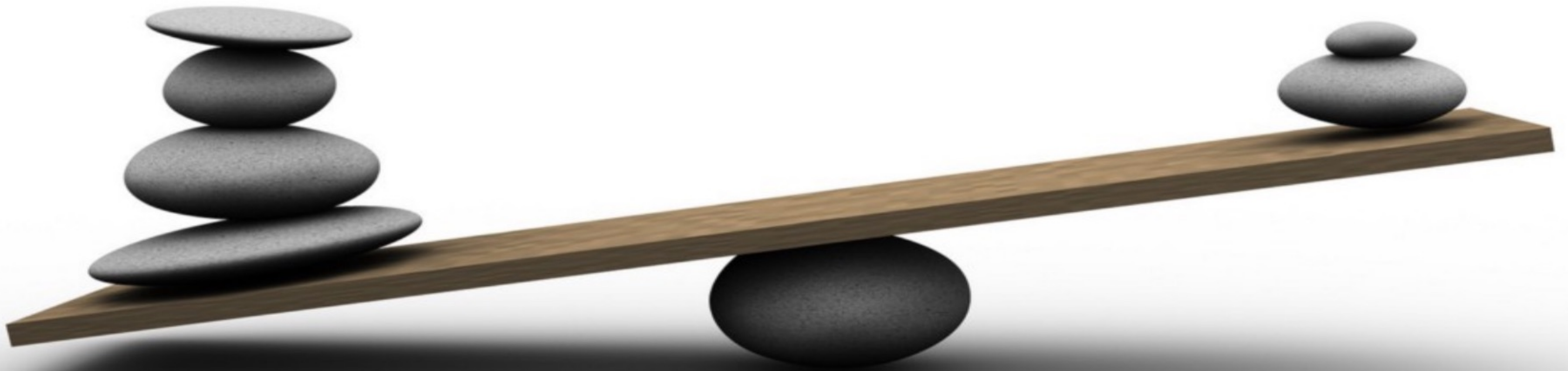


Kaggle KML Challenge 2021F



Contents

1. 데이터셋 생성 과정

- 1) Train data 기간설정
- 2) Feature set1
- 3) Feature set2
- 4) Feature set3
- 5) Feature Engineering

2. 모델링

- 1) ML modeling – Out_Of_Fold
- 2) Dnn Modeling
- 3) 최종 앙상블

3. Etc

- 1) Behind
- 2) 느낀점 및 배운점

Train data 기간 설정

Train 데이터를 시간 기준으로 두 개(예: train01, train02)로 나누어 feature를 생성한 1등 팀이 성능을 올릴 수 있었던 주요 이유는 다음과 같습니다.

1) 학습 데이터로 사용한 train02의 기간이 1년에서 5개월로 축소된 영향 때문(거의 대부분)

=> 다른 것은 그대로 두고 4개월로 한달 더 줄여봤더니 성능이 올라감

2) Data Leakage를 해결하기 위해 train01을 이용하여 만든 누적 feature(Points_earned)의 영향은 약간 정도

-> data leakage 문제를 해결한 features보다는 데이터가 축소된 영향

Feature를 만들 때 after_data만을 사용한 영향인가? 단순히 data 크기가 축소되어 그런 것인가?

-> 실험결과 전체 데이터로 feature set을 만들고, 데이터를 축소하는 것이 성능 향상에 도움이 되었음

-> 실험결과 2021-02-20 이후의 train data만 가져가는 것이 성능이 제일 향상됨

결론 : 전체 train data로 features를 만든 후, 2021-02-20 이후를 train set으로 사용

Feature set1. base

	column_name
17	Response_Probability
15	STATUS_ratio_CPI
18	STATUS_ratio_hour
1	TITLE
9	MONTH
10	WEEKDAY
8	DAY
16	TYPE_mean
13	STATUS_ratio_IR
14	STATUS_ratio_LOI
11	HOUR
2	IR
4	CATEGORIES
20	g_i
21	AGE
3	LOI
12	CATEGORIES_CNT
0	Questions_count
7	Points_earned_mean
6	Points_earned
19	weekday_mean
5	CPI
24	TYPE_B

-> 1, 2, 3등 features + 1round 실험결과 의미있는 우리팀의 features

1. Target 값을 활용한 mean encoding한 features

-> 요일별 응답확률, 타입별 응답확률, 유저아이디별 응답확률, 시간대별 응답확률, CPI, LOI, IR별 응답확률

2. 만들어낸 features

- > CATEGORIES를 교수님 코드에서 TITLE을 처리한 것과 같은 방식으로 영어 단어 빈도수를 뽑아냄(*CATEGORIES_cnt*)
- > 유저아이디별로 응답한 횟수에 대한 피쳐생성(*res_cnt*)
- > CATEGORIES를 Label Encoding

```
import re
# 띄어쓰기 구분해서 한글만 추출
def testing(s):
    hangul = re.compile('[^ㄱ-ㅣ가-힣+]')
    result = hangul.sub(" ", s)
    result = result.strip()
    return result

train["TITLE"] = train.TITLE.map(testing)
test["TITLE"] = test.TITLE.map(testing)

# 카테고리 count
test.query("CATEGORIES!=CATEGORIES").TITLE.unique()

array(['해외 일반인 의견 조사'], dtype=object)
```

-> test 데이터 내 CATEGORIES가 NaN값인 행의 TITLE

Feature set2. DEEPLARNING FEATURES

```
[ ] X_train.drop(columns = ["CATEGORIES_CNT", "STATUS_ratio_LOI", "STATUS_ratio_IR", "Points_earned"], inplace=True)
    X_test.drop(columns = ["CATEGORIES_CNT", "STATUS_ratio_LOI", "STATUS_ratio_IR", "Points_earned"], inplace=True)
    # #dnn best
```

```
#X_train.drop(columns = ["Points_earned_mean", "CATEGORIES_CNT", "Points_earned"])
#loss: 0.3732 - acc: 0.8308 - auc_8: 0.9119 - val_loss: 0.3745 - val_acc: 0.8305 - val_auc_8: 0.9115

# X_train.drop(columns = ["Points_earned_mean", "CATEGORIES_CNT"], inplace=True)
# loss: 0.3735 - acc: 0.8306 - auc_9: 0.9117 - val_loss: 0.3751 - val_acc: 0.8299 - val_auc_9: 0.9111

#X_train.drop(columns = ["Points_earned_mean", "Points_earned"], inplace=True)
# loss: 0.3736 - acc: 0.8306 - auc_10: 0.9117 - val_loss: 0.3750 - val_acc: 0.8301 - val_auc_10: 0.9112

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "CPI", "TYPE_B"], inplace=True)
#loss: 0.3741 - acc: 0.8304 - auc_11: 0.9114 - val_loss: 0.3754 - val_acc: 0.8299 - val_auc_11: 0.9110

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "TYPE_B"], inplace=True)
#loss: 0.3741 - acc: 0.8304 - auc_12: 0.9115 - val_loss: 0.3753 - val_acc: 0.8299 - val_auc_12: 0.9110

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "weekday_mean"], inplace=True)
#loss: 0.3737 - acc: 0.8303 - auc_13: 0.9116 - val_loss: 0.3746 - val_acc: 0.8302 - val_auc_13: 0.9113

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "g_i"], inplace=True)
#loss: 0.3739 - acc: 0.8306 - auc_14: 0.9115 - val_loss: 0.3748 - val_acc: 0.8303 - val_auc_14: 0.9113

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "CPI"], inplace=True)
#loss: 0.3743 - acc: 0.8304 - auc_16: 0.9114 - val_loss: 0.3755 - val_acc: 0.8302 - val_auc_16: 0.9111

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "MONTH"], inplace=True)
#loss: 0.3780 - acc: 0.8287 - auc_17: 0.9095 - val_loss: 0.3794 - val_acc: 0.8280 - val_auc_17: 0.9090

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "AGE"], inplace=True)
#loss: 0.3743 - acc: 0.8302 - auc_18: 0.9114 - val_loss: 0.3753 - val_acc: 0.8298 - val_auc_18: 0.9110

#X_train.drop(columns = ["Points_earned_mean", "Points_earned", "CATEGORIES_CNT", "STATUS_ratio_LOI"], inplace=True)
```

BASE로 지정한 FEATURE SET1에서 위의 feature제외

- > DNN모델의 성능이 제일 잘 나오도록 feature들을 빼보는 실험
- > 위의 피쳐들은 데이터를 생성해 test에 merge할 시 결측값이 많이 생기는 features였음
- > 실험 결과로 dnn feature set은 2021-02-01이후의 데이터만으로 feature생성 후 사용

Feature set3. BASE + 교수님 코드

```
[ 'RES_RATE_3M', 'STATUS_ratio_CPI', 'Response_Probability',  
  'HOURCLS3_RES_RATE', 'MONTH', 'STATUS_ratio_hour', 'Points_earned_mean',  
  'HOURCLS1_RES_RATE', 'WEEKDAY', 'TITLE_CNT', 'HOURCLS0_RES_RATE', 'DAY',  
  'RES_RATE_TREND', 'TYPE_B', 'WEEKDAY2_RES_RATE', 'HOUR', 'IR',  
  'HOURCLS', 'LOI', 'WEEKDAY5_RES_RATE', 'CATEGORIES', 'SQ7',  
  'WEEKDAY0_RES_RATE', 'CPI', 'STATUS_ratio_LOI', 'TYPE_mean',  
  'weekday_mean', 'T_R', 'STATUS_ratio_IR', 'AGE', 'CATEGORIES_CNT',  
  'HOURCLS2_RES_RATE', 'B1', 'SQ6', 'WEEKDAY6_RES_RATE', 'res_cnt',  
  'Points_earned', 'WEEKDAY4_RES_RATE', 'WEEKDAY1_RES_RATE', 'C_R', 'SQ5',  
  'WEEKDAY3_RES_RATE', 'A1', 'g_i', 'B3', 'F_R', 'D_R', 'X_R', 'B4', 'B5',  
  'SQ4', 'B2' ],
```

BASE

- + data_processing파일의 panel, survey, response 데이터의 전처리 과정을 거친 features 일부
- + data_processing파일의 파생변수 일부

Feature Engineering – 결측값 처리

```
# 범주형 변수는 가장 많이 나온 값으로 채움
apply_most_frequent_features = cat_features
features[apply_most_frequent_features] = SimpleImputer(strategy='most_frequent').\
fit_transform(features[apply_most_frequent_features])
```

범주형 변수 : 최빈값으로 결측값 대체

```
test.STATUS_ratio_CPI=test.STATUS_ratio_CPI.fillna(test.STATUS_ratio_CPI.mean())
test.STATUS_ratio_IR=test.STATUS_ratio_IR.fillna(test.STATUS_ratio_IR.mean())
test.STATUS_ratio_LOI=test.STATUS_ratio_LOI.fillna(test.STATUS_ratio_LOI.mean())
```

위의 수치형 변수 -> column의 평균값

```
# num feature 중 0으로 채울 대상
apply_zero_features = ['SQ_R', 'A_R', 'B_R', 'C_R', 'D_R', 'F_R', 'H_R', 'T_R', 'X_R', 'ALL_R',
                      'RES_RATE_3M', 'WEEKDAY0_RES_RATE', 'WEEKDAY1_RES_RATE', 'WEEKDAY2_RES_RATE', 'WEEKDAY3_RES_RATE',
                      'WEEKDAY4_RES_RATE', 'WEEKDAY5_RES_RATE', 'WEEKDAY6_RES_RATE',
                      'HOURCLS0_RES_RATE', 'HOURCLS1_RES_RATE', 'HOURCLS2_RES_RATE', 'HOURCLS3_RES_RATE']
features[apply_zero_features] = SimpleImputer(strategy='constant', fill_value=0).\
fit_transform(features[apply_zero_features])
```

위의 수치형 변수 -> 0

Feature Engineering – Encoding 및 Scaling

```
#StandardScaler
from sklearn.preprocessing import StandardScaler

# 스케일링 전 train, test 분할
X_train = features.iloc[:y_train.shape[0], :]
X_test = features.iloc[y_train.shape[0]:, :]

scaler = StandardScaler()
X_train[num] = scaler.fit_transform(X_train[num])
X_test[num] = scaler.transform(X_test[num])

# 원핫 인코딩
df_encoded = pd.get_dummies(pd.concat([X_train, X_test]), columns=cat)
X_train = df_encoded[:X_train.shape[0]]
X_test = df_encoded[X_train.shape[0]:]
```

- > CATEGORIES는 Label Encoding 적용
- > 그 외 일괄적으로 StandardScaling과 one-hot-encoding 적용

▾ LGBM OOF

```
[ ] from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import KFold
    kf = KFold(n_splits = 5, shuffle = True, random_state = 0)

[ ] model = LGBMClassifier(random_state = 44)
    lgbm_pred = np.zeros((X_test.shape[0]))
    auc_list = []
    for tr_idx, val_idx in kf.split(X_train, y_train):
        tr_x, tr_y = X_train.iloc[tr_idx], y_train.iloc[tr_idx]
        val_x, val_y = X_train.iloc[val_idx], y_train.iloc[val_idx]

        model.fit(tr_x, tr_y)
        pred = model.predict_proba(val_x)[:, 1]

        auc = roc_auc_score(val_y, pred)
        auc_list.append(auc)

        sub_pred = np.array(model.predict_proba(X_test)[:, 1]) / 5
        lgbm_pred += sub_pred
    print(f'{model.__class__.__name__}의 5fold 평균 AUC는 {np.mean(auc_list)}')
```

📄 LGBMClassifier의 5fold 평균 AUC는 0.9237755739630498

```
[ ] sub["STATUS"] = lgbm_pred
    sub.to_csv(path + "/lgbm_oof.csv", index = False)
```

* Out-of-Fold

- K = 5일 때 K-fold 교차검증 방법 사용
- 5번의 학습마다 test에 대한 예측값 계산 후 이들을 산술평균
- 1round 때 실험결과 튜닝한 모델보다 기본모델 + OOF방법이 성능향상에 도움
- K-fold에서 shuffle = True 지정으로 train의 기간별로 학습되는 것을 방지

Dnn Modeling

```
for i in tqdm(range(20)):
    SEED = np.random.randint(1, 10000)
    reset_seeds(SEED)

    # Define the NN architecture
    input = keras.Input(shape=(X_train.shape[1],))
    x = keras.layers.Dense(32, activation='relu')(input)
    x = keras.layers.Dense(16, activation='relu')(x)
    output = keras.layers.Dense(1, activation='sigmoid')(x)
    model = keras.Model(input, output)

    # Choose the optimizer and the cost function
    model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.Nadam(lr=0.001), metrics=['acc', keras.metrics.AUC()])

    # Train the model
    callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)]
    hist = model.fit(X_train, y_train, validation_data=(X_dev, y_dev), batch_size=4096, epochs=50,
                    callbacks=callbacks, shuffle=False, verbose=0)

    # Make submissions
    submission = pd.DataFrame({
        "ID": test_id["ID"],
        "STATUS": model.predict(X_test).flatten()
    })
    t = pd.Timestamp.now()
    fname = f"{folder}/loop_submission_{t.month:02}{t.day:02}_{SEED:05}.csv"
    submission.to_csv(fname, index=False)
    # inputp1.5mean_submission_1214_0337
```

```
nf = 0
for f in os.listdir(folder):
    ext = os.path.splitext(f)[-1]
    if ext == '.csv':
        s = pd.read_csv(folder+"/"+f)
    else:
        continue
    if len(s.columns) != 2:
        continue
    if nf == 0:
        slist = s
    else:
        slist = pd.merge(slist, s, on="ID")
    nf += 1

p = 1.5 # 이 값에 따라 성능이 달라짐 (p=0: 기하평균, p=1: 산술평균)
if nf >= 2:
    if p == 0:
        pred = 1
        for j in range(nf): pred = pred * slist.iloc[:,j+1]
        pred = pred**(1/nf)
    else:
        pred = 0
        for j in range(nf): pred = pred + slist.iloc[:,j+1]**p
        pred = pred / nf
        pred = pred**(1/p)
    submission = pd.DataFrame({'ID': slist.ID, 'STATUS': pred})
    t = pd.Timestamp.now()
    fname = f"p{p}mean_submission_{t.month:02}{t.day:02}_{t.hour:02}{t.minute:02}.csv"
    submission.to_csv(path + fname, index=False)
```


-> DNN 시드 앙상블을 통해 성능을 향상시키려함

앙상블 전 : 0.89079 -> 앙상블 후 : 0.89574

최종LB SCORE 앙상블과정

Feature set1  SUB1
RandomForest + Catboost
(LB 0.90170)

Feature set2  SUB2
DNN (LB 0.89547)

Feature set3  SUB3
RandomForest + Catboost
(LB 0.90455)

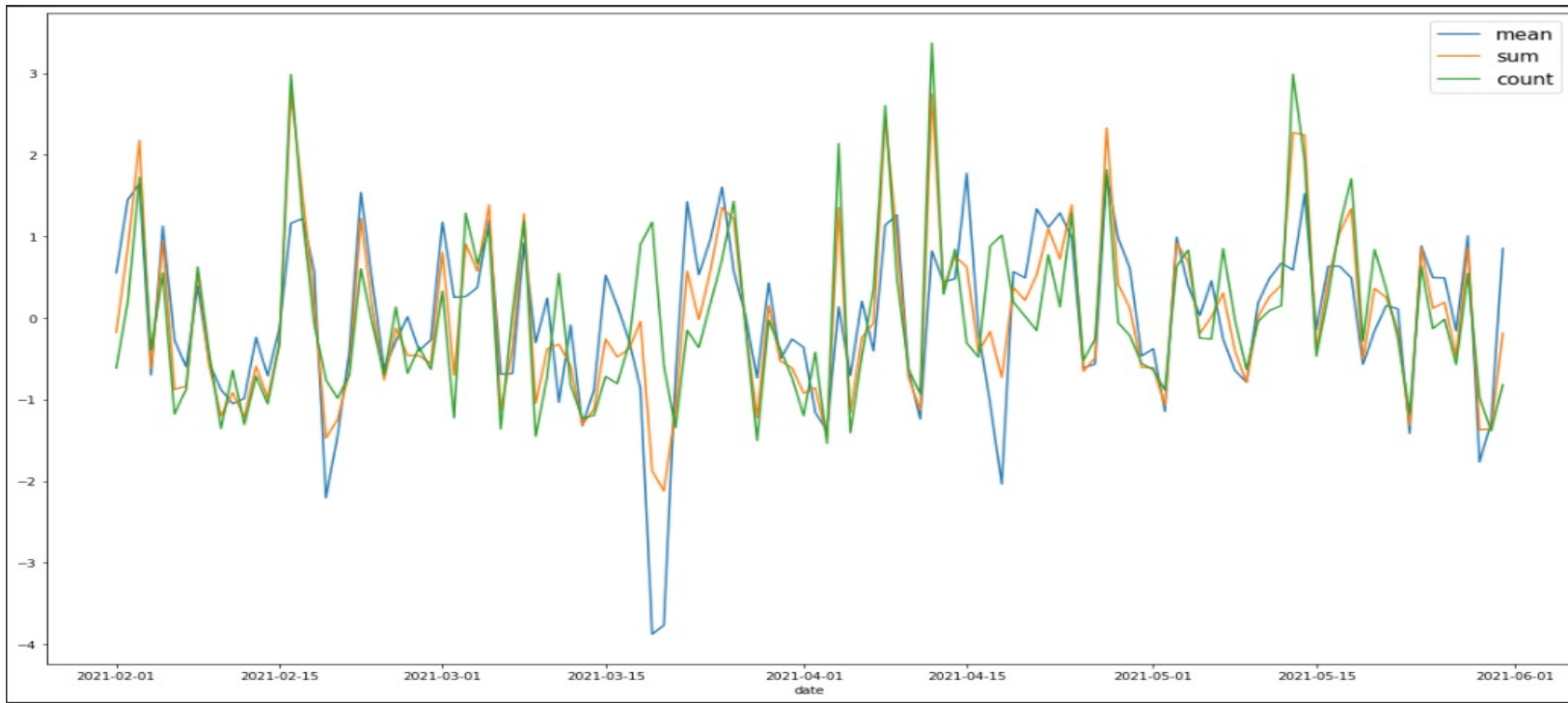
실험결과 가장 높은 성능

$$\text{SUB1} * 0.3 + \text{SUB2} * 0.2 + \text{SUB3} * 0.5$$

-> LB Public score : 0.90689

Behind

1. 시계열 데이터



- > train data 2021-02-21 ~ 2021-05-31
- > date(일자)별 응답확률, 응답횟수, survey 수신 횟수
- > 그래프 상으로 어느 정도 비례하며, 상관계수는 0.8이상
- !! 일자별 STATUS의 COUNT(survey 수신 횟수)는 test에서도 가능!!
- > 일자별 STATUS의 count()는 data leakage없이 test data 자체에서 만들 수 있어 일자별 응답확률에 가까운 역할을 할 것이라고 예상
- > feature로 추가 결과 성능 향상에 큰 도움이 안됨

2. dnn모델 시드값 조절 + 튜닝

```
for i in tqdm(range(20)):  
    SEED = np.random.randint(1, 10000)  
    reset_seeds(SEED)  
    # Define the NN architecture  
    def model_fn(hp):  
        inputs = keras.Input(shape=(X_train.shape[1],))  
        x = keras.layers.Dense(hp.Int('unit', 8, 16, step=8), hp.Choice('activation', ['relu', 'elu']))(inputs)  
        x = keras.layers.Dropout(hp.Float('dropout', 0, 0.25, step=0.25, default=0.25))(x)  
        outputs = keras.layers.Dense(1, activation='sigmoid')(x)  
        model = keras.Model(inputs, outputs)  
        model.compile(loss='binary_crossentropy',  
                      optimizer=tf.keras.optimizers.Adam(hp.Choice('learning_rate', [1e-2, 1e-3])),  
                      metrics=[keras.metrics.AUC()])  
        return model  
    # Choose the optimizer and the cost function  
    # model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.Nadam(lr=0.001), metrics=['acc', keras.  
    # print(tf.__version__)  
    tuner = kt.Hyperband(model_fn,  
                        objective=kt.Objective('val_auc', direction="max"),  
                        max_epochs=5,  
                        hyperband_iterations=2,  
                        overwrite=True,  
                        directory='dnn_tuning')  
  
    tuner.search(X_train, y_train, validation_data=(X_dev, y_dev),  
               callbacks=[tf.keras.callbacks.EarlyStopping(patience=1)])  
    model = tuner.get_best_models(1)[0]  
  
    # Train the model  
    callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)]  
    hist = model.fit(X_train, y_train, validation_data=(X_dev, y_dev), batch_size=4096, epochs=50,
```

- > 성능 향상에 큰 도움이 안됨
- > 시간이 너무 오래걸림(12시간 이상)

Behind

1. W2V

```
[ ] # test
total_vector_list_te=[]
for i,k in tqdm_notebook(enumerate(CAT_te_df.CATEGORIES)) :
    vec_list=np.array([0.0 for i in range(300)])

    if k==k: #nan이면 false가 나온다
        tempL=re.split(' - |,|, | -',k)
        if 'E' in tempL :
            tempL[tempL.index('E')]='Electronic'
        cnt=0
        for word in tempL:
            if word:
                cnt+=1
                vec_list+=model.wv[word]
        vec_list = vec_list/cnt
        total_vector_list_te.append(vec_list)
total_vector_list_te = np.array(total_vector_list_te)
```

-> CATEGORIES&TITLE에 대해 W2V 진행

-> 문장이 아닌 단어로 이루어져있기 때문에 단어의 뜻이 저장되어있어 바로 쓸수 있는 google w2v사용

-> 차원이 너무커져 과적합 발생한 것으로 추정

2. 소득 featuers

```
#DQ5 그룹의 평균으로 DQ6 결측치 채우기
#train
grouped = train[["DQ5","DQ6"]].groupby("DQ5")
fill_values = {1.0 : 5, 2.0 : 5, 3.0 : 6, 4.0 : 6, 5.0 : 7, 6.0 : 7, 7.0 : 7, 8.0 : 7 , 99.0 : 5}
fill_func = lambda g : g.fillna(fill_values[g.name])
train[["DQ5","DQ6"]] = grouped.apply(fill_func)
#test
grouped_te = test[["DQ5","DQ6"]].groupby("DQ5")
test[["DQ5","DQ6"]] = grouped_te.apply(fill_func)

#DQ5/DQ6 둘다 nan인것들 결측치 채우기
# 전체 참여자 소득의 평균
print("status 신경안쓰고 전체")
mean_income=round(train.groupby("userID")[ "DQ6" ].agg([ ("mean_income",np.mean) ] ).mean())
print(mean_income)
train["DQ6"]=train.DQ6.fillna(6)
test["DQ6"]=test.DQ6.fillna(6)
```

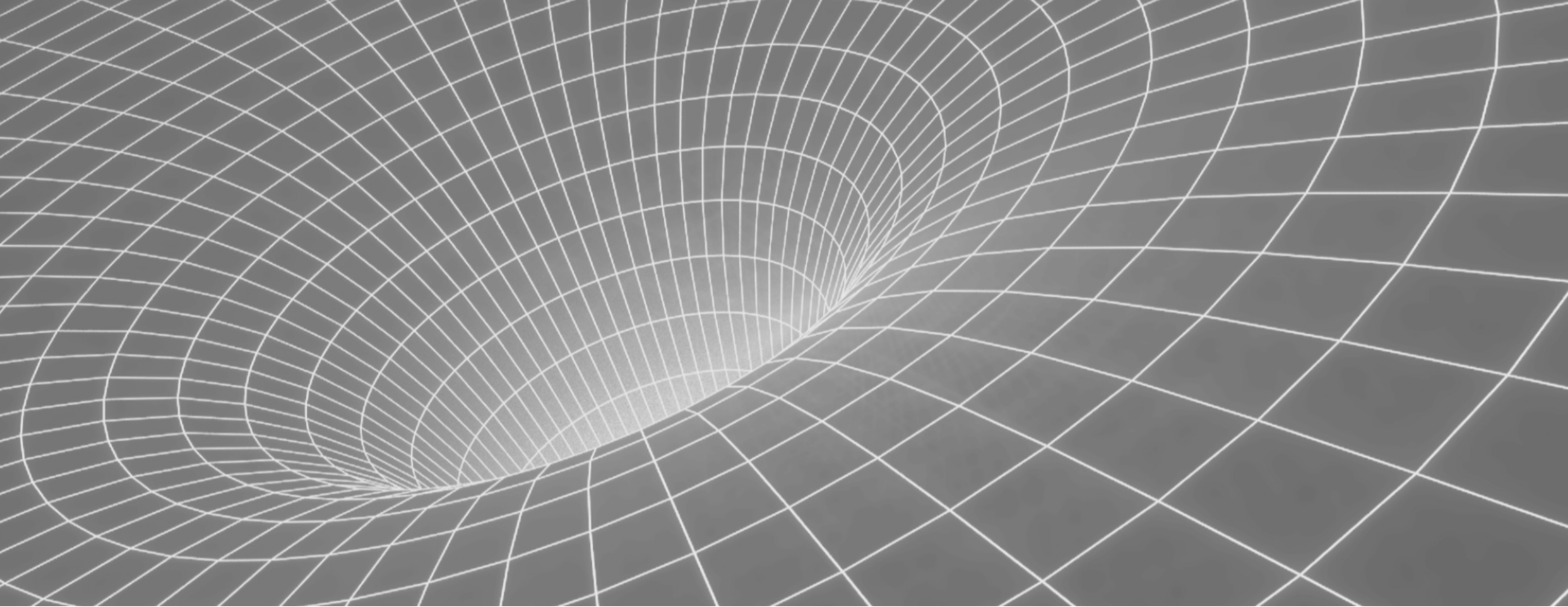
-> 직업과 소득이 상관관계가 커서 소득의 결측치를 그 소득에 많이 분포한 직업의 평균으로 채워줌

-> 실험결과 유의미하지 못했음

느낀점 및 배운점

1. **다른 feature set** & **다른 알고리즘의 모델**을 앙상블할 때 앙상블 효과가 잘 나온다.
2. LB점수와 CV점수를 함께 **기록**하며 과적합을 줄여나가는 방향으로 가는 것이 좋다.
3. 딥러닝의 경우 모델에 맞는 feature set을 찾는 과정이 필요하다.
4. Data leakage를 해결하는 것이 자료의 결측치 발생보다 중요한 데이터였다.
5. 제한된 제출기회 내에서 **계획된 실험**을 해야한다.
6. 점수를 올리고 싶으면 가설을 믿지 말고 **가설을 직접 확인**해야 한다.

!! 모든 것은 해봐야 안다 !!



THANK YOU