**제 4회 빅스타(빅데이터, 스타트업) 경진대회**

**코드 설명자료**

**2022년 12월 05일**

**팀 명 : KNP**

1. **라이브러리 및 데이터 (Library & Data)**

* torch, natsort 라이브러리 설치 필수
* 기상청 기상자료 개방포털 종관기상관측(ASOS) – 자료
  + 기간: 20190101 01 ~ 20211231 23
  + 자료형태 : 시간자료
  + 지점 : 울진(130) 전체
* 기상 자료는 파일 형식 문제로 cp949로 인코딩하여 불러온 후, concat
* test 데이터셋은 테스트 폴더 내에 들어있는 모든 csv 파일을 natsort를 이용하여 순서대로 불러온 뒤, 하나의 데이터셋으로 합침

| // start\_train\_model.py  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import sys  import warnings  import random  import os  import datetime  import torch  from torch.utils.data import TensorDataset # 텐서데이터셋  from torch.utils.data import DataLoader # 데이터로더  weather = load\_weather()  train = pd.read\_csv("../dataset/train/train.csv")  test = load\_test\_data()  // utils.py  def load\_weather(path="../dataset/weather/"):  # Load weather data  weather\_2019 = pd.read\_csv(f"{path}OBS\_ASOS\_TIM\_2019.csv", encoding="cp949")  weather\_2020 = pd.read\_csv(f"{path}OBS\_ASOS\_TIM\_2020.csv", encoding="cp949")  weather\_2021 = pd.read\_csv(f"{path}OBS\_ASOS\_TIM\_2021.csv", encoding="cp949")  weather = pd.concat([weather\_2019, weather\_2020, weather\_2021])  return weather  def load\_test\_data(path="../dataset/test/"):  file\_list = os.listdir(path)  sorted\_file\_list = natsort.natsorted(file\_list)  # print(sorted\_file\_list[:10])  test\_list = [  file for file in sorted\_file\_list if file.endswith(".csv")  ] # 파일명 끝이 .csv인 경우  # csv 파일들을 DataFrame으로 불러와서 concat  test = pd.DataFrame()  for i in test\_list:  data = pd.read\_csv(path + i)  test = pd.concat([test, data])  test = test.reset\_index(drop=True)  return test |
| --- |

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

* 학습 데이터
  + 시간 데이터인 TIME을 인덱스 컬럼으로 설정
  + YEAR, DAYOFYEAR, HOUR, MUNUTE 변수를 인덱스 컬럼을 통해 생성
  + 온도 절대온도로 변환
* 기상 데이터
  + 시간 데이터인 일시를 TIME 인덱스 컬럼으로 설정
  + YEAR, DAYOFYEAR, HOUR 변수를 인덱스 컬럼을 통해 생성
  + 일조량의 경우 결측값 0으로 대체
  + 온도, 압력, 풍속의 경우 이전값으로 대체
  + 온도 절대온도로 변환
  + 압력 kPa로 변환

| // train\_estimator.py  X["TIME"] = pd.to\_datetime(X["TIME"])  X = X.set\_index("TIME")  X["YEAR"] = X.index.year  X["DAYOFYEAR"] = X.index.dayofyear  X["HOUR"] = X.index.hour  X["MINUTE"] = X.index.minute  X["TI\_MEAN"] = X["TI\_MEAN"] + 273.15  // weather\_estimator.py  X = X.rename(columns={"일시": "TIME"})  X["TIME"] = pd.to\_datetime(X["TIME"])  X = X.set\_index("TIME")  X = X[self.origin]  X.columns = self.change  X["YEAR"] = X.index.year  X["DAYOFYEAR"] = X.index.dayofyear  X["HOUR"] = X.index.hour  X[self.zero\_cols] = X[self.zero\_cols].fillna(0)  X[self.interpolate\_cols] = X[self.interpolate\_cols].fillna(method="ffill")  X[self.pressure\_cols] = X[self.pressure\_cols] \* 0.1  X[self.temperature\_cols] = X[self.temperature\_cols] + 273.15 |
| --- |

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

* 학습 데이터 내 대기압 PREESURE-S의 이상치가 많아, 기상데이터의 대기압과 비교하여 대기압 차가 2.5kPa이상 난다면 학습 데이터의 대기압을 기상데이터 대기압으로 대체
* 데이터의 날짜와 시간이 주기적으로 순환하여 데이터의 연관성이 낮음
  + 데이터의 날짜를 평균적인 최저기온 날짜인 1월 15일을 기점으로 삼각함수로 변환하여 변수 생성
  + 데이터의 시간을 삼척의 태양 남중 시간인 12시 24분을 기점으로 삼각함수로 변환하여 변수 생성

| // train\_estimator.py  X["PRESSURE\_DIFF"] = X["Local\_atmospheric\_pressure"] - X["PRESSURE-S"]  X["PRESSURE-S"].loc[abs(X["PRESSURE\_DIFF"]) > self.threhold] = X["Local\_atmospheric\_pressure"]  X["PRESSURE\_DIFF"] = X["Local\_atmospheric\_pressure"] - X["PRESSURE-S"]  X["DAYOFYEAR\_sin"] = np.sin((X["DAYOFYEAR"] - 15) \* (2 \* np.pi / 365.2425))  X["DAYOFYEAR\_cos"] = np.cos((X["DAYOFYEAR"] - 15) \* (2 \* np.pi / 365.2425))  X["HOUR\_sin"] = np.sin((X["HOUR"] + 60 \* X["MINUTE"] - 24) \* (2 \* np.pi / 24))  X["HOUR\_cos"] = np.cos((X["HOUR"] + 60 \* X["MINUTE"] - 24) \* (2 \* np.pi / 24)) |
| --- |

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

* 측정 데이터 셋 기반 파생변수 생성

| PIA205B-02A\_DIFF | 탱크 내부 압력 최솟값 최대값 차이 |
| --- | --- |
| PRESSURE\_MAX\_DIFF | 대기압과 탱크 압력 최대값 차이 |
| BOG | BOG가스 유량 |
| TI\_SUM | 재액화기로 인입되는 유량 |
| OUTLET\_SUM | 계량설비로 인입되는 유량 |
| TI\_ACC | 주배관 송출 유량과 계량설비로 인입되는 유량의 차이 |
| TI\_P\_MAX | 재액화된 LNG 절대 온도를 탱크 내부 최대 압력으로 나눈 값  (PV = nRT => T/P = V/nR) |
| TI\_VOL\_MAX | 재액화된 LNG의 부피 상관값  (T/P x n = V/nR x n =V/R) |

* 기상 데이터 셋 연관 파생변수 생성

| PRESSURE\_DIFF | 공정 데이터의 대기압과 기상관측데이터의 대기압이 차이 |
| --- | --- |
| TI\_T\_DIV | 탱크 내부 온도와 기온의 차이 |
| T\_G\_DIFF | 지표 온도와 기온의 차이 |
| CONVEC | 탱크 내외부 기온차 x 풍속 (대류 상관값) |
| DAYOFYEAR\_sin | 날짜를 1월 15일 기준으로 삼각함수 변환(연중 최저/최고 기온 날짜) |
| DAYOFYEAR\_cos |
| HOUR\_sin | 시간을 12시 24분을 기준으로 삼각함수 변환(속초의 태양 남중시간) |
| HOUR\_cos |

| // train\_estimator.py  X["PIA205B-02A\_DIFF"] = X["PIA205B-02A\_MAX"] - X["PIA205B-02A\_MIN"]  X["PRESSURE\_MAX\_DIFF"] = X["PRESSURE-S"] - X["PIA205B-02A\_MAX"]  X["TI\_MEAN"] = X["TI\_MEAN"] + 273.15  X["BOG"] = X["FY\_SUM"] + X["FIA\_SUM"]  X["TI\_SUM"] = X["FY\_SUM"] + X["LP\_TOTAL"]  X["OUTLET\_SUM"] = X["TI\_SUM"] + X["FIA\_SUM"]  X["TI\_ACC"] = X["OUTLET\_SUM"] - X["STN-MFR-S"]  X["TI\_P\_MAX"] = X["TI\_MEAN"] / X["PIA205B-02A\_MAX"]  X["TI\_VOL\_MAX"] = X["TI\_P\_MAX"] \* X["TI\_SUM"]  X["TI\_T\_DIV"] = X["Temperature"] - X["TI\_MEAN"]  X["T\_G\_DIFF"] = X["Ground\_temperature"] - X["Temperature"]  X["CONVEC"] = X["TI\_T\_DIV"] \* X["Wind"] |
| --- |

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

* 기본 딥러닝 모델을 Input layer - LSTM - Dense - Output layer 순으로 구성
* Grid Search를 실시하여 최적의 수치로서 Input layer - LSTM(16) - LSTM(16) - Dense(2) - Output layer로 구성함
* custom loss로 weighted\_mae\_loss 함수를 생성하여 WMAE 구현
* Adam optimizer를 바탕으로 LearningRate Scheduler를 구성하여 CosineAnnealingWarmUpRestarts 클래스를 설정하여 주기적으로 학습률이 바뀔 수 있도록 구성
* EarlyStopping을 구성하여 patience를 100으로 설정
* epoch 1000으로 설정
* batch size를 2, 4, 8, 16, 32, 64, 128, 256 모두 실험하여 최적의 수치로 32 선정
* train\_test\_split을 0.6:0.4, 0.7:0.3, 0.8:0.2, 0.9:0.1 모두 실험하여 최적의 수치로 0.8 : 0.2로 선정
* 파이프라인을 모두 실험하여 최적의 파이프라인으로 현재 파이프라인 설정

| // LD.py  class LstmDense(nn.Module):  # # 기본변수, layer를 초기화해주는 생성자  def \_\_init\_\_(self, input\_dim, hidden\_dim, fc1\_dim, seq\_len, output\_dim, layers):  super(LstmDense, self).\_\_init\_\_()  self.hidden\_dim = hidden\_dim  self.seq\_len = seq\_len  self.output\_dim = output\_dim  self.layers = layers  self.lstm = nn.LSTM(  input\_dim,  hidden\_dim,  num\_layers=layers,  # dropout = 0.1,  batch\_first=True,  )  self.fc1 = nn.Linear(hidden\_dim, fc1\_dim, bias=True)  self.fc2 = nn.Linear(fc1\_dim, output\_dim, bias=True)  # 학습 초기화를 위한 함수  def reset\_hidden\_state(self):  self.hidden = (  torch.zeros(self.layers, self.seq\_len, self.hidden\_dim),  torch.zeros(self.layers, self.seq\_len, self.hidden\_dim),  )  # 예측을 위한 함수  def forward(self, x):  x, \_status = self.lstm(x)  x = self.fc1(x[:, -1])  x = self.fc2(x)  return x  // train\_model.py  def weighted\_mae\_loss(output, target, prev):  mae = torch.unsqueeze(torch.abs(output - target), dim=1)  weight = torch.abs(target - prev)  weight = torch.unsqueeze(  weight / torch.unsqueeze(torch.sum(torch.add(weight, 0.00000001), 1), dim=1),  dim=2,  )  return torch.sum(torch.bmm(mae, weight))  def train\_model():  criterion = weighted\_mae\_loss  optimizer = optim.Adam(model.parameters(), lr=0.000001)  scheduler = CosineAnnealingWarmUpRestarts(  optimizer, T\_0=50, T\_mult=1, eta\_max=lr, T\_up=10, gamma=0.5  )  early\_stopping = EarlyStopping(patience=patience, verbose=False)  iters = len(train\_loader)  for epoch in range(num\_epochs):  model.train()  for batch, (data, target) in enumerate(train\_loader, 1):  optimizer.zero\_grad()  output = model(data) loss = criterion(output, target, data[:, 5, :2])  loss.backward()  optimizer.step()  scheduler.step(epoch + batch / iters) |
| --- |

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

* 딥러닝 모델을 더 거대한 모델로 설정할 필요가 있어보임
* 여러 예측 모델을 구현하여 앙상블 모델로 구현하는 조치가 필요함
* 데이터 리키지가 일어나지 않도록 차분된 값을 사용하지 않았지만, 이후 차분된 값을 사용할 필요가 있음
* Optuna 등 하이퍼파라미터 튜닝 라이브러리를 통해 자동적으로 하이퍼파라미터를 설정할 필요가 있음
* epoch 100 이후로 추가적인 학습이 되지 않아 epoch 수치를 100으로 낮춤
* Loss Plot을 그렸을 때, 극단적인 Outlier가 발생하는 경우가 있어 이러한 수치를 제거할 필요가 있음
* 데이터 증강을 통한 추가적 학습 가능성이 남아있음

| |  | | --- | |
| --- | --- |