기계학습론 IC-PBL

반도체 제품의 불량 여부 예측과 성능 향상에 대한 연구

한양대학교 ERICA캠퍼스 공과대학

조 이름: 러닝 머신(4조)

로봇공학과 2016006999 김민재

로봇공학과 2017010782 송창연

로봇공학과 2019071385 임준희

디자인테크놀로지전공 2019041167 최다연

<목 차>

<표 목차>

<그림 목차>

Ⅰ. 서론

4 차 산업혁명시대와 함께 최근 정보통신 분야에서의 화두는 단연 빅데이터(Big Data)다. 빅데이터란, 기존 데이터 베이스로는 수집, 저장, 분석 등의 업무를 하기 어려울 만큼의 방대한 양을 가진 데이터를 의미한다. 세계적으로 유명한 컨설팅 기관인 맥킨지(Mckinsey)는 빅데이터를 기존 데이터 베이스의 수집, 저장, 분석 등의 업무 역량을 넘어서는 규모로서, 주관적이며 앞으로도 진화할 것이라고 언급한 바 있다. 또한, 맥킨지의 빅데이터 보고서에서는 빅데이터 분석을 위한 다양한 분석 기법들이 정리되어 있으며, 이 기법들을 대부분 대규모 데이터 처리를 위한 통계적, 확률적 방법론에 바탕을 두고 있다. 실제 분석에 필요한 빅데이터의 성격과 활용 분야에 맞게 시공간 분석, 시각화 분석 등이 선택적으로 활용되고 있다. 빅데이터 분석 기법으로는 여러 기법들이 존재 하나 대표적으로는 **분류(Classification), 군집화(Clustering), 기계학습(Machine Learning), 회귀분석(Regression)** 등이 이루어져 있다. 이러한 다양한 분석 기법을 간단히 정리하면, 다음과 같다.

분류(Classification)란, 미리 알려진 클래스들로 구분되는 훈련 데이터군을 학습시켜 새롭게 추가되는 데이터가 속할 만한 데이터 군을 찾는 학습 방법을 뜻하며, 다른 표현으로는 지도학습 방법이다. 군집화(Clustering)란, 비슷한 특성이 있는 데이터들을 합쳐가면서 유사 특성 군으로 분류하는 학습 방법을 뜻 한다. 최종적으로 정해져 있지 않은 클래스들의 묶음 군들로 분류되는데, 분류와 달리 훈련 데이터군이 이용되지 않기 때문에 비지도 학습이라고도 표현하다. **머신 러닝(Machine Learning, 기계 학습)**이란, 인공지능 분야에서 인간의 학습을 모델링한 것으로, 빅데이터 분석을 포함한 패턴 인식 등 다양한 분야에서 기본적으로 많이 활용되는 기법이다. 결정 트리와 같은 기호적 학습, 신경망이나 유전자 알고리즘과 같이 비기호적 학습, 베이지안 추론(Bayesian inference) 혹은 은닉 마르코프 모델(Hidden Markov Model)과 같은 확률적 학습 등 다양한 기계학습 기법이 존재한다. 마지막으로 **회귀분석(Regression)**이란, 통계학에서 가장 많이 사용하는 통계기반 분석기법으로, 어떠한 현상에 영향을 주는 원인에 해당하는 독립변수와 영향을 받는 종속 변수가 있을 때, 이러한 변수들 사이의 상관관계를 규명하고자 이용하는 분석 방법이다. 단순 회귀분석, 다중 회귀분석 등을 이용하여 판매량 예측, 생산량 예측 등 다양한 변화 예측에 주로 사용된다.

이러한 분석 기법을 통한 빅데이터의 관심이 급속히 늘어나는 이유는 스마트 팩토리(Smart Factory)의 핵심 기술로 빅데이터를 통한 통계 분석이 주가 되며, 서두에서 설명한 바와 같이 불량이 발생했을 때 다양한 자동화 시스템에서의 현상을 파악할 수 있기 때문이다. 또한 품질, 생산, 안전, 에너지 등 전영역에서 불량에 영향을 초래한 상황을 찾아낼 수 있고, 데이터 시각화를 통해 직관적으로 문제점 파악이 가능하다. 오차 초과 상황의 발생 공정을 확인할 수도 있다는 점도 가장 큰 장점이다. 과거에는 쉽게 찾지 못했던 특정 설비에 대한 기준점 위반 횟수도 찾을 수 있으며 빅데이터를 통한 시장동향, 장비 고장, 보수주기와 원가절감이나 경영판단 등과 같은 영업적인 문제에도 매우 가치 있는 분석을 제공한다.

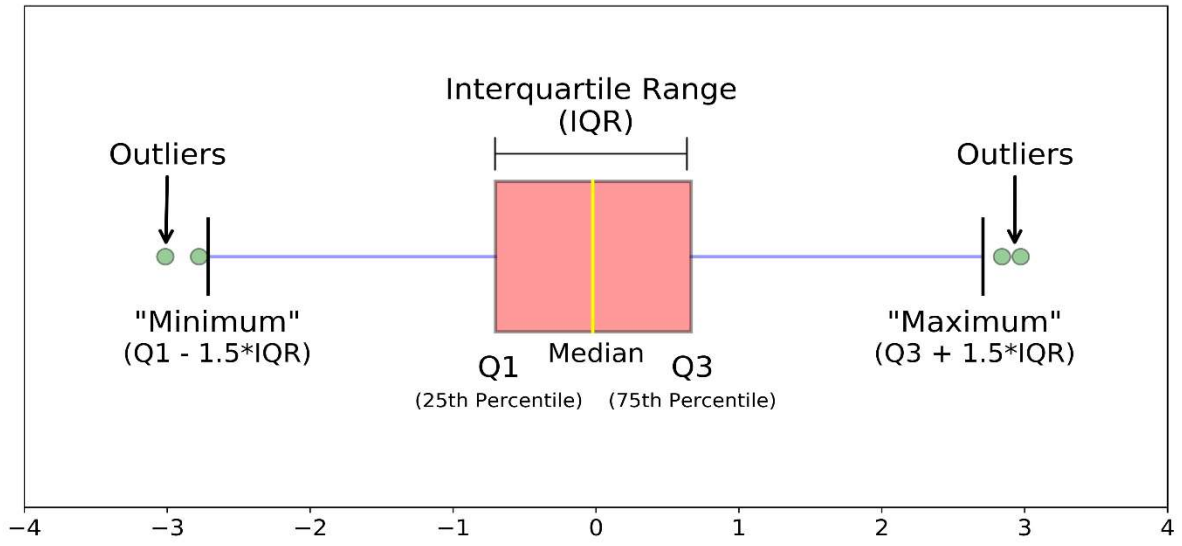
본 연구에서는 데이터 분석에 있어 다음과 같은 과정을 거친다. 우선적으로 교육 과정에 따른 기본적인 데이터 전처리 방법과 머신 러닝 모델을 통해 결과를 도출하고 그에 대한 분석을 한다. 이후 수업에서 다루지 않은 다양한 전처리 기법을 적용하여 최적의 방법을 찾고 검증한다. 다음으로 Pytorch, TensorFlow 와 같은 딥러닝 프레임 워크와 AutoML, AutoSklearn 과 같은 머신 러닝 프레임 워크를 통한 학습 모델 결과를 비교한다. 뿐만 아니라 파라미터 튜닝으로 각 모델이 어떤 영향을 끼치고 있는지 데이터 시각화를 해보며 다양한 평가지표를 통한 최고 성능을 찾기 위한 과정을 연구 목적으로 한다.

Ⅱ. 이론적 배경 및 연구 방법

A. 데이터 전처리

ⅰ. IQR(Inter Quantile Range)

이상치 데이터(Outlier)는 전체 데이터 패턴에서 벗어난 이상 값을 가진 데이터이다. Outlier 로 인해 머신 러닝 모델 성능에 영향을 받는 경우가 발생하기 쉽다. 이 때 IQR(Inter Quantile Range)방식을 이용하여 이상치 데이터를 찾을 수 있다. 이는 사분위(Quantile) 개념으로부터 출발하며, 사분위는 전체 데이터를 값이 오름차순으로 정렬하여 1/4(25%)씩 구간을 분할하는 것을 지칭한다. 여기서 데이터의 75% 백분위 수(Q_3)와 25% 백분위 수(Q_1) 간의 차이로 정의된다 (IQR = $Q_3 - Q_1$). 다음 Box Plot 을 통해 쉽게 이해할 수 있다.



<그림 2-1. IQR 의 정의>

이 IQR 에 1.5 를 곱해서 75% 지점(Q_3)의 값에 더하면 최댓값, 25% 지점의 값(Q_1)에서 빼면 최솟값으로 결정합니다. 이 때, 결정된 최댓값보다 크거나 최솟값보다 작은 값을 이상치라고 간주한다.

ⅱ. PCA(Principal Component Analysis)

차원 축소는 많은 feature 로 구성된 다차원 데이터 세트의 차원을 축소해 새로운 차원의 데이터 세트를 생성하는 것이다. 일반적으로 차원이 증가할수록, 즉 feature 가 많아질수록 예측 신뢰도가 떨어지고 과적합(overfitting)이 발생해 개별 feature 간의 상관관계가 높을 가능성이 있다.

PCA(Principal Component Analysis)는 입력 데이터의 상관 계수 행렬(correlation coefficient matrix)에 대해 고유값 분해를 해주어, 공헌도가 큰 고유값(즉, 상대적으로 큰 고유값들)과 매칭되는 고유벡터들을 찾아서 입력 데이터를 그 벡터들로 투영시킨다. 이 과정에서 공헌도가 작은 고유값과 매칭되는 고유벡터들은 무시된다. 이러한 방식으로 PCA 는 데이터에서 주성분

벡터를 찾아서 데이터의 차원을 축소시킨다. 차원을 축소시킴으로 얻을 수 있는 효과는 아래와 같다.

1) 2차원 또는 3차원으로 축소하면 시각화(visualization)가 가능해진다. 즉, 고차원 데이터를 저차원의 데이터로 축소해서 눈에 보이게 만들 수 있다는 것이다. 그러므로 데이터를 이해하는데 도움이 된다.

2) 데이터 내에는 첨가되어 있는 노이즈가 어느 정도 제거된다.

3) 차원이 축소되었으므로 이후 처리해야할 연산량이 줄어든다.

iii. Maxabs Normalization

데이터 값이 너무 크거나 작은 경우에 값이 0으로 수렴하거나 무한으로 발산하는 것을 방지하기 위해 스케일링(Scaling)을 해준다. 스케일링을 통해 overflow나 underflow를 방지하고 최적화 과정에서 안정성 및 수렴 속도를 향상시켜준다. 이 방법으로는 Standard Scaler, Robust Scaler, MinMax Scaler 등이 있다. 특히 Maxabs scaler는 최대 절대값과 0이 각 1, 0이 되도록 하여 양수 데이터로만 구성되도록 한다. 이때 각 로우마다 정규화를 시켜주게 되고 유클리드 거리가 1이 되도록 데이터를 조정한다.

iv. Yeo-johnson Transformation

Power transform은 데이터를 보다 가우스 함수와 비슷하게 만들기 위해 적용되는 파라메트릭, 단조로운 변환에 속한다. 이는 이분산성(비정수 분산)과 관련된 문제 또는 정규성이 필요한 다른 상황을 모델링하는데 유용하다. 현재 powertransformer는 box-cox transformation과 yeo-johnson transformation을 지원한다. 분산을 안정화하고 왜도를 최소화하기 위한 최적의 방법은 최대의 값을 통해 추정된다. Box-cox는 입력 데이터가 양수여야 하는 반면 yeo-johnson은 양수 또는 음수 데이터 모두를 지원하기 때문에 해당 변환을 이용한다.(기본적으로 0-평균 단위 분산 정규화가 변환된 데이터에 적용된다.)

v. Fix-imbalance

데이터 불균형은 일반적으로 데이터셋 내 클래스의 불균등한 분포를 말한다. 신용 카드 사기 탐지 데이터셋을 예로 들 수 있다. 이는 언더샘플링이나 오버샘플링 등을 통해 보완할 수 있는데 오버샘플링 중 smote 방식을 이용한다.

SMOTE란 Synthetic Minority Oversampling Technique의 약자로 데이터 개수가 적은 클래스의 표본을 가져온 뒤 임의의 값을 추가하여 새로운 샘플을 만들어 데이터에 추가하는 오버샘플링 방식이다.

vi. Remove-multicollinearity

Multicllinearity(다중공선성)는 회귀 분석에서 사용된 모형의 일부 예측 변수가 다른 예측 변수와 상관 정도가 높아 데이터 분석 시 부정적인 영향을 미치는 현상을 말한다. 이 문제를 해결하기 위해 문제를 일으키는 설명변수를 제거하거나 주성분 분석(PCA) 혹은 능형회귀분석(Ridge Regression)과 같은 다른 추정 방법을 이용할 수 있다. 이 중에서도 상관관계가 높은 독립변수 중 일부를 제거하는 방법을 택하여 다중공선성을 줄이는 방법을 택한다.

vii. Remove-outlier(with Z-score)

Z-score 는 관측자나 데이터 포인트의 값이 관측되거나 측정되는 것의 평균 값보다 높은 표준 편차의 부호 수이다. 직관적으로 데이터셋의 표준 편차 및 평균과의 관계를 찾아 각 데이터 포인트를 설명한다. 이때, 평균은 0 이고 표준 편차는 1 인 데이트의 분포를 찾는다. Z-score 를 계산하는 동안 데이터의 크기를 조정하고 중심을 조정하며 0 에서 너무 먼 데이터 포인트는 특이치(outlier)로 처리된다. 대부분 이 영역은 -3 으로부터 3 까지로 임계값이 사용된다.

viii. Feature-selection

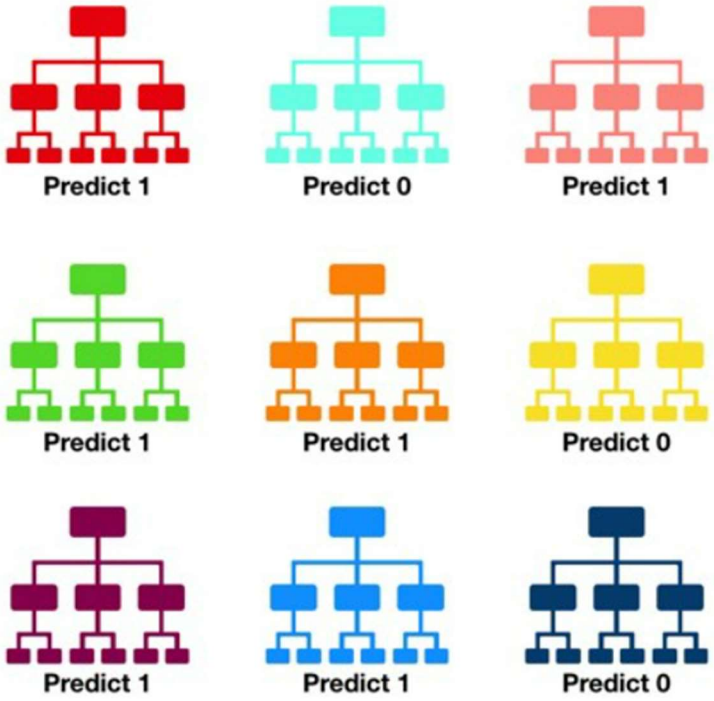
Feature selection 은 모델링 시 raw data 의 모든 feature 를 사용하는 것은 computing power 와 memory 측면에서 매우 비효율적이기 때문에 일부 필요한 feature 들만 선택해서 사용한다. 이 방법으로는 크게 3 가지로 나눌 수 있다. Feature 간 관련성을 측정하는 filter method, feature subset 의 유용성을 측정하는 wrapper method, 유사하지만 내장 metric 을 사용하는 embedded method 이다. 이때 훈련데이터에서 feature 를 고른다면 과적합 문제가 날 수 있으므로 데이터 분리가 중요하다.

B. 데이터 학습 방법

i. 머신 러닝

1. Random Forest

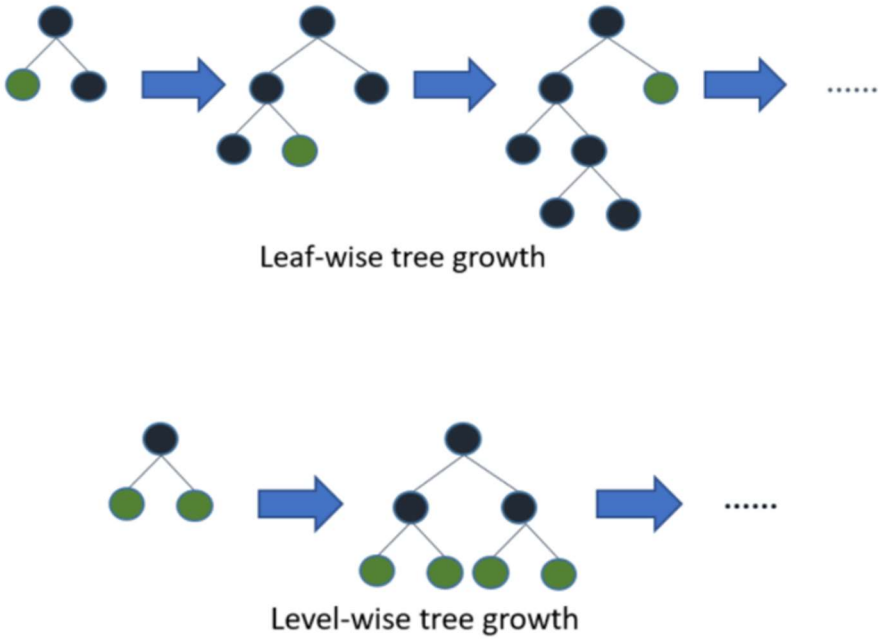
Decision Tree 는 over-fitting 될 가능성이 높다는 약점을 가진 반면 Random Forest 는 여러 개의 모델을 통과시키며 해당 문제를 예방할 수 있다. 따라서 각 모델 간 상대적 상관관계는 낮아야 한다.



<그림 2-2. 예측을 통한 Random Forest 모델 시각화>

2. LGBM(Light GBM)

Light GBM은 트리 기반의 학습 알고리즘인 gradient boosting 방식의 프레임 워크이다. LGBM은 나무(tree)를 수평으로 하는 다른 알고리즘과는 달리 수직으로 확장한다. Leaf-wise tree growth인 LGBM은 최대 delta loss가 증가하도록 잎의 개수를 정한다. Leaf-wise 알고리즘은 다른 level-wise 알고리즘보다 낮은 loss를 달성하는 경향이 있다. 데이터의 크기가 작은 경우 leaf-wise는 과적합(over-fitting)되기 쉬우므로 max-depth를 줄여줘야 한다.



<그림 2-3. LGBM>

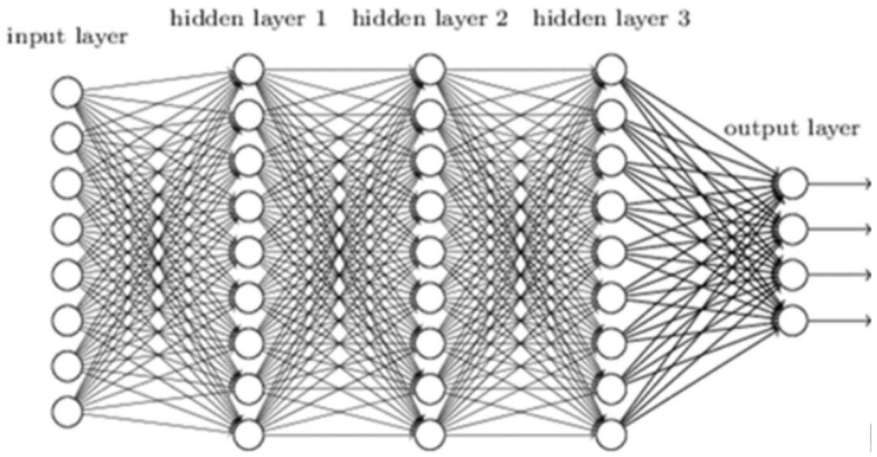
Light GBM은 속도가 빠른 것이 장점이며 메모리를 적게 차지하고 결과의 정확도가 높다. 또한 GPU를 활용할 수 있기 때문에 널리 사용되고 있다. 하지만 over-fitting에 민감하여 데이터의 크기가 작을 경우 기존의 머신 러닝 알고리즘이 더 좋을 수 있으므로 데이터 개수가 10,000개 이상일 때 사용하는 것이 유용하다.

ii. 딥러닝

1. DNN(Deep Neural Network)

심층신경망(Deep Neural Network, DNN) 입력층(input layer)과 출력층(output layer) 사이에 여러 개의 은닉층들로 이루어진 인공신경망(Artificial Neural Network, ANN)이다. DNN은 일반적인 인공신경망과 마찬가지로 복잡한 비선형 관계들을 모델링할 수 있다. 예를 들어 사물 식별 모델을 위한 심층 신경망 구조에서는 각 객체가 이미지 기본 요소들의 계층적 구성으로 표현될 수 있다. 이때 추가 계층들은 점진적으로 모여진 하위 계층들의 특징들을 종합할 수 있다. 이러한 특징은 ANN에 비해 더 적은 수의 노드들만으로도 복잡한 데이터를 모델링할 수 있게 해준다.

DNN의 장점으로는 변수 종류 상관없이 모두 분석 가능하며 입력 변수들 간의 비선형 조합이 가능하다. 예측력이 다른 머신 러닝 기법들에 비해 상대적으로 우수한 경우가 많으며 feature extraction이 자동적으로 수행된다. 반면 단점으로는 신경망이 복잡할수록 시간이 오래 걸리며 결과가 일정하지 않거나 가중치의 의미 때문에 결과 해석이 어렵다.



<그림 2-4. DNN>

DNN 모델링을 할 때 순환하는 과정을 몇 번 수행할지 정해주는 인자인 **Epoch**가 있다. 은닉 계층에 대한 활성화 함수는 네트워크 비선형성을 적용하는데 필요하다. 주로 signoid 함수가 많이 사용되는데 이는 0과 1사이에서 급격한 변화를 보여 최근에는 이를 보완한 **ReLU**(Rectified linear unit)함수를 자주 사용한다. 은닉 계층 노드를 무작위로 포기하는 드롭아웃(Dropout)을 적용한다.

iii. AutoML(Auto Machine Learning)

자동화된 머신 러닝(Auto Machine Learning, AutoML)은 시간 소모적이고 반복적인 기계 학습 모델 개발 작업을 자동화하는 프로세스이다. 데이터 전처리 과정에서부터 알고리즘 선택 및 튜닝까지의 과정에서 모델 개발자의 개입을 최소화하여 품질 좋은 모델을 효과적으로 개발할 수 있도록 연구되었다.

1. Auto-Sklearn

Auto-sklearn은 머신 러닝 사용자가 알고리즘 선택 및 하이퍼 파라미터 튜닝을 하지 않도록 만들어준다. 베이지안 최적화, 메타 학습 및 앙상블 구성의 최근 연구들을 반영한다.

2. Pycaret

Pycaret은 autoML을 하게 해주는 파이썬 라이브러리로 sklearn 패키지를 기반으로 하고 있으며, Classification, Regression, Clustering, Anomaly Detection 등등 다양한 모델을 지원한다.

3. AutoKeras

AutoKeras는 베이지안 최적화를 통해 효율적인 신경 구조 검색을 위한 네트워크 형태를 유도할 수 있는 프레임워크다. 검색 공간을 효율적으로 탐색하기 위해 신경망 커널과 트리 구조의 추천 함수 최적화 알고리즘이 들어있다. 현재 지원되는 기원으로는 이미지와 텍스트, 구조적 데이터의 classification, regression이 있고 현재 시계열 예측, 객체 검출, 이미지 segmentation이 개발 중에 있다.

Ⅲ. 연구 결과 및 분석

A. 데이터 분석

본격적으로 분석하기에 앞서 데이터의 특성(feature 종류, 통계, 결측치, 데이터 타입 등)을 확인한다.

	feature_1	feature_2	feature_3	...	feature_1557	feature_1558	Class
0	100	160	1.6000	...	0	0	0
1	20	83	4.1500	...	0	0	0
2	99	150	1.5151	...	0	0	0
3	40	40	1.0000	...	0	0	0
4	12	234	19.5000	...	0	0	0

[5 rows x 1559 columns]

<그림 3-1. 데이터 feature 종류>

	feature_1	feature_2	...	feature_1558	Class
count	1763.000000	1763.000000	...	1763.000000	1763.000000
mean	53.094158	126.587067	...	0.001134	0.081112
std	55.842014	129.859641	...	0.033672	0.273084
min	1.000000	1.000000	...	0.000000	0.000000
25%	12.000000	33.500000	...	0.000000	0.000000
50%	39.000000	96.000000	...	0.000000	0.000000
75%	75.000000	159.000000	...	0.000000	0.000000
max	640.000000	640.000000	...	1.000000	1.000000

<그림 3-2. 데이터 통계량>

	nan_count
feature_1	0
feature_2	0
feature_3	0
feature_4	0
feature_5	0
...	...
feature_1555	0
feature_1556	0
feature_1557	0
feature_1558	0
Class	0

1559 rows × 1 columns

<그림 3-3. 결측치 파악>

```
feature_1          int64
feature_2          int64
feature_3        float64
feature_4          int64
feature_5          int64
                    ...
feature_1555       int64
feature_1556       int64
feature_1557       int64
feature_1558       int64
Class              int64
Length: 1559, dtype: object
```

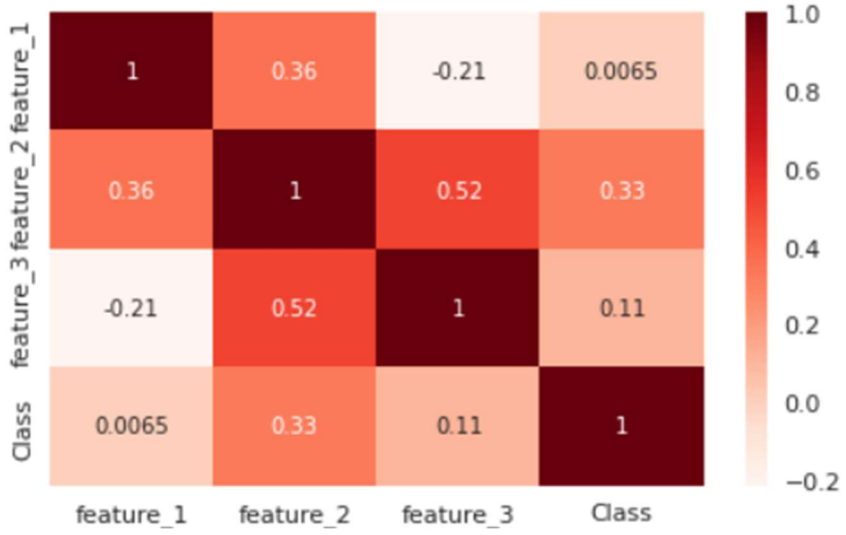
<그림 3-4. 데이터 타입>

해당 데이터는 feature1, feature2, feature3를 제외하고 모두 0과 1로 이루어진 이진 데이터로 구성되었음을 확인할 수 있다. 결측치는 따로 존재하지 않으며 feature3을 제외한 나머지는 모두 정수형(int64) 데이터를 갖는다.

```
0    1620
1     143
Name: Class, dtype: int64
```

<그림 3-5. Class 빈도 파악>

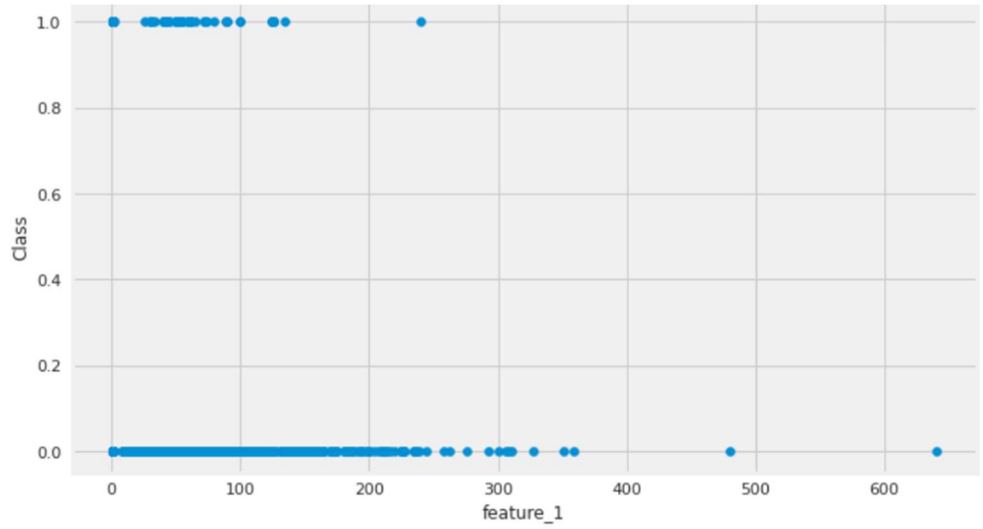
이때, 가장 큰 영향을 끼치는 Numeric data인 feature1, feature2, feature3와 class를 비교한다.



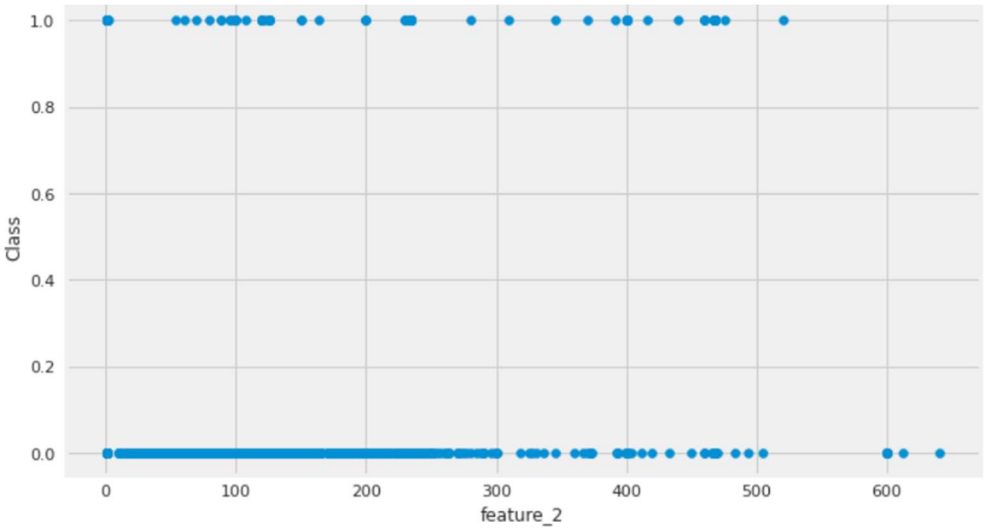
<그림 3-6. Feature1~3과 class 상관관계 비교>

그림에서 볼 수 있듯이 feature2와 class는 0.3으로 가장 약한 상관관계를 보인다.

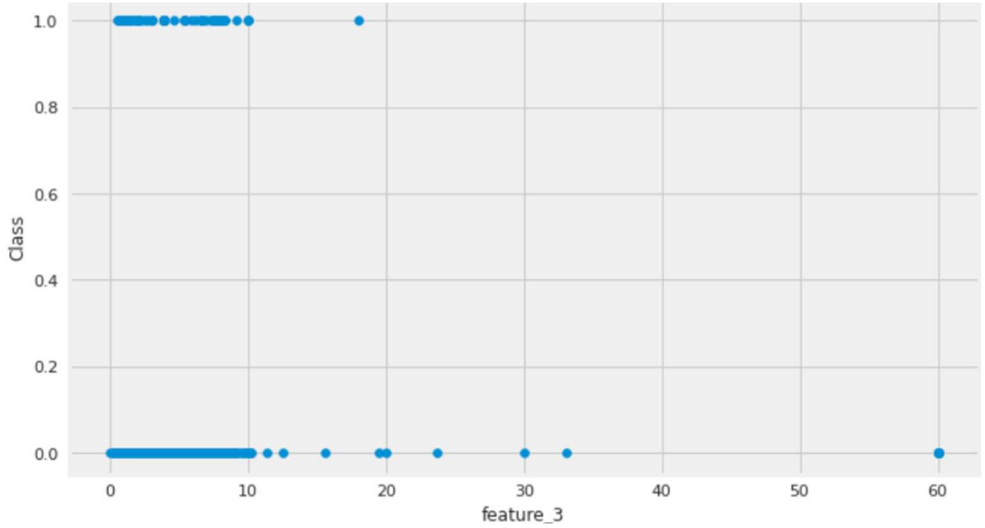
다음으로 각 feature 별로 분포도를 살펴본다.



<그림 3-7. Feature1 과 class>



<그림 3-8. Feature2 와 class>



<그림 3-9. Feature3 와 class>

분석된 데이터를 바탕으로 train 데이터와 test 데이터로 구분하여 train 데이터만으로 전처리를 진행하여 test 데이터와 결과를 비교한다.

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_13	feature_18	feature_21	feature_27
0	23.000000	26.000000	1.1304	1	0	0	0	0	0
1	31.000000	88.000000	2.8387	0	0	0	0	0	0
2	60.000000	128.983003	7.8000	1	0	0	0	0	0
3	55.974212	125.000000	1.0000	0	0	0	0	0	0
4	20.000000	134.000000	6.7000	1	0	0	0	0	0

5 rows × 394 columns

<그림 3-10. train 데이터>

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_13	feature_18	feature_21	feature_27
0	146.0	141.0	0.9657	0	0	0	0	0	0
1	1.0	1.0	2.0000	0	0	0	0	0	0
2	49.0	119.0	2.4285	1	0	0	0	0	0
3	1.0	1.0	2.0000	0	0	0	0	0	0
4	36.0	216.0	6.0000	1	0	0	0	0	0

5 rows × 394 columns

<그림 3-11. test 데이터>

가중치를 다르게 하여 데이터의 특이값을 검출한다.

```
Train 데이터 특이값 검출 (weight=1.1)
Feature_1 : Int64Index([  35,  163,  219,  250,  267,  269,  292,  369,  434,  470,  476,
              478,  491,  503,  504,  516,  542,  563,  583,  584,  624,  657,
              725,  751,  755,  772,  830,  909, 1016, 1029, 1064, 1077, 1085,
             1092, 1218, 1323, 1332, 1343, 1377, 1386, 1407],
            dtype='int64')
Feature_2 : Int64Index([  35,  219,  250,  267,  269,  434,  470,  504,  624,  725,  751,
              830, 1029, 1085, 1218, 1332, 1343, 1377, 1407],
            dtype='int64')
Feature_3 : Int64Index([478, 583, 1064], dtype='int64')

Test 데이터 특이값 검출 (weight=1.1)
Feature_1 : Int64Index([3, 20, 31, 170, 174, 189, 195, 197, 239, 278, 296, 330, 348, 350], dtype='int64')
Feature_2 : Int64Index([], dtype='int64')
Feature_3 : Int64Index([278], dtype='int64')
```

<그림 3-12. train 데이터와 test 데이터 특이값 검출(weight=1.1)>

```
Train 데이터 특이값 검출 (weight=0.3)
Feature_1 : Int64Index([   6,   13,   18,   74,   77,   80,   81,  130,  134,  143,  153,
             158,  166,  167,  184,  188,  189,  220,  234,  248,  266,  279,
             360,  366,  389,  396,  429,  441,  451,  457,  484,  492,  550,
             552,  558,  580,  661,  692,  717,  793,  852,  856,  857,  863,
             865,  879,  886,  923,  937,  945,  958,  989, 1010, 1024, 1027,
            1036, 1039, 1048, 1056, 1069, 1070, 1088, 1095, 1103, 1152, 1195,
            1219, 1248, 1258, 1273, 1295, 1305, 1327, 1350, 1352, 1363, 1368,
            1371, 1375, 1392, 1398],
           dtype='int64')
Feature_2 : Int64Index([  15,   48,   54,   59,   81,   87,   92,  111,  114,  116,  150,
             152,  156,  167,  174,  192,  199,  200,  247,  256,  277,  285,
             297,  314,  333,  335,  360,  372,  389,  431,  451,  466,  472,
             473,  482,  487,  490,  498,  499,  506,  522,  530,  533,  539,
             543,  554,  589,  600,  617,  635,  653,  664,  665,  674,  692,
             716,  745,  746,  789,  832,  833,  834,  839,  846,  870,  908,
             946,  997, 1028, 1050, 1052, 1053, 1068, 1069, 1160, 1164, 1178,
            1182, 1193, 1199, 1210, 1246, 1248, 1253, 1277, 1324, 1359, 1360,
            1403, 1409],
           dtype='int64')
Feature_3 : Int64Index([   9,   15,   37,   38,   44,   48,   54,   59,   83,   86,
             ...
            1324, 1325, 1354, 1359, 1396, 1400, 1401, 1403, 1404, 1409],
           dtype='int64', length=173)
```

<그림 3-13. train 데이터 특이값 검출(weight=0.3)>

```
Test 데이터 특이값 검출 (weight=0.3)
Feature_1 : Int64Index([10, 21, 61, 71, 121, 130, 135, 139, 140, 146, 180, 235, 261, 263,
            264, 304, 344],
           dtype='int64')
Feature_2 : Int64Index([  2,  34,  35,  51,  63,  72,  84, 131, 133, 146, 158, 163, 165,
            172, 198, 202, 203, 232, 244, 272, 279, 280, 287, 311, 314, 333],
           dtype='int64')
Feature_3 : Int64Index([ 17,  33,  49,  63,  92, 117, 124, 165, 178, 206, 208, 218, 232,
            267, 290, 298, 302, 313, 340, 342, 349],
           dtype='int64')
```

<그림 3-13. test 데이터 특이값 검출(weight=0.3)>

데이터 특이값 수를 나타내면 다음과 같다.

```
Train 데이터 특이값 수(weight=0.3)
Feature_1 : 81
Feature_2 : 90
Feature_3 : 173

Train 데이터 특이값 수(weight=1.1)
Feature_1 : 41
Feature_2 : 19
Feature_3 : 3

Test 데이터 특이값 수(weight=0.3)
Feature_1 : 17
Feature_2 : 26
Feature_3 : 21

Test 데이터 특이값 수(weight=1.1)
Feature_1 : 14
Feature_2 : 0
Feature_3 : 1
```

<그림 3-15. train 데이터와 test 데이터 특이값 수 비교>

분석된 데이터를 바탕으로 데이터에 스케일링을 진행한다.

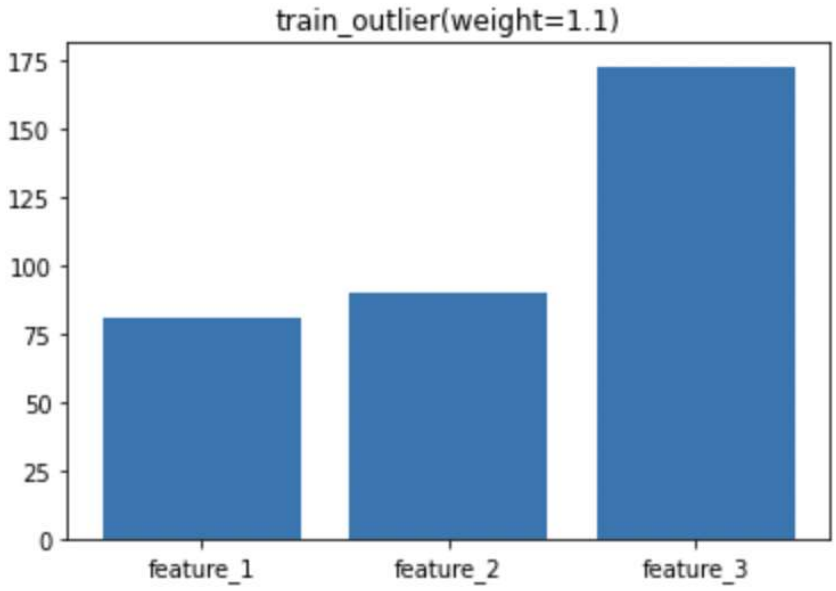
```
Train 데이터 MaxAbsScaler 적용 전
       feature_1   feature_2  feature_3
0     146.000000  141.000000     0.9657
1       1.000000    1.000000     2.0000
2      49.000000  119.000000     2.4285
3       1.000000    1.000000     2.0000
4      36.000000  216.000000     6.0000
...          ...         ...        ...
1405   52.000000  144.000000     2.7692
1406   60.000000  468.000000     7.8000
1407   52.352559  125.987000     2.0000
1408   31.000000   88.000000     2.8387
1409   60.000000  125.987234     7.8000

[1410 rows x 3 columns]
----------------------------------
Train 데이터 MaxAbsScaler 적용
[[0.97986577 0.27115385 0.016095   ... 0.         0.         0.        ]
 [0.00671141 0.00192308 0.03333333 ... 0.         0.         0.        ]
 [0.32885906 0.22884615 0.040475   ... 0.         0.         0.        ]
 ...
 [0.35135946 0.24228269 0.03333333 ... 0.         0.         0.        ]
 [0.20805369 0.16923077 0.04731167 ... 0.         0.         0.        ]
 [0.40268456 0.24228314 0.13       ... 0.         0.         0.        ]]
```

<그림 3-16. train 데이터 MaxAbsSclaer 적용 전/후>

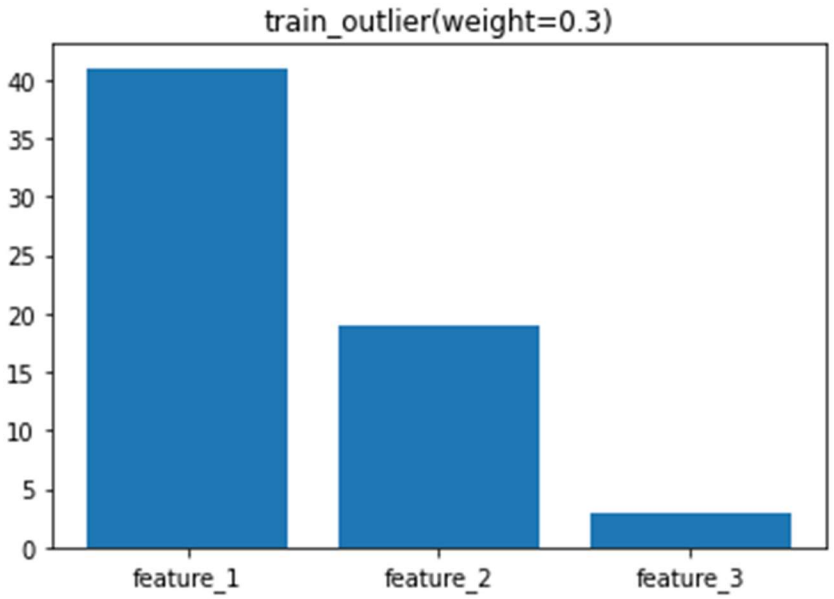
```
Test 데이터 MaxAbsScaler 적용 전
     feature_1   feature_2  feature_3
0    23.000000   26.000000   1.130400
1    31.000000   88.000000   2.838700
2    60.000000  128.983003   7.800000
3    55.974212  125.000000   1.000000
4    20.000000  134.000000   6.700000
..         ...         ...        ...
348  55.974212    1.000000   2.000000
349  11.000000   96.000000   3.380019
350  55.974212  120.000000   0.967700
351  43.000000  230.000000   5.348800
352   1.000000    1.000000   2.000000

[353 rows x 3 columns]
----------------------------------
Test 데이터 MaxAbsScaler 적용 후
[[0.14935065 0.05555556 0.11304    ... 0.         0.         0.        ]
 [0.2012987  0.18803419 0.28387    ... 0.         0.         0.        ]
 [0.38961039 0.27560471 0.78       ... 0.         0.         0.        ]
 ...
 [0.36346891 0.25641026 0.09677    ... 0.         0.         0.        ]
 [0.27922078 0.49145299 0.53488    ... 0.         0.         0.        ]
 [0.00649351 0.00213675 0.2        ... 0.         0.         0.        ]]
```

<그림 3-17. test 데이터 MaxAbsSclaer 적용 전/후>

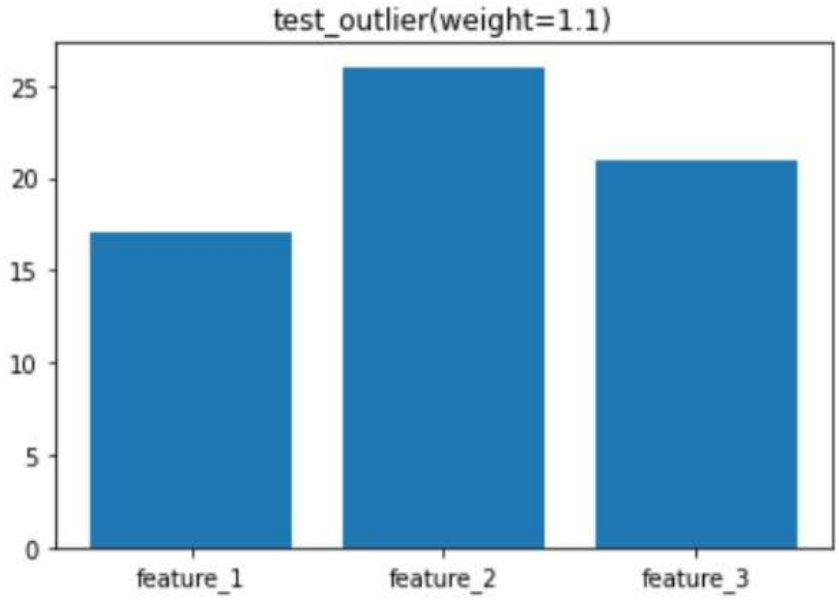
가중치에 따른 feature1~3 의 결측치를 그래프로 시각화하면 다음과 같다.



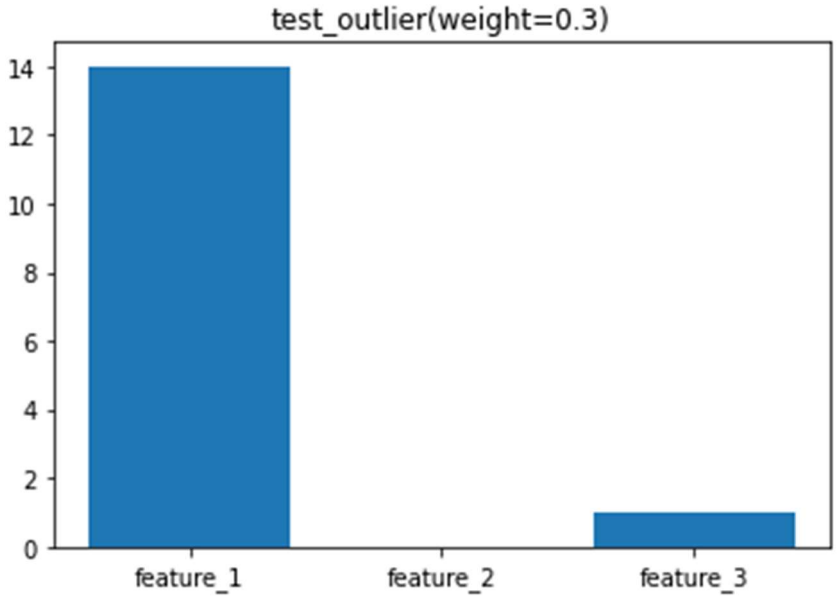
<그림 3-18. train 데이터 결측치(weight=1.1)>



<그림 3-19. train 데이터 결측치(weight=0.3)>



<그림 3-20. test 데이터 결측치(weight=1.1)>



<그림 3-21. test 데이터 결측치(weight=0.3)>

B. 데이터 학습

데이터 분석을 바탕으로 데이터 전처리를 가장 먼저 진행한다. 가장 기본적인 데이터 전처리를 바탕으로 가장 최적화된 전처리를 조사함과 더불어 머신 러닝 학습 모델을 선정하여 데이터를 학습시킨다. 이때 이번 연구는 머신 러닝 학습 모델은 LGBM과 Random Forest 두 가지로 진행하며 전처리 방식에 따른 성능 변화에 더 초점을 둔다.

가장 기본적으로 적용된 데이터 전처리는 아래와 같다.

1. Remove_Collinear_features(Threshold = 0.6)
2. MaxAbs Normalization
3. Smote Over-sampling(Random_state = 88)

<표3-1. 기본적인 데이터 전처리>

i. 머신 러닝

1) LGBM

기본적인 전처리가 된 데이터를 바탕으로 LGBM 방식과 Random Forest 방식을 비교한다.

	LGBM	Random Forest
ROC_AUC	0.862	0.901

<표 3-2. LGBM 과 Random Forest 비교>

Random Forest 모델이 LGBM 보다 우수한 성능을 보이므로 다음 과정은 Random Forest 를 통해 연구를 진행한다.

2) Random Forest(RF)

데이터에서 이상치를 단순히 제거한 것과 각 Feature 의 평균값을 대입하는 결과를 비교해본다.

	기본 전처리	이상치 단순 제거 (가중치 = 1.5)	이상치 제거 후 평균값 (가중치 = 1.5)
ROC_AUC	0.901	0.896	0.904

<표 3-3. 이상치 제거 결과에 대한 비교>

위 결과를 통해 데이터에 이상치를 제거하여 평균값을 대입하는 것이 성능을 다소 향상시킨다는 결론을 얻어냈다. 그렇다면 이상치 제거 가중치를 변경한 결과에 대해 한 번 더 성능 비교를 해본다.

	가중치 = 2	가중치 = 1.5	가중치 = 1
ROC_AUC	0.910	0.904	0.911

<표 3-4. 이상치 제거 가중치 비교>

정리를 간단하게 하자면 기본 전처리한 데이터를 이상치 제거(가중치 1) 후 평균값을 대입하면 성능이 향상되는 결과를 얻었다.

다음으로 평균값 대입을 Class 별로 0 과 1, 두 가지로 구분할 경우에 대해 비교했다.

	이상치 제거 후 평균값 (가중치 = 1)	Class 를 0 과 1 로 구분하여 이상치 제거 후 각각 평균값
ROC_AUC	0.911	0.912

<표 3-5. LGBM 과 Random Forest 비교>

Class 를 두 가지로 나누었을 때 좋은 결과를 얻어냈고 이때 가중치를 1 로 동일하게 처리하는 것이 아니라 각각 다르게 처리하면 성능이 향상될 것으로 기대했다.

	같은 가중치 (class0, 1: 1)	다른 가중치 (class 0: 1.1, class 1: 0.3)
ROC_AUC	0.912	0.914

<표 3-6. LGBM 과 Random Forest 비교>

이 결과로 class 를 구분하여 가중치를 어떻게 넣어주는지에 따라 성능이 변화되는 것을 얻어냈고 몇 번의 시행착오를 겪는다. 또한 파라미터 값을 조정해가면서 최종적으로 가장 좋은 성능을 구현해낸 결과는 다음과 같다.

- Class 가중치 평균값 조정
- n_estimators=131
- max_features='auto'
- min_samples_leaf=8
- min_samples_split=5

<표3-7. 최종 평가지표에 대한 과정>

위와 같은 과정 끝에 최종적으로 얻어낸 ROC_AUC Score 는 **'0.91821465'** 이다.

ii. 딥러닝

1. DNN(Deep Neural Network)

제공된 wafer_data에 SS(StandardScaler) 전처리만 진행한 데이터로 파라미터를 고정하여 각 설정별로 비교를 진행한다.

고정된 파라미터: Epoch=50, LR=0.002, Batch Size=64, Batch Nomalization와 Dropout 적용, Dropout=0.2, Optimizer=Adam, criterion=BCEWithLogitcLoss, Network Layer = (1558,1600)/(1600,800)/(800,1), 활성화함수: ReLU

1) 가중치 초기값 결정

Weight 초기값만 변경해주며 10번의 결과를 내어 평균치를 얻어낸다.

	Xavier init	Kaiming_Normal init	Normal init
ROC_AUC	0.658	0.53	0.67
ACC	0.901	0.912	0.913

<표 3-8. Wegiht 결정 비교표>

평가지표 중 ROC_AUC에 초점을 맞춰 초기값은 Normal init으로 고정한다.

2) Epoch 결정과정

이번에는 Epoch만 변경하여 10번의 결과를 내어 평균치를 얻어낸다.

	Epoch30	Epoch50	Epoch100	Epoch150
ROC_AUC	0.6	0.67	0.66	0.63
Acc	0.9	0.91	0.88	0.91

<표 3-9. Epoch 결정 비교표>

평가지표 중 ROC_AUC에 초점을 맞춰 Epoch는 50으로 고정한다.

3) 신경망 은닉층 개수 별 노드 수 결정과정

[은닉층 1개]	(1558 1600 800 1)	(1558 3200 1600 1)	(1558 800 400 1)
ROC_AUC	0.670	0.650	0.68
ACC	0.913	0.910	0.915

<표 3-10. 은닉층 1개 비교표>

은닉층이 1개인 노드는 (1558 800 400 1)가 가장 좋은 성능을 보인다.

[은닉층 2개]	(1558 3200 3200 800 1)	(1558 1600 3200 800 1)	(1558 3200 6400 1600 1)
ROC_AUC	0.672	0.66	0.653
ACC	0.911	0.910	0.910

<표 3-11. 은닉층 2개 비교표>

은닉층이 2개인 노드는 (1558 3200 3200 800 1)가 가장 좋은 성능을 보인다.

[은닉층 3개]	(1558 1600 3200 6400 1600 1)	(1558 3200 6400 6400 1600 1)	(1558 3200 6400 10000 2000 1)
ROC_AUC	0.66	0.69	0.67
ACC	0.89	0.92	0.92

<표 3-12. 은닉층 3개 비교표>

은닉층이 3개인 노드는(1558 3200 6400 6400 1600 1)가 가장 좋은 성능을 보인다.

따라서 가장 좋은 성능을 보일 것으로 예측되는 모델은 (1558 3200 6400 6400 1600 1)이다.

4) Layer 결정과정

전처리한 데이터로 딥러닝 신경망 1층, 2층, 3층을 통과시켜 성능을 비교한다.

	1층	2층	3층
ROC_AUC	0.71086	0.71844	0.74525
ACC	0.89501	0.89031	0.86402

<표 3-13. Layer 결정 비교표>

iii. AutoML

해당 방법의 장점은 자동 파라미터 튜닝과 평가지표를 통한 시각화이다. 다만 학습할 때마다 최적화된 파라미터가 변화된다. 다음은 최고 성능일 때의 파라미터이다.

	Parameters
bootstrap	True
ccp_alpha	0.0
class_weight	balanced_subsample
criterion	gini
max_depth	7
max_features	sqrt
max_leaf_nodes	None
max_samples	None
min_impurity_decrease	0.002
min_impurity_split	None
min_samples_leaf	5
min_samples_split	5
min_weight_fraction_leaf	0.0
n_estimators	270
n_jobs	-1
oob score	False

<그림 3-14. 최고 성능일 때 파라미터>

이때 얻어낸 결과에 대해 일부 조건들을 변화시키며 성능을 비교한다. 우선 feature selection threshold를 0.95로 고정시키고 outlier threshold를 변화시켜본다.

Outlier threshold	0.03	0.04	0.05	0.06
ROC_AUC	0.908	0.908	0.905	0.910

Outlier threshold	0.07	0.08	0.09
ROC_AUC	0.901	0.903	0.903

<표 3-14. Outlier threshold 비교>

Outlier threshold는 0.06일 때 성능이 가장 높게 나왔다. 이번에는 outlier threshold를 0.06으로 고정시켜 Feature selection threshold를 변화시키며 결과를 비교한다.

Outlier threshold	0.95	0.85	0.75	0.65	0.55
ROC_AUC	0.910	0.910	0.910	0.910	0.910

Feature selection threshold	0.45	0.35	0.25	0.15	0.05
ROC_AUC	0.910	0.910	0.908	0.909	0.914

<표 3-15. Feature selection threshold 비교>

즉, outlier threshold는 0.06, feature selection threshold 0.05로 고정하여 학습을 진행한다. 파라미터 중 다른 값들을 변경시키며 성능 향상시킨 결과들만 하단에 정리해보겠다.

<ROC_AUC>

- Remove perfect collinearity 적용 = 0.91436
  (상관관계가 1인 데이터 제거)
- 'classic'에서 'boruta'로 변경 = 0.91437
  (Feature selection method 변경)
- Feature interaction 적용 = 0.91919
  (H-statistic, 상호작용 강도 측정 예측)

<표 3-16. 최종 성능 평가>

Ⅳ. 결론

본 연구에서는 반도체 제품의 불량 여부 예측과 성능 향상에 대한 연구를 진행했다. 해당 연구는 크게 데이터 전처리와 딥러닝, 머신 러닝으로 구분 지어 최적의 성능을 얻기 위한 과정을 소개한다.

해당 데이터는 3 개의 numeric 데이터와 1,555 개의 binary 데이터로 이루어져 있다. 핵심이 되는 전처리 방법으로는 feature selection, maxabs normalization, smote oversampling, 세 가지라 할 수 있다. 1 천여개가 넘는 feature 를 우선적으로 줄이고 각 데이터들의 값을 조정하여 다차원의 데이터를 비교 분석하기 쉽게 만들었다. 또한 공분산 행렬의 조건 수를 감소시켜 최적화 과정에서 안정성 및 수렴 속도를 향상시킬 수 있었다.

다각도의 시행착오를 통해, 단순히 이상치를 제거하는 것이 아닌 각 feature 평균값을 대입하며 더 좋은 결과를 얻을 수 있었다. 탐색 과정에서 가중치는 높은 수가 아닌 1 일 때 가장 좋은 성능을 낸 결과도 의외라 할 수 있다. 이를 통해 데이터 상태에 따라 최적화된 전처리 방법이 다르며 상황에 맞게 다른 방향으로 해석해야 함을 알 수 있었다. 같은 모델에서 향상된 결과를 보아 전처리의 중요성을 여실히 깨달을 수 있던 계기가 되었다.

가장 좋은 성능을 내는 전처리 데이터를 바탕으로 학습 모델을 딥러닝과 머신 러닝으로 나누어 연구를 진행했다. 학습 결과에 있어 정확도를 위해 항상 10 번의 평균으로 성능 평가를 했다. 데이터가 단순하여 이번 연구에서는 머신 러닝이 딥러닝보다 성능이 뛰어날 것을 예상했고 딥러닝에 대한 연구를 우선순위에 두었다.

딥러닝 프레임워크 중 pytorch 를 이용하여 classification 을 진행했다. 딥러닝에서 성능에 있어 중요한 지표는 초기값, epoch, 신경망 은닉층 개수와 노드 수다. 초기값에 있어 Xavier init, Kaiming normal init, Normal init 중에서 Normal init 이 가장 좋은 성능을 보였고, epoch 는 50 일 때 가장 높은 성능을 나타냈다. 은닉층은 3 개일 때 (1558, 3200, 6400, 6400, 1600, 1)일 때 최종 0.92 의 ROC_AUC score 를 얻을 수 있었다. 비록 선순위를 딥러닝으로 두었으나 동시에 진행한 머신 러닝에서보다 낮은 결과를 보여 여기서 마무리하였다.

앞서 설명한 maxabs normalization, remove collinear features, 그리고 smote osversampling 을 기본적으로 적용하여 실험을 통해 remove collinear features threshold 는 0.7, smote oversampling random state 는 425 로 고정하였다. 머신 러닝에서 강력하다고 알려진 LGBM 과 Random Forest(RF)에 대한 비교에서 RF 가 우수한 결과를 보였다. Class 를 나누어 이상치 제거 후 평균값을 대입한 것이 그렇지 않은 것보다 성능이 더 좋았는데 이는 feature2 의 경우 class 별로 feature 평균 차이가 꽤 나기 때문이라 예상된다. 특히 이 때 class1 에 가중치 0.3 으로 변화를 준 결과를 통해 특이값을 더 민감하게 검출한 결과가 성능 향상에 도움이 됐다. 마지막으로 n_estimators 와 max_features 와 같은 파라미터 최적화를 통해 최종 0.9309 의 ROC_AUC score 를 얻을 수 있었다.

추가로 진행한 연구 중에 AutoML 은 처음부터 끝, 알고리즘 선택부터 모델의 hyper-parameter 조정, 모델링 반복 및 평가 등 모든 것을 자동화해주는 방법이다. 성능은 좋게 나오지만 이는 본 연구의 목적에는 적합하지 않다고 판단했다. 하지만 AutoML 을 통해 전처리 방식이나 파라미터에 대한 해법을 얻을 수는 있었다.

머신 러닝이 우리가 조사한 인공지능 중 가장 좋은 성능을 낼 수 있던 이유로는 sklearn 이 사용하기 매우 쉽다는 점과 다양한 전처리기법을 적용시킬 수 있기 때문이라 판단했다. 특히 본 연구에서 사용된 wafer data 의 데이터 분포가 간단하다는 점, 그리고 데이터 개수와 복잡성이 그리 크지 않다는 점이 가장 적합한 결과를 보인다는 결론을 내렸다.

Ⅴ. 참고 문헌

- 김건우, 염상준, “펭귄브로의 3 분 딥러닝, 파이토치맛”, PyTorch 코드로 맛보는 CNN, GAN, RNN, DQN, Autoencoder, ResNet, Seq2Seq, Adversarial Attack, 한빛미디어, 2019
- 커넥트재단, “파이토치로 시작하는 딥러닝 기초”, 모두의 연구소, 2021
- 공학 석사학위 논문, “반도체 생산 장비의 예측정비에 관한 빅데이터 분석 사례, 박준태(2020 년 2 월)”, p1~7
- 권철민, “파이썬 머신러닝 완벽 가이드”, 위키북스, 2019
- “PCA”, https://bskyvision.com/867
- “LGBM”, https://injo.tistory.com/48
- “Microsoft LightGBM Documentation”, accessed November 28, 2021, https://lightgbm.readthedocs.io/en/latest.
- “pytorch”, accessed November 25, 2021, https://pytorch.org/docs/stable/nn.init.html.
- “SMOTE”, https://john-analyst.medium.com/smote%EB%A1%9C-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EB%B6%88%EA%B7%A0%ED%98%95-%ED%95%B4%EA%B2%B0%ED%95%98%EA%B8%B0-5ab674ef0b32
- Yeo-Johnson Power Transformations, Sanford Weisberg, 2021 년 10 월(미네소타 대학)
- Guolin Ke, et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree.” Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.
- “DNN”, https://m.blog.naver.com/tjdudwo93/221072421443
- scikit-learn, “MaxAbsScaler”, last modified November, 2015, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html
- Wikipedia, “Power transform”, last modified 11 October, 2021 https://en.wikipedia.org/wiki/Power_transform#Yeo%E2%80%93Johnson_transformation
- Jim Frost, “Multicollinearity in Regression Analysis: Problems, Detection, and Solutions”, e-book, “Remove multicollinearity”, https://statisticsbyjim.com/regression/multicollinearity-in-regression-analysis/
- “Feature selection”, https://wooono.tistory.com/249
- Wikipedia, “Feature selection”, last modified 10 December, 2021 https://en.wikipedia.org/wiki/Feature_selection
- Isabelle Guyon, Andre Elisseeff, 2003, An Introduction to Variable and Feature Selection, Journal of Machine Learning Research 3, 3rd edition
- Towards data science, “Ways to Detect and Remove the Outliers”, edited May 22, 2018, https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba

Ⅵ. 부록(소감)

해당 과제를 통해 수업 이상의 보람과 깨달음을 얻었다. 다른 데이터와 달리 해당 wafer 데이터셋은 특이하여 전처리 과정이 힘들었고, 힘든 만큼 데이터 학습 모델뿐 아니라 전처리 과정이 매우 중요함을 느꼈다. 전처리에는 배운 내용 이외에도 많은 기법들이 있었고 다양하게 처리를 한다고 반드시 성능이 향상되지는 않았다. 데이터를 분석하고 시각화하여 시행착오를 통해 데이터 유형과 feature 특성에 맞는 방식을 택하는 것이 가장 중요했다.

전처리 후에는 수업 시간에 배운 다양한 모델들로 데이터 학습을 진행했다. 머신 러닝에서는 수업 시간에 배운 sklearn 을 이용하여 진행했고 딥러닝은 pytorch, 또 조사 중에 알게 된 autoML 까지 사용해봤다. 특히 autoML 의 경우 데이터 전처리부터 알고리즘, 튜닝까지 대부분의 과정에서 자동화되어 모델 개발에 대한 편의성을 제공해주었다. 여러 가지 모델을 짧은 코드로 구현할 수 있었고 모델 간 성능 비교도 용이했다. 덕분에 효과 좋은 전처리 및 튜닝된 파라미터 등에 대한 정보를 얻을 수 있었다. 하지만 이 모델의 경우 매번 파라미터와 알고리즘이 변하는 탓에 해당 연구 결과물로 제출하기에는 어려웠다. 다음으로 진행한 딥러닝 연구를 통해, 간단한 classification 문제 해결 측면에서는 sklearn 이 pytorch 보다 훨씬 잘 짜여진 라이브러리라는 것을 체감했다.

해당 IC-PBL 을 통해 데이터를 처리하고 모델을 통해 학습시키기까지 프로세스에 대해 배워보는 좋은 기회가 되었다. 또한 인공지능에 대한 이해도가 향상되었다. 굉장히 각광받던 autoML 을 접할 수 있어 신기했고, 해당 모델 내에서 전처리 기법의 제한으로 인해 로컬 환경에서 돌려본 모델이 더 좋은 결과를 내어 다소 뿌듯함을 느낀다.