센서를 이용한 워터펌프 중단 가능성 확인 프로그램

이름: 정성문

학번: 2118317

Github: https://github.com/jeongseongmoon2118317/A-water-pump-interruption-check-program-using-a-sensor

1. 안전 관련 머신러닝 모델 개발 관련 요약

센서의 입력 신호를 기반으로 펌프가 일정 기간 동안 중단될 수 있는지 여부의 상관관계를 예측하기 위해 LSTM모델을 적용했습니다. (LSTM 모델: LSTM은 순차적인 데이터를 다루는 데 특화된 신경망 모델로, 주로 시계열 데이터나 시간에 따른 의존성을 갖는 데이터에서 사용됩니다.

사용된 모델의 입력 매개변수는 센서의 입력 값입니다. 모델의 출력은 단일 매개변수로, 펌프 작동 상태는 0은 셧다운 상태, 1은 정상 작동 상태, 0.5는 복구 중으로 설정하였습니다.

이 프로그램은 입력 매개변수(센서) 간의 상관관계를 분석하고 매개변수를 결정하여 모델의 출력을 결정함으로써 입력 매개변수는 적지만 가장 정확한 예측 결과를 도출해낼 수 있는 모델을 만들었습니다.

2. 개발 목적 및 배경지식

워터펌프는 저희 일상생활이나 공업, 건축업 등 다양한 분야에서 많이 사용되고 있습니다. 그만큼 수요도 많고 인간에게 필수적인 장치라고 생각 했습니다.

하지만 이 펌프는 하루에도 셀 수 없이 많이 사용되고 고장이나 폭발 등 위험요인이 그만큼 더 많다고 생각했기에 이를 센서를 사용하면 보다 효율적이고 체계적으로 원인을 파악할 수 있다고 생각했기에 이러한 주제를 선정하게 되었습니다. 또 어떤 것이 문제가 생겼는지, 결과값을 토대로 분석하여 고장, 사고를 예방하는 기능도 포함되어 있습니다.

그렇기에 이 프로그램을 구성할 때 압력, 유량 또는 온도와 같은 단일 작동 매개를 센서

가 측정하는 것을 매개변수로 설계하고 각 상황에 따라 문제를 파악하여 대처, 예방하는 프로그램을 만들었습니다.

이 프로그램은 궁극적으로 일상생활이나 현장에서 어떠한 상황에 워터펌프가 고장이 발생하는지 분석하기에 이를 예방하고 방지하는 것에 도움이 될 것이라고 예측됩니다.

3. 개발 계획

데이터는 수업시간에 학습했던 내용 위주로 사용을 할 것이며 사용된 데이터는 For문, def 함수, if else문, return문, 등을 사용하였습니다.

머신러닝은 저희가 수업시간에 배웠던 회귀분석을 위주로 사용하였으며, 특히 LSTM(Long Short-Term Memory) 네트워크를 이용한 시계열 회귀분석을 사용할 계획입니다.

그리고 이 프로그램을 통해 그래프와 수치를 도출해내, 이를 분석하고 서술하는 방식으로 문제를 해결해 나가도록 구상을 했습니다.

프로그램을 완성한 후 성능을 확인해보기 위해서 프로그램을 실행하고 그래프를 분석하여 실제 사고사례를 검색하거나 데이터시트와 일치하는지 대조해봤습니다. 또 입력한 수치에 맞게 그래프가 형성되는 지도 확인하여 오류와 에러를 보완했습니다.

4. 개발내용 및 과정

첫번째 과정 - 기본적인 저장 및 데이터 시트 분석

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import pandas as pd
data =  pd.read_csv("./data/sensor.csv")

data.shape

data.columns

data.describe().transpose()

data.isnull().sum()

data.drop( labels: ['Unnamed: 0', 'timestamp','sensor_00','sensor_15','sensor_50','sensor_51'],axis=1, inplace=True)
```

위 코드를 사용하여 파일 경로를 지정해주고 csv 파일을 data라는 데이터프레임에 저장을 해줍니다. 그리고 describe()와 transpose()를 이용하여 기술 통계를 파악합니다. 마지

막으로 drop()을 이용해 불필요한 칼럼을 제거해줍니다. 그리고 csv파일을 분석 하기위해 시트를 분석해면

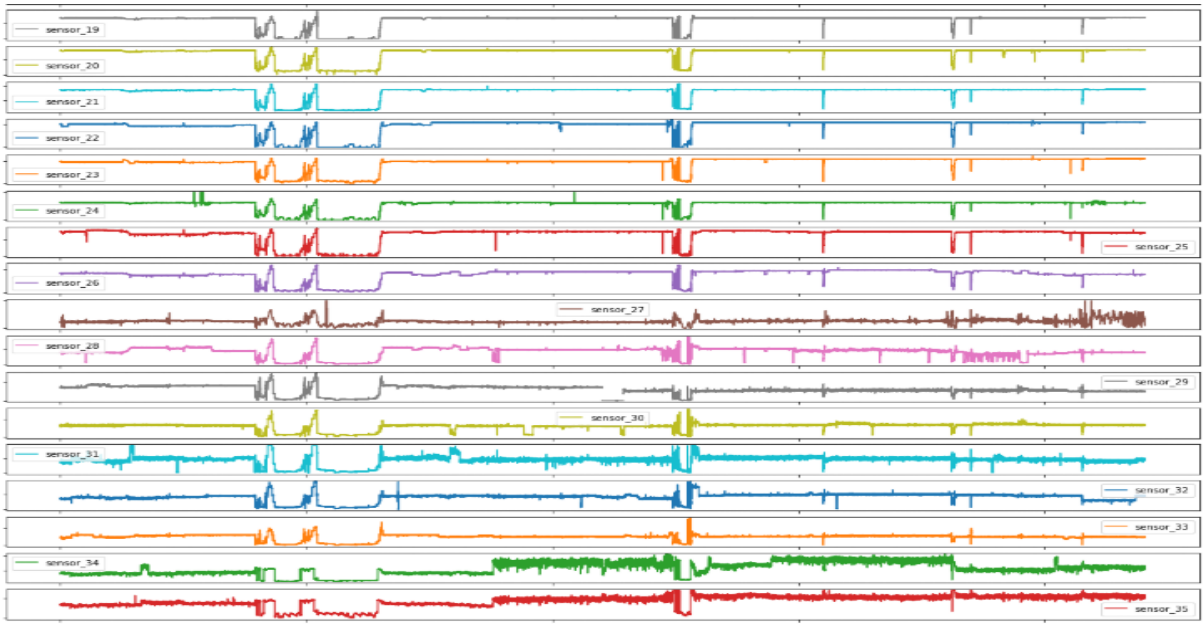
데이터 시트의 저자는 시스템에 1년 동안 7건의 시스템 장애가 발생하여 문제가 발생했다고 나와있습니다. 따라서 문제는 다음 장애가 언제 발생할지 예측하는 것이 매우 중요합니다. 이를 예방하기 위해 사례들을 분석해보면

Unnamed: 0	220320.0	110159.500000	63601.049991	0.000000	55079.750000	110159.500000	165239.250000	220319.000000
sensor_00	210112.0	2.372221	0.412227	0.000000	2.438831	2.456539	2.499826	2.549016
sensor_01	219951.0	47.591611	3.296666	0.000000	46.310760	48.133678	49.479160	56.727430
sensor_02	220301.0	50.867392	3.666820	33.159720	50.390620	51.649300	52.777770	56.032990
sensor_03	220301.0	43.752481	2.418887	31.640620	42.838539	44.227428	45.312500	48.220490
sensor_04	220301.0	590.673936	144.023912	2.798032	626.620400	632.638916	637.615723	800.000000
sensor_05	220301.0	73.396414	17.298247	0.000000	69.976260	75.576790	80.912150	99.999880
sensor_06	215522.0	13.501537	2.163736	0.014468	13.346350	13.642940	14.539930	22.251160
sensor_07	214869.0	15.843152	2.201155	0.000000	15.907120	16.167530	16.427950	23.596640
sensor_08	215213.0	15.200721	2.037390	0.028935	15.183740	15.494790	15.697340	24.348960
sensor_09	215725.0	14.799210	2.091963	0.000000	15.053530	15.082470	15.118630	25.000000
sensor_10	220301.0	41.470339	12.093519	0.000000	40.705260	44.291340	47.463760	76.106860
sensor_11	220301.0	41.918319	13.056425	0.000000	38.856420	45.363140	49.656540	60.000000
sensor_12	220301.0	29.136975	10.113935	0.000000	28.686810	32.515830	34.939730	45.000000
sensor_13	220301.0	7.078858	6.901755	0.000000	1.538516	2.929809	12.859520	31.187550
sensor_14	220299.0	376.860041	113.206382	32.409550	418.103250	420.106200	420.997100	500.000000
sensor_15	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
sensor_16	220289.0	416.472892	126.072642	0.000000	459.453400	462.856100	464.302700	739.741500
sensor_17	220274.0	421.127517	129.156175	0.000000	454.138825	462.020250	466.857075	599.999939
sensor_18	220274.0	2.303785	0.765883	0.000000	2.447542	2.533704	2.587682	4.873250
sensor_19	220304.0	590.829775	199.345820	0.000000	662.768975	665.672400	667.146700	878.917900
sensor_20	220304.0	360.805165	101.974118	0.000000	398.021500	399.367000	400.088400	448.907900
sensor_21	220304.0	796.225942	226.679317	95.527660	875.464400	879.697600	882.129900	1107.526000
sensor_22	220279.0	459.792815	154.528337	0.000000	478.962600	531.855900	534.254850	594.061100
sensor_23	220304.0	922.609264	291.835280	0.000000	950.922400	981.925000	1090.808000	1227.564000
sensor_24	220304.0	556.235397	182.297979	0.000000	601.151050	625.873500	628.607725	1000.000000
sensor_25	220284.0	649.144799	220.865166	0.000000	693.957800	740.203500	750.357125	839.575000

사진에는 다 못담았지만 데이터 시트에 따르면 220320이 기록된 55개의 열이 있습니다. 또한 측정값의 척도는 위 그림과 같이 모두 다릅니다.

먼저 sensor_15의 NaN 열을 제외하고 모든 열의 표준 값이 0입니다.

실제 작동에서 펌프 시스템에는 압력, 유량 또는 온도와 같은 매개변수에 대해 두 개 이상의 센서가 장착되는 경우가 많습니다. (이는 다음과 같이 분석되는 일부 센서의 측정 신호가 겹치는 원인이 될 수 있습니다)



센서가 패턴을 분석한 후 위 그림처럼 그래프로 나타낸 예시입니다.
위 그래프에서 알 수 있듯이 일반적인 상황에는 그래프의 파동이 거의 일정한 모습을 확인할 수 있습니다.

이 분석을 기반으로 펌프의 작동 상태에 영향을 미치는 센서 신호를 결정하는 것은 모델링이 큰 영향을 끼치는 것을 알 수 있습니다. 모델을 최적화하기 위해 다음 가설에 따라 모델에 입력할 매개 변수를 선택합니다.

```
data['machine_status'].value_counts()
```

이 코드를 넣어줌으로써 확인할 수 있는 결과는 데이터 시트에 기반하여 7개의 파손이 발생한 것이 있음을 알 수 있었습니다. 예시를 위해 고장 상태 전이 값은 0, 복구 상태 및 정상 작동 값은 각각 0.5와 1이라고 가정하고 이를 " Operation"이라는 이름의 새 열로 변환합니다.

```
import numpy as np
conditions = [(data['machine_status'] =='NORMAL'), (data['machine_status'] =='BROKEN'), (data['machine_status'] =='RECOVERING')]
choices = [1, 0, 0.5]
data['Operation'] = np.select(conditions, choices, default=0)

import matplotlib.pyplot as plt
data.plot(subplots =True, sharex = True, figsize = (20,50))
```

이 코드는 고장이 났을 때의 명확한 패턴이 있었는지, 만약 있었다면 그래프의 파동에 표시하고 'Operation'이라고 기입하는 코드입니다.

두번째 과정 – 불필요한 데이터 삭제

위에서 분석한 바와 같이 많은 측정값들은 모두 유사한 형태를 띄고있습니다. 그렇기에 유의해야 할 부분은 유지하고 나머지는 삭제하는 코드를 삽입했습니다.

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):  1개의 사용 위치  신규 *
    n_vars = 1 if type(data) is list else data.shape[1]
    dff = pd.DataFrame(data)
    cols, names = list(), list()
    for i in range(n_in, 0, -1):
        cols.append(dff.shift(-i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    for i in range(0, n_out):
        cols.append(dff.shift(-i))
        if i==0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1)) for j in range(n_vars)]
        agg = pd.concat(cols, axis=1)
        agg.columns = names
        if dropnan:
            agg.dropna(inplace=True)
        return agg
```

수업시간에 배웠던 def, for, if, else 문을 이용하여 필요없는 부분을 삭제하는 코드를 형성했습니다.

세번째 과정 – 모델학습과 예측을 구현하는 코드 삽입

```
from sklearn.preprocessing import MinMaxScaler

values = df.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
reframed = series_to_supervised(scaled,  n_in: 1,  n_out: 1)
r = list(range(df.shape[1]+1, 2*df.shape[1]))
reframed.drop(reframed.columns[r], axis=1, inplace=True)
reframed.head()

values = reframed.values
n_train_time = 50000
train = values[:n_train_time, :]
test = values[n_train_time:, :]
train_x, train_y = train[:, :-1], train[:, -1]
test_x, test_y = test[:, :-1], test[:, -1]
train_x = train_x.reshape((train_x.shape[0], 1, train_x.shape[1]))
test_x = test_x.reshape((test_x.shape[0], 1, test_x.shape[1]))
```

위 코드는 데이터를 정규화하고, 시계열 데이터 형식으로 변환한 후, 훈련 데이터와 테스

트 데이터를 나누어 LSTM 모델에 맞는 입력 형식으로 변환하는 내용을 담은 코드입니다. 위 코드를 통해 LSTM 모델이 시계열 데이터를 학습할 수 있게 됩니다.

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Dense
from sklearn.metrics import mean_squared_error,r2_score
import matplotlib.pyplot as plt
import numpy as np

model = Sequential()
model.add(LSTM( units: 100, input_shape=(train_x.shape[1], train_x.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(train_x, train_y, epochs=50, batch_size=70, validation_data=(test_x, test_y), verbose=2, shuffle=False)

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend( *args: ['train', 'test'], loc='upper right')
plt.show()
```

```
size = df.shape[1]

yhat = model.predict(test_x)
test_x = test_x.reshape((test_x.shape[0], size))

inv_yhat = np.concatenate( arrays: (yhat, test_x[:, 1-size:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

test_y = test_y.reshape((len(test_y), 1))
inv_y = np.concatenate( arrays: (test_y, test_x[:, 1-size:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

rmse = np.sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

위 코드는 LSTM 모델을 훈련하고 예측을 수행한 후, 예측 성능을 평가하는 전반적인 흐름을 나타냅니다. 주요 단계는 모델 훈련, 예측 및 RMSE 평가로 요약할 수 있습니다. RMSE는 모델 성능을 평가하는 중요한 지표로, 값이 작을수록 모델의 예측이 정확하다는 것을 의미합니다. 또, 이는 이후 결론에 있을 그래프를 도출하는 코드 중 하나입니다.

* 저는 이 코드를 작성할 때 'keras'를 설치하지 않아 오류가 발생하였습니다. 그렇기에 'keras'파일을 설치한 후 재실행 하니 오류가 해결되었습니다.

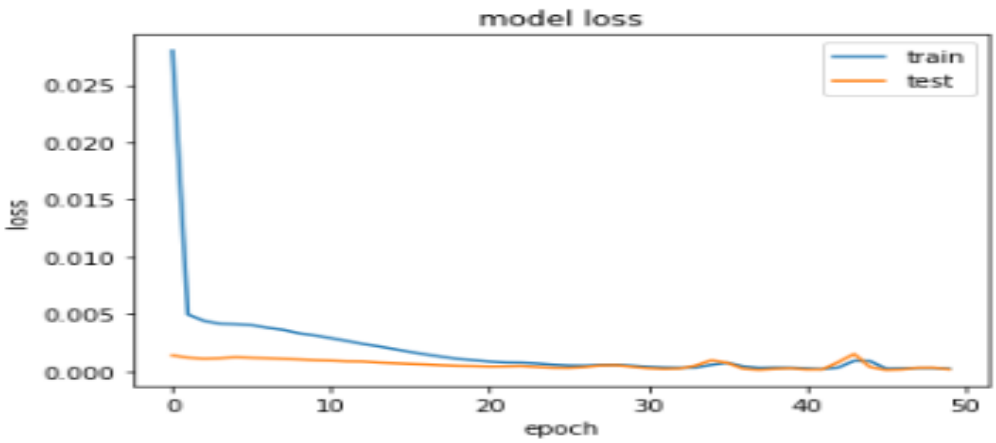
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend( *args: ['train', 'test'], loc='upper right')
plt.show()
```

위 코드는 손실 값을 시각화 하여 그래프를 출력하는 역할을 합니다.
또 값이 지속적을 감소하는지 등의 모델훈련을 하여 모델이 일반화되는 것을 확인하는 역할을 하고있습니다.

4. 개발 결과

그래프는 두가지 선으로 모델의 예측 값을 비교합니다. 만약 두 곡선 사이의 차이가 거의 없다면 모델은 매우 정확히 예측하고 있으며, 데이터에 대한 적합도가 높다고 결론지을 수 있습니다.

1번 그래프

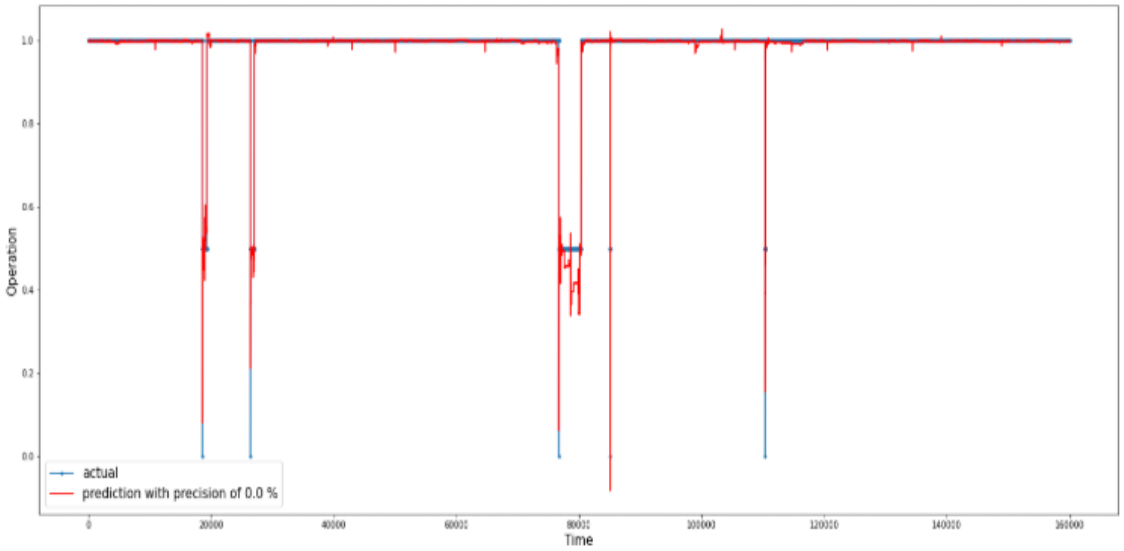


첫번째 코드에서 도출해낸 그래프입니다. Test RMSE는 0.013이라는 값을 도출해냈습니다.

그리고 그래프가 거의 맞물리는 모습을 보면 이 프로그램은 현재 매우

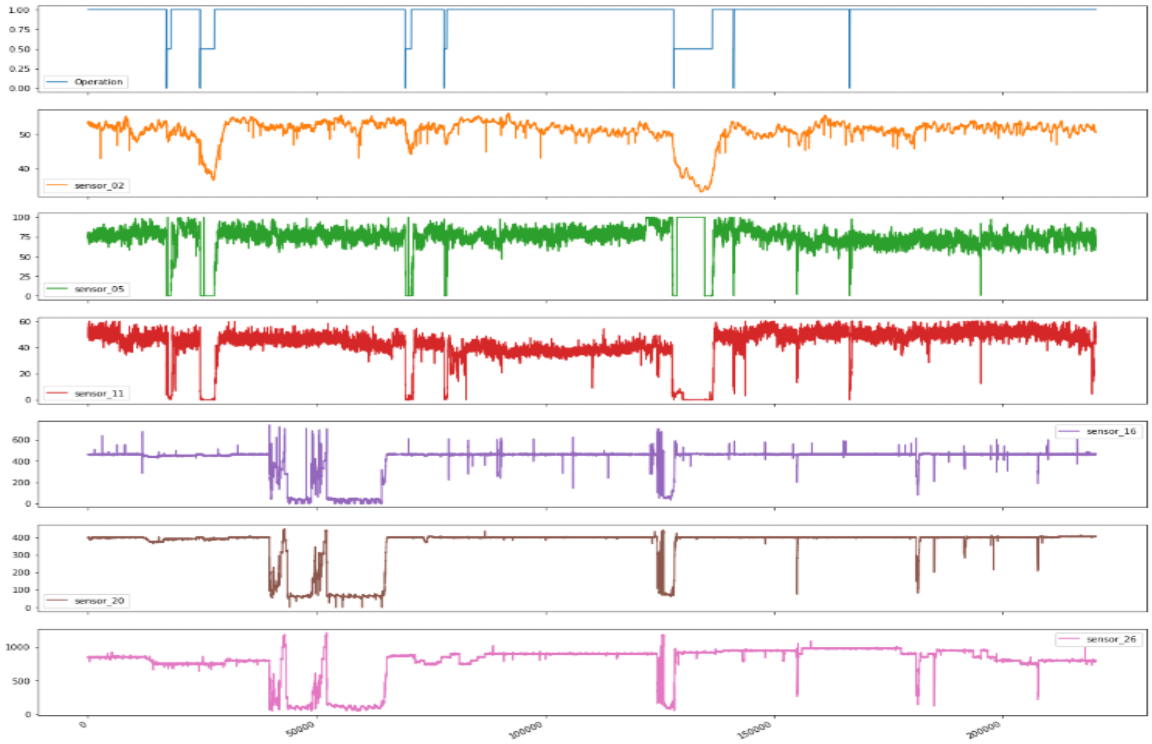
정확하게 예측하고 있는 것을 알 수 있습니다.

2번 그래프



두번째 코드에서 도출해낸 그래프입니다. 이 그래프도 마찬가지로 평상시 값은 동일한 값으로 평범하다는 것을 알 수 있지만 가끔 시간대에 짧게 오차를 보입니다. 그렇기에 이 시간대에 상황을 분석하여 문제를 파악해볼 수 있습니다.

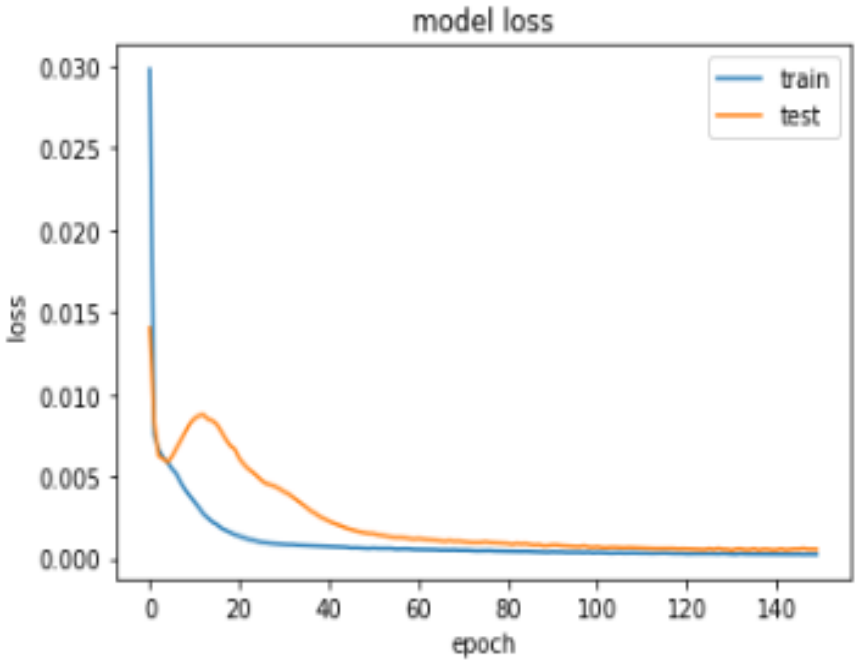
3번 그래프



위 그래프는 아까 문제가 있었던 7개의 sensor 부분을 도출하여 작동 문제를 파악한 그래프입니다. 이를 보고 정상적인 sensor와 비교하여 문제를 해결해 나갈 수 있습니다.

예를 들어, sensor_16이나 sensor_26 같은 센서에서 값이 안정적인 구간과 불안정한 구간이 명확히 구분되어 보입니다. 이는 데이터 수집 오류나 시스템의 동작 변화로 인한 것이라고 예측할 수 있습니다. 이를 통해 도출해낸 sensor들을 교체하거나 문제, 결함을 수리할 수 있습니다.

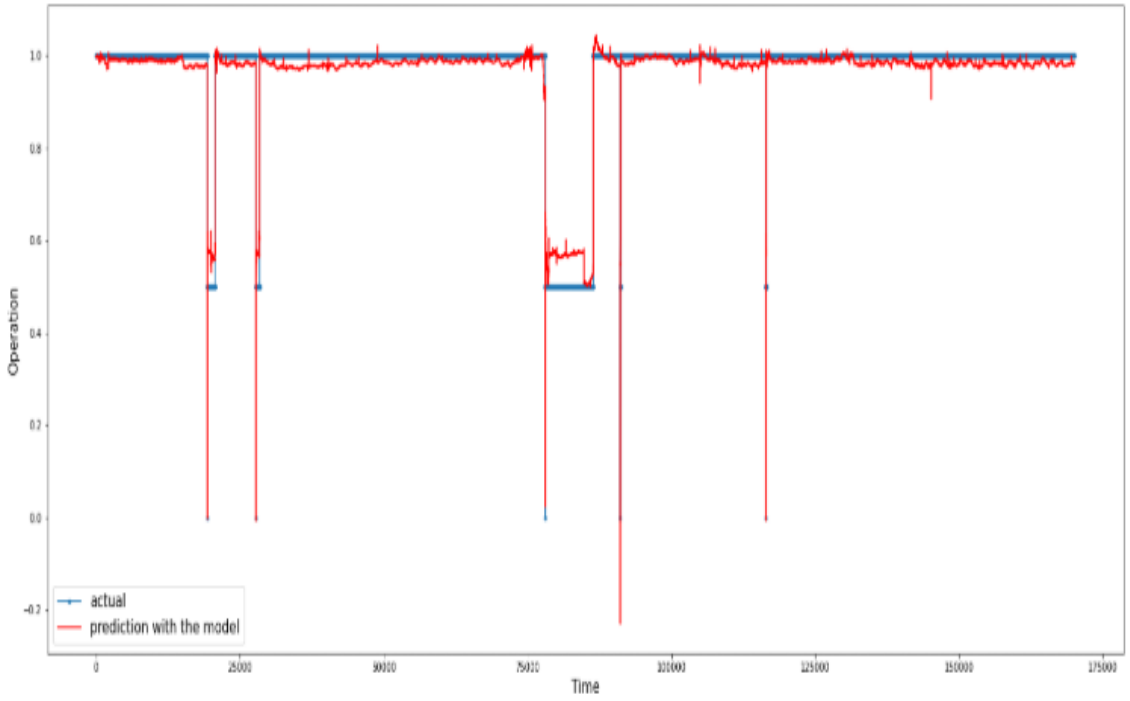
4번 그래프



이 그래프를 통해서는 Test RMSE값이 0.023인 것을 알 수 있습니다.

1번 그래프와 마찬가지로 매우 프로그램이 예측을 매우 정확하게 하고 있음을 알 수 있으며 오차가 있는 부분은 데이터 시트를 확인하여 수정하거나 교체하는 조치를 취한다.

5번 그래프



위 그래프도 결과 값과 예측 값이 매우 동일한 형태를 띄고 있는데 값이 많이 다른 경우는 데이터 시트를 분석하여 문제를 해결하고 값을 수정합니다.

이 프로그램을 토대로 어떤 문제가 있는지 센서가 분석하여 이상이 감지된 부분을 교체 수리할 수 있는 프로그램을 형성했습니다.

5. 한계와 향후 개선사항

이 프로그램은 현재 워터펌프에 문제가 발생했을 시점을 알려주고 있습니다. 하지만 어떤 문제 발생했는지는 상세히 알려주지 못하기에 약간의 보완점이 있다고 생각했습니다. 워터펌프는 온도, 습도, 파손, 동결 등 수많은 고장 요인이 존재합니다. 그렇기에 센서가 측정하는 범위에 이러한 요인들을 추가시켜 고장 요인까지 한 번에 나타내는 프로그램을 만들면 더욱 유익하고 도움이 될 것이라고 생각했습니다.

6. 느낀점

실제 일상생활에 사용되는 코드들의 프로그래밍을 직접 타이핑 해보고 어떻게 구성 되어

있는지를 고민하고 분석해봤는데, 가장 크게 배운 것은 "세상에 내가 모르는게 너무 많구나"였습니다. 제가 집에서 키보드로 잠깐 해도 이런 프로그램이 나오는데 전문가들이 심여를 기울여서 프로그래밍을 하면 얼마나 대단한 프로그램들이 나올까 같은 컴퓨터에 대한 지식이 부족해서 그런 생각이 들기도 하였고, 대한민국 IT분야가 이렇게까지 발전을 많이 했는데 지금까지 이를 느끼지 못하고 둔감하게 살았던 제 스스로의 무지 때문에도 느끼는 점이 많았던 바였습니다. 실제로 많은 분야에서 프로그래밍이 사용되고 있을 것이고 컴퓨터가 세상을 지배한다는 말이 나올 정도로 IT분야는 기하급수적으로 발달하고 있다고 생각했습니다. 그렇기에 제가 몸담고 있는 안전관련 분야에도 이러한 프로그래밍을 접목한다면 더욱 경쟁력을 갖출 수 있겠다는 생각이 들었습니다. 사실 요즘 졸업시기가 되면 자격증이나 토익이나 다들 비슷한 스펙을 갖춘다고 알고 있습니다. 그래서 저는 이들과 다른 경쟁력을 갖추려면 어떤 것을 해야할 지 고민에 빠진 상태였습니다. 그래서 색다른 자격증을 알아본다거나 선배들에게 조언을 많이 구하고 있었는데 이번 과제를 하며 프로그래밍에 대한 접근성을 기른 것 같아서 굉장히 뿌듯했습니다. 안 될 것만 같던 프로그래밍이 막상 해보니 결과물을 도출해냈고, 다양한 방면에서 이를 활용할 수 있을 것 같다는 자신감도 자연스레 얻게 되어서 제 앞으로의 삶에 시야를 넓힌 것 같아 굉장히 유익했습니다.