1115 수상작 리뷰

주제

환경 데이터로부터 청경채의 일별 잎 면적 증감률 예측하는 AI 알고리즘 개발

각 청경채 케이스의 생육 기간 동안 1분 간격으로 수집된 환경 데이터로부터 일별 잎 면적 증감률을 예측

https://dacon.io/competitions/official/235961/overview/description

데이터

- train_input - 총 58개 청경채 케이스
  - 각 청경채 케이스 별 환경 데이터(1분 간격)
- train_target - 총 58개 청경채 케이스
  - rate : 각 청경채 케이스 별 잎 면적 증감률 (1일 간격)
- test_input - 총 6개의 청경채 케이스
  - 각 청경채 케이스 별 환경 데이터(1분 간격)
- test_target - 총 6개의 청경채 케이스
  - rate : 각 청경채 케이스 별 잎 면적 증감률(1분 간격)
- sample_submission - 제출파일
  - test 청경채 케이스 6개에 대한 일별 추론 결

코드정리

(1)패키지 로드

```
# -*- coding: utf-8 -*-
import pandas as pd
```

```
import numpy as np
from tqdm import tqdm
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from datetime import datetime, timedelta
import copy
import zipfile
```

(2) 전처리

```
#날짜를 시간을 제외하고 년도월일 형태로 변환
def change_year(date):
    date = date[:10].replace("-","")
    return date
```

```
#장비들이 작동하고 있는 시간만 추출하는 함수
def filter_pump(x):
    return x[x['펌프작동남은시간']!=0]

def filter_cold(x):
    return x[x['냉방작동남은시간']!=0]

def filter_hot(x):
    return x[x['난방작동남은시간']!=0]

def filter_inside_air(x):
    return x[x['내부유동팬작동남은시간']!=0]

def filter_outside_air(x):
    return x[x['외부환기팬작동남은시간']!=0]

def filter_white(x):
    return x[x['화이트 LED작동남은시간']!=0]
```

```
def filter_blue(x):
    return x[x['블루 LED작동남은시간']!=0]

def filter_red(x):
    return x[x['레드 LED작동남은시간']!=0]

def filter_total_red(x):
    return x[x['적색광추정광량']!=0]

def filter_total_blue(x):
    return x[x['청색광추정광량']!=0]

def filter_total_white(x):
    return x[x['백색광추정광량']!=0]

def is_in(type_list, type_dict, i):
    try:
        type_list.append(type_dict[i])
    except:
        type_list.append(0)
```

```
#컬럼 추가/삭제 함수
def change_col(particular_case):
    particular_case['시간'] = particular_case['시간'].apply(chang
    day_list = list(particular_case['시간'].drop_duplicates(keep
    group_df = particular_case.groupby('시간')

    pump_list = []
    group_pump = group_df.apply(filter_pump)
    pump_dict = group_pump['시간'].value_counts().to_dict()

    cold_list = []
    group_cold = group_df.apply(filter_cold)
    cold_dict = group_cold['시간'].value_counts().to_dict()
```

```
    hot_list = []
    group_hot = group_df.apply(filter_hot)
    hot_dict = group_hot['시간'].value_counts().to_dict()

    inside_list = []
    group_inside = group_df.apply(filter_inside_air)
    inside_dict = group_inside['시간'].value_counts().to_dict()

    outside_list = []
    group_outside = group_df.apply(filter_outside_air)
    outside_dict = group_outside['시간'].value_counts().to_dict(

    white_list = []
    group_white = group_df.apply(filter_white)
    white_dict = group_white['시간'].value_counts().to_dict()

    blue_list = []
    group_blue = group_df.apply(filter_blue)
    blue_dict = group_blue['시간'].value_counts().to_dict()

    red_list = []
    group_red = group_df.apply(filter_red)
    red_dict = group_red['시간'].value_counts().to_dict()

    total_red_list = []
    group_total_red = group_df.apply(filter_total_red)
    group_total_red = group_total_red.reset_index(drop=True)
    group_total_red = group_total_red.groupby('시간')
    total_red_dict = (group_total_red['적색광추정광량'].max() - gro

    max_red = []
    max_red_dict = group_total_red['적색광추정광량'].max().to_dict(
    min_red = []
    min_red_dict = group_total_red['적색광추정광량'].min().to_dict(

    total_blue_list = []
```

```
    group_total_blue = group_df.apply(filter_total_blue)
    group_total_blue = group_total_blue.reset_index(drop=True)
    group_total_blue = group_total_blue.groupby('시간')
    total_blue_dict = (group_total_blue['청색광추정광량'].max() - g

    max_blue = []
    max_blue_dict = group_total_blue['청색광추정광량'].max().to_dic
    min_blue = []
    min_blue_dict = group_total_blue['청색광추정광량'].min().to_dic

    total_white_list = []
    group_total_white = group_df.apply(filter_total_white)
    group_total_white = group_total_white.reset_index(drop=True)
    group_total_white = group_total_white.groupby('시간')
    total_white_dict = (group_total_white['백색광추정광량'].max() -

    max_white = []
    max_white_dict = group_total_white['백색광추정광량'].max().to_d
    min_white = []
    min_white_dict = group_total_white['백색광추정광량'].min().to_d

    sleep_list = []
    sleep_time = particular_case[(particular_case['화이트 LED작동남
    sleep_time = sleep_time.groupby('시간')
    sleep_time = sleep_time.size().to_dict()

    for i in day_list:
        is_in(sleep_list, sleep_time, i)
        is_in(pump_list, pump_dict, i)
        is_in(cold_list, cold_dict, i)
        is_in(hot_list, hot_dict, i)
        is_in(inside_list, inside_dict, i)
        is_in(outside_list, outside_dict, i)
        is_in(white_list, white_dict, i)
        is_in(blue_list, blue_dict, i)
        is_in(red_list, red_dict, i)
```

```
            is_in(total_red_list, total_red_dict, i)
            is_in(total_blue_list, total_blue_dict, i)
            is_in(total_white_list, total_white_dict, i)

            is_in(max_red, max_red_dict, i)
            is_in(min_red, min_red_dict, i)
            is_in(max_blue, max_blue_dict, i)
            is_in(min_blue, min_blue_dict, i)
            is_in(max_white, max_white_dict, i)
            is_in(min_white, min_white_dict, i)

        new_group_df = group_df.mean()

        new_group_df['냉방 시간'] = cold_list
        new_group_df['난방 시간'] = hot_list
        new_group_df['내부 팬'] = inside_list
        new_group_df['외부 팬'] = outside_list
        new_group_df['백색'] = white_list
        new_group_df['청색'] = blue_list
        new_group_df['적색'] = red_list
        new_group_df['LED_off'] = sleep_list
        new_group_df['적색 크기차'] = total_red_list
        new_group_df['청색 크기차'] = total_blue_list
        new_group_df['백색 크기차'] = total_white_list

        new_group_df['적색 최대'] = max_red
        new_group_df['적색 최소'] = min_red
        new_group_df['청색 최대'] = max_blue
        new_group_df['청색 최소'] = min_blue
        new_group_df['백색 최대'] = max_white
        new_group_df['백색 최소'] = min_white

        new_group_df['펌프 작동시간'] = pump_list
        new_group_df['실내 일교차'] = list(group_df['내부온도관측치'].max
        new_group_df['습도_온도'] = new_group_df['내부온도관측치'] * new_
        new_group_df['습도_온도_CO2'] = new_group_df['습도_온도'] * new
```

```
    return new_group_df
```

```
#train, test set 생성 함수
def get_data(data_list):
    copy_data_list = copy.deepcopy(data_list)
    new_data = ""
    for i in range(len(copy_data_list)):
        if i == 0:
            new_data = copy_data_list[i]
        else:
            new_data = pd.concat([new_data, copy_data_list[i]])
    del_list = []
    for i in new_data.columns:
        if '남은시간' in i:
            del_list.append(i)
    new_data = new_data.drop(del_list, axis=1)
    return new_data
```

```
#데이터 스케일링
result = []
train['label'] = np.log1p(train['label'])

y_train = train['label']
X = train.drop(['label'],axis=1)
X_test = test.drop(['label'],axis=1)
X_col = X.columns

scaler = StandardScaler()

scaler.fit(X)
scaled_X_train  = scaler.transform(X)
scaled_X_train  = pd.DataFrame(scaled_X_train , columns=X_col)
scaled_X_test  = scaler.transform(X_test)
```

```
scaled_X_test  = pd.DataFrame(scaled_X_test , columns=X_col)
scaled_X_test
```

(3)train set, test set 생성

```
train = get_data(train_list)
test = get_data(test_list)
```

(4) 모델 생성

```
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', max_de
xg_reg.fit(scaled_X_train, y_train)
pred = xg_reg.predict(scaled_X_test)
for k in pred:
    result = np.hstack([result, k])
result
```

```
expm1_result = np.expm1(result)
expm1_result = pd.DataFrame(columns=['label'],data=expm1_result
total_result = []
for case in tqdm(range(1, 7)):
    if case < 10:
        case = '0'+str(case)
    particular_label = pd.read_csv(f'data/test_target/TEST_{case
    particular_label['rate'] = list(expm1_result['label'][:len(p
    expm1_result = expm1_result.drop([i for i in range(len(parti
    expm1_result = expm1_result.reset_index(drop=True)
    total_result.append(particular_label)
for i in range(len(total_result)):
    total_result[i].to_csv(f'TEST_0{i+1}.csv', index=False)
file_ls = ['TEST_01.csv','TEST_02.csv','TEST_03.csv','TEST_04.cs
with zipfile.ZipFile("submission.zip", 'w') as my_zip:
    for i in file_ls:
```

```
        my_zip.write(i)
    my_zip.close()
```

차별점 및 배울점

- 전처리 과정에서 결측값이 완전히 제거되지 않은 경우 **앞뒤 값을 채우는** 보조 방법을 제시한 점이 인상적이었다.
- **다양한 피처를 추가**하거나 모델 구조를 변경하며 성능 개선을 위해 반복적으로 검증한 점 또한 인상적이었다.
    - 이 부분을 통해 모델링의 유연성과 성능 최적화에서 중요한 부분을 확인할 수 있었다.