ANORMALY DETECTION

"알럽최적"

이주예, 김해인, 전해주, 김미섭

DATASET
분석 방법

AutoEncoder

Isolated Forest

→ 두 모델을 비교하여 더 나은 쪽을 택한다.

사출성형기 DATASET

- 데이터 수집 방법

제조 분야 : 자동차 앞유리 사이드 몰딩

제조 공정명 : 자동차 앞유리 사이드 몰딩사출 공정

수집장비 : 공장내 사출성형기 Data, 품질 Data

수집기간 : 2020년 10월 16일 ~ 2020년 11월 19일 (1개월)

수집주기 : 0.2 sec

사출 데이터 불량 원인

쇼트숏 플로우마크
플래시 기포
싱크마크 변형 휨

금형내 원료 부족
금형 온도&압력 저하
사출온도
불균일한 냉각 속도

최대한 많은 feature를 활용해 예측

공정 변수 조건		내 용
독립변수	온도 관련	스크류/실린더, 수지, 금형, 건조, 유압, 주변 환경
	압력 관련	충진 압력, 보압, 배압(계량시 발생하는 압력), 이형 압력, 형개 압력, 형체 압력
	시간 관련	충진 시간, 보압 시간, 냉각 시간, 건조 시간
	속도 관련	사출 속도, 스크류 회전 속도, 형개 속도, 이형(이젝팅) 속도
	양 관련	계량, 이형량, 쿠션량
종속변수	불량 여부	Y : 양품, N : 불량품 (Labeled) , 표시 없음 (Unlabeled)

데이터 preprocessing

- 완전성
- 유일성
- 유효성
- 일관성
- 정확성

데이터 전처리

- 유일성

```
a = len(labeled_data["_id"].unique())
print(a/len(labeled_data))
```

```
0.6543271635817909
```

```
result = labeled_data.drop_duplicates()
```

```
a = len(result["_id"].unique())
print(a/len(result))
```

```
1.0
```

- 완전성

```
labeled_data.isna().sum()
```

```
_id                          0
TimeStamp                    0
PART_FACT_PLAN_DATE          0
PART_FACT_SERIAL             0
PART_NAME                    0
EQUIP_CD                     0
EQUIP_NAME                   0
PassOrFail                   0
Reason                       0
Injection_Time               0
Filling_Time                 0
Plasticizing_Time            0
Cycle_Time                   0
Clamp_Close_Time             0
Cushion_Position             0
Switch_Over_Position         0
Plasticizing_Position        0
Clamp_Open_Position          0
Max_Injection_Speed          0
Max_Screw_RPM                0
Average_Screw_RPM            0
Max_Injection_Pressure       0
Max_Switch_Over_Pressure     0
Max_Back_Pressure            0
Average_Back_Pressure        0
Barrel_Temperature_1         0
Barrel_Temperature_2         0
```

데이터 전처리

- 유효성

```
labeled_data['TimeStamp'] = pd.to_datetime(labeled_data['TimeStamp'],format='%Y-%m-%dT%H:&M:SZ',errors="coerce")
labeled_data['PART_FACT_PLAN_DATE'] = pd.to_datetime(labeled_data['PART_FACT_PLAN_DATE'],format='%Y-%m-%d 오전 12:00:00',errors="coerce")
```

- 일관성 : 현상황에서 알기 어려움

데이터 전처리

- 정확성: PassOrFail & reason간의 상관관계 보기

```
labeled_data["Reason"].unique()
```

```
array(['None', '가스', '미성형', '초기허용불량'], dtype=object)
```

```
labeled_data["PassOrFail"].unique()
```

```
array(['Y', 'N'], dtype=object)
```

```
a = labeled_data['Reason'] == "None"
b = labeled_data['PassOrFail'] == "Y" # 정상이라면 불량 난 이유를 굳이 써줄 이유 없음
# 정상 == 정상?
print(len(labeled_data[a])-len(labeled_data[b]))
```

```
0
```

```
a = labeled_data['Reason'] != "None"
b = labeled_data['PassOrFail'] != "Y" # 정상이라면 불량 난 이유를 굳이 써줄 이유 없음
# 비정상 == 비정상?
print(len(labeled_data[a])-len(labeled_data[b]))
```

```
0
```

Data feature selection

사출 공정 데이터의 가정

- 기계&부품마다 데이터의 차이가 클 것이다

→ 기계&부품 별 데이터 따로 학습

- timestamp, part_fact_plan_date 와 같은 공정 기계와 무관한 외부 시계열 데이터는 공정 데이터에 영향을 주지 않을 것이다
- 값이 계속 0 인 값들은 abnormal 상태를 판별하는데 도움이 되지 않을 것이다.

"EQUIP_NAME„ & "PART_NAME"

```
labeled_data["EQUIP_NAME"].value_counts()
```

```
650톤-우진2호기    5230
1800TON-우진       1
650톤-우진          1
Name: EQUIP_NAME, dtype: int64
```

```
labeled_data["PART_NAME"].value_counts()
```

```
CN7 W/S SIDE MLD'G RH        1989
CN7 W/S SIDE MLD'G LH        1985
RG3 MOLD'G W/SHLD, LH         628
RG3 MOLD'G W/SHLD, RH         628
JX1 W/S SIDE MLD'G RH           1
SP2 CVR ROOF RACK CTR, RH       1
Name: PART_NAME, dtype: int64
```

앞의 세 글자가 같으면 같은
부품이라고 본다

```python
def delete_column(data, machine_name,product_name):
  machine_ = data["EQUIP_NAME"] == machine_name
  product_ = data["PART_NAME"] == product_name
  data = data[machine_ & product_]

  data.drop(['_id', 'TimeStamp', 'PART_FACT_PLAN_DATE', 'PART_FACT_SERIAL',
       'PART_NAME', 'EQUIP_CD', 'EQUIP_NAME', 'Reason',
       'Mold_Temperature_1', 'Mold_Temperature_2', 'Mold_Temperature_5', 'Mold_Temperature_6',
       'Mold_Temperature_7', 'Mold_Temperature_8', 'Mold_Temperature_9',
       'Mold_Temperature_10', 'Mold_Temperature_11', 'Mold_Temperature_12'],
       axis=1,inplace=True)
  return data
```

```python
machine_name ="650톤-우진2호기"
product_name = ["CN7 W/S SIDE MLD'G RH","CN7 W/S SIDE MLD'G LH","RG3 MOLD'G W/SHLD, RH","RG3 MOLD'G W/SHLD, LH " ]

cn7_rh = delete_column(labeled_data, machine_name,product_name[0])
cn7_lh = delete_column(labeled_data, machine_name,product_name[1])
rg3_rh = delete_column(labeled_data, machine_name,product_name[2])
rg3_lh = delete_column(labeled_data, machine_name,product_name[3])
```

```python
cn7 = pd.concat([cn7_lh,cn7_rh])
rg3 = pd.concat([rg3_lh,rg3_rh])
```

pass or fail 을 각각 0 ,1 으로 바꾸기

```python
cn7["PassOrFail"] = cn7["PassOrFail"].replace("Y",1).replace("N",0)
rg3["PassOrFail"] = rg3["PassOrFail"].replace("Y",1).replace("N",0)
```

기본적인 데이터 selecting끝
→부품별 data selecting

Cn7 부품 Data Selecting

PassOrFail	Injection_Time	Filling_Time	Plasticizing_Time	Cycle_Time	Clamp_Close_Time	Cushion_Position	Switch_Over_Position	Plasticizing_Position
1	9.60	4.48	16.91	59.580002	7.13	653.409973	0.0	68.839996
1	9.59	4.48	16.91	59.560001	7.13	653.419983	0.0	68.839996
1	9.58	4.46	16.90	59.580002	7.13	653.409973	0.0	68.839996
1	9.58	4.46	16.92	59.560001	7.13	653.409973	0.0	68.849998
1	9.57	4.45	16.91	59.520000	7.14	653.409973	0.0	68.830002

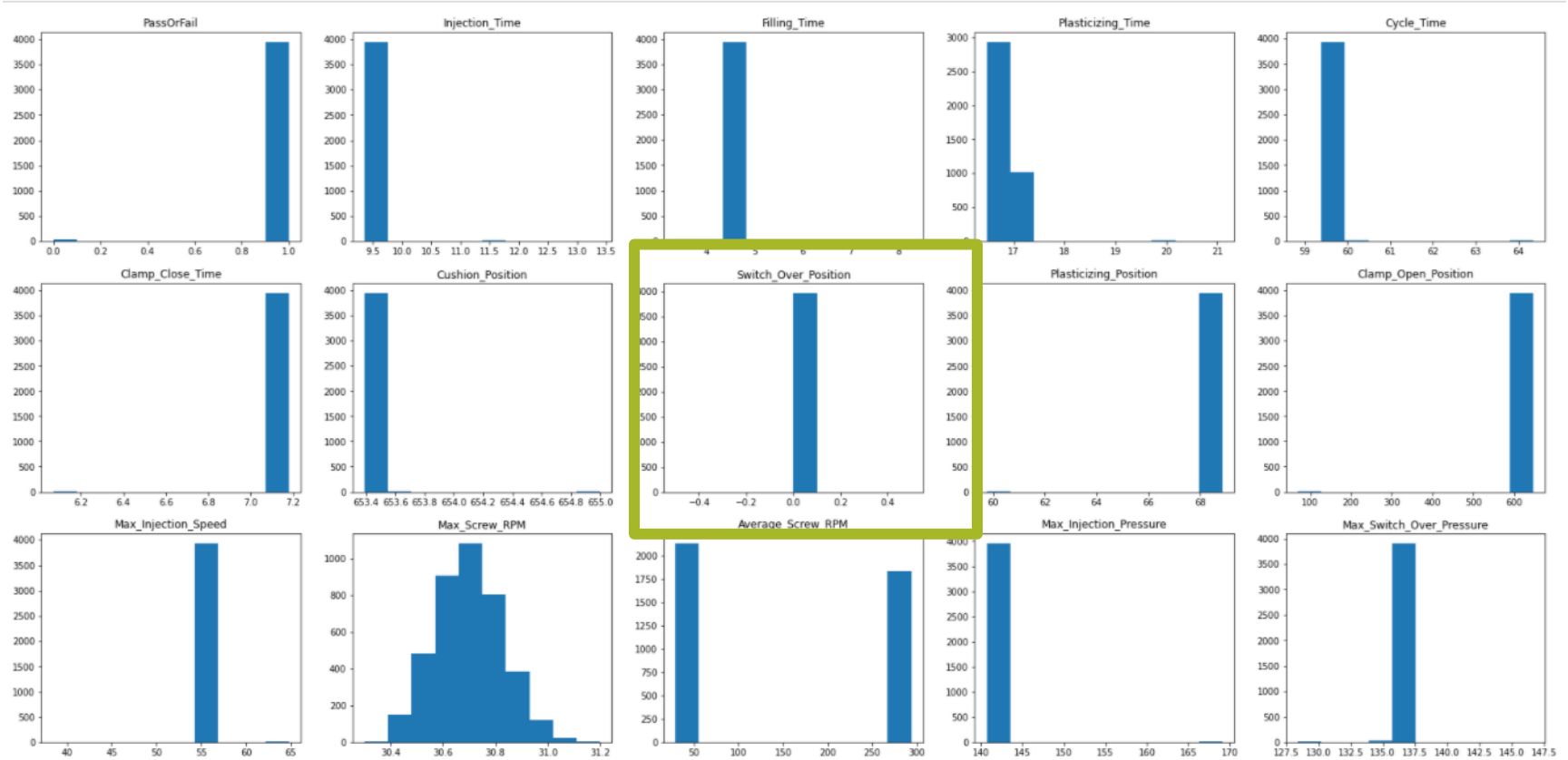
Clamp_Open_Position	Max_Injection_Speed	Max_Screw_RPM	Average_Screw_RPM	Max_Injection_Pressure	Max_Switch_Over_Pressure	Max_Back_Pressure
647.98999	55.299999	30.799999	292.500000	141.800003	136.899994	37.500000
647.98999	55.299999	31.000000	292.500000	141.800003	136.800003	37.500000
647.98999	55.599998	30.900000	292.500000	141.699997	136.399994	37.700001
647.98999	55.500000	30.600000	292.399994	141.800003	136.699997	37.400002
647.98999	55.700001	30.799999	292.500000	141.600006	136.399994	37.099998

Cn7 부품 Data Selecting

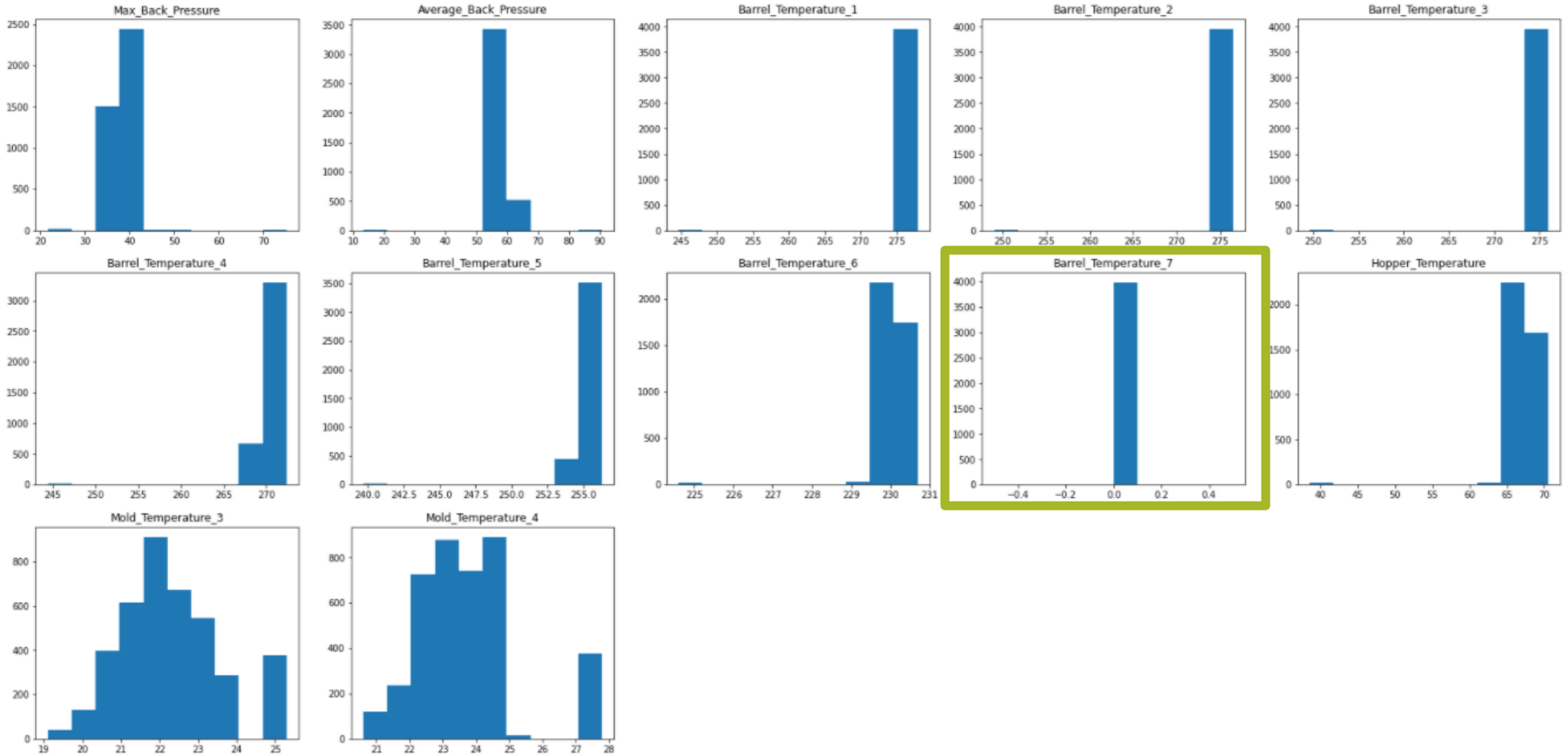
Average_Back_Pressure	Barrel_Temperature_1	Barrel_Temperature_2	Barrel_Temperature_3	Barrel_Temperature_4	Barrel_Temperature_5	Barrel_Temperature_6
59.299999	276.200012	275.500000	275.299988	270.799988	254.699997	229.500000
59.299999	276.500000	275.000000	275.399994	271.100006	254.899994	230.000000
59.400002	276.299988	275.299988	275.200012	271.399994	255.000000	230.000000
59.299999	275.799988	275.399994	275.000000	271.299988	255.000000	230.000000
59.099998	275.700012	274.899994	274.799988	270.799988	255.399994	230.100006

Barrel_Temperature_7	Hopper_Temperature	Mold_Temperature_3	Mold_Temperature_4
0.0	67.199997	24.799999	27.6
0.0	66.900002	25.000000	27.6
0.0	67.500000	25.000000	27.6
0.0	67.000000	25.000000	27.6
0.0	66.699997	24.799999	27.5

Cn7 부품 Data Selecting



Cn7 부품 Data Selecting



```
cn7['Switch_Over_Position'].unique()
```

```
array([0.])
```

```
cn7['Barrel_Temperature_7'].unique()
```

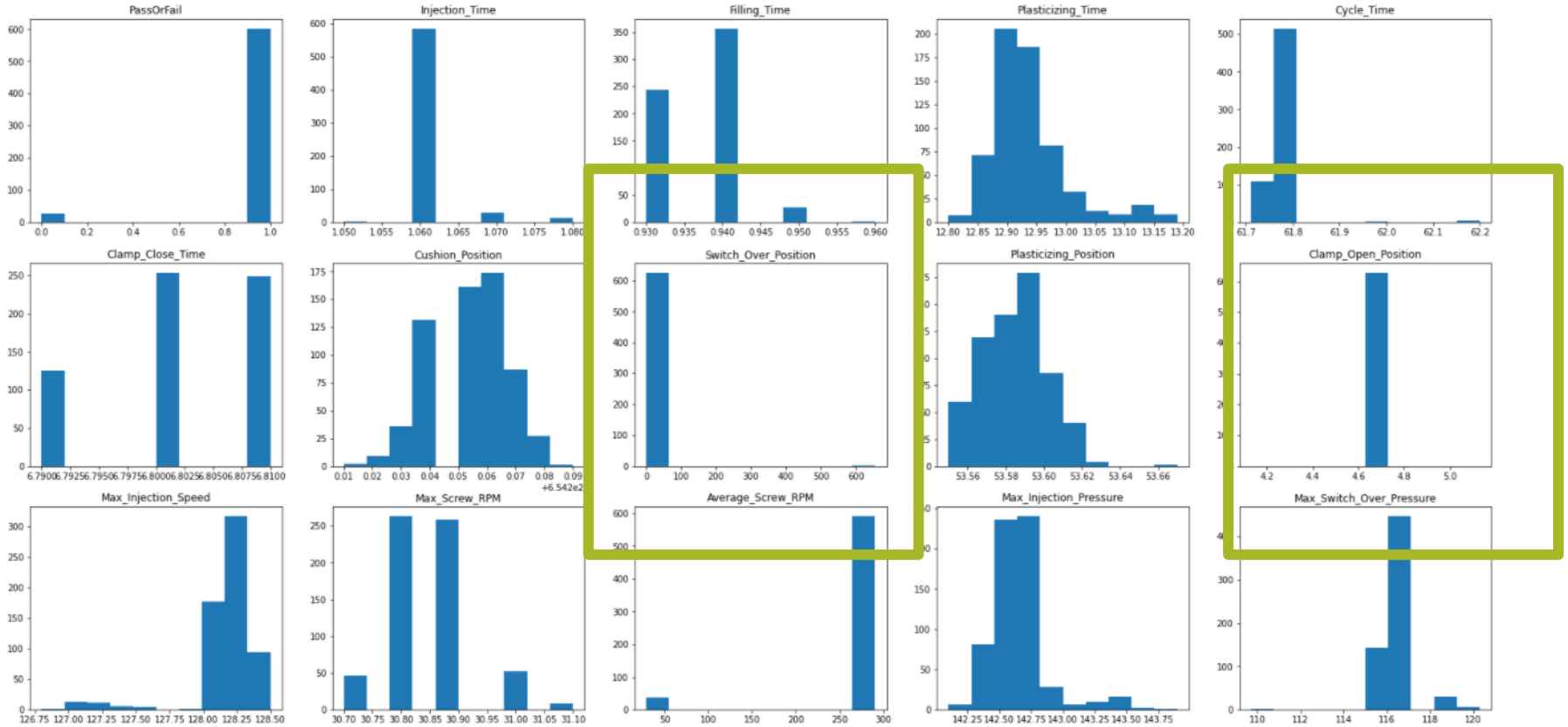
```
array([0.])
```

Switch_Over_Position Barrel_Temperature_7 의 값이 모두 unique함으로 cn7에서 위 두 칼럽을 지운다

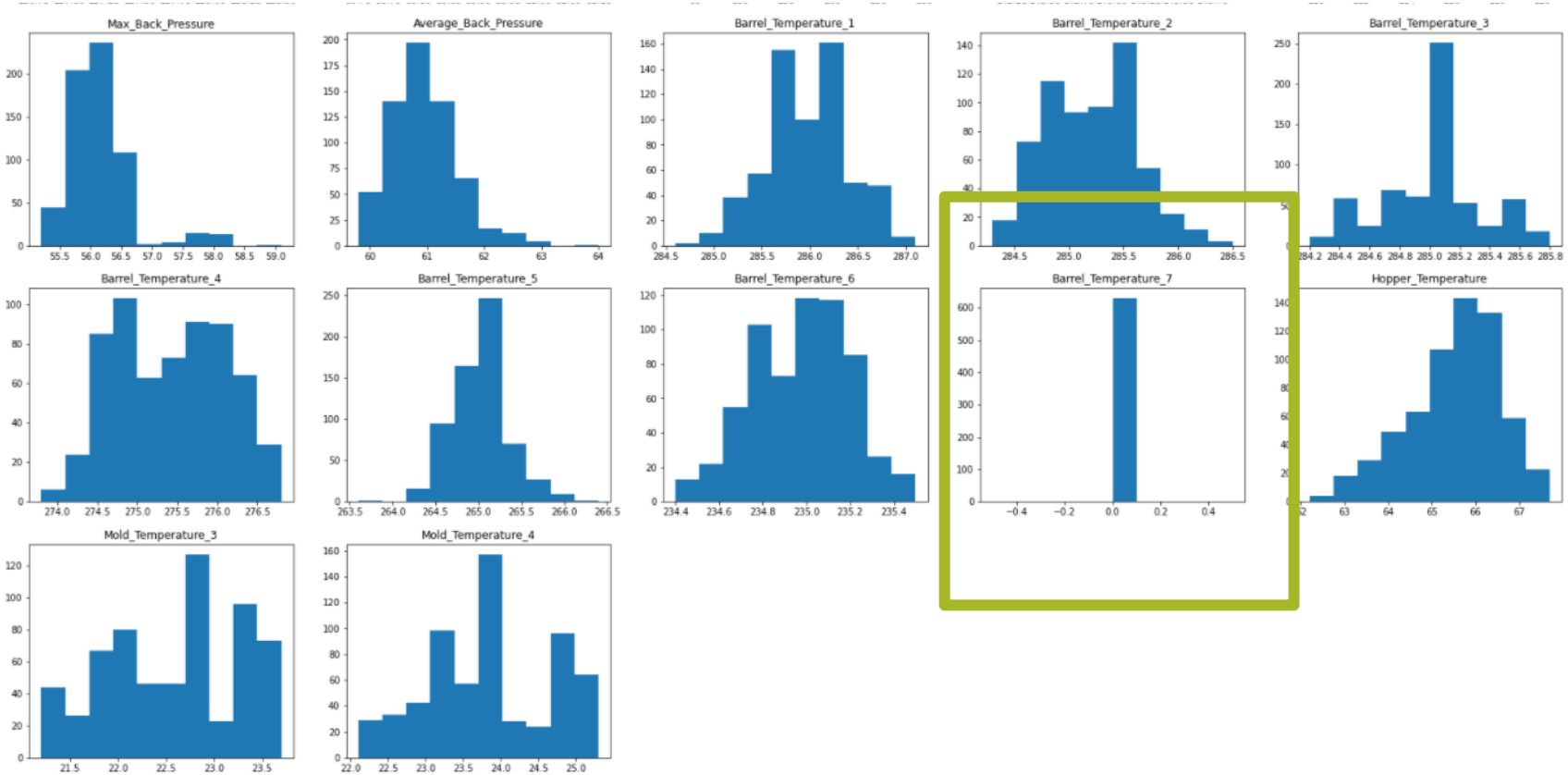
Switch_Over_Position Barrel_Temperature_7 의 값이 모두 unique함으로 cn7에서 위 두 칼럽을 지운다

```
cn7.drop(["Switch_Over_Position","Barrel_Temperature_7"],axis=1,inplace=True)
```

Rg3 부품 Data Selection



Rg3 부품 Data Selection



```
rg3['Switch_Over_Position'].unique()
```

```
array([  0.        , 655.30999756])
```

```
rg3["Clamp_Open_Position"].unique()
```

```
array([4.63000011])
```

```
rg3["Clamp_Open_Position"].describe()
```

```
count    628.00
mean       4.63
std        0.00
min        4.63
25%        4.63
50%        4.63
75%        4.63
max        4.63
Name: Clamp_Open_Position, dtype: float64
```

```
rg3['Barrel_Temperature_7'].unique()
```

```
array([0.])
```

```
rg3.drop(["Switch_Over_Position","Clamp_Open_Position","Barrel_Temperature_7"],axis=1,inplace=True)
```

```
# 불량 정상 갯수 확인

cn7_Y = cn7[cn7["PassOrFail"]==1]
cn7_N = cn7[cn7["PassOrFail"]==0]

print(f"cn7의 정상품 수 {len(cn7_Y)}  cn7의 불량품 수 {len(cn7_N)}")

rg3_Y = rg3[rg3["PassOrFail"]==1]
rg3_N = rg3[rg3["PassOrFail"]==0]

print(f"rg3의 정상품 수 {len(rg3_Y)}  rg3의 불량품 수 {len(rg3_N)}")
```

```
cn7의 정상품 수 3946  cn7의 불량품 수 28
rg3의 정상품 수 601  rg3의 불량품 수 27
```

모델에 대입하도록 데이터를 정규화

```
# y 종속변수 제거

cn7_Y_y = cn7_Y["PassOrFail"]
cn7_N_y = cn7_N["PassOrFail"]
cn7_Y_x = cn7_Y.iloc[:,1:]
cn7_N_x = cn7_N.iloc[:,1:]

rg3_Y_y = rg3_Y["PassOrFail"]
rg3_N_y = rg3_N["PassOrFail"]
rg3_Y_x = rg3_Y.iloc[:,1:]
rg3_N_x = rg3_N.iloc[:,1:]
```

```
scaler = MinMaxScaler()

cn7_Y = scaler.fit_transform(cn7_Y_x)
cn7_N = scaler.fit_transform(cn7_N_x)

rg3_Y =  scaler.fit_transform(rg3_Y_x)
rg3_N =  scaler.fit_transform(rg3_N_x)
```

training data test data

```
[47] rg3_Y_x.shape
```

```
(601, 23)
```

```
rg3_N_x.shape
```

```
(27, 23)
```

```
[48] cn7_Y_x.shape
```

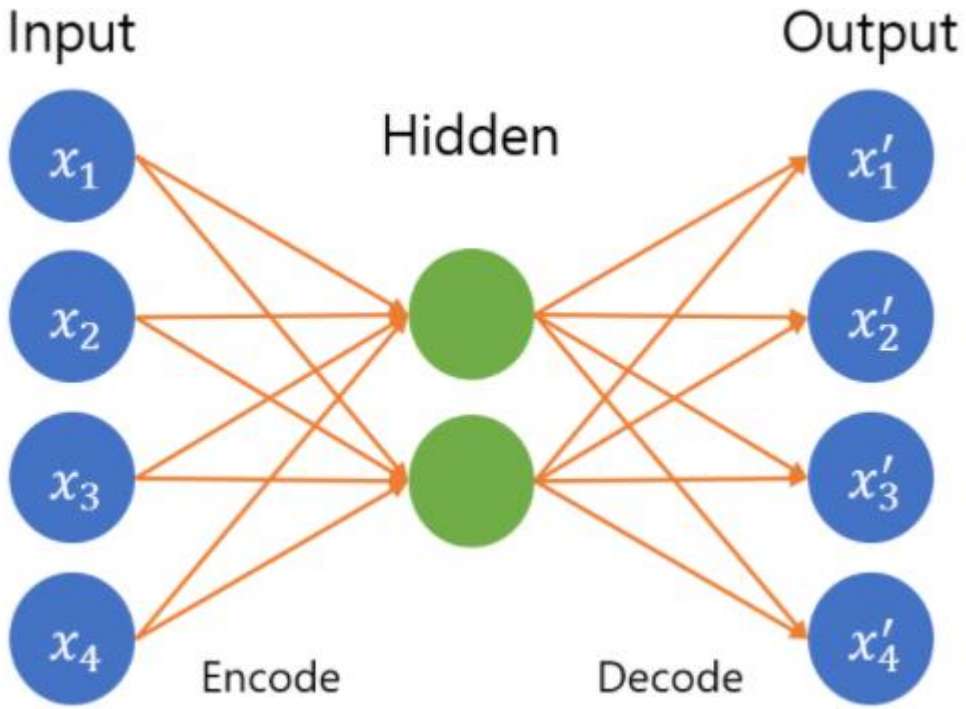
```
(3946, 24)
```

```
[49] cn7_N_x.shape
```

```
(28, 24)
```

AUTO ENCODING

- 라벨이 없는 인풋데이터를 학습하여 최대한 인풋과 비슷한 아웃풋을 내는 것을 목표로 하는 인공 신경망
- 그 과정에서 인풋데이터의 feature를 학습하게 됨
- 데이터를 압축하는 인코딩과 복원하는 디코딩 과정으로 구성



AUTO ENCODING

```
def AE(epoch, batch):
  # 인코더
  dropout_encoder = Sequential([

      Dropout(0.3),
      Dense(20,activation="relu"),
      Dense(10,activation="relu")])
  # 디코더
  dropout_decoder = Sequential([
      Dense(20,activation="relu",input_shape=[10]),
      Dense(cn7_train.shape[1],activation="relu")])
  dropout_AE = Sequential([dropout_encoder,dropout_decoder])

  # 손실함수 옵티마이저 정의
  dropout_AE.compile(loss="mse",optimizer=Adam(lr=0.03),metrics=["accuracy"])
  # 모델훈련
  history = dropout_AE.fit(cn7_train,cn7_train,batch_size=batch,epochs=epoch,validation_split=0.2,callbacks=[EarlyStopping(monitor="val
  plt.figure(1)
  plt.figure(figsize=(10,3))
  plt.plot(history.history["accuracy"],label="Training Acc")
  plt.plot(history.history["val_accuracy"],label="Validation Acc")
  plt.legend()
  plt.figure(2)
  plt.figure(figsize=(10,3))
  plt.plot(history.history["loss"],label="Training Loss")
  plt.plot(history.history["val_loss"],label="Validation Loss")
  plt.legend()
  plt.show()
```

AUTO ENCODING

```python
cn7_train_pred = dropout_AE.predict(cn7_train)
cn7_train_loss = np.mean(np.square(cn7_train_pred - cn7_train),axis=1)
threshold = np.mean(cn7_train_loss)+1.96*np.std(cn7_train_loss)
print(f"복원 오류 임계치: {threshold}") # 원가 이상
print("-"*60)
# 예측값

# 평가 데이터 정상
cn7_predict_y = dropout_AE.predict(cn7_test_y)
cn7_test_y_mse = np.mean(np.square(cn7_predict_y - cn7_test_y),axis=1)
# 시각화
plt.figure(3)
plt.hist(cn7_test_y_mse,bins = 30)
plt.xlabel("test mse loss")
plt.ylabel("no of samples")
plt.title("testing normal data")
plt.show()

#불량으로 판단한 데이터 확인
cn7_test_y_anomalies = len(cn7_test_y_mse[cn7_test_y_mse > threshold])
fn = cn7_test_y_anomalies
# 정상을 정상으로 판단
tp = len(cn7_test_y_mse)-fn
print(f"불량/정상 갯수 : {fn}")
print(f"정상/정상 갯수 : {tp}")
```

```python
# 평가 데이터 불량

cn7_predict_n = dropout_AE.predict(cn7_test_n)
cn7_test_n_mse = np.mean(np.square(cn7_predict_n - cn7_test_n),axis=1)
# 시각화
plt.figure(4)
plt.hist(cn7_test_n_mse,bins = 30)
plt.title("testing abnormal data")
plt.xlabel("test mse loss")
plt.ylabel("no of samples")
plt.show()

#불량으로 판단한 데이터 확인
cn7_test_n_anomalies = len(cn7_test_n_mse[cn7_test_n_mse > threshold])
tn = cn7_test_n_anomalies
# 실제 불량인데 정상으로 판단한 데이터
fp = len(cn7_test_n_mse)-tn
print(f"불량/불량 갯수 : {tn}")
print(f"정상/불량 갯수 : {fp}")
return (epoch,batch,fn,tp,tn,fp)
```

AUTO ENCODING

- 소요시간 : 6.517923[ms]

```
def allsummery(epochs,batches):
  arr = []
  for i in epochs:
    for j in batches:
      t = AE(i,j)
      arr.append(t)
  df = pd.DataFrame(arr,columns=["epoch","batch","fn","tp","tn","fp"])
  return df
relu_201020 = allsummery([500,100,50,30],[30,20,10])
```

Cn7: 20-10-20 relu + early stop

```
relu_201020.groupby(["epoch","batch"]).mean()
```

		fn	tp	tn	fp	precision	recall	accuracy	F1
epoch	batch								
30	10	35.133333	1510.866667	28.0	0.0	1.000000	0.977275	0.977679	0.988425
	20	28.133333	1517.866667	28.0	0.0	1.000000	0.981803	0.982126	0.990754
	30	40.433333	1505.566667	28.0	0.0	1.000000	0.973846	0.974312	0.986622
50	10	43.100000	1502.900000	27.6	0.4	0.999716	0.972122	0.972363	0.985511
	20	39.900000	1506.100000	28.0	0.0	1.000000	0.974191	0.974651	0.986848
	30	28.500000	1517.500000	28.0	0.0	1.000000	0.981565	0.981893	0.990643
100	10	38.366667	1507.633333	28.0	0.0	1.000000	0.975183	0.975625	0.987265
	20	42.400000	1503.600000	28.0	0.0	1.000000	0.972574	0.973062	0.985801
	30	40.400000	1505.600000	28.0	0.0	1.000000	0.973868	0.974333	0.986596

Cn7 : 20-10-20 relu + early stop + train data 많이

```
relu_201020_2.groupby(["epoch","batch"]).mean()
```

		fn	tp	tn	fp	precision	recall	accuracy	F1
epoch	batch								
30	10	7.900000	938.100000	28.0	0.0	1.0	0.991649	0.991889	0.995784
	20	6.000000	940.000000	28.0	0.0	1.0	0.993658	0.993840	0.996795
	30	6.733333	939.266667	28.0	0.0	1.0	0.992882	0.993087	0.996416
50	10	9.566667	936.433333	28.0	0.0	1.0	0.989887	0.990178	0.994881
	20	7.200000	938.800000	28.0	0.0	1.0	0.992389	0.992608	0.996168
	30	6.566667	939.433333	28.0	0.0	1.0	0.993058	0.993258	0.996506
100	10	5.933333	940.066667	28.0	0.0	1.0	0.993728	0.993908	0.996843
	20	7.166667	938.833333	28.0	0.0	1.0	0.992424	0.992642	0.996180
	30	6.533333	939.466667	28.0	0.0	1.0	0.993094	0.993292	0.996529

```
cn7_train = cn7_Y[:2400] #
cn7_test_y = cn7_Y[2400:]
cn7_test_n =cn7_N
```

```
cn7_train = cn7_Y[:3000] #
cn7_test_y = cn7_Y[3000:]
cn7_test_n =cn7_N
```

Cn7 : 20-10-20 relu + no early stop

epoch	batch	fn	tp	tn	fp	precision	recall	accuracy	F1
30	**10**	27.000000	1519.000000	28.0	0.0	1.0	0.982536	0.982846	0.991118
	20	39.833333	1506.166667	28.0	0.0	1.0	0.974235	0.974693	0.986819
	30	32.400000	1513.600000	28.0	0.0	1.0	0.979043	0.979416	0.989305
50	**10**	45.066667	1500.933333	28.0	0.0	1.0	0.970850	0.971368	0.985081
	20	39.200000	1506.800000	28.0	0.0	1.0	0.974644	0.975095	0.986984
	30	42.600000	1503.400000	28.0	0.0	1.0	0.972445	0.972935	0.985817
100	**10**	26.533333	1519.466667	28.0	0.0	1.0	0.982837	0.983143	0.991238
	20	32.400000	1513.600000	28.0	0.0	1.0	0.979043	0.979416	0.989307
	30	27.300000	1518.700000	28.0	0.0	1.0	0.982342	0.982656	0.990989

Cn7 : 20-10-20 relu: early stop vs no early stop

precision	recall	accuracy	F1
1.000000	0.977275	0.977679	0.988425
1.000000	0.981803	0.982126	0.990754
1.000000	0.973846	0.974312	0.986622
0.999716	0.972122	0.972363	0.985511
1.000000	0.974191	0.974651	0.986848
1.000000	0.981565	0.981893	0.990643
1.000000	0.975183	0.975625	0.987265
1.000000	0.972574	0.973062	0.985801
1.000000	0.973868	0.974333	0.986596

precision	recall	accuracy	F1
1.0	0.982536	0.982846	0.991118
1.0	0.974235	0.974693	0.986819
1.0	0.979043	0.979416	0.989305
1.0	0.970850	0.971368	0.985081
1.0	0.974644	0.975095	0.986984
1.0	0.972445	0.972935	0.985817
1.0	0.982837	0.983143	0.991238
1.0	0.979043	0.979416	0.989307
1.0	0.982342	0.982656	0.990989

큰 차이 없음
→ 시간 효율성을 위해 early stop condition을 사용하겠음

Rg3

- 낮은 accuracy –〉 Isolation Forest 이용

epoch	batch	fn	tp	tn	fp	elapsed time	precision	recall	accuracy	F1
40	10	185.0	36.0	26.0	1.0	6.517923	0.980000	0.162896	0.250000	0.275081
	15	171.0	50.0	26.0	1.0	6.461326	0.984615	0.226244	0.306452	0.363691
	20	181.5	39.5	26.5	0.5	4.378647	0.990196	0.178733	0.266129	0.299824
100	10	154.0	67.0	26.0	1.0	6.748734	0.985294	0.303167	0.375000	0.463668
	15	185.0	36.0	27.0	0.0	6.869397	1.000000	0.162896	0.254032	0.280156
	20	175.0	46.0	27.0	0.0	6.777915	1.000000	0.208145	0.294355	0.344569

Isolation Forest – 차원 축소

- PCA를 이용하여 12차원으로 축소

```
# 각 변수들의 설명력이 너무 약해서 차원축소
from sklearn.decomposition import KernelPCA, PCA
pca = PCA() #주성분 개수 지정하지 않고 클래스생성

pca.fit(cn7_x)  #주성분 분석
cumsum = np.cumsum(pca.explained_variance_ratio_) #분산의 설명량을 누적합
num_c = np.argmax(cumsum >= 0.95) + 1 # 분산의 설명량이 95%이상 되는 차원의 수
pca = PCA(n_components = num_c)

cn7_Y_x = pca.fit_transform(cn7_Y_x)
cn7_N_x = pca.fit_transform(cn7_N_x)
```

Isolation Forest – Rg3 data

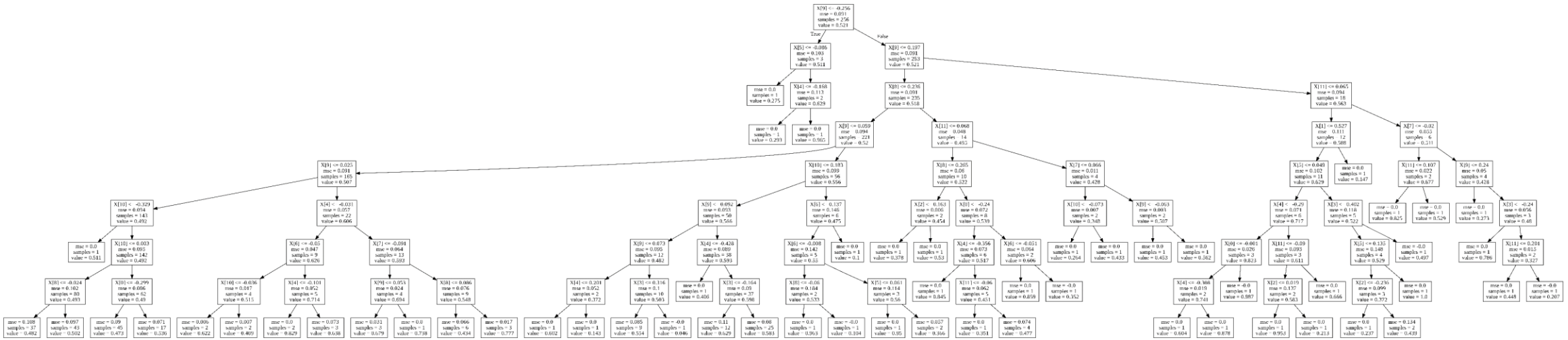
- GridSearchCV를 이용하여 hyperparameter 결정

```
#define hyperparameter
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score, make_scorer
import time

param_grid = {
    'contamination': [0.0001, 0.001, 0.01, 0.1],
    'bootstrap': [True, False],
    'n_estimators': [50, 70, 80, 100]
}

model = IsolationForest()
gs = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv= 5)
gs = gs.fit(rg3_x[380:], rg3_y[380:])
gs.best_params_
```

```
{'bootstrap': False, 'contamination': 0.0001, 'n_estimators': 80}
```

Isolation Forest – Rg3 data

- 학습시간 : 0.1476593017578125
- 예측시간 : 0.1302504539489746

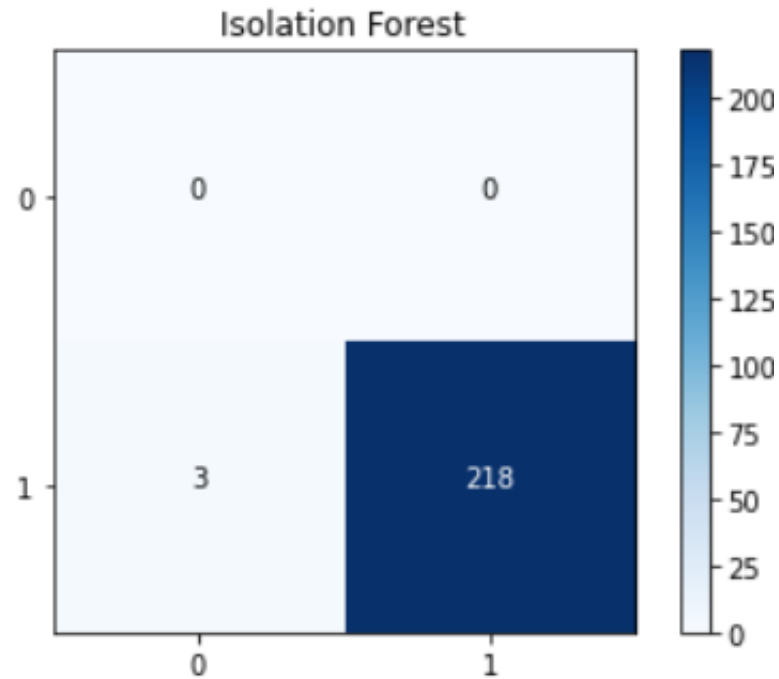


Isolation Forest – Rg3 data

- Normal data

Accuracy: 0.9864253393665159
classification_report

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	3
1	0.99	1.00	0.99	218
accuracy			0.99	221
macro avg	0.49	0.50	0.50	221
weighted avg	0.97	0.99	0.98	221

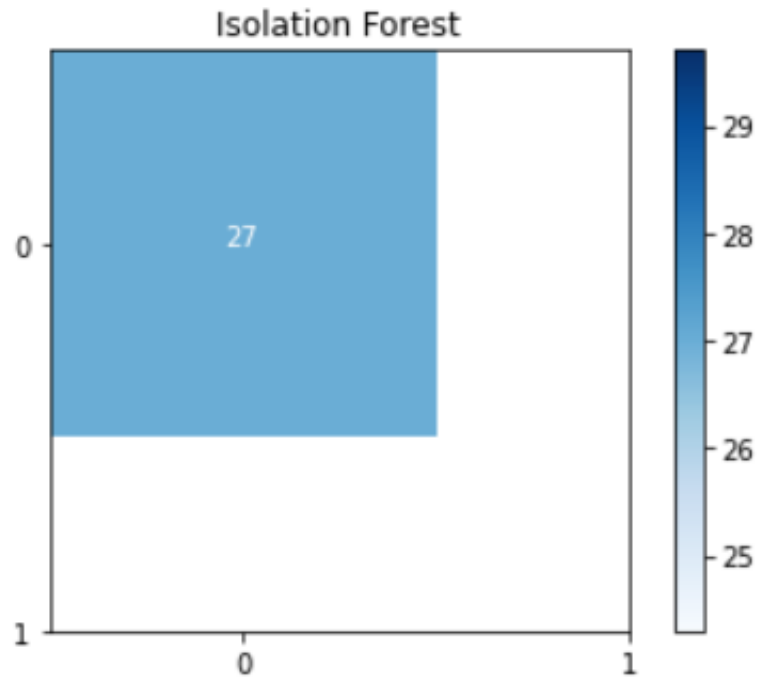


Isolation Forest – Rg3 data

- Abnormal data

Accuracy: 1.0
classification_report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	27
accuracy			1.00	27
macro avg	1.00	1.00	1.00	27
weighted avg	1.00	1.00	1.00	27



용해탱크 DATASET

	STD_DT	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP	TAG
0	2020-03-04 0:00	489	116	631	3.19	0
1	2020-03-04 0:00	433	78	609	3.19	0
2	2020-03-04 0:00	464	154	608	3.19	0
3	2020-03-04 0:00	379	212	606	3.19	0
4	2020-03-04 0:00	798	1736	604	3.21	0
...	...	...	...	...	...	...
835195	2020-04-30 23:59	755	1743	318	3.21	0
835196	2020-04-30 23:59	385	206	317	3.19	0
835197	2020-04-30 23:59	465	148	316	3.20	0
835198	2020-04-30 23:59	467	0	314	3.19	0
835199	2020-04-30 23:59	453	125	312	3.20	0

835200 rows × 6 columns

데이터 전처리

- 전체 데이터 중 앞의 10%의 데이터는 모두 양품 -〉 train set
- 나머지 데이터 -〉 test set

```
print(train.shape)
print(test.shape)
```

```
(83520, 5)
(751680, 5)
```

- X, y set 나누고 MinMaxScaler()로 표준화

```
X_train = train_sc[:,:-1]
y_train = train_sc[:,-1]
X_test = test_sc[:,:-1]
y_test = test_sc[:,-1]
```

데이터 전처리

- 시계열 데이터 분석은 변수의 정상성이 보장되어야 함 -> Dickey-Fuller test
- 독립변수 4개에 대한 p-value는 모두 0 -> 귀무가설을 기각. 정상성 유효함

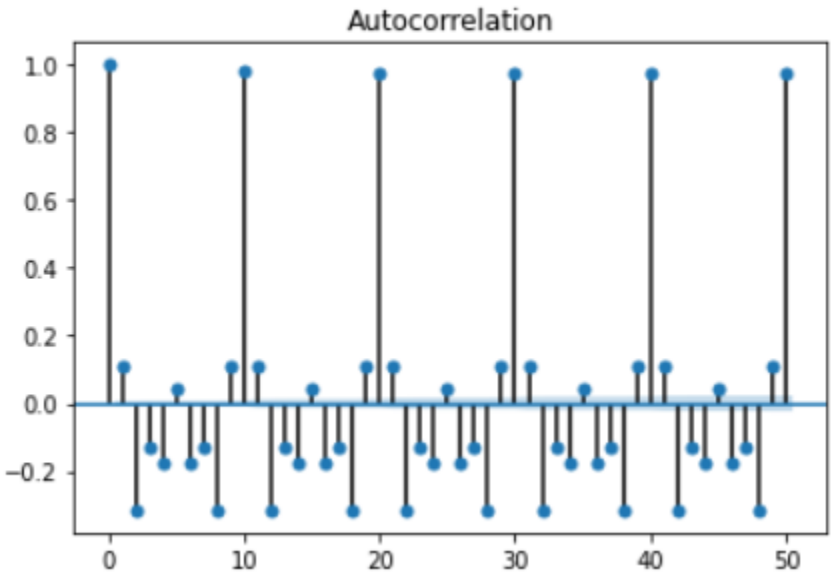
```
# 정상성 점검 - Dickey-Fuller
from statsmodels.tsa.stattools import adfuller
result = adfuller(train['MELT_TEMP'])
print(result[1])
```

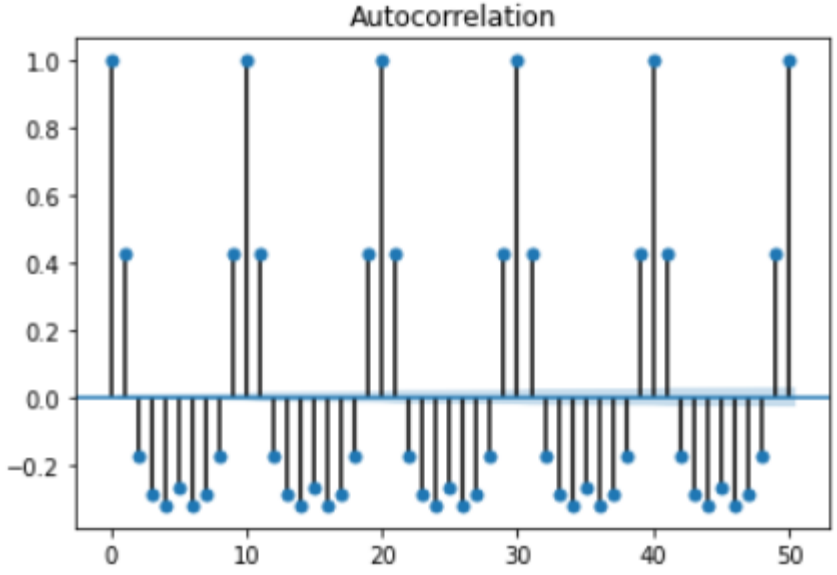
```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools,
  import pandas.util.testing as tm
0.0
```

프로세서:	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
설치된 메모리(RAM):	8.00GB(7.84GB 사용 가능)

Deseasonalize

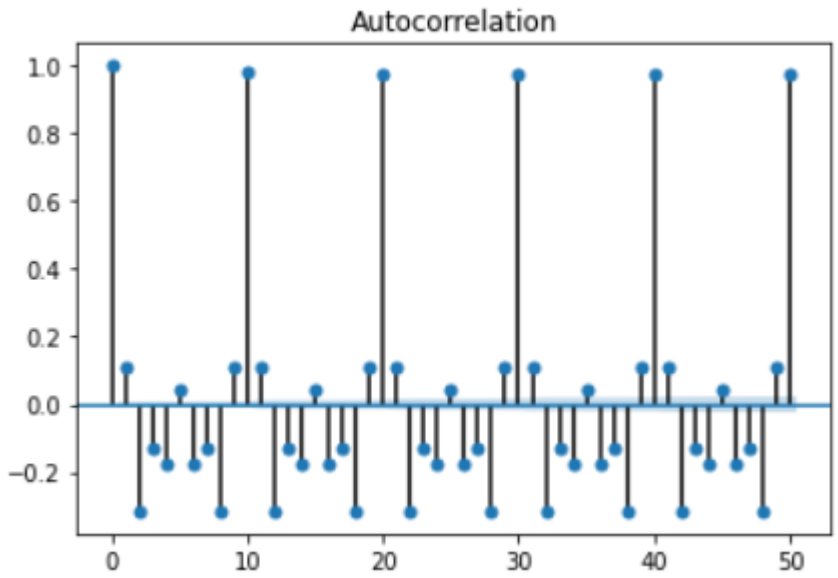
- 정상데이터의 Autocorrelation 확인

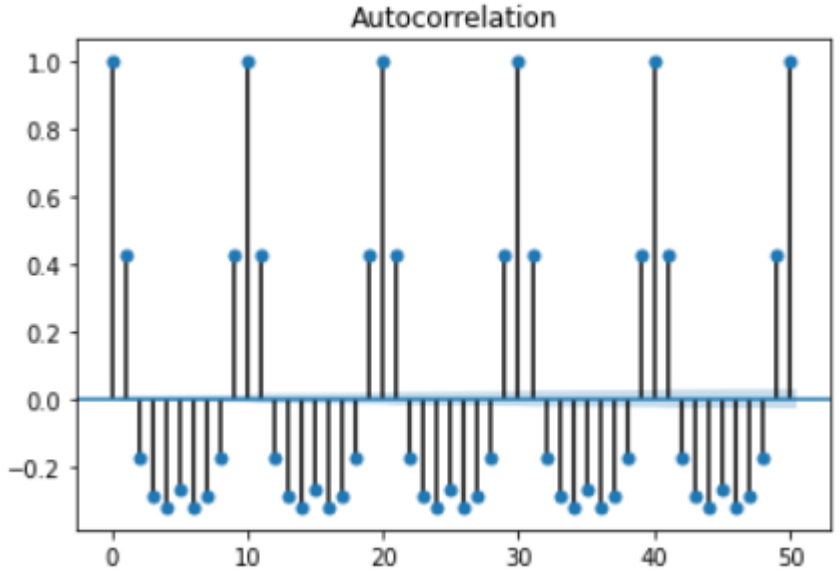




Deseasonalize

- 정상데이터의 Autocorrelation 확인
- 특히 lag=10일 때 강한 상관관계를 보임 → 10을 주기로 하는 패턴 존재





Deseasonalize

- 10은 동시간에 수집되는 데이터의 개수
- 수집되는 순서에 따라 데이터의 경향성이 달라짐
- 각 순서 별 평균값을 구함

```
[10]  tank_normal = tank.iloc[:10000,1:5]
      for i in range(10):
          globals()[f'tank_{i}'] = tank_normal.iloc[[10*j+i for j in range(1000)],:]
```

```
[11]  for i in range(10):
        globals()[f'mean_{i}'] = globals()[f'tank_{i}'].mean()
```

```
[12]  for i in range(10):
        for j in range(1000):
          for k in range(4):
              globals()[f'tank_{i}'].iloc[j,k] = globals()[f'tank_{i}'].iloc[j,k] - globals()[f'mean_{i}'][k
```

Deseasonalize

- Deseasonalize 함수 생성

```
[18]  def deseasonalize(df):
        global mean_0, mean_1, mean_2, mean_3, mean_4, mean_5, mean_6, mean_7, mean_8, mean_9

        num = int(len(df)/10)

        for i in range(num):
          for k in range(4):
            a = float(df.iloc[10*i,k])- mean_0[k]
            df.iloc[10*i,k] =a
            df.iloc[10*i+1,k] = df.iloc[10*i+1,k] - mean_1[k]
            df.iloc[10*i+2,k] = df.iloc[10*i+2,k] - mean_2[k]
            df.iloc[10*i+3,k] = df.iloc[10*i+3,k] - mean_3[k]
            df.iloc[10*i+4,k] = df.iloc[10*i+4,k] - mean_4[k]
            df.iloc[10*i+5,k] = df.iloc[10*i+5,k] - mean_5[k]
            df.iloc[10*i+6,k] = df.iloc[10*i+6,k] - mean_6[k]
            df.iloc[10*i+7,k] = df.iloc[10*i+7,k] - mean_7[k]
            df.iloc[10*i+8,k] = df.iloc[10*i+8,k] - mean_8[k]
            df.iloc[10*i+9,k] = df.iloc[10*i+9,k] - mean_9[k]

          return df
```

Autoencoding

- Epoch 와 batch 각 조합으로 10번 실행한 평균적인 결과

epoch	batch	fn	tp	tn	fp	precision	recall	accuracy	F1
30	10	814.8	33501.2	65.1	15618.9	0.681835	0.976256	0.671326	0.802796
	20	3878.9	30437.1	43.2	15640.8	0.659736	0.886965	0.609606	0.756227
	30	3240.9	31075.1	43.8	15640.2	0.664748	0.905557	0.622378	0.766440
50	10	3041.4	31274.6	49.8	15634.2	0.665789	0.911371	0.626488	0.768960
	20	1162.1	33153.9	57.5	15626.5	0.679510	0.966135	0.664228	0.797777
	30	3041.2	31274.8	47.5	15636.5	0.666181	0.911377	0.626446	0.769452
100	10	1104.1	33211.9	70.0	15614.0	0.679996	0.967826	0.665638	0.798649
	20	2311.9	32004.1	57.1	15626.9	0.671521	0.932629	0.641224	0.780599
	30	2140.5	32175.5	55.2	15628.8	0.672806	0.937624	0.644614	0.783292
500	10	2488.9	31827.1	44.8	15639.2	0.669736	0.927471	0.637438	0.777386
	20	2579.5	31736.5	52.5	15631.5	0.669759	0.924831	0.635780	0.776757
	30	2685.2	31630.8	56.9	15627.1	0.668626	0.921751	0.633754	0.774667

Autoencoding

- Epoch 와 batch 각 조합으로 10번 실행한 평균적인 결과

epoch	batch	fn	tp	tn	fp	precision	recall	accuracy	F1
30	10	814.8	33501.2	65.1	15618.9	0.681835	0.976256	0.671326	0.802796
	20	3878.9	30437.1	43.2	15640.8	0.659736	0.886965	0.609606	0.756227
	30	3240.9	31075.1	43.8	15640.2	0.664748	0.905557	0.622378	0.766440
50	10	3041.4	31274.6	49.8	15634.2	0.665789	0.911371	0.626488	0.768960
	20	1162.1	33153.9	57.5	15626.5	0.679510	0.966135	0.664228	0.797777
	30	3041.2	31274.8	47.5	15636.5	0.666181	0.911377	0.626446	0.769452
100	10	1104.1	33211.9	70.0	15614.0	0.679996	0.967826	0.665638	0.798649
	20	2311.9	32004.1	57.1	15626.9	0.671521	0.932629	0.641224	0.780599
	30	2140.5	32175.5	55.2	15628.8	0.672806	0.937624	0.644614	0.783292
500	10	2488.9	31827.1	44.8	15639.2	0.669736	0.927471	0.637438	0.777386
	20	2579.5	31736.5	52.5	15631.5	0.669759	0.924831	0.635780	0.776757
	30	2685.2	31630.8	56.9	15627.1	0.668626	0.921751	0.633754	0.774667

Autoencoding

- 가장 결과가 좋은 (epoch, batch) = (30,10) 조합으로 실행한 결과
- 훈련 소요 시간 : 27.75초
- 예측 소요 시간 : 1.33초
- Confusion Matrix

	실제 0	실제 1
예측 0	32067	15634
예측 1	2249	50

```
[43] train_end - train_start

     27.74671244621277

[44] test_end - train_end

     1.332848310470581

[45] relu_201020
```

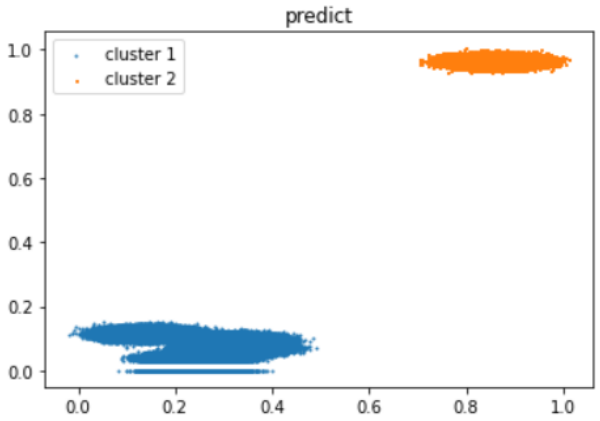
	epoch	batch	fn	tp	tn	fp
0	30	10	2249	32067	50	15634

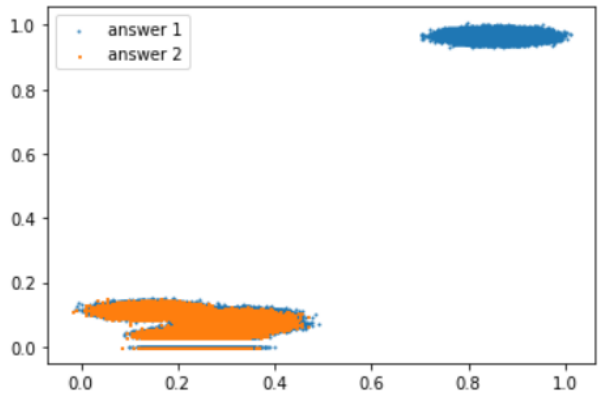
Clustering

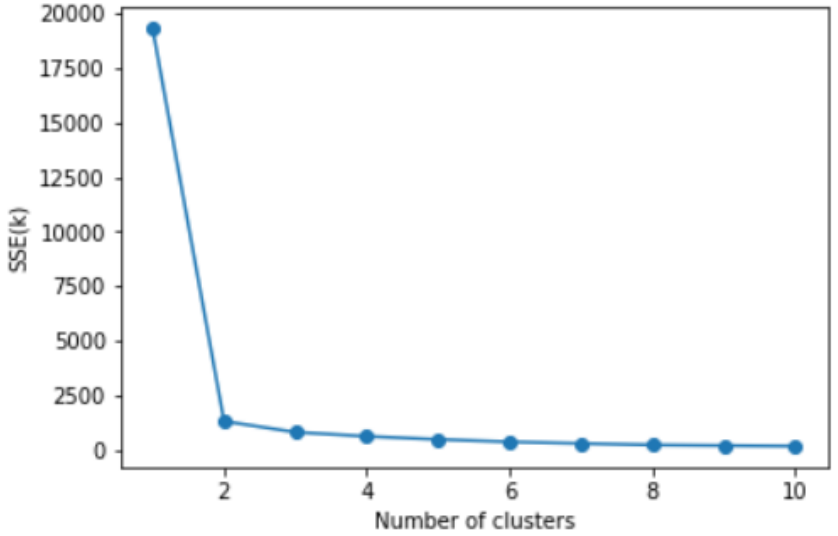
- Cluster 수는 2개가 적합해 보임
- Test set에 대하여 예측값과 실제값이

  다른 양상을 띠어, 좋지 못한 분류 방법

학습시간 0.29초, 예측시간 1.96초








```
confusion_matrix(y_km, y_test)
```

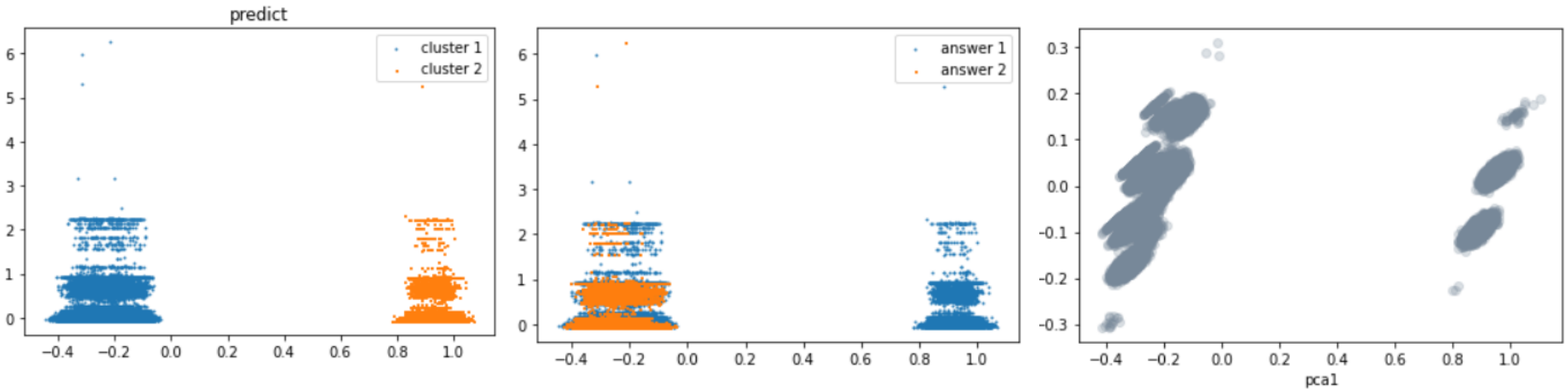
```
array([[424277, 177067],
       [150336,      0]])
```

Clustering with PCA

- PCA로 차원축소하여 종속변수의 가장 많은 분산을 설명하는 두 components를 선택
- Test set에 대하여 예측값과 실제값이 다른 양상을 띠어 좋지 못한 분류 방법
- 학습시간 : 2.06초, 예측시간 : 3.08초

```
confusion_matrix(y_km, y_test)
```

```
array([[424277, 177067],
       [150336,      0]])
```



Isolation Forest

- GridSearchCV()를 통해 초모수를 결정

```
gs.best_params_
```

```
{'bootstrap': True, 'contamination': 0.0001, 'max_features': 2}
```

	precision	recall	f1-score	support
0.0	1.00	0.76	0.87	751017
1.0	0.00	0.01	0.00	663
accuracy			0.76	751680
macro avg	0.50	0.39	0.43	751680
weighted avg	1.00	0.76	0.87	751680

- 학습시간 : 2.96초, 예측시간 : 43.60초