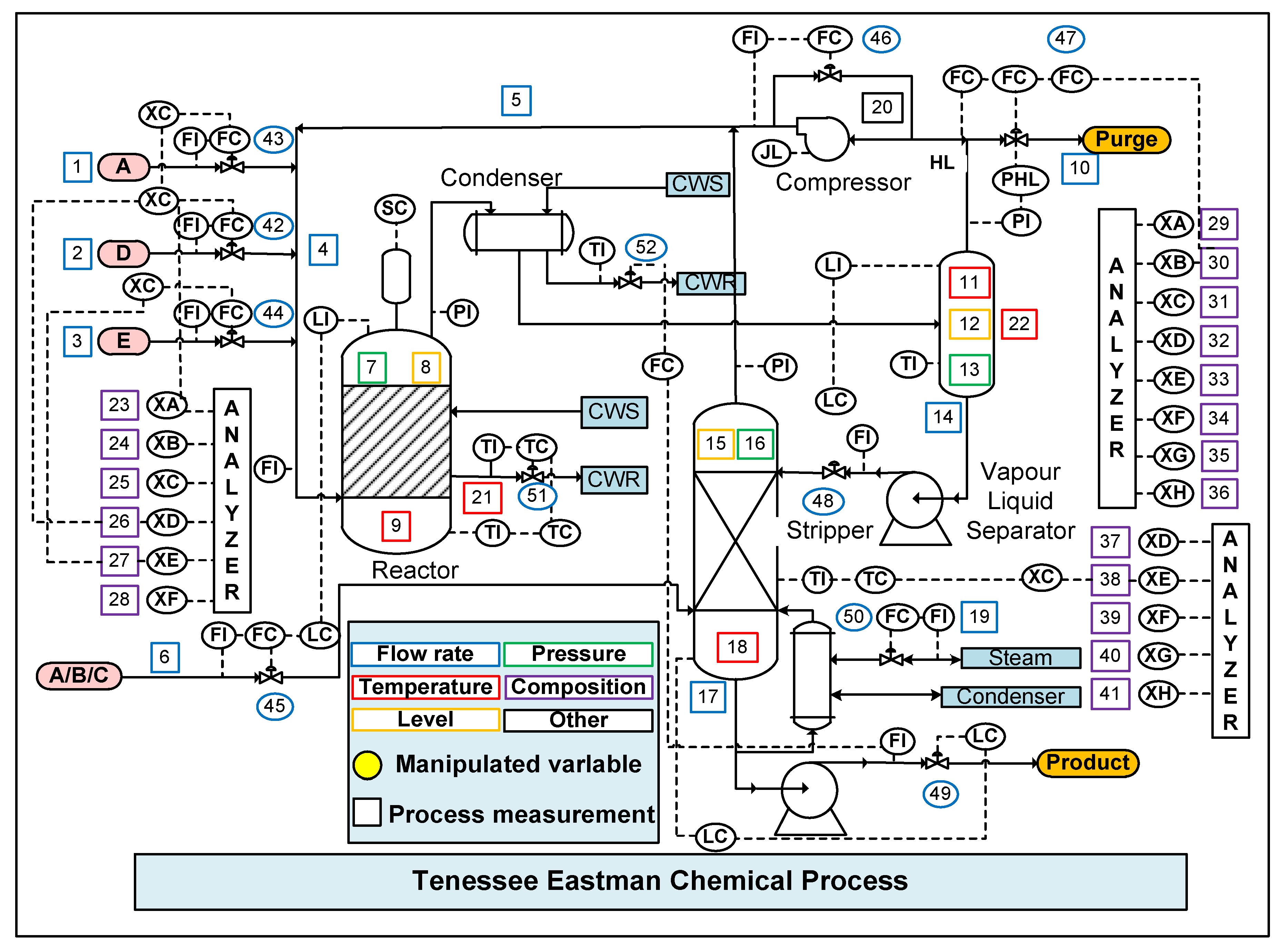
1. **인공지능 기반 예지보전 및 안전**

## 화학 공정 이상 감지 및 진단

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | 화학 공정의 운전 상태(정상/이상) 이해 |
| [방법] | 오토인코더 모델 구축 |
| [응용] | 이상 감지 모니터링 및 기여도 분석 |
| [요약] |  |

### [공정설명] 테네시 이스트만 공정



테네시 이스트만 공정(Tennessee Eastman Process, TEP)은 공정 제어 및 모니터링 방법을 평가하기 위하여 Eastman Chemical Company가 제공한 현실적인 공정이다. 시험 공정은 성분, 반응 속도, 운전 조건 등이 수정된 실제 화학 공정의 시뮬레이션을 기반으로 한다. 본 공정은 반응기, 응축기, 압축기, 분리기, 스트리퍼 등 5가지 주요 장치로 구성되어 있고, A, B, C, D, E, F, G, H의 8가지 성분이 포함되어 있다.

기체 반응물인 A, C, D, E와 불활성 기체 B가 반응기에 투입되어 다음과 같은 반응을 통해 액체 생성물인 G와 H가 생성된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

반응기 생성물은 응축기를 통해 냉각된 다음, 증기-액체 분리기로 투입된다. 분리기에서 나온 증기는 압축기를 통해 반응기 피드로 재순환된다. 분리기의 응축된 성분은 남은 반응물을 제거하기 위해 스트리퍼로 가고 스트리퍼 하단에서는 최종 생산물인 G와 H가 분리되어 나온다.

본 공정에는 41개의 측정 변수와 12개의 조작 변수가 포함되어 있다. 측정 변수는 3분마다 샘플링되고 XMEAS1부터 XMEAS22까지는 운전 변수이고 XMEAS23부터 XMEAS41까지는 성분 변수이다. 모든 측정 변수에는 가우시안 노이즈가 포함되어 있다.

TEP 시뮬레이션에는 21개의 사전 프로그래밍된 Fault가 있다. Fault 1-7은 공정 변수의 단계적 변화와 관련이 있고, Fault 8-12는 일부 공정 변수의 변동성 증가와 관련이 있다.

테이블이(가) 표시된 사진

자동 생성된 설명

### [문제] TEP 공정의 이상을 감지하고 이상 발생시 원인을 진단하여라.

- “./코드 및 데이터/8-1 TE process”의 데이터를 활용하여라.

- “data0.csv”을 학습 데이터로 “data0\_test”을 평가 데이터로 사용하여라.

### [방법] TEP 공정의 오토인코더 모델 구축

#### Q1. 오토인코더를 학습시키기 위한 정상 운전 데이터를 작업 환경으로 불러오자.

|  |
| --- |
| import pandas as pd  data\_normal = pd.read\_csv(“./8-1 TE process/data0.csv”, index\_col=0)  data\_test = pd.read\_csv(“./8-1 TE process/data0\_test.csv”, index\_col=0) |

pd.read\_csv 함수는 해당경로의 데이터를 불러와 데이터프레임 형태로 저장한다. 여기서 “index\_col=0”은 첫 열을 인덱스로 지정한다는 의미이다. data\_normal은 훈련 데이터이고 data\_test은 평가 데이터이다.

#### Q2. 데이터의 기본 정보를 파악해보자.

|  |
| --- |
| data\_normal.info() |

테이블이(가) 표시된 사진

자동 생성된 설명

info()는 데이터에 대한 간단한 설명을 보여준다. XMEAS1부터 XMEAS41까지 41개의 변수에 대해 960개의 데이터 샘플이 존재하고 결측치는 존재하지 않음을 확인할 수 있다.

|  |
| --- |
| data\_normal.describe() |



describe()는 각 변수의 샘플 개수, 평균값, 표준편차, 최솟값, 최댓값 등을 보여준다. 공정이 정상 상태일 때 각 공정 변수의 운전 조건을 알 수 있다.

#### Q3. 오토인코더 모델의 학습과 검증을 위해 데이터를 분리하여라.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  test\_ratio = 0.2  random\_state = 42  data\_train, data\_val = train\_test\_split(data\_normal, test\_size = test\_ratio,  random\_state=random\_state)  print(data\_train.shape, data\_val.shape, data\_test.shape) |



모델의 훈련과 검증을 위해 train\_test\_split 함수를 이용하여 훈련 데이터를 분리한다. 훈련 데이터의 20%를 검증 데이터로 사용하고 평가 데이터는 이전에 불러온 data\_test를 사용한다. shape 속성을 이용하여 각 데이터셋의 크기를 출력해보았다.

#### Q4. 입력 데이터의 값을 표준화하여라.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(data\_train)  X\_val\_scaled = scaler.transform(data\_val)  X\_test\_scaled = scaler.transform(data\_test) |

데이터 스케일링을 통해 각 공정 변수 사이의 범위를 일정한 수준으로 맞춰주어 모델이 편향성을 갖는 것을 방지한다. 여기서는 표준화를 위한 StandardScaler가 사용되었고, 모델 훈련 시 평가 및 검증 데이터의 정보가 반영되지 않도록 훈련 데이터에 맞추어 스케일링을 진행한다.

#### Q5. 공정 변수 차원 축소를 위한 오토인코더를 구축하여라.

|  |
| --- |
| from tensorflow import keras  from tensorflow.keras.layers import \*  from tensorflow.kears.optimizers import Adam  def Autoencoder(input\_dim, num\_nodes, num\_latents, learning\_rate):  model = keras.Sequential()  model.add(Dense(num\_nodes, input\_shape=(input\_dim, )))  model.add(BatchNormalization())  model.add(Activation(‘relu’))  model.add(Dense(num\_latents))  model.add(BatchNormalization())  model.add(Activation(‘relu’))  model.add(Dense(num\_nodes))  model.add(BatchNormalization())  model.add(Activation(‘relu’))  model.add(Dense(input\_dim, activation=’linear’))  optimizer = Adam(learning\_rate=learning\_rate, decay=0.001)  model.compile(loss='mse', metrics=’mae’, optimizer = optimizer)  return model |

오토인코더 함수는 입력 차원, 은닉뉴런 개수, 잠재공간 차원, 학습률을 하이퍼파라미터로 갖는다. 하이퍼파라미터는 문제에 따라 적절한 개수를 설정해주어야 한다. 이후에는 compile메소드를 이용하여 모델의 학습 환경을 설정한다. 최적화 알고리즘(optimizer)은 adam을, 손실 함수(loss)는 mean squared error를, 평가지표(metrics)로는 mean absolute error을 선택하였다.

다음으로 Autoencoder 함수를 이용하여 모델을 만든다.

|  |
| --- |
| input\_dim = X\_train\_scaled.shape[1]  num\_nodes = 64  num\_latents = 28  learning\_rate = 0.001  ae\_model = Autoencoder(input\_dim, num\_nodes, num\_latents, learning\_rate)  ae\_model.summary() |

테이블이(가) 표시된 사진

자동 생성된 설명

Autoencoder 함수에 입력할 인자로 입력 차원은 학습 데이터의 크기로부터 얻고, 은닉 뉴런과 잠재 공간 크기는 각각 64개, 28개로 설정하였다. 본 문제에서는 오토인코더를 차원 축소 용도로 사용하기 때문에 잠재 공간 크기는 입력 차원 수보다 작아야 한다. summary는 설정된 모델의 구조를 요약하여 출력해준다.

#### Q6. 오토인코더 모델을 정상 데이터를 이용하여 학습시켜라.

|  |
| --- |
| from kears.callbacks import EarlyStopping, ModelCheckpoint  training\_epoch = 1000  patience = 20  earlystopping = EarlyStopping(patience=patience, monitor='val\_loss')  checkpointer = ModelCheckpoint(filepath=’best model.h5’, save\_best\_only=True)  history = model.fit(X\_train\_scaled, X\_train\_scaled,  epochs=training\_epoch,  batch\_size=64,  callbacks=[earlystopping, checkpointer],  validation\_data=(X\_val\_scaled, X\_val\_scaled) |

테이블이(가) 표시된 사진

자동 생성된 설명

모델 훈련 시에는 fit 함수를 이용하고, 훈련 반복 횟수(epochs), 훈련 배치 크기(batch\_size), 검증 데이터(validation\_data)를 설정하였다.

또한, callback 함수를 통해 다양한 훈련 조건을 설정할 수 있다. 조기 종료(EarlyStopping)을 통해 정해진 횟수(patience)동안 검증 데이터에 대한 손실 값이 줄어들지 않으면 훈련을 종료한다. 학습된 모델을 저장하기 위해서 ModelCheckpoint을 사용하고 filepath로 모델을 저장할 파일 이름을 입력한다. save\_best\_only 옵션을 사용하면 검증 데이터 손실이 최고 성능 모델보다 낮아질 때만 저장한다.

#### Q7. 학습된 모델을 불러오고 평가 데이터에 대한 모델의 정확도를 평가하여라. 각 변수에 대한 parity plot을 그려보아라.

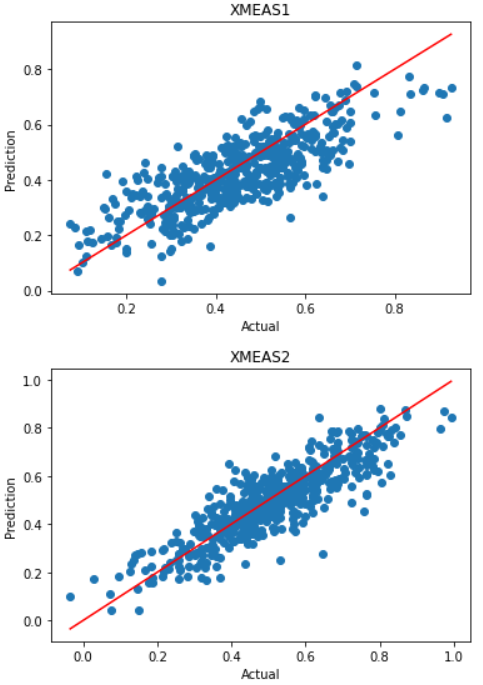
|  |
| --- |
| from keras.models import load\_model  from sklearn.metrics import mean\_squred\_error, mean\_absolute\_error  model = load\_model(‘best model.h5’)  X\_pred = model.predict(X\_test\_scaled)  mse = mean\_squared\_error(X\_test\_scaled, X\_pred)  mae = mean\_absolute\_error(X\_test\_scaled, X\_pred)  print(‘Mean squared error:’, mse)  print(‘Mean absolute error:’, mae) |

텍스트이(가) 표시된 사진

자동 생성된 설명

훈련이 끝나고 저장된 최고 성능 모델을 불러온다. 평가 데이터를 모델에 입력하여 예측값을 얻는다. 손실 값과 평가지표로 사용되었던 mean squared error와 mean absolute error를 계산하여 모델의 정확도를 평가할 수 있다. 0에 가까울수록 모델이 입력값을 잘 재구성(reconstruction)하고 있음을 뜻한다.

|  |
| --- |
| import matplotlib.pyplot as plt  for i in range(input\_dim):  plt.scatter(X\_test\_scaled[:,i], X\_pred[:,i])  plt.plot([min(X\_test\_scaled[:,i], max(X\_test\_scaled[:,i])], [min(X\_test\_scaled[:,i], max(X\_test\_scaled[:,i])], color=’r’)  plt.xlabel(‘Actual’)  plt.ylabel(‘Prediction’)  plt.title(data\_normal.columns[i])  plt.show() |



parity plot은 실제 데이터와 예측 데이터를 비교하는 산점도이고, 실제 값과 예측 값이 같으면 점이 y=x 위에 놓이게 된다. 입력 변수 41개에 대한 parity plot을 그려보면 모델의 변수 별 재구성 성능을 확인할 수 있다.

### [응용] 오토인코더 기반의 공정 이상 감지 및 진단

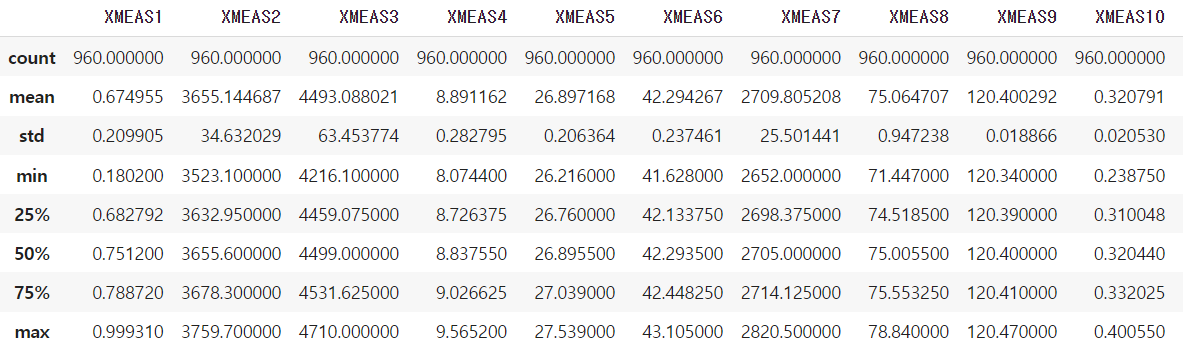
#### Q8. 공정의 상태를 판별하는 지표인 Squared prediction error(SPE)와 SPE의 control limit을 계산하는 함수를 만들어보자.

|  |
| --- |
| import numpy as np  from statsmodels.nonparametric.kde import KDEUnivariate  def SPE(reconstruction\_error):  spe = np.sum(reconstruction\_error\*\*2, axis=1)  return spe  def SPE\_CL(reconstruction\_error, alpha=0.95):  spe = np.sum(reconstruction\_error\*\*2, axis=1)  kde\_spe = KDEUnivariate(spe)  kde\_spe.fit()  spe\_cl = kde\_spe .icdf[int(alpha\*len(kde\_spe.icdf))]  return spe\_cl |

SPE는 입력 데이터에 대한 모델의 재구성 오차(reconstruction error)의 제곱의 합으로 계산된다. 통계적 한계선(control limit)은 커널 밀도 추정(kernel density estimation)을 통해 얻는다. 정상 데이터의 SPE 값에 대해 추정된 확률밀도함수의 역누적분포함수(inverse cumulative distribution function)를 구하고, 신뢰도 alpha값의 확률에 해당하는 SPE값을 계산하면 그 값이 한계선이다.

#### Q9. Q8에서 정의한 함수를 이용하여 Fault 1 데이터에 대한 SPE와 control limit을 계산하여라. 계산된 SPE와 control limit을 이용하여 모니터링 차트를 그려보아라.

|  |
| --- |
| data\_fault1 = pd.read\_csv(‘./Data/data1.csv’, index\_col=0)  data\_fault1.describe() |

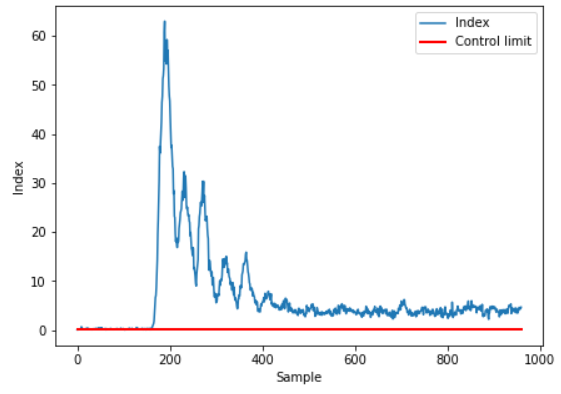


공정 모니터링 성능을 확인하기 위해 Fault 1 데이터를 불러온다. Fault 1 데이터는 960개의 샘플로 이루어져 있고 160번째 샘플부터 A/C feed ratio의 step change가 발생한다.

|  |
| --- |
| X\_fault1\_scaled = scaler.transform(data\_fault1)  X\_fault1\_pred = model.predict(X\_fault1\_scaled)  re\_fault1 = X\_fault1\_pred – X\_fault1\_scaled  spe = SPE(re\_fault1)  X\_scaled = scaler.transform(data\_normal)  X\_pred = model.predict(X\_scaled)  re\_normal = X\_pred – X\_scaled  spe\_cl = SPE\_CL(re\_normal, alpha=0.95) |

이전에 정상 데이터를 표준화했던 scaler에 적용하여 표준화를 진행한 후 훈련된 모델에 입력하였다. 모델의 예측 값과 입력 값 사이의 재구성 오차를 계산하고 이전에 정의한 SPE 함수에 입력하여 SPE 값을 계산한다. 같은 과정을 정상 데이터에 대해 반복하여 SPE\_CL 함수에 입력하면 한계선을 구할 수 있다. 신뢰도 alpha는 기본 값인 95%로 설정하였다.

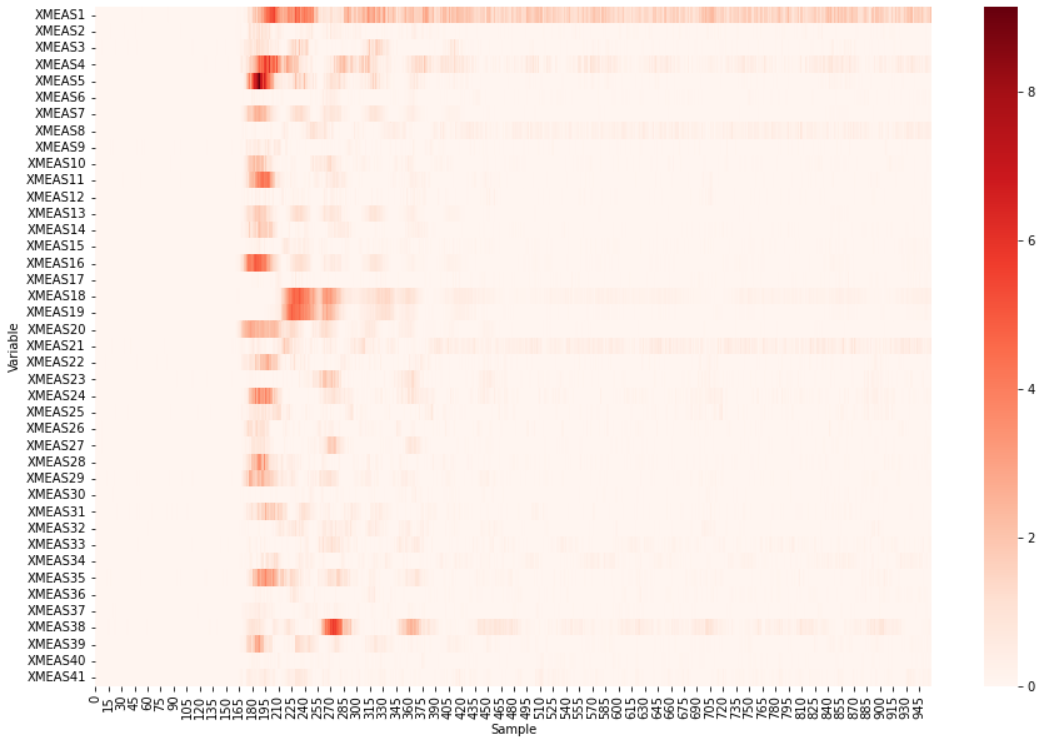
|  |
| --- |
| plt.figure(figsize=(7,5))  plt.plot(spe, label=’Index’)  control\_limit = np.full((len(spe), 1), spe\_cl)  plt.plot(np.arange(len(spe)), control\_limit, color=’r’, linewidth=2, label=’Control limit’)  plt.xlabel(‘Sample)  plt.ylabel(‘Index’)  plt.title(‘Fault1’)  plt.show() |



Sample별로 SPE 값과 한계선을 플롯해보면 SPE 값이 약 160번째 샘플 이후로 급격히 상승하여 한계선을 넘어서서 이상이 감지되는 것을 확인할 수 있다.

#### Q10. Fault 1 데이터에 대한 시간에 따른 기여도 변화를 나타내는 Contribution map을 그려보아라.

|  |
| --- |
| import seaborn as sns  fig, ax = plt.subplots(figsize=(15,10))  sns.heatmap((re\_fault1\*\*2).T, cmap=’Reds’)  ax.set\_yticklabels(data\_fault1.columns, rotation=0)  plt.show() |



기여도는 재구성 오차의 제곱으로 계산된다. re\_fault1을 제곱한 행렬을 전치하여 행을 변수, 열을 샘플로 하는 히트맵(heatmap)을 생성하였다. 마찬가지로 160번째 샘플부터 진한 빨간색이 나타나기 시작하고 제일 윗줄인 XMEAS1(A feed rate) 변수가 전반적으로 큰 기여도를 가지고 있음을 확인할 수 있다.

### [결론]

본 장에서는 오토인코더 모델을 이용하여 화학 공정의 이상 상태를 감지하고 기여도를 분석하였다. 비지도 학습을 이용함으로써 정상 운전 데이터만을 사용하여 모델을 훈련시켰고, 평가 데이터로 모델을 검증하여 신뢰성을 확보하였다. 모델의 재구성 오차를 활용하여 모니터링 지표와 신뢰 구간을 정의하고 이상 데이터에 대한 모니터링 차트를 그려보았다. 또한 이상 발생 시 변수 별 기여도를 시간에 따라 나타내어 어떤 변수들에 이상이 생겼는지 확인할 수 있다.

### [학습 결과]

* 학습 내용

화학 공정의 운전 상태 분석 방법론 익히기

* 학습 결과 확인하기

오토인코더 모델 훈련 방법 및 공정 상태 모니터링 방식 익히기

* 학습 결과 응용하기

본 장의 학습 내용에 기반해 실시간 모니터링 모델 만들기