K-인공지능 제조데이터 분석 경진대회 보고서

프로젝트명	용해탱크 장비이상 조기탐지
팀명	MM3
내용요약	현장에서는 완제품의 목표 품질, 투입원료의 특성, 생산량 등 주요 요인을 모두 고려하여 설정한 제품별 설비운영 기준값에 따라 공정을 운영하여도, 용해 품질에 영향을 미치는 다른 요인들은 항상 있으며, 현장작업자는 경험과 노하우 등 암묵지에 의존하여 대처할 수밖에 없게 된다. 주관적 판단에 의존한 대처 방법이 아닌 설비운영값과 주요 품질검사항목의 결과값을 분석하여 이를 바탕으로 생산 공정 중에도 생산품질을 예측하여 즉시 공정 제어에 이용하기 위한 AI알고리즘 모델을 만들고, 모델의 정확도와 견고성을 검증하였다. 또한 이 모델의 시사점과 중소기업 산업현장의 파급효과를 확인하여 보았다.

상기 본인(팀)은 위의 내용과 같이 K-인공지능 제조데이터 분석 경진대회 결과 보고서를 제출합니다.

2022 년 11 월 09 일

팀장 : [redacted]
팀원 : [redacted]
팀원 :

한국과학기술원장 귀중

용해탱크의 품질향상을 위한 자동화 품질측정모델 개발 (품질판단 시스템 구축)

(2022년 제2회 K-인공지능 제조데이터분석 경진대회)

분석팀 : MM3

분석자 : [redacted]

목 차

1. 문제 정의

1.1 과정요약

본 보고서는 제2회 K-인공지능 제조데이터 분석 경진대회의 해결 과제로 식품가공기업의 작업환경 개선 및 생산성 향상을 위한 아이디어를 제시하고 인공지능 알고리즘으로 구현 제출하는 것을 목적으로 작성되었다. 아래는 경진대회의 목적, 제공된 Data 및 본 연구에서 활용된 알고리즘을 서술하였다. 본 연구가 기업의 작업 및 생산성 향상에 기여될 것을 기대한다.

No	구 분		내 용
1	분석 목적 (현장 이슈, 목적) 제조공정		- 전처리 공정의 첫 번째 단계인 용해공정(용해탱크)에서는 분말 및 액상 원재료를 정제수 등에 용해/혼합 후, 후공정에서 다시 분말화하기 때문에 용해탱크에서 원재료가 균일하게 혼합되는 것이 매우 중요하다. - 이에 따라 용해탱크 운영데이터 분석을 통한 품질예측에 대한 의미가 커지며, 특히 불량 발생 후 원인분석을 진행하는 결과론적 품질분석이 아닌 예견된 불량을 효율적으로 방지할 수 있는 예지적 품질분석이 필요하게 되었다.
2	데이터셋 형태 및 수집방법		- 분석에 사용된 변수 : 용해온도, 교반속도, 수분함유량 - 데이터 수집 방법 : PLC, DBMS(RDB) - 데이터셋 파일 확장자 : csv
3	데이터 개수 데이터셋 총량		- 데이터 개수 : 835,200건 - 데이터셋 총량 : 35.2MB
4	분석적용 알고리즘	알고리즘	- 알고리즘 GRU(Gated Recurrent Unit)을 이용하여 시계열 데이터 기반의 품질예측을 수행하고 예측 정확도를 확보한다.
		알고리즘 간략소개	- GRU란, 시계열 예측 문제에서 순서 의존성을 학습함으로써 RNN이 갖는 장기 의존성 문제해결 및 LSTM의 복잡한 구조 단순화 및 실행시간의 단축을 위해 개발된 특수한 종류의 RNN 알고리즘이다. - 신경망의 중간 계층에서 각 유닛을 ‘GRU 블록’이라는 메모리 유닛으로 치환한 구조로, Reset gate, Update gate 2개의 gate를 활용하여 학습한다.
5	분석결과 및 시사점		- 공정중 용해탱크 설비운영값을 기준으로 제품의 최종 품질을 예측할 수 있는 AI 기법을 적용하여 용해 공정의 주요 운영변수가 전체적으로 품질에 미칠 영향 모델링을 도출하였다. - 용해공정의 설비운영 데이터와 최종품질검사 데이터를 수집하고, 데이터 가공/전처리, AI 모델 개발과 제조공정의 적용 및 검증을 통해 열악한 중소기업에 빅데이터 및 AI 기술을 적용하여 실질적인 품질 향상 및 비용절감에 기여하는 점에서 시사하는 바가 크다고 판단된다.

1.2 공정(설비) 정의 및 특징

- 용해공정은 분말 원재료를 액상 원재료에 녹이는 공정으로, 식품, 화학, 석유화학 등 다양한 분야에서 적용된다. 본 가이드북의 용해공정은 분말 유크림, 기능성 조제 분말 등을 생산하는 식품제조업의 용해공정으로, SD/MSD 건조생산라인의 원료 전처리 작업의 첫 번째 단계에 속한다.
- 살균, 분무건조 등 후공정에서의 공정품질을 보장하기 위해서는 용해공정에서 다양한 분말 및 액상 원료가 균일한 혼합물(용액)이 되는 것이 매우 중요하다. 용해공정의 품질, 즉 용해여부는 용질과 용매의 화학적 특성(극성), 용질과 용매의 상대적 용량, 용해온도, 물리적 힘 등에 영향을 받는다.

1.3 공정(설비)상의 문제현황 및 문제해결 장애요인

1) 공정(설비)상의 문제현황

- 용해공정은 원재료의 전처리를 수행하는 첫 번째 단계이니만큼 본 공정의 품질이 후 공정 및 완제품의 품질에 미치는 영향이 크다. 특히 분말 및 액상 원료를 용해 후, 후공정 단계에서 다시 분말화하기 때문에 전처리 단계에서 모든 원료가 균일하게 혼합되는 것이 매우 중요하다.
- 그러나 현장에서는 완제품의 목표 품질, 투입원료의 특성, 생산량 등 주요 요인을 모두 고려하여 설정한 제품별 설비운영 기준값에 따라 공정을 운영하여도, 용해 품질에 영향을 미치는 다른 요인들은 항상 있으며, 현장작업자는 경험과 노하우 등 암묵지에 의존하여 대처할 수밖에 없게 된다.
- 특히 대량의 제품을 생산하는 경우나 다수개의 원료를 투입하는 경우에는 원료를 한 번에 투입하여 용해하기보다는 시차를 두고 순차적으로 투입하게 되는데, 투입시마다 새로 투입되는 원료로 인해 내용물의 온도와 점도, 탱크 교반속도 등에 변화가 발생한다. 용해 품질을 유지하기 위해서는 이에 대응하여

설비운영값을 변경하여야 하는데, 모든 경우가 다르고, 정해진 가이드라인이 없어 작업자는 경험과 노하우에 의존하여 대처하고 있으며, 이마저도 인력에 공백이 생기는 경우 대처가 어렵다.

- 이 외에도 원료 납품업체가 변경되어 원료의 특성이 미묘하게 변경되거나 계절/날씨 등으로 인해 습도가 높은 날에 원료 상태나 용해탱크 상태가 영향을 받는 등 공정품질에 영향을 미치는 요인은 산재한다.

2) 문제해결 장애요인

- 일반적으로 용해탱크에는 자체적으로 용해상태를 확인할 수 있는 측정방법(센서 등)이 부재하기 때문에 작업자가 직접 공정 중에 용액샘플을 채취하여 육안으로 상태를 확인하거나, 품질검사를 진행하여 공정 중간중간 용해정도에 맞게 온도, 교반속도 등의 설비운영 셋팅값을 조정하는 방식으로 작업한다.
- 하지만 그보다 더 많은 경우에는 설비 또는 현장 상황으로 인해 중간에 내용물을 확인하는 것이 불가능하여 용해상태를 확인 못하고, 후공정까지 진행되어서야 품질확인이 가능한 경우도 있어, 생산단계가 상당히 많이 진행되기까지 품질이 보장되지 못한다.

1.4 해결하고자 하는 문제 및 제조데이터 분석 목표

- 공정 진행 중 계속 변화하는 내용물의 상태나 설비 상태, 공정 진행 중에는 용해상태를 직접 확인/검사하지 못하는 현장 상황 등의 조건 속에서 지금과 같이 주관적 판단에 의존한 대처 방법이 아닌 데이터에 기반한 품질예측 방법을 통한 공정품질 확보한다.
- 현실적으로 대부분의 제조업 상황에서는 모든 요인을 전반적으로 고려한 최적의 설비운영값을 산술적으로 도출해내기는 불가능에 가깝기 때문에, 영향요인 중 설비 PLC를 통해 직접 추출이 가능한 용해 온도, 교반속도, 내용량과 제품 품질검사를 수행하여 획득 가능한 수분함유량 등 주요 변수를 모두 활용하여 머신을

학습시키고,
공정 중 설비운영 데이터를 학습모델에 적용하여 완제품 품질을 예측하고자 한다.

- 다양한 요인으로 인해 공정 중 실시간으로 변화하는 설비운영값(온도, 교반속도, 내용량)과 주요 품질검사항목의 결과값을 모델링 하여 이를 바탕으로 생산 중에도 생산품질을 예측하여 즉시 공정 제어에 이용할 수 있을 것이다. 또한, 학습모델의 정확도를 파악하여 모델의 현장적용 가능성을 살펴본다.

2. 제조데이터 정의 및 처리과정

2.1 제조 데이터 정의 및 소개

- 원료의 용해상태는 용해 온도, 교반속도, 내용량에 영향을 받고, 원료 용해상태는 최종 제품품질에 영향을 미치기 때문에, 용해공정 설비운영 데이터와 제품 품질검사를 수행하여 획득한 수분함유량 데이터를 활용하여 품질예측을 하기 위한 머신학습 데이터이다.
- 설비운영 데이터는 용해탱크의 PLC를 통해 일반적으로 수집할 수 있는 데이터를 범위로 하였으며, 품질데이터는 식품제조업에서 일반적으로 진행하는 품질검사결과를 범위로 하여, 본 보고서를 따라 학습하고 유사 현장에 적용하는 데 있어 기존의 설비 및 프로세스 외에 추가 작업이 필요 없도록 하였다.

2.2 제조 데이터의 규격 속성 정의

1) 데이터 수집 방법
- 제조 분야 : 분무건조공법을 이용한 분말유크림 제조
- 제조 공정명 : 용해혼합
- 수집장비 : PLC(설비데이터) 및 DBMS(품질데이터)
- 수집기간 : 2020년 3월 4일 ~ 2020년 4월 30일 (약 2개월)
- 수집주기 : 사이클타임 약 6초

2) 데이터 유형/구조

- 데이터 크기, 데이터 수량 : 7개 칼럼, 835,200개의 관측치
- 데이터셋은 데이터 인덱스(NUM), 데이터 수집일시(STD_DT), 용해 온도(MELT_TEMP), 용해 교반속도(MOTORSPEED), 용해탱크 내용량(MELT_ WEIGHT), 수분함유량(INSP), 불량여부(TAG) 등 7개 칼럼으로 이뤄져 있으며, 인덱스, 수집일시 및 불량여부 제외 모두 연속형 수치 형태이다.
- 교반속도(MOTORSPEED)가 0인 경우는 공정완료, 원료 추가 투입, 설비이상, 작업자 휴식 등의 다양한 이유로 설비를 중지/정지한 경우이며, 내용량(MELT_WEIGHT)이 0인 경우는 용해탱크에 아직 원료가 투입되기 전이거나, 공정 완료 후 용액이 다음 공정으로 넘어가서 탱크가 빈 경우이다. 불량여부(TAG) 값 OK는 ‘양품’, NG는 ‘불량’을 의미한다. 또한, 용해온도와 교반속도 데이터는 소수점 1자리가 생략되어 있기 때문에 값 nnn은 실제로 nn.n을 의미한다(예: 501 → 50.1℃)

3) 독립변수와 종속변수 정의

구분	명칭	비고
독립변수	용해온도	용해온도의 차분 및 2차 차분도 함께 사용
	교반속도	교반속도의 차분 및 2차 차분도 함께 사용
	내용량	내용량의 이상치를 보간하고, 내용량의 차분 및 2차 차분도 함께 사용
	수분함유량	수분함유량을 표준화한 값을 사용
종속변수	양품/불량	양품을 1, 불량을 0으로 인코딩하여 사용

- 독립변수란 다른 변수에 영향을 받지 않는 변수로, 입력값이나 원인을 나타낸다. 종속변수란 독립변수의 변화에 따라 어떻게 변하는지를 알고자 하는 변수를 말하며, 결과물이나 효과를 나타낸다.
- 상기 독립변수 및 종속변수 정의에 의하면 공정관점에서 다른

변수에 영향을 받지 않는 변수인 용해온도, 교반속도, 내용량이 독립변수이고, 그 외 변수는 종속변수가 될 것이나, 본 보고서에서 수행하고자 하는 AI 알고리즘은 각 변수의 개별 영향이 아닌 전체적 영향을 분석하고자 하므로, 주어진 상기 독립변수에 추가하여 표와 같이 전처리한 독립변수와 종속변수를 설정한다. 따라서, 설비운영변수(속도, 온도, 내용량) 및 품질결과값(수분함유량)이 독립변수, 최종판정값(양품/불량)이 종속변수가 된다.

2.3 제조 데이터의 전처리[EDA] 방법

1) 라이브러리/데이터 불러오기

1)-1. 기본적인 라이브러리 불러오기

- 분석을 시작하기 위해서 분석에 사용할 라이브러리들을 불러오는 과정이 필요하다. 코드길이의 단축화를 위해 import package as A 약어형태의 import가 있다.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

[그림 1] 기본적인 라이브러리 불러오기 코드예시

1)-2. GRU 모델을 위한 라이브러리 불러오기

- 위의 라이브러리들은 Python으로 분석을 할 때 기본적으로 많이 활용하는 라이브러리들이다. 해당 분석에서는 GRU 모델을 활용할 것이므로 해당 모델을 구축하기 위한 라이브러리들이 추가로 필요하다.

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix
    , precision_score
    , recall_score
    , f1_score
    , accuracy_score
)

from keras.models import Sequential
from keras.optimizers import Nadam
from keras.layers import Dense, GRU
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

[그림 2] GRU 모델 구축을 위한 라이브러리 불러오기 코드

- 라이브러리 설명은 아래와 같다.
  - Pandas는 데이터 분석 시 가장 많이 쓰는 패키지로, 표 데이터를 다루기 위한 시리즈(Series) 클래스와 데이터프레임(DataFrame) 클래스를 제공한다.
  - numpy는 과학 계산을 위한 라이브러리로 다차원 배열을 처리하는데 여러 유용한 기능을 제공한다.
  - matplotlib는 자료를 차트(chart)나 플롯(plot)으로 시각화(visualization)하는 패키지이다. 코드에서 matplotlib.pyplot은 matplotlib에서 지원하는 모듈 중 하나인 pyplot에 접근하여 불러온다는 것을 의미한다. pyplot의 인터페이스는 겉으로 드러나지 않으면서 자동으로 figure와 axes를 생성하며, 정의된 플롯을 얻을 수 있도록 만들어준다.
  - %matplotlib inline은 플롯 결과를 notebook문서에 저장하여 notebook을 실행한 브라우저에서 바로 그림을 볼 수 있게 해주는 기능으로 Jupyter Notebook에서 사용한다. 해당 선언을 해주면 다음번에 열었을 때 코드를 실행하지 않고 그래프가 저장되어 유용하다.
  - sklearn은 데이터 과학 연산의 핵심 패키지로 다음을 지원한다.

데이터 전처리: preprocessing, train_test_split
지도 및 비지도 학습: 사용안함
모델 선택: 사용안함
검증 및 오차: confusion_matrix, precision_score,
recall_score, f1_score, accuracy_score

※ negative positive value(npv)는 지원하지 않아 직접 계산하였다.

○ keras은 파이썬 기반 딥러닝 라이브러리로, 처음 딥러닝을 시작하는 사용자에게 편리하고 직관적인 API이며 다음을 지원한다.

모델 레이어 구성: Sequential, Dense, GRU
학습 옵티마이저: Nadam
학습 중 콜백: EarlyStopping, ModelCheckpoint

- 만약 에러가 나는 경우는 해당 라이브러리 설치가 되어있지 않아서 발생하는 것이기 때문에 에러가 나는 라이브러리를 아래와 같이 !pip install을 통해 설치해야 한다.

예시) !pip install pandas

2) 분석 데이터 불러오기

```
DATA_PATH = "../data/data.csv"
df = pd.read_csv(DATA_PATH)
```

[그림 3] 분석 데이터 불러오기 코드

- 데이터를 불러올 때, 데이터가 저장되어 있는 위치가 동일하다면 바로 불러오기를 해도 되지만 다른 폴더에 저장되어 있다면 DATA_PATH를 지정해주어야 한다. 또다른 파일경로 지정은 직접 read_csv()에 경로를 입력하는 방법이 있다.
- 만약 위치를 상대경로로 하고싶은 경우에는 os라이브러리를 사용하면 된다. 여기서 주의할 점은 경로를 나타낼 때 구분자로 / 혹은 \\로 표시해야 한다는 것이다. 하지만 linux 기반의 운영체제(mac, centos 등)과 윈도우 기반 운영체제 모두 /을 지원하기 때문에 /을 사용하는 편이 좋다.
- csv 파일을 불러올 때는 pandas의 read_csv("파일경로") 명령어를 활용한다. 불러올 때 encoding이 제대로 되지 않는다면

오류가 발생할 수 있으므로 지정해주어야 하지만 기본 encoding이 한글을 지원하는 utf-8으로 설정되어있으므로 생략해도 무방하다.

3) 데이터 기본 구조 파악

```
df.head(11)
```

	STD_DT	NUM	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP	TAG
0	2020-03-04 0:00	0	489	116	631	3.19	OK
1	2020-03-04 0:00	1	433	78	609	3.19	OK
2	2020-03-04 0:00	2	464	154	608	3.19	OK
3	2020-03-04 0:00	3	379	212	606	3.19	OK
4	2020-03-04 0:00	4	798	1736	604	3.21	OK
5	2020-03-04 0:00	5	743	1722	603	3.21	OK
6	2020-03-04 0:00	6	390	212	602	3.19	OK
7	2020-03-04 0:00	7	493	152	600	3.19	OK
8	2020-03-04 0:00	8	427	0	599	3.19	OK
9	2020-03-04 0:00	9	489	148	598	3.20	OK
10	2020-03-04 0:01	10	507	128	596	3.19	OK

[그림 4] 데이터의 첫 11행 확인코드 및 결과

- 위의 결과를 보면 알 수 있듯이 똑같은 시간에 10개의 행이 존재하며 11번째 행은 이전 10번째 열에 1분정도 경과한 것을 볼 수 있다. 즉 한 행은 60초 / 10개의 행 즉, 6초로 하나의 데이터(행)은 6초간격으로 측정된 것을 알 수 있다.
  또한 NUM변수는 0부터 행이 증가함에 따라 1씩 증가하는 것을 볼 수 있다.
- df.info()를 활용하여 데이터의 기본적인 구조를 파악할 수 있다. 해당 함수는 관측치 수, 각 칼럼의 이름 및 특성, 데이터 타입 등을 결과로 반환한다.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 835200 entries, 0 to 835199
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   STD_DT       835200 non-null  object
 1   NUM          835200 non-null  int64
 2   MELT_TEMP    835200 non-null  int64
 3   MOTORSPEED   835200 non-null  int64
 4   MELT_WEIGHT  835200 non-null  int64
 5   INSP         835200 non-null  float64
 6   TAG          835200 non-null  object
dtypes: float64(1), int64(4), object(2)
memory usage: 44.6+ MB
```

[그림 5] 데이터 기본 구조 확인 결과

- 위와 같이 데이터는 835,200개의 관측치, 7개의 칼럼(변수)로 구성되어있으며, 각 칼럼에는 null값(결측치)이 존재하지 않는 것을 알 수 있다.
- 데이터의 구조를 확인한 후 df.describe() 함수를 활용하여 수치형 데이터들의 요약통계량을 확인한다. 해당 함수를 활용하면 아래와 같이 데이터 개수, 평균, 표준편차, 최소값, 최대값 등을 쉽게 확인할 수 있다. NUM은 시간이 흐름에 따라 증가하는 값이므로 분석에서 제외하며 MELT_TEMP, INSP, MOTORSPEED, MELT_WEIGHT 변수의 통계치를 확인한다. 상대적으로 변화가 작은 변수는 INSP이며 MELT_TEMP, OTORSPEED, MELT_WEIGHT은 최소, 최대값의편차가 크다.

```
df.describe()
```

	NUM	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP
count	835200.000000	835200.000000	835200.000000	835200.000000	835200.000000
mean	417599.500000	509.200623	459.782865	582.962125	3.194853
std	241101.616751	128.277519	639.436413	1217.604433	0.011822
min	0.000000	308.000000	0.000000	0.000000	3.170000
25%	208799.750000	430.000000	119.000000	186.000000	3.190000
50%	417599.500000	469.000000	168.000000	383.000000	3.190000
75%	626399.250000	502.000000	218.000000	583.000000	3.200000
max	835199.000000	832.000000	1804.000000	55252.000000	3.230000

[그림 6] 데이터의 요약통계량 확인 결과

4) STD_DT 변수의 타입 변환

```
df['STD_DT'] = pd.to_datetime(df['STD_DT'], format="%Y-%m-%d %H:%M:%S")
df['STD_DT_SEC'] = df['STD_DT'] + pd.to_timedelta((df['NUM']%10).astype(int) * 6, unit='s')
```

[그림 7] 시간정보(datetime) 형태 변환을 위한 코드

- 위의 data.info()의 결과에서 STD_DT 변수의 타입이 Object 이므로 datetime 형태로 변경하여 인덱스로 지정해주면 분석에 용이하게 활용할 수 있다. 변환을 위하여 pd.to_datetime을 사용하였으며 시,분,초까지 설정하여 변환한다. 그 후, 시간이 증가함에 따라 같이 증가하는 NUM변수를 이용해 초를 설정해준뒤 STD_DT_SEC컬럼에 저장해준다. 위에도 언급한대로 6초간격의 행을 다음의 공식에 의해 변경시킨다.

$$NUM \mod 10 * 6\ sec$$

[그림 8] NUM을 이용한 초계산

- 변형 후의 값은 각 행의 데이터의 측정 시작시간을 나타낸다. 추가적으로 각 loop를 사용하는 apply()대신 계산할 데이터를 백터화(vectorization)하여 속도를 높혔다.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 835200 entries, 0 to 835199
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   STD_DT       835200 non-null  datetime64[ns]
 1   NUM          835200 non-null  int64
 2   MELT_TEMP    835200 non-null  int64
 3   MOTORSPEED   835200 non-null  int64
 4   MELT_WEIGHT  835200 non-null  int64
 5   INSP         835200 non-null  float64
 6   TAG          835200 non-null  object
 7   STD_DT_SEC   835200 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(1), int64(4), object(1)
memory usage: 51.0+ MB
```

[그림9] 시간정보(datetime) 형태 변환 결과

- STD_DT, STD_DT_SEC 둘다 datetime64[ns]로 변경된 것을 볼 수 있다. 데이터 타입은 python에서 지원하는 datetime.datetime과 똑같다.

```
df.set_index(['STD_DT_SEC'], drop=True, inplace=True)
```

[그림 10] 특정 칼럼 인덱스 지정을 위한 코드

- DataFrame.set_index([“칼럼”]) 함수를 활용하여 해당 칼럼을 인덱스로 지정할 수 있다. inplace=True는 원본값에 연산한 값을 대체하라는 설정이다. pandas의 DataFrame, Series의 연산 후의 원본값은 그대로이며, 연산한 값은 새로 생성된 값이 되므로 원본에 연산한 값을 대체하라는 파라미터 설정이다. 만약 위의 파라미터를 설정하지 않았다면 다음의 코드로 대체가능하다.

```
df = df.set_index(['STD_DT_SEC'], drop=True)
```

[그림 11] 대체 코드

- drop=True를 통해 인덱스의 값이 칼럼값으로 변하는 것을 방지한다. 즉 기존 인덱스의 0,1,2 등의 값이 새로운 컬럼으로 변하는 현상을 방지해주는 파라미터이다.

5) Histogram 및 Plot 그리기

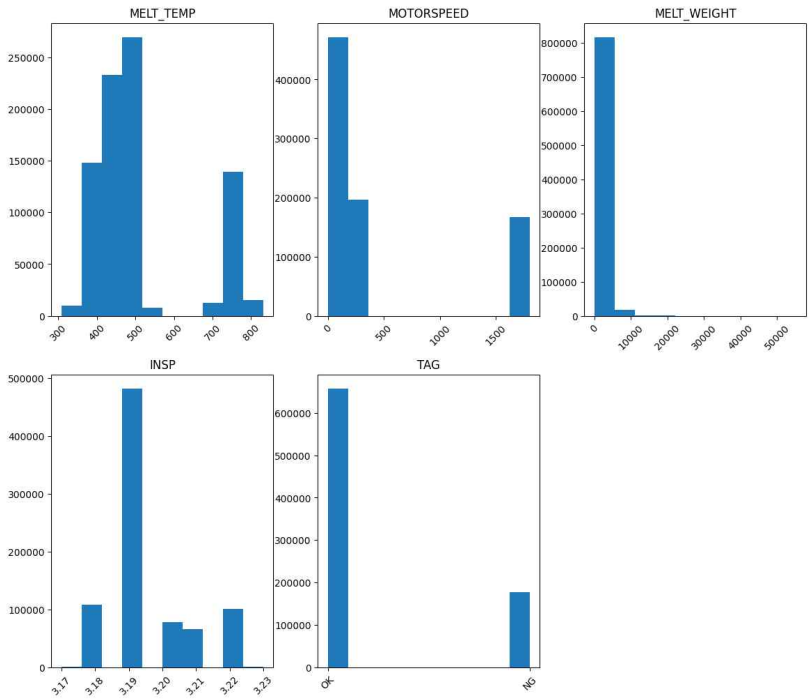
```
col_name = ['MELT_TEMP','MOTORSPEED','MELT_WEIGHT','INSP','TAG']
```

[그림 12] 칼럼명 지정을 위한 코드

- 관측치의 구간별 빈도수를 확인하기 위해서 각 변수의 histogram을 그린다. 변수가 여러 개이기 때문에 그래프를 모두 그리면 비효율적이고 가시성도 떨어진다. 따라서 반복문을 활용하여 여러 그래프를 한 번에 그릴 것이다. col_name은 예측에 활용할 변수들의 이름이 들어간 리스트이다. 아래의 코드를 사용하여 각 데이터의 histogram을 그렸다.

```
plt.figure(figsize=(14,12)) # 14(넓이) X 12(높이)로 설정
for i in range(len(col_name)): # 231 ~ 235 까지의 그래프 내부의 서브플롯번호 탐색
    num = 231 + i # 서브플롯번호 생성
    plt.subplot(num) # 서브플롯번호 생성 및 지정
    plt.hist(df[col_name[i]]) # 각 변수의 값을 그래프에 입력
    plt.xticks(rotation=45) # x축 눈금의 표시를 45도 기울이기
    plt.title(col_name[i]) # 그래프의 이름을 변수이름으로 지정
plt.show() # 그래프를 화면에 표시
```

[그림 13] 반복문 활용을 통해 다수개의 히스토그램을 한 번에 그리기 위한 코드

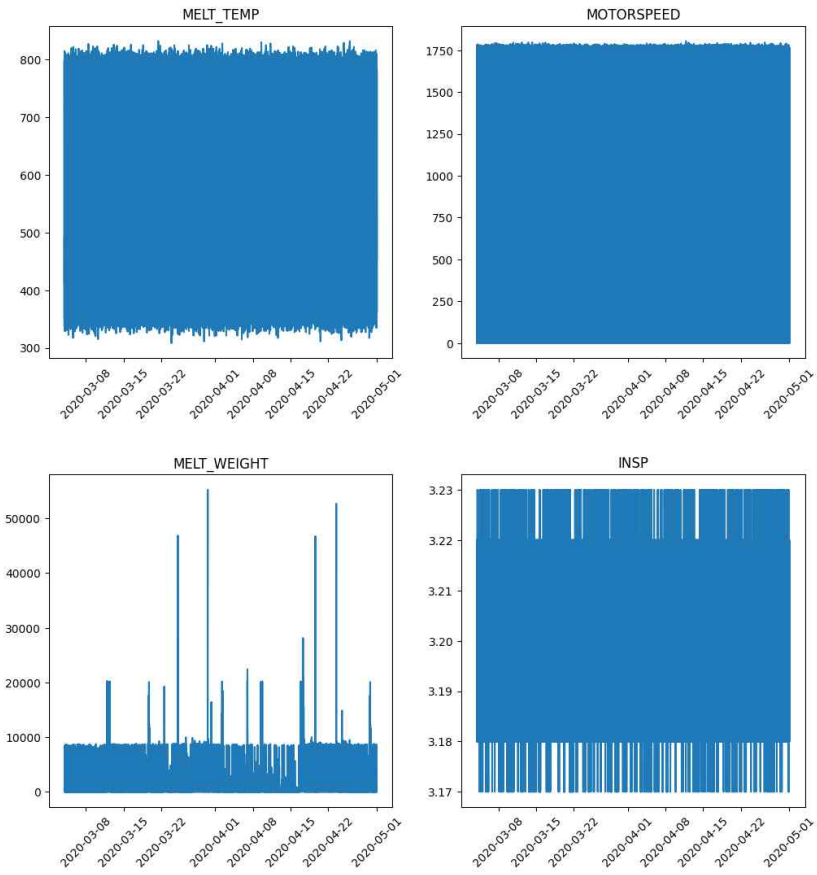


[그림 14] 히스토그램 그리기 결과

- 이제 관측치의 패턴을 확인하기 위하여 plt.plot() 함수를 활용하여 plot을 그린다.

```
plt.figure(figsize=(12,12))
for i in range(len(col_name)):
    num = 221 + i
    plt.subplot(num)
    plt.plot(df[col_name[i]])
    plt.xticks(rotation=45)
    plt.title(col_name[i])
plt.subplots_adjust(
    left=0.125
    ,bottom=0.1
    ,right=0.9
    ,top=0.9
    ,wspace=0.2
    ,hspace=0.35) #subplot들의 간격을 조정
plt.show()
```

[그림 15] 각 변수별의 전체데이터를 그리기위한 코드

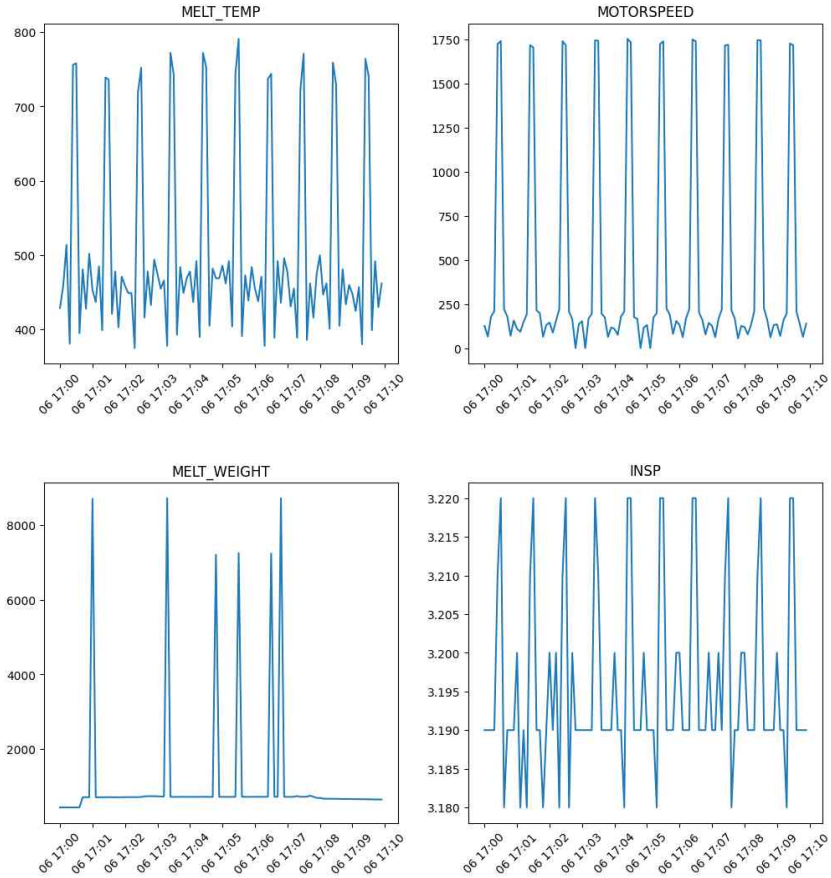


[그림 16] 전체 데이터 결과

- 소스코드 부분은 새로이 추가 및 변경된 부분만 주석을 추가하였다. 그래프를 그린 결과 관측값이 너무 많아 그래프의 패턴을 살펴보기 어렵다. 하지만 그래프의 상한선과 하한선은 확인 가능하다. ‘MELT_WEIGHT‘ 변수에 이상치가 많이 있음을 확인하였다.

```
plt.figure(figsize=(12,12))
for i in range(len(col_name)):
    num = 221 + i
    plt.subplot(num)
    plt.plot(df[col_name[i]][39000:39100])
    plt.xticks(rotation=45)
    plt.title(col_name[i])
plt.subplots_adjust(left=0.125,bottom=0.1,right=0.9,top=0.9,wspace=0.2,hspace=0.35)
plt.show()
```

[그림 17] 10분간의 각 변수별의 값변동그래프를 그리기 위한 코드



[그림 18] 10분간의 각 변수별의 값변동에 관한 그래프

- 해당 데이터에서는 각 분(10개의 관측치)마다 패턴을 보이며 MELT_WEIGHT의 경우에는 몇몇 이상치를 볼 수 있다. 이 패턴을패턴을 활용할 예정이며 MELT_WEIGHT는 이상치 제거 및 보간을 할 예정이다.

- MELT_WEIGHT 칼럼의 데이터값을 확인하여 보았을 때, 이상치가 하나의 인덱스 또는 두 개의 인덱스 사이(6초~12초)안에 10배이상의 굉장히 큰 값이 나왔다가 정상 범주의 데이터 값으로 돌아가는 것을 확인 할 수 있다. 해당 MELT_WEIGHT의 수치가 기록된 수치처럼 급변하기는 어렵다고 판단되며, 이는

연속적인 실측치가 기록되었다고 생각되기보다 PLC를 통해 입력된 값이 겹쳐서 기록 또는 오입력된 것으로 추정된다. 이에 각 값들을 해당 인덱스의 위아래의 MELT_WEIGHT값들을 통해 보간 하여 주었다.

```
df_pre = df.copy()

# MELT_WEIGHT 이상치 보간
df_pre["MELT_WEIGHT_DIFF"] = df_pre["MELT_WEIGHT"].diff()
df_pre.loc[df_pre["MELT_WEIGHT_DIFF"].abs() > 80,"MELT_WEIGHT"] = np.NaN
df_pre["MELT_WEIGHT_PRE_P"] = df_pre["MELT_WEIGHT"].interpolate().astype(int)

# MELT_WEIGHT_PRE_P 의 이상치 보간
df_pre["MELT_WEIGHT_PRE_P_FOR2"] = df_pre["MELT_WEIGHT_PRE_P"]
df_pre["MELT_WEIGHT_PRE_DIFF"] = df_pre["MELT_WEIGHT_PRE_P_FOR2"].diff()
df_pre.loc[df_pre["MELT_WEIGHT_PRE_DIFF"].abs() > 80,"MELT_WEIGHT_PRE_P_FOR2"] = np.NaN
df_pre["MELT_WEIGHT_PRE_P2"] = df_pre["MELT_WEIGHT_PRE_P_FOR2"].interpolate().astype(int)
```

[그림19] MELT_WEIGHT 이상치 보간 코드

- 위의 MELT_WEIGHT 이상치 보간 코드를 통하여 MELT_WEIGHT의 불특정한 이상치를 위아래 값들을 기준으로 보간하여 준다.

```
# TAG binary
df_pre["TAG_BINARY"] = 1
df_pre.loc[df_pre["TAG"] == "NG","TAG_BINARY"] = 0

# INSP 정규화
df_pre["S_INSP"] = (df_pre["INSP"] - df_pre["INSP"].mean()) / df_pre["INSP"].std()
```

[그림20] TAG 숫자인코딩, INSP 정규화(표준화) 코드

- 종속변수인 ‘TAG’가 범주형(OK, NG) 형태로 되어있기 때문에 모형에 넣기 위하여 해당 변수를 숫자 형태로 인코딩해 주어야 한다. 인코딩에는 위의 코드를 사용하여 ‘OK’는 1로, ‘NG’는 0의 INT 형태로 변경
- INSP의 수분함유량은 그 값들 간의 편차가 데이터값 자체에 비하여 매우 작다. 이에 정규화 방법으로 위의 코드를 사용하여 표준화하여 그 값을 사용하였다.

```
first_idx = df_pre.head(1).index
# MOTOR_SPEED , MELT_WEIGHT, MELT_TEMP가 단위구간당 변화량을 구하는 컬럼 생성
df_pre["MOTORSPEED_DIFF"] = df_pre["MOTORSPEED"].diff()
df_pre.loc[first_idx,"MOTORSPEED_DIFF"] = 0
df_pre["MELT_WEIGHT_DIFF"] = df_pre["MELT_WEIGHT_PRE_P"].diff()
df_pre.loc[first_idx,"MELT_WEIGHT_DIFF"] = 0
df_pre["MELT_TEMP_DIFF"] = df_pre["MELT_TEMP"].diff()
df_pre.loc[first_idx,"MELT_TEMP_DIFF"] = 0

# MOTOR_SPEED , MELT_WEIGHT, MELT_TEMP가 단위구간당 변화량의 변화량 구하는 컬럼 생성
df_pre["MOTORSPEED_DIFF2"] = df_pre["MOTORSPEED_DIFF"].diff()
df_pre.loc[first_idx,"MOTORSPEED_DIFF2"] = 0
df_pre["MELT_WEIGHT_DIFF2"] = df_pre["MELT_WEIGHT_DIFF"].diff()
df_pre.loc[first_idx,"MELT_WEIGHT_DIFF2"] = 0
df_pre["MELT_TEMP_DIFF2"] = df_pre["MELT_TEMP_DIFF"].diff()
df_pre.loc[first_idx,"MELT_TEMP_DIFF2"] = 0
```

[그림21] MOTOR_SPEED , MELT_WEIGHT, MELT_TEMP가 단위구간당 변화량(차분) 및 변화량의 변화량(2차 차분)을 구하는 코드

- MOTOR_SPEED , MELT_WEIGHT, MELT_TEMP의 경우 그 변화량도 제품의 품질에 영향을 미칠 수 있다고 판단하여 데이터를 전처리하여 그 차분(변화량)과 2차 차분(변화량의 변화량)을 구하여 분석에 사용하였다. 위의 코드를 사용하여 각각의 값들을 구하였다.

6) 상관분석

```
corr = df_pre.corr(method = 'pearson')
corr
```

[그림 22] 상관분석 수행 코드

- 종속변수인 TAG와 각 독립변수 간의 상관관계를 살펴보기 위하여 corr() 함수를 활용하여 상관분석을 진행한다. 상관분석은 연속형 변수로 측정된 변수 간의 선형관계를 분석하는 기법이다.

	NUM	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP	MELT_WEIGHT_DIFF	MELT_WEIGHT_PRE_P	MELT_WEIGH
NUM	1.000000e+00	0.000188	-0.000050	0.048692	-0.000524	5.200566e-07	0.069138	
MELT_TEMP	1.884495e-04	1.000000	0.944929	-0.000623	0.916295	-5.815489e-04	0.000189	
MOTORSPEED	-5.035526e-05	0.944929	1.000000	-0.000373	0.887813	-1.594435e-04	0.000714	
MELT_WEIGHT	4.869156e-02	-0.000623	-0.000373	1.000000	-0.000690	4.240329e-01	1.000000	
INSP	-5.236376e-04	0.916295	0.887813	-0.000690	1.000000	-8.296464e-05	0.000111	
MELT_WEIGHT_DIFF	5.200566e-07	-0.000582	-0.000159	0.424033	-0.000083	1.000000e+00	0.226479	
MELT_WEIGHT_PRE_P	6.913768e-02	0.000189	0.000714	1.000000	0.000111	2.264793e-01	1.000000	
MELT_WEIGHT_PRE_P_FOR2	5.801000e-02	-0.000218	-0.000042	1.000000	-0.000384	-1.694350e-02	1.000000	
MELT_WEIGHT_PRE_DIFF	5.200608e-07	-0.000582	-0.000159	0.424033	-0.000083	1.000000e+00	0.226479	
MELT_WEIGHT_PRE_P2	6.228884e-02	0.000083	0.000214	0.794957	-0.000020	2.662950e-04	0.723217	
TAG_BINARY	8.669780e-02	0.310586	0.264693	-0.021556	0.272580	8.078499e-04	-0.010335	
S_INSP	-5.236376e-04	0.916295	0.887813	-0.000690	1.000000	-8.296464e-05	0.000111	
MOTORSPEED_DIFF	-2.051652e-06	0.635701	0.536329	-0.001138	0.593174	-3.793321e-04	-0.000104	
MELT_TEMP_DIFF	-9.246249e-07	0.667921	0.510122	-0.000905	0.605363	-5.000473e-04	-0.000029	
MOTORSPEED_DIFF2	2.704290e-07	0.110281	-0.017975	-0.000655	0.096200	-3.826332e-04	-0.000050	
MELT_WEIGHT_DIFF2	-1.525095e-08	0.000215	0.000121	-0.023946	0.000717	5.691070e-01	-0.140174	
MELT_TEMP_DIFF2	-1.066162e-07	0.220837	0.077515	-0.000534	0.188491	-9.222164e-05	0.000101	

MELT_WEIGHT_PRE_P_FOR2	MELT_WEIGHT_PRE_DIFF	MELT_WEIGHT_PRE_P2	TAG_BINARY	S_INSP	MOTORSPEED_DIFF	MELT_TEMP_DIFF
0.058010	5.200608e-07	0.062289	0.086698	-0.000524	-0.000002	-9.246249e-07
-0.000218	-5.815489e-04	0.000083	0.310586	0.916295	0.635701	6.679208e-01
-0.000042	-1.594436e-04	0.000214	0.264693	0.887813	0.536329	5.101220e-01
1.000000	4.240329e-01	0.794957	-0.021556	-0.000690	-0.001138	-9.048807e-04
-0.000384	-8.296464e-05	-0.000020	0.272580	1.000000	0.593174	6.053625e-01
-0.016944	1.000000e+00	0.000266	0.000808	-0.000083	-0.000379	-5.000473e-04
1.000000	2.264793e-01	0.723217	-0.010335	0.000111	-0.000104	-2.916132e-05
1.000000	-1.694353e-02	1.000000	-0.028775	-0.000384	-0.000291	-3.308758e-04
-0.016944	1.000000e+00	0.000266	0.000808	-0.000083	-0.000379	-5.000473e-04
1.000000	2.662951e-04	1.000000	-0.024762	-0.000020	-0.000334	-2.276328e-04
-0.028775	8.078501e-04	-0.024762	1.000000	0.272580	0.200727	2.408307e-01
-0.000384	-8.296464e-05	-0.000020	0.272580	1.000000	0.593174	6.053625e-01
-0.000291	-3.793321e-04	-0.000334	0.200727	0.593174	1.000000	9.514483e-01
-0.000331	-5.000473e-04	-0.000228	0.240831	0.605363	0.951448	1.000000e+00
-0.000080	-3.826332e-04	-0.000175	0.082950	0.096200	0.698778	7.633506e-01
-0.035295	5.691070e-01	-0.034576	0.000360	0.000717	-0.000220	-2.780187e-04
-0.000156	-9.222164e-05	-0.000107	0.137821	0.188491	0.670976	7.945800e-01

MOTORSPEED_DIFF2	MELT_WEIGHT_DIFF2	MELT_TEMP_DIFF2
2.704290e-07	-1.525095e-08	-1.066162e-07
1.102813e-01	2.154852e-04	2.208370e-01
-1.797538e-02	1.214280e-04	7.751523e-02
-6.551682e-04	-2.394580e-02	-5.341572e-04
9.620043e-02	7.171032e-04	1.884905e-01
-3.826332e-04	5.691070e-01	-9.222164e-05
-5.041085e-05	-1.401737e-01	1.007712e-04
-7.983882e-05	-3.529517e-02	-1.559940e-04
-3.826332e-04	5.691070e-01	-9.222164e-05
-1.751741e-04	-3.457595e-02	-1.068291e-04
8.295005e-02	3.602304e-04	1.378213e-01
9.620043e-02	7.171032e-04	1.884905e-01
6.987776e-01	-2.200710e-04	6.709761e-01
7.633506e-01	-2.780187e-04	7.945800e-01
1.000000e+00	-4.936373e-04	9.604555e-01
-4.936373e-04	1.000000e+00	-2.559706e-04
9.604555e-01	-2.559706e-04	1.000000e+00

[그림 23~25] 상관분석 수행 결과

- 상관 행렬의 결과 'MELT_WEIGHT' 및 차분화한 변수는 종속변수와의 상관관계가 낮은 것으로 나타났다. 하지만 기본적인 머신러닝의 학습 알고리즘상 단일 독립변수가 종속변수에 영향을 끼치지 못하더라도 여러 독립변수간의 상호작용으로 인해 종속변수에 영향을 미칠 수가 있기 때문에 해당 변수를 머신러닝 분석을 위한 데이터로 사용하였다.

7) MinMax 정규화 및 변수분리

```
#정규화
scaler = preprocessing.MinMaxScaler()

scale_sc = scaler.fit_transform(df1)

X_values = scale_sc[:,:-1]
y_values = scale_sc[:,-1]
```

[그림 26] 정규화 및 변수분리코드

- 데이터의 scale을 맞추면 안정된 데이터가 나올 수 있기 때문에 정규화를 해주어야 한다. sklearn 라이브러리에 있는 preprocessing.MinMaxScaler() 함수를 이용하여 분포

추정 및 변환해준다. 그 후 독립변수(feature)와 종속변수(label)를 분리해준다.

8) Window의 정의

```
WINDOW_SIZE = 11

x_feature = np.array([
    X_values[i:i+WINDOW_SIZE]
    for i in np.arange(len(X_values) - WINDOW_SIZE)
])
y_label = np.array([
    np.array([y_values[i+WINDOW_SIZE-1]])
    for i in np.arange(len(y_values) - WINDOW_SIZE)
])
```

[그림 27] window별로 데이터 생성

- 윈도우 사이즈는 위에서 언급했던 주기인 10개에 다음 예측할 구간의 데이터의 독립변수까지를 하나의 윈도우 범위로 간주하여 11로 설정하였으며 예측할 구간은 11번째의 데이터가 OK/NG인지를 확인하는 로직으로 만들었다. 윈도우는 6초의 구간마다 이동하며 설정하며 각 윈도우로 나눈 구간은 독립변수(feature)와 종속변수(label)의 리스트에 담긴다.

```
(
    train_feature
    , test_feature
    , train_label
    , test_label
) = train_test_split(x_feature, y_label, random_state=42, test_size=0.3, stratify=y_label)
(
    x_train
    , x_valid
    , y_train
    , y_valid
) = train_test_split(train_feature, train_label, random_state=42, test_size=0.3, stratify=train_label)
```

[그림28] 학습데이터(train)와 검증데이터(validation) 예측데이터(test)분리 코드

- 학습데이터(train)와 검증데이터(validation), 예측데이터(test) 분리
- 전체 데이터에서 종속변수인 ‘TAG’에 클래스 OK/NG의 비율을 확인해 보았을 때, OK: 658133, NG: 177067 비율이 약 3.7:1로 학습하기에 불균형이 매우 심하지는 않으며, 오히려 이정도의 불균형을 해결하기 위하여 ‘SMOTE’를 통하여 데이터를 생성하게 되면, 이미 ‘SMOTE’의 생성원리

인 K-neighbors classifier로 클러스터화(분류)된 데이터가 추가되어 오히려 학습 및 예측을 하는데 불필요한 노이즈를 주게 될 것이다.

- 또한, 만약 위의 데이터를 시간순으로 학습데이터(시작 데이터에서부터 전체 데이터의 0.7비율 부분까지)와 예측데이터(전체 데이터에서 0.7부분에서 마지막 데이터까지)를 나누게 되면 전체 데이터의 마지막 30% 부분인 테스트 데이터에서 심각한 불균형(OK:NG 비율이 약 100:1)이 나타나게 된다. 이런 경우 모델이 모두 OK로 예측하더라도 accuracy가 99% 가까이 나타나게 되는 문제가 생긴다.
- 따라서 전체 데이터의 비율을 유지하면서, 무작위로 섞어준 데이터로 학습 및 검증 테스트 데이터를 분리하였다. 학습 및 테스트할 데이터는 위의 코드, skit learn의 stratify를 사용하여 같은 비율을 유지한 채로 무작위로 나누어 준 후 학습 및 테스트에 사용하였다.

3. 분석 모델 개발

3.1 AI 분석 방법론(알고리즘) 구축 절차

○ 본 공정의 용해탱크 데이터를 활용한 품질예측 분석모델 구축 절차는 다음과 같다.

○ 첫째, 데이터 요약 통계량, 그래프 등을 활용하여 데이터 탐색을 진행하고, 분석에 활용하기 위하여 데이터를 전처리(정제)한다.

○ 둘째, 모델링을 위하여 전체 데이터를 훈련 데이터(train data set)와 테스트 데이터 (test data set)로 분리한다. 학습 및 테스트할 데이터를 skit learn의 stratify를 사용하여 전체 데이터셋의 양품과 불량 간 비율과 같은 비율을 유지한 채로 무작위로 나누어 학습 및 테스트를 수행한다.

○ 셋째, 학습 데이터셋에 MinMaxScaler를 이용하여 정규화(Scaling)를 실시한다.

MinMaxScaler()는 데이터의 최대값이 1, 최소값이 0이 되도록 변환한다.
딥러닝 모델을 통한 학습 시, 데이터의 scale을 맞추면 가중치(weight)의 scale도
일관성 있게 나올 수 있다. 사용 방법은 다음의 3가지와 같다.
① 훈련 데이터의 분포 추정: 훈련 데이터를 입력으로 하여 fit 함수를 실행하여
분포 모수를 객체 내에 저장,
② 훈련 데이터의 변환: 훈련 데이터를 입력으로 하여 transform 함수를 실행
하여 훈련 데이터를 변환,
③ 테스트 데이터의 변환: 테스트 데이터를 입력으로 하여 transform 함수를 실행
하여 테스트 데이터를 변환,
훈련 데이터는 ①, ②를 합쳐서 fit_transform 함수를 사용하고, 테스트 데이터는
transform 함수를 사용하여 스케일링을 진행한다.

- 넷째, window를 생성한다. timestep만큼 window_size를 지정해 주면 되며,
  window가 11개라면 과거 시간 데이터 11개를 사용하여 해당 시간 데이터에서 나타날 값을 예측하게 된다.
- 다섯째, GRU 모델을 구축하고, 검증 데이터로 모델이 잘 만들어졌는지 확인, 테스트
  데이터로 실제 예측을 진행한다. 예측 결과는 혼동 행렬(Confusion Matrix)을 활용
  하여 모델의 분류 성능을 평가한다.

3.2 적용하고자하는 AI[GRU]선정 이유와 구체적 소개

1) GRU 분석모델

- 해당 AI 방법론(알고리즘) 선정 이유

본 공정의 용해탱크 데이터의 경우 시간순으로 기록된 시계열

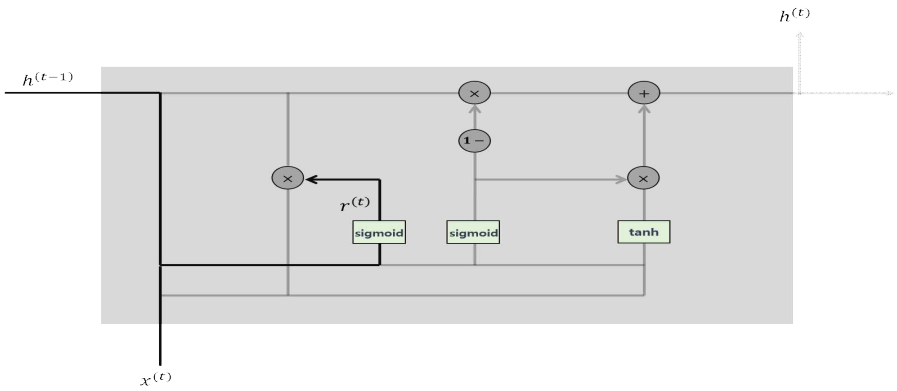
데이터이며 해당 공정 중에서 시간순으로 선행된 데이터가 뒤따라오는 데이터에 영향을 줄 수 있으므로 일련의 시계열 데이터로부터 결과값을 도출 할 수 있는 모델인 RNN기반 AI 분석모델을 선정 하였다.
RNN 기반 모델 중 기존의 장기의존성 문제(은닉층의 과거의 정보가 마지막까지 전달되지 못하는 현상)와 분석속도가 느린 부분을 획기적으로 개선한 GRU모델을 사용하였다. 분석예측 향상과 빠른 분석속도로 현장에서 빠르게 적용할 수 있을 것을 기대한다.

2) 적용하고자 하는 AI 분석 방법론(알고리즘)의 구체적 소개

- GRU(Gated Recurrent Unit)

LSTM은 RNN의 치명적인 한계점이었던, 장기의존성 문제를 해결하면서 긴 길이의 입력 데이터에서도 좋은 성능을 내는 모델이다, 하지만 복잡한 구조 때문에 RNN에 비하여 파라미터가 많이 필요하여 분석속도가 느리다. 또한, 파라미터가 많아지는 것에 비해 데이터가 충분하지 않은 경우, 과적합이 발생하게 된다. 이러한 단점을 개선하기 위하여 LSTM의 변형인 GRU가 등장하게 되었다.

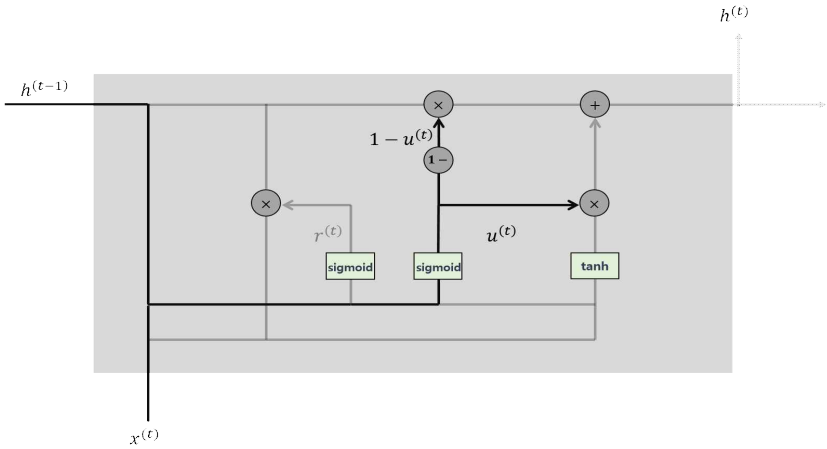
- GRU의 기본 구조



[그림29] GRU 개요

1. Reset Gate

Reset Gate는 과거의 정보를 적당히 리셋 시키는게 목적으로 sigmoid 함수를 출력으로 이용해 (0 ,1)(0 ,1) 값을 이전 은닉층에 곱해줍니다.

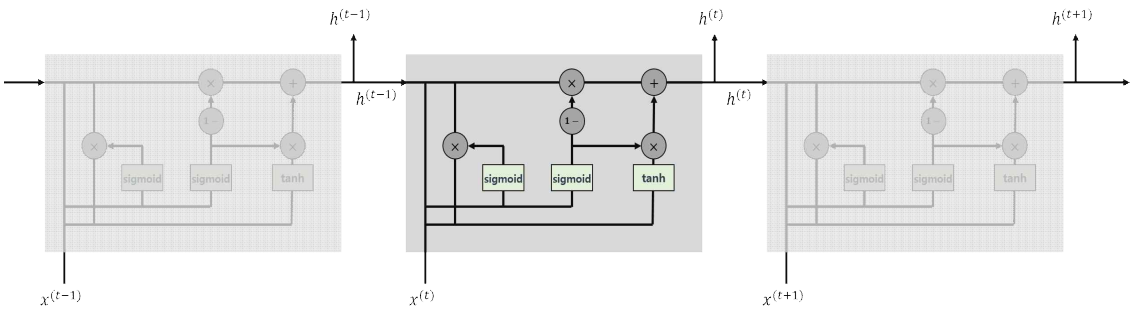


[그림30] GRU, Reset Gate

직전 시점의 은닉층의 값과 현시점의 정보에 가중치를 곱하여 얻을 수 있습니다.

2. Update Gate

Update Gate는 LSTM의 forget gate와 input gate를 합쳐놓은 느낌으로 과거와 현재의 정보의 최신화 비율을 결정 합니다. Update gate에서는 sigmoid로 출력된 결과(u (t)u (t))는 현시점의 정보의 양을 결정하고, 1에서 뺀 값(1 −u (t)1 −u (t))는 직전 시점의 은닉층의 정보에 곱해주며, 각각이 LSTM의 input gate와 forget gate와 유사합니다.



[그림31] GRU, Update Gate

3. Candidate
현 시점의 정보 후보군을 계산 하는 단계입니다. 핵심은 과거 은닉층의 정보를 그대로 이용하지 않고 리셋 게이트의 결과를 곱하여 이용해줍니다.

3.2 GRU 분석 모델의 설계

1) GRU 모델 구축

```
model = Sequential()
model.add(GRU(11,
              input_shape=(train_feature.shape[1], train_feature.shape[2]),
              activation= 'tanh',
              return_sequences=False)
        )


model.add(Dense(1, activation='sigmoid'))
model.summary()
```

[그림32] GRU 모델 구축을 위한 코드

- keras는 레이어를 쌓아서 모델을 구축한다. 이 중 순차모델인 'Sequential()'을 model에 불러오고, 해당 모델에 '.add()' 방식으로 층을 쌓아간다.
- 모델의 층을 설계할 때, 여러개의 GRU층이나 Dense층 기타 LSTM층들을 조정하여 쌓을 수 있으며, 본 연구에서는 LSTM 2중층 , GRU 2중층, 3중층 모델, GRU와 LSTM, Dense를 조합한 모델등을 설계하여 실험하였고, 결과를 비교하였을 때, 가장 좋은 결과를 나타내는 모델 조합 그룹 중에서, 가장 간략하고 빠른 모델인 GRU 단층 모델을 사용하였다.
- 'GRU()' 레이어에는 output_shape, input_shape, activation 등을 입력한다. output_shape는 GRU 레이어의 결과로 몇 개의 노드가 나올지를 나타내는 것이며, 본 분석에서는 노드의 수를 11, 30, 50, 64, 100, 128, 256개로 변경해가며 실험 하였으며, 유사한 좋은 결과를 보인 11, 50, 100개 중에서 분석 속도가 가장 빠른 11개의 노드를 노드의 개수로 지정해주었다. output 노드의 개수는 사용자가 적절한 개수를 지정해주면 된다. input_shape는 feature의 개수 즉, 독립변수의 개수와 timestep의 수를 지정해주어야 한다. 위에서

test_feature.shape의 결과로 (관측치 개수, timestep, feature)가 나온 것을 확인할 수 있었다. 이를 활용하여 1번(두 번째) 인덱스인 timestep과 2번(세 번째) 인덱스인 feature를 불러준다. activation 함수는 비선형 함수인 'tanh'을 사용한다.

- 모델의 마지막 층에는 Dense층을 넣어주어야 하며, 본 분석에서의 결과는 1가지 (OK/NG의 여부)가 나와야 하므로 output_shape를 1로 지정해준다. 결과를 0에서 1사이의 값으로 도출하기 위해서 activation 함수는 'sigmoid'를 지정하였다.

2) 모델 훈련

```
MODEL_PATH = "../data/model"
EARLY_STOPPING_PATIENCE = 3
BETA_1 = 0.9
LREARNING_RATE = 0.016

nadam_opt = Nadam(
    beta_1=BETA_1
    , learning_rate=LREARNING_RATE
)
model.compile(loss='binary_crossentropy', optimizer=nadam_opt, metrics=['accuracy'])
early_stop = EarlyStopping(monitor='val_loss', patience=EARLY_STOPPING_PATIENCE)
filename = "/".join([MODEL_PATH, 'checkpoint.h5'])
checkpoint = ModelCheckpoint(filename, monitor='val_loss', verbose=0,
                             save_best_only=True, mode='auto')
```

[그림 33] model_path 및 compile optimizer 설정

- 모델 훈련 전에 model의 check_point를 저장하기 위한 model_path를 지정해준다. 처음 working directory를 지정하였던 path를 그대로 활용하는 것을 예시로 두었으며, 사용자의 편의에 따라 변경하면 된다.
- 'model.compile()' 함수를 활용하여 모델 학습을 진행하며, loss 함수는 현재 가중치에서 틀린 정도를 알려주는 함수이다. 본 분석은 이진분류이기 때문에 이에 가장 적합한 'binary_crossentropy'를 활용한다. optimizer는 가중치의 최적값을 찾아가는 방법을 설정하는 것이다. 본 분석에서는 대체적으로 성능이 좋아 주로 사용하는 optimizer인 'adam'과 'RMSprop', 'Nadam'을 가지고 모델을 학습해본 결과 이 데이터셋 모델에서 가장 성능이

좋았던 ‘Nadam’을 사용하였다. ‘Nadam’은 기존의 ‘adam’에서 nesterov accerlated gradient 방식을 적용한 optimizer 이다.

- optimizier ‘Nadam’에서 하이퍼파라미터인 Beta1과 Learing rate등을 조정 할 수 있다. 하이퍼파라미터란 모델 구축시 사용자가 직접 설정해 주는 값을 말한다. 하이퍼파라미터 설정에 따라서 모델의 성능이 확연히 달라질 수 있으며, 본 연구에서는 Beta1값과 Learning rate를 각각 [0.3, 0.5, 0.7, 0.9], [0.0002, 0.0005, 0.016, 0.036, 0.05] 범위에서 탐색한 결과 총 20개의 조합 중 가장 좋은 결과를 보인 Beta1 0.9, Learning rate 0.016을 사용하여 평가를 진행하였다.
- ‘EarlyStopping()’은 검증 데이터의 loss 값을 기준으로, 가장 loss가 최적화되는 시점에 학습을 종료하는 것이다. 너무 과도하게 학습을 진행하다 보면 훈련 데이터의 특성을 너무 많이 반영해버려 과적합(overfitting)이 발생한다. 과적합이 발생하게 되면 훈련 데이터에는 성능이 좋지만 다른 데이터를 모델에 넣게 되면 해당 데이터에는 성능이 좋지 못한 모델이 된다. 이를 방지하기 위하여 모델이 최적인 지점에 멈추어 주어야 하는데 이 지점을 저장하기 위하여 ‘EarlyStopping()’을 사용한다. filename과 checkpoint 모두 모델의 최적 학습 지점을 저장하기 위하여 지정해주며, 최적 학습 지점의 기준은 ‘val_loss’ 즉, 검증 데이터의 loss 값으로 지정해준다. 저장 경로나 파일 이름 등은 사용자의 편의에 맞게 지정해주면 된다.

```
history = model.fit(
    x_train
    , y_train
    , epochs=50
    , batch_size=256
    , validation_data=(x_valid, y_valid)
    , callbacks=[
        early_stop,
        checkpoint
    ]
)
```

[그림 34] model 학습을 위한 코드

- keras에서는 모델 학습을 위하여 ‘fit()’ 함수를 사용한다. 이때, 리턴 값으로 history 정보를 리턴한다. history 정보에는 매 epoch마다의 loss(훈련 손실값), acc(훈련정확도), val_loss(검증 손실값), val_acc(검증 정확도) 값이 저장된다. history에 모델 학습 정보를 저장하면 학습이 끝난 후에 history에 저장된 값을 코드로 쉽게 확인이 가능하다.
- ‘fit()’ 안에 넣는 파라미터들을 설명하면, 먼저 훈련 데이터인 x_train, y_train을 데이터로 넣어준다. 전체 데이터셋에 대해 한 번 학습 과정이 완료되는 것을 의미하는 epochs, 한 번의 batch마다 주는 데이터 샘플의 size인 batch_size를 지정해준다. epochs과 batch_size는 사용자의 컴퓨팅 환경이나 분석 상황에 맞게 정해주면 된다. validation_data 파라미터에는 검증 데이터로 분류해뒀던 x_valid, y_valid를 지정하고 위에서 설명한 최적의 학습 지점에서 학습을 멈추기 위해 callbacks에 저장한 early_stop과 checkpoint 정보를 입력해준다.
- 위의 코드를 실행하면 아래와 같은 학습 과정이 나타난다.

```
Epoch 1/50
1599/1599 [==============================] - 11s 6ms/step - loss: 0.1551 - accuracy: 0.9346 - val_loss: 0.0977 - val_accuracy: 0.9620
Epoch 2/50
1599/1599 [==============================] - 10s 6ms/step - loss: 0.0869 - accuracy: 0.9669 - val_loss: 0.0728 - val_accuracy: 0.9711
Epoch 3/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0634 - accuracy: 0.9767 - val_loss: 0.0800 - val_accuracy: 0.9673
Epoch 4/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0516 - accuracy: 0.9797 - val_loss: 0.0460 - val_accuracy: 0.9809
Epoch 5/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0457 - accuracy: 0.9816 - val_loss: 0.0377 - val_accuracy: 0.9847
Epoch 6/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0420 - accuracy: 0.9831 - val_loss: 0.0395 - val_accuracy: 0.9835
Epoch 7/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0412 - accuracy: 0.9835 - val_loss: 0.0397 - val_accuracy: 0.9845
Epoch 8/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0380 - accuracy: 0.9848 - val_loss: 0.0369 - val_accuracy: 0.9841
Epoch 9/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0367 - accuracy: 0.9853 - val_loss: 0.0408 - val_accuracy: 0.9838
Epoch 10/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0365 - accuracy: 0.9853 - val_loss: 0.0387 - val_accuracy: 0.9838
Epoch 11/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0356 - accuracy: 0.9856 - val_loss: 0.0303 - val_accuracy: 0.9875
Epoch 12/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0340 - accuracy: 0.9859 - val_loss: 0.0375 - val_accuracy: 0.9851
Epoch 13/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0388 - accuracy: 0.9845 - val_loss: 0.0325 - val_accuracy: 0.9866
Epoch 14/50
1599/1599 [==============================] - 11s 7ms/step - loss: 0.0357 - accuracy: 0.9856 - val_loss: 0.0467 - val_accuracy: 0.9816
```

[그림 35] 학습과정

```
model.load_weights(filename)
pred = model.predict(test_feature)
```

```
7830/7830 [==============================] - 15s 2ms/step
```

[그림 36] model 가중치를 로드 및 예측을 위한 코드

- load_weights()함수를 활용하여 학습된 모델의 가중치를 불러온다. 해당 모델에 테스트 데이터의 독립변수를 넣고 예측을 진행하여 pred에 저장한다.

4. 분석 결과 및 시사점

1) GRU 의 모델 평가 지표

- 용해탱크 데이터를 이용하여 품질을 예측하는 모델의 성능을 확인하기 위한 평가 척도로 혼동 행렬(Confusion Matrix)을 이용한다. 실제(Condition)의 양성(Positive)은 실제 불량일 경우이며, 음성(Negative)은 실제 불량이 아닌 경우이다. 예측(Prediction)의 양성(Positive)은 표준 데이터를 이용하여 검출했을 때 불량인 경우, 음성(Negative)은 불량이 아닌 경우이다. 예측값과 실제값이 일치 혹은 불일치하는 경우를 아래의 표와 같이 나타낼 수 있다. 표의 각 항목에 대한 설명은 다음과 같다.

  ○ TP(True Positive)는 참이라고 예측했으며 실제로도 참인 경우의 횟수이다.

  ○ FP(False Positive)는 참이라고 예측했지만 실제로 거짓인 경우의 횟수이다.

  ○ FN(False Negative)은 거짓이라고 예측했지만 실제로 참인 경우, 즉 거짓이라는 예측 결과가 틀린 횟수이다.

  ○ TN(True Negative)은 거짓이라고 예측한 결과가 실제로 거짓인 경우의 횟수이다.

		Condition(실제)	
		Positive	Negative
Prediction (예측)	Positive	TP-True Positive (참 양성)	FP-False Positive (긍정 오류)
	Negative	FN-False Negative (부정 오류)	TN-True Negative (참 음성)

[그림 37] 혼동 행렬(Confusion Matrix) 지표

- 위의 4가지 지표를 통해 분류 성능평가 척도인 정확도(Accuracy),정밀도(Precision), 민감도(Sensitivity), 재현율(Recall), 특이도(Specificity), 음성 예측값(Negative Predict Value(NPV))을 계산할 수 있다. 정확도는 참과 거짓이 모두 올바르게 분류된 확률을 의미한다. 정밀도는 참으로 예측한 것 중 실제 참이 맞는 경우의 비율이다. 민감도는 실제 참인 경우 중 참으로 예측된 경우의 비율이다. 재현율은 민감도와 동일한 의미의 지표이다. 특이도는 실제 거짓인 경우 중 거짓으로 예측된 것의 비율을 의미한다. 음성 예측값(Negative Predict Value(NPV))는 거짓으로 예측한 것 중 실제 거짓이 맞는 경우의 비율이다.

  5가지 성능평가 지표를 위의 표에 기반하여 식으로 나타내면 다음과 같다.

  * 정확도 = (TP+TN)/(TP+FP+FN+TN)
  * 정밀도 = TP/(TP+FP)
  * 민감도 = 재현율 = TP/(TP+FN)
  * 특이도 = TN/(FP+TN)
  * 음성 예측값 = TN/(FN+TN)

- 정확도는 분류 결과를 총체적인 관점에서 살펴보기에 적절하다. 하지만 참과 거짓을 구분하여 고려했을 때, 각각 잘 분류된 것인지 판단하기 어렵다. 따라서 위의 성능평가 척도를 이용하여 분류 결과를 비교적 세밀하게 확인할 수 있다.
- 여기서 주목할 점은, 최종적으로 각 성능 지표가 높게 산출되는 것이 현실적으로 어렵다는 것이다. 민감도와 재현율은 동일한 의미이지만 이를 구분하여 생각한다면, 먼저 정밀도와 재현율은 서로 상충관계라고 할 수 있다. 만약 분류기가 실제로 참일 확률이 높은 것만 최종적으로 참이라 분류해낸다면, 정밀도와 재현율은 동시에 높아지기 어렵다. 정밀도는 참으로 예측된 것에 관한 비율이며 재현율은 실제 참인 것의 비율이기 때문이다. 이러한 문제를 해결하는 지표로 F1-score를 사용한다. F1-score는 정밀도와 재현율의 조화평균이다. 즉, 정밀도와 재현율을 동시에 고려한 지표로 F1-score가 높은 경우 두 지표가 적절히 조화되어 분류가 잘 되었다고 판단할 수 있다. 이에 대한 계산식은 다음과 같다.

  F1-score = 2*{(정밀도*재현율)/(정밀도+재현율)}

- 다음으로, 민감도와 특이도 또한 상충관계이다. 전체 데이터 수는변하지 않고 그 안에서 참과 거짓이 분류되기 때문에, 참으로 분류된 값이 많아지면 자동적으로 거짓이라고 분류되는 값의 수는 줄어든다.
- 음성 예측값의 경우 음성으로 예측한 값 중 실제 음성인 값의 비율

으로 음성 예측값이 높을수록 모델이 실제 음성인 값을 잘 분류한다고 볼 수 있다. 이번 목표가 NG값을 발견하는 것이 목표이이므로 이 값 또한 중요하다.

○ 따라서 이러한 개념을 바탕으로 각 성능평가 척도를 기준으로 한 충분한 F1-score, 성 예측값이 확인되어야 하며 이를 통해 분석 결과의 신뢰성을 확보할 수 있다.

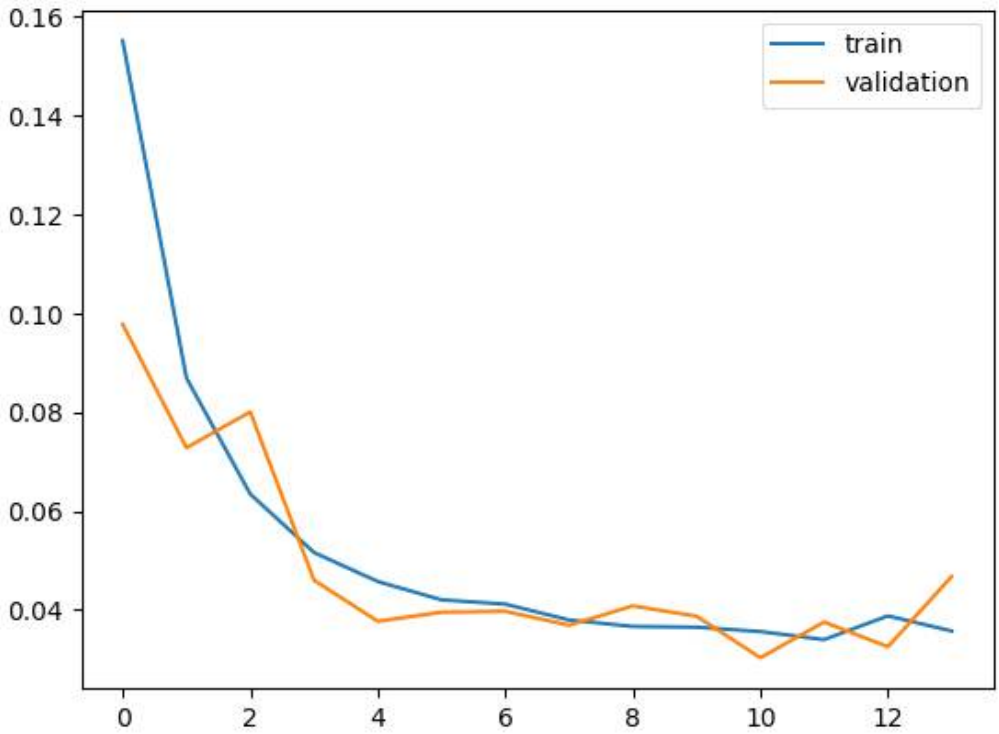
2) 모델 평가 및 결과 해석

- loss 변화 그래프

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='validation')
plt.legend()
plt.show()
```

[그림 38] 훈련 및 검증 데이터의 loss 그래프를 그리기 위한 코드

- plot()을 활용하여 history에 저장된 훈련 데이터의 loss, 검증 데이터의 loss를 그래프로 그린다.



[그림 39] 훈련 및 검증 데이터의 loss 그래프 결과

- 모델 학습이 진행될 때마다 loss 값이 점점 줄어드는 것(성능이 좋아지는 것)을 확인할 수 있다. 모델이 훈련 데이터에 과적합 되는 것을 방지하고 적절한 지점까지 학습하도록 earlystopping을 진행하였다.

- 테스트 데이터의 예측 결과 평가

```
pred_df = pd.DataFrame(pred, columns=['TAG'])

pred_df.loc[pred_df['TAG'] >= 0.5,'TAG'] = 1
pred_df.loc[pred_df['TAG'] < 0.5,'TAG'] = 0
```

[그림 40] 테스트 데이터의 예측 결과를 데이터프레임 형태로 변환하는 코드

- pred_df에 저장한 테스트 데이터의 예측 결과를 데이터프레임의 형태로 변환시킨다. 예측 결과는 0~1 사이의 값, 확률로 나타나는데, 이를 실제 값인 0과 1로 비교하기 위하여 값들을 변환해준다. 1일 확률이 50% 이상인 경우 1로, 미만인 경우는 0으로 예측값들을 변환해주었다.

```
print(pred_df['TAG'].value_counts())

classify = confusion_matrix(test_label, pred_df)
print(classify)
```

[그림 41] TAG 칼럼의 각 값의 개수 확인및 혼동 행렬(confusion matrix)을 위한 코드

- value_counts()로 확인해보면, 아래와 같이 나오며 모델이 1(OK)로 예측한 값은 198,592개, 0(NG)으로 예측한 값은 51,965개이다. 또한 sklearn의confusion_matrix()’ 를 활용하여 모델이 예측한 데이터가 실제로 OK/NG인지 등을 비교해보면 아래와 같은 결과가 나온다. 행렬에서 행은 모델의 예측값을 의미하며 첫 번째 행은 모델이 0(NG)으로 예측한 데이터, 두 번째 행은 모델이 1(OK)로 예측한 데이터이다. 열은 실제 데이터를 의미하며, 첫 번째 열은 실제 0(NG)인 데이터, 두 번째 열은 실제 1(OK)인 데이터를 의미한다. 행렬을 해석해보면, 모델이 NG로 분류한 데이터 중 실제 NG인 데이터는 50,974개, NG로 분류했지만 실제로는 OK인 데이터는 991개이다. 모델이 OK로 분류한 데이터 중에서 실제로 OK인 데이터를 잘 분류한 것은 199,446개이며, 실제 NG인데 OK로 잘못 분류한 것은 2,146개이다.

```
1.0    198592
0.0     51965
Name: TAG, dtype: int64
[[ 50974   2146]
 [   991 196446]]
```

[그림 42] TAG칼럼의 각 값의 개수와 혼동 행렬(confusion matrix) 확인 결과

- 조금 더 정밀한 해석을 위하여 정밀도, 재현율, f1-score, 정확도,

음성 예측값을 계산하여 출력한다.

```
tn, fp, fn, tp = classify.ravel()

print("############################################################")
p = precision_score(test_label, pred_df)
print(f"precision: {p:.4}")
r = recall_score(test_label, pred_df)
print(f"recall: {r:.4}")
f1 = f1_score(test_label, pred_df)
print(f"f1-score: {f1:.4}")
acc = accuracy_score(test_label, pred_df)
print(f"accuracy: {acc:.4}")
npv = tn / (tn + fn)
print(f"negative predict value: {npv:.4}")
print("############################################################")
```

```
############################################################
precision: 0.9892
recall: 0.995
f1-score: 0.9921
accuracy: 0.9875
negative predict value: 0.9809
############################################################
```

[그림 43] 정밀도, 재현율, f1-score, 정확도, 음성예측값 계산 코드와 결과값

- precision_score, recall_score, f1_score, accuracy_score 모두 sklearn라이브러리에서 제공하고 있으며, 이들 함수를 활용하여 쉽게 각각의 지표들을 계산할 수 있으나 음성예측값은 제공하지않아 직접 계산하였다. 분석 결과 정밀도(모델이 1로 예측한 데이터 중 실제로 1인 데이터의 비율)는 0.9892, 재현율(실제로 1인 데이터를 모델이 1이라고 예측한 데이터의 비율)은 0.995, f1-score(정밀도와 재현율의 조화평균)는 0.9921, 정확도는 0.9809, 음성예측값은 0.9809로 나타났다.

3) 분석 결과 해석하기

- 분석 결과 모델의 정확도는 0.9809로 나타났다. 훈련 데이터와 검증 데이터에 대한 정확도는 98%이다. 하지만 분류 모델에서 정확도만으로 성능을 평가하지 않으며, 다양한 지표들을 활용하여 모델의 분류 성능을 측정해야한다.

- 정밀도(모델이 1로 예측한 데이터 중 실제로 1인 데이터의 비율)는 0.9892로 모델이 OK로 예측한 데이터 중 98.9%가 실제로 OK였다는 것을 의미한다.

- 재현율(실제로 1인 데이터를 모델이 1이라고 예측한 데이터의 비

율)은 0.995으로 실제로 OK인 데이터 중 모델이 OK로 예측한 데이터는 99%이다.

- 정밀도와 재현율은 모델의 성능을 측정하는 데 유용하게 사용되지만 둘 다 완벽한 통계치는 아니기 때문에 이들의 최종적으로 조화평균인 f1-score과 목적에 부합하는 음성예측치로 모델이 얼마나 효과적인지를 설명한다. 본 분석에서 모델의 f1-score(정밀도와 재현율의 조화평균)는 0.9921(99%)로 높게 나타났다. 이를 통해 모델이 효과적으로 품질을 예측하고 있다는 것을 확인할 수 있다. 마지막으로 음성예측치는 0.9809(98%)이다.

4) 확률밀도함수와 누적분포함수를 통한 모델의 강건성 테스트

- 위의 모델으로 한번의 독립시행결과 모델이 높은 정확도를 나타내었다고 항상 위 모델으로 학습 및 평가를 하였을때 같은 정확도를 나타내는 것은 아니다. 상기 모델을 가지고 50회 학습 및 예측을 독립시행하여 진행, 모델의 성능을 평가하여 보았다.

- 모집단이 정규 분포를 따른다고 가정하였을 때, 시행횟수(50회)가 충분히 크므로 표본집단은 중심극한정리에 따라 정규분포를 따른다는 것이 알려져 있다. 상기 정리에 따라 위 모델로 평가를 진행하였을 때 각 평가 지표의 확률 밀도 함수와 누적확률밀도 함수를 그려 모델의 재현성과 성능평가를 진행하였다.

```
def search_window(n_loop, X_values, y_values):

    best_score_list = []

    for i in range(n_loop):

        x_feature = np.array([
            X_values[i:i+WINDOW_SIZE]
            for i in np.arange(len(X_values) - WINDOW_SIZE)
        ])
        y_label = np.array([
            np.array([y_values[i+WINDOW_SIZE-1]])
            for i in np.arange(len(y_values) - WINDOW_SIZE)
        ])

        (
            train_feature
            , test_feature
            , train_label
            , test_label
        ) = train_test_split(x_feature, y_label, random_state=42, test_size=0.3, stratify=y_label)
        (
            x_train
            , x_valid
            , y_train
            , y_valid
        ) = train_test_split(train_feature, train_label, random_state=42, test_size=0.3, stratify=train_label)

        model = Sequential()
        model.add(GRU(11,
                    input_shape=(train_feature.shape[1], train_feature.shape[2]),
                    activation= 'tanh',
                    return_sequences=False)
                )


        model.add(Dense(1, activation='sigmoid'))

        # 모델 훈련
        nadam_opt = Nadam(
            beta_1=BETA_1
            , learning_rate=LREARNING_RATE
        )
        model.compile(loss='binary_crossentropy', optimizer=nadam_opt, metrics=['accuracy'])
        early_stop = EarlyStopping(monitor='val_loss', patience=EARLY_STOPPING_PATIENCE)
```

```
        model.fit(
            x_train
            , y_train
            , epochs=50
            , batch_size=256
            , validation_data=(x_valid, y_valid)
            , callbacks=[
                early_stop
            ]
            , verbose=0
        )

        pred = model.predict(test_feature, verbose=0)
        pred_df = pd.DataFrame(pred, columns=['TAG'])

        pred_df.loc[pred_df['TAG'] >= 0.5,'TAG'] = 1
        pred_df.loc[pred_df['TAG'] < 0.5,'TAG'] = 0

        classify = confusion_matrix(test_label, pred_df)
        tn, fp, fn, tp = classify.ravel()

        p = precision_score(test_label, pred_df)
        r = recall_score(test_label, pred_df)
        f1 = f1_score(test_label, pred_df)
        acc = accuracy_score(test_label, pred_df)
        npv = tn / (tn + fn)
```

```
        best_score_list.append({
            "precision": p
            , "recall": r
            , "f1": f1
            , "accuracy": acc
            , "npv": npv
        })
    return pd.DataFrame(best_score_list)
```

```
score_df = search_window(50, X_values, y_values)
```

[그림 44~46] 모델의 강건성 테스트 코드

- 50번의 루프를 통해 전에 시행했던 데이터 생성, 모델생성, 옵티마이저 설정, 학습, 예측코드를 하나의 함수로 만들었다. 차이점은 이번에는 체크포인트를 설정하지 않아 한번 예측한 모델은 파기되고 다음 루프가 실행되면 다시 모델을 생성하는 방식으로 하였다.

```
score_df.describe()
```

	precision	recall	f1	accuracy	npv
count	50.000000	50.000000	50.000000	50.000000	50.000000
mean	0.988298	0.988297	0.988275	0.981514	0.956953
std	0.006443	0.004923	0.003163	0.005021	0.017216
min	0.972561	0.976737	0.976100	0.962196	0.918721
25%	0.984864	0.985150	0.987168	0.979796	0.946492
50%	0.989858	0.988748	0.988735	0.982240	0.958564
75%	0.993657	0.991512	0.990113	0.984422	0.967919
max	0.996552	0.998030	0.993321	0.989495	0.991937

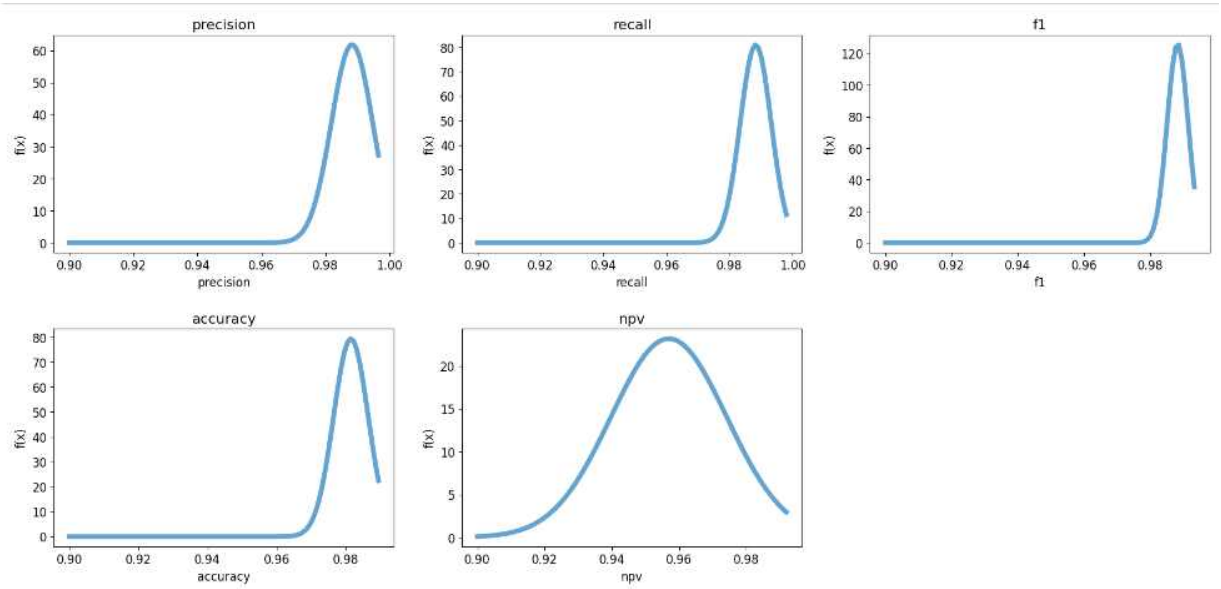
[그림 47] 50번 예측한 결과의 통계치 코드 및 결과

- 위의 코드를 실행한 결과 루프당 모델이 예측한 평가값에 대한 통계치이다. 최소평가치는 음성평가치이며 약 0.91이다.

```
plt.style.use('default')
plt.rcParams['figure.figsize'] = (21,9) # 21(넓이) X 9(높이)로 설정
plt.rcParams['font.size'] = 12 # 폰트크기
plt.rcParams['lines.linewidth'] = 5 # 선의 넓이

g_name = ["precision", "recall", "f1", "accuracy", "npv"] # 각 score의 이름 리스트

for i, g in enumerate(g_name):
    mul, sigma1 = score_df[g].mean(), score_df[g].std() # 평균과 표준편차
    x = np.linspace(0.9, score_df[g].max(), 100) # 0.9 ~ 1까지의 분포를 보여줌
    y1 = (1 / np.sqrt(2 * np.pi * sigma1**2)) * np.exp(-(x-mul)**2 / (2 * sigma1**2)) # 확률밀도 계산
    num = 231 + i # 서브플롯번호 생성
    plt.subplot(num) # 서브플롯생성
    plt.plot(x, y1, alpha=0.7)  # 그래프에 데이터 입력

    plt.title(g) # 그래프 타이틀
    plt.xlabel(g) # x축의 라벨
    plt.ylabel('f(x)') # y축의 라벨
plt.subplots_adjust(left=0.125,bottom=0.1,right=0.9,top=0.9,wspace=0.2,hspace=0.35)
plt.show()
```

[그림 48] 각 평가값을 확률밀도함수로 그래프를 그리기 위한 코드

- 위의 데이터프레임을 활용하여 확률 밀도 함수를 그래프로 그리면 아래와 같다.

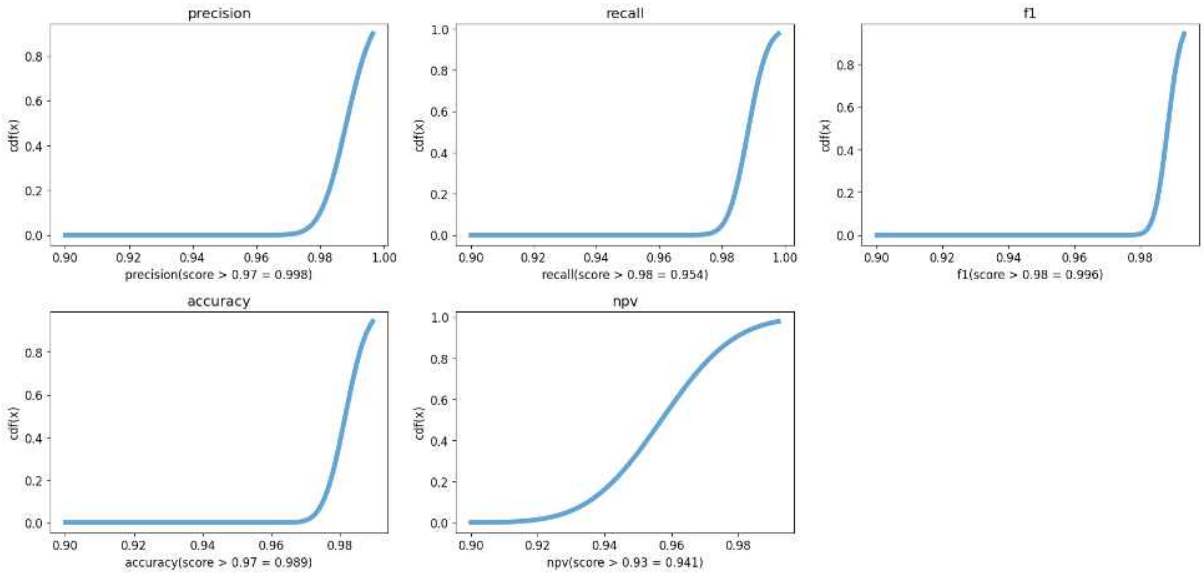


[그림 49] 각 평가값의 확률밀도함수 그래프

```
plt.style.use('default')
plt.rcParams['figure.figsize'] = (21,9) # 21(넓이) X 9(높이)로 설정
plt.rcParams['font.size'] = 12 # 폰트크기
plt.rcParams['lines.linewidth'] = 5 # 선의 넓이

for i, (g, cdf_from) in enumerate(zip(g_name,[0.97,0.98,0.98,0.97,0.93])):
    mu1, sigma1 = score_df[g].mean(), score_df[g].std() # 평균과 표준편차
    x = np.linspace(0.9, score_df[g].max(), 100) # 0.9 ~ 1까지의 분포를 보여줌
    y_cum = 0.5 * (1 + erf((x - mu1)/(np.sqrt(2 * sigma1**2)))) # 누적분포계산

    cdf_val = 1 - scipy.stats.norm(
                    loc = score_df[g].mean()
                    , scale = score_df[g].std()
                ).cdf(cdf_from) #
    num = 231 + i # 서브플롯번호 생성
    plt.subplot(num) # 서브플롯생성
    plt.plot(x, y_cum, alpha=0.7)  # 그래프에 데이터 입력

    plt.title(g) # 그래프 타이틀
    plt.xlabel(f"{g}(score > {cdf_from} = {cdf_val:.3f})") # x축의 라벨
    plt.ylabel('cdf(x)')

plt.subplots_adjust(left=0.125,bottom=0.1,right=0.9,top=0.9,wspace=0.2,hspace=0.3)
plt.show()
```

[그림 50] 각 평가값을 누적분포함수로 그래프를 그리기 위한 코드

- 위의 데이터프레임을 활용하여 누적분포함수를 그래프로 그리면 아래와 같다.



[그림 51] 각 평가값을 누적분포함수로 그래프를 그리기 위한 코드

- 두 그래프를 보았을시 첨도가 높아 모델을 여러번 돌렸을 시 같은 확률이 나올 가능성이 크며 전체 평가값 중 정확도가 93%이상 나올확률은 94%이다.

5) 제조 데이터 분석 기대효과

- 본 분석을 통해 설비운영값과 생산품질의 연관성을 도출하여, 설비운영조건에 따른 생산품질을 예측할 수 있게 하며, 생산성은 높이고 불량률은 낮추는 설비운영 최적화를 위한 분석 데이터셋을 구축한다. 공정설비 운영데이터에 따라 생산품질을 예측할 수 있는 제품 생산 공정에 넓게 활용할 수 있을 것으로 판단된다.
- 특히, 불량 발생 후 원인분석을 진행하는 결과론적 품질분석이 아닌 예견된 불량을 효율적으로 방지할 수 있는 예지적 품질분석을 수행할 수 있게 되어, AI 추론을 통한 생산시점의 실시간 불량 발생 예측 알람을 활용할 수 있을 것으로 판단된다.

6) 시사점(implication) 요약기술

- 식품제조업체 H사 공장에서는 용해공정의 공정품질(원료 용해상태)을 육안으로만 확인하거나, 샘플 확인도 못 한 상태로 다음 공정으로 진행되기 때문에 공정 중 발생하는 변동 상황에 대해 숙련자의 판단으로 설비운영 셋팅값을 조정하여야 했고, 인력 부재시에는 이같은 주관적 판단에 의한 품질관리마저도 하기 힘들었다.
- 따라서 공정중 설비운영값을 기준으로 제품의 최종 품질을 예측할 수 있는 AI 기법을 적용하여, 주요 공정 운영변수의 전체적인 변화가 품질에 미칠 영향 모델링을 도출하고자 한다. 용해공정 설비의 생산 데이터를 예제로 기본적인 데이터 탐색을 통해 전체 데이터 및 각 변수의 특성을 파악하는 방법을 학습하고, 이를 바탕으로 생산품 품질에 주된 영향을 미치는 공정변수 및 그 연관성 등을 파악하며, 최종 품질을 예측하는 모델을 구성한다.
- 용해공정의 설비운영 데이터와 최종품질검사 데이터를 수집하고, 데이터 가공/전처리, AI 모델 개발과 제조공정의 적용 및 검증을

통해 열악한 중소기업에 빅데이터 및 AI 기술을 적용하여 실질적인 품질향상 및 비용절감에 기여한다는 점에서 시사하는 바가 크다고 판단된다.

5. 중소제조기업에 미치는 파급효과

- 상기 분석 결과를 통해, GRU 알고리즘을 현장에 적용하여 용해공정의 품질예측을 매우 효과적으로 수행할 수 있음을 알 수 있다.
- 본 분석에서 사용된 GRU 알고리즘은 딥러닝 기반의 알고리즘으로 개별 변수가 품질에 미치는 영향에 대한 구체적인 내용은 알아내기 힘들다는 단점이 있으나, 전체적인 품질예측력에 초점을 맞추고 있으며, 본 보고서에서 사용된 예제와 같이 다수 변수가 상호영향을 미치고 있는 복잡한 영향 관계성을 띠고 있는 용해공정에서는 사용 목적에 적합하다고 할 수 있다.
- 본 보고서에서 개발한 분석 모델을 실제 현장에 적용하여 실시간 품질예측을 수행하기 위해서는 실시간 스트리밍 분석 인공지능 추론 플랫폼에 해당 인공지능 모델과 가중치(weight parameters)를 탑재하고, 실시간 시계열 데이터를 입력값(input)으로 넣을 시, 실시간 품질예측 결과를 출력값(output)으로 얻을 수 있다. 물론, 이와 같이 실시간 시계열 데이터와 추론 플랫폼을 활용하기 위해서는 데이터 수집 및 관리 플랫폼(스마트공장) 구축이 수반되어야 한다.

- 유사 타 현장의 「용해탱크 AI 데이터셋」 분석 적용
  - ○ 연속공정을 사용하는 대부분의 중소기업 생산현장에서는 공정중에 생산탱크나 관에서 내용물 샘플을 채취하여 중간 품질검사를 수행하기 쉽지 않은 상황이다. 또, 공정검사/자주검사가 가능한 상황이더라도 품질에 영향을 미치는 다양한 공정변수, 특히 공정변수들 간에 상호영향을 미치는 경우에는 숙련자의 경험과 노하우에 의존하여 상황에 맞게 설비운영을 변경할 수밖에 없는 상황이다.
  - ○ 이와 같은 어려움이 있는 산업현장에서 만약 공정운영 데이터를 설비 PLC 또는 센서를 통해 확인 및 수집이 가능하고, 품질검사

결과를 전산화하여 관리하고 있다면 본 분석은 적용 가능하다.

- 본 「용해탱크 AI 데이터셋」 분석을 원용하여 타 제조현장 적용시, 주요 고려사항
  - ○ 생산제품 및 투입원료에 따라 분석 결과가 달라지기 때문에 해당 제품의 데이터로 분석 모델을 다시 학습시켜야 한다.
  - ○ 용해공정에서 측정되는 다른 변수(공정데이터, 환경데이터, 품질데이터 등)들도 포함하여 모델을 학습시켜 사용할 경우 더 정확한 모델을 생성할 수도 있다.
  - ○ 본 분석에서 활용한 특성값은 타 제조현장의 상황과 맞지 않을 수 있기 때문에 현장 전문가의 의견을 반영하여 적용 여부를 결정해야 한다.