### **1.1. 수성가스 전이 촉매 분석**

최근 수소가 환경 친화적인 차세대 에너지 전달체로 부각되어 수소 생산을 위한 효율적인 기술이 요구되는 실정이다. 수성 가스 전이 반응 (Water gas shift reaction, WGSR)은 고순도 수소를 생산하는 기술로, 미래 청정에너지 생산을 위한 기술로 각광을 받고있다. WGSR은 합성가스 내의 일산화탄소와 수증기가 반응하여 수소와 이산화탄소를 생성하는 반응으로, 아래와 같은 반응식을 따른다.

고효율의 WGSR 공정 설계에는 고성능 WGSR 촉매 개발이 중요하다. 촉매 성능은 활성 (전환율, 선택도) 및 안정성 (기계적 내구성, 열적 안정성)으로 대표되고, 현재 촉매 성능 개선을 위한 촉매 물질의 특성에 대한 연구가 활발히 진행되고 있다. 하지만, 촉매 성능은 촉매 재료와 운전 조건에 따른 복잡한 현상으로 실험단계에서 예측하는 것은 상당히 어렵다.

그러므로, 고성능 촉매 설계를 위해서 촉매 성능을 예측할 정확하고 정밀한 방법론이 요구되는 실정이다. 만약, 촉매의 성능을 예측할 수 있다면 촉매 개발과 공정 적용의 긴 R&D 생애주기를 효과적으로 줄일 수 있다. 본 실습에서 제안한 인공신경망은 고차원 데이터를 처리할 수 있는 알고리즘으로, 복잡계 해석을 위한 유용한 방법론이다.

본 실습에서는 인공신경망 기반 고성능 WGSR 촉매 예측을 목적으로한다.

### **[문제]**

**촉매 물질 조건 (금속 함량, 가공 환경)과 운전 조건 (반응물 조성, 온도, 압력 등)을 이용하여 촉매 성능 예측 모델을 개발하고 모델의 정확도를 평가하라.**

**- “코드 및 데이터/4-1. WGSR.csv” 데이터를 활용하여라.**

**-학습 및 평가 데이터는 7:3 비율로 분리하여라.**

**- 95% 이상의 정확도를 갖는 모델을 개발하라.**

### **[방법] 인공신경망 예제**

#### 인공신경망을 활용하기 위한 라이브러리를 조사하라.

1. 인공신경망 학습을 위한 다양한 라이브러리가 존재한다.

|  |
| --- |
| Library(‘neuralnet’) |

|  |
| --- |
| Library(‘H2O’) |

|  |
| --- |
| Library(‘keras’) |

#### 인공신경망 학습에는 다양한 하이퍼파라미터가 선행적으로 결정되야한다. 이들 중 모델 정확도와 직접적으로 연관있는 하이퍼파라미터 중 활성화 함수에대해 설명하라.

1. Activation function은 인공신경망의 node를 활성화시키는 활성화 함수로, input된 가중치 합을 출력 신호로 변환하는 함수이다. 인공 신경망에서 이전 레이어에 대한 가중 합의 크기에 따라 활성 여부가 결정된다. 목적 및 역할에따라 선택적으로 사용된다.

### **[응용] 인공 신경망 기반 촉매 반응 예측**

예제는 R 4. 0. 2 프로그래밍 언어를 기준으로 Rstudio 개발환경에서 작성되었다. 예제 실습을 위해 Anaconda 프로그램을 아래 URL로부터 다운로드할 수 있다.

https://www.anaconda.com/

#### 데이터를 Jupyter환경으로부터 불러오고 데이터를 확인하라.

1. 다음과 같은 code를 사용하여 불러올 수 있다.

|  |
| --- |
| WGSR <- read.csv(file.choose(), header = T) |

‘read.csv’ 함수를 사용하여 데이터 파일이 저장된 장소를 직접 찾아 ‘WSGR’ 이름으로 데이터를 불러온다. 해당 데이터는 column 이름이 이미 존재하는 데이터로 ‘header = T’ arg를 통해 이를 밝힌다.

|  |
| --- |
| View(WGSR) |

‘view()’ 함수를 사용하여 데이터를 확인할 수 있다.

|  |
| --- |
| WGSR\_re <- WGSR |

‘WGSR’ 데이터의 손상을 막기위해 ‘WGSR\_re’ 데이터를 생성하여 사용한다.

#### 인공지능 학습을 위해서 데이터의 단위를 무시한 상대적 영향력을 파악할 필요가 있다. 촉매 반응 데이터를 표준화하라. 또한, 데이터의 결측치를 제거하라.

1. 다음과 같은 절차를 통해 표준화 및 결측치 제거가 가능하다.

|  |
| --- |
| WGSR\_scale <- scale(WGSR\_re) |

‘scale()’ 함수를 사용하여 ‘WGSR\_re’ 데이터를 전처리한다.

|  |
| --- |
| WGSR\_scale <- data.frame(replace(WGSR\_scale, is.na(WGSR\_scale), 0) |

‘replace()’ 함수를 사용하여 ‘WGSR\_scale’ 데이터의 공백값 (‘NA’)를 제거하도록 한다. 이때, ‘is.na’ arg를 활용해 해당 데이터의 공백값만을 인식시킬 수 있다.

#### 인공신경망 기반 예측 모델의 정확도를 검증하기 위해 데이터를 모델 학습 데이터와 모델 검증 데이터로 분할하라.

1. 다음과 같은 절차를 통해 데이터의 분할이 가능하다.

|  |
| --- |
| smp\_size <- floor(0.7 \* nrow(WGSR\_scale))  train\_ind <- sample(seq\_len(nrow(WGSR\_scale)), size = smp\_size) |

‘floor()’ 함수를 사용하여 ‘WGSR\_scale’ 데이터의 행 개수의 70%에 해당되는 값을 생성한다. 이 값은 ‘smp\_size’로 명명하여, 후에 무작위 데이터 추출에 사용한다. ‘sample()’ 함수를 사용해서 ‘WGSR\_scale’ 데이터로부터 행 번호를 무작위 추출을 진행한다. ‘size’ arg를 ‘smp\_size’로 설정하여 총 70% 데이터를 추출한다. ‘seq\_len()’ 함수를 통해 1부터 임의의 지정된 숫자까지 순차 데이터를 생성할 수 있다.

|  |
| --- |
| neural\_train <- WGSR\_scale[train\_ind,]  neural\_test <- WGSR\_scale[-train\_ind,] |

임의로 추출된 행 번호 (‘train\_ind’)를 사용해서 ‘WGSR\_scale’ 데이터의 행들을 추출한다. 70%가 학습에 사용할 데이터로 ‘neural\_train’ 으로 명명한다. ‘train\_ind’ 행 번호를 제외한 데이터는 검증에 사용될 데이터이므로 ‘neural\_test’로 명명한다.

|  |
| --- |
| name\_col <- c(colnames(WGSR\_scale))  name\_col <- name\_col[1:38] |

인공신경망 학습을 위해 변수 이름을 지정하는 과정이 필요하다. ‘WGSR\_scale’ 데이터의 열 이름을 추출하여 ‘name\_col’로 명명한다. 해당 ‘name\_col’의 38번까지가 독립변수이므로 해당 부분을 추출하여 ‘name\_col’을 재 정의한다.

#### 인공 신경망 학습을 시도하고 정확도를 검증하라.

1. 다음과 같은 절차를 통해 인공 신경망 학습을 시도할 수 있다.

|  |
| --- |
| vari\_name <- as.formula(paste(‘CO.Conversion ~’,  paste(name\_col,  collapse = ‘+’))) |

‘CO.conversion’은 데이터의 종속변수에 해당한다. ‘paste()’함수를 사용해서 ‘name\_col’의 변수들을 ‘+’ 기호로 연결된 일련의 문장으로 만든다. 그리고 ‘CO.conversion~’과 결합한다. 해당 문자열을 ‘as.formula’ 함수를 통해 수식으로 인식시키고, ‘vari\_name’으로 명명한다.

|  |
| --- |
| set.seed(7)  WGSR\_scale\_neural <- neuralnet(vari\_name, neural\_train,  hidden = c(34, 10, 8, 8, 1)) |

‘set.seed()’를 ‘7’로 지정하여 계산의 무작위성을 고정한다. ‘neuralnet()’ 함수를 사용해서 인공신경망 모델을 구축한다. Arg로 ‘vari\_name’을 사용해 데이터의 독립 변수 및 종속 변수 관계를 지정하고, ‘neural\_train’ 데이터를 사용한다는 것을 명시한다. ‘hidden’ arg를 통해 인공신경망의 구조를 설정한다. 구조는 다섯개의 은닉층과 각 층에 해당하는 node인 34-10-8-8-1로 구성된다. 해당 인공신경망을 ‘WGSR\_scale\_neural’로 명명한다.

|  |
| --- |
| WGSR\_scale\_neural$result.matrix |

‘$result.matrix’를 사용하면 ‘WGSR\_scale\_neural’의 weight, bias, 그리고 학습 결과를 확인할 수 있다.

|  |
| --- |
| pred\_result <- data.frame(compute(WGSR\_scale\_neural, neural\_test[,1:38]))  pred\_result\_train <- data.frame(compute(  WGSR\_scale\_neural, neural\_train[,1:38]) |

‘compute’ 함수를 통해 ‘WGSR\_scale\_neural’모델을 사용할 수 있다.‘neural\_test’ 데이터의 독립변수 부분을 지정하여 예측해보도록 한다.마찬가지로 ‘neural\_train’도 예측하여 학습 정확도를 확인해보도록 한다.

|  |
| --- |
| Pred\_train\_unscaled <- data.frame(mean(WGSR$CO.Conversion) +  pred\_result\_train$net.result \* sd(WGSR$CO.Conversion))  neural\_train\_unscaled <- data.frame(mean(WGSR$CO.Conversion) +  neural\_train$net.result \* sd(WGSR$CO.Conversion)) |
| Pred\_result\_unscaled <- data.frame(mean(WGSR$CO.Conversion) +  pred\_result$net.result \* sd(WGSR$CO.Conversion))  neural\_test\_unscaled <- data.frame(mean(WGSR$CO.Conversion) +  neural\_test$net.result \* sd(WGSR$CO.Conversion)) |

예측된 데이터는 scale된 값이므로 unscale할 필요가 있다. 따라서, 수식을 역전시켜 예측된 학습 데이터와 검증 데이터의 원본 값을 추출하도록 한다.

|  |
| --- |
| rsq(neural\_train\_unscaled[,1], pred\_train\_unscaled[,1])  Metrics::mse(neural\_train\_unscaled[,1], pred\_train\_unscaled[,1])  Metrics::rmse(neural\_train\_unscaled[,1], pred\_train\_unscaled[,1]) |
| rsq(neural\_test\_unscaled[,1], pred\_ test\_unscaled[,1])  Metrics::mse(neural\_ test\_unscaled[,1], pred\_ test\_unscaled[,1])  Metrics::rmse(neural\_ test\_unscaled[,1], pred\_ test\_unscaled[,1]) |

정확도 파악을 위해 각 unscale된 데이터의 R2, MSE, RMSE 값을 확인한다.

### **[결론]**

본 장에서는 인공신경망 기법에 기반한 수성 가스 전이 반응 촉매의 성능 예측 모델을 개발하고, 정확도를 평가하였다. 모델 개발을 위해 데이터 전처리 과정으로 데이터 표준화를 진행했다. 또한, 모델 개발을 위해 인공신경망의 하이퍼파라미터를 조정하여 모델 정확도 개선 방법을 학습했다. 결과로, 높은 모델 정확도를 확보하기위한 과정으로 데이터 전처리와 모델 하이퍼파라미터 튜닝의 중요성을 이해하였다.

### **학습 결과**

* 학습 내용

고성능 수성화 전이 반응 촉매 개발을 위한 인공 신경망 기반 예측 방법론 익히기.

* 학습 결과 확인하기

인공 신경망 알고리즘의 활용 방법 및 예측 모델 학습을 위한 데이터 구조화 익히기.

* 학습 결과 응용하기

본 장의 학습내용에 기반해 촉매 성능의 예측과 예측 알고리즘의 정확도 향상시키기.

### **1.2. 이온성 액체 활성도 분석**

이온성 액체 (Ionic liquids, ILs)는 구조를 사용목적에 따라 선택적으로 합성 및 사용할 수 있는 designer material로, 흔히 탄화수소 추출 공정에 이용한다. 이온성 액체는 독특한 물리/화학적 특성 (전기화학적 안정성, 높은 이온전도성 등)을 갖아, 기존의 유기 용매 기반의 추출 공정을 대체할 친환경적인 대안으로 각광을 받고있다.

이온성 액체의 거동 원리는 보통 묽은 용액에서의 활동도 계수 (Infinite diluent activity coefficient, IDAC)에 기반해 이해됐다. 이를 계산하기 위한 방법으로 UNIFAC, COSMO-RS, 혹은 Abraham과 같은 mechanistic model들이 사용됐다. 하지만, 활동도 계수 계산을 위한 mechanistic model들은 낮은 정확도와 한정된 데이터를 기반으로 작동한다는 단점이 명확하다. 또한, 수많은 용매와 용질 사이의 활동도 계수 실험은 상당한 시간과 예산을 필요로 하는 작업으로, 현재의 이온성 액체 개발과 적용은 비효율적인 상태에 머물러있다.

인공신경망 (Artificial neural networks)은 데이터 기반 학습을 통한 복잡계 현상의 예측 모델을 생성하는 인공지능 알고리즘이다. 인공신경망은 고차원의 데이터 학습에 높은 성능을 보여주고 있고, 오늘날 다양한 물질 개발 및 공정 개선 연구에 활용되고 있다.

본 실습에서는 이온성 액체의 물리/화학적 특성 데이터 베이스를 활용하여, 인공신경망을 통한 활동도 계수 예측모델을 개발한다.

### **[문제]**

**이온성 액체의 물리/화학적 특성 (밀도, 임계 온도 등)을 이용하여 이온성 액체의 열역학성 특성 예측 모델을 개발하고 모델의 정확도를 평가하라.**

**- “코드 및 데이터/4-2. IL\_data.csv” 데이터를 활용하여라.**

**-학습 및 평가 데이터는 7:3 비율로 분리하여라.**

**- 95% 이상의 정확도를 갖는 모델을 개발하라.**

### **[방법] 인공신경망 예제**

#### 인공신경망 하이퍼파라미터 최적화를 위한 방법론을 조사하라.

1. 하이퍼파라미터 최적화에는 전통적이게 manual search, grid search, random search가 있고, 통계적인 혹은 기계학습 방법으론 genetic algorithm, Bayesian optimization이 있다.

#### 인공신경망 학습을 위해 데이터의 정규화/표준화가 이루어지는 이유를 설명하고 일반적인 정규화/표준화 이외의 데이터 전처리 방법을 조사하라.

1. 인공신경망은 데이터의 변위 및 크기를 인식하는 과정이 가중치를 설정에 영향을 준다. 따라서 데이터의 단위를 무시한 상대적 영향력을 파악을 위해 데이터의 정규화/표준화가 필요가 있다.

일반적인 정규화/표준화 이외의 데이터 전처리 방법은 아래와 같은 예시가 있다.

* L1 (Manhattan distance) normalization
* L2 (Euclidean distance) normalization
* L infinity normalization

### **[응용] 인공 신경망 기반 이온성 액체의 열역학적 특성 예측**

예제는 Python 3.5 프로그래밍 언어를 기준으로 Jupyter 개발환경에서 작성되었다. 예제 실습을 위해 Anaconda 프로그램을 아래 URL로부터 다운로드할 수 있다.

https://www.anaconda.com/

#### 데이터를 Jupyter환경으로부터 불러오고 데이터를 확인하라.

1. 다음과 같은 code를 사용하여 불러올 수 있다.

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt |

‘pandas’ package는 데이터 편집을 위한 도구로, python 언어에서 매우 유용하게 사용된다. OSN 데이터 불러오기 및 데이터 편집 (컬럼 및 이름 설정)을 위해 본 실습에서 사용한다.

‘numpy’ package는 수학적 기능이 탑재된 도구로, 난수 발생과 대수적 계산을 위해 사용된다. 본 실습에서는 데이터의 형태 변환 및 벡터 계산에 사용한다.

‘matplotlib.pyplot’ package는 시각화를 위한 그래프 도구로, 다양한 그래프를 그리기 위해 사용된다.

‘pandas’ package를 import하고 ‘pd’로 축약해 사용한다.

‘numpy’ package를 import하고 ‘np’로 축약해 사용한다.

‘matplotlib.pyplot’ package를 import하고 ‘plt’로 축약해 사용한다.

|  |
| --- |
| IDAC = pd.read\_csv('IL\_data.csv')  df = pd.DataFrame(data = IDAC)  df\_copy = df.copy()  df\_copy |

테이블이(가) 표시된 사진

자동 생성된 설명

‘pd’를 사용하여 Jupyter 환경에 있는 ‘IL\_data.csv’ 데이터를 Jupyter script 환경에 불러온다..

‘IL\_data.csv’는 ‘pd’를 활용, 데이터 프레임 형태인 df로 정의한다.

‘.copy’ 함수를 사용하여 ‘df’ 데이터의 복사본인 ‘df\_copy’를 만든다. 이 과정은 원본 데이터의 회손을 피하기위해 실행된다..

#### 재현성있는 모델 구축을 위해 계산의 무작위성을 고정하고 데이터를 정규화하라.

1. 다음과 같은 절차를 통해 무작위성 고정과 데이터 정규화가 가능하다.

|  |
| --- |
| import random as rn |

‘random’ module은 무작위 숫자 생성 등 무작위 관련 함수를 제공한다. 본 실습에서 훈련 및 검증 데이터셋 생성을 위해 사용한다.

‘random’ module을 import하고 ‘rn’으로 축약해 사용한다.

|  |
| --- |
| import sklearn  from sklearn import preprocessing  from sklearn.preprocessing import MinMaxScaler |

‘sklearn’ package는 계산을 위한 유용한 함수를 다수 내장하고 있다. ‘sklearn.preprocessing’의 ‘MinMaxScaler’ 함수를 import한다.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_squared\_error, r2\_score |

‘sklearn.model\_selection’의 ‘train\_test\_split’ 함수를 import하도록 한다. ‘train\_test\_split’ 함수는 모델 학습 및 검증을 위한 모델 학습 데이터와 검증 데이터를 분할하기 위해 사용한다.

‘sklearn.metrics’의 ‘mean\_square\_error’ 함수와 ‘r2\_score’ 함수를 import하도록 한다. ‘mean\_square\_error’와 ‘r2\_score’는 모델 검증을 위한 정확도 지표를 내장하고 있는 함수이다.

|  |
| --- |
| from keras import Sequential, optimizers, metrics  from tensorflow.keras import backend as K  from keras.layers import Dense, BatchNormalization, Dropout, Activation  from keras.optimizers import Adam, SGD  from keras.initializers import lecun\_normal |

‘keras’는 ‘TensorFlow’와 같은 머신러닝 백앤드 엔진 플랫폼을 지원하는 라이브러리 이자 API (Application programming interface)이다.

‘keras’로부터 ‘Sequential’, ‘optimizer’, ‘metrics’ 함수들을 import한다.

‘tensorflow.keras’ 함수로부터 ‘backend’ import하고 ‘K’로 축약한다.

‘keras.layers’로부터 ‘Dense’, ‘BatchNormalization’, ‘Dropout’, ‘Activation’ 함수들을 import한다.

‘keras.optimizers’로부터 ‘Adam’ 함수를 import한다.

‘keras.initializers’로부터 ‘lecun\_normal’ 함수를 import한다.

|  |
| --- |
| sc = MinMaxScaler() |

‘MinMaxScaler()’ 함수를 ‘sc’로 명명하여 사용.

|  |
| --- |
| standardized\_copy = df\_copy  standardized\_copy.iloc[:,2:] = sc.fit\_transform(df\_copy.iloc[:,2:])  standardized\_copy = pd.DataFrame(standardized\_copy,  columns = df\_copy.columns)  df\_copy = standardized\_copy |

‘sc.fit\_transform’ 함수를 사용하여 ‘df\_copy’ 데이터의 numerical value만 (3열부터 17열)을 정규화한다.

정규화한 데이터는 ‘standardized\_copy’로 명명하고 데이터 프레임 형태로 변환한다.

‘standardized\_copy’ 데이터를 ‘df\_copy’ 로 명명한다.

|  |
| --- |
| X = df\_copy.iloc[:,2:-1]  y = df\_copy[['ln IDAC']] |

‘df\_copy’ 데이터의 독립변수 부분인 2열부터 16열까지를 추출하여 ‘X’로 명명한다.

‘df\_copy’ 데이터의 종속변수 부분인 ‘in IDAC’을 추출하여 ‘Y’로 명명한다.

#### 인공신경망 기반 예측 모델의 정확도를 검증하기 위해 데이터를 모델 학습 데이터와 모델 검증 데이터로 분할하라.

1. 다음과 같은 절차를 통해 데이터의 분할이 가능하다.

|  |
| --- |
| tf.random.set\_seed(34)  randomstate=8 |

‘tf’의 ‘.random.set\_seed’ 함수를 사용하여 데이터 추출의 무작위성을 고정시킨다.

마찬가지로 인공신경망 학습의 무작위성을 고정시키기위해 ‘randomstate’를 생성하고 차후 학습에 사용한다.

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size = .3,  random\_state = randomstate) |

‘train\_test\_split’ 함수를 사용하여 학습데이터와 검증데이터를 생성한다.

‘X\_train’과 ‘Y\_train’은 각각 학습데이터의 독립변수와 종속변수 부분이고 마찬가지로 ‘X\_test’와 ‘Y\_test’는 각각 검증데이터의 독립변수와 종속변수 부분이다.

본 실습에서는 학습에 전체 데이터의 70%를 할당할 것이므로, ‘test\_size’ arg를 ‘.3’으로 설정한다. 모델 학습의 재현성을 위해 데이터 추출의 무작위성을 고정한다 (‘random\_state’ arg).

#### 인공신경망 기반 예측 모델을 구축하라. 모델 학습을위해 활성화함수로 ‘swish’, optimizer로 ‘adam’, 그리고 손실함수는 ‘mean\_square\_error’를 이용하라.

1. 다음과 같은 절차를 통해 인공신경망 학습이 가능하다.

|  |
| --- |
| model = Sequential()  model.add(Dense(units=50, kernel\_initializer=lecun\_normal(seed=**None**),  activation='swish', input\_dim=X.shape[1]))  model.add(Dense(units=100, kernel\_initializer=lecun\_normal(seed=**None**),  activation = 'swish'))  model.add(Dense(units=150, kernel\_initializer=lecun\_normal(seed=**None**),  activation = 'swish'))  model.add(Dense(units=200, kernel\_initializer=lecun\_normal(seed=**None**),  activation = 'swish'))  model.add(Dense(units=250, kernel\_initializer='normal', activation = 'swish'))  model.add(Dense(units=150, kernel\_initializer='normal', activation = 'swish'))  model.add(Dense(units=100, kernel\_initializer='normal', activation = 'swish'))  model.add(Dense(units=50, kernel\_initializer='normal', activation = 'swish'))  model.add(Dense(units=1, kernel\_initializer='normal'))  model.compile(loss='mean\_squared\_error',  optimizer=Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999,  epsilon=1e-8), metrics=[metrics.mse]) |

‘sequential()’ 함수를 사용하여 인공신경망 모델을 생성한다. 모델은 ‘model’로 명명하고 인공신경망 층은 ‘.add’를 통해 추가할 수 있다.

일반적인 신경망 모델을 사용하기 위해 ‘Dense’ 기능을 넣고, 각 노드를 지정한다.

Activation argument는 인공신경망의 활성화 함수에 해당하며, Input shape argument는 데이터의 column 차원이다.

본 실습에선 ‘swish’를 활성화 함수로 사용한다.

‘.compile’ 함수를 통해 학습의 optimizer와 loss function을 설정할 수 있다.

본 실습에서 사용된 Optimizer는 adam optimizer이고 loss function으로 MSE (Mean square error)를 사용한다.

|  |
| --- |
| model.fit(X\_train, y\_train, batch\_size=10, epochs=1000, verbose=0) |

‘.fit’ 함수를 사용하여 모델 학습을 진행한다.

모델 학습에 ‘X\_train’와 ‘y\_train’ 데이터를 사용한다. ‘batch\_size’ arg를 통해 한번에 10개 데이터셋을 고려하도록 한다.

또한, ‘epochs’ arg를 통해 반복 학습의 주기를 1000으로 지정한다.

‘verbose’ arg는 모델 학습의 진행상황을 표현하는 것으로, 0으로 설정하여 진행상황을 표시하지 않는 것으로 지정한다.

#### 인공신경망 기반 예측 모델의 정확도를 파악하라.

1. 다음과 같은 절차를 통해 정확도 검증이 가능하다.

|  |
| --- |
| y\_train\_pred = model.predict(X\_train)  train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)  train\_r2 = r2\_score(y\_train, y\_train\_pred) |

학습이 끝나면, ‘.predict’ 함수를 사용해서 ‘X\_train’ 데이터를 예측한다. 예측된 결과값은 ‘y\_train\_pred’로 명명한다.

‘mean\_squared\_error’와 ‘r2\_score’함수를 사용해서 ‘y\_train’과 ‘y\_train\_pred’의 정확도를 확인한다.

|  |
| --- |
| train\_mse |



|  |
| --- |
| train\_r2 |



학습의 정확도를 확인한다.

|  |
| --- |
| y\_pred = model.predict(X\_test)  test\_mse = mean\_squared\_error(y\_test, y\_pred)  test\_r2 = r2\_score(y\_test, y\_pred) |

마찬가지로 ‘mean\_squared\_error’와 ‘r2\_score’함수를 사용해서 ‘y\_test’과 ‘y\_test\_pred’의 정확도를 확인한다.

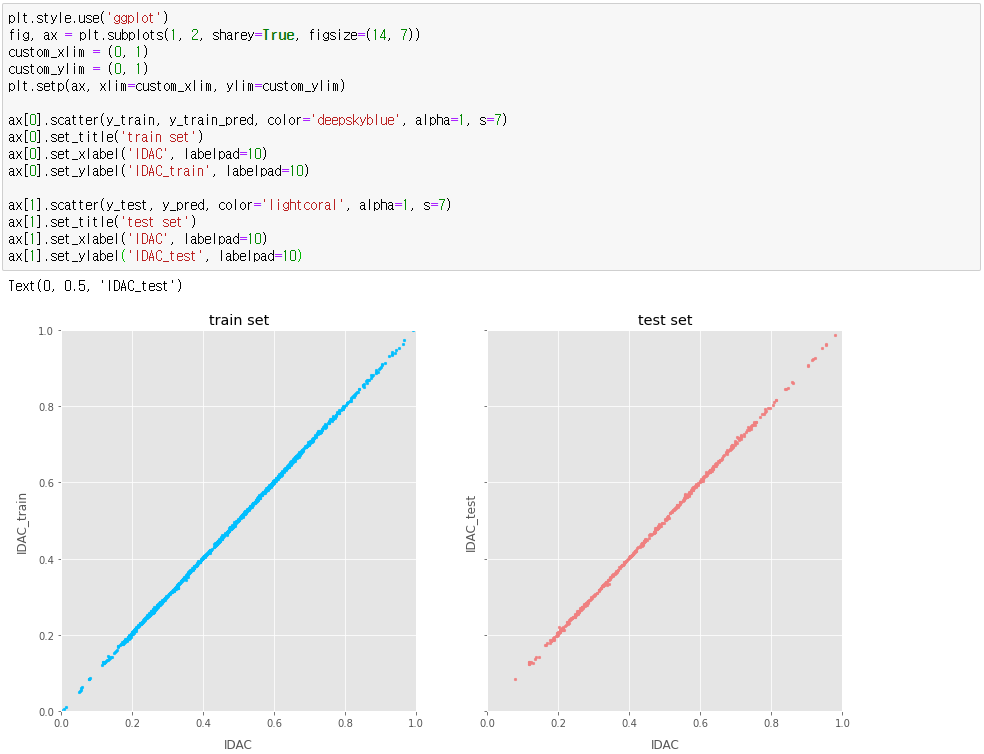
|  |
| --- |
| test\_mse |



|  |
| --- |
| test\_r2 |



|  |
| --- |
| plt.style.use('ggplot')  fig, ax = plt.subplots(1, 2, sharey=**True**, figsize=(14, 7))  custom\_xlim = (0, 1)  custom\_ylim = (0, 1)  plt.setp(ax, xlim=custom\_xlim, ylim=custom\_ylim)  ax[0].scatter(y\_train, y\_train\_pred, color='deepskyblue', alpha=1, s=7)  ax[0].set\_title('train set')  ax[0].set\_xlabel('IDAC', labelpad=10)  ax[0].set\_ylabel('IDAC\_train', labelpad=10)  ax[1].scatter(y\_test, y\_pred, color='lightcoral', alpha=1, s=7)  ax[1].set\_title('test set')  ax[1].set\_xlabel('IDAC', labelpad=10)  ax[1].set\_ylabel('IDAC\_test', labelpad=10) |



‘plt’를 사용하여 정확도를 시각화한다.

### **[결론]**

본 장에서는 이온성 액체의 열역학적 특성 (활동 계수)을 예측하기위해 인공신경망을 이용했다. 데이터 정규화를 통해 데이터 전처리를 진행했다. 또한, 모델 개발을 위해 인공신경망의 하이퍼파라미터를 조정하여 모델 정확도 개선 방법을 학습했다. 결과로, 열역학적 특성 예측을 위한 모델 개발 방법을 학습했다.

### **학습 결과**

* 학습 내용

추출 공정의 열역학 특성 분석을 위한 인공 신경망 기반 예측 방법론 익히기.

* 학습 결과 확인하기

인공 신경망 알고리즘의 활용 방법 및 예측 모델 학습을 위한 데이터 구조화 익히기.

* 학습 결과 응용하기

본 장의 학습내용에 기반해 추출 공정 설계를 위한 인공 신경망 적용 방법 조사하기.

### **1.3. 유기용매 막 소재 분석**

유기용매 나노여과분리막 (Organic solvent nanofiltration, OSN)은 용질의 선택적 분리 기능과 효율적인 물질 투과 능력을 갖고있어 난분해성 폐수의 분리에 탁월한 성능을 보인다. 이로 인해 유기용매 나노여과분리막은 화학, 환경, 의료, 제약, 식품공업 등 다양한 응용이 가능한 까닭에 산업적 수요 기대가 높아지고 있다.

유기용매 나노여과 분리막은 친환경적이고 효율적인 분리 기술이므로, 분리막의 성능 향상을 통해 분리 공정 효율을 개선시키려는 시도가 많이 진행된다. 하지만 분리막 성능은 분리막의 특성 (Property)과 사용 환경 (Operating condition)에 따라 큰 차이가 발생하여 예측이 상당히 어려운 실정이다.

따라서, 고성능 분리막 설계를 위해서는 반복적인 실험에 의존하는 것이 보편적이다. 하지만, 시행착오법은 많은 예산과 시간을 소모하여 다양한 조건에서의 분리막 설계를 지연시키는 한계점을 갖는다. 따라서 새로운 분리막 개발을 통한 분리 공정 개선은 상당히 소극적이게 이루어지며 낮은 성공률을 갖는다. 결국, 분리막 R&D의 효율을 증가시키기 위해서는 분리막 성능 예측이 필수적이다. 특히, 분리막의 주요 변수를 규명하여 정형화된 분리막 성능의 화학적 차원 (Chemical dimension)을 정의한다면, 분리막 성능 예측 모델 개발의 초석이 될 수 있다.

본 실습에서는 차원 축소 방법론 기반 분리막 성능의 경향 파악과 주요 변수 규명을 목적으로한다.

### **[문제]**

**유기용매 막분리 소재의 화학적 차원 조사를 위해 주성분 분석 방법론을 이용하고, 막분리 소재의 성능 결정의 주요 변수를 규명하라.**

**- “코드 및 데이터/4-3. OSN\_membrane\_data.csv” 데이터를 활용하여라.**

**- 데이터의 80~90%를 설명할 수 있는 성분 수를 파악하라.**

**- 가장 높은 데이터 설명력을 갖는 두가지 성분에 기반해 데이터를 정보 차원에서 시각화하고 두 성분에 대한 변수 기여도를 파악하라.**

### **[방법] PCA 예제**

#### PCA을 활용하기 위한 라이브러리를 조사하라.

1. PCA를 활용하기 위한 라이브러리는 대표적이게 sklearn의 sk learn. decomposition이 있다. PCA와 함께 다양한 차원축소 관련 알고리즘을 제공한다.

|  |
| --- |
| From sklearn.decomposition import PCA |

#### 화학적 차원에서 정보 차원으로의 투영에는 데이터 손실이 필연적으로 발생한다. 데이터 손실의 허용 수준을 조사하라.

1. PCA는 변수의 수 (n) 만큼 정보 차원의 축을 생성한다. 축은 principal component (PC) 라고 부르며, 가장 데이터를 잘 설명하는 차원 (PC1) 부터 가장 설명을 못하는 차원 (PCn) 순으로 분류된다. 여기서 데이터 설명력이 80~90% 되는 수준의 PC 수를 채택해야한다.

### **[응용] PCA 기반 여과 공정 소재 특성 분석**

예제는 Python 3.5 프로그래밍 언어를 기준으로 Jupyter 개발환경에서 작성되었다. 예제 실습을 위해 Anaconda 프로그램을 아래 URL로부터 다운로드할 수 있다.

https://www.anaconda.com/

#### 데이터를 Jupyter환경으로부터 불러오고 데이터를 확인하라.

1. 다음과 같은 code를 사용하여 불러올 수 있다.

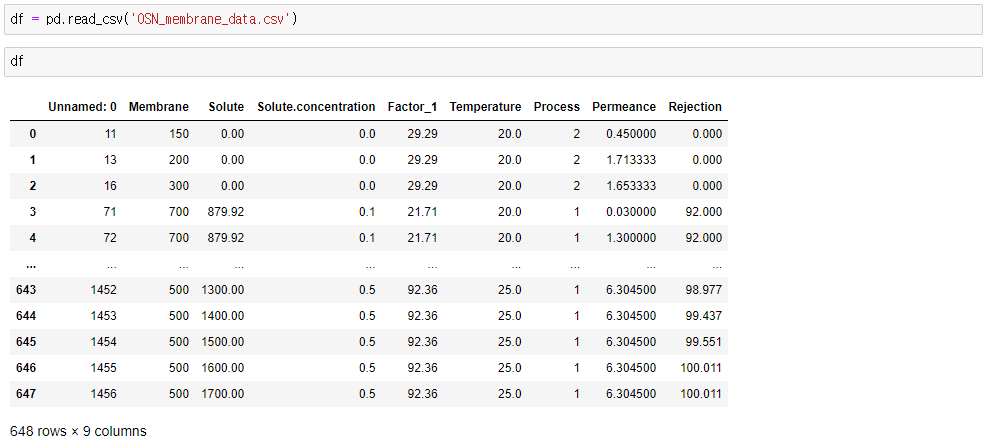
|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt |

‘pandas’ package는 데이터 편집을 위한 도구로, python 언어에서 매우 유용하게 사용된다. OSN 데이터 불러오기 및 데이터 편집 (컬럼 및 이름 설정)을 위해 본 실습에서 사용한다. ‘pandas’ package를 import하고 ‘pd’로 축약해 사용한다.

‘numpy’ package는 수학적 기능이 탑재된 도구로, 난수 발생과 대수적 계산을 위해 사용된다. 본 실습에서는 데이터의 형태 변환 및 벡터 계산에 사용한다. ‘numpy’ package를 import하고 ‘np’로 축약해 사용한다.

‘matplotlib.pyplot’ package는 시각화를 위한 그래프 도구로, 다양한 그래프를 그리기 위해 사용된다. ‘matplotlib.pyplot’ package를 import하고 ‘plt’로 축약해 사용한다.

|  |
| --- |
| df = pd.read\_csv('OSN\_membrane\_data.csv')  df |



데이터는 ‘OSN\_membrane\_data.csv’로 Jupyter 환경에 저장됐다. ‘pd’에 내장된 read.csv 함수를 사용하여 해당 데이터를 Jupyter script 환경에 불러온다.

해당 데이터는 Jupyter script 환경에서 df로 명명되었으며, df를 타이핑하여 데이터를 읽어온다.

|  |
| --- |
| OSN\_data = df.iloc[:,1:9]  OSN\_data.columns = ['Membrane\_MWCO','Solute\_MW','Solute\_concentration', 'Solvent\_factor', 'Temperature','Process\_configuration', 'Permeance', 'Rejection']  OSN\_data |

테이블이(가) 표시된 사진

자동 생성된 설명

행 번호 (row name)가 포함된 첫번째 열을 제거하고 사용한다. 해당 열은 첫번째 열이고, python에서는 0번째 열이다.

행 번호가 삭제된 데이터는 ‘OSN\_data’로 명명하여 사용한다. ‘.colums’ 함수를 사용하여 ‘OSN\_data’의 열 이름을 변수 이름으로 설정한다.

#### 주성분 분석은 데이터의 스케일을 무시한 영향력을 확인이 가능하다. 주성분 분석을 위해 데이터를 표준화하라.

1. 다음과 같은 절차를 통해 표준화한다.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler |

‘sklearn’ package는 계산을 위한 유용한 함수를 다수 내장하고 있다. ‘sklearn.preprocessing’의 ‘StandardScaler’ 함수를 import하도록 한다.

|  |
| --- |
| OSN\_data\_x = OSN\_data.loc[:, :].values |



‘OSN\_data’의 값만을 ‘OSN\_data\_x’로 가져온다. 이때 ‘.loc’ 함수를 사용하여 데이터를 지정하고, ‘.value’함수를 사용하여 값을 가져온다.

|  |
| --- |
| OSN\_data\_x |

테이블이(가) 표시된 사진

자동 생성된 설명

‘OSN\_data\_x’를 확인한다. 데이터는 Array 형식으로 편집됐다.

|  |
| --- |
| OSN\_data\_scale = StandardScaler().fit\_transform(OSN\_data\_x)  OSN\_data\_scale |

테이블이(가) 표시된 사진

자동 생성된 설명

‘StandardScaler’ 함수를 사용하여 ‘OSN\_data\_x’ 데이터의 표준화를 진행한다.

#### PCA를 통해 화학적차원에서 정보차원으로 데이터를 투영하라.

1. 다음과 같은 절차를 통해 데이터를 정보차원에 투영할 수 있다.

|  |
| --- |
| from sklearn.decomposition import PCA |

‘sklearn.decomposition’의 ‘PCA’함수를 import하도록 한다.

|  |
| --- |
| pca = PCA(n\_components=8) |

‘PCA’함수를 사용하여 8개의 성분을 갖는 주성분을 생성한다.

|  |
| --- |
| principalComponents = pca.fit\_transform(OSN\_data\_scale) |

‘pca.fit\_transform’ 함수를 사용하여 정규화된 ‘OSN\_data\_scale’ 데이터를 정보차원으로 변환한다. 정보 차원으로 변환된 데이터는 ‘principalComponents’로 명명한다.

|  |
| --- |
| principalDf = pd.DataFrame(data = principalComponents  , columns = ['principal component 1', 'principal component 2',  'principal component 3', 'principal component 4',  'principal component 5', 'principal component 6',  'principal component 7', 'principal component 8'])  principalDf |

테이블이(가) 표시된 사진

자동 생성된 설명

‘pd’ 패키지의 ‘DataFrame’ 함수를 사용하여 ‘principalComponents’ 데이터을 ‘principalDf’로 명명하고 열 이름 (Column name)을 수정한다.

|  |
| --- |
| pca.explained\_variance\_ratio\_  sum(pca.explained\_variance\_ratio\_) |



‘.explained\_variance\_ratio’ 함수를 통해 미리 생성해둔 ‘pca’의 각 주성분의 데이터 설명도 (dimension coverage)를 확인할 수 있다.

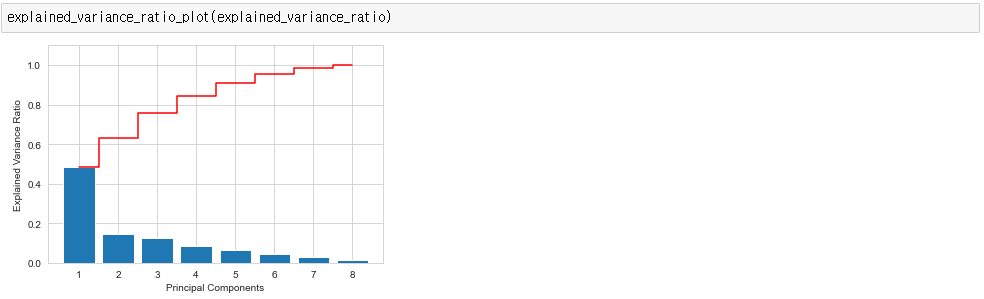
앞에서부터 첫번째 주성분 (PC1)이며 혼자서 전체 데이터의 48.3% 정도 설명할 수 있는 것을 확인한다.

전체 주성분 (PC1~8)의 데이터 설명도 총합은 100%이다.

#### PCA 방법론은 데이터 손실을 최소화할 차원수가 선택되야한다. 데이터 해석을 위해 최소 몇 개의 차원이 필요한가?

1. 다음과 같은 절차를 통해 각 차원의 데이터 설명력을 확인할 수 있다.

|  |
| --- |
| explained\_variance\_ratio = pca.explained\_variance\_ratio\_  **def** explained\_variance\_ratio\_plot(explained\_variance\_ratio):  x\_axis = range(1, len(explained\_variance\_ratio)+1)  plt.bar(x\_axis, explained\_variance\_ratio,  align = 'center', label = 'Individual Explained Variance Ratio')  plt.step(x\_axis, np.cumsum(explained\_variance\_ratio),  where = 'mid', color='red',  label='Cumulative Explained Variance Ratio')  plt.ylim(0, 1.1)  plt.xticks(x\_axis)  plt.xlabel('Principal Components')  plt.ylabel('Explained Variance Ratio')  plt.grid("true")  plt.show()  explained\_variance\_ratio\_plot(explained\_variance\_ratio) |
|  |



차원 축소는 전체 데이터의 80~90% 이상의 설명력을 갖는 주성분 개수를 선택하면 된다. 여기서 누적 그래프를 확인하면 최소 4개이상의 주성분이 선택되야한다는 것을 알 수 있다.

시각화를 위해 함수를 정의하여 사용한다. ‘def’를 사용하여 함수를 정의할 수 있고, ‘plt’ package를 사용해 시각화 한다.

#### 가장 데이터 설명력이 높은 두개의 차원을 채택해 데이터를 시각화하라.

1. 다음과 같은 절차를 통해 데이터 시각화가 가능하다.

|  |
| --- |
| PCs\_12 = principalDf.iloc[:,0:2]  PCs\_12 |

텍스트이(가) 표시된 사진

자동 생성된 설명

‘pd’ package의 ‘iloc’ 함수를 활용하여 ‘principalDf’ 데이터의 1~2번째 열을 추출하여 ‘PC\_12’로 명명한다.

* 나노여과 분리막 정보 차원 시각화 (2D visualization)

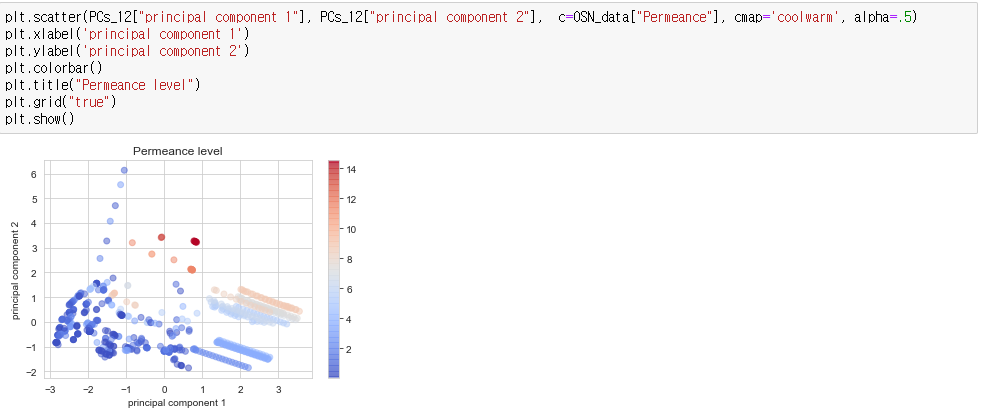
|  |
| --- |
| plt.scatter(PCs\_12["principal component 1"], PCs\_12["principal component 2"], color='grey', alpha=.5)  plt.xlabel('principal component 1')  plt.ylabel('principal component 2')  plt.grid("true")  plt.show() |

텍스트이(가) 표시된 사진

자동 생성된 설명

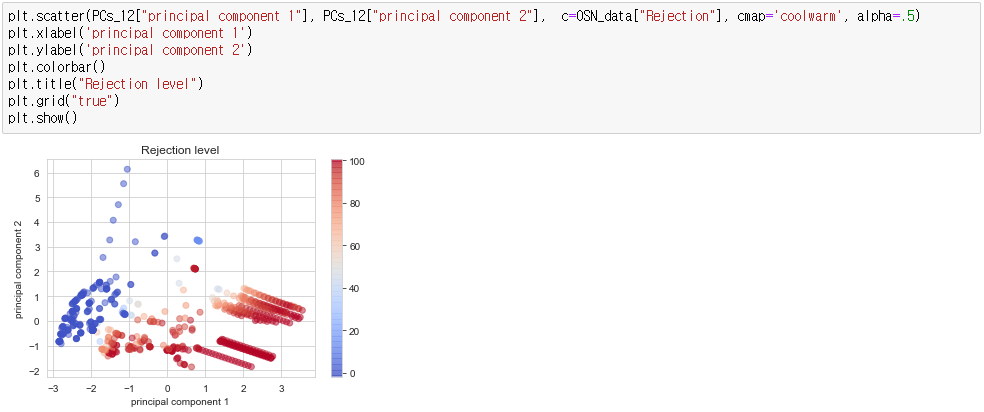
‘plt’ package의 ‘scatter’ 함수를 활용하여 시각화한다. ‘PC\_12’의 첫번째 열과 두번째 열을 좌표로 사용한다.

|  |
| --- |
| plt.scatter(PCs\_12["principal component 1"], PCs\_12["principal component 2"],  c=OSN\_data["Permeance"], cmap='coolwarm', alpha=.5)  plt.xlabel('principal component 1')  plt.ylabel('principal component 2')  plt.colorbar()  plt.title("Permeance level")  plt.grid("true")  plt.show() |



‘plt’ package의 ‘scatter’ 함수를 활용하여 시각화한다. ‘c‘ 함수 인자 (Argument)를 사용하여 ‘OSN\_data’의 Permeance 열을 색으로 표현할 수 있다.

|  |
| --- |
| plt.scatter(PCs\_12["principal component 1"], PCs\_12["principal component 2"],  c=OSN\_data["Permeance"], cmap='coolwarm', alpha=.5)  plt.xlabel('principal component 1')  plt.ylabel('principal component 2')  plt.colorbar()  plt.title("Rejection level")  plt.grid("true")  plt.show() |



‘plt’ package의 ‘scatter’ 함수를 활용하여 시각화한다. ‘c‘ 함수 인자 (Argument)를 사용하여 ‘OSN\_data’의 Rejection 열을 색으로 표현할 수 있다.

#### 가장 데이터 설명력이 높은 두개의 차원에대한 변수 기여도를 확인하라.

1. 다음과 같은 절차를 통해 각 차원에대한 변수 기여도를 확인할 수 있다.

|  |
| --- |
| Feature\_importance = pd.DataFrame(data = pca.components\_, columns =  ['principal component 1', 'principal component 2',  'principal component 3', 'principal component 4',  'principal component 5', 'principal component 6',  'principal component 7', 'principal component 8'],  Index = ['Membrane\_MWCO', 'Solute\_MW', 'Solute\_concentration',  'Solvent\_factor', 'Temperature', 'Process\_configuration',  'Permeance', 'Rejection'])  Feature\_importance |

테이블이(가) 표시된 사진

자동 생성된 설명

‘.components\_’ 함수를 사용하면 ‘pca’의 변수들의 주성분에 대한 기여도를 확인할 수 있다.

각 성분의 이름을 열 이름으로, 행은 변수 이름으로 설정하고 데이터 를 ‘Feature\_importance’로 명명한다.

|  |
| --- |
| Feature\_importance\_pc1 = Feature\_importance.iloc[:,0]  Feature\_importance\_pc1 |

텍스트이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| Feature\_importance\_pc2 = Feature\_importance.iloc[:,0]  Feature\_importance\_pc2 |

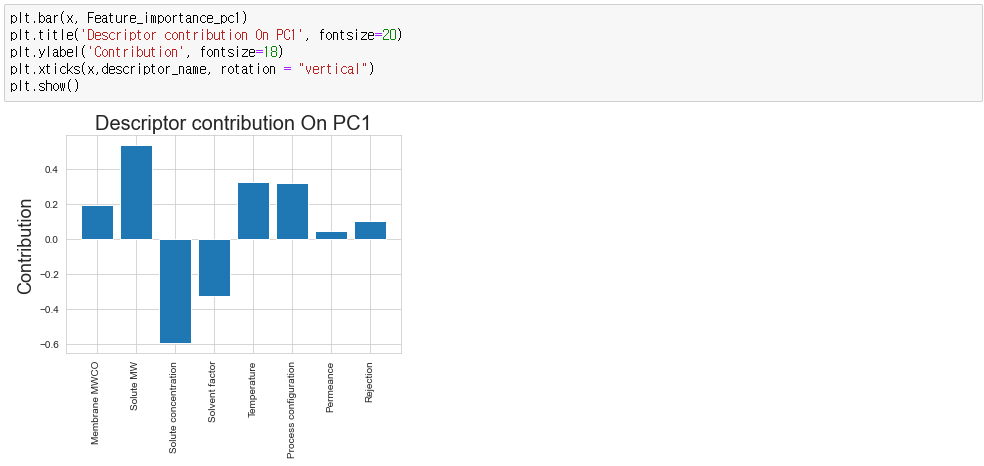
텍스트이(가) 표시된 사진

자동 생성된 설명

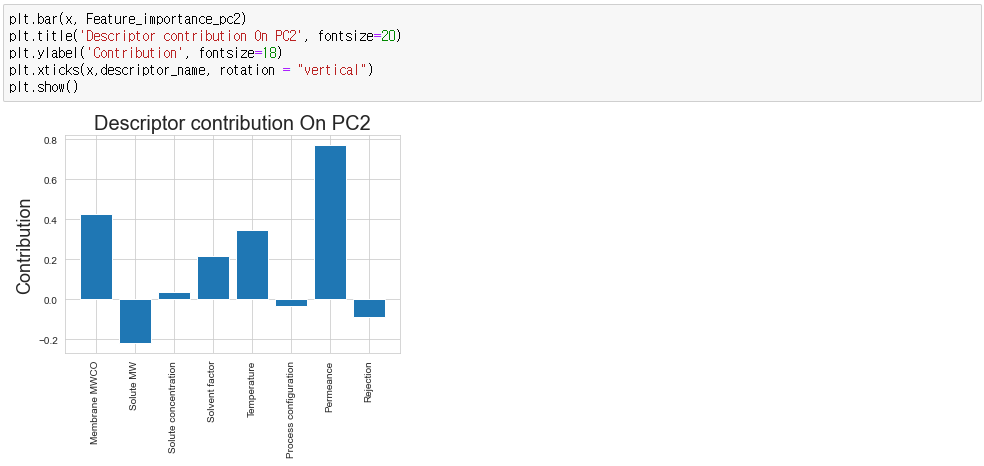
가장 데이터 설명력이 좋은 두 성분 (PC1, PC2)를 추출해 사용한다.

각 PC1과 PC2에대한 기여도 정보를 추출하여 ‘Feature\_importance\_pc1’과 ‘Feature\_importance\_pc2’로 명명한다.

|  |
| --- |
| x = np.arange(8)  descriptor\_name = ['Membrane MWCO','Solute MW','Solute concentration', 'Solvent factor', 'Temperature','Process configuration', 'Permeance', 'Rejection']  plt.bar(x, Feature\_importance\_pc1)  plt.title('Descriptor contribution On PC1', fontsize=20)  plt.ylabel('Contribution', fontsize=18)  plt.xticks(x, descriptor\_name, rotation = "vertical")  plt.show() |
|  |



|  |
| --- |
| plt.bar(x, Feature\_importance\_pc2)  plt.title('Descriptor contribution On PC2', fontsize=20)  plt.ylabel('Contribution', fontsize=18)  plt.xticks(x, descriptor\_name, rotation = "vertical")  plt.show() |



‘np’ package의 ‘array’ 함수를 사용하여 8개의 name space를 갖는 ‘x’ array를 생성한다.

‘plt’ package의 ‘bar’ 함수를 사용하여 ‘Feature\_importance\_pc1’과 ‘Feature\_imporatnce\_pc2’를 시각화한다.

결과로 PC1에서 Solute M.W와 Solute Concentration, PC2에서는 MWCO가 가장 주요한 변수로 식별되었다.

### **[결론]**

본 장에서는 나노 여과 막분리 소재 성능의 화학적 차원 탐색을 위해 주성분 분석 방법론을 활용했다. 화학적 차원에서 정보 차원으로 데이터를 변환 시켰고, 데이터 설명력을 기반으로 데이터 손실과 주요 차원을 확인했다. 결과로, 막분리 소재 성능에 가장 중요한 변수를 규명하기 위해 변수들의 주성분 기여도를 학습했다.

### **학습 결과**

* 학습 내용

유기용매 나노여과 분리막 개발을 위한 주성분 분석 방법론 익히기.

* 학습 결과 확인하기

주성분 분석 알고리즘의 활용 방법 및 변수 기여도 해석 방법 익히기.

* 학습 결과 응용하기

본 장의 학습내용에 기반해 나노여과 분리막의 성능 향상을 위한 특성 이해와 주요 원인 규명하기.

### **1.4 접착용 에폭시 고분자 개발**

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | 에폭시 최적 생산조건 탐색 |
| [방법] | 접착용 에폭시 생산 모사 함수 구축 |
| [응용] | 베이지안 최적화 기반 최적 생산 조건 도출 |
| [요약] | * 접착용 에폭시 생산 반응 이해 * 생산 모사 함수 구축 * 베이지안 최적화를 사용하여 최적의 생산 조건 도출 |

### **[이론] - 능동학습을 사용한 고분자 생산조건 최적화.**

2010년대에 들어서부터 능동학습 프레임워크를 활용한 기능성 신물질 탐색이 점점 각광을 받고 있다. 기계학습을 통한 능동학습(active learning)의 프레임 워크의 하에서의 의사결정 과정의 장점이 기존의 의사결정 과정들과 비교해서 여러 장점이 있기 때문이다. 인간의 의사결정보다는 의사결정과정이 더욱 정교하기 때문에 탐험-탐사(exploration-exploitation)사이의 균형을 고차원의 의사결정에서도 잘 맞출 수 있고, 지도학습 기반 의사결정 보다는 필요한 초기 데이터의 수가 훨씬 적고 또한, 데이터 생성이 시간이 많이 걸리고 비용이 드는 신물질 개발에서 능동학습은 점점 중요도가 올라가고 있다.

능동학습 기반 의사과정은 다음과 같은 세가지 과정들을 거친다. 첫째로는, 현재 존재하는 소수의 데이터를 바탕으로 모델을 구축하여 input-output 사이의 상관관계를 기계가 학습을 한다 (modeling). 둘째로는 기계가 직접 다음 실험 지점을 추천한다 (proposal). 이때 추천이 되는 점은 모델에서의 이해를 바탕으로 우리가 원하는 물성이 최대화가 될 수 있는 지점이면서, 또한 탐색되지 않은 지점의 정보를 얻어낼 수 있는, 즉 탐험-탐사 간의 균형이 맞은 지점이다. 셋째로는 추천된 실험 후보를 사람이 직접 실험을 하여 얻어진 결과를 확인하는 것 이다. 이때 얻어진 결과가 목적에 부합하는지 여부에 따라서 실험이 종결되거나 지속된다.

탐험-탐사간의 균형이 맞은 의사결정을 내리는 방법론중에서 가장 대표적인 방법은 베이지안 최적화이다. 따라서 베이지안 최적화를 접착용 에폭시 개발 문제에 직접 적용하는 것이 이번 실습의 목적이다. 베이지안 최적화의 이론 관한 자세한 설명은 챕터 8, 기계학습과 최적화에 서술되어있다.

베이지안 최적화를 실제 실험 적용해 보면 가장 좋겠지만, 교과서적인 한계로 인해 고분자를 실제로 합성해 나가며 물질 개발을 할 수 없기에, 접착용 에폭시 고분자의 input – output의 상관관계를 Sum-squares function이라는 기준 함수(benchmark function)으로 정의한다. 이때, 함수를 모른다고 가정한 뒤, 초기의 특정한 실험 데이터 (spot data)만을 시작으로 베이지안 최적화 기반의 의사결정을 해 나가면서 의사결정이 효율적으로 이루어짐을 확인한다. 효율성을 보이기 위해, 베이지안 최적화를 진행한 결과와, 불확실성에 대한 고려 없는 greedy-policy (대체 모델의 평균값이 최대화 되는 지점을 실험. 즉, 지도학습 기반 의사결정) 기반 최적화를 진행한 퍼포먼스를 비교한다.

### **[문제] 새로운 기능성 물질을 효율적으로 개발하기 위한 베이지안 최적화 기반의 의사결정 방법론을 접착용 에폭시 고분자 개발이라는 예시에 적용하여, 블랙박스 최적화에서의 효율적인 의사결정을 가이드 해 보자.**

- “part3\_접작용 에폭시 고분자 개발.py 코드를 활용하여라.

- sklearn, scipy 라이브러리를 설치 하여라

- 챕터 8을 읽고 베이지안 최적화의 이론을 이해하여라.

### **[방법] 접착용 에폭시 생산 모사 함수 구축**

2019년에 science and technology of advanced materials 출간된 Sirawit P.의 논문에서는 접착용 에폭시 고분자를 DGEBA 레진, propylene glycol(경화제)의 두가지 재료를 사용하여 접작용 에폭시 고분자를 생성한다. 이러한 시스템에서 조작변인은 4가지로써, 레진과 글리콜의 비율, 레진의 분자량, 경화제의 분자량, 그리고 반응 온도이다. 논문에서는 이러한 상황 하에서 어떠한 비율, 분자량, 반응 온도를 설정해야 가장 접착력이 높은 에폭시 폴리머를 만들 수 있는지를 베이지안 최적화를 사용하여 찾아낸다(실험 가능한 조합의 수: 256). 이를 간단하게 모사하기 위하여 본 실습에서는 2차원의 input을 받아들이는 Real\_epoxy 함수를 정의하여 베이지안 최적화를 진행한다. 와 같은 Sum-squares 함수를 이번 실습의 에폭시 폴리머의 input-output 상관관계라고 정의한다. 이때 최적해 (global maximum)는 (0,0)에서 0의 꼴로 존재하며, 과의 가능한 실험 후보는 -30 ~ 30까지 0.1의 단위로 존재한다 (실험 가능한 조합의 수: ).

#### 관련 라이브러리를 설치하자.

1. 베이지안 최적화를 진행하기 위해선 numpy, sklearn, scipy등의 라이브러리와 GaussianProcessRegressor, WhiteKernel, RBF 등의 함수가 요구되기에, 이를 불러온다.

|  |
| --- |
| import numpy as np from sklearn.gaussian\_process import GaussianProcessRegressor from sklearn.gaussian\_process.kernels import WhiteKernel, RBF import itertools from scipy.stats import norm |

#### 문제 조건대로 에폭시 접착제의 생산 모사 class를 정의해보자.

1. 문제 상황에서 정의한대로의 코드를 구성해보면 다음과 같이 구성할 수가 있다. 이때 이상적인 상황을 가정하여 실험 결과가 특정 input을 넣었을 때 같은 output이 매번 나온다면 .real의 함수를 사용할 수 있고, 현실적인 상황을 가정하여 실험 결과가 noise에 의해 오염된 상황을 실험해보고 싶다면 .noisy 함수를 사용할 수 있다.

|  |
| --- |
| class Real\_epoxy:  def \_\_init\_\_(self):  pass   def real(self, x):  x1 = x[:,0]  x2 = x[:,1]  return -x1\*\*2 - 2\*x2\*\*2   def noisy(self, x, noise\_var):  x1 = x[0]  x2 = x[1]  return -x1\*\*2 - 2\*x2\*\*2 + np.sqrt(noise\_var) \* np.random.standard\_normal()  test\_func = Real\_epoxy() noise\_var = 0.01 GP\_lengthscale = np.sqrt(0.1) search\_space\_org = np.array(list(itertools.product(np.arange(-30,30,0.1), np.arange(-30,30,0.1)))) y\_max\_real = 0 action\_min = np.array([-30, -30]) action\_max = np.array([30, 30]) |

### **[응용] 베이지안 최적화 기반 최적 생산조건 도출**

모사된 에폭시 input-output 함수를 바탕으로 초기 데이터 확보, 데이터 전처리, 획득함수 정의, 그리고 베이지안 최적화를 진행해보자

#### 의사 결정을 위한 초기 데이터셋을 구축해보자.

1. 초기 데이터셋은 전체 가능한 실험 조합인 360,000개의 점들 중에서 20개를 알고 있다고 가정을 한다. 이때, 어떠한 초기 데이터를 안다고 가정 하는지에 따라 베이지안 최적화의 성능이 많이 다를 수 있다. 따라서 베이지안 최적화의 평균적인 성능을 보기 위하여 랜덤하게 20 실험 포인트의 결과를 알고있는 상태(num\_initial\_data)에서 20번 최적화를 진행을 한 뒤(num\_experiments), 이러한 과정을 초기 데이터를 바꾸어가며 10번 진행을 하게끔 세팅을 한다(num\_runs). 이때 매 의사결정마다 최적해로부터 얼마나 떨어진 지점을 탐색했는지 성과를 수치화 하기 위해 regret\_store라는 빈 list를 만들어 둔다.

|  |
| --- |
| np.random.seed(1) num\_runs = 10 num\_experiments = 20 num\_initial\_data = 20 regret\_store = [] initial\_indices\_store = [] for i in range(num\_runs):  random\_indices = np.random.choice(search\_space\_org.shape[0], size=num\_initial\_data, replace=False)  initial\_indices\_store.append(random\_indices) |

#### 데이터 전처리를 위한 class를 설정해보자.

1. 기계학습을 위하여 input-output 로우 데이터(raw data)를 사용하는 것 보다 데이터를 표준화하여 스케일 된 데이터를 사용하면 기계학습의 성능이 더 나아진다는 것은 널리 알려진 사실이다. 따라서 에폭시 모사 시스템의 대체모델과 획득함수를 정의 하기에 앞서 로우 데이터를 표준화된 데이터로 스케일링 할 수 있는 함수와 표준화된 형태로 나타내어진 데이터를 우리에게 친숙한 로우 데이터의 형태로 다시 바꾸는 함수를 정의 하여야 한다.

|  |
| --- |
| class Scaler:  def \_\_init\_\_(self):  pass   def fit(self, x):  self.mu = np.mean(x,0)  self.std = np.std(x,0)   def transform(self, x):  return (x-self.mu)/self.std   def inverse\_transform\_mean(self, x):  return x\*self.std + self.mu   def inverse\_transform\_std(self, x):  return x\*self.std |

#### Expected improvement 기반 획득함수 class를 설정해보자.

1. 챕터 8에서 소개된 expected improvement의 식은 와 같고, 이를 예측 평균치(와 표준편차(를 활용하여 표현하면 다음과 같이 나타낼 수 있다: . 이때 는 기존에 관찰된 데이터 중에서 가장 성능이 좋았던 값이고, 는 표준 정규 누적 분포 함수이고, 는 표준 정규 분포 함수이다. 이를 코드로 구현하면 다음과 같다.

|  |
| --- |
| class Base\_EI:  def \_\_init\_\_(self):  self.scaler\_EI = Scaler()   def evaluate(self, x, model, y\_max, y\_train):  self.scaler\_EI.fit(y\_train)   mean, std = model.predict(x, return\_std=True)  mean = self.scaler\_EI.inverse\_transform\_mean(mean)  std = self.scaler\_EI.inverse\_transform\_std(std)   a = (mean - y\_max).reshape(-1,)  z = a / std  ei = a \* norm.cdf(z) + std \* norm.pdf(z)   index\_next = np.argmax(ei)  X\_next = x[index\_next]  return X\_next, index\_next |

#### 초기 데이터를 사용한 베이지안 최적화를 통해 의사결정을 진행해보고 의사결정들의 평균 퍼포먼스를 print해보자.

1. 사용하는 획득함수를 Q5에서 정의한 expected improvement로 정의한다. 이후, 실험이 진행됨에 따라 초기 데이터에 가우시안 프로세스 모델을 사용하여 대체모델을 구축한다. 이를 바탕으로 획득함수가 최대화 되는 점을 찾고, real\_epoxy 함수에 넣어서 결과값을 얻고, 얻어진 결과값을 최적해와 비교한 성과지표(regret)를 기록하는것을 하나의 에피소드가 끝날 때까지 20번 반복을 하고(num\_experiments), 초기 데이터를 바꾸어 가며 총 10번의 에피소드(num\_run)을 반복하여 베이지안 최적화를 진행한다.

|  |
| --- |
| acq\_func = Base\_EI() kernel = 1.0 \* RBF(length\_scale=GP\_lengthscale) + WhiteKernel(noise\_level=noise\_var) gpr = GaussianProcessRegressor(kernel=kernel, alpha=0.0, optimizer=None, random\_state=0) scaler = Scaler() for i in range(num\_runs):  print(**"run: "** + str(i))  regrets = []  random\_indices = initial\_indices\_store[i]  print(random\_indices)  X\_train = search\_space\_org[random\_indices,:]  y\_train = test\_func.real(X\_train).reshape(-1,1)  search\_space = np.delete(search\_space\_org, random\_indices, axis=0)   y\_max = np.max(y\_train)  regret = y\_max\_real - y\_max  regrets.append(regret)  for j in range(num\_experiments):  print(**"experiment: "** + str(j))  scaler.fit(y\_train)  y\_train\_scaled = scaler.transform(y\_train)  gpr.fit(X\_train, y\_train\_scaled)   X\_next, index\_next = acq\_func.evaluate(search\_space, gpr, y\_max, y\_train)  print(X\_next)  y\_next = test\_func.noisy(X\_next, 0)  search\_space = np.delete(search\_space, index\_next, axis=0)   X\_train = np.vstack((X\_train, X\_next))  y\_train = np.vstack((y\_train, y\_next))   y\_max = np.max(y\_train)  regret = y\_max\_real - y\_max  print(regret)  regrets.append(regret)   regret\_store.append(regrets)  regret\_store = np.array(regret\_store) avg\_regret = np.mean(regret\_store, axis=0) print(avg\_regret) |

이때 print(avg\_regret)의 결과로는 다음과 같은 list가 프린트 된다.

[101.293 99.311 98.47 98.47 97.717 97.717 96.724 96.125 95.136 92.865 92.254 90.46 88.852 87.793 86.667 85.133 84.711 82.068 81.889 80.708 79.627]

즉, 평균적으로 매 time-step마다 베이지안 최적화 기반의 의사결정이 최적해에 근접한 값들을 제시해 나갔다는 것을 알 수 있다.

#### 초기 데이터를 사용한 greedy-policy기반 최적화를 통해 의사결정을 진행해보고 의사결정의 효율성을 베이지안 기반 최적화와 비교해보자.

1. Q6의 성능 지표가 다른 의사결정 방법론보다 좋은지 확인할 대조군 제작 위해, 대체 모델의 불확실성에 대한 고려 없이 예측값의 최댓값만을 실험해 나가는 greedy policy 기반의 의사결정을 진행한 뒤(supervised learning기반 의사결정), 그 결과(avg\_regret)를 베이지안 최적화의 결과와 비교하여 베이지안 최적화의 의사결정 효율성을 인식한다.

|  |
| --- |
| class Base\_EI:  def \_\_init\_\_(self):  self.scaler\_EI = Scaler()   def evaluate(self, x, model, y\_max, y\_train):  self.scaler\_EI.fit(y\_train)   mean, std = model.predict(x, return\_std=True)  mean = self.scaler\_EI.inverse\_transform\_mean(mean)  std = self.scaler\_EI.inverse\_transform\_std(std)   a = (mean - y\_max).reshape(-1,)  z = a / std  ei = a \* norm.cdf(z) + std \* norm.pdf(z)   index\_next = np.argmax(mean)  X\_next = x[index\_next]  return X\_next, index\_next |

코드상으론 기존의 베이지안 최적화 기반 코드와 대부분이 똑같지만, 획득함수를 정의함에 있어서 index\_next = np.argmax(ei)가 아닌, index\_next = np.argmax(mean)로 바꾸어주면 예측값의 최대치를 추천하는 greedy policy 기반의 의사결정 실험이 된다. 이때 결과를 print해보면 다음과 같은 list가 출력된다.

[101.293 100.358 99.959 99.959 98.678 98.678 98.678 97.699 97.348 96.92 96.236 95.887 95.27 94.678 94.312 93.779 93.037 92.675 92.164 91.426 91.083]

Q6의 결과와 비교하여 불확실성을 고려한 베이지안 최적화 기반 의사결정의 데이터 효율성을 입증할 수 있다.

### **[결론]**

본 장에서는 베이지안 최적화 기반 의사결정의 효율성을 모사된 접착용 에폭시 폴리머 생산 공정에 적용하여 능동학습을 실제로 진행해보았고, 이러한 의사결정을 탐욕정책(greedy policy)기반 의사결정과정과 성능을 비교하여 불확실성을 고려한 의사결정과정의 효율성을 몸소 체험해보는 공부를 하였다.

### **학습 결과**

* 학습 내용

접착용 에폭시 폴리머 생산 공정을 모사해보기.

* 학습 결과 확인하기

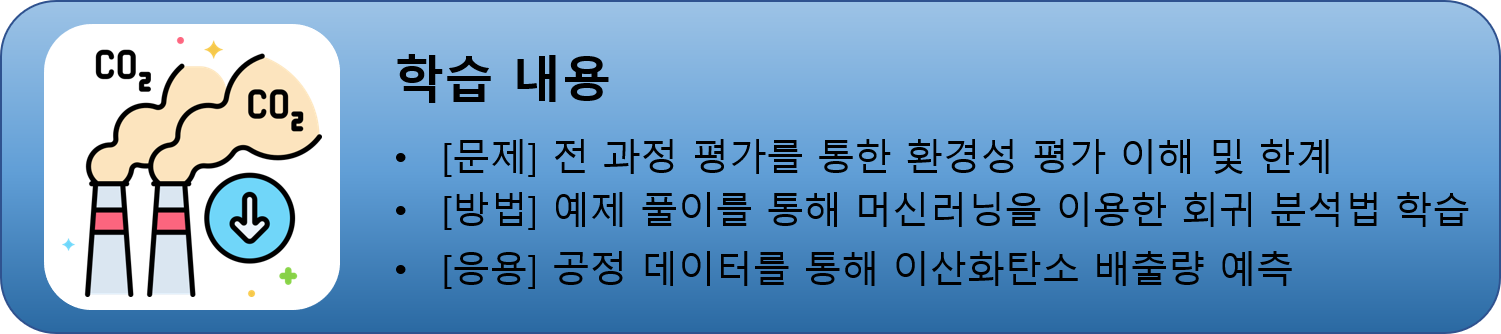
베이지안 최적화 기반 의사결정을 접착용 에폭시 폴리머 공정에 적용해보기.

* 학습 결과 응용하기

베이지안 최적화 기반 의사결정과 탐욕정책 기반 의사결정의 데이터 효율성 비교해보기

### **2.1. 바이오공정 전과정 평가**

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | 전 과정 평가를 통한 환경성 분석 이해 |
| [방법] | 머신러닝을 이용한 전 과정 평가 |
| [응용] | 공정 데이터를 통한 이산화탄소 배출량 예측 |
| [요약] | * 전과정 평가의 방법론 및 중요성 이해 * 전과정 평가의 머신러닝 접목의 필요성 * 머신러닝을 이용해 제지 공장의 이산화탄소 배출량 예측 |



### **[이론] 전과정 평가**

전 과정 평가(life-cycle assessment)는 제품의 생산, 사용, 폐기, 전 과정에 걸쳐 수지(투입물질, 에너지, 배출물질)와 해당 과정에서 미치는 환경 영향을 정량화하는 환경경영기법을 말한다. 전 과정 평가는 공정의 환경성을 진단함으로써 환경라벨링 인증 획득이나 친환경제품 개발 등에서 의사결정을 하는 근거로 활용된다. 특히 화공산업은 대기, 수질, 토양 등 다양한 분야에 넓게 연계되어 있으므로, 지속가능한 산업 발전을 위해서도 환경성 평가는 매우 중요하다.

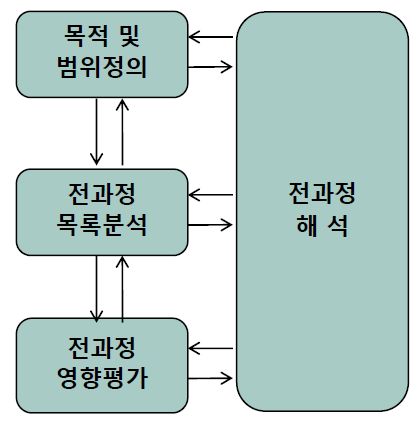


그림 1. 전 과정 평가 방법론 개요 (국제 표준화 기구 제공)

그림 1에 보인 바와 같이, 전 과정 평가는 목적 및 범위 정의, 전 과정 목록 분석, 전 과정 영향 평가, 전 과정 해석의 4단계로 구성된다.

### **목적 및 범위 정의(goal and scope definition)**

목적 및 범위 정의 단계에서는 전 과정 평가를 수행하고자 하는 목적과 대상 (제품 또는 서비스), 데이터 수집범위와 수집방법, 결과형태 및 활용방법을 결정한다. 이 단계에서 전 과정 평가의 기준이 되는 ‘기능적 단위(functional unit)’와 전 과정 평가 수행의 범위는 ‘시스템 경계(system boundary)’를 설정한다.

### **전 과정 목록 분석(life-cycle inventory analysis)**

전 과정 목록 분석은 목적 및 범위 정의 단계에서 설정한 시스템 경계에 대하여 물질수지 데이터를 수집하고, 기능적 단위를 기준으로 정량화하는 단계이다.

### **전 과정 영향 평가(life-cycle impact assessment)**

전 과정 영향 평가는 전 과정 목록 분석을 바탕으로 환경에 미치는 영향을 종합적으로산출하는 단계이다. 각 평가 항목을 산출하는 방법론에는 IPCC, ReCiPe 등이 존재한다.

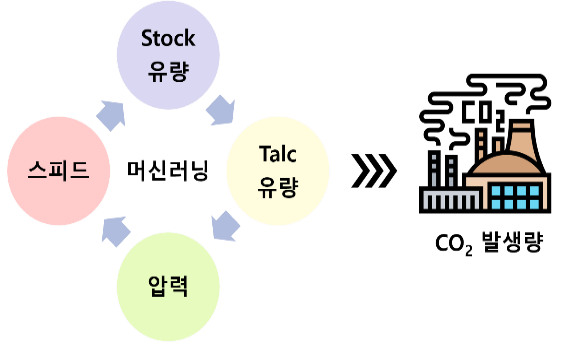
### **전 과정 해석(life-cycle interpretation)**

전 과정 해석은 전 과정 영향 평가 결과를 바탕으로 각각의 평가 항목에 중요한 영향을 미치는 환경인자를 분석하는 단계이다.

본 장에서는 머신 러닝 기반의 회귀분석을 전 과정 영향 평가에 적용하여, 이산화탄소 배출량 예측 모델을 제작함으로써, 공정 변수의 변화에 따른 정확한 온실가스 배출량 산정을 목적으로 한다.

### **[문제]**

**엑셀파일 paperdata.xlsx는 어느 공장의 샘플 운전데이터를 나타낸다. 해당 파일의 각 열은 차례로 stock 유량, talc 유량, 압력, 속도, 이산화탄소 발생량을 나타낸다. stock 유량, talc 유량, 압력, 속도는 독립변수이자 예측변수이며, 이산화탄소 발생량은 출력변수이다. 운전 데이터의 40%를 홀드아웃 검증에 사용한다고 할 때, SVM 회귀모델을 통해 새로운 운전데이터(newdata.xlsx)의 이산화탄소 발생량을 예측하여라.**

****

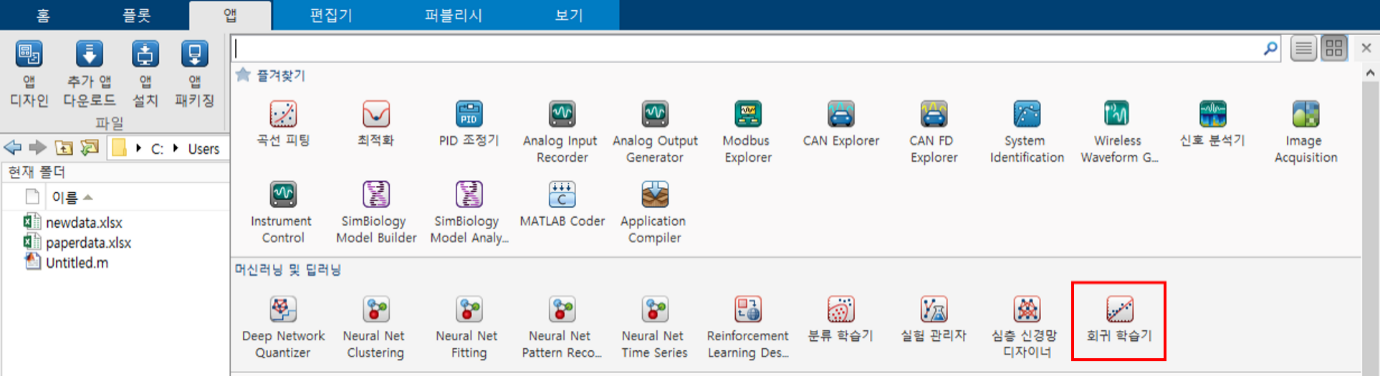
**[방법] 머신 러닝을 통한 회귀분석법**

#### 매트랩에서 SVM회귀분석을 하기 위한 명령어는 무엇이 있으며, 이들의 차이는 무엇인가?

1. SVM 회귀분석을 위한 명령어는 ‘fitrsvm’과 fitrlinear’가 있다. ‘fitsvm’은 비교적 크기가 작은 차원의 데이터 집합들에 대한 SVM 회귀모델을 생성하는 반면, ‘fitrlinear’는 비교적 크기가 큰 차원의 데이터 집합들에 대한 SVM 선형회귀 모델을 형성한다.

#### 명령어를 직접 입력하지 않고, 매트랩 내부의 앱을 통해 회귀분석을 하기 위해서는 어떻게 해야 하는가?

1. A그림 5와 같이 리본 메뉴의 ‘앱’ 탭에서 앱들 중 [머신러닝 및 딥러닝]에서 ‘회귀 학습기’를 선택한다.

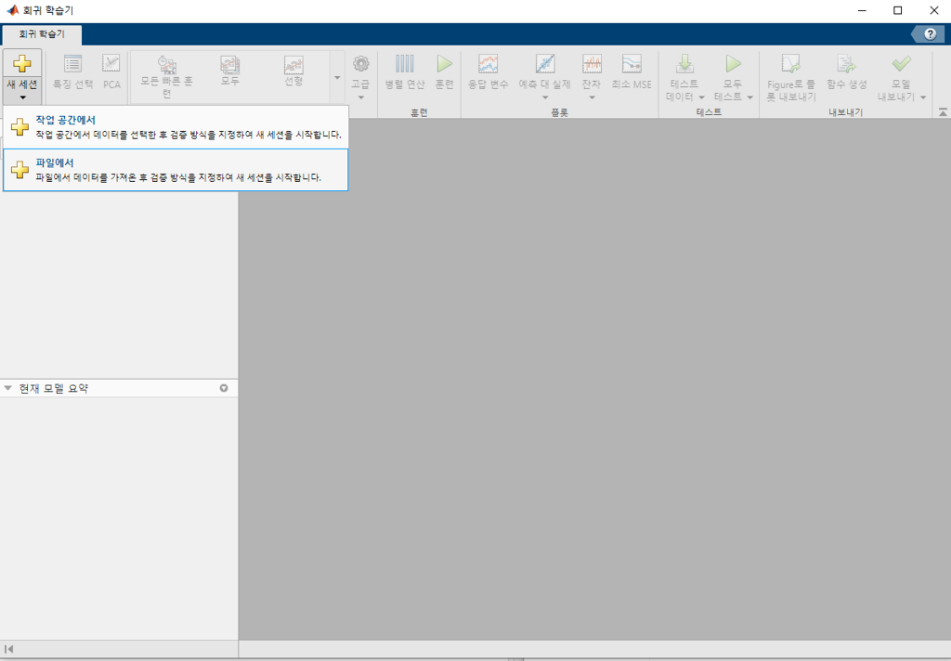


**그림 5. 회귀 분석을 위한 회귀 학습기**

**[응용] 머신 러닝을 통한 이산화탄소 배출량 예측**

#### 매트랩의 회귀 학습기를 사용해 샘플 운전데이터(paperdata.xlsx)를 입력하여라.

1. 그림 6~8과 같은 절차를 통해 샘플 운전데이터를 입력할 수 있다.



**그림 6. 샘플 운전데이터 불러오기**

상단 리본 메뉴에서 ‘새 세션’을 클릭한 후 ‘파일에서’를 선택한다. 그리고 ‘paperdata.xlsx’를 선택한다.

테이블이(가) 표시된 사진

자동 생성된 설명

**그림 7. 불러온 샘플 운전데이터의 항목 설정**

상단 리본 메뉴의 ‘선택 범위/내역’에 필요한 데이터가 모두 선택되었는지 확인한 후, 리본 메뉴 우측의 ‘선택 항목 가져오기’를 클릭한다.

테이블이(가) 표시된 사진

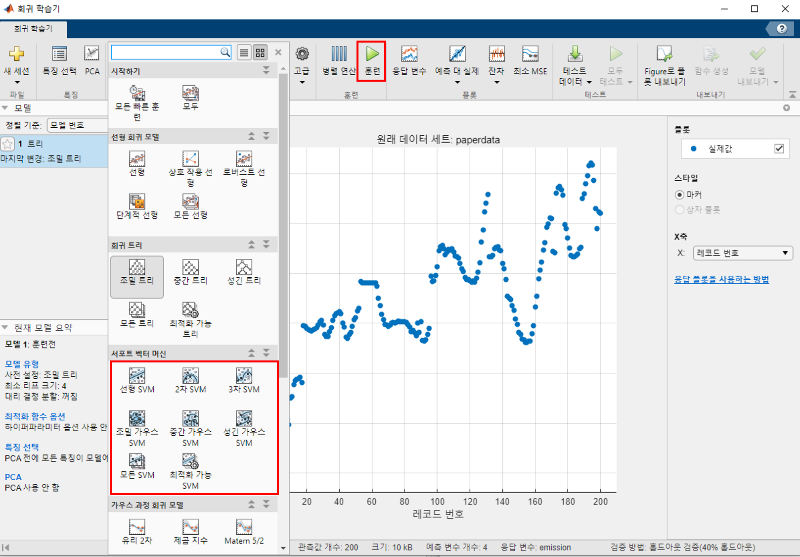
자동 생성된 설명

**그림 8. 응답변수 및 검증 방법 설정**

응답변수에 이산화탄소 발생량이 설정된 것을 확인한다. 그 후 검증 방법에서 ‘홀드아웃 검증’을 선택하고, ‘홀드아웃 비율’에는 40을 입력한 후 ‘세션 시작’을 클릭한다.

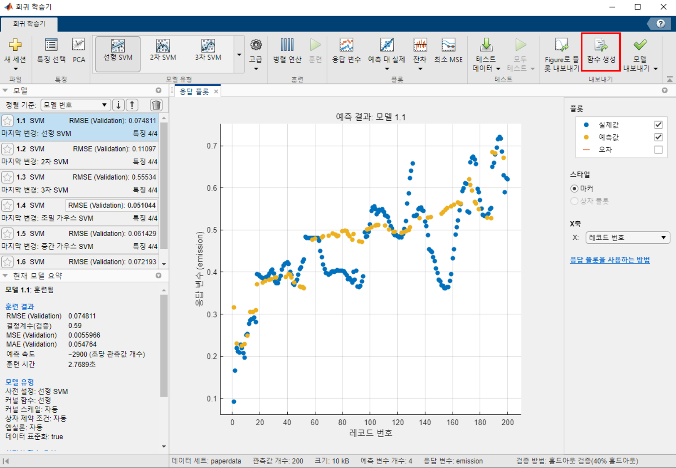
#### 입력한 샘플 운전데이터를 이용해 SVM 회귀 모델을 학습하여 이산화탄소 발생량 예측 함수를 생성하여라.

1. SVM 회귀 모델을 학습 및 이산화탄소 발생량 예측 함수를 생성은 아래의 그림9와 그림 10과 같은 절차를 통해 가능하다.



**그림 9. SVM 회귀 모델 설정 및 훈련**

모델 유형에서 SVM 모델을 선택한다. 특정 SVM 모델을 선택할 수도 있고, ‘모든 SVM’을 통해 모든 종류의 SVM을 동시에 학습시킬 수 있다. (이번 예제에서는 ‘모든 SVM’을 선택하였다.) 모델을 선택한 후, 상단의 훈련 버튼을 클릭하면 SVM 회귀 분석 모델이 생성된다.

 텍스트이(가) 표시된 사진

자동 생성된 설명

**그림 10. 훈련된 모델의 함수 생성**

리본 메뉴에서 ‘함수 생성’을 클릭하면 오른쪽 그림과 같이 코드가 생성된 것을 확인할 수 있다.

#### 생성한 SVM 회귀 함수를 이용해 새로운 운전데이터(newdata.xlsx)의 이산화탄소 발생량을 예측하여라.

1. 그림 11~13과 같은 절차를 통해 새로운 운전데이터의 이산화탄소 발생량을 예측할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**그림 11. 샘플 운전데이터 지정 및 범위 입력**

샘플 운전데이터를 불러와 데이터 범위를 지정하여 trainingData로 입력한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**그림 12. 생성된 모델을 이용한 새로운 운전데이터의 이산화탄소 발생량 예측**

샘플 운전데이터와 마찬가지로 새로운 운전데이터를 불러와 범위를 지정한다(T로 설정). 이를 통해, 샘플 운전데이터로 학습한 모델을 통해 새로운 운전데이터의 이산화탄소 발생량을 예측한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**그림 13. 모델을 통해 예측한 새로운 운전데이터의 이산화탄소 발생량**

### **[결론]**

전 과정 평가는 공정의 환경성을 판단할 수 있는 분석 방법 중 하나로, 기후 위기가 심각해짐에 따라 그 중요성이 더욱 강조되고 있다. 본 장에서는 전 과정 평가와 머신 러닝을 접목함으로써 효과적으로 이산화탄소의 발생량을 예측하였다. 주어진 예제에서는 계산의 용이성을 위해, 예측에 200개의 샘플을 이용하였으나, 더 많은 샘플을 이용할 경우 정확도를 크게 향상시킬 수 있다. 또한, 전 과정 평가에서 이산화탄소 배출량 뿐만 아니라 연료 소모량, 오존 발생량 등 다양한 환경 지표를 함께 산정하였다.

### **[학습 결과]**

* 학습 내용

환경성 평가를 위한 전 과정 평가의 방법론 및 이해

* 학습 결과 확인하기

머신러닝을 이용한 회귀 분석 절차 및 방법을 익히기

* 학습 결과 응용하기

본 장의 학습내용에 기반해 회귀분석함으로써 새로운 공정에서 발생하는 이산화탄소 배출량을 예측

### **2.2. 단원자 증착 공정 설계**

### **단원자 증착법**

증착 공정 (Deposition process)은 반도체 소자 재조를 위한 단위공정들 중 하나로 반도체 산업에 핵심적인 기술이다. 보통 증착 공정은 단결정의 규소기판에 반도체, 절연체, 도체등의 재료를 박막의 형태로 증착시켜 집적회로의 미세구조 형성하는데 기여한다. 일반적으로 증착 공정은 기판위 회로 규모에 가장 큰 영향을 주므로, 소자의 집적도에 가장 큰 기여를 한다. 따라서 고집적 박막 형성을 위한 효율적이고 섬세한 증착 공정 개발이 차세대 반도체 개발을 위해 핵심적이다.

기존의 반도체 공정에서의 증착법은 전통적으로 물리기상증착법 (Pysical vapor deposition, PVD)과 화학기상증착법 (Chemical vapor deposition, CVD)에 의존한다. PVD는 도포에 사용할 물질들을 높은 에너지나 열에 노출시켜 입자를 기판 표면에 방사시켜 증착하는 방식이다. 이 기술은 높은 생산성을 갖고 준수한 박막 두께 조절력을 갖는다. CVD는 기질 표면에서 반응기체를 증착시키는 방법으로, 박막 두께 조절에 우수하고 적절한 생산성을 갖는다. 하지만 CVD와 PVD는 모두 200 Å 이하의 박막 증착에는 사용할 수 없는데, 반도체 박막 증착 두께가 높은 집적도의 소자 개발에 필수적임으로 새로운 기술이 요구되는 실정이다. 단원자 증착법 (Atomic layer deposition, ALD)은 소자 미세화와 집적도 향상을위한 가장 이상적인 무기 및 금속 박막 증착 기술이다. ALD은 원자 단위에서 단원자층을 증착하는 기술로써 200 Å 이하의 박막 형성이 가능하여 고용량 반도체 소자 개발에 핵심적이다. 또한, 3차원 구조의 반도체 제조 공정에 응용할 수 있어 반도체 생산에 큰 각광을 받고있다.

ALD은 기판과의 전구체 물질 (Precursor) 사이의 선택적 표면 화학반응을 통한 반응물 분자의 증착을 유도한다. 이를 위해 밀도범함수 이론 (Density functional theory, DFT), kinetic Monte Carlo 방법 등이 기판-전구체 사이의 반응 예측과 해석에 이용된다. 하지만, 수많은 전구체 물질, 반응물, 그리고 기판 사이의 현상 실험은 비용과 시간의 한계점이 명확히 존재한다. 따라서 비효율적인 단원자 증착법을 탈피해 실용화를 위한 고성능 물질 탐색을 위해 혁신이 요구된다.

서포트 벡터 회귀 (Support vector regression)은 고차원 데이터 식별을 위한 서포트 벡터 머신 (Support vector machine) 으로부터 발달한 인공지능 알고리즘이다. 서포트 벡터 회귀는 단순한 하이퍼파라미터 구조를 통해 신속한 모델 생성이 가능한 특징을 지녔다. 오늘날 서포트 벡터 회귀는 다양한 모델과 결합된 hybrid 형태로 화학공정의 많은 복잡계 문제를 해결하고 있다.

본 실습에서는 탈실험화된 단원자 증착 공정 설계를 위해 서포트벡터머신 기반의 전구체 성능 예측 모델을 개발한다.

### **[문제]**

**고성능 반도체 소재 개발에 핵심적인 단원자 증착 과정 제어를 위해 서포트 벡터 머신을 활용해 증착 두께 예측 모델을 개발하라.**

**- “코드 및 데이터/4-3. ALD.csv” 데이터를 활용하여라.**

**- 서포트 벡터 머신의 하이퍼파라미터를 이해하고 정확도 개선을 위한 하이퍼파라미터 튜닝을 시도하라.**

### **[방법] SVM 예제**

#### SVM을 활용하기 위한 라이브러리를 조사하라. SVM을 regression 형태로 사용하기위한 명령어는 무엇인가?

1. SVM 활용을 위한 라이브러리로 ‘e1071’이 있다. SVM은 input데이터 형식에 따라 자동으로 classification/regression을 설정한다. 따라서 ‘svm()’함수를 불러오고, 데이터를 독립/종속변수를 설정한다.

|  |
| --- |
| Library(‘e1071’)  Model\_SVM <- e1071::svm(Data[,:1:x], Data[,x], gamma = , cost = ) |

#### SVM의 학습은 하이퍼파라미터에 따라 다른 성능의 예측모델을 생성한다. SVM의 하이퍼파라미터는 무엇인가? 하이퍼파라미터가 의미하는 것은 무엇인가?

1. SVM의 하이퍼파라미터에는 ‘gmma’와 ‘cost’가 있다. ‘gamma’는 데이터 샘플의 영향력 행사 거리 (분산)을 가정한다. ‘gamma’ 가 클수록 데이터가 다른 데이터에 더 큰 영향을 줄 수 있다고 판단한다. ‘cost’는 데이터 샘플의 최대/최소 허용 클래스 수를 가정한다. ‘cost’가 크면 이상치의 가능성을 매우 작게 판단한다.

### **[응용] SVM 기반 고성능 전구체 물질 예측 모델 개발**

예제는 R 4. 0. 2 프로그래밍 언어를 기준으로 Rstudio 개발환경에서 작성되었다. 예제 실습을 위해 Anaconda 프로그램을 아래 URL로부터 다운로드할 수 있다.

https://www.anaconda.com/

#### 데이터를 Jupyter환경으로부터 불러오고 데이터를 확인하라.

1. 다음과 같은 code를 사용하여 불러올 수 있다.

|  |
| --- |
| DFT\_data <- read.csv(file.choose(), header = T) |

‘read.csv’ 함수를 사용하여 데이터 파일이 저장된 장소를 직접 찾아 ‘DFT\_data’ 이름으로 데이터를 불러온다. 해당 데이터는 column 이름이 이미 존재하는 데이터로 ‘header = T’ arg를 통해 이를 밝힌다.

|  |
| --- |
| str(DFT\_data) |

‘str()’ 함수를 사용하여 데이터의 형태를 확인할 수 있다.

|  |
| --- |
| DFT\_data |

테이블이(가) 표시된 사진

자동 생성된 설명

불러온 데이터를 확인해본다.

#### 인공지능 학습을 위해서 데이터의 단위를 무시한 상대적 영향력을 파악할 필요가 있다. DFT데이터를 정규화하라.

1. 다음과 같은 절차를 통해 정규화한다.

|  |
| --- |
| DFT\_wo\_NA <- data.frame(DFT\_data[,1:2], DFT\_data[,4:28]) |

데이터의 3번째 열은 전구체 물질의 이름이므로 이를 제거한다. 추출된 데이터는 ‘DFT\_wo\_NA’로 명명한다.

|  |
| --- |
| DFT\_wo\_NA\_symbol <- data.frame(DFT\_wo\_NA)  DFT\_wo\_NA\_symbol\_scale <- data.frame(scale(DFT\_wo\_NA\_symbol)) |

데이터를 ‘DFT\_wo\_NA’로 명명하고 ‘scale()’ 함수를 사용해서 정규화한다. 정규화된 데이터는 ‘DFT\_wo\_NA\_symbol\_scale’로 명명한다

#### 9개의 종속변수 각각 예측하는 SVM 모델을 생성할 것이다. 데이터를 이에 맞춰 나누어라.

1. 다음과 같은 절차를 통해 데이터를 나눌 수 있다.

|  |
| --- |
| DFT\_EadH <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,10])  DFT\_EadOH <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,11])  DFT\_EadNH2 <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,12]) |

데이터의 종속변수인 3가지 표면에서의 adsorption 에너지 (Ead\_H, Ead\_OH, Ead\_NH2)에 대해 각각 독립변수-종속변수 관계를 규명한 데이터를 생성한다

|  |
| --- |
| DFT\_EaH <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,13])  DFT\_EaOH <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,14])  DFT\_EaNH2 <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,15]) |

데이터의 종속변수인 3가지 표면에서의 active barrier 에너지 (Ea\_H, Ea\_OH, Ea\_NH2)에 대해 각각 독립변수-종속변수 관계를 규명한 데이터를 생성한다

|  |
| --- |
| DFT\_dEH <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,16])  DFT\_dEOH <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,17])  DFT\_dENH2 <- data.frame(DFT\_wo\_NA\_symbol\_scale[,1:9], DFT\_wo\_NA\_symbol\_scale[,18]) |

데이터의 종속변수인 3가지 표면에서의 dE (dE\_H, dE\_OH, dE\_NH2)에 대해 각각 독립변수-종속변수 관계를 규명한 데이터를 생성한다

#### SVM 학습을 위해 데이터를 학습/검증 데이터로 나누어야 한다. 학습/검증 데이터를 각각 70%와 30%로 나누어라.

1. 다음과 같은 절차를 통해 데이터를 나눌 수 있다.

|  |
| --- |
| smp\_size <- floor(0.7 \* nrow(DFT\_EadH))  set.seed(1)  train\_ind <- sample(seq\_len(nrow(DFT\_wo\_NA)), size = smp\_size) |

‘floor()’ 함수를 사용하여 ‘WGSR\_scale’ 데이터의 행 개수의 70%에 해당되는 값을 생성한다. 이 값은 ‘smp\_size’로 명명하여, 후에 무작위 데이터 추출에 사용한다.

‘sample()’ 함수를 사용해서 ‘WGSR\_scale’ 데이터로부터 행 번호를 무작위 추출을 진행한다. ‘size’ arg를 ‘smp\_size’로 설정하여 총 70% 데이터를 추출한다. ‘seq\_len()’ 함수를 통해 1부터 임의의 지정된 숫자까지 순차 데이터를 생성할 수 있다.

|  |
| --- |
| SVM\_train <- DFT\_EadH[train\_ind,]  SVM\_test <- DFT\_EadH[-train\_ind,]  SVM\_train <- DFT\_EadOH[train\_ind,]  SVM\_test <- DFT\_EadOH[-train\_ind,]  SVM\_train <- DFT\_EadNH2[train\_ind,]  SVM\_test <- DFT\_EadNH2[-train\_ind,] |

SVR의 학습과 검증을 위해 세가지 표면에서의 adsorption 에너지 대한 데이터를 생성한다.

|  |
| --- |
| SVM\_train <- DFT\_EaH[train\_ind,]  SVM\_test <- DFT\_EaH[-train\_ind,]  SVM\_train <- DFT\_EaOH[train\_ind,]  SVM\_test <- DFT\_EaOH[-train\_ind,]  SVM\_train <- DFT\_EaNH2[train\_ind,]  SVM\_test <- DFT\_EaNH2[-train\_ind,] |

SVR의 학습과 검증을 위해 세가지 표면에서의 active barrier 에너지 대한 데이터를 생성한다.

|  |
| --- |
| SVM\_train <- DFT\_dEH[train\_ind,]  SVM\_test <- DFT\_dEH[-train\_ind,]  SVM\_train <- DFT\_dEOH[train\_ind,]  SVM\_test <- DFT\_dEOH[-train\_ind,]  SVM\_train <- DFT\_dENH2[train\_ind,]  SVM\_test <- DFT\_dENH2[-train\_ind,] |

SVR의 학습과 검증을 위해 세가지 표면에서의 dE 대한 데이터를 생성한다.

#### 위에서 만든 데이터를 사용하여 SVM 모델을 생성할 것이다. Gamma = 0.1, cost = 10 환경에서 SVM 모델을 생성하라.

1. 다음과 같은 절차를 통해 SVM을 생성할 수 있다.

|  |
| --- |
| library(e1071)  library(Metrics) |

SVR 모델을 내장하고있는 ‘e1071’ package를 import한다.

SVR 모델의 검증을 예측 정확도 평가 지표를 내장하고 있는 ‘Metrics’ package를 import 한다.

|  |
| --- |
| model\_SVM <- e1071::svm(SVM\_train[,1:9], SVM\_train[,10],  gamma = 0.001, cost = 60) |

SVR의 학습을 진행한다. Gamma와 cost는 ‘svm()’ 함수의 ‘arg’로, 하이퍼파라미터에 해당한다. 이는 여러 번의 시행착오를 통해 얻도록 한다.

|  |
| --- |
| summary(model\_SVM) |

텍스트이(가) 표시된 사진

자동 생성된 설명

‘summary()’ 함수를 통해 모델의 정보를 얻을 수 있다..

#### 모델의 예측 정확도를 파악하라. 예측 정확도는 R2, MSE, RMSE, 그리고 MAE 지표를 사용해 판단하라.

1. 다음과 같은 절차를 통해 정확도를 평가할 수 있다.

|  |
| --- |
| pred\_train <- predict(model\_SVM, SVM\_train[,1:9])  pred\_test <- predict(model\_SVM, SVM\_test[,1:9]) |

SVR의 모델을 통해 학습 데이터와 검증 데이터를 예측한다.

|  |
| --- |
| rsq <- function (x, y) cor(x, y) ^ 2 |

예측 정확도 평가를 위한 R2 계산식을 만든다. 해당 함수는 ‘rsq()’로 명명한다.

|  |
| --- |
| rsq(pred\_train,SVM\_train[,10])  Metrics::mse(pred\_train,SVM\_train[,10])  Metrics::rmse(pred\_train,SVM\_train[,10])  Metrics::mae(pred\_train,SVM\_train[,10])  rsq(pred\_test,SVM\_test[,10])  Metrics::mse(pred\_test,SVM\_test[,10])  Metrics::rmse(pred\_test,SVM\_test[,10])  Metrics::mae(pred\_test,SVM\_test[,10]) |

학습 데이터 예측에 대한 정확도와 검증 데이터 예측에 대한 정확도를 계산한다.

|  |
| --- |
| plot(pred\_train, SVM\_train[,10])  plot(pred\_test, SVM\_test[,10]) |

예측 값과 실제 값을 시각화 한다.

### **[결론]**

본 장에서는 증착 두께 제어가능한 단원자 증착법을 위해 서포트 벡터 머신을 활용하여 증착 두께 예측 모델을 개발했다. 이를 위해 데이터를 표준화하고 예측 모델 학습을 위한 데이터 구조를 설계했다. 예측 정확도 평가를 위해 통계적 정확도 지표를 사용과 정확도 개선을 위한 하이퍼파라미터 조사를 학습했다. 결과로, 서포트 벡터 머신을 활용한 증착 두께 예측 모델 개발 방법을 학습할 수 있었다.

### **학습 결과**

* 학습 내용

반도체 소재 개발을 위한 SVM 방법론 기반 단원자 증착 설계 익히기.

* 학습 결과 확인하기

SVM 알고리즘의 활용 방법 및 예측 결과 해석 방법 익히기.

* 학습 결과 응용하기

본 장의 학습내용에 기반해 단원자 증착을 위한 전구체 물질 탐색 및 효과적 표면 화학반응 설계를 위한 전구체 물질 성능 예측하기.

### **3.1. 천연가스를 사용한 수소 생산 공정 최적화**

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | 수소 생산을 위한 수증기 개질공정 이해 |
| [방법] | 인공신경망 모델 구축 및 최적화 |
| [응용] | 인공신경망 기반 수증기 개질공정 최적 변수 탐색 |
| [요약] | * 수증기 개질 공정 개념 및 주요 반응식 설명 * 운전 데이터 기반의 인공신경망 모델 개발 * 예측 모델을 이용한 운전조건 최적화 수행 |

**[공정설명] 수증기 개질 공정의 이해**

전 세계적으로 다양한 환경문제에 대응하기 위해서, 이산화탄소 등 온실가스 배출을 줄이기 위해 발전 및 수송 산업에서 석탄과 석유등의 탄화수소 사용을 지양하고 신재생에너지로의 에너지 전환을 추진하고 있다. 신재생에너지들 중 수소는 탄소중립을 실현하기 위한 친환경 에너지로서 큰 주목을 받고 있다. 환경과 경제적인 측면에서 수소 경제로 나아가야 한다는 전문가들의 목소리가 커지고 있으나, 수소경제의 실현을 위해서는 수소의 안정적인 생산과 공급을 위한 기반을 먼저 구축하는 것이 선행되어야 한다.

수소에너지는 생산방식에 따라 회색(또는 그레이), 청색(또는 블루), 녹색(또는 그린)의 세 가지 색깔로 분류된다. 먼저, 회색 수소는 천연가스와 같은 화석연료를 기반으로 수소를 생산하는 방식은 부산물로 이산화탄소를 함께 배출하는 특징을 갖는다. 현재 생산되고 있는 대부분의 수소는 회색 수소다. 다음으로, 청색 수소는 회색 수소를 생산하는 공정에서 발생하는 이산화탄소를 포집하고 활용 및 저장하는 기술을 접목한 것을 의미한다. 청색 수소는 회색 수소에 비해 대기 중으로 배출하는 이산화탄소량이 적어 환경적으로 친화적이지만 탄소배출을 완전히 제거할 수는 없는 한계가 있다. 마지막으로, 녹색 수소는 세 가지 수소중에 환경적으로 가장 깨끗한 수소를 의미하며 신재생 에너지를 활용한 물의 전기분해 방식을 통해 생산된다. 녹색 수소의 환경적 이점은 가장 크지만 생산 비용과 제한적인 생산량 등을 고려했을 때 아직 실용적이지 않은 한계가 있다.

완전한 탄소중립에 도달하려면 앞으로 오랜 기간 동안에 회색 수소에서 녹색 수소로 점진적으로 발전해 나가야 한다. 아직까지는 수소를 대량으로 그리고 경제적으로 생산할 수 있는 방법은 탄화수소의 개질(reforming)을 통한 수소 생산 공정이 지배적이다. 따라서 녹색 수소로 전환하는 단계에 앞서 회색 수소 생산 공정의 최적화를 통해 더 경제적이고 친환경적인 공정운전을 수행하는 것이 필요하다.

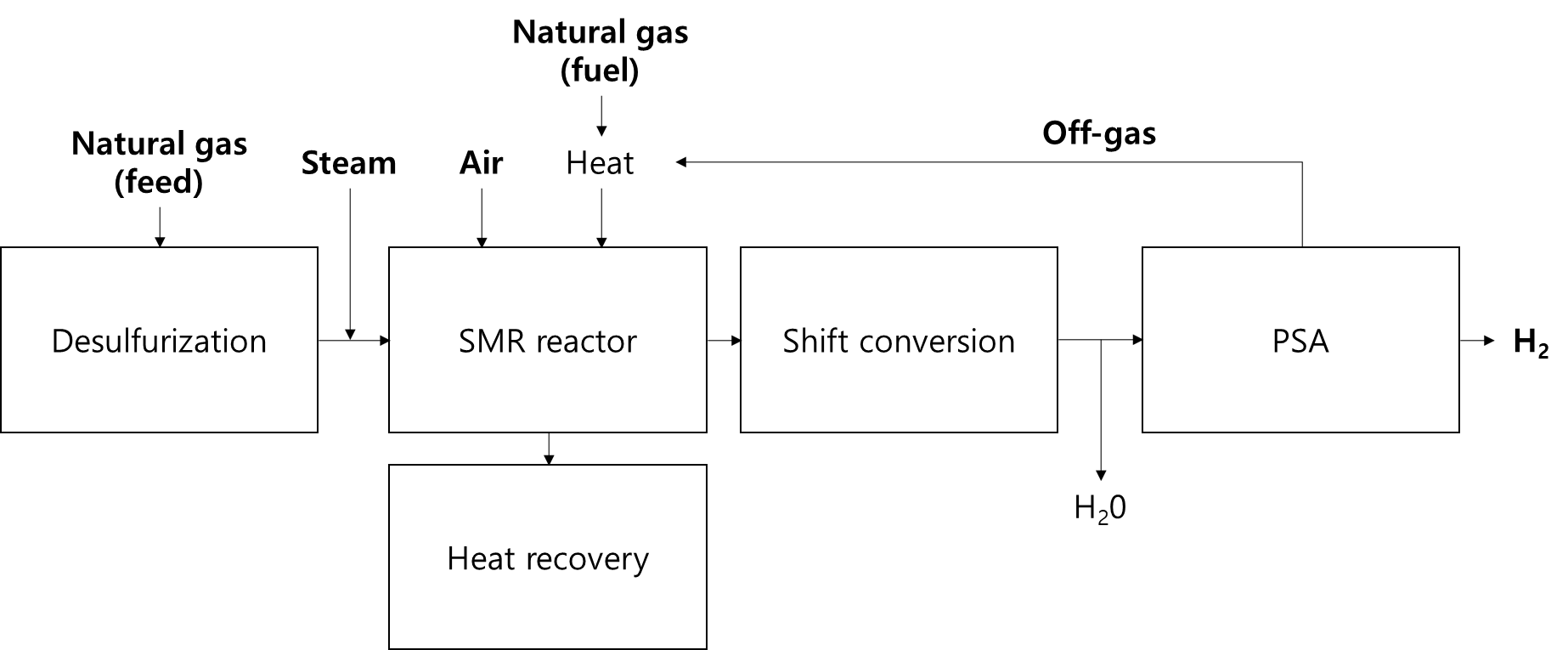


그림 1. 수증기 개질 공정의 흐름도

수증기 개질(steam methane reforming, SMR) 공정은 크게 주요 반응이 이루어지는 SMR 반응기, 황성분을 제거하는 탈황장치(hydrodesulfurization, HDS), 잔존 CO를 H2로 전환하는 전환반응기(shift reactor), 생성된 합성가스로부터 고순도의 수소를 분리하는 압력 스윙 흡착(pressure swing adsorption, PSA) 장치, 열회수를 위한 열교환기(heat exchanger)로 이루어져 있다 (그림 1).

원료인 천연가스는 한국 전역에 설치되어 있는 천연가스배관 인프라를 통하여 공급된다. 천연가스는 C1~C5의 혼합물로 구성되어 있으며 주성분은 CH4로 약 93%를 차지한다. 천연가스는 일부 황을 포함하고 있는데, 황 성분은 촉매독 역할을 하여 촉매에 큰 악영향을 가하므로 원료 투입 전에 제거 되어야한다. 원료인 천연가스는 약 300℃로 예열되어 CoMo와 ZnO 촉매로 충전된 탈황장치로 투입되고 황 성분이 제거된다. 또한, 각종 불순물이 제거된 물은 열교환기를 통하여 180℃ 이상으로 승온 되어 과열 증기상태로 공급된다. 결과적으로 황 성분이 제거된 천연가스와 과열 스팀이 섞인 후 약 580℃의 조건으로 승온되어 SMR 반응기로 투입된다. SMR 반응기에서 아래의 세가지 주요 반응이 진행되며 수소가 생성된다. 전체적인 SMR 반응은 강한 흡열반응이며 약 700~800℃, 그리고 8barg 조건에서 반응이 일어난다.

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |
|  |  | (3) |

반응 종료된 합성가스는 CO, CO2, CH4, H2, H2O로 이루어지며 약 750℃ 조건으로 SMR 반응기를 빠져나와 후단 공정에 열을 전달한다. 이 합성가스는 180–250℃ 조건으로 저온수성가스화(low temperature shift, LTS) 반응기로 투입된다. LTS 반응기에서는 합성가스에 잔존하고 있는 CO를 제거하는 반응이 일어난다. CO는 수소 연료전지에 악영향을 미치기 때문에 최대한 제거하는 것이 좋다. 또한 LTS 반응기에서 CO가 제거되며 추가적인 H2를 생성하는 장점이 있다. LTS 반응을 거치고 나온 흐름은 40℃까지 온도가 낮춰지고, 물이 제거된 합성가스를 순도 99.999%, CO농도 0.2ppm 이하의 스펙을 갖는 수소로 분리하기 위해서 PSA 장치에 투입된다.

### **[문제] 투입되는 천연가스, 스팀, 공기의 유량과 PSA 장치의 회수율에 따라서 최종적으로 생성되는 수소의 유량을 계산하고 SMR 공정의 최적운전조건을 구하라.**

- “2. 코드 및 데이터/7-1. SMR process.csv” 데이터를 활용하여라.

- 학습 및 평가 데이터는 7:3 비율로 분리하여라.

- 민감도 분석은 평균값의 5% 범위 내에서 진행하여라.

### **[방법] SMR 공정의 인공신경망 모델 구축**

#### 데이터를 작업 환경으로 불러오자. 데이터의 불러오기와 저장하기 등 데이터 작업에 유용한 파이썬 라이브러리는 무엇인가?

1. 일반적으로 데이터는 Excel과 같은 관계형 데이터베이스에 많이 들어있다. 데이터는 pandas 라이브러리를 이용하여 다음 명령어를 통해 읽어 들인다.

|  |
| --- |
| import pandas as pd  data\_smr = pd.read\_csv(“./smr\_process.csv”, header=0) |

pd.read\_csv 함수는 해당경로의 데이터를 불러와 데이터프레임 형태로 저장한다. 여기서 “header=0”은 첫 행을 열 이름으로 지정한다는 의미이다.

#### 데이터의 기본 정보를 파악하여라. 데이터 샘플 개수, 특성 개수, 각 특성의 타입, 최솟값, 최댓값, 평균값은 어떻게 되는가?

1. head() 사용하여 불러온 데이터의 정보를 확인해보자.

|  |
| --- |
| data\_smr.head() |

테이블이(가) 표시된 사진

자동 생성된 설명

각 행은 하나의 데이터 샘플을 나타낸다. head() 괄호 안에 숫자를 입력하면 처음부터 그 숫자만큼의 행을 보여주고, 기본 값은 5개로 설정 되어있다.

info()는 데이터에 대한 간단한 설명을 나타낸다. 데이터 샘플의 수, 각 특성의 데이터 타입, 결측치 개수 등의 정보를 파악할 수 있다.

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

텍스트, 영수증이(가) 표시된 사진

자동 생성된 설명

describe()는 숫자형 데이터의 통계 정보를 보여준다.

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

테이블이(가) 표시된 사진

자동 생성된 설명

count는 결측치를 제외한 데이터 샘플의 개수, mean, min, max는 각 데이터 특성의 평균값, 최솟값, 최댓값을 나타낸다. std 행은 값이 퍼져 있는 정도를 측정하는 표준편차이다. 25%, 50%, 75% 행은 백분위수를 나타낸다.

#### 신경망 모델의 학습과 평가를 위해 데이터를 분리하여라. 학습 데이터와 평가 데이터의 차이점은 무엇인가?

1. 모델 훈련과 평가를 진행하기 위해서 사이킷런의 train\_test\_split 함수를 이용하여 훈련 데이터와 평가 데이터를 분리한다. 학습 데이터는 모델 학습을 할 때 사용하는 데이터이며, 평가 데이터는 학습된 모델의 성능을 평가하는 데이터이다. 평가 데이터는 모델의 일반화 오류 정도를 평가하기 위해 마지막에 한번만 사용되며, 이를 통해 모델의 과적합을 방지할 수 있다.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  test\_ratio = 0.3  random\_state = 42  data\_train, data\_test = train\_test\_split(data\_modeling, test\_size = test\_ratio,  shuffle=True, random\_state=random\_state) |

test\_ratio 값은 데이터 샘플들로부터 테스트 데이터셋을 얼마만큼 분리할 지를 결정한다. 여기서는 전체의 30% 만큼 테스트 데이터셋으로 분리하였다. random\_state는 랜덤으로 분리되는 데이터를 다음 시행에서도 동일하게 분리하기 위해(동일한 실험 결과를 얻기 위해서) 고정하는 값으로, 42로 설정되었다.

#### 모델의 입력변수와 출력변수를 정의하고, 스케일러를 이용하여 데이터의 값을 조정하여라. 훈련데이터와 평가데이터의 스케일링 시 주의사항은 무엇인가?

1. 모델이 데이터의 정보를 효율적으로 파악하고 학습할 수 있도록 모든 공정 변수의 값을 표준화하는 데이터 스케일링을 진행한다. 데이터 스케일링을 진행하기에 앞서, 스케일러 용이성을 위하여 모델의 입력변수와 출력변수를 구분하여 스케일링을 진행한다.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  var\_x = ["NG feed”, "NG fuel”, "Water”, "Air”, "PSA recovery”]  var\_y = ["H2”]  scaler\_x = StandardScaler()  scaler\_y = StandardScaler()  train\_x = scaler\_x.fit\_transform(data\_train[var\_x])  train\_y = scaler\_y.fit\_transform(data\_train[var\_y])  test\_x = scaler\_x.transform(data\_test[var\_x])  test\_y = scaler\_y.transform(data\_test[var\_y]) |

모델의 입력변수는 시스템에 주입되는 천연가스의 원료와 연료, 공기와 물의 유량, PSA 장치에서의 회수율로 설정되었고, 출력변수는 시스템에서 빠져나가는 합성가스의 유량과 그 조성, 그리고 각 장치의 온도로 설정되었다. 스케일러는 표준화를 위한 StandardScaler가 사용되었으며, 모델 훈련 시 테스트 데이터셋의 정보가 반영되지 않도록 훈련 데이터셋에 맞추어 스케일링을 진행한다.

#### 공정 모델링을 위한 인공신경망을 구축하여라. 인공신경망 구성 시 필요한 요소들은 무엇인가?

1. 정제된 데이터를 이용하여 SMR 공정을 모델링하기 위해서는 데이터 학습을 위한 인공신경망 구축이 필요하다. 인공신경망 함수는 다음과 같이 정의된다.

|  |
| --- |
| from tensorflow import keras  from tensorflow.keras.layers import \*  def NeuralNet(num\_x, num\_y, num\_layers, num\_neurons, learning\_rate):  model = keras.Sequential()  model.add(Dense(num\_neurons,  activation="relu",  input\_shape=[num\_x]))  for n in range(num\_layers-1):  model.add(Dense(num\_neurons,  activation="relu"))  model.add(Dense(num\_y))  optimizer = keras.optimizers.Adam(learning\_rate = learning\_rate,  beta\_1=0.9, beta\_2=0.999)  model.compile(loss='mse', optimizer = optimizer)  return model |

인공신경망 함수는 입력변수 개수, 출력변수 개수, 은닉층 개수, 은닉뉴런 개수, 학습률을 인자로 갖는다. 은닉층과 은닉뉴런의 개수는 증가할수록 모델 파라미터의 수가 증가하여 더 섬세한 모델링이 가능하지만 오버피팅(overfitting) 문제가 발생할 수 있다. 따라서 문제에 따라 적절한 개수로 설정해주어야 하며, 일반적으로 하이퍼파라미터 최적화 과정을 통해 결정된다.

다음으로 인공신경망 함수를 이용하여 모델을 만든다.

|  |
| --- |
| num\_x = len(var\_x)  num\_y = len(var\_y)  num\_layers = 3  num\_neurons = 20  learning\_rate = 0.001  nn\_model = NeuralNet(num\_x, num\_y, num\_layers, num\_neurons, learning\_rate) |

신경망의 구조는 3개의 은닉층, 각 층마다 20개의 뉴런, 0.001의 학습률로 설정하였다. 그리고 학습 종료조건으로 조기종료(early stopping) 기법을 사용하였다. 이제 모델을 훈련해보자. 모델 훈련 시에는 다양한 훈련조건이 설정된다.

|  |
| --- |
| training\_epoch = 10000  patience = 30  early\_stopping\_cb = keras.callbacks.EarlyStopping(patience=30,  restore\_best\_weights= True,  monitor='val\_loss')  # 모델 저장 경로  import os  def CreateFolder(directory):  try:  if not os.path.exists(directory):  os.makedirs(directory)  except OSError:  print ('Error: Creating directory. ' + directory)  path\_save = f'{Path\_project}/Model'  model\_name = 'nn\_model'  CreateFolder(path\_save)  # 모델 학습  if not os.path.exists(f"{path\_save}/{model\_name}"):  history = nn\_model.fit(train\_x, train\_y,  epochs = training\_epoch, callbacks=[early\_stopping\_cb],  verbose = 2,  validation\_data=(test\_x, test\_y))  # 모델 저장  nn\_model.save(f'{path\_save}/{model\_name}')  # 모델 불러오기  else:  nn\_model = keras.models.load\_model(f"{path\_save}/{model\_name}") |

텍스트이(가) 표시된 사진

자동 생성된 설명

조기종료는 정해진 횟수(tolerance)동안 검증 데이터에 대한 손실 함수값이 더 이상 줄어들지 않으면 학습을 종료한다. 따라서 정해진 훈련 에포크(training epoch)만큼의 반복이 끝나기 전에 훈련이 조기종료될 수 있으며, 이 조건은 모델이 훈련데이터에 과적합(overfitting)되는 문제를 방지할 수 있다.

#### 학습된 모델의 정확도를 평가하여라. 평가지표 R2와 RMSE의 차이점에 대해서 설명하여라.

1. 훈련이 끝난 모델은 R2와 RMSE 등의 평가지표를 통해 성능을 평가한다. R2과 RMSE 함수는 다음과 같이 설정된다.

|  |
| --- |
| import numpy as np  def R\_Squared(prediction, actual\_value):  actual\_mean = np.mean(actual\_value, axis=0)  SSR = np.sum( (prediction - actual\_value)\*\*2 , axis=0)  RSS = np.sum( (prediction - actual\_mean )\*\*2, axis=0 )  TSS = np.sum( (actual\_value - actual\_mean )\*\*2, axis=0 )  r2 = 1 - SSR/TSS  return r2  def RMSE(prediction, actual\_value):  rmse = np.sqrt(np.mean((prediction - actual\_value)\*\*2, axis=0))  return rmse |

예측(prediction)은 훈련된 모델에 입력 변수를 입력하였을 때 출력되는 예측값을 나타내고, 실제값(actual value)은 입력 변수 값에 해당하는 실제 출력 데이터값을 나타낸다. R2 지표는 0과 1사이의 값을 가지며, 1에 가까울수록 예측과 실제가 유사하다는 것을 의미한다. 반면에 RMSE 지표는 예측과 실제의 오차를 나타내는 값으로, 0에 가까울수록 두 값이 일치한다는 것을 의미한다.

|  |
| --- |
| prediction = pd.DataFrame(scaler\_y.inverse\_transform(nn\_model.predict(test\_x)),  index=data\_test.index,  columns=var\_y)  actual\_value = data\_test[var\_y]  r2 = R\_Squared(prediction, actual\_value)  rmse = RMSE(prediction, actual\_value)  display(pd.DataFrame([r2, rmse], index=["R2", "RMSE"])) |

텍스트이(가) 표시된 사진

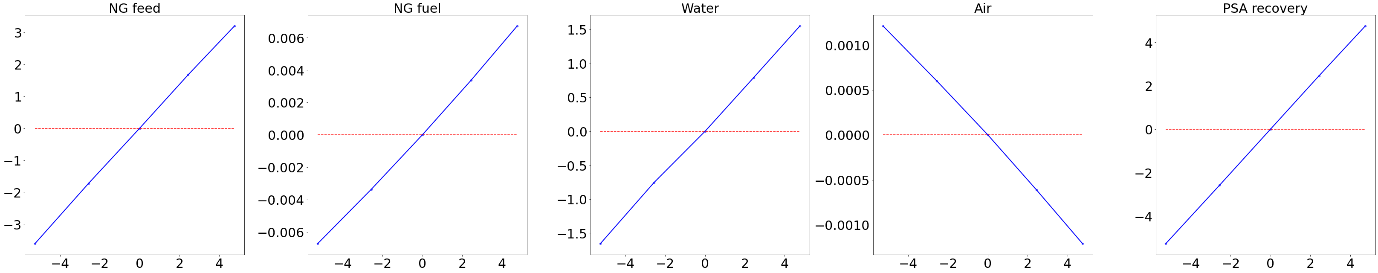
자동 생성된 설명

이 결과는 SMR 공정의 수소생산량에 대한 모델의 예측 성능을 나타낸다.

#### 민감도 분석을 이용하여 공정 모델에 대한 각 입력변수의 영향도를 분석하여라.

1. 민감도 분석은 각 입력변수를 평균값의 -5%~+5% 범위에서 변화시키면서 공정의 반응을 관찰한다. 이 분석을 통해 각 변수들의 영향도를 알 수 있다.

|  |
| --- |
| x\_base = data\_smr[var\_x].mean()  sensitivity\_points = 5  sensitivity\_result = SensitivityAnalysis(x\_base, sensitivity\_points,  nn\_model, var\_x, var\_y, scaler\_x, scaler\_y) |

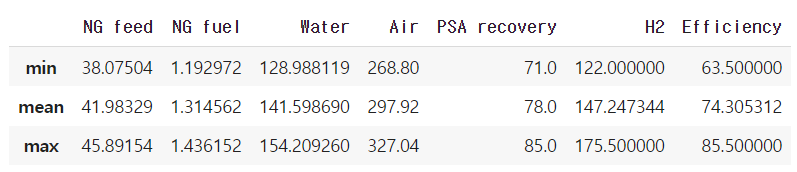
수소생산량에 대해 3 개의 변수(천연가스 원료, 물, PSA 장치의 회수율)가 유의미한 영향도를 나타내고, 2 개의 변수(천연가스 연료, 공기)가 미미한 영향도를 나타냈다. 또한, PSA 장치의 회수율, 천연가스 원료, 물, 천연가스 연료, 공기의 순서대로 수소 생산량에 큰 영향력을 갖는다.

### **[응용] 인공신경망 기반 최적 공정변수 탐색**

#### 학습된 모델을 기반으로 격자탐색법을 이용하여 SMR 공정의 운전 조건을 최적화하여라. 최적화하고자 하는 목적함수에 따라서 최적 운전 조건이 어떻게 달라지는가?

1. 앞에서 구축된 인공신경망 모델의 빠른 계산속도를 활용하여 최적의 운전조건을 탐색한다. 운전조건은 수집된 데이터에서 각 변수의 최솟값과 최댓값 사이에서 탐색하며, 이 범위를 탐색공간이라고 정의한다.

|  |
| --- |
| data = data\_smr  data.describe().loc[['min', 'mean', 'max']] |



최적 조건을 찾기 위한 알고리즘으로 격자탐색법(grid search)을 사용하였다. 격자탐색법은 탐색공간 내에서 격자점을 생성하여 모든 격자점에서의 목적함수 값을 확인하며 최적 해를 탐색하는 방법으로, 간단하고 빠르지만 탐색공간이 커지거나 결정 변수의 개수가 많아질수록 계산비용이 크게 증가하는 특징이 있다.

|  |
| --- |
| bins = 11  gridsearch\_result = GridSearch(data, bins, nn\_model, var\_x, var\_y, scaler\_x, scaler\_y) |



5개 입력변수마다 11개의 격자점을 조합하여 총 161,051개의 운전조건을 생성하여 최적의 조건을 탐색하였다. 생성된 운전조건들에 대해서 수소 생산량을 계산하고, 내림차순으로 정렬함으로써 수소 생산량이 최대가 되는 운전조건을 찾을 수 있다.

|  |
| --- |
| objective\_y = "H2"  Optimization(gridsearch\_result, var\_x, objective\_y) |

테이블이(가) 표시된 사진

자동 생성된 설명

수소 생산량이 최대가 되는 최적 운전조건을 탐색한 결과, 모든 입력변수의 값이 최대일 때 수소가 가장 많이 생산되었다. 한편, 수소 생산량을 바탕으로 SMR 공정의 열효율을 계산할 수 있다. 공정의 열효율은 다음과 같이 정의된다.

|  |  |
| --- | --- |
|  | (4) |

투입되는 원료 및 연료의 양을 고려했을 때, 수소를 효율적으로 생산하기 위해서는 공정 열효율을 높이는 것이 중요하다.

|  |
| --- |
| objective\_y = "Efficiency"  Optimization(gridsearch\_result, var\_x, objective\_y) |

테이블이(가) 표시된 사진

자동 생성된 설명

공정 열효율이 최대가 되는 운전조건을 탐색한 결과, 천연가스의 원료 및 연료, 공기의 유량은 줄고 물과 PSA 장치의 회수율은 증가하였다. 이와 같이 격자탐색법을 이용하여 격자점을 한번 생성하고 나면, 원하는 목적함수에 맞는 최적의 운전조건을 빠르게 탐색할 수 있다.

### **[결론]**

본 장에서는 수소 생산을 위한 수증기 개질 공정을 인공신경망 기법을 이용하여 모델링하고, 최적의 운전조건을 탐색하였다. 문제 해결 과정에서 개발된 모델의 신뢰성 및 타당성을 검증하고, 민감도 분석을 통해 예측 변수에 가장 영향력 있는 입력 변수를 분석하였다. 그 다음, 인공신경망을 기반으로 격자탐색법을 이용하여 탐색공간 내에서 SMR 공정의 운전조건을 최적화하였다. 수소 생산량을 최대화하는 경우와 공정 열효율을 최대화하는 경우의 최적 운전조건이 다르게 나타났으며, 목적에 따라 공정의 운전조건을 적절하게 바꿔주어야 한다는 것을 이해하였다. 이와 같이, 인공신경망 모델과 격자탐색법을 공정 최적화에 적용하면 다양한 목적함수에 대해 신속하고 정확하게 최적의 해답을 얻을 수 있다.

### **[학습 결과]**

* 학습 내용

공정 운전 조건 최적화를 위한 신경망 모델 구축 및 최적화 적용 방법 익히기

* 학습 결과 확인하기

민감도 분석을 통한 주요 변수 확인하기 및 목적함수에 따른 최적화 결과 분석하기

* 학습 결과 응용하기

수학적으로 표현하기 어려운 시스템을 대리 모델 구축과 기울기를 사용하지 않는 최적화를 이용하여 효율적으로 최적화하기

### **3.2. 친환경적 폭발성 폐기물 처리 공정 운전 최적화**

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | 폭발성 폐기물 처리 공정의 이해 |
| [방법] | 폭발성 폐기물 처리 공정의 대리 모델(Surrogate model) 구축 |
| [응용] | 대리 모델(Surrogate model)의 최적 운전 조건 도출 |
| [요약] | * 폭발성 폐기물 공정에 대한 이해 * 대상 시스템의 입력과 응답의 관계를 인공신경망으로 모사 * 기울기 정보를 사용하지 않는 최적화를 수행하여 최적 조건 도출 |

### **[공정 설명] 폭발성 폐기물 처리 공정의 이해**

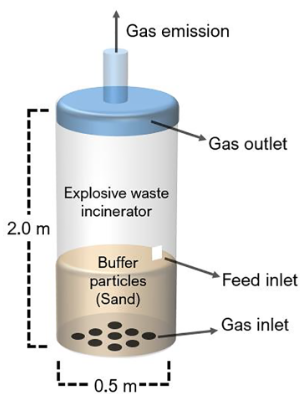


그림 1. 폭발성 폐기물 처리를 위한 유동층 반응기 구조

화학 반응을 이용한 군용 무기 제작 시 2-methyl-1,3,5-trinitrobenzene(TNT)와 같은 폭발성 물질들이 다량 사용되며 그 결과 많은 양의 폭발성 폐기물이 배출된다. 이러한 폐기물은 영구적으로 보관될 수 없으며, 안전을 고려하여 조속히 폐기되어야 한다. 일반적으로, 폭발성 폐기물 처리를 위해 open detonation 방식과 open burning 방식을 사용하지만 이들은 많은 양의 폐기물을 처리할 수 없을 뿐만 아니라 다량의 질소산화물(Nitrogen oxides; NOx) 발생을 동반하여 환경적으로도 한계를 지닌다.

이러한 한계를 극복하고자 고온가스를 이용한 로터리 킬른(Rotary kiln) 반응기 기반의 소각 공정에 대한 논의가 이루어져 왔다. 소각 공정은 많은 양의 폐기물을 연속적으로 처리할 수 있고 기존 방법에 비하여 NOx 배출량을 90% 이상 줄일 수 있다는 장점이 있다. 하지만, 로터리 킬른 반응기는 반응기 내부 온도 조절이 불가능하며, 이로 인해 반응기 내부에서 온도가 급격히 증가하는 핫스팟(hot spot)과 그로 인한 NOx 배출량 증가 시 대처가 어렵다는 단점을 지닌다. 유동층 반응기(Fluidized bed reactor)는 로터리 킬른 반응기에 비하여 반응기 내부 온도 조절이 용이하기 때문에 핫스팟 발생을 억제할 수 있고, 결과적으로 NOx 배출량도 대폭 감소시킬 수 있다.

폭발성 폐기물을 다루는 공정의 특성상 실제 실험 전에 반응기 내부 유동의 정확한 모사가 필수적이다. Computational fluid dynamics(CFD)는 식 (1), (2)의 Navier-Stokes 방정식 등을 기반으로 유체의 복잡한 전달 현상(transport phenomena)을 정확하게 모사할 수 있는 전산모사 방법이다.

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |

CFD는 높은 정확도를 보이지만 많은 계산을 필요로 한다. CFD를 이용하여 최적화를 수행하는 경우, 계산량 감축을 위해 일반적으로 각 변수를 특정 범위 내에서 임의의 간격으로 분할하여 부분집합을 생성함으로써 수행한다. 하지만 이러한 접근법은 국부해(local optimum)로 수렴되어 전역해(global optimum)를 구하지 못할 가능성이 높다. 계산량 감축과 전역해 도달을 위한 한 가지 대안으로, 실제 함수(underlying function)를 모사하는 대리 모델(surrogate model)을 생성하고, 이를 통해 최적화를 수행하는 방법이 있다. 대리 모델은 많은 반복 계산을 필요로 하는 최적화 과정에서, 계산량을 대폭 감축시킴으로써 빠른 연산을 가능케 한다.

최적화는 크게 기울기 기반 방법(derivative-based optimization)과 기울기를 사용하지 않는 방법(derivative-free optimization)이 있다. 기울기 기반 방법은 수렴 속도가 빠른 반면에 국부해로 수렴될 가능성이 높으며, 이로 인해 많은 경우 유전 알고리즘(genetic algorithm) 등의 기울기를 사용하지 않는 방법이 선호된다.

본 챕터에서는 CFD 시뮬레이션 데이터를 통해 인공신경망 대리 모델을 구축하고 기울기를 사용하지 않는 최적화 방식을 사용하여 효율적인 최적화를 수행하는 것을 목적으로 한다.

### **[문제]**

**온도, 압력, 가스 유입 속도 등 반응기 운전 조건과 유입되는 입자의 조성, 입자의 크기 등 원료 조건이 질소산화물 배출에 미치는 영향을 나타내는 인공신경망 모델을 구축하고 메타휴리스틱(Metaheuristics) 최적화 방법을 적용하여 질소산화물 배출량을 최소로 하는 운전조건을 규명하자.**

* **MATLAB의 Statistic and Machine Learning Toolbox를 이용하자**
* **표준화된 데이터 ‘RawX.xlsx’과 ‘RawY.xlsx’를 이용하자**

**\* 데이터는 보안 문제로 인하여 표준화된 형태로 제공됨**

### **[방법] 폭발성 폐기물 처리 공정의 인공신경망 모델 구축**

#### 관련 라이브러리를 설치하고 데이터를 작업 환경으로 입력하시오.

1. MATLAB을 사용하는 경우에 Statistics and Machine Learning Toolbox를 설치해야 한다. 해당 Toolbox를 사용하여 간편하게 인공신경망을 구축하고 최적화를 수행할 수 있다.

xlsread 를 통해 해당 경로에 있는 엑셀 파일을 불러올 수 있다.

|  |
| --- |
| RawX = xlsread('RawX.xlsx');  Response = xlsread('RawY.xlsx'); |

RawX는 xlsread 를 통해 해당 경로에 있는 엑셀 파일을 불러올 수 있으며, 아래 그림과 같이 Workspace에 두 변수 double 형태로 만들어진다.



#### 제공된 데이터를 표준화하시오.

1. 질소산화물 배출에 영향을 미치는 다섯 파라미터를 표준화하여 모델의 정확도를 향상시킬 수 있다. RawX(:, 1)은 배열 RawX을 인덱싱하는 것이다. x1에 RawX 중 1번째 열(column)에 있는 모든 값을 할당한다.

모든 파라미터를 0부터 1 사이의 값으로 변환시키기 위하여 각 파라미터의 최댓값과 최솟값을 정의하면 Std\_x1과 같이 표준화된 파라미터가 생성된다.

\*데이터 보안 상 실제 수치는 다를 수 있다.

표준화를 마친 다섯 변수 Std\_x1, Std\_x2, … , Std\_x5를 학습에 사용하기 위하여 다시 결합하여 StdX 라는 배열을 생성한다.

|  |
| --- |
| x1 = RawX(:, 1); %배열 인덱싱  x2 = RawX(:, 2);  x3 = RawX(:, 3);  x4 = RawX(:, 4);  x5 = RawX(:, 5);  min\_x1 = k1; % 각 파라미터 최댓값 및 최솟값 설정  max\_x2 = k2;  Std\_x1 = (x1-min\_x1)/(max\_x1-min\_x1); %표준화  Std\_x2 = (x2-min\_x2)/(max\_x2-min\_x2);  Std\_x3 = (x3-min\_x3)/(max\_x3-min\_x3);  Std\_x4 = (x4-min\_x4)/(max\_x4-min\_x4);  Std\_x5 = (x5-min\_x5)/(max\_x5-min\_x5);  StdX = [Std\_x1, Std\_x2, Std\_x3, Std\_x4, Std\_x5]; %데이터 결합 |

테이블이(가) 표시된 사진

자동 생성된 설명

#### 인공신경망 구조를 설정하시오.

1. 인공신경망에 사용하는 데이터는 열(column)을 기준으로 한다. 따라서 표준화된 데이터를 전치하여 데이터 형태를 맞춘다. Training function은 경사하강법 등 여러 가지를 사용할 수 있지만 일반적으로 다른 방법에 비하여 비교적 빠르게 수렴하는 Levenberg-Marquardt 방법을 사용한다. 은닉층의 노드(node) 개수는 hiddenLayerSize로 정의하는데 은닉층(hidden layer)이 한 층인 경우는 아래와 같이 선언하며 여러 층인 경우는 [10, 5, 4]와 같이 행벡터(Row vector)로 표현한다. net = fitnet(hiddenLayerSize,trainFcn)은 은닉노드의 크기가 hiddenLayerSize이고 trainFcn을 training function으로 하는 신경망 구조를 생성한다. 전체 데이터 중 학습데이터를 70%, 검증데이터를 15%, 평가데이터를 15%로 설정하였다. 학습을 진행할 최대 epochs를 1500으로 설정하였으며 학습은 최대 100초 동안 진행한다.

\*net에 대한 자세한 설명은 MATLAB Command window에 net을 입력하면 볼 수 있다.

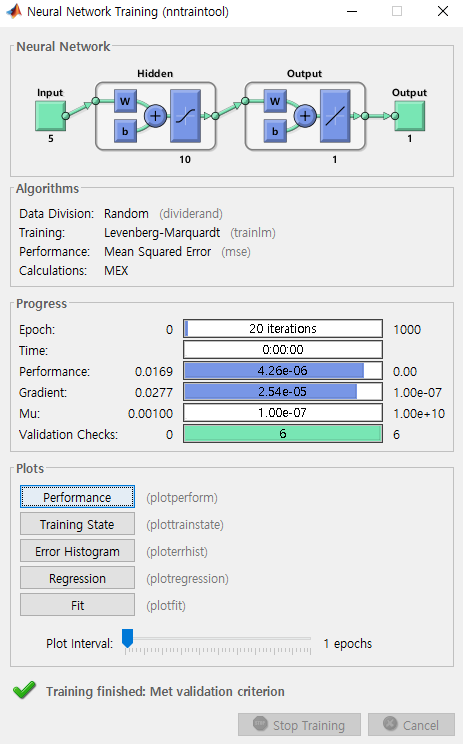
|  |
| --- |
| x = StdX' ; %데이터 구조 변환  t = Response' ; %데이터 구조 변환  trainFcn = 'trainlm'; %Training function 설정  hiddenLayerSize = 10; %은닉층 노드(hidden node) 설정  net = fitnet(hiddenLayerSize,trainFcn); %신경망 생성  net.divideParam.trainRatio = 70/100; %학습데이터 비율 설정  net.divideParam.valRatio = 15/100; %검증데이터 비율 설정  net.divideParam.testRatio = 15/100; %평가데이터 비율 설정  net.trainParam.epochs = 1500; %학습할 최대 Epoch 횟수 설정 (default : 1000)  net.trainParam.time = 100; %학습할 최대 시간 설정(단위 : 초) (default : inf) |

위 단계를 수행하면 workspace에 class가 network인 net이 생성된다.

#### 인공신경망을 학습해보자.

1. Train 커맨드를 사용해서 신경망을 훈련시키며 train(net, x, t)와 같이 사전에 선언한 신경망 구조와 입력과 응답을 입력한다. gsubtract 커맨드를 통해 실제 응답(t)과 신경망 응답(y) 의 차이를 구하고, perform 커맨드를 통해 신경망 성능을 정량화한다. net과 훈련 기록을 반환하며, workspace에 class가 struct인 훈련 기록 tr이 저장된다(그림 2). view 커맨드는 직접 구성한 신경망 구조를 확인할 수 있다(그림 3). Parity plot을 통해서 인공신경망의 성능과 오버피팅(overfitting) 여부를 확인한다(그림 4).

|  |
| --- |
| [net,tr] = train(net,x,t); %신경망 훈련 및 net, tr(훈련 기록) 반환  y = net(x);  e = gsubtract(t,y); %두 배열 또는 셀의 성분의 차를 구함  performance = perform(net,t,y); %신경망 성능을 계산함  view(net) %신경망 구조 반환 |

테이블이(가) 표시된 사진

자동 생성된 설명

그림 2. 훈련 결과(좌)와 훈련 기록(우)

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

그림 3. 신경망 구조

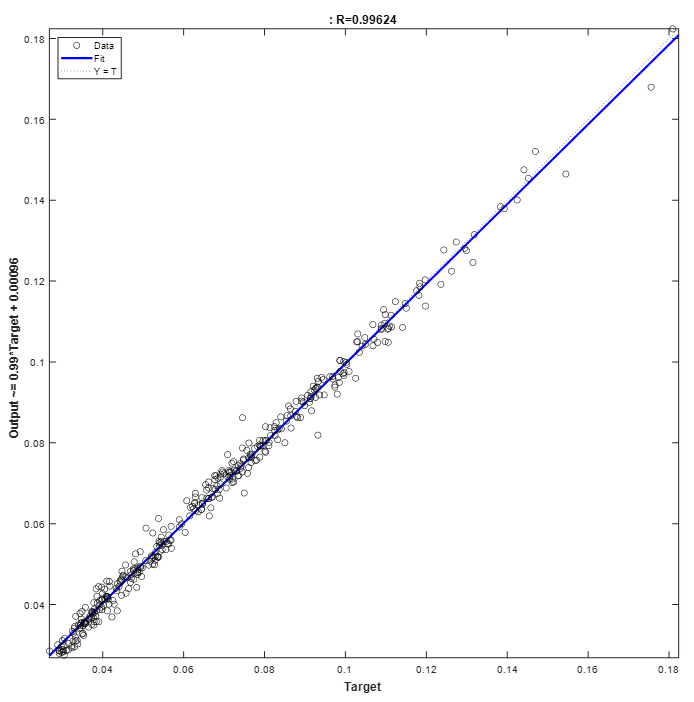


그림 4. Parity plot을 통한 모델 결과 확인

#### 인공신경망을 최적화에 사용하기 위하여 함수로 불러오자.

1. genFunction을 통해서 학습한 신경망인 net을 함수 형태로 저장할 수 있다. NNfunction.m 이라는 파일이 MATALB 경로에 생성된다. MATLAB은 이처럼 m파일(‘-.m’형태의 파일)을 함수처럼 사용할 수 있는 기능을 제공한다. fun = @(x) NNfunction(x)는 NNfunction이라는 함수를 x에 대한 함수 fun으로 선언하는 것을 의미한다. 이때, @(x) 는 NNfunction이 x, y, z에 대한 함수인 경우에도 x만을 변수로 한다. Function handle 성능 확인을 위해 x의 첫 번째 열을 인덱싱한 후 fun의 응답을 확인하고 이를 y와 비교한다.

인공신경망 함수를 최적화의 목적 함수로 사용하기 위하여 형태를 변형시켜야 한다. 먼저, genFunction은 함수를 셀(cell)형으로 반환하기 때문에 계산된 값이 스칼라가 아닌 셀형태로 반환된다. 스칼라로 목적 함수를 반환받기 위하여 ‘MatrixOnly’를 ‘Yes’로 선언한다. 다음으로, 데이터 입력 형태에 관한 변환이다. 현재 열을 기준으로 학습되어 있는 인공신경망을 최적화에 사용하기 위하여 행으로 변환해야한다. 따라서 NNfunction의 인수를 전치하여 입력한다.

|  |
| --- |
| genFunction(net,'NNfunction', ‘MatrixOnly’, ‘yes’); %학습된 신경망 코드 생성  fun = @(x) NNfunction(x’); %신경망 구조를 갖는 함수 생성(function handle)  [Command Window]  >> y1 = fun(x(:, 1)) %function handle 결과 확인  ans = 0.0495 |

### **[응용] 메타휴리스틱 방법을 통한 최적 조건 규명**

대상 함수의 기울기 정보를 사용하는 최적화는 기울기 정보가 정확하고 얻기 쉬운 경우에는 최적해를 빠르게 얻을 수 있다. 하지만, 실제로 기울기에 대한 정보를 얻을 수 있는 경우는 제한적이며 이에 대한 신뢰도를 보장할 수 없다. 이러한 한계는 기울기 정보를 사용하지 않는 derivative-free optimization(DFO)를 통해 극복할 수 있다. 유전알고리즘(Genetic algorithm)은 메타휴리스틱 최적화 방법 중 가장 널리 사용되는 방법으로 변이(mutation), 교배(crossover) 등을 기반으로 최적화를 진행한다.

|  |
| --- |
| Numvar = 5; %입력 파라미터 개수  lowerbound = [0 0 0 0 0]; %각 변수의 최솟값  Upperbound = [1 1 1 1 1]; %각 변수의 최댓값  opts = optimoptions(@ga, 'PlotFcn',{@gaplotbestf}); %최적화 옵션  [x,fval,exitFlag,Output] = ga(fun,Numvar,[],[],[],[],lowerbound,Upperbound,[], opts); |

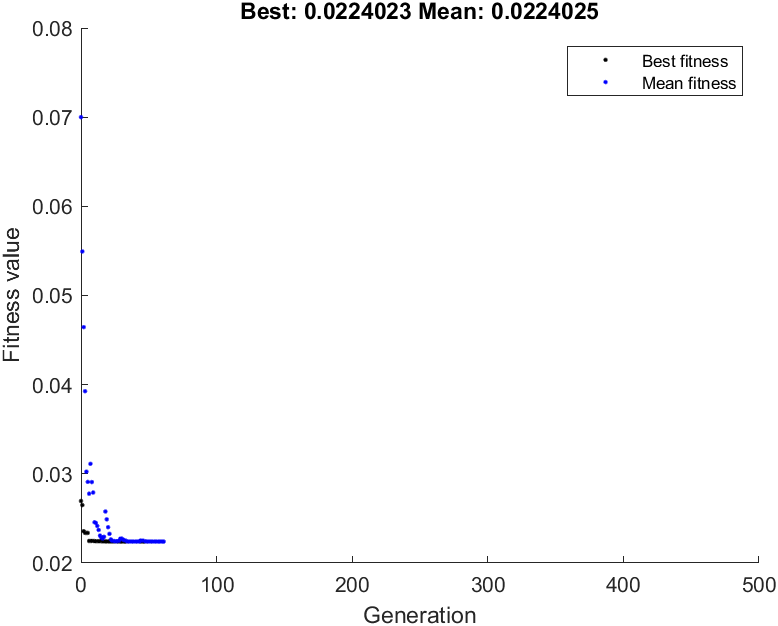


그림 5. 유전알고리즘 수행 과정

최적화 옵션을 통해 다양한 파라미터를 설정할 수 있으며 그림 5와 같이 최적화가 수행되는 과정을 도식화 할 수 있다**.** 최적화 결과 질소산화물의 최솟값은 0.0224로 구해지며 그때의 운전조건은 다음과 같다.

하지만, 이처럼 유전 알고리즘과 같은 DFO 알고리즘을 통해 얻은 해를 전역 최적해(global optimal)이라고 보장할 수 없으며 다양한 알고리즘을 적용하여 최대 혹은 최솟값을 도출하는 알고리즘의 해를 선택한다.

### **[결론]**

공학 분야에서 대상으로 하는 대부분의 공정은 입∙출력 변수들 간의 관계가 매우 복잡하여, 시스템의 응답을 정확하게 예측하고 최적화하는데 어려움이 있다. 본 챕터에서는 폭발성 폐기물을 처리하는 공정에서 발생하는 질소산화물 저감을 목적으로, 인공신경망 기반의 대리 모델을 구축하고, 기울기 정보를 사용하지 않는 최적화기법을 통해 최적화함으로써, 최적의 운전 조건 및 원료 조건을 도출하였다. 모든 변수에 대하여 최적 운전조건이 결정론적으로 도출되었으며 해당 운전조건에서 0.0224 만큼의 질소산화물이 배출되는 것을 확인하였다.

### **[학습 결과]**

* 학습 내용

공정 운전 조건 최적화를 위한 신경망 대리 모델 구축 및 최적화 적용 방법 익히기

* 학습 결과 확인하기

신경망을 통한 시스템 모사 방법 및 파라미터 변경 방법, 최적화 진행 과정 분석하기

* 학습 결과 응용하기

수학적으로 표현하기 어려운 시스템을 대리 모델 구축과 기울기를 사용하지 않는 최적화를 이용하여 효율적으로 최적화하기

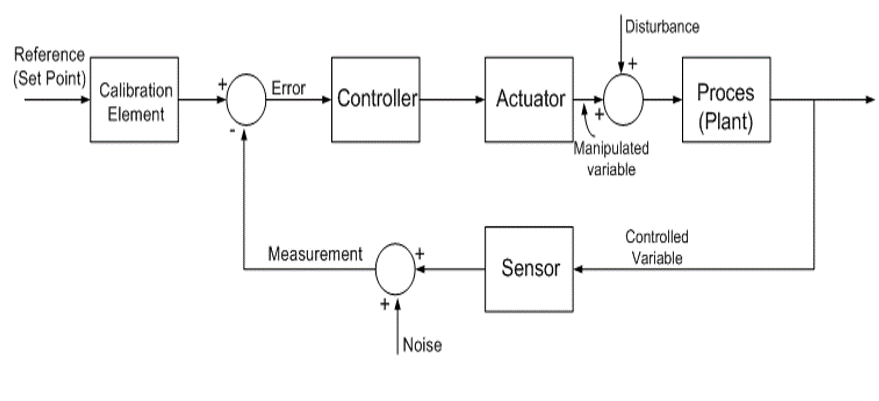
### **4.1. PID(Proportional Integral Derivative) 공정 제어**

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | PID 제어 시스템의 이해 |
| [방법] | PID 제어 시스템의 조율 |
| [응용] | 딥러닝을 적용한 PID 제어 시스템의 조율 |
| [요약] | * PID 제어 시스템의 이해 * 기존 PID 제어 시스템의 조율 방법 * 딥러닝을 이용한 PID 제어 시스템의 조율 |

### **[이론] PID 제어 시스템**

### **공정제어의 개요**

**공정제어**(process control)란 온도, 압력, 유량 등 공정의 상태를 측정하여, 해당 상태가 우리가 원하는 조건으로 도달될 수 있도록 공정의 변수를 조작하는 행위를 일컫는다. 제어시스템은 제어기를 중심으로, 제어기가 조절하는 조작 변수(manipulated variable; MV), 제어기가 제어하고자 대상이 되는 제어 변수(controlled variable; CV), 도달하고자 하는 설정 변수(목표값, set variable; SV), 그리고 제어 변수에 영향을 미치나 제어기가 조작할 수 없는 외란 변수(disturbance variable; DV)로 구성된다. 그림 1에 보인 바와 같이, 제어기는 센서를 이용하여 공정의 상태인 제어 변수 값을 측정한 후, 이를 설정 변수와 비교하여, 그 차이가 최소화되도록 조작 변수의 값을 결정한다.



**그림 1. 피드백 제어 시스템의 블록 흐름도 예시**

### **PID 제어의 이해**

PID 제어는 산업계에서 가장 널리 이용되는 제어기법으로, 계산량이 적고 그 적용이 쉽다는 장점이 있다. PID 제어는 P 제어(proportional control; 비례 제어), I 제어(integral control; 적분 제어), 그리고 D 제어(derivative control; 미분 제어)로 구성된다.

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

**그림 2. PID 제어 시스템의 블록 흐름도**

**P 제어**는 현 시점의 제어오차에 비례하는 제어동작을 취한다. P 제어는 계산이 간단하여 응답이 빠르다는 장점이 있으나 잔류오차(offset)가 발생한다는 단점이 있다. 여기에서, 잔류 오차란 시간의 흐름에 따라 설정 변수와 제어 변수의 차이가 감소하지 않고 유지된 채 지속되는 오차를 가리킨다.

**I 제어**는 시간의 흐름에 따라 계속해서 제어오차를 누적하고, 누적된 제어오차에 비례하여 제어동작을 산출한다. 이러한 제어오차의 누적과 이를 통한 제어동작은, 제어오차가 0이 될 때까지 지속되므로, P 제어의 단점인 잔류 오차 문제를 해결할 수 있다. 다만, I 제어의 경우, 충분한 제어오차가 누적되어야 큰 제어동작을 산출하므로, 제어 응답이 느리다는 단점을 지닌다.

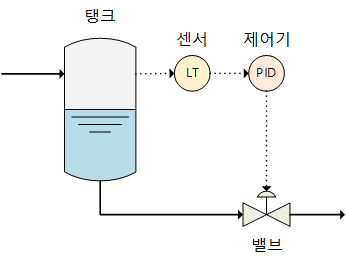
**D 제어**는 제어오차의 기울기에 비례하는 제어동작을 산출한다. D 제어는 P 제어와 마찬가지로 잔류오차를 발생시키며, 미분(즉, 제어오차의 기울기)에 기반하여 제어 동작을 산출하므로, 높은 주파수를 갖는 노이즈에 민감하다는 단점을 지닌다.일반적으로, D 제어는 그 특성상, P 제어나 I 제어와 달리 단독으로 사용하지 않는다.

### **인공지능 기반의 PID 제어**

PID 제어는 비례 이득, 적분 시간, 미분 시간으로 구성된 조율 변수에 따라 그 성능이 결정된다. 오랜 기간 동안 이러한 PID 제어의 최적 조율 변수 값을 결정하기 위한 많은 방법들이 제안되었으나, 그 적용 과정이 복잡하고 많은 시행착오 과정을 동반하여 널리 적용되지 않고 있다. 본 장에서는 인공지능 학습을 통해 PID 제어 시스템을 조율하고자 한다.

### **[문제]**

**수조에 찬 물의 수위를 PID 제어 시스템을 통해 제어하고자 한다. 관련 데이터는 ‘matplotlib.pyplot’를 불러와 이용할 때, PID 제어 시스템에 딥러닝을 적용해 PID 제어 시스템의 KP, KI, KD를 조율하라. 반복 계산 횟수는 100,000번으로 하고, 절댓값 오차를 사용한다. 초기의 조건은 KP=1.0, KI=0.5, KD=0.5이고, 오차축적 횟수는 100, 학습률(learning rate)은 0.001, 초기 오차는 10이다.**



### **[방법] PID 제어 시스템의 조율**

#### PID 제어 시스템의 조율 방법 중 가장 대표적인 지글러-니콜스 조율(Ziegler-Nichols tuning) 방법에 대해 설명하여라.

1. 지글러-니콜스 조율은 1942년 Ziegler와 Nichols에 의해 제안된 방법 중 하나로, 경계안정상태의 특성에 근거한 제어기 조율 방법 중 하나이다. 지글러-니콜스 조율은 먼저 I 제어와 D 제어를 끄고 P 제어기만 켠 상태 또는 KI에 무한에 가까운 수를 넣고, KD에는 0을 입력한 상태에서 진행한다. 이 상태에서 그림 3과 같이 공정 출력에 지속적인 진동 현상이 나타날 때까지 KC를 작은 값에서부터 서서히 증가시킨다.

텍스트, 하늘, 다른이(가) 표시된 사진

자동 생성된 설명

**그림 3. 공정 출력의 지속적인 진동 현상**

KC가 작을 때에는 목표값에 수렴하다가 KC가 점점 커지면서 오차에 민감해지기 때문에 그림 3과 같은 규칙적인 진동이 관측되기 시작한다. 만약 이보다 KC가 더 커지면 공정 출력은 발산한다. 규칙적인 진동의 주기를 Pu(ultimate period)라고 하며, 이때의 KC는 KCu(ultimate controller gain)이라고 한다. 마지막으로, KCu을 표 1에 제시된 표에 대입하여 제어 시스템을 조율한다.

테이블이(가) 표시된 사진

자동 생성된 설명

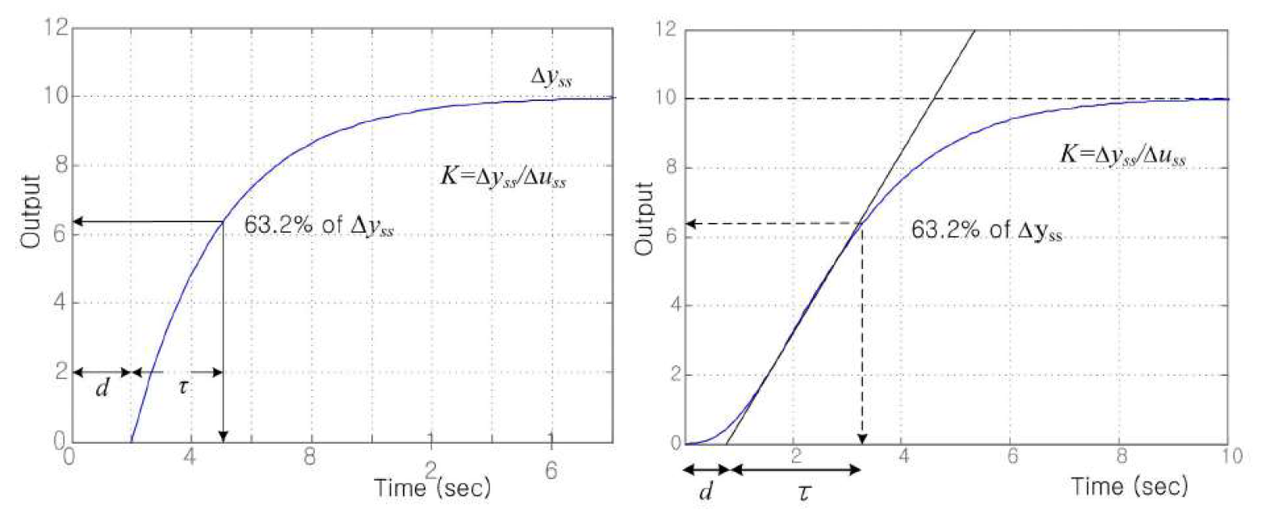
**표 1. 제어 시스템의 지글러-니콜스 조율 값**

#### 또 다른 PID 제어 시스템 조율 방법인 1차 시간지연(first-order plus dead time) 모델의 전제 조건은 무엇이며, 조율하는 방법을 설명하여라.

1. 1차 시간지연 모델은 식 (4)와 같이 나타낼 수 있다. 식 (4)는 시간 t를 라플라스 변환하여 나타낸 식이다.

식 (4)

식 (4)에서 d와 τ는 그림 4를 통해 알 수 있다. d는 지연 시간(dead time)으로, 제어기를 통해 제어출력이 나올 때까지의 시간을 의미한다. 실제 공정에서는 대부분 2차 함수와 같은 모습이 나타나는데, 이 경우 변곡점에서 접선을 그린 후 0 초부터x축과 만나는 점까지의 시간을 지연 시간으로 정의한다. τ는 시정수를 의미하며 제어 출력이 정상 상태의 제어 출력의 63.2%인 지점의 시간을 말한다.

**그림 4. (a) 1차 함수와 (b) 2차 함수의 반응응답곡선**

1차 시간지연모델의 전제 조건은 ‘0.1<d/τ<1.0’이다. d/τ<0.1일 때에는 지연 시간이 무시되는 1차 공정에 근접한다. 1차 공정의 경우 KC를 크게 할수록 제어가 잘된다. 반면, d/τ>1.0에서는 지연시간이 매우 커 일반적인 PID 제어 시스템으로는 제어성능에 한계가 생기므로 dead time compensator(ex. Smith predictor)가 필요하다.

|  |  |  |  |
| --- | --- | --- | --- |
| 제어기 | KC | KI | KD |
| P | (τ/KPd) | - | - |
| PI | 0.9(τ/KPd) | 3.33d | - |
| PID | 1.2(τ/KPd) | 2.0d | 0.5d |

**표 2. 제어 시스템의 1차 시간지연 모델 조율 값**

마지막으로, 1차 시간지연 모델로부터 지글러-니콜스 조율에 필요한 Kcu와 Pu를 구하여 지글러-니콜스 조율을 적용한다. Kcu와 Pu를 쉽게 계산하기 위해 표2와 같이 KP, d, τ값으로도 나타낼 수 있다.

### **[응용] 딥러닝을 통한 이산화탄소 배출량 예측**

#### ‘matplotlib.pyplot’을 불러오고, 탱크의 수위를 PID제어기를 정의하여라.

1. 먼저, 그림 5와 같이 윈도우 명령 프롬프트에 pip을 활용해 matplotlib을 컴퓨터에 설치한다. 그리고 아래의 절차에 따라 탱크의 수위와 PID 제어 시스템을를 정의한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| import matplotlib.pyplot as plt   #데이터베이스 불러오기 |

**그림 5.** matplotlib.pyplot **설치하기**

명령 프롬프트에 ‘python -m pip install -U matplotlib’를 입력하여 matplotlib.pyplot를 설치한다.

먼저, 파이썬에 설치한 matplotlib.pyplot를 불러온다.

|  |
| --- |
| class Liquid:      def \_\_init\_\_(self, error):          self.current\_error = error          self.last\_action = 0      def take\_action(self, action):          self.current\_error += 0.1 \* action          self.last\_action = action  class PID:      err\_sum = 0      old\_err = 0      def pid(self, current, goal, kp, ki, kd):          err = goal - current          self.err\_sum += err          delta\_err = err - self.old\_err          self.old\_err = err          return kp\*err + ki\*self.err\_sum + kd\*delta\_err |

위의 코드와 같이 탱크의 수위는 목표값과 실제 탱크의 수위만큼의 차인 error를 가져와 제어동작을 취함으로써 수위를 조절한다. 이때, ‘current’는 현재 탱크의 수위, ‘goal’은 제어기의 목표값을 나타내며, kp, ki, kd는 각 제어기의 조율 인자(KP, KI, KD)를 의미한다.

#### PID 제어 시스템을 조율하기 위한 함수를 정의해라

1. PID 제어 시스템 조율에 필요한 함수는 아래의 절차대로 정의한다.

|  |
| --- |
| class Derivative:      def \_\_init\_\_(self):          self.last\_x = 0          self.last\_y = 0      def get\_gradient(self, x, y):          d = (y - self.last\_y) / (x - self.last\_x)          self.last\_x = x          self.last\_y = y          return d |

위의 코드와 같이 미분 클래스를 정의한다. 미분 클래스는 기울기를 통해 조율 인자를 최적화할 때 사용된다.

|  |
| --- |
| class Train:      kp = 1.0; ki = 0.5; kd = 0.5      goal = 0      episode\_length = 100      learning\_rate = 0.001      def \_\_init\_\_(self):          self.dp = Derivative()          self.di = Derivative()          self.dd = Derivative()          self.step = 0          self.last\_loss = 0 |

위의 코드는 제어 시스템의 조율을 위해 해당 모델을 훈련시키기 위해 훈련 클래스를 정의하는 코드이다. 가장 먼저 문제에 주어진 초기 조건들을 입력하고 변수들을 정의한다. 목표값과 실제 탱크의 수위의 차이는 0이 되는 것이 이상적이므로 goal은 0으로 설정하고, 오차축적 횟수(episode\_length)는 100, 학습률은 0.001로 설정한다.

|  |
| --- |
| def abs\_mean(self, list):          sum = 0          for i in list:              sum += abs(i)          return sum / len(list)      def loss(self):          liquid = Liquid(10)          pid = PID()          error = []          for i in range(self.episode\_length):              error.append(liquid.current\_error)              liquid.take\_action(pid.pid(liquid.current\_error, self.goal, self.kp, self.ki, self.kd))          return self.abs\_mean(error)      def optimize(self):          self.kp = self.kp - self.learning\_rate \* self.dp.get\_gradient(self.kp, self.loss())          self.ki = self.ki - self.learning\_rate \* self.di.get\_gradient(self.ki, self.loss())          self.kd = self.kd - self.learning\_rate \* self.dd.get\_gradient(self.kd, self.loss())          self.last\_loss = self.dd.last\_y          print("step={}, kp={}, ki={}, kd={}, loss={}".format(self.step, self.kp, self.ki, self.kd, self.last\_loss))          self.step += 1 |

위의 코드는 훈련 클래스에서 정의된 함수들이다. ‘abs\_mean’은 문제에서 정의된 것처럼 오차를 구할 때 사용할 절댓값 평균 함수를 정의한다. ‘loss’는 for문을 통한 반복 계산을 통해 탱크의 수위를 보정하며 실제 탱크의 수위와 목표값의 차이(error)를 축적하는 함수이다. ‘loss’에서 구해진 error는 abs\_mean 함수를 통해 절댓값 평균으로 출력된다. 마지막으로, ‘optimize’는 앞서 정의한 함수들을 통해 최종적으로 조율 인자들을 최적화함으로써 PID 제어 시스템을 조율한다. 또한 이때, 뒤쪽의 그림 6과 같이 반복 계산을 하는 각 단계의 값들을 볼 수 있도록 설정했다.

#### 앞서 정의한 함수들을 통해 KP, KI, KD 값을 최적화하고 이를 도식화하라.

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':      loss = []      kp = []      ki = []      kd = []      train = Train()      for j in range(100000):          train.optimize()          loss.append(train.last\_loss)          kp.append(train.kp)          ki.append(train.ki)          kd.append(train.kd) |

1. 아래와 같은 절차를 통해 조율 인자들을 최적화할 수 있다. 또한 그 결과는 그림 6과 7을 통해 각각 확인할 수 있다.

먼저, 위의 코드와 같이 계산된 데이터를 담을 배열을 생성하고, Train의 인스턴스를 저장할 변수(train)을 생성한다. 또한 앞서 정의한 ‘optimize’함수를 사용해 조율 인자들을 최적화한다. 이때 반복 횟수는 문제에서 언급한 것처럼 100000회로 설정한다.

텍스트이(가) 표시된 사진

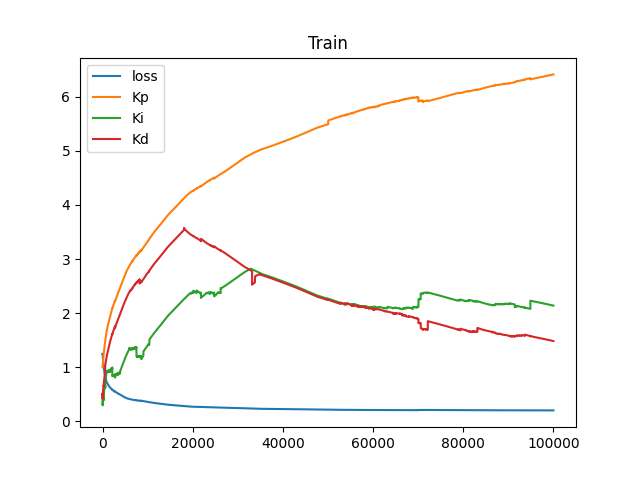
자동 생성된 설명

**그림 6. 조율 인자 최적화 결과 예시**

그림 6은 각 반복 횟수에 따른 최적화 결과를 나타낸다.

|  |
| --- |
| plt.plot(loss, label="loss")      plt.plot(kp, label="Kp")      plt.plot(ki, label="Ki")      plt.plot(kd, label="Kd")      plt.legend()      plt.title("Train")      plt.savefig("train.png") |

위의 코드는 그림 6에서 나타낸 최적화 과정을 도식화하는 코드이다. 위의 코드를 실행하면 그림 7과 같은 결과를 얻을 수 있다.



**그림 7. 조율 인자 도식화 코드 및 결과**

### **[결론]**

PID는 가장 기본적인 제어 시스템 중 하나로, 공정의 목표 값과 실제 데이터 간의 차이가 0이 되도록 조정하며 제어한다. 본 장에서는 딥러닝 기법을 적용하여 PID 제어 시스템을 튜닝했다. 그 결과, 반복 계산을 100,000회 반복하자 오차(loss)가 0으로 거의 수렴한 것을 확인할 수 있다. 더 많은 횟수를 반복한다면 더 정확하게 조율된 PID 제어 시스템을 얻을 수 있을 것이다.

### **[학습 결과]**

* 학습 내용

기본적인 PID 제어 시스템의 이해와 인공지능과의 접목이 필요한 이유 이해.

* 학습 결과 확인하기

대표적인 PID 제어 시스템 조율 방법 익히기.

* 학습 결과 응용하기

본 장의 학습내용에 기반해 실제 공정데이터를 이용해 딥러닝을 적용하여 PID 제어기를 조율하는데 응용.

### **4.2. 인공신경망 모델 기반 예측 제어를 이용한 공정제어**

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | 모델 예측 제어(MPC)의 이해 |
| [방법] | 인공신경망 모델 기반 예측 제어 |
| [응용] | 인공신경망 모델 기반 예측 제어를 이용한 CSTR 제어 |
| [요약] | * 모델 기반 예측 제어의 이해 * Simulink를 이용한 인공신경망 모델 기반 예측 제어 구축 * 인공신경망 모델 기반 예측 제어를 이용한 CSTR 제어 |

**[이론] 모델 예측 제어를 이용한 다변수 공정의 제어**

다변수 공정이란 입출력 변수가 두 개 이상인 공정을 의미하며, MIMO (multi-input multi-output) 공정이라 부른다. 이와 대비되는 개념으로, 입출력 변수가 각각 하나만 존재하는 단변수 공정 (single-input single-output; SISO)이 있다. 증류탑을 비롯한 화학공학에서 우리가 다루는 대부분의 공정들은 다변수 공정에 해당하며, 변수들 간에 상호간섭이 존재한다. 그림 1을 예로 들면, 공정입력(u1 및 u2)이 변하는 경우 상호간섭(G12 및 G21)이 발생하여, 각각의 공정 입력이 여러 개의 공정 출력(y1 및 y2)에 영향을 미치게 된다. 이러한 상호간섭은 단변수 제어인 PID 제어의 성능을 저하시키는 요인으로 작용하며, 상호간섭이 심각할 경우 단변수 제어로는 좋은 제어성능을 기대하기 어렵다.

한편, 실제 공정에서는 공정 입력 및 공정 출력 변수들에 제약조건이 존재한다. PID 제어의 경우, 알고리즘 특성 상 이러한 제약조건을 고려할 수 없는 어려움으로 인해, low selector 또는 high selector의 도입을 통한 override 제어루프의 구성 또는 clamping 등의 방법을 통해 제어시스템을 구성하나, 그 설계가 복잡하고 성능에 한계를 지닌다.

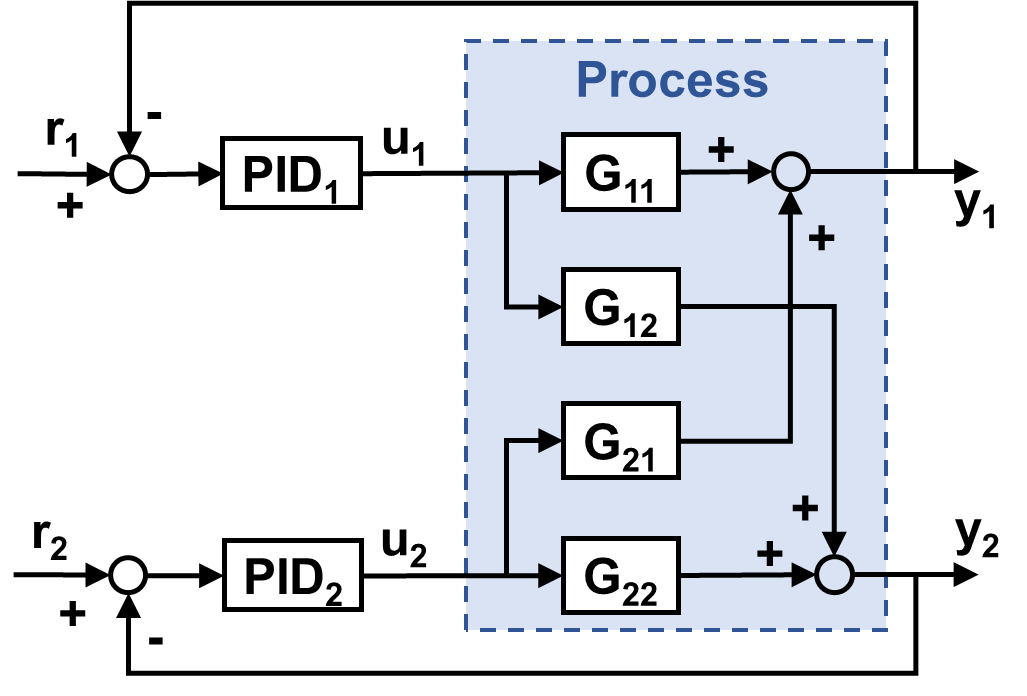


그림1. PID 제어를 이용한 다변수 공정의 제어

다변수 공정을 여러 개의 단변수 제어로 조작하는 것에 대한 어려움과 성능의 한계를 극복하기 위한 좋은 대안으로 다변수 제어가 있다. 연속공정에 대한 대표적인 다변수 제어로서, 그림 2에 보인 바와 같이 모델 예측 제어(model predictive control)가 있다.

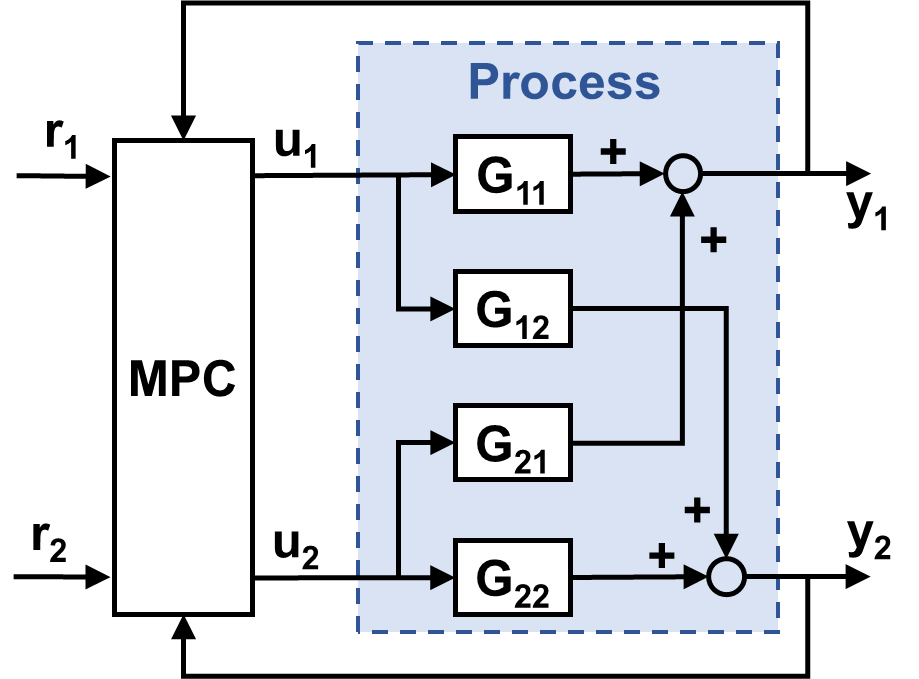


그림 2. 모델 예측 제어를 이용한 다변수 공정의 제어

그림 3은 모델 예측 제어의 동작 원리를 나타낸다. 모델 예측 제어는 실제 공정의 거동을 잘 모사할 수 있는 공정 모델(가상 공정)을 통해, 공정 입력의 변화에 따른 공정 출력을 예측하고, 이를 통해 최적의 공정 입력 시나리오(즉, 단시간 내에 제어오차를 최소화할 수 있는 공정 입력의 변화)를 도출함으로써, 공정을 제어한다. 제어기가 결정하는 공정 입력 값이, 모델을 통해 예측된 공정 출력에 의해 좌우되므로, 제어기의 성능은 공정 모델의 정확성에 크게 의존한다.

텍스트, 영수증이(가) 표시된 사진

자동 생성된 설명

그림 3. 모델 예측 제어기의 동작 원리

전통적으로, 이러한 공정 모델은 부공간 모델인식(subspace model identification) 기법 등을 이용하여, 공정 입력과 공정 출력의 상호관계, 즉, 공정 입력의 변화에 따른 공정 출력의 거동을 잘 표현할 수 있도록 제작되었으나, 최근에는 인공지능 기법을 통해, 더욱 성능이 개선된 새로운 기법들이 제안되고 있다.

본 장에서는 매트랩을 통해 신경망 모델 기반의 모델 예측 제어를 구축함으로써, 성능이 뛰어난 다변수 제어 설계 능력을 함양하는 것을 목적으로 한다.

### **[문제]**

**매트랩을 이용하여 인공신경망 모델 기반 예측 제어를 구축하고 이를 통해 흐름 w1(t)를 조정하여 제품농도를 유지하도록CSTR을 제어하라.**

**- 이때 입력농도는 Cb1=24.9, Cb2=0.1, CSTR 시스템의 동적 모델의 상수 k1과 k2는 각각 1이다.**

**- 또한, 계산의 복잡함을 피하기 위해 w2(t)는 0.1로 가정하고 탱크의 수위 h(t)는 제어되지 않는다고 한다.**

### **[방법] 인공신경망 모델 기반 예측 제어를 이용한 CSTR 제어**

#### 인공신경망 모델 기반 예측 제어의 첫 번째 단계는 ‘시스템 식별(system identification)’이다. 이 단계에 대해 설명하라.

1. 시스템 식별은 그림 4과 같이 플랜트의 순방향 동적 요소를 표현하도록 신경망을 훈련시키는 것을 말한다. 플랜트 출력값과 신경망 출력값 사이의 예측 오차는 신경망 훈련 신호로 사용된다.

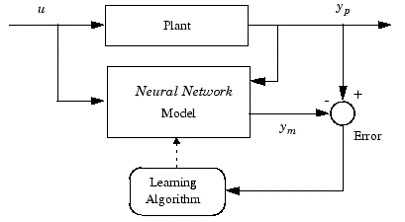


그림4. 인공신경망 훈련 개요

인공신경망 모델은 이전의 직전 입력값과 직전 플랜트 출력값을 사용하여 플랜트 출력값의 미래의 값을 예측한다. 인공신경망 모델의 구조는 그림5와 같이 표현된다.

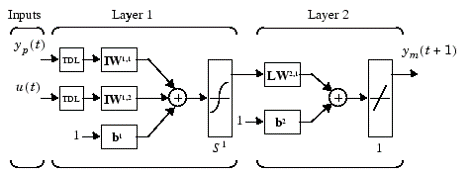


그림5. 인공신경망 모델 개요

#### 인공신경망 모델 기반 예측 제어의 두 번째 단계는 ‘예측 제어(predictive control)’이다. 이 단계에 대해 설명하라.

1. 모델 예측 제어 방법은 이동구간(receding horizon) 기법(SoHa96)을 기반으로 한다. 신경망 모델은 지정된 시간 지평에 대해 플랜트 응답을 예측한다. 예측값은 수치 최적화 프로그램이 지정된 지평에서 다음과 같은 성능 조건을 최소화하는 제어 신호를 파악할 때 사용된다.

텍스트이(가) 표시된 사진

자동 생성된 설명 식 (1)

수식 1. 수치 최적화 수식

여기서 N1, N2, Nu는 추적 오차와 제어 증분이 계산되는 지평을 정의한다. u’ 변수는 잠정적 제어 신호이고 yr은 원하는 응답, ym은 신경망 모델 응답이다. ρ는 제어 증분의 제곱합이 성능 지수에서 차지하는 비중을 결정한다.

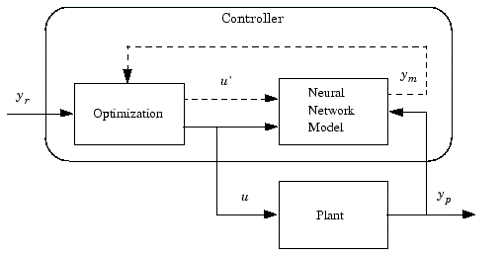


그림6. 모델 예측 제어 과정의 블록 흐름도

그림 6는 모델 예측 제어 과정을 나타낸 것이다. 제어기는 신경망 플랜트 모델과 최적화 블록으로 구성된다. 최적화 블록은 J를 최소화하는 u’의 값을 결정하며, 그런 다음 최적의 u가 플랜트에 입력된다.

#### CSTR(연속 교반 탱크 반응기; continuous stirred-tank reactor)를 도식화하고, 동적 모델을 나타내어라.

1. CSTR을 도식화하면 그림 7와 같이 나타낼 수 있다.

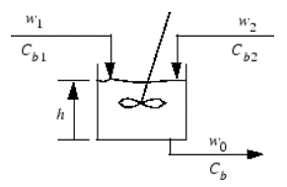


그림 7. 도식화한 CSTR

그림 7를 동적 모델로 나타내면 식 (2)과 식 (3)와 같이 나타낼 수 있다.

식 (2)

식 (3)

여기서 h(t)는 액체 수위이고, Cb(t)는 이 과정의 출력에서의 제품 농도이고, w1(t)는 농축된 공급 용액 Cb1의 유량이고, w2(t)는 희석된 공급 용액 Cb2의 유량이다.

### **[응용] 인공신경망 모델 기반 예측 제어를 이용한 CSTR 제어**

#### 매트립에서 ‘Simulink’ 기능을 활용해 공정 식별(plant identification)을 진행하라.

1. 공정 식별은 그림 8~10와 같은 절차를 통해 할 수 있다.

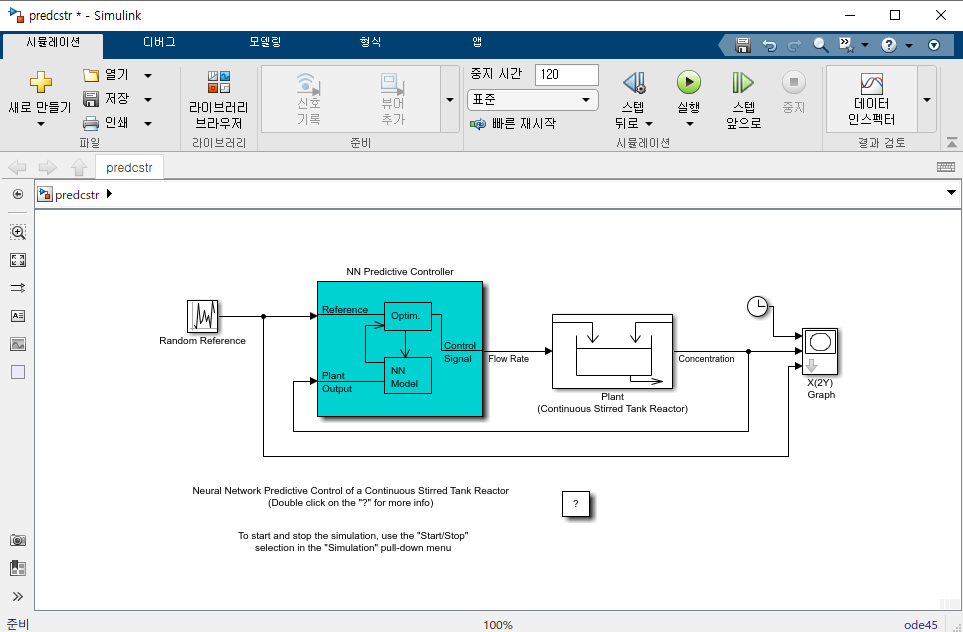


그림8. 실행된 Simulink 모델

매트랩을 실행하여 명령창에 ‘predcstr’을 입력한다. 입력한 명령을 실행하면 그림8과 같이 Simulink 편집기가 실행된다. Plant 블록에는 Simulink CSTR 플랜트 모델이 포함되어 있다. 파랗게 표시된 NN Predictive Controller 블록신호에서 ‘Control Signal은 Plant 모델의 입력에 연결되고, ‘Plant Output’은 Plant 블록 출력에 연결된다. 그리고 ‘Reference’는 Random Reference 신호에 연결된다.

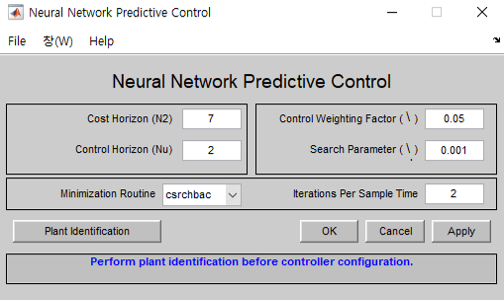


그림9. 모델 예측 제어기 설정창

NN Predictive Controller 블록을 더블 클릭하면 그림 9과 같이 모델 예측 제어기 설정창이 열린다. 이 창에서는 N2와 Nu를 변경할 수 있고, ρ도 정의할 수 있다. 이때, N1은 1로 고정되어 있다. α는 최적화를 위해 필요한 성능 감소분을 결정한다. 또한, 최적화 알고리즘이 사용할 선형 최소화 루틴(Minimization Routine)과 각 샘플 시간에 최적화 알고리즘을 몇 회 반복하여 수행할지 횟수(Iterations Per Sample Time)을 설정할 수 있다.

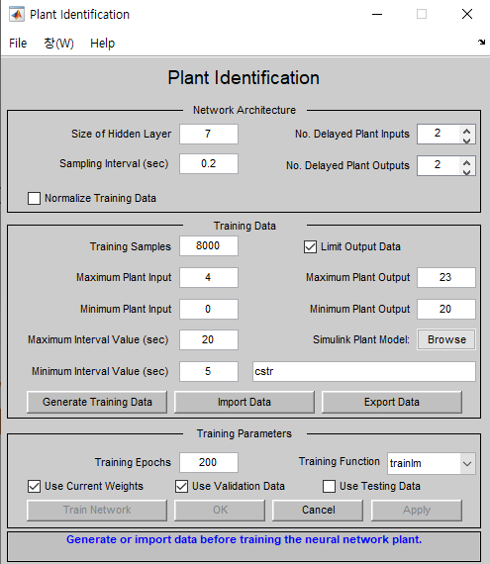


그림10. 인공신경망 플랜트 모델 설정창

플랜트 모델은 미래의 플랜트 출력값을 예측한다. 최적화 알고리즘은 이러한 예측값을 사용하여 미래의 성능을 최적화하는 제어 입력값을 결정한다. 플랜트 모델 신경망에는 앞에서 본 것처럼 하나의 은닉 계측이 있다. 이 창에서 은닉 계층의 크기, 지연 입력값과 지연 출력값의 개수, 훈련 함수를 선택한다.

#### 공정 식별 과정을 통해 생성한 인공 신경망 플랜트 모델을 훈련한 후 시뮬레이션을 실행하여 플랜트 출력값과 기준 신호를 비교하여라.

1. 인공 신경망 플랜트 모델의 훈련은 아래의 그림 11~13를 통해 이루어진다.

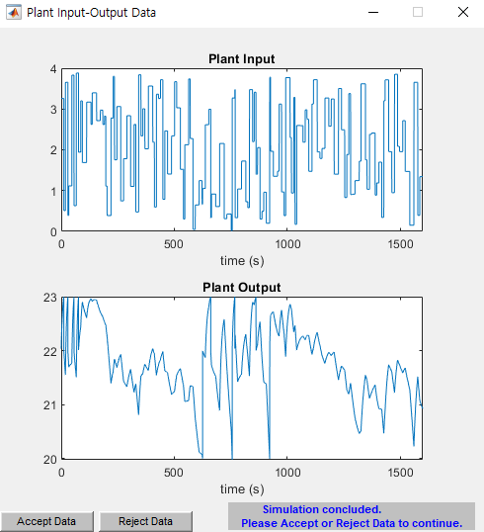


그림11. 인공신경망 플랜트 모델 훈련 결과창

그림10에서 ‘훈련데이터 생성(Generate Training Data)’을 클릭한다. 그림 11와 같이 Simulink 플랜트 모델에 일련의 무작위 스텝 입력값을 적용하여 훈련 데이터를 생성한다. 또한 이를 바탕으로 그림 11의 아래와 같이 잠정적 훈련 데이터가 표시된다. 이 데이터를 통해 미래의 공정 거동을 잘 예측한다고 판단되면 ‘Accept data’를 선택하고, 잘 예측하지 못한다고 판단되면 ‘Refuse Data’를 선택한 후 공정의 거동을 잘 예측할 수 있도록 조건을 변경한다.

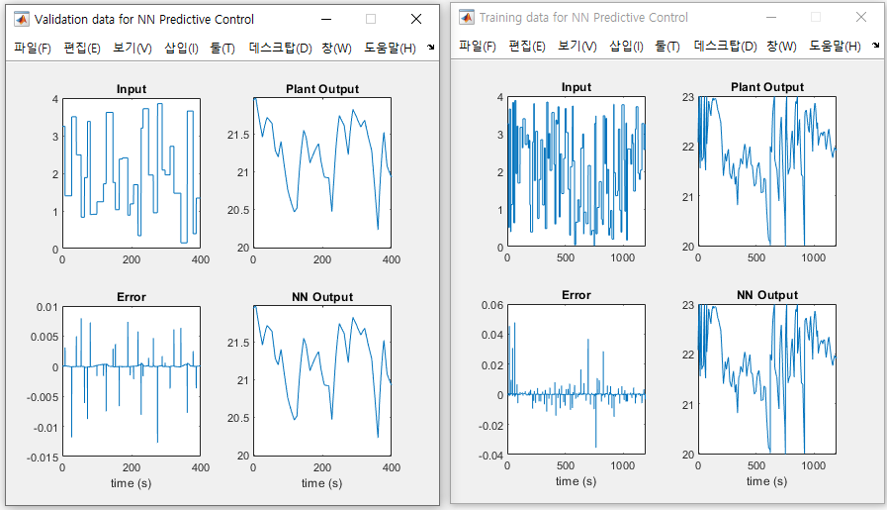


그림12. 플랜트 모델의 응답

그림 11에서 ‘Accept Data’를 클릭하고, 인공신경망 플랜트 모델 설정창(그림 10)에서 ‘Train Network’를 클릭한다. 클릭 시 플랜트 모델 훈련이 시작되고 훈련이 완료되면 그림 12과 같이 플랜트 모델의 응답이 표시된다. 이때, 검증 데이터와 테스트 데이터가 존재하는 경우 각각에 대한 별도의 플롯도 표시된다.

훈련이 완료된 후, 플랜트 모델 설정창(그림 10)에서 ‘Train Network’를 다시 선택하여 동일한 데이터 세트로 훈련을 계속할 수도 있고, ‘Erase Generated Data’를 선택하여 새로운 데이터 세트를 생성할 수도 있고, 현재 플랜트 모델을 수락하고 폐루프 시스템의 시뮬레이션을 시작할 수도 있다.

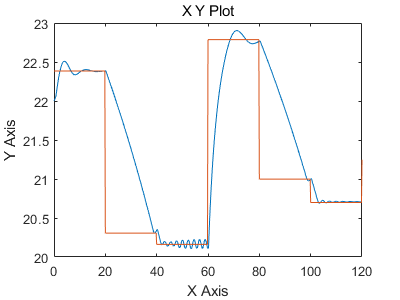


그림13. 플랜트 출력값과 기준 신호의 비교

플랜트 모델 설정창(그림 10)에서 ‘OK’를 선택하면 NN Predictive Controller 블록으로 훈련된 신경망 플랜트 모델을 불러오게 된다. 또한, 모델 예측 제어기 설정창(그림 9)에서 ‘OK’를 선택하면 NN Predictive Controller 블록으로 제어기 파라미터를 불러오게 된다. 이후 Simulink 편집기로 돌아가서 시뮬레이션을 실행하면 그림 13과 같이 플랜트 출력값과 기준 신호가 표시된다.

### **[결론]**

모델 기반 예측 제어는 고급 제어 기법 중 하나로, 공정의 거동을 잘 모사하는 모델을 생성해 이를 제어하는 방법을 통해 제어한다. 본 장에서는 인공신경망을 접목해 인공신경망 모델 기반 예측 제어를 통해 CSTR을 제어하였다. 같은 방법을 통해 보다 복잡한 공정으로 적용분야를 확대하여 공정 제어를 수행할 수 있다.

### **[학습 결과]**

* 학습 내용

모델 기반 예측 제어의 이해 및 인공신경망을 접목한 인공신경망 모델 기반 예측 제어 방법 익히기

* 학습 결과 확인하기

Simulink를 이용한 인공신경망 모델 기반 예측 제어 공정 모델 생성 및 설정하기

* 학습 결과 응용하기

인공신경망 모델 기반 예측 제어를 통해 CSTR 제어 모델 만들기

### **4.3 강화학습(reinforcement learning) 기반 공정제어**

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | Van de Vusse’s reaction 이해 및 강화 학습 기반 공정제어 |
| [방법] | Van de Vusse’s reaction 모델 구축 및 강화 학습을 위한 MDP 정의 |
| [응용] | 강화 학습(DQN 알고리즘)을 이용한 공정제어 |
| [요약] | * Van de Vusse’s reaction 이해 및 모델링 * MDP 설정 및 DQN 이해 * 강화 학습을 사용한 공정제어 |

### **[이론] - Van de Vusse’s reaction control 문제에 DQN 알고리즘 바탕의 강화 학습을 사용한 공정제어.**

Van de Vusse’s reaction control 문제는 1964년에 제안된 isothermal CSTR반응기에서 진행되는 공정으로써 다음과 같은 두 개의 반응이 존재한다. (출처: Van de Vusse, J. G., “Plug-Flow Type Reactor versus Tank Reactor”, *Chemical Engineering Science*, **19**, 964 (1964))

|  |  |
| --- | --- |
|  | (1) |
| 텍스트이(가) 표시된 사진  자동 생성된 설명 | (2) |

이때, 반응 상수와 농도를 사용하여 mass balance를 통해 해당 공정의 다이나믹스(dynamics)를 표현하면 다음과 같이 표현될 수 있다.

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |

여기서 은 반응물 A의 농도, 는 중간 생성물인 B의 농도, 는 feed에 있는 반응물의 농도, 그리고 는 희석률(dilution rate)이다. 각 상수의 값은 , , , , 그리고 이다.

이번 실습문제에서는 불확실성을 포함하는 시스템을 다루고자 하는데 이를 위해 반응 상수 의 값이 연속균등분포(continuous uniform distribution)를 갖는다고 가정하고 CSTR반응기를 제어하고자 한다. 이러한 시스템 하에서 제어의 목적은 0.1 시간 동안 희석률()을 바꾸어 가며 반응 중간생성물의 농도()를 에 맞추는 것이며, 샘플링 시간은 0.002시간이다. 이때, 현실적이고 부드러운 제어를 위하여 현재의 희석률과 과거의 희석률의 차이(delta)를 줄이는 것도 목적중의 하나이다.

이러한 제어 문제를 풀기 위하여 강화 학습을 도입할 수 있다. 챕터 8에서 강화 학습의 개념이 설명이 되는데, 이는 에이전트가 MDP로 표현된 시스템 하에서 상태(state)를 관찰하여 여러 행동(action)을 취해본 뒤, 받는 보상(reward)을 토대로 스스로 최적의 행동 정책을 찾아가는 기계학습 방법론이다. 이번 예제에서 사용될 강화 학습 알고리즘은 DQN이다. 이는 deep q network의 줄임 말로써, Google의 DeepMind (알파고를 만든 회사)에서 개발한 알고리즘이다. 챕터 8에서 설명한 큐러닝 알고리즘과 같은 off-policy의 TD-learning 알고리즘이지만, 핵심적인 차이는 큐함수를 구함에 있어서 인공신경망을 사용하여 근사치를 구한다는 것이고, 또한 특정 상황에서 행동을 했을 때 받은 결과를 replay buffer에 저장을 한 후, buffer에서 여러 개의 batch를 랜덤하게 뽑아서 학습을 진행한다. 이는 강화학습을 진행함에 있어 데이터의 시계열 적인 상관관계를 깨서 더욱 안정적인 정책 학습을 가능하게끔 한다. 딥 마인드에서 아타리 게임에 DQN을 적용하여 게임 해 나가는 동영상으로 유명한 알고리즘이다 (<https://www.youtube.com/watch?v=V1eYniJ0Rnk>).

문제와 시스템 상황을 명확하게 표현할 수 있는 MDP를 정의하는 것은 강화 학습에서 가장 중요한 일이다. MDP는 챕터 8에서와 같이 상태, 행동, 보상, 상태 변환 함수, 그리고 할인률 5가지로 이루어진다. 본 챕터의 공정 문제를 MDP로 표현한다면 다음과 같이 표현이 가능하다:

* 상태:
* 행동:
* 보상:
* 상태 변환 함수: 본 챕터의 식 (3), (4)
* 할인율: 0.95

본 실습에서는 위와 같이 정의된 MDP에서 DQN 알고리즘이 300번 에피소드 동안 스스로 학습해 나가며 최적의 제어 정책을 찾아가는 과정을 실습할 것이다. 강화학습에서의 에피소드는 의사결정의 시작으로부터 특정한 정책을 가지고 의사결정을 해 나간 뒤, 종결 상태 (terminal state)에 도달하여 의사결정이 끝나기 까지의 과정을 의미한다. 이때, 큐함수는 특정 상태에서 어떠한 정책을 가지고 행동을 하였을 때, 에피소드가 끝나기 전까지 어떠한 보상을 받게 될 지에 대한 함수이다. 따라서, 강화학습에서 에이전트는 현재 상태에서 각 행동의 큐함수를 계산하여 큐함수의 값이 가장 높은 행동을 취하게 된다. 현 제어 문제에서 에피소드는 전체 제어의 지평선(horizon)인 0.1 시간이 끝나는 시점에 종결이 되게 된다. 큐함수는 수학적으로 식 (5)와 같이 표현될 수 있다.

|  |  |
| --- | --- |
|  | (5) |

이번 실습에서 에이전트가 관찰하게 되는 상태는 특정 시간의 농도이고, 할 수 있는 행동은 0~70까지의 5단위의 희석률이다. 일반적인 큐러닝 알고리즘을 통한 큐함수의 업데이트는 수식 (6) 과 같이 표현되지만, DQN의 경우, 큐함수를 근사하기 위하여 인공신경망을 사용하기에, 가 로 표현되는 특징이 있고, 이때 인공신경망의 파라미터인 를 업데이트 하기 위해서 사용하는 오류함수는 식 (7)과 같이 표현된다. 오류함수는 의 형태로 정의되는데 이때, 정답()을 내는 함수가 매 업데이트 마다 값이 계속 변하게 되면 러닝이 안정적으로 되지 않을 수 있기에, 오류 함수에서 정답 부분에 사용되는 인공신경망은 일정 시간 유지하다가 특정한 간격마다 업데이트를 하는 특징이 있다.

|  |  |
| --- | --- |
|  | (6) |
|  | (7) |

### **[문제] MDP로 표현된 Van de Vusse’s reaction control 문제를 코드화 하여 DQN 알고리즘이 학습될 수 있는 기반을 마련하고 실제로 에이전트를 트레이닝 하며 점점 나아지는 제어 성능을 관찰한다.**

- “main.py, Environment.py, Agent.py” 코드를 활용하여라.

- matplotlib, scipy, keras 라이브러리를 설치 하여라

- 챕터 8을 읽고 강화 학습의 이론을 이해하여라.

### **[방법] Van de Vusse’s reaction control 문제를 MDP로 표현하여 강화 학습을 위한 environment 구축 및 DQN 에이전트 셋업**

#### 관련 라이브러리를 설치하고 정의한 MDP를 environment로 코드화 해보자.

1. 필요한 라이브러리 등을 불러온 이후, Env()라는 class를 정의하게 되는데, 이때 class로 객체를 만들었을 때, \_init\_에서 MDP에서 표현한 것과 같이 state은 를 받아드릴 수 있게끔 2개의 차원을 할당, action은 15개의 가능성이 있기에 15개의 차원을 할당, 초기 의값을 설정 sampling time 설정, 그리고 시간을 초기화 한다. 이후 reset이라는 함수를 정의하여 에피소드가 끝난 후, 다시 새로운 환경이 주어질 수 있게끔 한다. 그리고 step, system\_functions, reward\_function이라는 함수를 Env() class 내에 정의하여 MDP를 코드로 표현한다.

|  |
| --- |
| class Env():  def \_\_init\_\_(self):  self.s\_dim = 2  self.a\_dim = 15 *#action (0, 5, 10, ..., 65, 70)* self.x0 = np.array([[0., 0.]])  self.u0 = np.array([[7.]])  self.dt = 0.002  self.t0 = 0.  self.system\_eval = partial(self.system\_functions)   def reset(self):  self.state = self.x0  self.action = self.u0  self.prev\_action = self.u0  self.done = False  self.set\_point = 1.2  self.time = self.t0  self.uncert\_param = np.random.uniform(45., 55.)  return self.state, self.action, self.uncert\_param, self.time   def step(self, u):  x = self.state  t = self.time  u = np.array([float(u)])  dx = partial(self.system\_eval, u=u)  x = np.reshape(x, [-1, ])  sol = solve\_ivp(dx, [0, self.dt], x)  next\_x = np.reshape(sol.y[:, -1], [1, -1])  t += self.dt  self.time = t  reward = self.reward\_function(next\_x, u)  done = self.done\_function(next\_x)  self.state = next\_x  self.prev\_action = u  return next\_x, reward, done, t   def system\_functions(self, t, x, u):  x = x  x1, x2 = np.reshape(x, [-1, ]) *#unpack states* u = np.reshape(u, [-1, ])  u = u \* 5 *# (0, 1, ... 14) -> (1, 5, 10, 15, ..., 7)* k1 = self.uncert\_param *#(h-1)* k2 = 100 *#(h-1)* k3 = 10 *#(l/mol/h)* x1f = 10 *#concentration of the reactant (mol/l)* dx = [- k1\*x1 - k3\*(x1\*\*2) + (x1f - x1)\*u,  k1\*x1 - k2\*x2 - x2\*u]  dx = np.reshape(dx, [1, -1])  return dx   def reward\_function(self, x, u):  x = x  x1, x2 = np.reshape(x, [-1, ])  u = np.reshape(u, [-1, ])[0]  prev\_u = np.reshape(self.prev\_action, [-1, ])[0]  reward = -(x2 - self.set\_point)\*\*2 - 0.01\*(u-prev\_u)\*\*2  return reward |

#### DQN agent를 코드로 표현해보자.

1. DQN class를 정의하기 위하여, 처음에 구동되는 \_init\_함수에는 기본적인 세팅이 들어가야 한다. 상태의 크기, 행동의 크기, 할인율, 그리고 러닝과 관련된 하이퍼 파라미터들(buffer의 크기, 입실론 관련 파라미터들)을 지정해준다. 하이퍼 파라미터인 입실론은 exploration에 초반에 중점을 두는 러닝에서 시간이 감에 따라 exploitation에 중점을 두는 러닝으로 서서히 바뀌게끔 유도를 한다. 이러한 이해를 기반으로 파라미터를 직접 바꾸어 가며 어떻게 러닝이 되는 양상이 바뀌어 가는지에 대한 이해를 늘려가는 것을 추천한다. 이후, DQN의 핵심인 인공신경망을 build\_model 함수에 정의한다. 이때 신경망의 깊이와 node의 개수, activation 함수의 형태 모두 바꿀 수 있는 하이퍼 파라미터이다. 또한 학습을 위한 remember 함수와, 상태를 받아서 어떠한 행동을 하는지에 대한 act함수를 정의함으로써 DQN 에이전트를 정의할 수 있다.

|  |
| --- |
| class DQNAgent:  def \_\_init\_\_(self, state\_size, action\_size):  self.state\_size = state\_size  self.action\_size = action\_size  self.memory = deque(maxlen=2000)  self.gamma = 0.95 *# discount rate* self.epsilon = 1.0 *# exploration rate* self.epsilon\_min = 0.001  self.epsilon\_decay = 0.9995  self.learning\_rate = 0.001  self.model = self.\_build\_model()   def \_build\_model(self):  *# Neural Net for Deep-Q learning model* model = Sequential()  model.add(Dense(24, input\_dim=self.state\_size, activation=**'relu'**))  model.add(Dense(24, activation=**'relu'**))  model.add(Dense(self.action\_size, activation=**'linear'**))  model.compile(loss=**'mse'**,  optimizer=Adam(lr=self.learning\_rate))  return model   def remember(self, state, action, reward, next\_state, done):  self.memory.append((state, action, reward, next\_state, done))   def act(self, state):  if np.random.rand() <= self.epsilon:  return random.randrange(self.action\_size)  act\_values = self.model.predict(state)  return np.argmax(act\_values[0]) *# returns action* |

### **[응용] 정의된 MDP내부에서 에피소드를 진행하며 강화 학습 기반 공정제어를 진행해보자.**

#### 한 에피소드 동안 특정 정책을 가지고 환경(environment)와 상호작용을 통해 정책 업데이트를 위한 reply buffer를 구축해보자.

1. 한 에피소드 내에서 상태, 행동 등을 reset한 뒤, 500번 동안 현재 가지고 있는 정책을 가지고 행동을 하여 그때 받은 보상들을 data\_history (replay buffer)에 차곡차곡 저장을 한다. 이때, 모은 데이터가 batch의 사이즈를 넘어가면 모인 데이터를 바탕으로 정책 업데이트를 진행한다. 또한 한 에피소드가 진행됨에 따라 얻어진 reward의 총 합은 score라는 변수 아래에 저장이 된다. (score에 저장되는 값은 강화학습에서 return이라고 일반적으로 지칭한다)

|  |
| --- |
| state, action, uncert\_param, t = env.reset() state = np.reshape(state, [1, state\_size]) score = 0 ep\_data\_history = np.zeros([1, 2 \* state\_size + 3]) *# time, state, action, next\_state, reward* for n in range(500):  action = agent.act(state)  next\_state, reward, done, t = env.step(action)  next\_state = np.reshape(next\_state, [1, state\_size])  agent.remember(state, action, reward, next\_state, done)  temp\_data\_history = np.append(t, state)  temp\_data\_history = np.append(temp\_data\_history, action\*5)  temp\_data\_history = np.append(temp\_data\_history, next\_state)  temp\_data\_history = np.append(temp\_data\_history, reward)  temp\_data\_history = np.reshape(temp\_data\_history, [1, -1])  ep\_data\_history = np.concatenate([ep\_data\_history, temp\_data\_history])  state = next\_state  score += reward   if len(agent.memory) > batch\_size:  agent.replay(batch\_size) |

#### 정책을 업데이트 해가며 300회 에피소드를 진행하고, 각 에피소드의 공정 제어 결과를 저장하여라.

1. env 라는 환경을 모사한 class와 agent라는 DQN 에이전트를 생성을 한다. 이후, reply buffer에서 몇 개를 뽑아서 정책 업데이트를 진행할지에 대한 하이퍼 파라미터인 batch\_size를 정의하고, 총 에피소드의 수를 300으로 결정한다. 이후 에피소드의 횟수만큼 A3. 의 코드를 진행하여 공정제어와 정책 학습을 반복해 나간다.

|  |
| --- |
| env = Env() state\_size = env.s\_dim action\_size = env.a\_dim agent = DQNAgent(state\_size, action\_size) *# agent.load("./save/project.h5")* done = False batch\_size = 32 EPISODES = 300 RANDOM\_SEED = 1234 np.random.seed(RANDOM\_SEED) ep\_score\_history = np.array([])  for e in range(EPISODES):  A3.의 코드 |

#### 1번째 에피소드, 101번째, 201번째, 300 번째 에피소드의 공정 제어 결과 (시간에 따른 score, , 의 그래프)를 나타내어라.

1. 아래와 같은 코드를 활용하여 1번째, 101번째, 201번째, 그리고 300번째 에피소드의 score값, ,의 시간에 따른 변화 값을 프린트 할 수 있다.

|  |
| --- |
| if done:  print(**"episode: {}/{}, score: {: .3}, e: {: .3}"**.format(e, EPISODES, score, agent.epsilon))  ep\_score\_history = np.append(ep\_score\_history, score)   plt.subplot(311)  plt.title(**"score"**)  plt.scatter(e, score, c=**'blue'**)   if (e % 100 == 0) or (e == EPISODES-1):  t\_history = ep\_data\_history[:, 0]  x2\_history = ep\_data\_history[:, 2]  u\_history = ep\_data\_history[:, 3]  plt.subplot(312)  plt.title(**"x2"**)  plt.plot(t\_history, x2\_history, label=(e+1))  plt.legend()  plt.ylim(0, 1.5)  plt.axhline(y=1.2, color=**'r'**, linestyle=**'--'**, linewidth=1)   plt.subplot(313)  plt.title(**"u"**)  plt.step(t\_history, u\_history, label=(e+1))  plt.legend() |

첫번째로 볼 수 있는 에피소드에 따른 score 값의 변화는 에피소드의 번호가 뒤로 감에 따라 (학습이 더 진행된 정책기반 의사결정) score 값이 증가함을 알 수 있다. 즉, 점점 reward를 많이 받는 행동을 매 의사결정마다 자주 했었다는 의미이고, reward를 많이 받았다는 것은 우리가 처음에 설정한 목적을 더욱 잘 달성했다는 의미이다. (그림1)

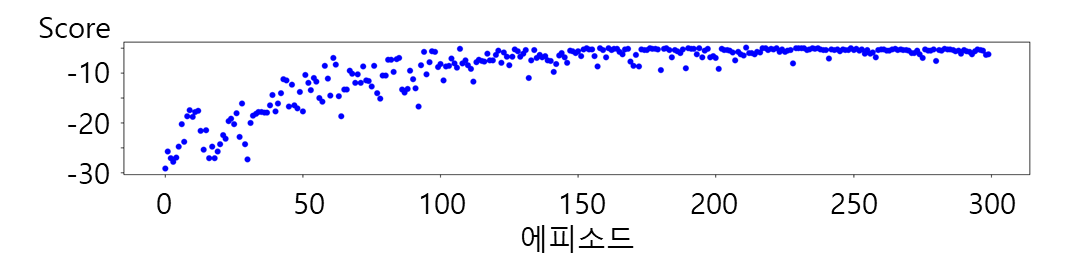


그림1. 에피소드 별 score 값

두번째로 불 수 있는 의 경우, 첫번째, 101번째, 201번째, 300번째의 에피소드로 학습이 진행됨에 따라 점점 기존의 set point였던 1.2 라는 값에 빠르고 안정적으로 안착을 하는 모습을 볼 수 있다. (그림2)

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명

그림2. 에피소드 마다 시간에 따른 의 값

마지막으로 볼 수 있는 의 경우 1번째 에피소드의 경우 의 값이 각 스텝마다 굉장히 급격하게 바뀌는 모습을 솔 수 있지만, 101번, 201번, 300번 에피소드로 정책이 진화함에 따라 굉장히 부드럽고 천천히 바뀌기에, 안정적인 제어를 할 수 있는 노하우를 DQN 에이전트가 터득했다고 볼 수 있다. (그림3)

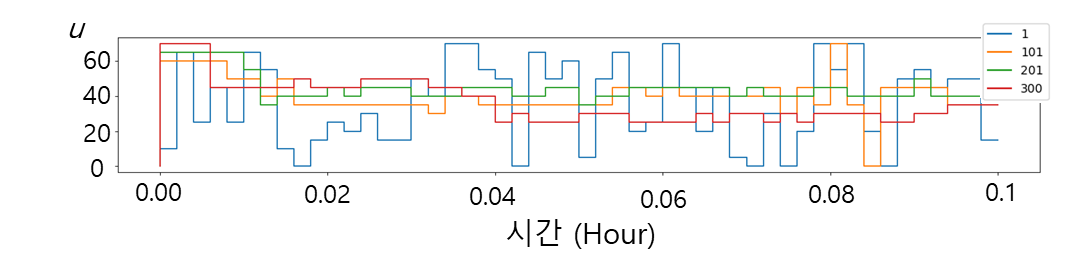


그림3. 에피소드마다 시간에 따른 의 값

### **[결론]**

본 장에서는 강화학습 기반 공정 제어를 Van de Vusse’s reaction control 문제에 적용하여 공정제어와 에이전트 학습을 실제로 진행해보았고, 이러한 공정 제어 결과를 에피소드 별로 plot 하여 에피소드가 진행함에 따라 처음 설정한 목적에 맞는 제어 방식을 강화학습 에이전트가 실시간으로 배워 나가는 과정을 몸소 확인해보는 공부를 하였다.

### **[학습 결과]**

* 학습 내용

Van de Vusse’s reaction control 문제를 이해하고 MDP로 코드화 시켜 보기

* 학습 결과 확인하기

DQN 알고리즘 기반 강화학습 에이전트를 구성해보기.

* 학습 결과 응용하기

생성된 강화학습 에이전트를 Van de Vusse’s reaction control 문제에 적용하여 공정 제어 퍼포먼스를 실시간으로 확인해보기

### **5.1. 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] | * 화학 공정의 운전 상태 분석 방법 이해 * 오토인코더 모델 훈련 후 검증 * 공정 상태 모니터링 및 기여도 분석 |

### **[공정설명] 테네시 이스트만 공정**

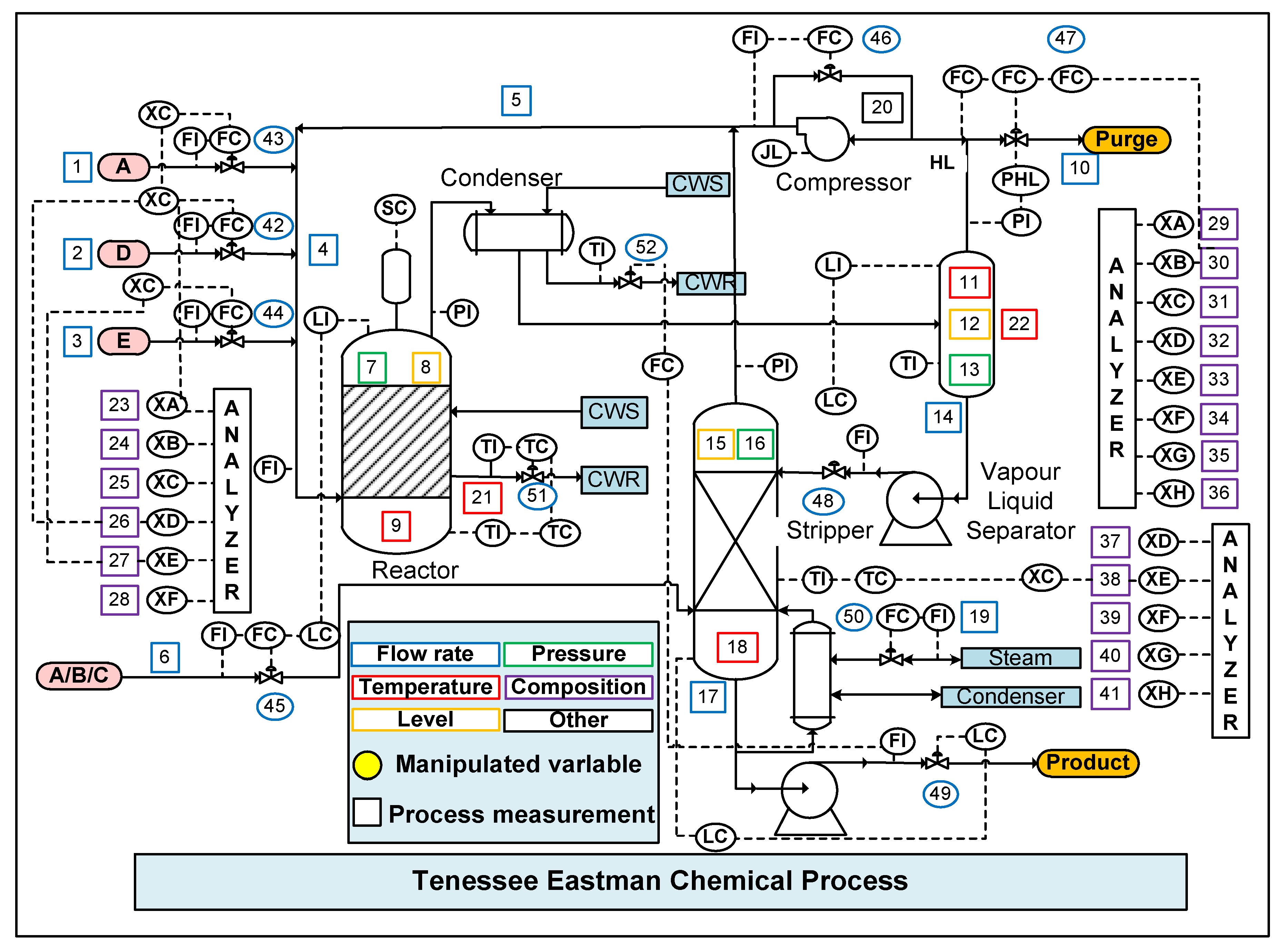


그림 1. 테네시 이스트만 공정의 흐름도

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.

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.

표 1. 테네시 이스트만 공정의 이상 시나리오

|  |  |  |
| --- | --- | --- |
| Fault | Description | Type |
| 1 | A/C feed ratio, B composition constant | Step |
| 2 | B composition, A/C ratio constant | Step |
| 3 | D feed temperature | Step |
| 4 | Reactor cooling water inlet temperature | Step |
| 5 | Condenser cooling water inlet temperature | Step |
| 6 | A feed loss | Step |
| 7 | C header pressure loss - reduced availability | Step |
| 8 | A, B, C feed composition | Random variation |
| 9 | D feed temperature | Random variation |
| 10 | C feed temperature | Random variation |
| 11 | Reactor cooling water inlet temperature | Random variation |
| 12 | Condenser cooling water inlet temperature | Random variation |
| 13 | Reaction kinetics | Slow drift |
| 14 | Reactor cooling water valve | Sticking |
| 15 | Condenser cooling water valve | Sticking |
| 16-20 | Unknown | Unknow |
| 21 | Valve for stream 4 | Constant position |

### **[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**

#### 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.

#### 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.

#### 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.

#### 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.

#### 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.

#### 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.

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

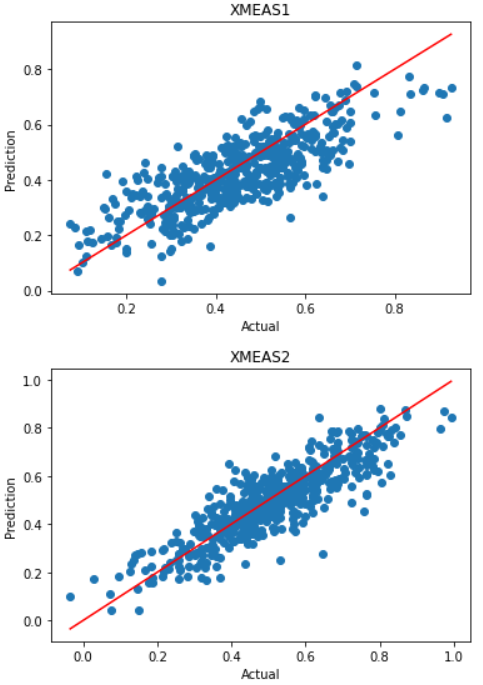
|  |
| --- |
| from keras.models import load\_model  from sklearn.metrics import mean\_squred\_error, mean\_absolute\_error  model = load\_model(‘best model.h5’)  X\_pred = model.predict(X\_test\_scaled)  mse = mean\_squared\_error(X\_test\_scaled, X\_pred)  mae = mean\_absolute\_error(X\_test\_scaled,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

#### 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.

#### 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() |

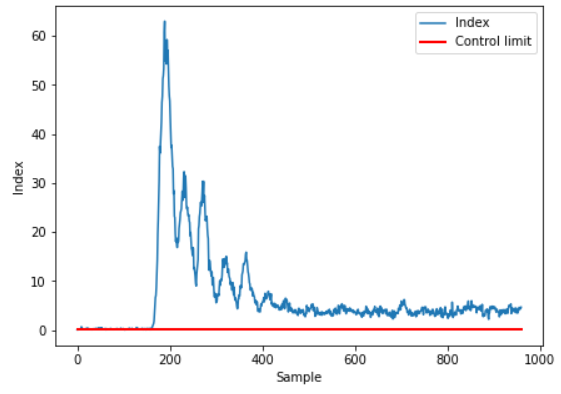


공정 모니터링 성능을 확인하기 위해 Fault 1 데이터를 불러온다. Fault 1 데이터는 960개의 샘플로 이루어져 있고 160번째 샘플부터 A/C feed ratio의 step change가 발생한다.

|  |
| --- |
| 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.

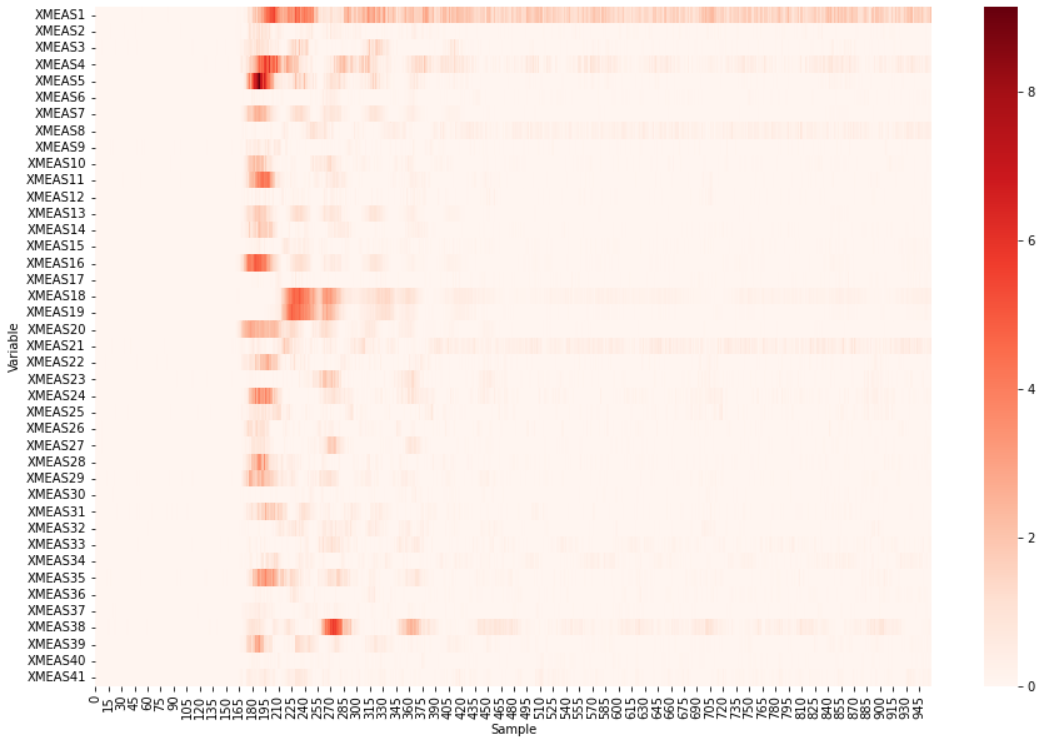
|  |
| --- |
| plt.figure(figsize=(7,5))  plt.plot(spe, label=’Index’)  control\_limit = np.full((len(spe), 1), spe\_cl)  plt.plot(np.arange(len(spe)), control\_limit, color=’r’, linewidth=2, label=’Control limit’)  plt.xlabel(‘Sample)  plt.ylabel(‘Index’)  plt.title(‘Fault1’)  plt.show() |



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

#### 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 XMEAS1 변수가 Fault1에 가장 큰 원인임을 밝혀냈다.

### **[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