**제2회 통신망 안정성 확보를 위한 인공지능 해커톤**

**모델 설명서**

**팀 명 :** 신사임당

**2023년 8월 24일**

1. **라이브러리 및 데이터 (Library & Data)**

* Q1\_train, Q1\_test 파일을 pandas의 read\_csv로 읽음
* 기본적인 연산을 위해 numpy, pandas 호출
* 시계열 데이터의 결측값을 선형 보간법으로 채워주기 위해 scipy 호출
* train/valid 데이터 분리를 위해 from sklearn.model\_selection import train\_test\_split 호출
* scaling을 위해 from sklearn.preprocessing import StandardScaler 호출
* mae 계산을 위해 from sklearn.metrics import mean\_absolute\_error 호출
* XGBoost 모델링을 위해 from xgboost import XGBRegressor 호출
* 모델의 parameter 튜닝을 위해 optuna 호출
* 모델을 pickle 형태로 저장하기 위해 joblib 호출

|  |
| --- |
| import pandas as pd  import numpy as np  from scipy import interpolate  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import mean\_absolute\_error  import xgboost as xgb  from xgboost import XGBRegressor  import optuna  from optuna import Trial, visualization  from optuna.samplers import TPESampler  import joblib  q1\_data = pd.read\_csv('Q1\_train.csv')  q1\_test = pd.read\_csv('Q1\_test.csv', index\_col = 0) |

1. **데이터 전처리 (Data Cleansing & Pre-Processing)**

* scgfail, redirectiontolte\_emergencyfallback, numrar 제거
* 연결 시도한 횟수(att), 연결 성공한 횟수(succ)를 모두 가지고 있는 7개 변수에 대해서 연결 성공 비율(백분율) 변수(ratio)를 새롭게 생성

연결 성공 비율 = (연결 성공한 횟수) / (연결 시도한 횟수) \* 100

7개 변수 : erabaddsucc, endcaddsucc, endcmodbymenbsucc, endcmodbysgnbsucc, connestabsucc, handoversucc, reestabsucc

* 연결 성공한 횟수(succ) 변수들은 삭제
* 연결 시도한 횟수(att)가 0이어서 연결 성공 비율이 nan이 된 값들은 0으로 처리
* Train 데이터에 존재하는 결측값은 선형 보간법으로 채워줌

- (06-07-00:55) & (06-30-00:55) & (07-09-00:55) 시점에서의 BaseStation E, F, G

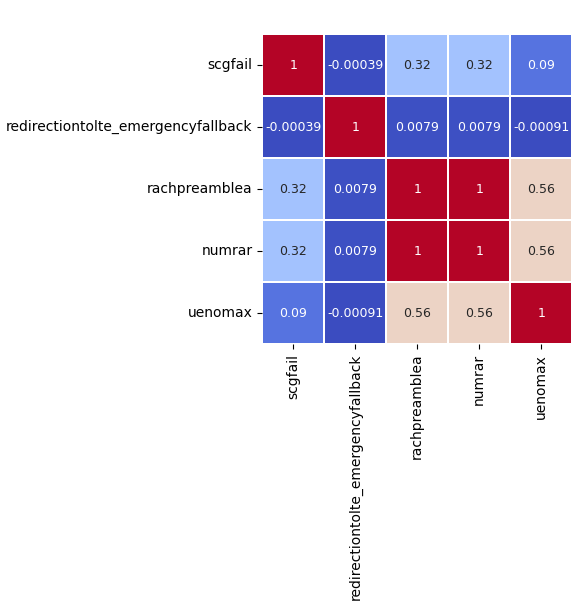
- (06-28 -00:55) 시점에서는 BaseStation E, F, G의 데이터 행 자체가 존재하지 않아서 새로운 행을 생성하여 결측값을 채움

* Int 형태의 변수이지만 보간법으로 인해 float 형태가 된 변수들에 대해서는 반올림을 통해 소수점을 제거해줌

|  |
| --- |
| **# train 전처리**  q1\_data['datetime'] = pd.to\_datetime(q1\_data['datetime'])  # 연결 성공한 횟수 -> 연결 성공 백분율로 변경  q1\_data['erabaddratio'] = (q1\_data['erabaddsucc'] / q1\_data['erabaddatt']) \* 100  q1\_data['endcaddratio'] = (q1\_data['endcaddsucc'] / q1\_data['endcaddatt']) \* 100  q1\_data['endcmodbymenbratio'] = (q1\_data['endcmodbymenbsucc'] / q1\_data['endcmodbymenbatt']) \* 100  q1\_data['endcmodbysgnbratio'] = (q1\_data['endcmodbysgnbsucc'] / q1\_data['endcmodbysgnbatt']) \* 100  q1\_data['connestabratio'] = (q1\_data['connestabsucc'] / q1\_data['connestabatt']) \* 100  q1\_data['handoverratio'] = (q1\_data['handoversucc'] / q1\_data['handoveratt']) \* 100  q1\_data['reestabratio'] = (q1\_data['reestabsucc'] / q1\_data['reestabatt']) \* 100  # 연결 시도한 횟수가 0이어서 성공 백분율이 nan이 된 값들을 0으로 바꿔주기  ratios = ['erabaddratio', 'endcaddratio', 'endcmodbymenbratio', 'endcmodbysgnbratio', 'connestabratio', 'handoverratio', 'reestabratio']  for i in ratios :    q1\_data[i] = q1\_data[i].replace(np.nan, 0)  # succ 변수들 drop  list = ['erabaddsucc', 'endcaddsucc', 'endcmodbymenbsucc', 'endcmodbysgnbsucc', 'connestabsucc', 'handoversucc', 'reestabsucc']  q1\_data = q1\_data.drop(list, axis = 1)  # 순서 재배치 + scgfail, redirectiontolte\_emergencyfallback, numrar drop  col = q1\_data.columns.to\_numpy()  col = col[[0,1,3,4,32,5,33,6,34,7,35,8,36,9,10,12,37,13,38,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,30,31]]  q1\_data = q1\_data[col]  **## 결측치 처리**  # Station별로 분리  StationA = q1\_data[q1\_data['ru\_id'] == 'BaseStationA']  StationC = q1\_data[q1\_data['ru\_id'] == 'BaseStationC']  StationD = q1\_data[q1\_data['ru\_id'] == 'BaseStationD']  StationE = q1\_data[q1\_data['ru\_id'] == 'BaseStationE']  StationF = q1\_data[q1\_data['ru\_id'] == 'BaseStationF']  StationG = q1\_data[q1\_data['ru\_id'] == 'BaseStationG']  StationH = q1\_data[q1\_data['ru\_id'] == 'BaseStationH']  StationI = q1\_data[q1\_data['ru\_id'] == 'BaseStationI']  StationA.reset\_index(drop = True, inplace = True)  StationC.reset\_index(drop = True, inplace = True)  StationD.reset\_index(drop = True, inplace = True)  StationE.reset\_index(drop = True, inplace = True)  StationF.reset\_index(drop = True, inplace = True)  StationG.reset\_index(drop = True, inplace = True)  StationH.reset\_index(drop = True, inplace = True)  StationI.reset\_index(drop = True, inplace = True)  # 선형 보간법으로 채워준 후, int형으로 다시 바꿔주기  missing\_col = ['rlculbyte', 'rlcdlbyte', 'totprbulavg', 'totprbdlavg', 'dlreceivedriavg', 'dltransmittedmcsavg',                 'airmaculbyte', 'airmacdlbyte', 'bler\_ul', 'bler\_dl', 'rachpreamblea', 'nummsg3', 'attpaging',                 'rssipathavg', 'dlreceivedcqiavg']  for i in missing\_col :    StationE[i].interpolate(method = 'linear', inplace = True)    StationF[i].interpolate(method = 'linear', inplace = True)    StationG[i].interpolate(method = 'linear', inplace = True)  int\_list = ['erabaddatt', 'endcaddatt', 'endcmodbymenbatt', 'endcmodbysgnbatt', 'connestabatt', 'redirectiontolte\_coverageout', 'redirectiontolte\_epsfallback', 'handoveratt', 'reestabatt', 'endcrelbymenb', 'uenomax', 'rachpreamblea', 'nummsg3', 'attpaging']  df1 = pd.DataFrame(['2023-06-28 00:55:00', 'BaseStationE'], index = ['datetime', 'ru\_id']).transpose()  df2 = pd.DataFrame((StationE.iloc[11529, 2:] + StationE.iloc[11530, 2:]) / 2).transpose()  E\_new = pd.concat([df1, df2], axis = 1)  E\_new['datetime'] = pd.to\_datetime(E\_new['datetime'])  StationE = pd.concat([StationE.iloc[:11530], E\_new, StationE.iloc[11530:]], ignore\_index = True)  df1 = pd.DataFrame(['2023-06-28 00:55:00', 'BaseStationF'], index = ['datetime', 'ru\_id']).transpose()  df2 = pd.DataFrame((StationF.iloc[11529, 2:] + StationF.iloc[11530, 2:]) / 2).transpose()  F\_new = pd.concat([df1, df2], axis = 1)  F\_new['datetime'] = pd.to\_datetime(F\_new['datetime'])  StationF = pd.concat([StationF.iloc[:11530], F\_new, StationF.iloc[11530:]], ignore\_index = True)  df1 = pd.DataFrame(['2023-06-28 00:55:00', 'BaseStationG'], index = ['datetime', 'ru\_id']).transpose()  df2 = pd.DataFrame((StationG.iloc[11529, 2:] + StationG.iloc[11530, 2:]) / 2).transpose()  G\_new = pd.concat([df1, df2], axis = 1)  G\_new['datetime'] = pd.to\_datetime(G\_new['datetime'])  StationG = pd.concat([StationG.iloc[:11530], G\_new, StationG.iloc[11530:]], ignore\_index = True)  StationE[int\_list] = StationE[int\_list].round(decimals = 0)  StationE[int\_list] = StationE[int\_list].astype(int)  StationF[int\_list] = StationF[int\_list].round(decimals = 0)  StationF[int\_list] = StationF[int\_list].astype(int)  StationG[int\_list] = StationG[int\_list].round(decimals = 0)  StationG[int\_list] = StationG[int\_list].astype(int)  **# test 전처리(train과 동일)**  q1\_test['datetime'] = pd.to\_datetime(q1\_test['datetime'])  # 연결 성공한 횟수 -> 연결 성공 백분율로 변경  q1\_test['erabaddratio'] = (q1\_test['erabaddsucc'] / q1\_test['erabaddatt']) \* 100  q1\_test['endcaddratio'] = (q1\_test['endcaddsucc'] / q1\_test['endcaddatt']) \* 100  q1\_test['endcmodbymenbratio'] = (q1\_test['endcmodbymenbsucc'] / q1\_test['endcmodbymenbatt']) \* 100  q1\_test['endcmodbysgnbratio'] = (q1\_test['endcmodbysgnbsucc'] / q1\_test['endcmodbysgnbatt']) \* 100  q1\_test['connestabratio'] = (q1\_test['connestabsucc'] / q1\_test['connestabatt']) \* 100  q1\_test['handoverratio'] = (q1\_test['handoversucc'] / q1\_test['handoveratt']) \* 100  q1\_test['reestabratio'] = (q1\_test['reestabsucc'] / q1\_test['reestabatt']) \* 100  # 연결 시도한 횟수가 0이어서 성공 백분율이 nan이 된 값들을 0으로 바꿔주기  for i in ratios :    q1\_test[i] = q1\_test[i].replace(np.nan, 0)  # succ 변수들 drop  q1\_test = q1\_test.drop(list, axis = 1)  # 순서 재배치 + scgfail, redirectiontolte\_emergencyfallback, numrar drop  col = q1\_test.columns.to\_numpy()  col = col[[0,1,3,4,31,5,32,6,33,7,34,8,35,9,10,12,36,13,37,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,30]]  q1\_test = q1\_test[col]  **# 전처리 완료한 데이터**  train\_df = pd.concat([StationA, StationC, StationD, StationE, StationF, StationG, StationH, StationI])  train\_df.reset\_index(drop = True, inplace = True)  test\_df = q1\_test |

1. **탐색적 자료 분석 (Exploratory Data Analysis)**

* 예측 대상인 uenomax와의 redirectiontolte\_emergencyfallback의 상관계수가 0에 가까워, 예측에 영향을 미치지 않는다고 판단하여 제거함
* - scgfail 은 실패 횟수이므로 실패 백분율(scgfailratio)를 남기고 해당변수는 제거
* rachpreamblea와 numrar의 상관계수가 1이고, 실제로 데이터를 살펴보았을 때 train의 모든 데이터에 대해서 두 변수가 완전히 동일한 값을 가지고 있었으므로 두 변수 중 하나만 사용해도 무관하겠다고 판단하여numrar은 제거함



* 연결 성공한 횟수(succ) 와 연결 시도한 횟수 (att)는 수치가 매우 비슷하므로 두 변수를 같이 모델에 넣는 것은 다중공선성을 일으킬 수 있다고 생각함.

따라서, 연결 성공한 횟수보다는 기존에 연결 실패한 백분율 변수가 있음에 착안하여 연결 성공한 백분율 변수를 새로 생성하는 것이 바람직할 것이라고 판단함.

* 시계열 데이터에서 결측치를 채울 때 여러 가지 방법이 존재하는데(이전, 이후, 평균 보간법 등), 본 데이터에선 결측치 행 기준으로 앞뒤 시점을 확인했을 때 값 차이가 꽤 존재하여 앞뒤 시점의 평균값을 사용하는 선형 보간법으로 값을 채움.
* 시계열 데이터를 일반 regression 문제로 변환하기 위해 시간 관련 변수 추가함
  + 기존에 존재하던 datetime에서 hour를 뽑으면 23시와 0시 사이에 큰 갭차이가 존재하기 때문에 시간 순환성을 위해 cos, sin함수를 적용함
  + 무선 네트워크 관점에서 주말과 공휴일에 사용량이 급증할 것으로 예상되므로 파생변수를 만들 필요가 있다고 생각됨

|  |  |
| --- | --- |
| 주말 및 공휴일이 아닐때 단말기 사용량 | 주말 및 공휴일일 때 단말기 사용량 |
|  |  |

1. **변수 선택 및 모델 구축 (Feature Engineering & Initial Modeling)**

* pycaret을 통해 분야 1의 데이터에서 우수한 성능을 보이는 XGBoost와 LightGBM, Catboost를 비교해보고, 가장 좋은 성능을 보였던 XGBoost를 사용하여 모델을 구축하기로 결정

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

[ 모델 1 ] 전처리를 마친 데이터에 변수 추가 없이 모델링

* datetime, ru\_id 변수 제거
* train : valid = 7 : 3 으로 분할 후 Standard scaling 진행

[ 모델 2 ] 전처리를 마친 데이터에 시간, 공휴일 여부 변수 추가하여 모델링

* datetime을 이용하여 hour(시간), day(요일) 변수 생성
* hour 변수의 경우, 0시와 23시는 가까운 시간대이지만 숫자 상으로는 멀게 받아들여질 수 있어, sin, cos 변환을 통해 시간의 순환적 성격을 반영하기 위해 sin\_time, cos\_time 변수 생성
* day 변수를 활용하여 공휴일 여부를 나타내는 holiday 변수 생성
* datetime, ru\_id, hour, day 변수 제거
* train : valid = 7 : 3 으로 분할 후 Standard scaling 진행

[ 모델 3 ] 전처리를 마친 데이터에 시간, 공휴일 여부 변수 추가 후, 공휴일 여부 별로 나누어서 모델링

* datetime을 이용하여 hour(시간), day(요일) 변수 생성
* hour 변수의 경우, 0시와 23시는 가까운 시간대이지만 숫자 상으로는 멀게 받아들여질 수 있어, sin, cos 변환을 통해 시간의 순환적 성격을 반영하기 위해 sin\_time, cos\_time 변수 생성
* day 변수를 활용하여 공휴일 여부를 나타내는 holiday 변수 생성
* datetime, ru\_id, hour, day 변수 제거
* holiday = 1인 경우와 holiday = 0인 경우에 대해 각각 모델링 진행
* holiday별로 각각 train : valid = 7 : 3으로 분할 후 Standard scaling 진행

|  |
| --- |
| **# 모델 1.**  # datetime과 ru\_id 제거  data = train\_df.iloc[:, 2:]  # X와 y로 분할  X, y = data.iloc[:, :-1], data.iloc[:, [-1]]  # train : valid = 7 : 3 분할  X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size = 0.3, shuffle = False, random\_state = 30)  print(X\_train.shape, X\_valid.shape)  # scaling  ss = StandardScaler()  X\_train\_scale = ss.fit\_transform(X\_train)  X\_valid\_scale = ss.transform(X\_valid)  **# 모델 2.**  train\_df2 = train\_df.copy()  test\_df2 = test\_df.copy()  # hour, day 추가  date = pd.to\_datetime(train\_df2.datetime)  train\_df2['hour'] = date.dt.hour  train\_df2['day'] = date.dt.weekday  date2 = pd.to\_datetime(test\_df2.datetime)  test\_df2['hour'] = date2.dt.hour  test\_df2['day'] = date2.dt.weekday  # 공휴일 여부(공휴일이면 1, 아니면 0)  ## 주말  train\_df2['holiday'] = train\_df2['day'].apply(lambda x: 1 if x >= 5 else 0)  test\_df2['holiday'] = test\_df2['day'].apply(lambda x: 1 if x >= 5 else 0)  ## 공휴일(5/29, 6,6)  train\_df2[('2023-05-29' <= train\_df2.datetime) & (train\_df2.datetime < '2023-05-30')]['holiday'] = 1  train\_df2[('2023-06-06' <= train\_df2.datetime) & (train\_df2.datetime < '2023-06-07')]['holiday'] = 1  test\_df2[('2023-05-29' <= test\_df2.datetime) & (test\_df2.datetime < '2023-05-30')]['holiday'] = 1  test\_df2[('2023-06-06' <= test\_df2.datetime) & (test\_df2.datetime < '2023-06-07')]['holiday'] = 1  # 0시와 23시는 가까운 시간대이지만, 숫자 상으로는 멀게 받아들여질 수 있음  # 따라서 sin, cos 변환을 하여 시간의 순환적 성격을 반영하기로 함  train\_df2['cos\_time'] = np.cos(2\*np.pi\*(train\_df2.hour/24))  train\_df2['sin\_time'] = np.sin(2\*np.pi\*(train\_df2.hour/24))  test\_df2['cos\_time'] = np.cos(2\*np.pi\*(test\_df2.hour/24))  test\_df2['sin\_time'] = np.sin(2\*np.pi\*(test\_df2.hour/24))  # hour, day 변수 제거  train\_df2.drop(['hour', 'day'], axis = 1, inplace = True)  test\_df2.drop(['hour', 'day'], axis = 1, inplace = True)  # 변수 순서 바꿔주기  train\_df2 = train\_df2[train\_df2.columns[:35].to\_list() + train\_df2.columns[36:].to\_list() + train\_df2.columns[35:36].to\_list()]  # X와 y로 분할  X, y = train\_df2.iloc[:, 2:-1], train\_df2.iloc[:, [-1]]  # train : valid = 7 : 3 분할  X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size = 0.3, shuffle = False, random\_state = 30)  print(X\_train.shape, X\_valid.shape)  # scaling  ss2 = StandardScaler()  not\_holiday = [col for col in X\_train.columns if col not in ['hoilday']]  X\_train\_scale = ss2.fit\_transform(X\_train[not\_holiday])  X\_valid\_scale = ss2.transform(X\_valid[not\_holiday])  print(X\_train\_scale.shape, X\_valid\_scale.shape)  **# 모델 3. 모델 2의 train\_df2, test\_df2에서부터 시작**  StationB = test\_df2[test\_df2['ru\_id'] == 'BaseStationB']  StationJ = test\_df2[test\_df2['ru\_id'] == 'BaseStationJ']  StationB.reset\_index(drop = True, inplace = True)  StationJ.reset\_index(drop = True, inplace = True)  train\_h = train\_df2[train\_df2['holiday'] == 1]  train\_n = train\_df2[train\_df2['holiday'] == 0]  # holiday 여부로 데이터 분리  B\_h = StationB[StationB['holiday'] == 1]  B\_n = StationB[StationB['holiday'] == 0]  J\_h = StationJ[StationJ['holiday'] == 1]  J\_n = StationJ[StationJ['holiday'] == 0]  **## holiday = 1**  data\_h = train\_h.iloc[:, 2:]  X\_h, y\_h = data\_h.iloc[:, :-1], data\_h.iloc[:, [-1]]  # train : valid = 7 : 3 분할  X\_h\_train, X\_h\_valid, y\_h\_train, y\_h\_valid = train\_test\_split(X\_h, y\_h, test\_size = 0.3, shuffle = False, random\_state = 30)  print(X\_h\_train.shape, X\_h\_valid.shape)  # scaling  ss3 = StandardScaler()  X\_h\_train\_scale = X\_h\_train.drop(['holiday'], axis = 1)  X\_h\_valid\_scale = X\_h\_valid.drop(['holiday'], axis = 1)  X\_h\_train\_scale = ss3.fit\_transform(X\_h\_train\_scale)  X\_h\_valid\_scale = ss3.transform(X\_h\_valid\_scale)  print(X\_h\_train\_scale.shape, X\_h\_valid\_scale.shape)  **## holiday = 0**  data\_n = train\_n.iloc[:, 2:]  X\_n, y\_n = data\_n.iloc[:, :-1], data\_n.iloc[:, [-1]]  # train : valid = 7 : 3 분할  X\_n\_train, X\_n\_valid, y\_n\_train, y\_n\_valid = train\_test\_split(X\_n, y\_n, test\_size = 0.3, shuffle = False, random\_state = 30)  print(X\_n\_train.shape, X\_n\_valid.shape)  # scaling  ss4 = StandardScaler()  X\_n\_train\_scale = X\_n\_train.drop(['holiday'], axis = 1)  X\_n\_valid\_scale = X\_n\_valid.drop(['holiday'], axis = 1)  X\_n\_train\_scale = ss4.fit\_transform(X\_n\_train\_scale)  X\_n\_valid\_scale = ss4.transform(X\_n\_valid\_scale)  print(X\_n\_train\_scale.shape, X\_n\_valid\_scale.shape) |

1. **모델 학습 및 검증 (Model Tuning & Evaluation)**

* optuna를 통해 valid 데이터에서의 mae를 가장 작게 하는 parameter 값 도출
* optuna 튜닝 및 모델 학습 시 gpu 사용
* train 데이터로 모델을 학습시킨 후, valid 데이터로 검증을 진행함
* 모델 재현을 위해 XGBRegressor 사용 시 random\_state = 1로 고정
* 분야 1에서 예측해야 하는 uenomax는 사용자 단말 개수의 최댓값이므로 실수형으로 나오는 모델의 예측값을 반올림하여 정수형으로 변환하여 mae 계산

[ 모델 1 ]

* optuna로 parameter 튜닝 시 시도 횟수는 100번으로 함
* 학습에 사용한 parameter

{'n\_estimators': 1492, 'max\_depth': 15, 'min\_child\_weight': 2, 'gamma': 0.4, 'learning\_rate': 0.0038565758063650217, 'colsample\_bytree': 0.7, 'subsample': 0.8, 'lambda': 4.076587859420142, 'alpha': 0.5270409349856588}

* validation 데이터로 계산한 mae(반올림하기 전) : 0.4954013516641977
* validation 데이터로 계산한 mae(반올림 후) : 0.4384867224445253

[ 모델 2 ]

* optuna로 parameter 튜닝 시 시도 횟수는 50번으로 함
* 학습에 사용한 parameter

{'n\_estimators': 3015, 'max\_depth': 19, 'min\_child\_weight': 1, 'gamma': 0.02, \_rate': 0.014374633885619992, 'colsample\_bytree': 0.7, 'subsample': 0.9, 'lambda': 4.673092181226084, 'alpha': 0.006110253574773731}

* validation 데이터로 계산한 mae(반올림하기 전) : 0.49829781669686307
* validation 데이터로 계산한 mae(반올림 후) : 0.4417363889899357

[ 모델 3 ]

**- holiday = 1 모델**

* optuna로 parameter 튜닝 시 시도 횟수는 50번으로 함
* 학습에 사용한 parameter

{'n\_estimators': 2814, 'max\_depth': 19, 'min\_child\_weight': 5, 'gamma': 0.09, 'learning\_rate': 0.010568097997401979, 'colsample\_bytree': 0.8, 'subsample': 0.7, 'lambda': 5.94489114614049, 'alpha': 0.19738241566039705}

* validation 데이터로 계산한 mae(반올림하기 전) : 0.5039922573539765
* validation 데이터로 계산한 mae(반올림 후) : 0.4475968493811284

**- holiday = 0 모델**

* optuna로 parameter 튜닝 시 시도 횟수는 50번으로 함
* 학습에 사용한 parameter

{'n\_estimators': 2944, 'max\_depth': 35, 'min\_child\_weight': 5, 'gamma': 0.42000000000000004, 'learning\_rate': 0.009641077051022224, 'colsample\_bytree': 1.0, 'subsample': 0.8, 'lambda': 2.5880158011337313, 'alpha': 2.336338346210088}

* validation 데이터로 계산한 mae(반올림하기 전) : 0.49835927376579703
* validation 데이터로 계산한 mae(반올림 후) : 0.44139200500121556

|  |
| --- |
| **# 모델 1.**  params = {'n\_estimators': 1492, 'max\_depth': 15, 'min\_child\_weight': 2, 'gamma': 0.4,            'learning\_rate': 0.0038565758063650217, 'colsample\_bytree': 0.7, 'subsample': 0.8,            'lambda': 4.076587859420142, 'alpha': 0.5270409349856588}  xgboost = XGBRegressor(random\_state = 1, tree\_method = 'gpu\_hist',                          \*\*params)  xgboost.fit(X\_train\_scale, y\_train,              eval\_metric = 'mae', eval\_set = [(X\_valid\_scale, y\_valid)], early\_stopping\_rounds = 100,              verbose = 100)  # validation set에 대한 예측값으로 계산한 mae  y\_pred = xgboost.predict(X\_valid\_scale)  valid\_mae = mean\_absolute\_error(y\_valid, y\_pred)  print('valid set에서의 mae :', valid\_mae)  # 예측값을 반올림 한 뒤의 mae  y\_pred = xgboost.predict(X\_valid\_scale)  y\_pred\_round = y\_pred.round(0)  valid\_mae2 = mean\_absolute\_error(y\_valid, y\_pred\_round)  print('valid set에서의 mae(반올림 한 뒤) :', valid\_mae2)  # 모델 저장  joblib.dump(xgboost, 'model\_1.pkl')  **# 모델 2.**  params2 = {'n\_estimators': 3015, 'max\_depth': 19, 'min\_child\_weight': 1, 'gamma': 0.02,             'learning\_rate': 0.014374633885619992, 'colsample\_bytree': 0.7, 'subsample': 0.9,             'lambda': 4.673092181226084, 'alpha': 0.006110253574773731}  xgboost2 = XGBRegressor(random\_state = 1, tree\_method = 'gpu\_hist',                          \*\*params2)  xgboost2.fit(X\_train\_scale, y\_train,               eval\_metric = 'mae', eval\_set = [(X\_valid\_scale, y\_valid)], early\_stopping\_rounds = 100,               verbose = 100)  # validation set에 대한 예측값으로 계산한 mae  y\_pred = xgboost2.predict(X\_valid\_scale)  valid\_mae = mean\_absolute\_error(y\_valid, y\_pred)  print('valid set에서의 mae :', valid\_mae)  # 예측값을 반올림 한 뒤의 mae  y\_pred = xgboost2.predict(X\_valid\_scale)  y\_pred\_round = y\_pred.round(0)  valid\_mae2 = mean\_absolute\_error(y\_valid, y\_pred\_round)  print('valid set에서의 mae(반올림 한 뒤) :', valid\_mae2)  # 모델 저장  joblib.dump(xgboost2, 'model\_2.pkl')  **# 모델 3.**  **## holiday = 1**  params3 = {'n\_estimators': 2814, 'max\_depth': 19, 'min\_child\_weight': 5, 'gamma': 0.09,             'learning\_rate': 0.010568097997401979, 'colsample\_bytree': 0.8, 'subsample': 0.7,             'lambda': 5.94489114614049, 'alpha': 0.19738241566039705}  xgboost3 = XGBRegressor(random\_state = 1, tree\_method = 'gpu\_hist',                          \*\*params3)  xgboost3.fit(X\_h\_train\_scale, y\_h\_train,               eval\_metric = 'mae', eval\_set = [(X\_h\_valid\_scale, y\_h\_valid)], early\_stopping\_rounds = 100,               verbose = 100)  # validation set에 대한 예측값으로 계산한 mae  y\_pred = xgboost3.predict(X\_h\_valid\_scale)  valid\_mae = mean\_absolute\_error(y\_h\_valid, y\_pred)  print('valid set에서의 mae :', valid\_mae)  # 예측값을 반올림 한 뒤의 mae  y\_pred = xgboost3.predict(X\_h\_valid\_scale)  y\_pred\_round = y\_pred.round(0)  valid\_mae2 = mean\_absolute\_error(y\_h\_valid, y\_pred\_round)  print('valid set에서의 mae(반올림 한 뒤) :', valid\_mae2)  # 모델 저장  joblib.dump(xgboost3, 'model\_3.pkl')  **## holiday = 0**  params4 = {'n\_estimators': 2944, 'max\_depth': 35, 'min\_child\_weight': 5, 'gamma': 0.42000000000000004,             'learning\_rate': 0.009641077051022224, 'colsample\_bytree': 1.0, 'subsample': 0.8,             'lambda': 2.5880158011337313, 'alpha': 2.336338346210088}  xgboost4 = XGBRegressor(random\_state = 1, tree\_method = 'gpu\_hist',                          \*\*params4)  xgboost4.fit(X\_n\_train\_scale, y\_n\_train,               eval\_metric = 'mae', eval\_set = [(X\_n\_valid\_scale, y\_n\_valid)], early\_stopping\_rounds = 100,               verbose = 100)  # validation set에 대한 예측값으로 계산한 mae  y\_pred = xgboost4.predict(X\_n\_valid\_scale)  valid\_mae = mean\_absolute\_error(y\_n\_valid, y\_pred)  print('valid set에서의 mae :', valid\_mae)  # 예측값을 반올림 한 뒤의 mae  y\_pred = xgboost4.predict(X\_n\_valid\_scale)  y\_pred\_round = y\_pred.round(0)  valid\_mae2 = mean\_absolute\_error(y\_n\_valid, y\_pred\_round)  print('valid set에서의 mae(반올림 한 뒤) :', valid\_mae2)  # 모델 저장  joblib.dump(xgboost4, 'model\_4.pkl') |

1. **결과 및 결언 (Conclusion & Discussion)**

* 인터넷 사용량이므로 주말 및 공휴일 변수가 중요할 것이라고 판단했지만 기대한 것보다 mae 변화가 미미했음. 따라서 이부분을 좀 더 세분화하여 점심/퇴근시간 혹은 요일별 등으로 나눠보는 것도 유의미한 접근이 될 수 있음
* 본 분석에서는 예측 값을 일반적인0.5 기준으로 반올림하였는데 더 적합한 반올림 기준 값을 찾아볼 필요가 있음
* Xgboost 이외에 다른 모델로 예측한 값도 최종 voting에 포함시켜볼 필요가 있음