2021 Dacon Competition

- 대회명: **구내식당 식수 인원 예측 AI 경진대회**
- 대회 링크 : https://dacon.io/competitions/official/235743/overview/description
- 참여 기간: 2021/06/22 ~ 2021/07/23
- 참여자: 김민찬, 위홍복, 정구홍
- 대회 배경:

> "지금까지는 **단순한 시계열 추세**와 **담당자의 직관적 경험**에 의존하여 한국토지주택공사 구내식당 식수 인원을 예측하였으나, **빅데이터 분석으로 얻어지는 보다 정확도 높은 예측**을 통해 잔반 발생량을 획기적으로 줄이고자 합니다."

- 평가 산식: MAE(Mean Absolute Error)
- 사용 가능 언어: Python, R

1. 데이터 수집

A. 제공 데이터

▼ train.csv

- 일자
- 요일
- 본사정원수
- 본사휴가자수 / 본사출장자수
- 시간외근무명령서승인건수
- 현본사소속재택근무자수
- 조식메뉴 / 중식메뉴 / 석식메뉴
- **중식계 / 석식계 → Label Data**

▼ test.csv

- 일자
- 요일
- 본사정원수
- 본사휴가자수 / 본사출장자수
- 시간외근무명령서승인건수
- 현본사소속재택근무자수
- 조식메뉴 / 중식메뉴 / 석식메뉴

B. 사용한 외부 데이터

▼ 전일 / 익일 휴일 데이터

- 출처: 공공데이터포털
- 데이터명: 한국천문연구원_특일 정보
- 내용:

  기존 연구 참조 및 활용

  - (전종식 외 2명 "Predicting the Number of People for Meals of an Institutional Foodservice by Applying Machine Learning Methods: S City Hall Case")

    1. API 활용 공휴일 정보 조회 및 획득
    2. Feature 데이터에 '익일휴일여부', '전일휴일여부' 컬럼 추가

▼ 기상 데이터

- 출처: 기상청 기상자료개방
- 데이터명: 종관기상관측(ASOS) - 파일셋
- 내용:
  - 팀 내 아이디에이션 후 활용

    1. LH 본사 위치가 진주시이므로, 진주시 기상데이터 활용
    2. 중식 및 석식 외출 시간대 기상 자료 활용

       a. 중식: 11:00 ~ 12:00, 석식: 17:00 ~ 18:00

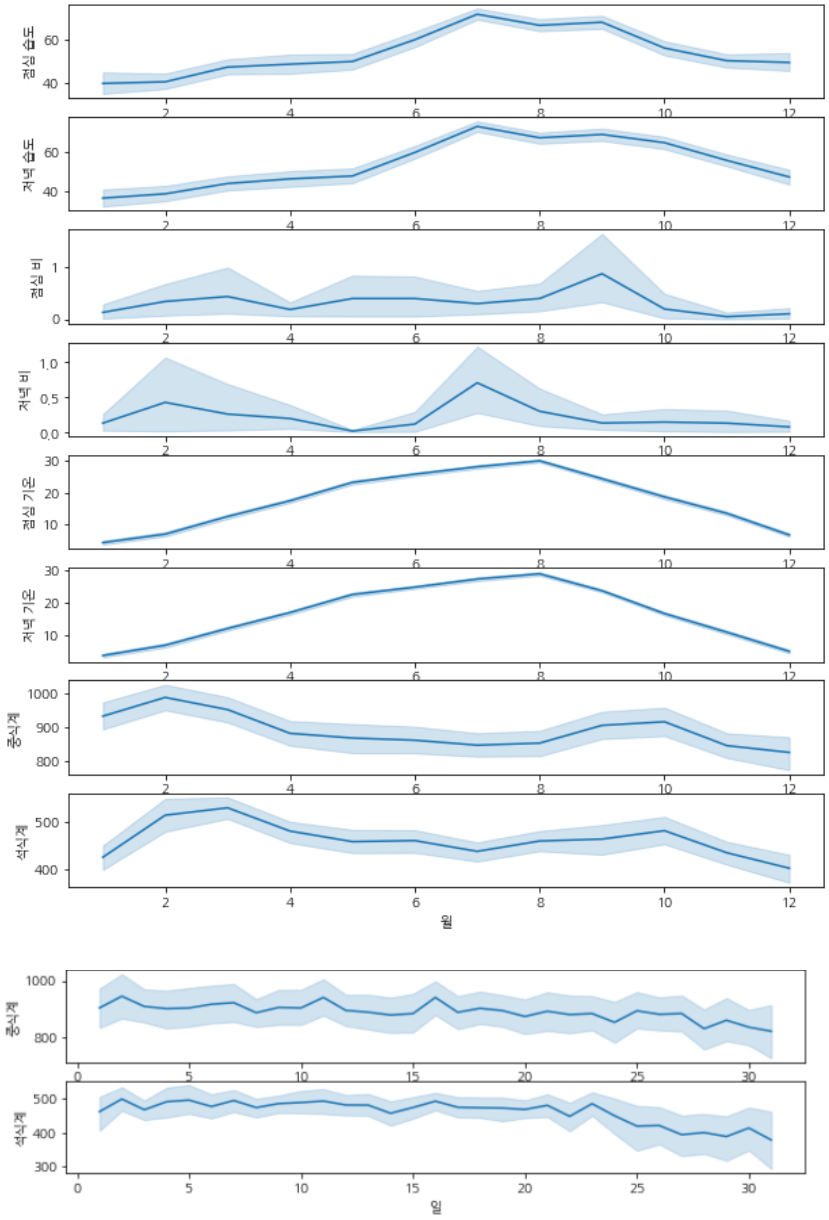
       Feature 데이터에 중식 / 석식별 '강수량', '기온', '습도' 추가

▼ 시도하려 했으나, 미사용 데이터

1. 코로나 확진자 수 데이터 및 거리두기 단계 데이터
   a. 데이콘 커뮤니티에서 유의미한 영향 없는 것으로 판별되어 미사용
2. 전주동일요일중식계(1주일 전 중식계)
   a. 간접적인 시계열 분석 효과를 위해 사용하려 했으나,
   테스트 데이터를 학습에 활용하는 Data Leakage 예상되어 미사용
3. 식단 메뉴 데이터
   a. 前) 메뉴에서 명사 단위 토큰화 / 사전 생성 후, 원-핫 인코딩으로 사용 → 예측 성능 하락으로 제외

2. 데이터 전처리

A. Exploratory Data Analysis (EDA)



- 논문을 기반으로, 요일별 데이터 활용

  → 요일을 원-핫 인고딩으로 피처 데이터 추가

- 사후검토:

  1. 월/일 평균 중식/석식계 데이터에서 유의미한 차이가 발견 되어서, '월' 데이터를 추가하는 방향으로 할 수 있었음
  2. EDA를 체계적으로 활용했으면 더 좋은 결과 기대 가능

B. Data Preprocessing

- 2016 ~ 2020년 기상데이터는 시단위 데이터, 2021년 이후는 분단위데이터였기 때문에, 전처리 과정 중 하나의 함수로 처리함에 어려움이 있어 다른 방법으로 처리
- 2016 ~ 2020년 기상 데이터 가공 (시간 단위)

```
#2016~2020 기상 데이터 가공

df_train = pd.read_csv('/content/drive/MyDrive/Dacon/data/train.csv', index_col=False, encoding = 'cp949')
```

```
df_test = pd.read_csv('/content/drive/MyDrive/Dacon/data/test.csv', index_col=False, encoding = 'cp949')

def create_weather_col(df, test_df=False): # 기상데이터 저장할 컬럼 초기화, 선언.
    if test_df == False:
        df.drop(['조식메뉴'], axis='columns', inplace=True)
    df['점심 비'] = [0.0 for i in range(len(df))]
    df['저녁 비'] = [0.0 for i in range(len(df))]
    df['점심 습도'] = [0.0 for i in range(len(df))]
    df['저녁 습도'] = [0.0 for i in range(len(df))]
    df['점심 기온'] = [0.0 for i in range(len(df))]
    df['저녁 기온'] = [0.0 for i in range(len(df))]

    return df

df_train = create_weather_col(df_train)
df_test = create_weather_col(df_test, test_df=True)

def unuse_day_del(df, w_df):
    # df에 있는 날짜 기반으로 w_df에서 필요한 날의 시간대 데이터만 남김.
    day_list = []
    for i in df['일자']:
        day_list.append(i)

    use_time = ['11:00', '12:00', '17:00', '18:00']

    for i in range(len(w_df)-1, -1, -1):
        if w_df['일시'].iloc[i][:10] not in day_list:
            w_df.drop(i, inplace=True)
        else:
            if w_df['일시'].iloc[i][-5:] not in use_time:
                w_df.drop(i, inplace=True)
    return w_df

df_weather = unuse_day_del(df_train, df_weather)
df_weather.info()

df_weather.reset_index(drop=False, inplace=True)# 인덱스 초기화(필요없는 데이터 삭제후 인덱스가 연속적이지 않다.)


# 결측치(NaN)를 0으로 대체
import math
def nan_to_zero(df): # 현재 사용할 컬럼인 강수량, 기온, 습도는 NaN인 데이터가 많다. 사용하기 위해 0으로 변환한다.
    if math.isnan(df) == True:
        return float(0)
    else :
        return df
df_weather['강수량(mm)'] = df_weather['강수량(mm)'].apply(nan_to_zero) # 21년 데이터의 경우 '누적강수량(mm)' 이다.
df_weather['기온(°C)'] = df_weather['기온(°C)'].apply(nan_to_zero)
df_weather['습도(%)'] = df_weather['습도(%)'].apply(nan_to_zero)
df_weather.info()


# 기상 데이터 컬럼 생성
def insert_wheather_col_item(df, w_df, df_col_name, w_df_col_name,):
    lunch_time = ['11:00', '12:00']
    dinner_time = ['17:00', '18:00']

    lu_dict = {}
    di_dict = {}

    for i in range(len(w_df)):
        if w_df['일시'].iloc[i][-5:] in lunch_time:
            if df[df['일자'] == w_df['일시'].iloc[i][:10]][df_col_name[0]].values[0] < w_df[w_df_col_name].iloc[i]:
                lu_dict[w_df['일시'].iloc[i][:10]] = w_df[w_df_col_name].iloc[i]
        elif w_df['일시'].iloc[i][-5:] in dinner_time:
            if df[df['일자'] == w_df['일시'].iloc[i][:10]][df_col_name[1]].values[0] < w_df[w_df_col_name].iloc[i]:
                di_dict[w_df['일시'].iloc[i][:10]] = w_df[w_df_col_name].iloc[i]

    for i in range(len(df)):
        if df['일자'].iloc[i] in lu_dict:
            df[df_col_name[0]].iloc[i] = lu_dict[df['일자'].iloc[i]]
        if df['일자'].iloc[i] in di_dict:
            df[df_col_name[1]].iloc[i] = di_dict[df['일자'].iloc[i]]

    return df


df_col_list = [['점심 비', '저녁 비'], ['점심 습도', '저녁 습도'], ['점심 기온', '저녁 기온']]
weather_col_list = ['강수량(mm)', '습도(%)', '기온(°C)']
for df_item, weather_item in zip(df_col_list, weather_col_list):
```

```
    df_train = insert_wheather_col_item(df_train, df_weather, df_item, weather_item)
df_train.info()
```

- 2021년 기상 데이터 가공
  - 2016 ~ 2020년 데이터와 '컬럼명', '기준 시간(분)'이 달라서 다른 방식의 전처리가 필요

```
# 실행시간이 약 19분 정도 걸린다.
# 동작이 끝난뒤 저장을 해준다.
use_time = ['11:00', '12:00', '17:00', '18:00']

for i in range(len(df_2021_weather)-1, -1, -1):
    if df_2021_weather['일시'].iloc[i][-5:] not in use_time:
        df_2021_weather.drop(i, inplace=True)
    elif df_2021_weather['일시'].iloc[i][-2:] != '00':
        df_2021_weather.drop(i, inplace=True)

# 인덱스 리셋
df_2021_weather.reset_index(drop=False, inplace=True)

# 결측치 처리
df_2021_weather['누적강수량(mm)'] = df_2021_weather['누적강수량(mm)'].apply(nan_to_zero)
df_2021_weather['기온(°C)'] = df_2021_weather['기온(°C)'].apply(nan_to_zero)

#저장
df_2021_weather.to_csv('/content/drive/MyDrive/Dacon/weather/2021_weather.csv',index=False, encoding='cp949')

# 인덱스 리셋 train 채울 데이터
use_train_2021_weather = unuse_day_del(df_train, df_2021_weather)
use_train_2021_weather.reset_index(drop=False, inplace=True)

# df_train.iloc[1188:] 확인하면  21년 기상 데이터가 존재하지 않는다. 위에서 전처리한 21년 기상데이터를 이용해 채워준다.
df_col_list = [['점심 비', '저녁 비'], ['점심 습도', '저녁 습도'], ['점심 기온', '저녁 기온']]
weather_col_list = ['누적강수량(mm)', '습도(%)', '기온(°C)']

for df_item, weather_item in zip(df_col_list, weather_col_list):
    df_train = insert_wheather_col_item(df_train, use_train_2021_weather, df_item, weather_item)


# 같은 방식으로 df_test도 기상데이터를 채워준다.

# 인덱스 리셋 test 채울 데이터
use_test_2021_weather = unuse_day_del(df_test, df_2021_weather)
use_test_2021_weather.reset_index(drop=False, inplace=True)

for df_item, weather_item in zip(df_col_list, weather_col_list):
    df_test = insert_wheather_col_item(df_test, df_2021_weather, df_item, weather_item)
df_test.info()
```

  - 전일/익일 휴일 여부

```
from datetime import datetime

def holidays_check(df, holidays): # 국자지정 공휴일 정보 반영.
    df['일자'] = df['일자'].apply(lambda x: pd.to_datetime(x))

    tomm_holi = [0 for i in range(len(df))]
    yest_holi = [0 for i in range(len(df))]
    for i in range(len(df)):
        if df.loc[i]['일자'] + pd.DateOffset(days= 1) in list(holidays['date']):
            tomm_holi[i] = 1
        if df.loc[i]['일자'] - pd.DateOffset(days= 1) in list(holidays['date']):
            yest_holi[i] = 1
    df['익일휴일여부'] = tomm_holi
    df['전일휴일여부'] = yest_holi

    return df

holidays['date'] = holidays['locdate'].apply(lambda x: pd.to_datetime(str(x)[:4] + '-' + str(x)[4:6]+ '-'+ str(x)[6:]))
holidays.drop(['locdate'], axis = 1)
```

```
#holidays = 공휴일 정보 데이터
train = holidays_check(train, holidays)
test = holidays_check(test, holidays)
```

- 요일 데이터 정수화 및 '월', '금' 에 전일/익일 휴일 여부 반영

```
def day_to_num(d):
    if d == '월':
        return 1
    elif d == '화':
        return 2
    elif d == '수':
        return 3
    elif d == '목':
        return 4
    elif d == '금':
        return 5

weekday = {
    '월': 1,
    '화': 2,
    '수': 3,
    '목': 4,
    '금': 5
}

train['요일'] = train['요일'].apply(day_to_num)
test['요일'] = test['요일'].map(weekday)

def holidays_check_2(df): # 요일 기준 휴일정보 반영
    for i in range(len(df)):
        if df['요일'].iloc[i] == 1:
            df['전일휴일여부'].iloc[i] = 1

        if df['요일'].iloc[i] == 5:
            df['익일휴일여부'].iloc[i] = 1
    return df

train = holidays_check_2(train)
test = holidays_check_2(test)
```

- 요일 데이터 원-핫 인코딩

```
def day_one_hot_encoding(df):
    day_list = ['월요일', '화요일', '수요일', '목요일', '금요일']
    df[day_list] = 0
    for i in range(len(df)):
        for idx, day in enumerate(day_list):
            if df['요일'].iloc[i] == idx+1:
                df[day].iloc[i] = 1
            else :
                df[day].iloc[i] = 0
    return df

train = day_one_hot_encoding(train)
test = day_one_hot_encoding(test)
```

3. Machine Learning

Models

▼ 단일 모델

- RandomForest
- CatBoost

- XGBoost

▼ 앙상블 모델

▼ Voting Regressor (Soft Voting)

- CatBoost
- XGBoost
- LightGBM

→ **앙상블 사용 후 성능의 급격한 상승**

Hyperparameter Tuning

- GridSearchCV
- RandomizedSearchCV

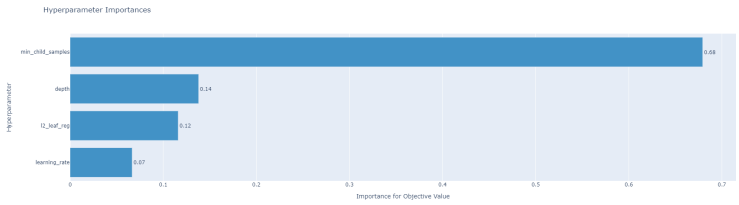
▼ Optuna (GitHub link)

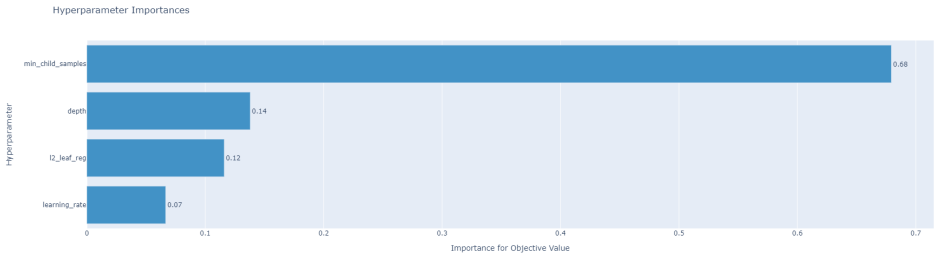
▼ 참고한 링크

- Catboost Hyperparameter tuning with Optuna
- XGBoost Hyperparameter tuning with Optuna
- LightGBM Hyperparameter tuning with Optuna

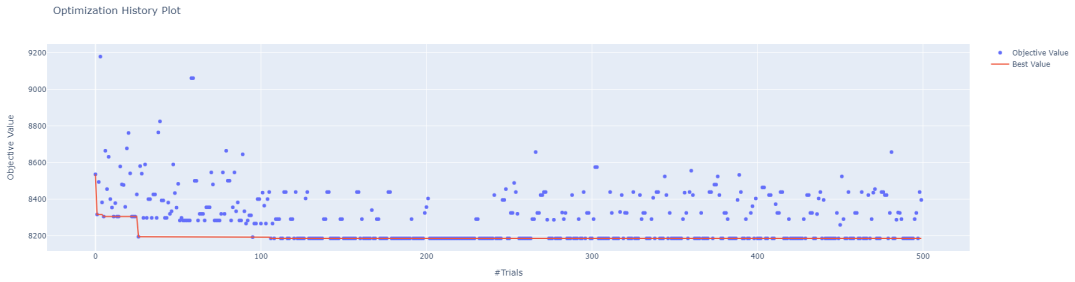
- 자동 하이퍼파라미터 최적화 프레임워크
  - 학습을 통해 Loss를 최소화 할 수 있는 파라미터 반환
  - 하이퍼 파라미터 튜닝 과정을 시각화를 통해 직관적으로 확인 가능
  - 하이퍼 파라미터 튜닝 시간 최소화

하이퍼마라미터 별 중요도 그래프





하이퍼파라미터 최적화 과정(History) Plot



4. 결과

- Private Score

#	팀	팀 멤버	최종점수	제출수	등록일
259	감사님강사합니다		130.14328	33	4일 전

- Public Score

93	감사님강사합니다		70.09804	33	4일 전

GitHub Link: https://github.com/andyjung12/Dacon_Competition_2021