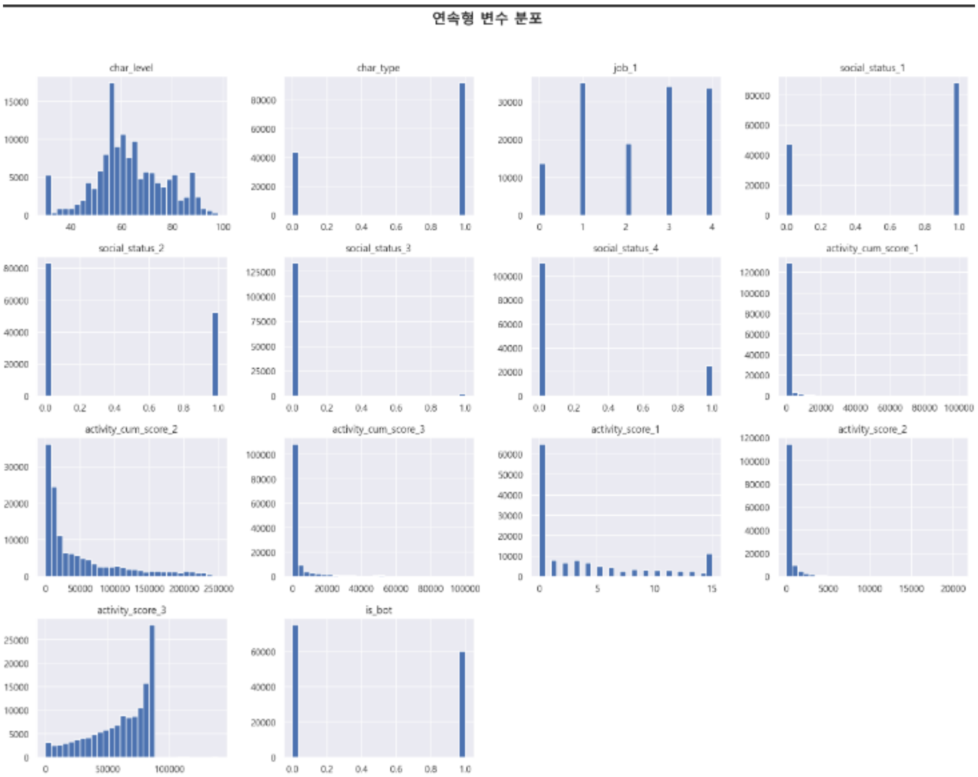
<AxesSubplot:>



Histogram

- 연속형 변수의 histogram을 확인했습니다.

```
g = train1.set_index('account_id').hist(bins=30,figsize=(20,15))
plt.suptitle("연속형 변수 분포", x=0.5, y=0.95, ha='center', fontsize='xx-large', fontweight=800)
plt.show()
```



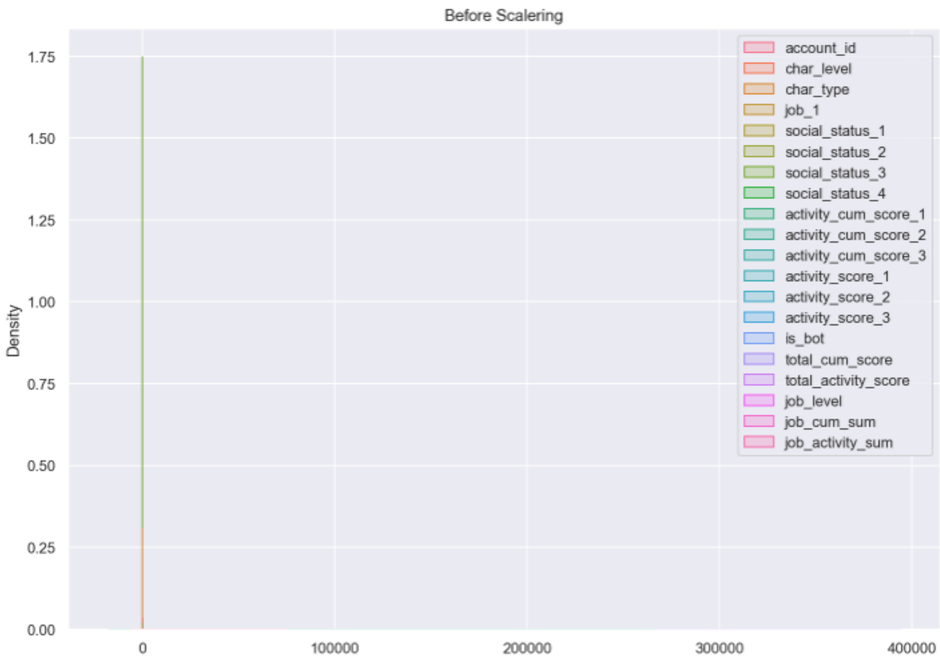
파생 변수 생성

```
#총 활동점수
train1['total_cum_score'] = train1['activity_cum_score_1'] + train1['activity_cum_score_2'] + train1['activity_cum_score_3']
#직업별 캐릭터 레벨
train1['total_activity_score'] = train1['activity_score_1'] + train1['activity_score_2'] + train1['activity_score_3']
#직업별 총 누적 점수
train1['job_level'] = train1.groupby(['job_1'])['char_level'].transform('mean')
#직업별 총 활동 점수
train1['job_cum_sum'] = train1.groupby(['job_1'])['total_cum_score'].transform('mean')
#날짜별 캐릭터 레벨
train1['job_activity_sum'] = train1.groupby(['job_1'])['total_activity_score'].transform('mean')
```

3. Scaling

- 정확도를 높이기 위해 scaling을 진행했습니다.

```
#Before Scalering
numerical_feats = X.dtypes[X.dtypes == "int64"].index.tolist()
li = X.dtypes[X.dtypes == "float64"].index.tolist()
numerical_feats = numerical_feats + li
sns.set(rc={'figure.figsize':(11.7,8.27)},)
sns.kdeplot(data = X[numerical_feats], shade = True).set_title('Before Scalering')
plt.show()
```



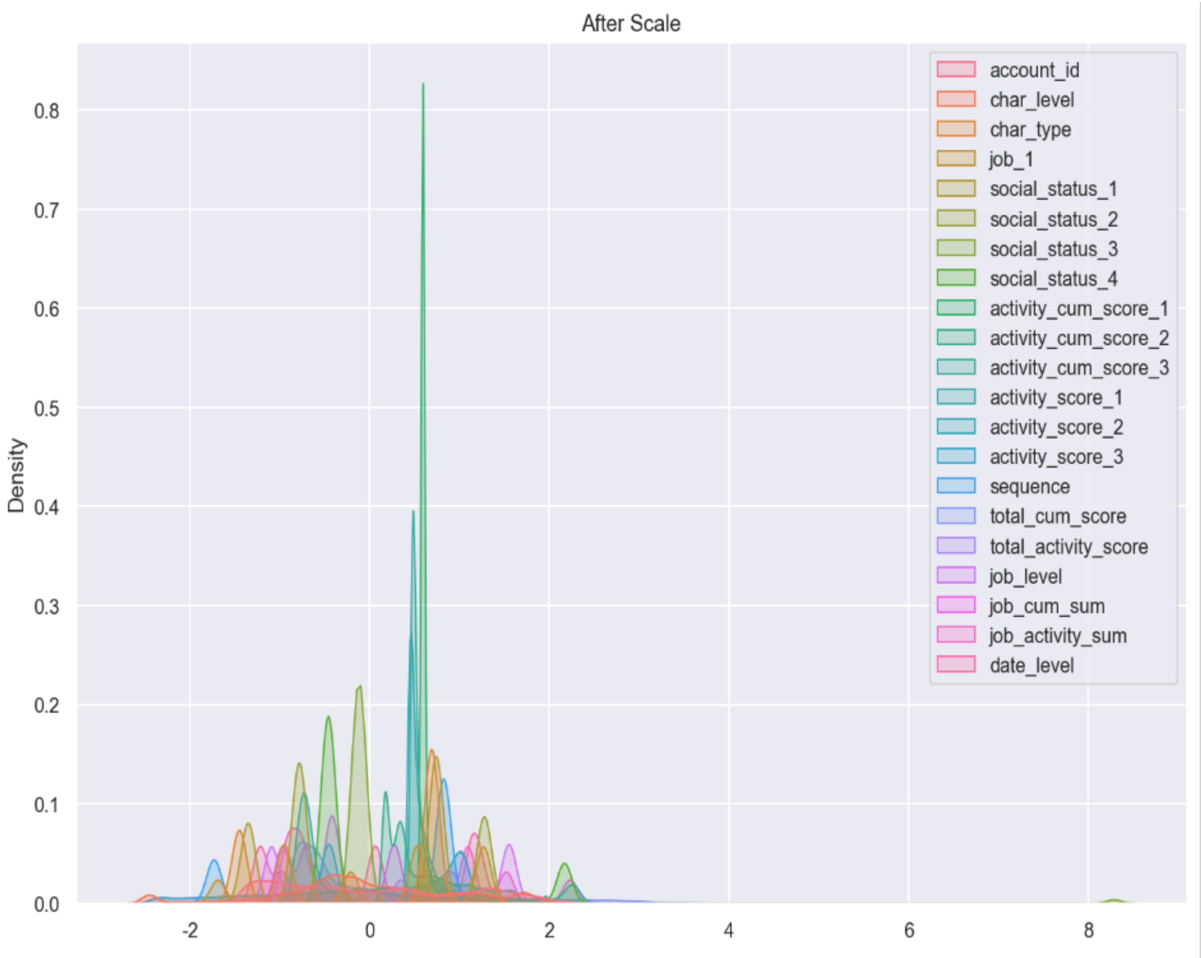
Standard Scale

```
#standard Scaler
scaler = StandardScaler()
train1[numerical_feats] = scaler.fit_transform(train1[numerical_feats])
```

Log Scale

```
train1['activity_cum_score_1'] = np.log1p(1+train1['activity_cum_score_1'])
train1['activity_cum_score_2'] = np.log1p(1+train1['activity_cum_score_2'])
train1['activity_cum_score_3'] = np.log1p(1+train1['activity_cum_score_3'])
train1['activity_score_2'] = np.log1p(1+train1['activity_score_2'])

#After Scalering
sns.set(rc={'figure.figsize':(11.7,8.27)},)
sns.kdeplot(data = train1[numerical_feats], shade = True).set_title('After Scale')
plt.show()
```



위의 과정을 validation data와 test data에도 동일하게 진행해주었습니다.

validation data, test data Sequence칼럼 생성

- sequence를 target으로 지정해서 Randomforest모델을 훈련시킨 후 validation data와 test data의 sequence를 예측했습니다.

1. val 데이터의 sequence column을 생성합니다.

```
#train데이터에서 target값으로 'sequence' column설정

X_train = train1
y_train = train['sequence']
X_train
```

```
[ ] val
```

	account_id	char_level	char_type	job_1	social_status_1	social_status_2	social_status_3	social_statu
0	-1.439992	-2.311883	0.689309	0.528488	-1.342962	1.301525	-0.117026	-0.456
1	-1.407956	1.388120	0.689309	1.266048	0.744623	1.301525	-0.117026	-0.456
2	-1.386284	-0.077919	-1.450729	-0.209071	0.744623	1.301525	-0.117026	-0.456
3	-1.386284	2.016422	-1.450729	0.528488	0.744623	1.301525	-0.117026	-0.456
4	-1.386284	-1.055278	-1.450729	-0.946631	0.744623	1.301525	-0.117026	-0.456
...	...	...	...	...	...	...	...	
57064	1.843422	-0.776033	0.689309	-0.209071	0.744623	-0.768330	-0.117026	2.189
57065	1.853787	-0.776033	-1.450729	1.266048	-1.342962	-0.768330	-0.117026	2.189
57066	1.935962	-1.334524	0.689309	1.266048	-1.342962	-0.768330	-0.117026	2.189
57067	1.964229	-1.055278	0.689309	-0.946631	-1.342962	-0.768330	-0.117026	2.189
57068	1.981091	-1.404335	0.689309	1.266048	-1.342962	-0.768330	-0.117026	2.189

57069 rows × 21 columns

```
#val데이터에 0으로 채워진 sequence 칼럼 추가

val['sequence'] = np.nan
val = val.fillna(0)
X_val = val.drop(columns=['sequence'])
y_val = val['sequence']
X_val
```

```
#sequence를 target 지정해서 모델 훈련시킨 후 validation data의 sequence예측

rmf = RandomForestClassifier(n_estimators=200, criterion='entropy',random_state=42)
rmf.fit(X_train,y_train)
rmf_pred = rmf.predict(X_val)
seq = pd.DataFrame(rmf_pred).rename(columns={0:'sequence'})

#위에서 예측한 seq값을 validation데이터의 sequence에 할당
X_val['sequence'] = seq
val = X_val
val
```

job_activity_sum	sequence
-0.586712	2
1.133165	1
1.466911	2
-0.586712	1
-0.958068	2
...	...
1.466911	2
1.133165	1
1.133165	2
-0.958068	2
1.133165	2

2. test 데이터도 동일한 방식으로 진행합니다.

```
#test데이터에 0으로 채워진 sequence 칼럼 추가

test['sequence'] = np.nan
test = test.fillna(0)
X_test = test.drop(columns=['sequence'])
y_test = test['sequence']

#sequence를 target 지정해서 모델 훈련시킨 후 test data의 sequence예측
rmf_pred = rmf.predict(X_test)
seq = pd.DataFrame(rmf_pred).rename(columns={0:'sequence'})

#위에서 예측한 seq값을 validation데이터의 sequence에 할당
X_test['sequence'] = seq
test = X_test
```

```
test
```

	account_id	char_level	char_type	job_1	social_status_1	social_status_2	social_status_3	social_statu
0	-1.404971	1.176131	-1.455868	-0.953552	0.750720	1.278274	-0.124867	-0.43
1	-1.403865	0.900773	0.686876	-1.691339	0.750720	1.278274	-0.124867	-0.43
2	-1.403865	0.281218	0.686876	1.259809	0.750720	1.278274	-0.124867	-0.43
3	-1.403865	1.726847	-1.455868	-0.953552	0.750720	1.278274	-0.124867	-0.43
4	-1.403865	0.281218	0.686876	-0.953552	0.750720	1.278274	-0.124867	-0.43
...	...	...	...	...	...	...	...	
60898	1.807958	-1.715128	0.686876	1.259809	-1.332055	-0.782305	-0.124867	-0.43
60899	1.866409	-1.095573	0.686876	1.259809	-1.332055	-0.782305	-0.124867	-0.43
60900	1.888491	-0.682536	0.686876	0.522022	0.750720	-0.782305	-0.124867	-0.43
60901	1.922070	-0.957894	0.686876	1.259809	-1.332055	-0.782305	-0.124867	-0.43
60902	1.945884	-1.783968	0.686876	1.259809	-1.332055	-0.782305	-0.124867	2.27

60903 rows × 21 columns

m	job_activity_sum	sequence
9	-0.786995	2
1	-0.626449	1
4	1.101774	1
9	-0.786995	1
9	-0.786995	1
..	...	...
4	1.101774	2
4	1.101774	2
6	-0.845760	2
4	1.101774	2
4	1.101774	2

3. 마무리로 X_train에 다시 sequence를 추가하고 각 데이터의 column수가 동일한지 확인합니다

```
#colab
sequence = pd.read_csv('/content/drive/MyDrive/FA_COACHING/train.csv')['sequence']
#local
#sequence = pd.read_csv('train.csv')['sequence']
train = pd.concat([X_train,sequence],axis=1)
print(len(train.columns))
print(len(test.columns))
print(len(val.columns))
```

```
21
21
21
```

4. Feature Importance

Feataue Importance 시각화

- PermutationImportance 라이브러리를 이용해서 피쳐들의 중요성을 데이터프레임으로 제작해서 시각화하였습니다.

```
import eli5
from eli5.sklearn import PermutationImportance

imp = rmf.feature_importances_
imp_df = pd.DataFrame(imp, index=X_train.columns, columns=["imp"])
imp_df = imp_df.sort_values("imp", ascending=False)
imp_df
```

→ 새롭게 만든 activity_cum_score_2, activity_cum_3, activity_cum_score가 유의하다는 것을 알 수 있었고 전반적으로 score피쳐들의 중요성이 높다는 것을 확인하였습니다.

	imp
activity_cum_score_2	0.122298
account_id	0.114834
activity_score_3	0.113263
total_activity_score	0.111246
activity_cum_score_3	0.106151
total_cum_score	0.104191
activity_score_2	0.098671
char_level	0.052897
activity_cum_score_1	0.041985
activity_score_1	0.039976
job_activity_sum	0.012131
job_level	0.011801
job_cum_sum	0.011625
job_1	0.011445
char_type	0.010982
social_status_1	0.010299
social_status_2	0.009282
social_status_4	0.007846
is_bot	0.007521
social_status_3	0.001556

5. Modeling

Data Split

-데이터를 X_train, y_train, X_val, y_val, X_test, y_test로 분할했습니다.

```
train = X[:len(train)]
test = X[len(train):len(train)+len(test)]
val = X[len(train)+len(test):]
```

```
X_train = train
X_test = test
X_val = val
#colab
y_train = pd.read_csv('/content/drive/MyDrive/프로젝트/게임플레이이력바탕_이상유저분류/train.csv')['is_bot']
y_test = pd.read_csv('/content/drive/MyDrive/프로젝트/게임플레이이력바탕_이상유저분류/test.csv')['is_bot']
y_val = pd.read_csv('/content/drive/MyDrive/프로젝트/게임플레이이력바탕_이상유저분류/valid.csv')['is_bot']
# #local
# y_train = pd.read_csv('train.csv')['is_bot']
# y_test = pd.read_csv('test.csv')['is_bot']
# y_val = pd.read_csv('valid.csv')['is_bot']
```

```
(135368, 20) (60903, 20) (57069, 20)
(135368,) (60903,) (57069,)
```

Modeling : 분할된 X_train, y_train으로 모델 학습후, val 데이터를 통해 검증을 진행하며 최적화된 모델을 선정했습니다.

DecisionTreeClassifier

```
dt = DecisionTreeClassifier(random_state=20)
dt.fit(X_train,y_train)
dt_pred = dt.predict(X_val)
print('accuracy : ', accuracy_score(y_val,dt_pred))
print('f1 : ', f1_score(y_val,dt_pred))
```

```
accuracy :  0.6611119872435123
f1 :  0.5587094418838133
```

RandomForestClassifier

```
rmf = RandomForestClassifier(n_estimators=200, criterion='entropy',random_state=42)
rmf.fit(X_train,y_train)
rmf_pred = rmf.predict(X_val)
print('accuracy : ', accuracy_score(y_val,rmf_pred))
print('f1 : ', f1_score(y_val,rmf_pred))
```

```
accuracy :  0.7748690182060313
f1 :  0.6809535634467346
```

ExtraTreeClassifier

```
ex = ExtraTreesClassifier(random_state = 42)
ex.fit(X_train, y_train)
ex_pred = ex.predict(X_val)
print('accuracy : ', accuracy_score(y_val,ex_pred))
print('f1 : ', f1_score(y_val,ex_pred))
```

```
accuracy :  0.8119995093658554
f1 :  0.7202711510885151
```

LightBGM

```
lgbm = LGBMClassifier()
lgbm.fit(X_train,y_train)
lgbm_pred = lgbm.predict(X_val)
print('accuracy : ', accuracy_score(y_val,lgbm_pred))
print('f1 : ', f1_score(y_val,lgbm_pred))
```

```
accuracy :  0.697278732762095
f1 :  0.5647266313932982
```

AdaBoostClassifier

```
ada = AdaBoostClassifier(n_estimators=100) #아다부스트
ada.fit(X_train, y_train)
ada_pred = ada.predict(X_val)
print('accuracy : ', accuracy_score(y_val,ada_pred))
print('f1 : ', f1_score(y_val,ada_pred))
```

```
accuracy :  0.7613941018766756
f1 :  0.6463759835873996
```

KNN

```
knn  = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_val)
```

```
print('accuracy : ', accuracy_score(y_val,knn_pred))
print('f1 : ', f1_score(y_val,knn_pred))
```

```
accuracy :  0.7631113213828874
f1 :  0.6301737108466694
```

모델 평균 성능 확인

- 여러개의 모델을 사용해서 accuracy를 확인하였고 가장 높은 score가 나온 모델을 최종모델로 선정하였습니다.

[모델 목록]

> 💡 DecisionTreeClassification, RandomforestClassifier, ExtraTreeClassifier, LightBGM, #AdaBoostClassifier

```
train_X = X_train
train_y = y_train
# accuracy_score 함수
def ACCURACY(y_val, y_pred_val):
    accuracy = accuracy_score(y_val, y_pred_val)
    return accuracy

# Cross Validation 함수
def accuracy_cv(model):
    # cv별로 학습하는 함수
    tscv = TimeSeriesSplit(n_splits=10)
    accuracy_list = []
    model_name = model.__class__.__name__
    for _, (train_index, test_index) in tqdm(enumerate(tscv.split(train_X), start=1), desc=f'{model_name} Cross Validations...', total
        X_train, X_test = train_X.iloc[train_index], train_X.iloc[test_index]
        y_train, y_test = train_y.iloc[train_index], train_y.iloc[test_index]
        clf = model.fit(X_train, y_train)
        pred_val = clf.predict(X_val)
        accuracy = ACCURACY(y_val, pred_val)
        accuracy_list.append(accuracy)
    return model_name, accuracy_list

#cv별 프린팅, 평균 저장
def print_accuracy_score(model):
    # cv별 프린팅, 평균 저장
    model_name, score = accuracy_cv(model)
    for i, r in enumerate(score, start=1):
        print(f'{i} FOLDS: {model_name} RMSLE: {r:.4f}')
    print(f'\n{model_name} mean ACCURACY: {np.mean(score):.4f}')
    print('='*40)
    return model_name, np.mean(score)
```

```
#모델 정의
dt = DecisionTreeClassifier(random_state=20)
rmf = RandomForestClassifier(n_estimators=200, criterion='entropy',random_state=42)
ex = ExtraTreesClassifier(random_state = 42)
lgbm = LGBMClassifier()
ada = AdaBoostClassifier(n_estimators=100)
knn  = KNeighborsClassifier(n_neighbors=4)

models = []
scores = []
for model in [dt, rmf, ex, lgbm, knn]:
    model_name, mean_score = print_accuracy_score(model)
    models.append(model_name)
    scores.append(mean_score)
```

```
result_df = pd.DataFrame({'Model': models, 'Score': scores}).reset_index(drop=True)
result_df
```

	Model	Score
0	DecisionTreeClassifier	0.630919
1	RandomForestClassifier	0.746449
2	ExtraTreesClassifier	0.770860
3	LGBMClassifier	0.696639
4	AdaBoostClassifier	0.649177
5	KNeighborsClassifier	0.715395

⇒가장 좋은 성능을 나타내는 ExtraTreeClassifier을 최종 모델로 선정했습니다.

HpyerParameter Tuning

- 종합적으로 가장 좋은 성능을 나타내는 모델인 ExtraTreetClassifier로 파라미터 튜닝을 진행했습니다.
- ExtraTreetClassifier로 Optuna 실행하였습니다.

```
def objective(trial):

    ### define params grid to search maximum accuracy
    n_estimators = trial.suggest_int('n_estimators', 50, 300)
    max_depth = trial.suggest_int('max_depth', 10, 30)
    max_leaf_nodes = trial.suggest_int('max_leaf_nodes', 15, 30)
    criterion = trial.suggest_categorical('criterion', ['gini', 'entropy'])

    ### modeling with suggested params
    model = ExtraTreesClassifier(n_estimators = n_estimators,
                                 max_depth = max_depth,
                                 max_leaf_nodes = max_leaf_nodes,
                                 criterion = criterion,
                                 random_state = 0) # do not tune the seed

    ### fit
    model.fit(X_train, y_train)
    preds = model.predict(X_val)
    y_pred_val = model.predict(X_val)
    score = accuracy_score(y_val,y_pred_val)
    score_mean = score.mean()

    return score_mean

study = optuna.create_study(direction='maximize') # maximize accuracy
study.optimize(objective, n_trials=30)
print('Number of finished trials:', len(study.trials))
print('Best trial:', study.best_trial.params)
print('Best score:', study.best_value)
```

```
Number of finished trials: 30
Best trial: {'n_estimators': 264, 'max_depth': 21, 'max_leaf_nodes': 29, 'criterion':
Best score: 0.7575916872557781
```

```
params = {'n_estimators': 250, 'class_weight':'balanced','max_features':'sqrt','min_samples_split':8,'random_state':42}

ex = ExtraTreesClassifier(**params)
ex.fit(X_train, y_train)
ex_pred = ex.predict(X_val)
print('accuracy : ', accuracy_score(y_val,ex_pred))
print('f1 : ', f1_score(y_val,ex_pred))
```

```
accuracy :  0.8404913350505528
f1 :  0.7618574231523871
```

⇒ 시도해보았지만 성능개선을 가져오지 않아서 튜닝파라미터는 적용하지 않았습니다.

6. test파일 생성 및 성능 확인

test파일 생성

- 성능이 가장 좋은 ExtraTreesClassifier로 test파일의 target인 is_bot를 예측하였고 최종 test파일을 생성하였습니다.

```
#test 파일생성
ex = ExtraTreesClassifier(random_state = 42)
ex.fit(X_train, y_train)
pred = ex.predict(X_test)
pred = pd.DataFrame(pred)
test = pd.concat([test.reset_index(drop=True),pred.reset_index(drop=True)],axis=1)
test = test.rename(columns={0:'is_bot'})
test.to_csv('./test_made_03161257.csv',index=False)
test
```

	account_id	char_level	char_type	job_1	social_status_1	social_status_2	social_status_3
0	-1.404971	1.176131	-1.455868	-0.953552	0.750720	1.278274	-0.124867
1	-1.403865	0.900773	0.686876	-1.691339	0.750720	1.278274	-0.124867
2	-1.403865	0.281218	0.686876	1.259809	0.750720	1.278274	-0.124867
3	-1.403865	1.726847	-1.455868	-0.953552	0.750720	1.278274	-0.124867
4	-1.403865	0.281218	0.686876	-0.953552	0.750720	1.278274	-0.124867
...	...	...	...	...	...	...	...
60898	1.807958	-1.715128	0.686876	1.259809	-1.332055	-0.782305	-0.124867
60899	1.866409	-1.095573	0.686876	1.259809	-1.332055	-0.782305	-0.124867
60900	1.888491	-0.682536	0.686876	0.522022	0.750720	-0.782305	-0.124867
60901	1.922070	-0.957894	0.686876	1.259809	-1.332055	-0.782305	-0.124867
60902	1.945884	-1.783968	0.686876	1.259809	-1.332055	-0.782305	-0.124867

60903 rows × 21 columns

test파일 성능 확인

RandomForestClassifier을 사용해서 최종 test파일의 최종 성능을 확인했습니다.

```
#colab
X_test = pd.read_csv('/content/drive/MyDrive/프로젝트/게임플레이이력바탕_이상유저분류/test_made_03161257.csv').drop(columns='is_bot')
y_test = pd.read_csv('/content/drive/MyDrive/프로젝트/게임플레이이력바탕_이상유저분류/test_made_03161257.csv')['is_bot']
# #local
# X_test = pd.read_csv('test_made_03161257.csv').drop(columns='is_bot')
# y_test = pd.read_csv('test_made_03161257.csv')['is_bot']
ex = ExtraTreesClassifier(n_estimators=200, criterion='entropy',random_state=42)
ex.fit(X_train,y_train)
ex_pred = ex.predict(X_test)
print('accuracy : ', accuracy_score(y_test,ex_pred))
print('f1 : ', f1_score(y_test,ex_pred))
```

```
accuracy :  0.9566031229988671
f1 :  0.9289344196176496
```

✏️결론

- 최종 선택 모델 및 성능

최종선택 모델 : ExtraTreeClassifier

최종 성능 : accuracy : **0.9566031229988671** / f1 : **0.9289344196176496**

- 코드

https://drive.google.com/file/d/1Thp2WmYsu2jGRNWqkBGREPwwsCDkixBa/view?usp=sharing

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/bee2bb96-18e2-4976-9071-e3a087296072/%EC%9D%B4%EC%83%81%EC%9C%A0%EC%A0%80_%EB%B6%84%EB%A5%98_%EC%B5%9C%EC%A2%85%EC%BD%94%EB%93%9C.ipynb

- 최종 test 파일

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/f82e4122-7403-45c7-868f-cc89a0d202fb/test_made_03121154.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	account_id	char_level	char_type	job_1	social_stat	social_stat	social_stat	social_stat	activity_cu	activity_cu	activity_cu	activity_sc	activity_sc	activity_sc	total_cum_	total_activi	job_level	job_cum_s	job_activit	sequence	is_bot
2	-1.404971	1.176131	-1.455868	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.693748	0.195165	1.003543	-0.525081	0.781812	1.061008	-0.617058	1.07072	1.499331	0.223679	-0.786995	2	FALSE
3	-1.403865	0.900773	0.686876	-1.691339	0.75072	1.278274	-0.124867	-0.439512	0.604169	1.366947	0.57523	2.245857	0.784013	0.907787	1.735669	0.917926	0.373664	2.206631	-0.626449	1	FALSE
4	-1.403865	0.281218	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	0.59431	1.366947	0.57523	2.245857	0.784013	0.907787	1.734738	0.917926	-1.187124	-0.960314	1.101774	1	FALSE
5	-1.403865	1.726847	-1.455868	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.898862	1.366947	0.57523	2.245857	0.784013	0.907787	1.768195	0.917926	1.499331	0.223679	-0.786995	1	FALSE
6	-1.403865	0.281218	0.686876	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.59431	1.366947	0.57523	2.245857	0.784013	0.907787	1.734738	0.917926	1.499331	0.223679	-0.786995	1	FALSE
7	-1.403865	1.176131	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	0.685517	1.366947	0.57523	2.245857	0.784013	0.907787	1.743712	0.917926	-1.187124	-0.960314	1.101774	1	FALSE
8	-1.403865	1.795687	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	0.851391	1.366947	0.57523	2.245857	0.784013	0.907787	1.762287	0.917926	-1.187124	-0.960314	1.101774	1	FALSE
9	-1.403865	0.281218	0.686876	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.59448	1.366947	0.57523	2.245857	0.784013	0.907787	1.734754	0.917926	1.499331	0.223679	-0.786995	1	FALSE
10	-1.403865	0.281218	-1.455868	-0.215765	0.75072	1.278274	-0.124867	-0.439512	0.59448	1.366947	0.57523	2.245857	0.784013	0.907787	1.734754	0.917926	-0.359988	1.022909	1.526764	1	FALSE
11	-1.403865	0.831934	0.686876	-0.215765	0.75072	1.278274	-0.124867	-0.439512	0.595334	1.366947	0.57523	2.245857	0.784013	0.907787	1.734834	0.917926	-0.359988	1.022909	1.526764	1	FALSE
12	-1.403865	2.277563	-1.455868	-1.691339	0.75072	1.278274	-0.124867	-0.439512	1.794405	1.366947	0.57523	2.245857	0.784013	0.907787	1.952802	0.917926	0.373664	2.206631	-0.626449	1	FALSE
13	-1.403865	0.281218	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	0.59431	1.366947	0.57523	2.245857	0.784013	0.907787	1.734738	0.917926	-1.187124	-0.960314	1.101774	1	FALSE
14	-1.403865	1.107292	-1.455868	-1.691339	0.75072	1.278274	-0.124867	-0.439512	0.690029	1.366947	0.57523	2.245857	0.784013	0.907787	1.744178	0.917926	0.373664	2.206631	-0.626449	1	FALSE
15	-1.403865	0.281218	-1.455868	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.59431	1.366947	0.57523	2.245857	0.784013	0.907787	1.734738	0.917926	1.499331	0.223679	-0.786995	1	FALSE
16	-1.403865	1.176131	0.686876	-1.691339	0.75072	1.278274	-0.124867	-0.439512	0.661688	1.366947	0.57523	2.245857	0.784013	0.907787	1.741288	0.917926	0.373664	2.206631	-0.626449	1	FALSE
17	-1.403865	1.726847	0.686876	0.522022	0.75072	1.278274	-0.124867	-0.439512	1.069299	1.366947	0.57523	2.245857	0.784013	0.907787	1.791875	0.917926	-0.326684	-0.700316	-0.84576	1	FALSE
18	-1.403865	0.281218	0.686876	0.522022	0.75072	1.278274	-0.124867	-0.439512	0.59431	1.366947	0.57523	2.245857	0.784013	0.907787	1.734738	0.917926	-0.326684	-0.700316	-0.84576	1	FALSE
19	-1.403865	0.281218	0.686876	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.59448	1.366947	0.57523	2.245857	0.784013	0.907787	1.734754	0.917926	1.499331	0.223679	-0.786995	1	FALSE
20	-1.403865	0.694255	0.686876	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.592942	1.366947	0.57523	2.245857	0.784013	0.907787	1.734609	0.917926	1.499331	0.223679	-0.786995	1	FALSE
21	-1.401267	0.969613	0.686876	0.522022	-1.332055	-0.782305	-0.124867	-0.439512	0.603323	0.693594	0.531373	0.266616	0.574498	-2.174403	-0.052351	-2.188075	-0.326684	-0.700316	-0.84576	1	FALSE
22	-1.400978	1.176131	-1.455868	-0.953552	0.75072	-0.782305	-0.124867	-0.439512	0.673279	0.660854	0.558513	-0.723005	0.558103	-0.211783	-0.098106	-0.222711	1.499331	0.223679	-0.786995	2	FALSE
23	-1.397515	-0.407178	-1.455868	-0.953552	-1.332055	1.278274	-0.124867	-0.439512	0.592771	0.36262	0.498421	-0.723005	1.210521	0.090687	-0.582557	0.146193	1.499331	0.223679	-0.786995	2	FALSE
24	-1.397515	-0.476017	0.686876	0.522022	-1.332055	1.278274	-0.124867	-0.439512	0.592771	0.36262	0.498421	-0.723005	1.210521	0.090687	-0.582557	0.146193	-0.326684	-0.700316	-0.84576	2	FALSE
25	-1.387027	1.31381	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	0.639566	0.174662	0.514806	2.245857	0.474835	0.069574	-0.801601	0.054115	-1.187124	-0.960314	1.101774	2	FALSE
26	-1.387027	2.071045	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	2.105545	0.174662	0.514806	2.245857	0.474835	0.069574	-0.473999	0.054115	-1.187124	-0.960314	1.101774	2	FALSE
27	-1.387027	1.38265	0.686876	-0.953552	0.75072	1.278274	-0.124867	-0.439512	0.649961	0.174662	0.514806	2.245857	0.474835	0.069574	-0.800573	0.054115	1.499331	0.223679	-0.786995	2	FALSE
28	-1.387027	1.244971	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	0.600441	0.174662	0.514806	2.245857	0.474835	0.069574	-0.805374	0.054115	-1.187124	-0.960314	1.101774	2	FALSE
29	-1.387027	1.726847	0.686876	1.259809	0.75072	1.278274	-0.124867	-0.439512	0.976404	0.174662	0.514806	2.245857	0.474835	0.069574	-0.76222	0.054115	-1.187124	-0.960314	1.101774	2	FALSE
30	-1.381928	2.208724	-1.455868	-1.691339	0.75072	1.278274	-0.124867	-0.439512	2.223853	0.170862	1.775744	2.245857	0.477824	-0.90173	0.202316	-0.919032	0.373664	2.206631	-0.626449	2	FALSE
31	-1.376924	0.350057	0.686876	-0.953552	0.75072	-0.782305	-0.124867	-0.439512	0.596017	0.16881	0.491883	-0.723005	0.701581	-1.764477	-0.817832	-1.767635	1.499331	0.223679	-0.786995	2	FALSE
32	-1.369179	1.38265	0.686876	0.522022	0.75072	1.278274	-0.124867	-0.439512	0.77553	1.125561	0.509157	1.256236	0.489695	-0.681511	0.958983	-0.69776	-0.326684	-0.700316	-0.84576	0	FALSE
33	-1.369179	2.002205	0.686876	-0.215765	0.75072	1.278274	-0.124867	-0.439512	2.088323	1.125561	0.509157	1.256236	0.489695	-0.681511	1.264992	-0.69776	-0.359988	1.022909	1.526764	0	FALSE

▼ 버전 기록

v1: randomforest_기존_accuracy_score : 0.8464490353782264 fl_score : 0.7652117996945583
v2: randomforest_이상치 처리_accuracy_score : 0.8464490353782264 fl_score : 0.7652117996945583(그대로)
v3: randomforest_minmaxscaling_ accuracy_score : 0.8464490353782264 fl_score : 0.7652369598414016(조금 향상)
v4: randomforest_log_scale_ accuracy_score : 0.8465191259703166 fl_score : 0.7653441208776488(조금 향상)
v5:lgbm_accuracy_score_ 0.8243529762217666 fl_score : 0.7211372614477273 (lgbm성능 별로)
v6:catboost_score_ accuracy_score : 0.8429094604776673 fl_score : 0.7590248098271644 (catboost 성능보통)
v7: randomfoerst_account_id제외_accuracy_score : 0.8285584117471833f1_score : 0.7341593305075534 (크게 감소)
v8: randomforest_val data에 sequence열 만들기_accuracy : 0.6775657537367047 f1 : 0.0(크게 감소)
v9: randomforest_pca로 차원 축소 : 성능 감소
v10:randomforest_standard : accuracy : 0.8462562862499781 f1 : 0.764923373700568
v11:Extraforest : accuracy : 0.8478683698680545 f1 : 0.7718025548020817
v12: Extraforest_optuna : accuracy : 0.6933361369570169 f1 : 0.24878739751899384
v13 : sequence data추가_extratree : accuracy : accuracy : 0.859275613730747 f1 : 0.7883404053448594

v3.1 : feature 추가_extratree: accuracy : 0.851916101561268 f1 : 0.7746219697575806

v4.1 : test파일 검증_extratree: accuracy : 0.9271628655402854 f1 : 0.8717622571692877