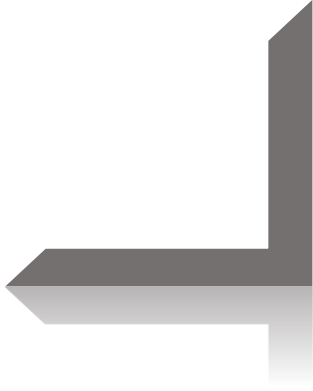




# 동서발전 태양광 발전량 예측 AI 경진대회

투빅스 팀  
이성범 강재영 강지우



# 목차

---

1. 데이터 수집
  2. 데이터 전처리
  3. 피쳐 엔지니어링
  4. 모델 설명
  5. 모델 학습
  6. 모델 예측
  7. 개발 환경
  8. 제출파일 설명
-

# 1. 데이터 수집

---

# 1. 데이터 수집

번호	출처	수집데이터	데이터 세부 항목	파일명	기간
1	기상청 <a href="https://data.kma.go.kr/data/rmt/rmtList.do?code=420&amp;pgmNo=572">https://data.kma.go.kr/data/rmt/rmtList.do?code=420&amp;pgmNo=572</a>	날씨 데이터	Temperature, PrecipitationForm, PrecipitationProb, Humidity, WindSpeed, WindDirection, Cloud, Precipitation	당진데이터_완료.csv, 울산데이터_완료.csv	2016.12.01 ~ 2020.08.13
2	공공데이터포털 <a href="https://www.data.go.kr/data/15003553/fileData.do">https://www.data.go.kr/data/15003553/fileData.do</a>	발전량 데이터	dangjin_floating, dangjin_warehouse, dangjin, ulsan	new_energy.csv	2016.12.02 ~ 2021.01.31 (dangjin_floating 2018 이전 데이터 존재 x)
3	한국천문연구원 <a href="https://astro.kasi.re.kr/life/pageView/10">https://astro.kasi.re.kr/life/pageView/10</a>	태양의 고도 및 방위각 데이터	방위각, 고도, 적경, 적위	dangjin_sun_height.csv, ulsan_sun_height.csv	2014.01.01 ~ 2021.10.30

# 1. 데이터 수집

- 한국천문연구원 - 태양 방위각, 고도, 적경, 적위 크롤링 코드

위치입력

충청남도 당진시 석문면 교로길 30

```
1 url = 'https://astro.kasi.re.kr/life/pageView/10?useElevation=1&lat=37.54980&lng=126.9671&elevation=0&output_range=1&date=2021-01-01'
2 driver.get(url)
3 # 페이지 소스 가져오기
4 html=driver.page_source
5 soup = BeautifulSoup(html)
6 time.sleep(2)
```

```
1 start = '2014-01-01'
2 end = '2021-12-30'
```

```
1 day =pd.date_range(start='2014-01-01', end='2021-12-30', freq='D')
```

```
1 day.astype(str)[0]
```

```
'2014-01-01'
```

```
1 # angle_1 = 방위각 , _2 = 고도 , _3 = 적경 , _4 = 적위
```

```
1 Chrome_directory = 'D:/02.users/wodud2468/chromedriver/chromedriver.exe'
2 driver = webdriver.Chrome(Chrome_directory)
3 options = webdriver.ChromeOptions()
4 date = []
5 hour = []
6 angle_1 = []
7 angle_2 = []
8 angle_3 = []
9 angle_4 = []
10 for d in day.astype(str):
11     url = 'https://astro.kasi.re.kr/life/pageView/10?useElevation=1&lat=37.05127236188373&lng=126.51409076261092&elevation=1'
12     driver.get(url)
13     for i in range(1,25):
14         date.append(d)
15         hour.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[1]'.format(i)).text)
16         angle_1.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[2]'.format(i)).text)
17         angle_2.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[3]'.format(i)).text)
18         angle_3.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[4]'.format(i)).text)
19         angle_4.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[5]'.format(i)).text)
```

위치입력

울산광역시 남구 용잠로 623

```
1 Chrome_directory = 'D:/02.users/wodud2468/chromedriver/chromedriver.exe'
2 driver = webdriver.Chrome(Chrome_directory)
3 options = webdriver.ChromeOptions()
4 date = []
5 hour = []
6 angle_1 = []
7 angle_2 = []
8 angle_3 = []
9 angle_4 = []
10 for d in day.astype(str):
11     url = 'https://astro.kasi.re.kr/life/pageView/10?useElevation=1&lat=19.694477084885456&lng=117.99260273191031&elevation=1'
12     driver.get(url)
13     for i in range(1,25):
14         date.append(d)
15         hour.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[1]'.format(i)).text)
16         angle_1.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[2]'.format(i)).text)
17         angle_2.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[3]'.format(i)).text)
18         angle_3.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[4]'.format(i)).text)
19         angle_4.append(driver.find_element_by_xpath('//*[@id="sun-height-table"]/table/tbody/tr/{}/td[5]'.format(i)).text)
```

```
1 df2 = pd.DataFrame([date, hour, angle_1, angle_2, angle_3, angle_4]).T
2 df2.columns = ['date', 'hour', '방위각', '고도', '적경', '적위']
```

```
1 df2.to_csv('ulsan_sun_height.csv')
```

## 2. 데이터 전처리

---

## 2. 데이터 전처리

- 기상데이터 3시간 예보변수, 6시간 예보변수 결합

```
# 데이터 결합 (3시간)
data_year = data_year_temperature[['format: day', 'hour', 'forecast']]
data_year['Temperature'] = data_year_temperature[data_year_temperature.columns[-1]] #3시간 기온
data_year['PrecipitationForm'] = data_year_precipitationform[data_year_precipitationform.columns[-1]] #강수형태
data_year['PrecipitationProb'] = data_year_precipitationprob[data_year_precipitationprob.columns[-1]] #강수확률
data_year['Humidity'] = data_year_humidity[data_year_humidity.columns[-1]] #3시간 습도
data_year['WindSpeed'] = data_year_windspeed[data_year_windspeed.columns[-1]] #3시간 풍속
data_year['WindDirection'] = data_year_winddirection[data_year_winddirection.columns[-1]] #3시간 풍향
data_year['Cloud'] = data_year_cloud[data_year_cloud.columns[-1]]
```

```
# 데이터 결합 (6시간)
data_year = data_year_precipitation[['format: day', 'hour', 'forecast']]
data_year['Precipitation'] = data_year_precipitation[data_year_precipitation.columns[-1]] #강수량
data_year['Snow'] = data_year_snow[data_year_snow.columns[-1]] #적설량
```

## 2. 데이터 전처리

---

- 가장 최신의 예보를 바탕으로 선형보간

```
# 비어있는 결측치 선형보간으로 채우는 함수
def interpolate_df(df_, method='linear'):

    df = df_.copy()
    df['Forecast_time'] = pd.to_datetime(df['Forecast_time'])

    new_df = pd.DataFrame()
    new_df['Forecast_time'] = pd.date_range(start=df['Forecast_time'].iloc[0], end=df['Forecast_time'].iloc[-1], freq='H') #Forecast_time에 맞는 날짜 범위 생성
    new_df = pd.merge(new_df, df, on='Forecast_time', how='outer')

    return new_df.interpolate(method=method)
```



## 2. 데이터 전처리

- 각 발전소별로 energy 결측치가 존재하는 날짜 제거
- 일별 기준 energy value 합계가 0 인 경우 제거
  - 기계 고장 및 점검으로 간주

```
def get_energy_df(col):  
    if 'dangjin' in col:  
        _df = dangjin_interpolated.copy()  
    else:  
        _df = ulsan_interpolated.copy()  
  
    _energy = energy.copy()  
  
    null_idx = energy[energy[col].isna()][['time']].values  
  
    floating = energy.groupby('date')[col].sum()  
    zero_floating_idx = floating[floating == 0].index  
  
    _df = _df[~_df['Forecast_time'].isin(null_idx)]  
    _energy = _energy[~_energy['time'].isin(null_idx)]  
  
    _df = _df[~_df['date'].isin(zero_floating_idx)]  
    _energy = _energy[~_energy['date'].isin(zero_floating_idx)]  
  
    _df = _df.reset_index(drop=True)  
    _energy = _energy.reset_index(drop=True)  
    cols = ['time', col]  
  
    _df = pd.concat([_df, _energy[cols]], axis=1)  
  
    return _df
```

### 3. 피쳐 엔지니어링

### 3. 피쳐 엔지니어링

---

- 각도와 시간을 나타내는 칼럼은 모두 **sin, cos** 값을 추가로 구함
- **Temperature**의 경우 계절과 시간을 기준으로 평균 **Temperature**를 추가로 구함

```
def angle_to_cos(x):  
    return np.cos(np.pi/180*(x-90))  
  
def angle_to_sin(x):  
    return np.sin(np.pi/180*(x+90))
```

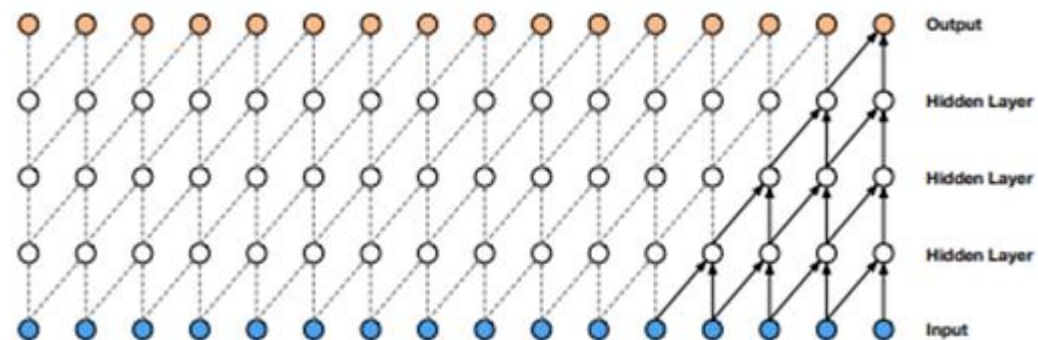
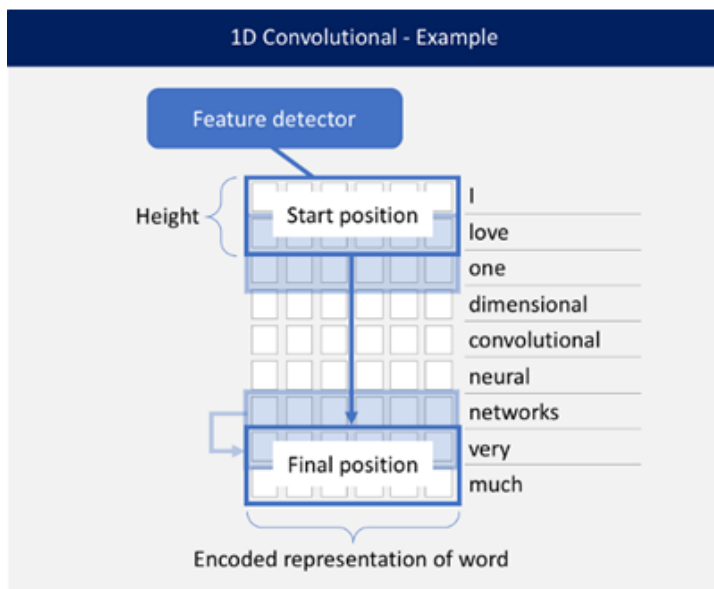
```
# diurnal_Temperature  
hour_mean = _df.groupby(_df['hour']).Temperature.mean()  
hour_mean = pd.DataFrame(hour_mean)  
hour_mean.columns = ['hour_mean_temp']  
  
_df = _df.merge(hour_mean, on='hour', how='left')  
_df['diurnal_Temperature'] = _df['Temperature'].sub(_df['hour_mean_temp'])  
  
# seasonal_Temperature  
month_mean = _df.groupby(_df['month']).Temperature.mean()  
month_mean = pd.DataFrame(month_mean)  
month_mean.columns = ['month_mean_temp']
```

## 4. 모델

---

## 4. 모델

- Causal Padding을 사용한 CNN1D 모델 구축
- feature selection과 추가 적인 feature engineering을 모두 모델에 맡기기 위해서 모든 feature를 사용함
- 예측하고자 하는 시간을 기준으로 이전 3일치 데이터를 가지고 예측을 하는 방식으로 모델을 구성



<CNN1D, Causal Padding>

## 4. 모델

### Causal Padding을 사용한 CNN1D 모델

```
22 # 시간에 대한 상관성을 함성곱으로 나타내고자 CNN1D 모델 구축
23 def set_model(): # Causal Padding을 사용한 CNN1D 모델 구축함수
24
25     nf = 16
26     fs = 3
27     padding = 'causal' # Convolution을 진행할 때, 매 step에서 output이 오직 현시점의 input과 과거 시점들의 데이터에만 종속되도록하기 위해서 Causal Padding을 사용
28     activation = 'elu' # Gradient Vanishing 문제를 방지하기 위해서 Elu 사용
29
30     model = Sequential()
31
32     model.add(keras.layers.InputLayer((72, 30)))
33
34     model.add(Conv1D(filters = nf, kernel_size = fs, padding = padding))
35     model.add(BatchNormalization()) # 비선형 성질을 유지 하면서 학습 될 수 있게 해주고, regularization 효과를 가지기 위해 배치정규화 설정
36     model.add(Activation(activation = activation))
37     # model.add(Dropout(0.4))
38
39     model.add(Conv1D(filters = nf * 2, kernel_size = fs, padding = padding))
40     model.add(BatchNormalization()) # 비선형 성질을 유지 하면서 학습 될 수 있게 해주고, regularization 효과를 가지기 위해 배치정규화 설정
41     model.add(Activation(activation = activation))
42     # model.add(Dropout(0.4))
43
44     model.add(Conv1D(filters = nf * 4, kernel_size = fs, padding = padding))
45     model.add(BatchNormalization()) # 비선형 성질을 유지 하면서 학습 될 수 있게 해주고, regularization 효과를 가지기 위해 배치정규화 설정
46     model.add(Activation(activation = activation))
47     # model.add(Dropout(0.4))
48
49     model.add(Conv1D(filters = nf * 8, kernel_size = fs, padding = padding))
50     model.add(BatchNormalization()) # 비선형 성질을 유지 하면서 학습 될 수 있게 해주고, regularization 효과를 가지기 위해 배치정규화 설정
51     model.add(Activation(activation = activation))
52     # model.add(Dropout(0.4))
53
54     model.add(Conv1D(filters = nf * 16, kernel_size = fs, padding = padding))
55     model.add(BatchNormalization()) # 비선형 성질을 유지 하면서 학습 될 수 있게 해주고, regularization 효과를 가지기 위해 배치정규화 설정
56     model.add(Activation(activation = activation))
57     # model.add(Dropout(0.4))
58
59     model.add(Conv1D(filters = nf * 32, kernel_size = fs, padding = padding))
60     model.add(BatchNormalization()) # 비선형 성질을 유지 하면서 학습 될 수 있게 해주고, regularization 효과를 가지기 위해 배치정규화 설정
61     model.add(Activation(activation = activation))
62     # model.add(Dropout(0.4))
```

## 5. 학습

## 5. 학습

---

- Loss의 경우 Metric과 유사한 MAE로 선택
- Optimization의 경우 RMSprop으로 선택(시계열의 경우 RMSprop이 조금 더 우수)

```
# Optimization의 경우 RMSprop으로 선택(시계열의 경우 RMSprop이 조금 더 우수한 성능을 보임)
optimizer = keras.optimizers.RMSprop() # 실제로 RMSprop, SGD, Adam 등을 비교해본 결과 RMSprop가 가장 우수했음

model.compile(loss = 'mae', optimizer = optimizer, metrics=[my_metric]) # Loss는 Metric과 가장 유사한 MAE 사용
```



## 5. 학습

- Train Data를 Random Sampling 하고 CV를 5로 학습
- 8 : 2로 Train과 Val Data를 나눠서 학습

```
9 # Train Data를 Random Sampling 하고 CV를 5로 학습
10 # - 랜덤샘플링을 통해 특정 Fold에만 강한 모델이 아닌 전체적으로 강건한 모델이 만들어짐
11 # 8 : 2로 Train과 Val Data를 나눠서 학습
12 n_split = 5
13 kfold = KFold(n_splits = n_split, shuffle=True, random_state=22)
14
15 accuracy = []
16 losses=[]
17
18 for i, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train)):
19     train_X, val_X = X_train[train_idx], X_train[val_idx]
20     train_y, val_y = y_train[train_idx], y_train[val_idx]
21
22     model = set_model()
23     mc = ModelCheckpoint(PATHS + m + f'_cv_study{i + 1}.h5', save_best_only=True, verbose=0, monitor = 'val_my_metric', mode = 'min', save_weights_only=True) # 모델 저장
24     reLR = ReduceLROnPlateau(monitor = 'val_my_metric', patience = 7, verbose = 1, factor = 0.5) # 학습률이 개선되지 않을 때, 학습률을 동적으로 조정하여 학습률을 개선하는 효과를 기대하는 역할
25     # val_my_metric을 기준으로 epoch 7 동안 개선되지 않으면 학습률을 절반으로 줄이는 콜백함수
26
27     history = model.fit(train_X, train_y, epochs = EPOCHS, validation_data = (val_X, val_y), # 모델 적합
28                         verbose=1, batch_size=BATCH_SIZE, callbacks = [mc, reLR])
29
30     model.load_weights(PATHS + m + f'_cv_study{i + 1}.h5') # 모델 로드
31
32     k_accuracy = '%.4f' % (model.evaluate(val_X, val_y)[1]) # 정확도 계산
33     k_loss = '%.4f' % (model.evaluate(val_X, val_y)[0]) # Loss 계산
34
35     accuracy.append(k_accuracy)
36     losses.append(k_loss)
```

## 6. 예측

---

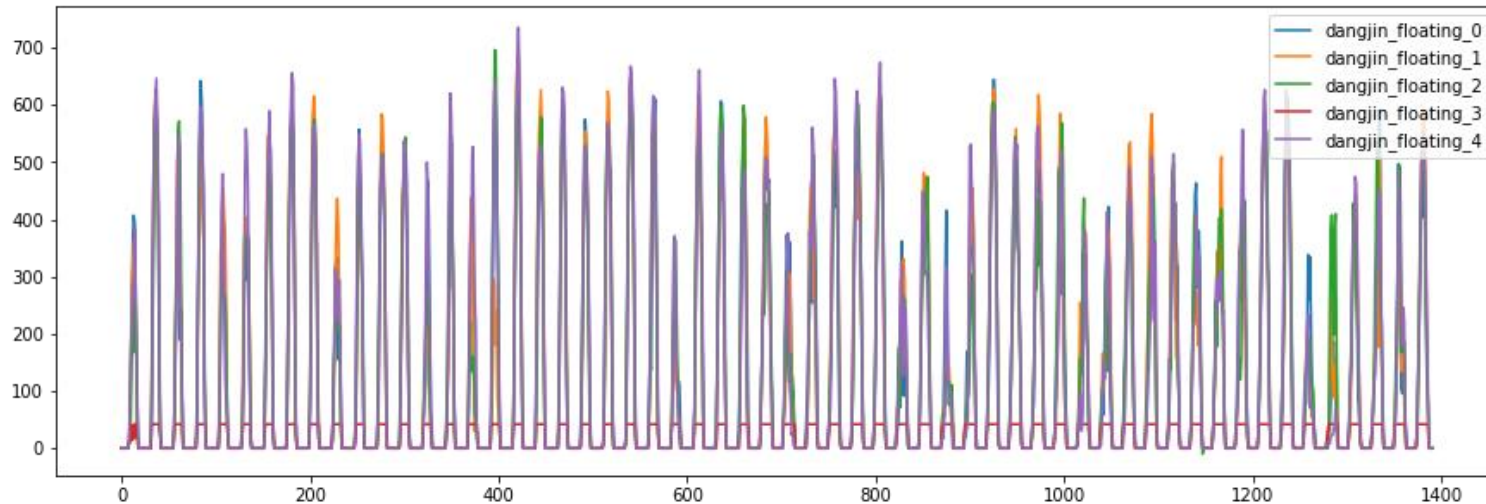
## 6. 예측

- CV별 예측 값을 평균 내어 최종 예측 값으로 활용
- CV별 예측 시에 학습이 제대로 안된 모델은 예측에서 제외 ( Example, dangjin\_floating\_3 )
- Random Sampling의 효과로 CV별 모델이 서로 잘 맞추는 부분이 달라짐으로써 강건한 모델이 만들어짐

```
preds = []
m = 'dangjin_floating'

for i in range(5):
    if i == 3: pass
    else:
        model = set_model()
        model.load_weights(PATHS + m + f'_cv_study{i + 1}.h5')
        pred = model.predict(X_test)
        preds.append(pred)

dangjin_floating_pred = np.mean(preds, axis=0)
```



## 7. 개발환경

---

## 7. 개발환경

---

- COLAB 환경에서 학습

- OS 버전

Google Colaboratory GPU 사용

OS : Linux-5.4.104+-x86\_64-with-Ubuntu-18.04-bionic

Process information : x86\_64

Process Architecture : x86\_64

RAM Size : 13(GB)


- requirements.txt 참고

## 8. 제출파일 설명


---


## 8. 제출파일 설명



이름 ↑



 .ipynb\_checkpoints



 data

 my\_model

 train\_model



 (1) 데이터 수집.ipynb 


 (2) 학습 및 추론.ipynb 

 (외) 데이콘 리더보드 복원.ipynb 

 CNN1D\_0707\_17\_ret.csv 

Final  
Submission

 requirements.txt 

 sample\_submission.csv 

- data : 모델에 사용된 data
- my\_model : 투빅스 팀의 예측 모델 저장폴더
- train\_model : 데이콘 측 검증을 위한 모델 저장폴더
- (1) 데이터 수집 : 태양 각도 데이터 크롤링  
(크롬드라이버를 설치해야하기 때문에 따로 분류했습니다.)
- (2) 학습 및 추론 : 데이콘 측에서 학습과정을 검증할 수 있도록 작성한 코드입니다.
- (외) 데이콘 리더보드 복원 : 저희 팀에서 예측에 사용했던 모델을 Load해서 추론할 수 있는 코드입니다.

코드를 모듈화 혹은 하나의 코드로 보내달라고 하셨는데, (1) 데이터 수집은 '크롬 드라이버 이슈'를 고려하여 구분하였습니다.

(외) 데이콘 리더보드 복원은 투빅스 팀에서 예측에 사용한 모델을 첨부하는 차원에서 필수코드는 아니지만 따로 코드를 정리해서 포함하였습니다.