- 머신러닝 활용 프로젝트 -

공정 데이터를 활용한
설비 오류 발생 예측 시스템

2023. 06. 08.

충북대학교 대학원 산업인공지능학과
2023254018 김연지

요 약

최근 4차 산업혁명이 도래되며 각 기업에서는 제조 시간의 극대화와 수리비용의 최소화를 최신 기술을 통해 예방하고자 한다. 특히, 설비 및 장비가 고장 나기 전 고장을 예측하여 유지 보수함으로써 비용을 절감하고 운영 효율성을 극대화 할 수 있는 예지보전에 대한 관심이 높아지고 있다.

설비 예지보전이란 설비의 이상 상태를 감지하고, 이것이 설비의 결함에 의해 발생한 것인지 혹은 일시적 과도현상인지, 또는 설비 자체의 문제인지 등을 진단하여 파악한 후, 어떻게 진전되어 고장으로 이어질 것인가를 예측하는 설비 관리 솔루션으로, 더 나아가서는 이 문제를 해결하기 위해 어떤 작업을 실시하여 어떻게 완료할 것인가를 결정하는 예지보전 통합시스템으로 확대될 수 있다.

본 프로젝트에서는 식품 제조 공장의 열풍건조기에서 발생하는 시계열 데이터(시간 변화에 따른 설비 온도 및 전류 측정 데이터)를 5초 간격으로 수집하고, 설비 오류 발생 데이터와 함께 분석(딥러닝)을 진행하여 설비의 오류 발생을 예측하는 분석 모델을 만들었다.

측정 데이터가 시간에 따라 변화하는 시계열 데이터이므로, 딥러닝 분석 모델 중 LSTM(Long Short Term Memory) 기반의 분석 모델을 적용하였고, 설비 오류가 드물게 발생하여 정상 데이터가 대부분이었기 때문에 정상 데이터가 많을 시에 효과적인 Auto-Encoder를 활용하였다.

분석 모델의 학습 후, 성능 평가를 위하여 오차 행렬 작성 및 정확도, 정밀도, 재현율, F1 score, ROC curve 및 AUC를 계산 하였고, 그 결과 ROC_AUC = 0.972로 우수한 결과를 보였다.

따라서, 해당 설비의 예지보전을 위하여 위에서 제시한 분석모델을 적용할 시에 오류 발생 예측에 있어 효과가 있을 거라 판단되며, 본 실험의 결과를 통해 타 제조 설비들의 고장 진단에도 도움을 줄 수 있을 것이라 예상된다.

키워드: 딥러닝, 기계학습, 예지 보전, LSTM, 오토인코더
Keyword: Deep Learning, Machine learning, Predictive Maintenance, LSTM(Long Short Term Memory), Auto-Encoder

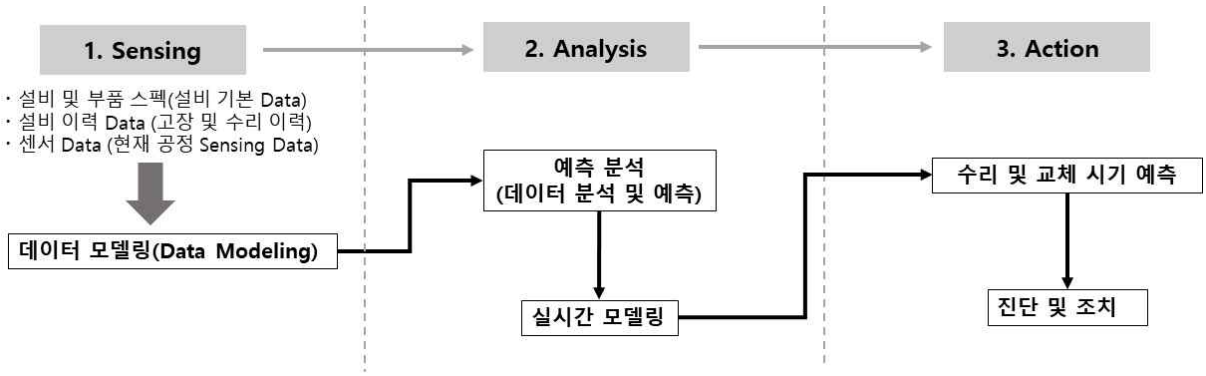
Ⅰ. 서 론

최근 4차 산업혁명이 도래되며 다양한 산업 분야에서 실시간 공정 및 설비 상태에 대한 모니터링이 가능해졌고, 각 기업들에서는 예기치 못한 설비 정비로 인한 유휴 시간의 최소화와 다운타임(기계나 시스템의 고장으로 운용될 수 없는 시간)의 사전 예방을 필요로 하고 있다.

제품을 생산하는 설비의 고장이나 오류 발생은 곧 제품의 결함 및 생산라인 가동 중단으로 이어져 제조업체의 막대한 경제적 손실의 원인이 되며, 또한 생산 설비 자체가 제조 기업 자산의 큰 부분을 차지하기 때문에 큰 손실을 초래할 수 있다.

최근 스마트 팩토리 서비스의 확산으로 공장에서 많은 양의 데이터가 수집됨에 따라, 이를 활용하여 제조 현장의 효율이나 제조 설비의 고장 예측 및 진단을 위한 인공지능 기반의 연구가 활발히 이어지고 있다

공장에서는 제조 시간의 극대화와 수리비용의 최소화를 최신 기술을 통해 예방하고자 하며, 특히, 설비 및 장비가 고장 나기 전 고장을 예측하여 유지 보수함으로써 비용을 절감하고 운영 효율성을 극대화 할 수 있는 예지보전에 대한 관심이 높아지고 있다.



[그림] 예지보전의 단계별 정의

설비 예지보전이란 설비의 이상 상태를 감지하고, 이것이 설비의 결함에 의해 발생한 것인지 혹은 일시적 과도현상인지, 또는 설비 자체의 문제인지 등을 진단하여 파악한 후, 어떻게 진전되어 고장으로 이어질 것인가를 예측하는 설비 관리 솔루션으로, 더 나아가서는 이 문제를 해결하기 위해 어떤 작업을 실시하여 어떻게 완료할 것인가를 결정하는 예지보전 통합시스템으로 확대될 수 있다.

이러한 이상탐지 모델의 개발은 수집된 제조 데이터의 특성에 크게 좌우되며, 일반적으로 제조 설비 데이터는 정상 상태 데이터에 비해 비정상 상태 데이터의 수가 현저히 부족하다. 뿐만 아니라, 제조 데이터에는 생산설비의 정상 상태와 비정상 상태를 구분하는 레이블 정보가 포함되어 있지 않은 경우가 대부분이기 때문에, 이 경우에는 지도학습(Supervised Learning)의 분류 모델을 통한 이상치 판별보다 오토인코더(Auto-encoder)와 같은 비지도학습(Unsupervised Learning) 모델을 통한 방식이 더욱

적합하다.

본 프로젝트에서는 식품 제조 공장의 열풍건조기에서 발생하는 시계열 데이터(시간 변화에 따른 설비 온도 및 전류 측정 데이터)를 5초 간격으로 수집하였고, 정상 상태와 비정상 상태를 구분하는 레이블 정보인 설비 오류 발생 데이터도 함께 수집하였다.

설비의 센싱 데이터(온도 및 전류 데이터)에 맞추어 오류 발생 데이터의 레이블을 병합하였고, 센싱 데이터를 Feature /오류 발생 데이터를 Label로 지정하여 설비의 오류 발생을 예측하는 딥러닝 분석 모델(LSTM-AE)을 구축하였다.

온도와 전류 측정 데이터는 시간에 따라 변화하는 시계열 데이터이고, 36번의 측정을 기준으로 프로세스가 변경되는 형태이므로, 36개의 데이터를 주기로 하는 시퀀스를 생성하여, LSTM(Long Short Term Memory) 기반의 분석 모델을 적용 하였으며, 설비 오류가 드물게 발생하여 정상 데이터가 대부분이었기 때문에 정상 데이터가 많을 시에 효과적인 Auto-Encoder를 활용하였다.

본 프로젝트 보고서의 구성은 다음과 같다. 2장에서는 수집된 설비의 제조 데이터에 관한 내용과 이상탐지 알고리즘에 대하여 설명하고 실제 코드로 분석 구현 및 평가 하였으며, 마지막으로 3장에서는 결론 및 향후 실제 적용에 관하여 기술하였다.

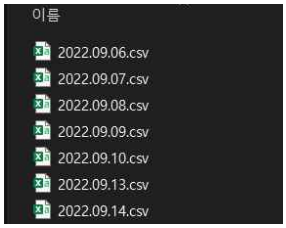
Ⅱ. 관련 연구 및 코드 구현

요 약	
목적	열풍 건조기의 설비 온도와 전류 데이터를 분석하여 설비의 오류 발생 예측
시스템명	공정 데이터를 활용한 설비 오류 발생 예측 시스템
실증 진행기간	2022. 09. 06 ~ 2022. 10. 27
데이터 셋 현황	33개의 csv 파일 * 파일 당 1,548개의 데이터 = 총 51,048개 (Index / Process / Time / Temp(온도) / Current(전류) / Date / Label(0,1))
Feature Data (X)	X1: Temp (온도) / X2: Current(전류)
Label Data (Y)	Y : Label (오류발생시 : 0 / 정상상태 : 1)
데이터 분석 S/W	Anaconda / Python 3.9 / Jupyter notebook
분석 적용 알고리즘	LSTM-AE (Auto-Encoder)
분석 결과	**정확도: 0.946 / 예측률: 1.0 / 재현율: 0.944 / F1 score: 0.971 / ROC_AUC: 0.972**

1. 데이터 개요

- 공정 데이터 : 프로세스(1~43), 날짜, 시간, 공정 온도, 공정 전류
(날짜별로 총 33개의 csv파일, 총 row: 51,084개)

	A	B	C	D	E	F
1	Index	Process	Time	Temp	Current	Date
2	1	1	오후 4:24:03.0	75.13914228	1.61	2022-09-06
3	2	1	오후 4:24:08.0	76.66042142	1.53	2022-09-06
4	3	1	오후 4:24:13.0	77.17766014	1.701	2022-09-06
5	4	1	오후 4:24:18.0	76.58643441	1.736	2022-09-06
6	5	1	오후 4:24:23.0	77.87710396	1.748	2022-09-06
7	6	1	오후 4:24:28.0	76.01353467	1.734	2022-09-06
8	7	1	오후 4:24:33.0	71.00784573	1.763	2022-09-06



[공정 데이터의 csv 파일 내부(왼쪽)와 전체 리스트 중의 일부(오른쪽)]

- 오류 발생 리스트 : 날짜별 설비 오류 발생 프로세스(1~43) 및 에러 유형(1~11)
(1개의 csv 파일)

	A	B	C	D	E	F	G	H	I	J	K	L
1	0	1	2	3	4	5	6	7	8	9	10	11
2	2022-09-06	32	33	20	21	22	31					
3	2022-09-07	32	33	34								
4	2022-09-08											
5	2022-09-09	15	16	17	21	22	23	29	30	31		
6	2022-09-10	32	28	29	30	31						
7	2022-09-13	27	28	29								
8	2022-09-14											

[에러 리스트의 csv 파일 내부(왼쪽)와 해당 파일(오른쪽)]

- 라벨링 작업: 데이터를 간단하게 바꾸기 위해 에러 유형에 관계없이 모든 에러 발생 시, 해당 프로세스는 1로 라벨링하고, 그 외에는 0으로 라벨링하여 수정하도록 함

	A	B	C	D	E	F	G
1	Index	Time	Temp	Current	Date	Label	
677	676	오후 5:26:44.8	66.7404	1.523	2022-09-06	0	
678	677	오후 5:26:49.8	68.0452	1.506	2022-09-06	0	
679	678	오후 5:26:54.8	70.4725	1.511	2022-09-06	0	
680	679	오후 5:26:59.8	64.0078	1.54	2022-09-06	0	
681	680	오후 5:27:04.8	65.1194	1.596	2022-09-06	0	
682	681	오후 5:27:09.8	64.1218	1.448	2022-09-06	0	
683	682	오후 5:27:14.8	66.062	1.473	2022-09-06	0	
684	683	오후 5:27:19.8	65.0865	1.451	2022-09-06	0	
685	684	오후 5:27:24.8	67.2568	1.495	2022-09-06	0	
686	685	오후 5:27:51.3	123.927	0.92035	2022-09-06	1	
687	686	오후 5:27:56.3	122.738	0.91714	2022-09-06	1	
688	687	오후 5:28:01.3	121.825	1.19303	2022-09-06	1	
689	688	오후 5:28:06.3	117.936	1.10014	2022-09-06	1	
690	689	오후 5:28:11.3	125.562	1.11335	2022-09-06	1	

[수집데이터 현황 및 라벨 추가(CSV 파일)]

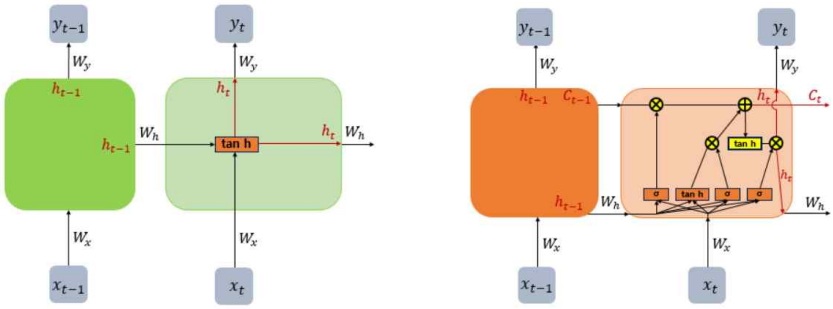
- 전류와 온도를 Feature(X1, X2)로 하고 오류발생을 Label(Y)로 하여, X1, X2와 Y간의 상관관계를 알아보고, 딥러닝을 활용하여 오류 발생을 예측하는 시스템을 만들어보도록 함

Feature		Label
X1	X2	Y
설비 온도 (℃)	설비 전류 (A)	오류 발생 여부 (0: 정상 / 1: 오류발생)

2. 관련 연구

(1) LSTM (Long Short Term Memory)[1)]

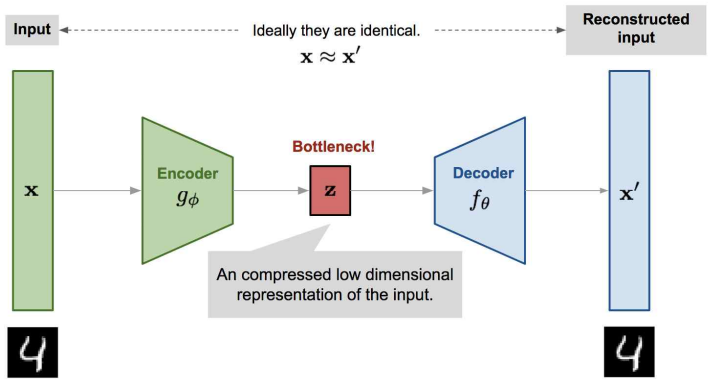
: 기존의 RNN은 출력 결과가 이전의 계산 결과에 의존하지만, 비교적 짧은 시퀀스(sequence)에 대해서만 효과를 보이는 단점이 있으며, 시점(time step)이 길어질수록 앞의 정보가 뒤로 충분히 전달되지 못하는 현상이 발생함(장기 의존성 문제). RNN의 이러한 단점을 보완한 LSTM은 은닉층의 메모리 셀에 입력 게이트, 망각 게이트, 출력 게이트를 추가하여 불필요한 기억을 지우고, 기억해야할 것들을 정함으로써, 은닉 상태(hidden state)를 계산하는 식이 조금 더 복잡해졌으며 셀 상태(cell state)라는 값을 추가하여, 긴 시퀀스의 입력을 처리하는데 탁월한 성능을 보임. 주로 시계열 처리나 자연어 처리에 사용



[RNN 내부구조(왼쪽)와 LSTM의 내부구조(오른쪽)]

(2) Auto-Encoder[2)]

: 오토인코더는(auto encoder)는 라벨이 없는 훈련 데이터를 사용한 학습(즉, 지도 학습) 없이도 입력 데이터의 표현을 효율적으로 학습할 수 있는 인공신경망이다. 오토인코더는 아래 그림과 같이 input 레이어, hidden 레이어, output 레이어로 구성되어 있으며 일반적으로 Input 유닛보다 훨씬 낮은 차원의 hidden 유닛을 가지므로 주로 차원 축소(Dimensionality Reduction) 목적으로 사용된다. 또한 오토인코더는 강력한 feature extractor로 작동하기 때문에 비지도 사전훈련에 사용될 수 있고, 훈련 데이터와 매우 비슷한 새로운 데이터를 생성하는 생성 모델(generative model)로서 사용될 수 있다.



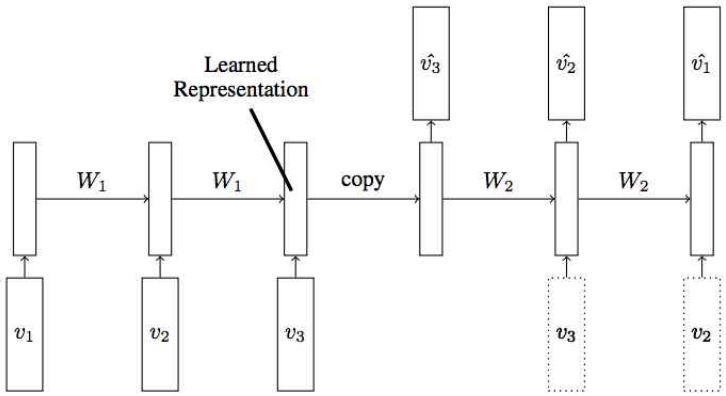
[Auto Encoder 구조]

---

1) 출처: https://wikidocs.net/22888
2) 출처: https://velog.io/@jaehyeong/Autoencoder%EC%99%80-LSTM-Autoencoder

(3) LSTM Auto-Encoder (LSTM-AE)[3)]

: LSTM Autoencoder는 시퀀스(sequence) 데이터에 Encoder-Decoder LSTM 아키텍처를 적용하여 구현한 오토인코더이다. 아래 그림은 LSTM 오토인코더의 구조이며 입력 시퀀스가 순차적으로 들어오게 되고, 마지막 입력 시퀀스가 들어온 후 디코더는 입력 시퀀스를 재생성하거나 혹은 목표 시퀀스에 대한 예측을 출력한다.



[LSTM Auto Encoder]

① Reconstruction LSTM Autoencoder

재구성(reconstruction)을 위한 LSTM Autoencoder 구조이다. 즉, input과 최대한 유사하게 output을 디코딩하며, LSTM 학습을 위해 데이터를 우선 (samples, timesteps, feature)와 같은 3d형태로 변환한다.

② Prediction LSTM Autoencoder

시계열적 예측을 위한 LSTM 구조이며 input 시퀀스는 현재 시점(t) output 시점은 (t+1)로 두어 한 시점 앞을 학습하도록 데이터를 구성한다. 여기서 autoencoder는 학습 시 encoder에는 t 시점이 입력되지만 decoding 후에는 (t+1)시점과 reconstruction error를 계산하며 결국 t 시점이 t+1 시점을 학습하게 된다.

결과적으로 예측 결과는 1이 입력되면 2와 가까운 수를, 2가 입력되면 3과 가까운 수를 예측하게 된다.

③ Composite LSTM Autoencoder

Reconstruction과 Prediction 모델을 통합한 모델이다. 결과적으로 출력 시 reconstruction결과와 prediction결과가 함께 출력된다.

---

3) 출처: https://velog.io/@jaehyeong/Autoencoder%EC%99%80-LSTM-Autoencoder

3. LSTM-AE 데이터 분석 진행

(1) 요약

개발환경 및 사용언어	Windows10 / Python 3.9 (Anaconda - Jupyter notebook)	
사용 패키지 (라이브러리)	os	운영체제에서 제공되는 여러 기능을 Python에서 사용
	glob	파일들의 경로를 가져와 이를 리스트로 반환
	pandas	다양한 데이터 분석 기능을 수행하는 패키지
	numpy	행렬 및 다차원 배열 핸들링, 수학적 계산에 활용
	matplotlib.pyplot	데이터 시각화에 활용 (데이터 분포 및 결과 그래프 작성)
	seaborn	데이터 시각화에 활용 (Heatmap 작성 등)
	sklearn.preprocessing	데이터 전처리에 활용 (데이터 정규화)
	sklearn.metrix	confusion matrix를 활용한 결과 분석 (ROC 커브 등)
	tensorflow/keras	딥러닝 모델링(학습)에 활용
분석 적용 알고리즘	LSTM-AE	과거의 데이터를 현재 학습에 반영하며, 시계열 데이터(Sequential data) 학습에 주로 사용함. 정상데이터가 많을 경우 AE(Auto Encoder) 방식을 적용)
분석 결과	정확도: 0.946 / 예측률: 1.0 / 재현율: 0.944 / F1 score: 0.971 / ROC_AUC: 0.972	

(2) 소스코드 및 결과

① 라이브러리 설치(install) 및 불러오기(import)

```
import pandas as pd
import numpy as np
import os
import glob
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import *
```

```
import keras
import tensorflow
from keras.models import Sequential, Model
from keras.layers import Dense
from keras.layers import LSTM
from keras import backend as K
from keras.layers import *
from keras.optimizers import Adam
```

[사용할 패키지 및 클래스 불러오기(설치 생략)]

② 데이터 불러오기(os/glob) 및 데이터 병합(pandas)

```
new_path=os.path.join(os.getcwd(),'data')

pathName=os.path.join(new_path,'Dataset/라벨수정')
pathName
```

```
dataList=list()

for pth in glob.glob(pathName+'/*')[1:]:
    origin = pd.read_csv(pth,index_col=False)
    dataList.append(origin)

CatList=pd.concat(dataList,axis=0,ignore_index=True)
CatList
```

	Index	Process	Time	Temp	Current	Date	Label
0	1	1	오후 4:24:03.0	75.139142	1.610	2022-09-06	0
1	2	1	오후 4:24:08.0	76.660421	1.530	2022-09-06	0
2	3	1	오후 4:24:13.0	77.177660	1.701	2022-09-06	0
3	4	1	오후 4:24:18.0	76.586434	1.736	2022-09-06	0
4	5	1	오후 4:24:23.0	77.877104	1.748	2022-09-06	0
...	...	...	...	...	...	...	...
51079	1544	43	오후 6:45:03.4	67.312474	1.560	2022-10-27	0
51080	1545	43	오후 6:45:08.4	65.533664	1.472	2022-10-27	0
51081	1546	43	오후 6:45:13.4	66.740424	1.523	2022-10-27	0
51082	1547	43	오후 6:45:18.4	68.045185	1.483	2022-10-27	0
51083	1548	43	오후 6:45:23.4	68.472491	1.511	2022-10-27	0

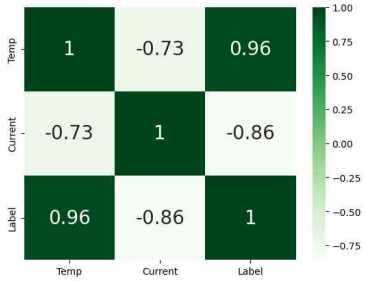
51084 rows × 9 columns

[현재 경로 확인 및 csv파일 불러오기, 데이터 병합(왼쪽) / 결과화면(오른쪽)]

③ 상관관계 파악 : 온도-전류-라벨 간의 상관관계를 확인하고 시각화

```
df = df.drop(['Index','Process','Time','Date'], axis=1)
df
```

```
corr=df.corr()

sns.heatmap(corr, annot=True, cmap='Greens', annot_kws={'size':20})
plt.show()
```



[Pandas와 Seaborn(heatmap)을 활용한 상관관계 시각화(왼쪽) / 결과화면(오른쪽)]

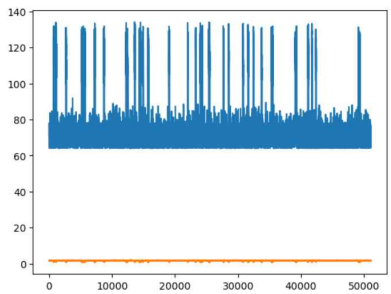
데이터	상관도	비고
온도 - 라벨 간의 상관성	0.96	(매우 높은 상관성) 오류 발생에 있어 온도의 영향이 매우 큼
전류 - 라벨 간의 상관성	-0.73	(높은 상관성) 오류 발생에 있어 전류의 영향이 다소 큼
온도 - 전류 간의 상관성	-0.86	(높은 상관성) 온도와 전류 사이에도 상관성이 큼

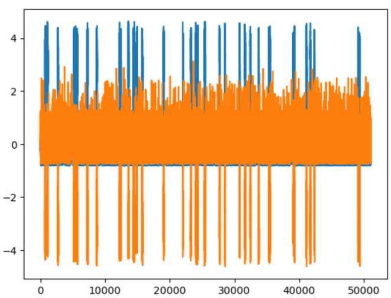
④ 데이터 전처리 (sklearn)

- 데이터 정규화

```
scaler = StandardScaler()
df_data = df.drop(['Label'], axis=1)
df_scaled = scaler.fit_transform(df_data)
```

[StandardScaler()를 활용한 데이터 정규화]






```
array([[ 0.04834502,  0.27659986],
       [ 0.16649822, -0.23033417],
       [ 0.20667061,  0.85323733],
       ...,
       [-0.60395836, -0.2746909 ],
       [-0.50262144, -0.52815792],
       [-0.46943391, -0.35073101]])
```

[정규화를 진행전의 데이터분포(왼쪽) / 정규화 진행 후의 데이터 분포(가운데) / 정규화 된 데이터(오른쪽)]

: 온도 데이터(파란색)와 전류데이터(주황색)의 분포가 광범위하므로, 평균과 분산을 각각 0과 1로 조정하는 정규화를 진행하여. 데이터 분포를 축소시킨 후 학습을 진행하도록 함

- 데이터 분리(학습용 : 약 35600개 / 평가용 : 약 15200개) 및 오토인코더 적용을 위해 학습용 데이터에서 정상 데이터만을 분리(32328개)

```
df_scale = pd.DataFrame(df_scaled, columns=['Temp', 'Current'])
df_scale['Label'] = df['Label']

train_data=df_scale[:35604]
test_data=df_scale[35604:]
train_data.info()

ng_idx_train=train_data[train_data['Label']==1].index
ok_idx_train=train_data[train_data['Label']==0].index
ok_train=train_data.loc[ok_idx_train]
ng_train=train_data.loc[ng_idx_train]
```

	Temp	Current	Label
0	0.048345	0.276600	0
1	0.166498	-0.230334	0
2	0.206671	0.853237	0
3	0.160752	1.075021	0
4	0.260994	1.151061	0
...	...	...	...
35599	-0.532505	-0.249344	0
35600	-0.492333	-0.300038	0
35601	-0.460584	-0.217661	0
35602	-0.671010	-0.147957	0
35603	-0.582747	-0.090927	0

32328 rows × 3 columns

[학습용, 평가용의 데이터 분리 및 정상 데이터만 분리(왼쪽) / 결과화면(오른쪽)]

⑤ 데이터 준비

- 학습용 데이터 준비 (LSTM-AE(Auto Encoder) 입력용 시퀀스 데이터 생성)

: 시퀀스 데이터 생성을 위한 함수를 만들고, 학습용 정상 데이터들을 함수에 넣어 LSTM-AE 입력을 위한 시퀀스 데이터를 생성함

```
TIME_STEP=36
def sequences(x,y,time_steps=TIME_STEP):
    xs,ys=[],[]
    for i in range(len(x)-time_steps):
        xs.append(x.iloc[i:(i+time_steps)].values)
        ys.append(y.iloc[i:(i+time_steps)].values)

    return np.array(xs), np.array(ys)

X_train, Y_train = sequences(ok_train[['Temp','Current']],ok_train[['Temp','Current']])
```

```
array([[[ 0.04834502,  0.27659986],
        [ 0.16649822, -0.23033417],
        [ 0.20667061,  0.85323733],
        ...,
        [-0.60395836, -0.41409776],
        [-0.50262144, -0.52815792],
        [-0.54710092, -0.35706768]],

       [[ 0.16649822, -0.23033417],
        [ 0.20667061,  0.85323733],
        [ 0.16075187,  1.07502097],
        ...,
        [-0.50262144, -0.52815792],
        [-0.54710092, -0.35706768],
        [ 0.05338504, -0.16696742]],
```

[시퀀스 생성함수 및 학습용 데이터의 시퀀스화(왼쪽) / 학습용 데이터의 시퀀스화 결과(오른쪽)]

- 평가용 데이터 준비 (LSTM-AE(Auto Encoder) 입력용 시퀀스 데이터 생성)

: 시퀀스 데이터 생성 함수에 평가용 데이터(정상/비정상 데이터 모두)를 넣어 LSTM-AE 입력을 위한 시퀀스 데이터를 생성

⑥ 모델링(LSTM-AE)

- LSTM-AE 모델 구조는 2개의 Encoder 계층과 Decoder 계층으로 구성.
  Activation Function은 ReLU로 설정
  input 레이어의 feature는 2차원이므로, output 레이어도 동일한 차원으로 구성하여 출력

```
timesteps = TIME_STEP
features = 2


Lstm_AE = Sequential()

Lstm_AE.add(LSTM(140,activation='relu', input_shape=(timesteps, features), return_sequences=True))
Lstm_AE.add(LSTM(70,activation='relu', return_sequences=False))
Lstm_AE.add(RepeatVector(timesteps))

Lstm_AE.add(LSTM(70,activation='relu', return_sequences=True))
Lstm_AE.add(LSTM(140,activation='relu', return_sequences=True))
Lstm_AE.add(TimeDistributed(Dense(features)))

Lstm_AE.compile(loss='mse', optimizer=keras.optimizers.Adam(0.01))
Lstm_AE.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 36, 140)           80080

lstm_1 (LSTM)                (None, 70)                59080

repeat_vector (RepeatVector  (None, 36, 70)            0
)

lstm_2 (LSTM)                (None, 36, 70)            39480

lstm_3 (LSTM)                (None, 36, 140)           118160

time_distributed (TimeDistr  (None, 36, 2)             282
ibuted)

=================================================================
Total params: 297,082
Trainable params: 297,082
Non-trainable params: 0
_________________________________________________________________
```

종류	LSTM-AE (Auto Encoder)
활성함수(activation)	Relu
손실함수(loss)	mse (mean squre error)
최적화(optimizer)	Adam
학습률	0.01

[LSTM-AE 모델링 결과화면 및 파라미터 설정 값]

⑦ 모델학습 (정상데이터만 학습)

- 데이터 학습

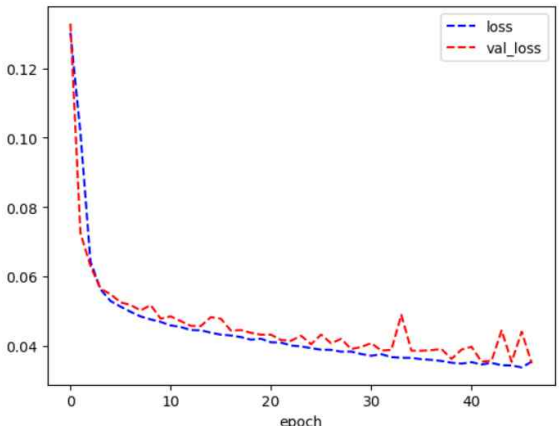
```
Epoch 42/50
808/808 [==============================] - 315s 390ms/step - loss: 0.0348 - val_loss: 0.0388
Epoch 43/50
808/808 [==============================] - 317s 392ms/step - loss: 0.0352 - val_loss: 0.0397
Epoch 44/50
808/808 [==============================] - 315s 390ms/step - loss: 0.0345 - val_loss: 0.0353
Epoch 45/50
808/808 [==============================] - 316s 391ms/step - loss: 0.0350 - val_loss: 0.0356
Epoch 46/50
808/808 [==============================] - 315s 389ms/step - loss: 0.0343 - val_loss: 0.0444
Epoch 47/50
808/808 [==============================] - 315s 390ms/step - loss: 0.0343 - val_loss: 0.0353
Epoch 48/50
808/808 [==============================] - 314s 388ms/step - loss: 0.0337 - val_loss: 0.0440
Epoch 49/50
808/808 [==============================] - 317s 393ms/step - loss: 0.0354 - val_loss: 0.0349
Epoch 50/50
808/808 [==============================] - 370s 458ms/step - loss: 0.0337 - val_loss: 0.0378
```

Epoch(반복횟수)	50
Batch(입력데이터 개수)	32
검증 데이터 비율	0.2(20%)

[LSTM-AE 모델학습 결과화면 및 파라미터 설정 값(아래)]

⑧ 학습 결과

- 결과(손실함수 비교 - 학습VS검증용)



[손실함수 비교]

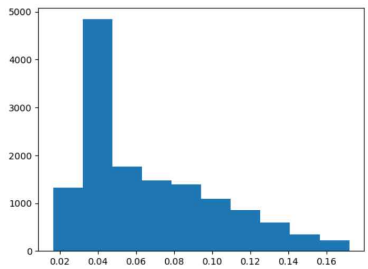
- Reconstruction error : 정상데이터가 입력될 땐 출력값과 입력값이 유사하기 때문에 mse가 0에 가깝지만, 비정상 데이터가 입력될 시에는 차이가 커짐

```
prediction = Lstm_AE.predict(X_test)
mse=np.mean(np.power(X_test-prediction,2),axis=1)
error_df=pd.DataFrame(mse)
error_df.columns = ['Temp_error', 'Current_error']
error_df['reconstruction_error'] = error_df['Temp_error'] + error_df['Current_error']
error_df['Label'] = Y_test[:,0]
error_df.describe()
```

[reconstruction error 추출]

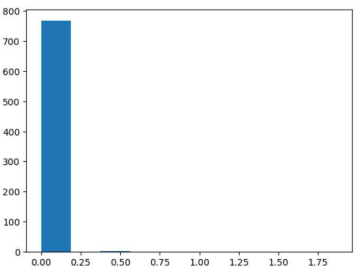
- threshold(임계값) 설정 : Reconstruction error 데이터의 분포는 대부분 0에 가까우며 실제 검증용 데이터도 정상데이터가 대부분이므로, 임계값(thr)을 상위 90%(0.17)으로 설정함 (90%을 넘어가는 경우에는 비정상 데이터가 정상으로 판단되어지는 FP(False Positive)가 증가하기 시작함)

```
thr = np.percentile(error_df['reconstruction_error'],90)
fig=plt.figure()
ax=fig.add_subplot(111)
error_df_0=error_df[error_df['reconstruction_error']<thr]
print(len(error_df_0))
_=ax.hist(error_df_0.reconstruction_error.values,bins=10)
```



[임계값(0.17) 이하의 데이터(정상) 개수 분포(총 13899개) (왼쪽) / 결과 히스토그램(오른쪽)]

```
fig=plt.figure()
ax=fig.add_subplot(111)
error_df_1=error_df[error_df['reconstruction_error']>thr]
print(len(error_df_1))
_=ax.hist(error_df_1.reconstruction_error.values,bins=10)
```

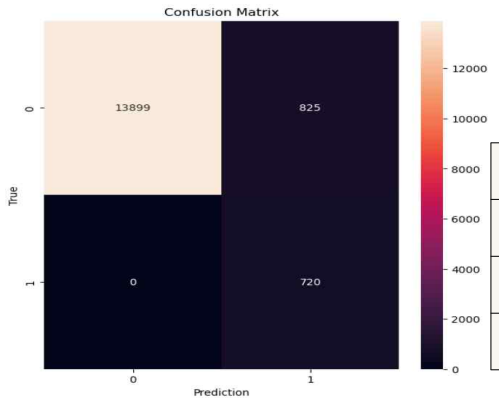


[임계값(0.17) 이상(비정상)의 데이터 개수 분포(총 1545개) (왼쪽) / 결과 히스토그램(오른쪽)]

⑨ 평가 및 해석

- 성능지표: 오차행렬(Confusion matrix)

```
Y_predict=[1 if e>thr else 0 for e in error_df['reconstruction_error'].values]
conf_matrix=confusion_matrix(error_df['Label'],Y_predict)
plt.figure(figsize=(7,7))
sns.heatmap(conf_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Prediction'); plt.ylabel('True')
plt.show()
```



TP(True Positive)	13899
FP(False Positive)	0
TN(True Negative)	720
FN(False Negative)	825

[Confusion Matrix를 히트맵으로 나타냄(위) / 결과(아래)]

- 성능지표: 정확도, 예측률, 재현률, F1 Score (1에 가까울수록 좋은 성능)

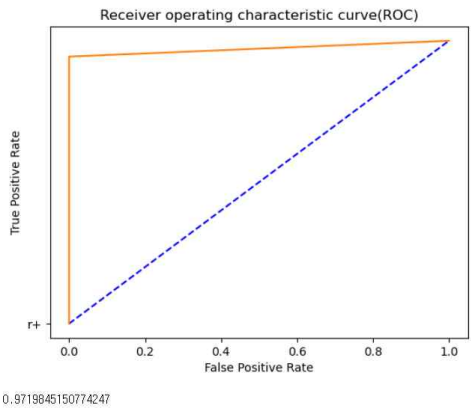
```
TP = conf_matrix[0][0]
FN = conf_matrix[0][1]
FP = conf_matrix[1][0]
TN = conf_matrix[1][1]

Recall=TP/(TP+FN)
Precision = TP/(TP+FP)
Accuracy=(TP+TN)/(TP+FP+FN+TN)
F1_Score=2*(Recall*Precision)/(Recall+Precision)
```

[재현율, 예측률 ,정확도,F1 score]

정확도(accuracy)	예측률(Precision)	재현율(Recall)	F1 score
0.946	1.0	0.944	0.971

- 성능지표: ROC Curve 및 ROC_AUC (1에 가까울수록 좋은 성능)

```
fpr, tpr, thresholds = roc_curve(error_df['Label'], Y_predict)
plt.plot([0,1], [0,1], "b--", "r+")
plt.plot(fpr, tpr, label='RandomForest')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve(ROC)')
plt.show()
```



0.971984150774247

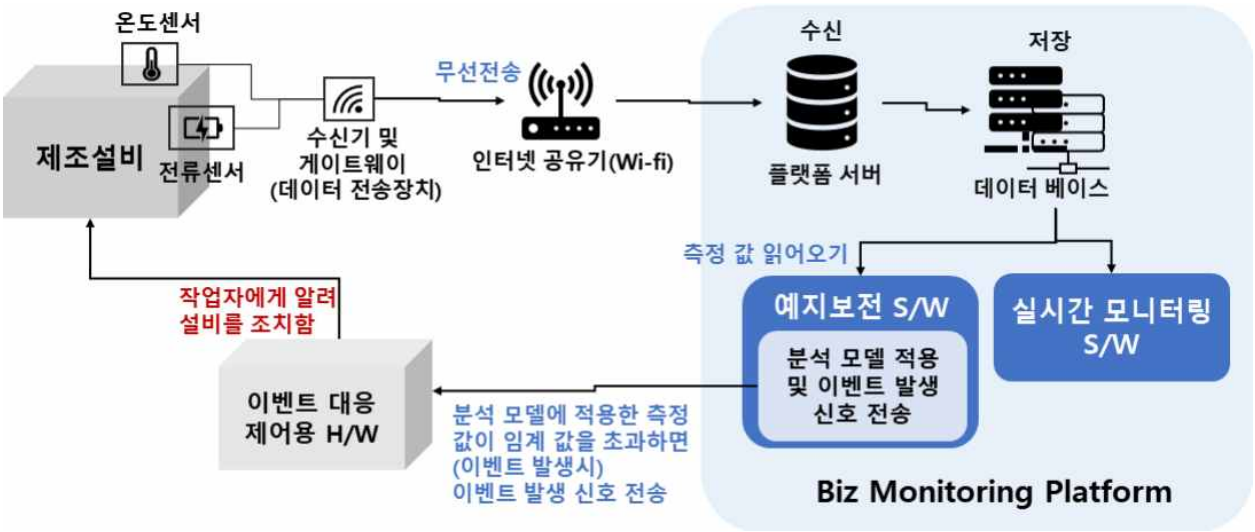
[ROC Curve 및 ROC_AUC(왼쪽) / 결과그래프 및 ROC_AUC(=0.972)(오른쪽)]

Ⅲ. 결론 및 향후 방향

(1) 결론

- 본 시스템의 성능지표는 최종 ROC_AUC=0.972로 우수한 성능으로 확인됨
- 따라서, 해당 설비의 예지보전을 위하여 위에서 제시한 분석모델을 적용할 시에 오류 발생 예측에 있어 상당부분 효과가 있을 거라 판단됨
- 추후의 예지보전 솔루션 구축 시에는, 오류 발생이 예상될 때 알릴 수 있는 이벤트 대응 제어용 모듈을 추가 적용하여, 실제 이벤트 발생 시 대응할 수 있도록 함

(2) 향후 예지보전 솔루션 구축 내용



[데이터 측정부터 이벤트 알림까지의 전체 데이터 흐름]

① 데이터 수집 및 전송 (센서 → 플랫폼 서버)

② 데이터베이스에 저장된 측정 데이터들을 실시간 모니터링하며 간단한 데이터 분석 결과를 확인하는 한편(실시간 모니터링), 분석 모델에 입력하여 출력된 값을 확인하고 임계 값과 비교하여, 초과 시(이벤트 발생 시) 이벤트 대응 제어용 H/W로 이벤트 발생 신호 전송(설비 예지보전)

③ 경광등으로 이벤트 발생을 알리거나, 설비 전원을 자동으로 On/Off 하도록 함

④ 한편, 플랫폼에서는 측정 데이터들을 데이터베이스에 저장하여 실시간 모니터링하고(실시간 모니터링 SW), 데이터들의 변화를 기반으로 새로이 분석 모델링을 진행하여 임계 값과 파라미터 값들을 업데이트 함(예지보전 S/W)