LG Aimers Phase 2

스마트 공장 제품 품질 상태 분류

참가자: 얄루링

발표

CONTNENTS

01

1. 개발 및 분석 환경

1. 개발 및 분석 환경

Python Version

Python 3.7.13

기타 Library Version

Scikit-learn 1.0.2

pandas 0.3.1

numpy 1.19.5

fancyimpute 0.7.0

런타임 유형: CPU

02

2. EDA

2-1. 데이터 확인

2-2. 주요 데이터 시각화

2-1. 데이터 확인

초기 데이터

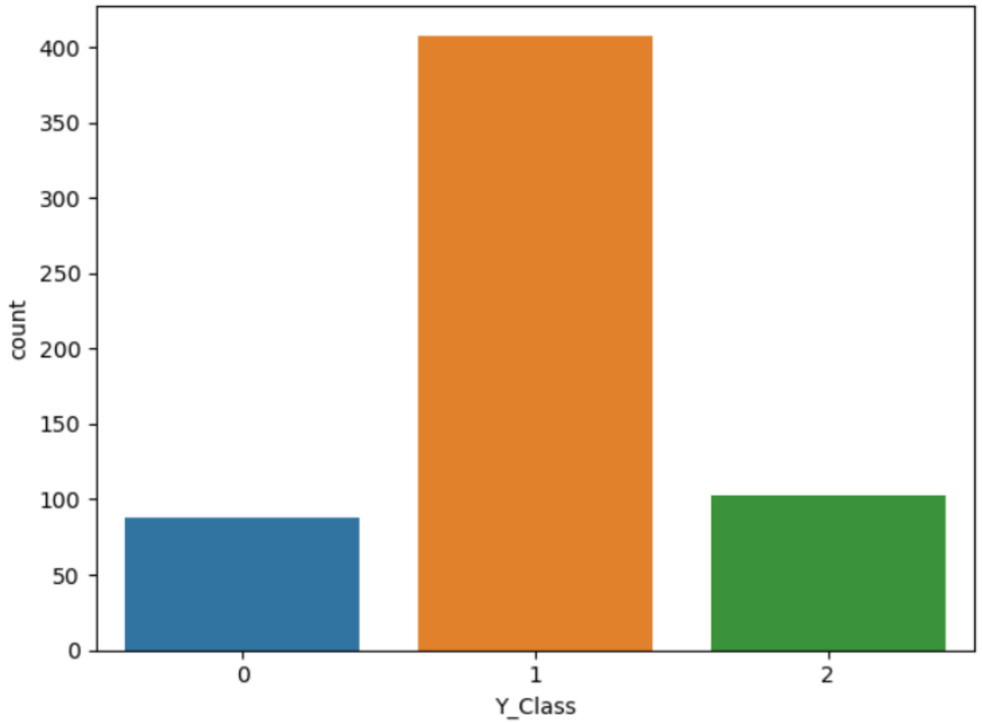
598 rows × 2880 columns

	PRODUCT_ID	Y_Class	Y_Quality	LINE	PRODUCT_CODE	X_1	X_2	X_3	X_4	X_5	...	X_2866	X_2867	X_2868	X_2869	X_2870	X_2871	X_2872	X_2873	X_2874	X_2875
0	TRAIN_000	1	0.533433	T050304	A_31	NaN	NaN	NaN	NaN	NaN	...	39.34	40.89	32.56	34.09	77.77	NaN	NaN	NaN	NaN	NaN
1	TRAIN_001	2	0.541819	T050307	A_31	NaN	NaN	NaN	NaN	NaN	...	38.89	42.82	43.92	35.34	72.55	NaN	NaN	NaN	NaN	NaN
2	TRAIN_002	1	0.531267	T050304	A_31	NaN	NaN	NaN	NaN	NaN	...	39.19	36.65	42.47	36.53	78.35	NaN	NaN	NaN	NaN	NaN
3	TRAIN_003	2	0.537325	T050307	A_31	NaN	NaN	NaN	NaN	NaN	...	37.74	39.17	52.17	30.58	71.78	NaN	NaN	NaN	NaN	NaN
4	TRAIN_004	1	0.531590	T050304	A_31	NaN	NaN	NaN	NaN	NaN	...	38.70	41.89	46.93	33.09	76.97	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
593	TRAIN_593	1	0.526546	T100306	T_31	2.0	95.0	0.0	45.0	10.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
594	TRAIN_594	0	0.524022	T050304	A_31	NaN	NaN	NaN	NaN	NaN	...	49.47	53.07	50.89	55.10	66.49	1.0	NaN	NaN	NaN	NaN
595	TRAIN_595	0	0.521289	T050304	A_31	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
596	TRAIN_596	1	0.531375	T100304	O_31	40.0	94.0	0.0	45.0	11.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
597	TRAIN_597	1	0.533702	T100306	O_31	21.0	87.0	0.0	45.0	10.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- 초기 제공된 데이터는 데이터 수는 적고 피쳐 수가 아주 많은 특징을 지님
- 또한, 다수의 NaN 데이터도 눈에 띔

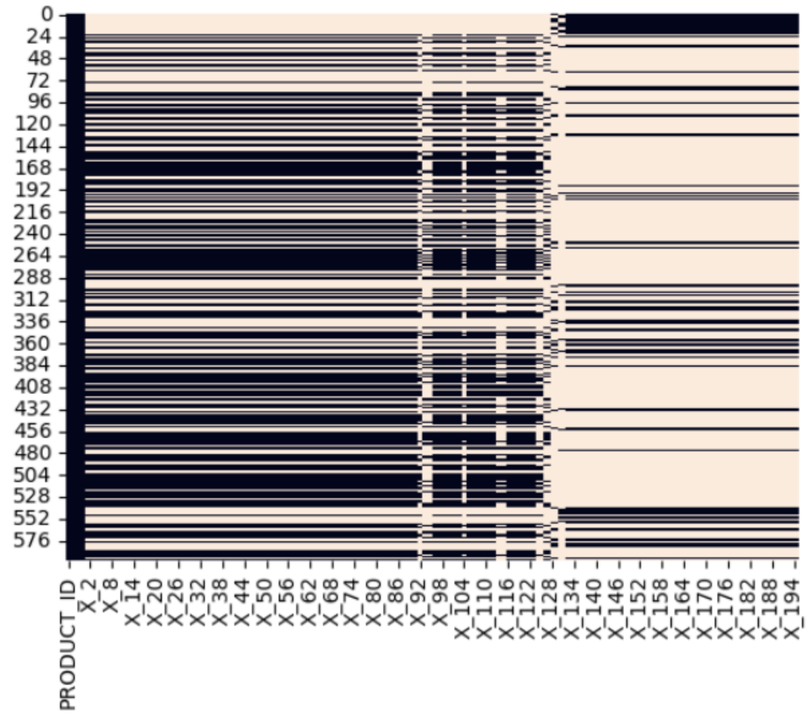
2-2. 주요 데이터 시각화

Y_Class 시각화



- 0/1/2 세 가지 class의 Mulit-classification이며 상당한 비율의 Class 불균형을 보임

결측치 시각화



- 검은색이 아닌 부분이 결측인 부분인데, 상당한 수의 결측을 확인할 수 있음

03

3. Feature Engineering

3-1. Feature Pre-Processing

3-2. Feature Engineering

3-1. Feature Pre-Processing

```
train_T050304 = train[train['LINE']=='T050304'] # 78
train_T050307 = train[train['LINE']=='T050307'] # 42
train_T100304 = train[train['LINE']=='T100304'] # 175
train_T100306 = train[train['LINE']=='T100306'] # 174
train_T010306 = train[train['LINE']=='T010306'] # 70
train_T010305 = train[train['LINE']=='T010305'] # 59

test_T050304 = test[test['LINE']=='T050304']
test_T050307 = test[test['LINE']=='T050307']
test_T100304 = test[test['LINE']=='T100304']
test_T100306 = test[test['LINE']=='T100306']
test_T010306 = test[test['LINE']=='T010306']
test_T010305 = test[test['LINE']=='T010305']
```

1. 라인별 분리

```
for col in train_T050304.columns:
    if pd.notnull(train_T050304[col]).sum()==0:
        notwith1.append(col)

for col in train_T050307.columns:
    if pd.notnull(train_T050307[col]).sum()==0:
        notwith2.append(col)

for col in train_T100304.columns:
    if pd.notnull(train_T100304[col]).sum()==0:
        notwith3.append(col)

for col in train_T100306.columns:
    if pd.notnull(train_T100306[col]).sum()==0:
        notwith4.append(col)

for col in train_T010306.columns:
    if pd.notnull(train_T010306[col]).sum()==0:
        notwith5.append(col)

for col in train_T010305.columns:
    if pd.notnull(train_T010305[col]).sum()==0:
        notwith6.append(col)
```

2. 라인 내 사용하지 않는 (전부 결측값인) feature 제거

3-1. Feature Pre-Processing

```
train_T05 = pd.concat([train_T050304, train_T050307])
train_T10 = pd.concat([train_T100304, train_T100306])
train_T01 = pd.concat([train_T010306, train_T010305])

test_T05 = pd.concat([test_T050304, test_T050307])
test_T10 = pd.concat([test_T100304, test_T100306])
test_T01 = pd.concat([test_T010306, test_T010305])
```

3. 총 6라인 중 2라인씩 거의 유사한 feature를 지니고 있는 것을 확인해 소수의 feature 제외하고 concat 후 총 3개의 데이터셋으로 구성

```
le1 = LabelEncoder()
for x in [train_T01, train_T05, train_T10]:
    le1 = le1.fit(x['PRODUCT_CODE'])
    x['PRODUCT_CODE'] = le1.transform(x['PRODUCT_CODE'])

for t in [test_T01, test_T05, test_T10]:
    for label in np.unique(t['PRODUCT_CODE']):
        if label not in le1.classes_:
            le1.classes_ = np.append(le1.classes_, label)
    t['PRODUCT_CODE'] = le1.transform(t['PRODUCT_CODE'])
print('CODE encode Done.')
```

4. Label Encoder로 'LINE'과 'PRODUCT_CODE' 두 가지 인코딩

3-1. Feature Pre-Processing

```
imputer1 = IterativeImputer(random_state=37)
imputer2 = IterativeImputer(random_state=37)
imputer3 = IterativeImputer(random_state=37, max_iter=5)

trans_x1 = imputer1.fit_transform(tmp_T05)
trans_x2 = imputer2.fit_transform(tmp_T01)
trans_x3 = imputer3.fit_transform(tmp_T10)
```

5. 결측치의 경우, 한 feature의 10프로까지 결측이라면 median값으로 결측 대치 후, 더 많은 결측치를 가진 남은 feature들은 MICE라는 imputer 라이브러리를 통해 회귀 결측 대치

3-2. Feature Engineering

```
one_data_col_train_T05 = []
one_data_col_train_T10 = []
one_data_col_train_T01 = []

for col in train_T05.filter(regex='X').columns:
    if len(train_T05[col].unique().tolist()) ==1:
        one_data_col_train_T05.append(col)

for col in train_T10.filter(regex='X').columns:
    if len(train_T10[col].unique().tolist()) ==1:
        one_data_col_train_T10.append(col)

for col in train_T01.filter(regex='X').columns:
    if len(train_T01[col].unique().tolist()) ==1:
        one_data_col_train_T01.append(col)
```

- 아직 feature 수가 여전히 많이 남았기 때문에 추가적인 방법으로 feature 수를 줄이면서 새로 engeineering 시도함
- 한 가지 값만 가지는 feature들을 모두 값을 합쳐 새로운 feature로 생성

04

4. Modeling

4-1. Validation Set Configuration

4-2. Modeling

4-1. Validation set Configuration

```
val_T05_0 = int_T05[int_T05['Y_Class']==0][:8] # 41
val_T05_1 = int_T05[int_T05['Y_Class']==1][:9] # 46
val_T05_2 = int_T05[int_T05['Y_Class']==2][:6] # 33
val_T05 = pd.concat([val_T05_0, val_T05_1, val_T05_2]) # 120

val_T10_0 = int_T10[int_T10['Y_Class']==0][:5] # 28
val_T10_1 = int_T10[int_T10['Y_Class']==1][:57] # 289
val_T10_2 = int_T10[int_T10['Y_Class']==2][:6] # 32
val_T10 = pd.concat([val_T10_0, val_T10_1, val_T10_2]) # 349

val_T01_0 = int_T01[int_T01['Y_Class']==0][:4] # 19
val_T01_1 = int_T01[int_T01['Y_Class']==1][:14] # 72
val_T01_2 = int_T01[int_T01['Y_Class']==2][:7] # 38
val_T01 = pd.concat([val_T01_0, val_T01_1, val_T01_2]) # 129
```

```
val_T10
✓ 0.1s
```

	LINE	PRODUCT_CODE	X_1	X_2	X_5	X_7	X_8
0	0	1	2.0	102.0	11.0	45.0	10.0
1	0	1	2.0	102.0	11.0	45.0	10.0
10	0	1	2.0	102.0	11.0	45.0	10.0
11	0	1	2.0	95.0	11.0	45.0	10.0
12	0	1	2.0	101.0	11.0	45.0	10.0
...	...	...	...	...	...	...	...
54	0	1	1.0	97.0	11.0	45.0	10.0
58	0	1	2.0	100.0	10.0	45.0	10.0
68	0	1	2.0	100.0	11.0	45.0	10.0
72	0	1	2.0	99.0	10.0	45.0	10.0
82	0	1	2.0	95.0	10.0	45.0	10.0

68 rows × 553 columns

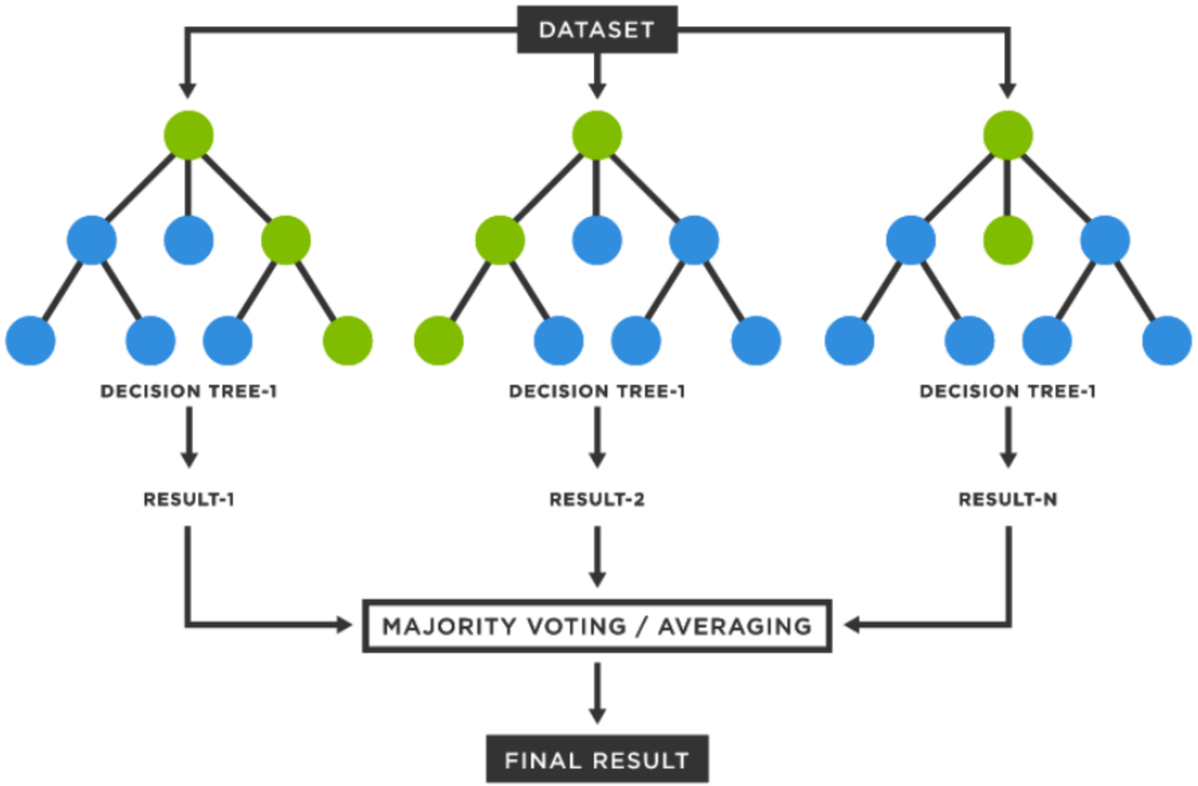
- Unbalance한 class 비율을 고려해서, 최종적으로 3개로 나누어진 데이터셋 각각 validation set을 구성함
- 비율을 고려하지 않으면 데이터가 적고 feature가 많은 현재 대회 데이터 특성 상 train set에 overfitting이 될 확률이 큼
- 또 너무 많은 수를 validation set에 배치하면 underfitting으로 데이터 패턴이 학습되지 않을 확률이 큼

4-2. Modeling

Y_Quality 회귀 모델 > **Y_Class 분류 모델**

- 우선 데이터셋에서 주요하게 판단되는 feature는 첫 번째로 우리가 구하고자 하는 정답인 'Y_Class'가 있음
- 두 번째로 제품의 품질을 수치화한 'Y_Quality' feature가 존재하는데, 이 feature는 test dataset에 없음
- 따라서, train dataset에서 'Y_Quality'를 산출하는 회귀모델을 구성해, test dataset에서 새로운 feature를 생성함
- 그 후 'Y_Quality' 까지 test datset에 추가해 최종적으로 구하고자 하는 'Y_Class'의 분류 모델을 구성함

4-2. Modeling



- 이번 데이터는 양이 적고, feature가 굉장히 다양하기 때문에 딥러닝 모델은 학습이 어려울 것이라고 판단함
- 또한, 앙상블 모델 중 boosting은 모델을 연속적으로 학습하는데 여러 실험 결과 상 train dataset과 test dataset 또한 괴리가 있을 것으로 판단되어 overfitting을 유발할 수 있으므로 bagging 방식을 이용해 모델링
- Bagging 방식 중 가장 대표적인 RandomForest를 이용하여 회귀/분류 모델 모두 모델링

감사합니다.