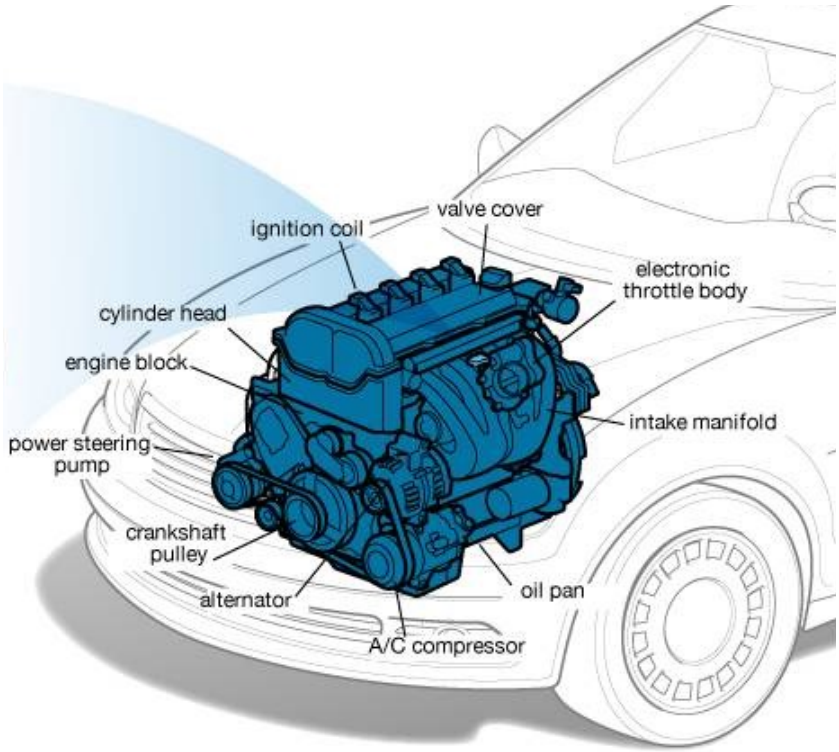
자동차 엔진 주변 센서 값 학습을 통한 엔진 상태 예측

엔진 주변 500개 센서 값 학습을 통한 엔진 상태 예측 모델 만들기



2023. 08. 22

박성현

분석 배경

❑ **문제 발생 장소** : Ford 자동차 제조사

❑ **문제 발생 공정** : 엔진 상태 검사 공정

❑ **문제 발생 내용** :

1) 자동차에서 엔진 이상 여부를 검사하기 위해 엔진을 들어내는 것은 큰 손실 발생

2) 고도로 숙련된 작업자만 엔진 이상여부를 판단 할 수 있음

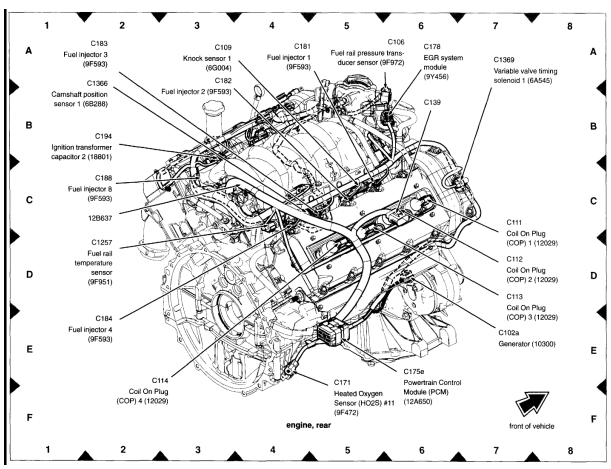
3) 판단이 잘 못 될 경우, 차량 운행에서 문제가 발생 될 수 있음

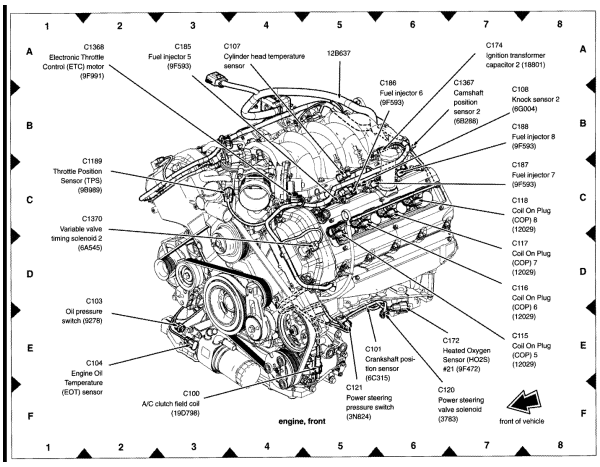
❑ **문제 해결 방안 :**

1) 자동차에서 엔진을 들어내지 않고도 엔진 이상여부를 판단 할 수 있어야 함

2) 객관적인 데이터로 검사가 될 수 있어야 함

→ 엔진 주변 500개 센서 값에 의한 엔진 이상 여부 판단 (자동 검사 가능)





분석 목적

➢ 엔진을 차량에서 들어내지 않고도 객관적인 센서 데이터로 엔진상태를 판단 할 수 있는 모델 개발

분석 효과

➢ 엔진을 차량에서 들어내지 않으므로 시간과 비용의 손실을 줄일 수 있음

➢ 고도로 숙련된 작업자가 필요 하지 않으므로 검사에 제약이 없음

➢ 객관적인 데이터로 자동으로 판정하므로 사람의 실수가 없음

학습 결과

❑ 결과 요약

- 베스트 모델 : **CNN 모델 (Convolutional Neural Network)**
- 베스트 모델 정확도 : **97.4%**

❑ 모델별 평가 결과

Model	Accuracy	Precision	Recall	F1 Score	AUC
WeightedEnsemble_L2	84.7%	84.4%	83.9%	84.1%	92.0%
XGBoost	77.1%	77.4%	74.3%	75.8%	85.3%
LightGBM	78.6%	78.3%	77.2%	77.7%	86.4%
CatBoost	84.6%	85.3%	82.3%	83.8%	91.3%
RandomForest	73.3%	74.1%	69.0%	71.5%	80.7%
LogisticRegression	49.6%	47.8%	45.9%	46.8%	48.7%
CNN	97.4%	97.0%	97.5%	97.3%	99.4%

- CNN 모델은 주로 이미지나 지역적 패턴이 있는 데이터를 학습 및 예측할 때 사용되는 모델로, 분석 대상인 데이터셋도 인접 하는 센서 값과 함께 패턴을 이루는 특성이 있기 때문에 CNN 모델에서 효과적으로 학습 되는 것으로 보임

❑ 실제 공정에 적용 방법

- 자동 검사를 초기에 100% 신뢰할 수 없기 때문에 기존 검사자와 검사를 병행하며 모델을 개선, 점차 검사자의 검사 비율을 줄여가는 방식으로 공정에 적용 가능

[실제 공정 적용 예시]

분석 환경

❑ **사용 언어** : Python3

❑ **사용 패키지** : numpy, pandas, matplotlib, seaborn, sklearn, tensorflow, xgboost, lightgbm, catboost

❑ **분석 환경** : [CPU] Apple M2, [RAM] 8GB, [GPU] T4 (colab)

학습 데이터셋 구조 및 개수

❑ **데이터 형태** : ARFF 파일 (정형 데이터)

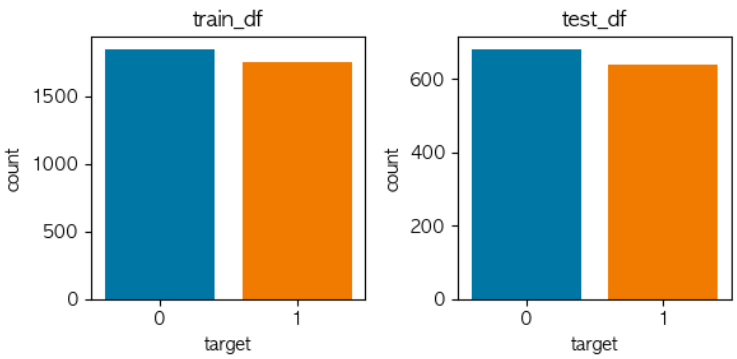
❑ **수집 장소** : Ford 자동차 제조사 (데이터 분류 대회 오픈 데이터셋)

❑ **데이터 개수** : 4,921개 (train : 3,601개, test : 1,320개)

❑ **데이터 용량** : 30MB

❑ **데이터 Features** : 센서 500개의 측정 값 (소음, 압력, 진동, 온도 등)

❑ **데이터 Class** : 비정상(0), 정상(1)



- `train_df_0_class : 1846`
- `train_df_1_class : 1755`
- `test_df_0_class : 681`
- `test_df_1_class : 639`

- 비정상 : class_0
- 정상 : class_1

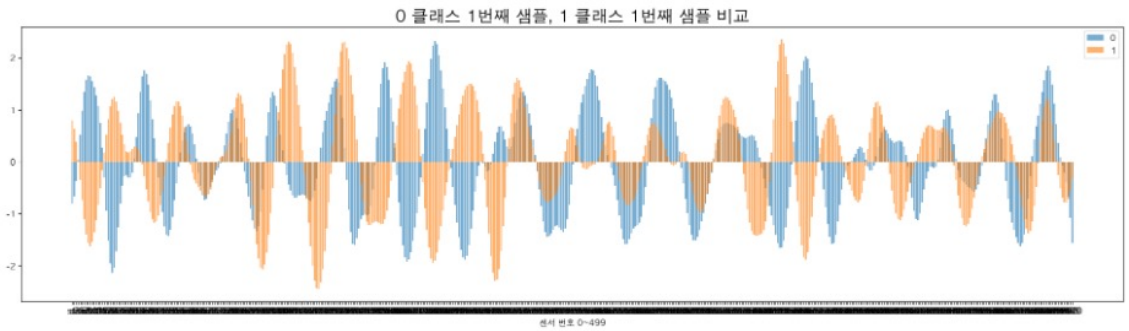
→ 클래스 빈도의 균형은 맞음

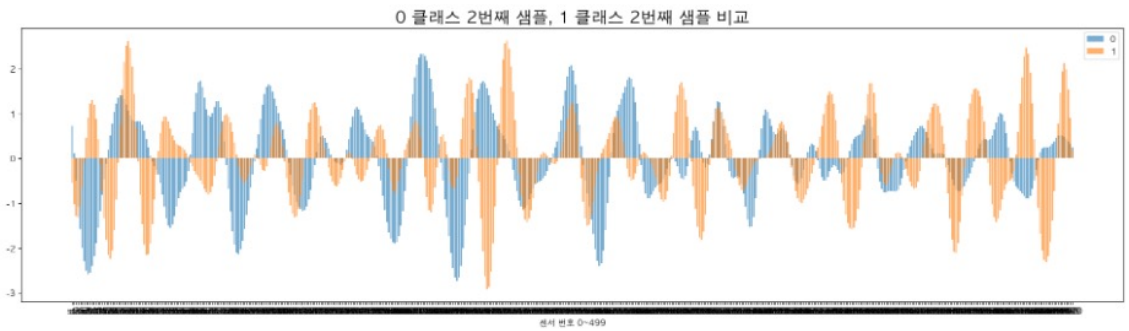
데이터 분석(EDA) 및 전처리

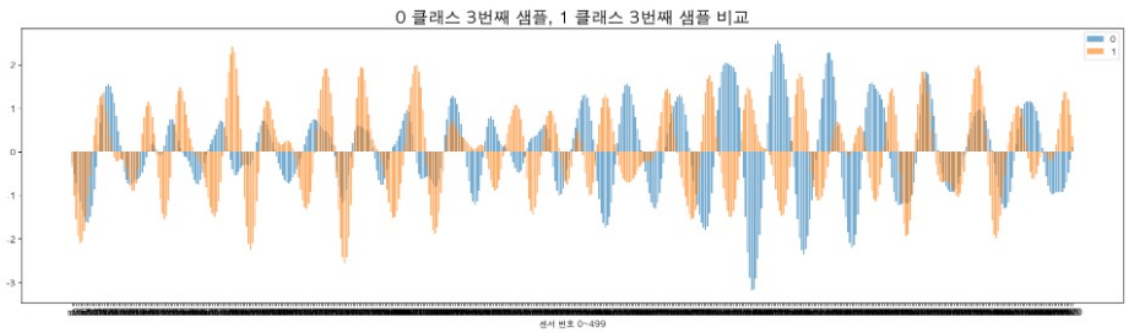
❑ 0과 1 클래스 각 샘플의 센서 값 분포 비교 (각 3개 샘플 비교)

```
# 클래스가 0, 1인 인덱스 추출
idx_0 = train_df.loc[train_df.target==0].index
idx_1 = train_df.loc[train_df.target==1].index

# 가로: 센서 번호, 세로: 센서 값
for i in range(3): # 비교 하고 싶은 개수 설정 (3개 비교)
    plt.figure(figsize=(24,6))
    plt.bar(train_df.columns[:-1], train_df.iloc[idx_0[i], :-1], label='0', alpha=0.6)
    plt.bar(train_df.columns[:-1], train_df.iloc[idx_1[i], :-1], label='1', alpha=0.6)
    plt.title(f'0 클래스 {i+1}번째 샘플, 1 클래스 {i+1}번째 샘플 비교', fontsize=20)
    plt.xlabel('센서 번호 0~499')
    plt.legend()
    plt.show()
```

✓ 3.9s MagicPython







- 클래스별 센서 값에 차이가 나타남 -> 모델 학습 가능

데이터 분석(EDA) 및 전처리

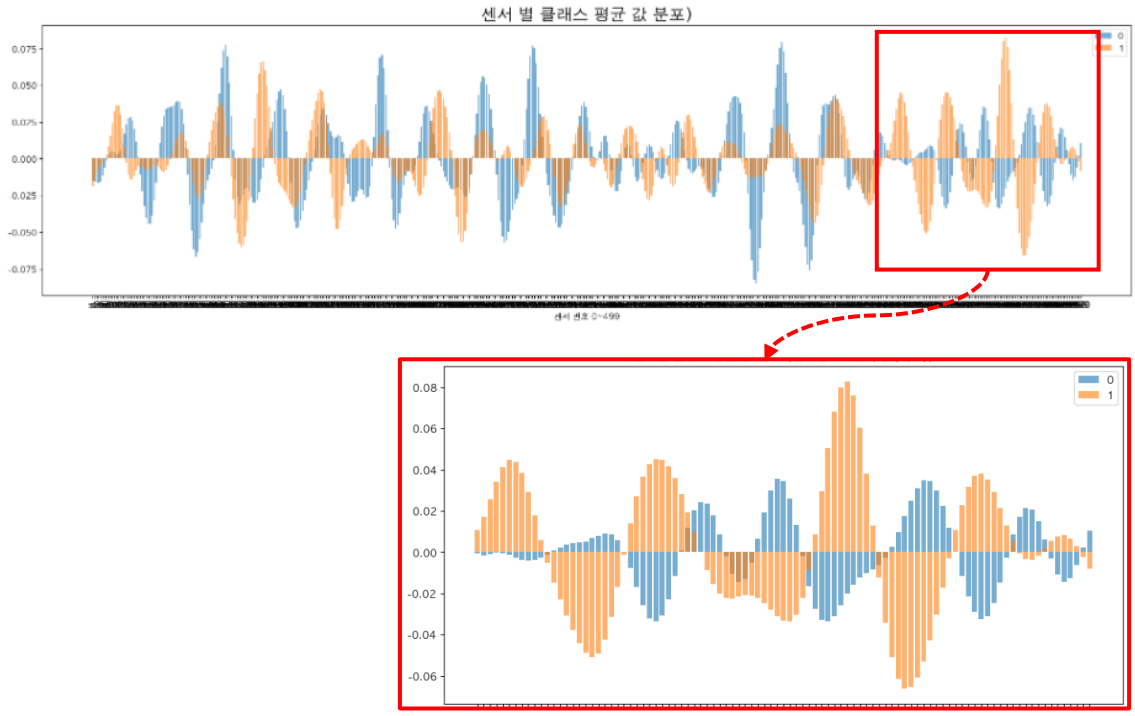
❑ 센서별 클래스 평균 값 분포

```
# 각 센서의 클래스별(0,1) 평균 값
s_means = train_df.groupby(by='target').mean()
```

✓ 0.0s MagicPython

```
# 가로: 센서 번호, 세로: 센서 별 클래스 평균 값
#   -> 센서 별로 2개의 클래스(0, 1)
plt.figure(figsize=(20,6))
plt.bar(s_means.columns, s_means.iloc[0,:], alpha=0.6, label='0')
plt.bar(s_means.columns, s_means.iloc[1,:], alpha=0.6, label='1')
plt.title('센서 별 클래스 평균 값 분포', fontsize=20)
plt.xlabel('센서 번호 0~499')
plt.legend()

plt.show()
```

✓ 1.2s MagicPython

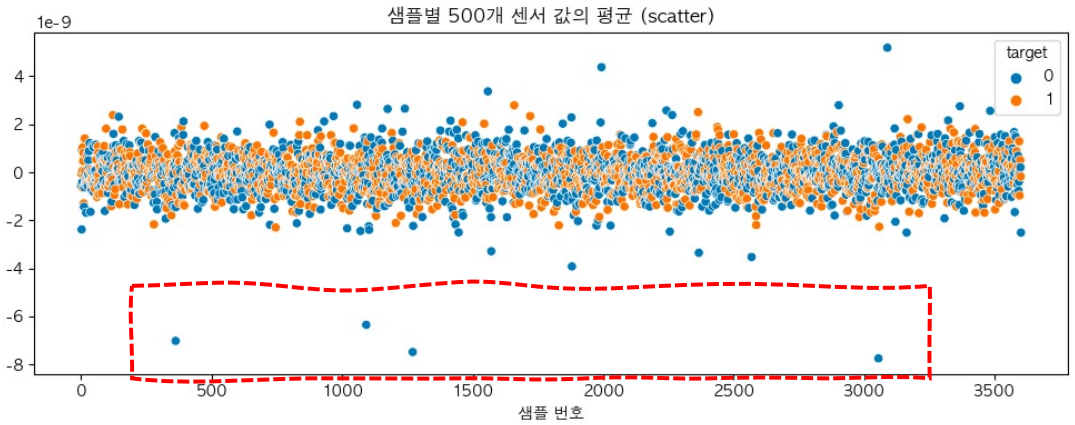


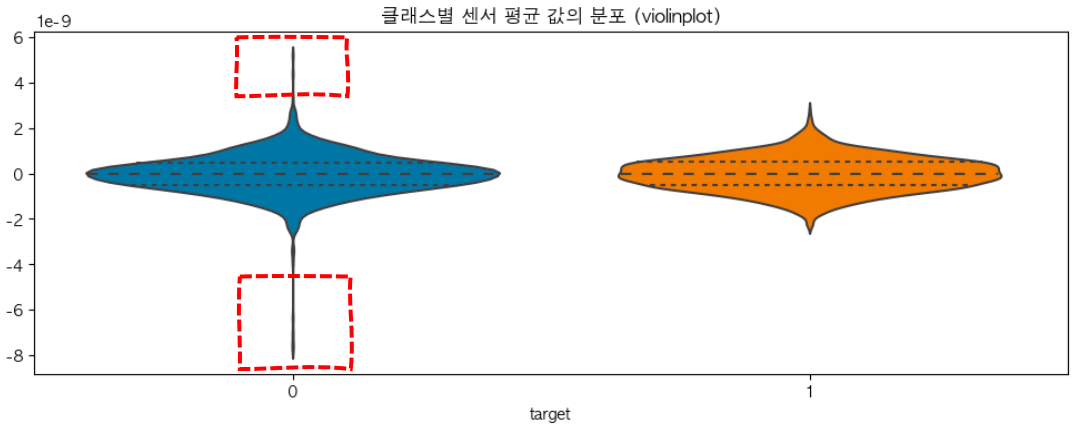
- 후반부 센서 (s403~)에서 클래스에 따라 센서 값이 양수와 음수로 구분 됨
- 후반부 센서만 학습시켜볼 필요 있음

데이터 분석(EDA) 및 전처리

❑ 샘플들의 전체 센서 값 평균

```
# 가로: 샘플 번호, 세로: 500개 센서 값 평균

# 클래스별 데이터프레임 생성
train_0 = train_df.loc[train_df.target == 0]
train_1 = train_df.loc[train_df.target == 1]

# 샘플별 500개 센서 값의 평균 (scatter)
plt.figure(figsize=(12,4))
sns.scatterplot(x=train_df.index, y=train_df.iloc[:,:-1].mean(axis=1),
                hue=train_df.target)
plt.title('샘플별 500개 센서 값의 평균 (scatter)')
plt.xlabel('샘플 번호')

# 클래스별 센서 평균 값의 분포 (violinplot)
plt.figure(figsize=(12,4))
sns.violinplot(x=train_df.target, y=train_df.iloc[:, :-1].mean(axis=1), inner="quart")
plt.title('클래스별 센서 평균 값의 분포 (violinplot)')
plt.xlabel('target')
```



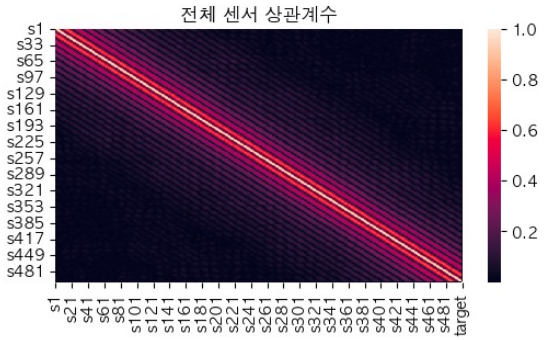


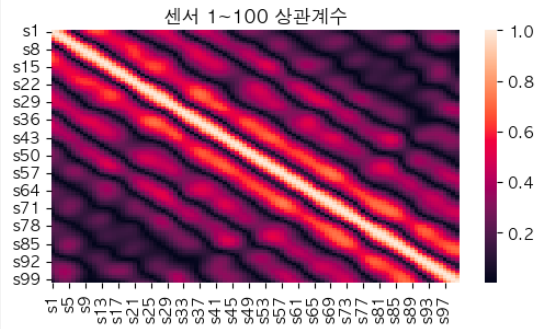
- 0클래스 데이터에 노이즈 샘플이 있음 (red box 부분)
- 노이즈 샘플 제거 후 학습시켜볼 필요 있음

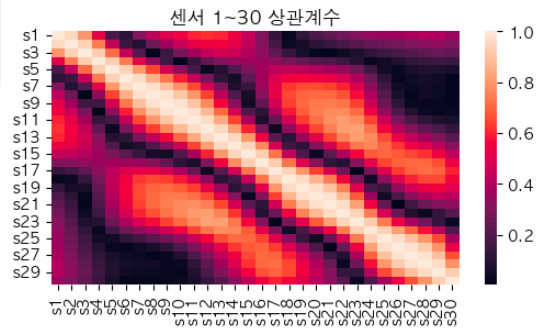
데이터 분석(EDA) 및 전처리

❑ 센서별 target과 상관 관계 분석

```
# target과 각 센서 값의 상관관계 분석

# 전체 센서 상관계수
plt.figure(figsize=(6,3))
plt.title('전체 센서 상관계수')
train_corr = abs(train_df.corr())
sns.heatmap(train_corr)

# 센서 1~100 상관계수
plt.figure(figsize=(6,3))
plt.title('센서 1~100 상관계수')
corr_100 = abs(train_df.iloc[:100, :100].corr())
sns.heatmap(corr_100)

# 센서 1~30 상관계수
plt.figure(figsize=(6,3))
plt.title('센서 1~30 상관계수')
corr_30 = abs(train_df.iloc[:30, :30].corr())
sns.heatmap(corr_30, xticklabels=1)

fig.tight_layout()
plt.show()
```

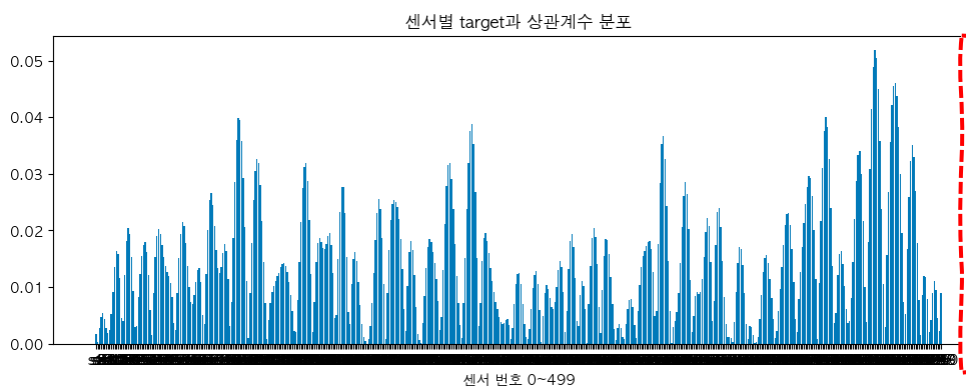






- 인접한 3개의 센서가 밀접한 연관성이 있다.
  → CNN 으로 학습 시 Kernel size를 3으로 하면 효과가 있을 것으로 예상

- Target과 상관관계 상위 센서
  → 앞서 확인 한대로 후반부 센서가 target 과 상관관계가 높다. (s430 이후)



target과 상관관계 상위 센서

s461	0.051789
s462	0.050375
s460	0.048899
s473	0.045961
s472	0.045462
s463	0.044951
s474	0.043726
s471	0.042066
s459	0.041494
s432	0.040031

학습 모델 구축 및 예측

❑ Auto ML (AutoGluon) 모델 학습 결과

Best Model : WeightedEnsemble_L2 (정확도 : 84.7%)

```
## AutoGluon ML로 학습 및 예측, 정확도 출력
from autogluon.tabular import TabularPredictor

# TabularPredictor.fit() 함수는 자동으로 훈련 데이터를 훈련 데이터와 검증 데이터로 분할(기본: 20%)
predictor = TabularPredictor(label='target').fit(train_df) # train dataset 학습시키기
y_pred = predictor.predict(x_test) # test dataset 결과 예측
proba_df = predictor.predict_proba(x_test)
probs = proba_df.iloc[:, 1].values # 클래스 1의 예측 확률

# 평가지표 계산
accuracy = accuracy_score(y_test, y_pred) # 정확도
precision = precision_score(y_test, y_pred) # 정밀도: 양성이라고 예측한것중 실제 양성의 비율
recall = recall_score(y_test, y_pred) # 재현율(민감도): 실제 양성 중 양성으로 예측된 비율
f1 = f1_score(y_test, y_pred) # f1: 2 * (정밀도*재현율)/(정밀도+재현율)
auc_score = roc_auc_score(y_test, probs)  # AUC 계산 (ROC 곡선 아래의 면적)

# 평가지표 출력
print(f"{predictor.get_model_best()} Metrics")
print(f" - Accuracy: {round(accuracy*100,2)}%")
print(f" - Precision: {round(precision*100,2)}%")
print(f" - Recall: {round(recall*100,2)}%")
print(f" - F1 Score: {round(f1*100,2)}%")
print(f" - AUC: {round(auc_score*100,2)}%")

fig, axs = plt.subplots(1, 2, figsize=(12,4))

# 혼동 행렬(Confusion Matrix)
confusion = confusion_matrix(y_test, y_pred)
# 히트맵으로 시각화
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues",
            xticklabels=['0', '1'], yticklabels=['0', '1'], ax=axs[0])
axs[0].set_ylabel('Actual')
axs[0].set_xlabel('Predicted')
axs[0].set_title('혼동 행렬(Confusion Matrix)')

# ROC 그래프 그리기
fpr, tpr, thresholds = roc_curve(y_test, probs) #fpr, tpr, thresholds 값 변수에 할당
axs[1].plot(fpr, tpr, color='red', label=f'AUC = {auc_score:.2f}') # ROC Curve 그리기
axs[1].plot([0, 1], [0, 1], color='blue', linestyle='--')
axs[1].set_title(f'{predictor.get_model_best()} Curve')
axs[1].set_xlabel('FPR')
axs[1].set_ylabel('TPR')
axs[1].legend(loc="lower right")

# 최적의 threshold 값 찾기
optimal_idx = np.argmax(tpr - fpr) # tpr-fpr 차이가 가장 큰 인덱스 찾기
optimal_threshold = thresholds[optimal_idx] # 찾은 인덱스에 해당하는 threshold
# optimal threshold 값을 사용하여 예측
y_pred_optimal = np.where(probs > optimal_threshold, 1, 0)
# 새로운 예측 값을 사용하여 정확도 계산
accuracy_optimal = accuracy_score(y_test, y_pred_optimal)
print(f'threshold {optimal_threshold:.2f} -> Accuracy : {round(accuracy_optimal*100, 2)}%
```

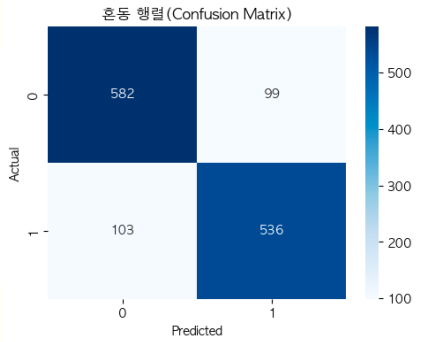
학습 모델 구축 및 예측

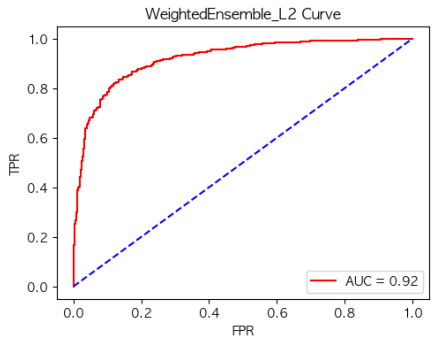
❑ Auto ML (AutoGluon) 모델 학습 결과

Best Model : WeightedEnsemble_L2 (정확도 : 84.7%)

[평가지표, 혼동행렬, ROC Curve]

```
WeightedEnsemble_L2 Result
 - Accuracy: 84.7%
 - Precision: 84.41%
 - Recall: 83.88%
 - F1 Score: 84.14%
 - AUC: 91.95%
threshold 0.53 -> Accuracy : 85.08%
```





[Leader Board]

	model	score_test	score_val	pred_time_test	pred_time_val	fit_time	pred_time_test_marginal	p
0	WeightedEnsemble_L2	0.846970	0.878	0.033788	0.024838	30.755167	0.002213	
1	CatBoost	0.833333	0.844	0.014045	0.013746	27.483624	0.014045	
2	LightGBMXT	0.823485	0.846	0.031736	0.012811	6.750521	0.031736	
3	NeuralNetTorch	0.815152	0.852	0.017531	0.009969	2.951555	0.017531	
4	XGBoost	0.805303	0.832	0.049488	0.020155	17.854703	0.049488	
5	NeuralNetFastAI	0.798485	0.822	0.034888	0.015098	2.355973	0.034888	
6	LightGBM	0.788636	0.820	0.028717	0.013888	11.991835	0.028717	
7	LightGBMLarge	0.788636	0.798	0.028761	0.015605	24.211305	0.028761	
8	ExtraTreesEntr	0.767424	0.788	0.050318	0.022893	0.532084	0.050318	
9	RandomForestEntr	0.749242	0.766	0.047728	0.022684	2.963529	0.047728	
10	ExtraTreesGini	0.747727	0.748	0.049245	0.021267	0.546680	0.049245	
11	RandomForestGini	0.735606	0.772	0.060625	0.019617	2.480748	0.060625	
12	KNeighborsUnif	0.718939	0.718	0.072461	0.040759	0.079863	0.072461	
13	KNeighborsDist	0.718939	0.718	0.087420	0.034975	0.054802	0.087420	

학습 모델 구축 및 예측

❑ 4가지 머신러닝 모델 학습 결과 (XGBoost, LightGBM, CatBoost, RandomForest)

Best Model : CatBoost (정확도 : 84.55%)

```
##  ML 모델별로 학습 및 예측 결과

# 학습 및 예측 모델 4가지 리스트 (XGBoost, LightGBM, CatBoost, RandomForest)
model = [xgb.XGBClassifier(), LGBMClassifier(),
         CatBoostClassifier(verbose=0), RandomForestClassifier()]

# 4개 ML 모델 학습 후 정확도, 혼동행렬, ROC 출력하기
for i in range(len(model)): # 모델 리스트 크기만큼 반복
    model[i].fit(x_train, y_train) # 모델 한개씩 불러와서 학습
    y_pred = model[i].predict(x_test) # 각 모델 예측값
    probs = model[i].predict_proba(x_test)[:, 1] # 각 모델 클래스 1의 예측 확률

    # 평가지표 계산
    accuracy = accuracy_score(y_test, y_pred) # 정확도
    precision = precision_score(y_test, y_pred) # 정밀도: 양성이라고 예측한것중 실제 양성의 비
    recall = recall_score(y_test, y_pred) # 재현율(민감도): 실제 양성 중 양성으로 예측된 비율
    f1 = f1_score(y_test, y_pred) # f1: 2 * (정밀도*재현율)/(정밀도+재현율)
    auc_score = roc_auc_score(y_test, probs)  # AUC 계산 (ROC 곡선 아래의 면적)

    # 각 모델의 혼동 행렬을 히트맵으로 그리기
    conf_mat = confusion_matrix(y_test, y_pred) # 혼동 행렬 만들기 (2,2)
    sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', ax=axs[i])
    axs[i].set_title(model_list[i])
    axs[i].set_ylabel('Actual')
    axs[i].set_xlabel('Predicted')

    # 각 모델의 ROC Curve 그리기
    fpr, tpr, thresholds = roc_curve(y_test, probs) # fpr, tp, thresholds 값 할당
    axs2[i].plot(fpr, tpr, color='red', label=f'AUC = {auc_score:.2f}') # ROC Curve
    axs2[i].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    axs2[i].set_title(model_list[i])
    axs2[i].set_xlabel('FPR')
    axs2[i].set_ylabel('TPR')
    axs2[i].legend(loc="lower right")

    # 최적의 threshold 값 찾기
    optimal_idx = np.argmax(tpr - fpr) # tpr-fpr 차이가 가장 큰 인덱스 찾기
    optimal_threshold = thresholds[optimal_idx] # 찾은 인덱스에 해당하는 threshold
    # optimal threshold 값을 사용하여 예측
    y_pred_optimal = np.where(probs > optimal_threshold, 1, 0)
    # 새로운 예측 값을 사용하여 정확도 계산
    accuracy_optimal = accuracy_score(y_test, y_pred_optimal)
    print(f'threshold {optimal_threshold:.2f} -> Accuracy : {round(accuracy_optimal
```

- 출력 관련 코드는 생략

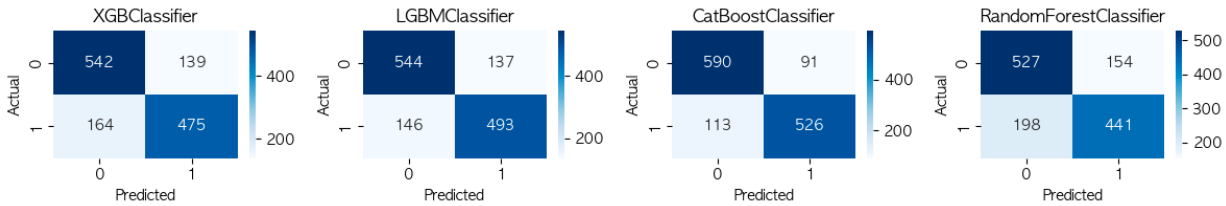
학습 모델 구축 및 예측

❑ 4가지 머신러닝 모델 학습 결과 (XGBoost, LightGBM, CatBoost, RandomForest)

Best Model : CatBoost (정확도 : 84.55%)

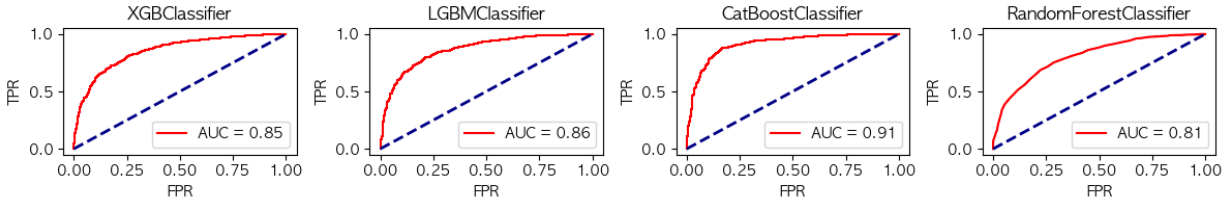
[각 모델의 혼동 행렬]

〔각 모델의 혼동 행렬〕



[각 모델의 ROC Curve]

〔각 모델의 ROC Curve〕



[각 모델의 평가 지표]

XGBClassifier:
- Accuracy: 77.05%
- Precision: 77.36%
- Recall: 74.33%
- F1 Score: 75.82%
- AUC: 85.26%

threshold 0.42 -> Accuracy : 77.42%

LGBMClassifier:
- Accuracy: 78.56%
- Precision: 78.25%
- Recall: 77.15%
- F1 Score: 77.7%
- AUC: 86.37%

threshold 0.49 -> Accuracy : 79.09%

CatBoostClassifier:
- Accuracy: 84.55%
- Precision: 85.25%
- Recall: 82.32%
- F1 Score: 83.76%
- AUC: 91.25%

threshold 0.46 -> Accuracy : 85.3%

RandomForestClassifier:
- Accuracy: 73.33%
- Precision: 74.12%
- Recall: 69.01%
- F1 Score: 71.47%
- AUC: 80.7%

threshold 0.49 -> Accuracy : 73.03%

학습 모델 구축 및 예측

❑ Logistic Regression 모델 학습 결과

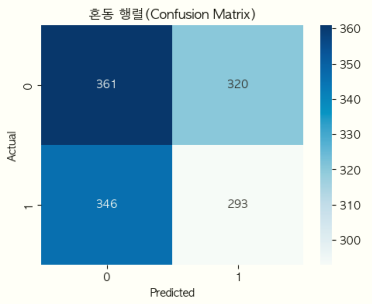
학습 안 됨 (정확도 : 49.55%)

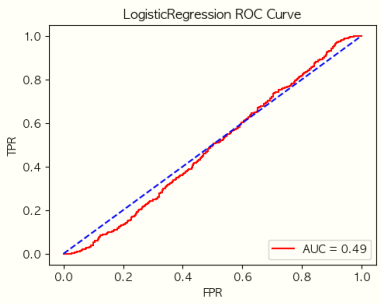
```
## logistic regression

# 로지스틱 회귀 모델 생성 및 학습
model = LogisticRegression(max_iter=1000)  # 최대 반복 횟수를 설정
model.fit(x_train, y_train)
y_pred = model.predict(x_test) # test dataset 결과 예측
probs = model.predict_proba(x_test)[:, 1]  # 클래스 1에 대한 예측 확률

# 평가지표 계산
accuracy = accuracy_score(y_test, y_pred) # 정확도
precision = precision_score(y_test, y_pred) # 정밀도: 양성이라고 예측한것중 실제 양성의 비율
recall = recall_score(y_test, y_pred) # 재현율(민감도): 실제 양성 중 양성으로 예측된 비율
f1 = f1_score(y_test, y_pred) # f1: 2 * (정밀도*재현율)/(정밀도+재현율)
auc_score = roc_auc_score(y_test, probs)  # AUC 계산 (ROC 곡선 아래의 면적)
```

- 출력 관련 코드는 생략

[평가지표, 혼동행렬, ROC Curve]

```
Logistic Regression metrics
  - Accuracy: 49.55%
  - Precision: 47.8%
  - Recall: 45.85%
  - F1 Score: 46.81%
  - AUC: 48.7%
```





- 정확도 49.55%, AUC 48.7% 로 Logistic Regression 모델로는 학습이 안 됨

학습 모델 구축 및 예측

❑ CNN (Convolutional Neural Network) 모델 학습 결과

Best Model : CNN (정확도 : 97.35%)

```
## CNN 학습 및 예측 결과

# CNN 학습을 위해 데이터 shape 변환
x_train_exp = np.expand_dims(x_train, -1) # x_train 배열의 마지막 차원에 새로운 축을 추가
x_test_exp = np.expand_dims(x_test, -1) # x_test 배열의 마지막 차원에 새로운 축을 추가

# 하이퍼파라미터 설정
epoch = 400 # 학습 반복 횟수
act = 'LeakyReLU' # 활성화 함수
opt = 'adam' # optimizer
filter = 30 # Conv1D Layer의 filter수
batch = 16 # batch size
val_rate = 0.18 # 검증데이터 비율
kernel = 3 # filter의 kernel size

# 모델 정의(구조)
def make_cnn_model():
    model = Sequential()
    # Conv1D 1Layer
    model.add(Conv1D(filters=filter, kernel_size=kernel, activation=act,
                     padding='same', input_shape=(500, 1)))
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화

    # Conv1D 2Layer
    model.add(Conv1D(filters=filter, kernel_size=kernel, activation=act,
                     padding='same'))
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화

    # Conv1D 3Layer
    model.add(Conv1D(filters=filter, kernel_size=kernel, activation=act,
                     padding='same'))
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화

    # GlobalAveragePooling1D Layer
    model.add(GlobalAveragePooling1D())
    # - 각 feature map의 평균을 계산하여 차원을 축소
    # - Flatten() 대신 사용하여 파라미터 수를 줄이고 과적합을 방지

    # Dense Layer
    model.add(Dense(2, activation='softmax'))
    return model

cnn_model = make_cnn_model() # 모델 초기화

# 모델 컴파일
cnn_model.compile(optimizer=opt, # optimizer: 학습 최적화 알고리즘
                  loss='sparse_categorical_crossentropy', # 사용할 손실 함수
                  metrics=['sparse_categorical_accuracy']) # 모델의 평가 지표

# monitor 지표를 기준으로 베스트 모델을 저장
callbacks = [ModelCheckpoint('best_model.h5', save_best_only=True,
                             monitor='val_sparse_categorical_accuracy', mode='max')]

history = cnn_model.fit(x_train_exp, y_train, # 학습데이터, 라벨
                        batch_size=batch, # batch 크기 지정
                        validation_split=val_rate, # 검증용 데이터의 비율 지정
                        epochs=epoch, # 학습 반복 횟수
                        callbacks=callbacks) # 검증 정확도가 올라갈때만 모델 저장
```

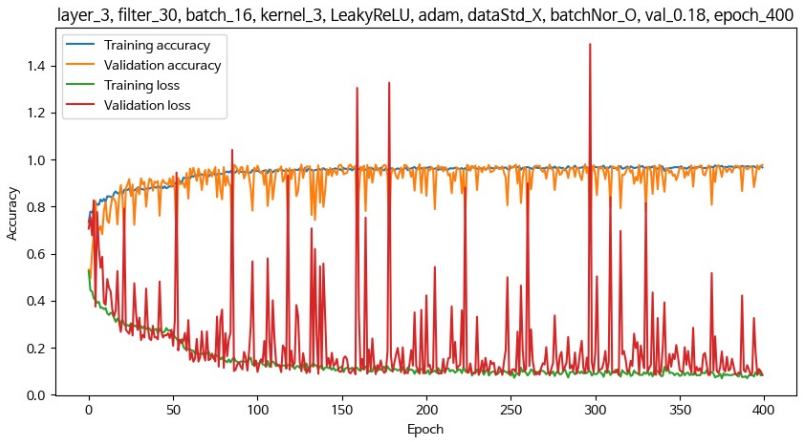
- 출력 및 평가지표 계산 관련 코드는 생략

학습 모델 구축 및 예측

❑ CNN (Convolutional Neural Network) 모델 학습 결과

Best Model : CNN (정확도 : 97.35%)

[Accuracy, Loss gragh]



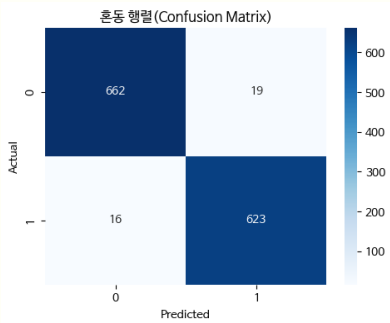
- Training accuracy
- Validation accuracy
- Training loss
- Validation loss

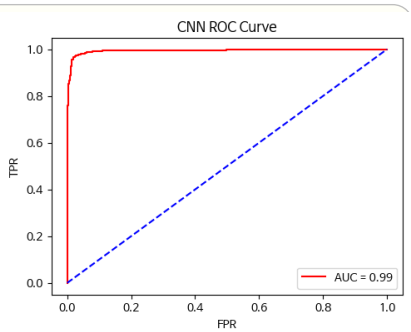
- 과적합이 일어나지 않고, epoch 400 동안 잘 학습 되었음

[평가지표, 혼동행렬, ROC Curve]

[CNN Model metrics]

- Accuracy: 97.35%
- Precision: 97.04%
- Recall: 97.5%
- F1 Score: 97.27%
- AUC: 99.43%





[하이퍼 파라미터 튜닝 최적 조건 : 파란색 셀]

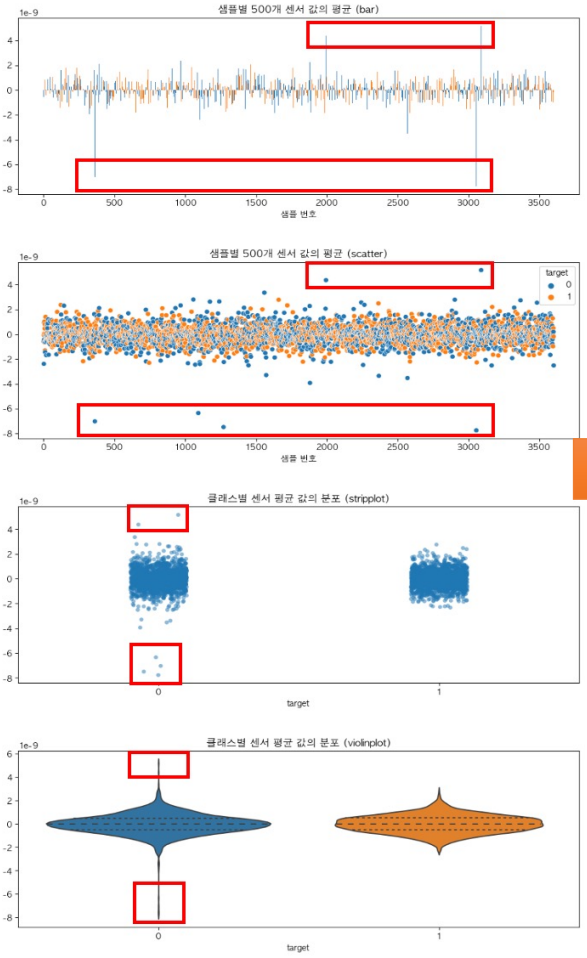
하이퍼파라미터	조건1	조건2	조건3	조건4
Conv 1D 레이어수	2	**3**	4	5
활성함수	relu	swish	softplus	**LeakyReLU**
optimizer	**adam**	SGD		
필터 수	16	**30~32**	64	
Batch_size	**16~24**	32	64	128
최적모델 저장 기준 (monitor)	**val_sparse_categorical_accuracy**	val_loss		
검증 데이터 비중	10%	**16~20%**	30%	
kernel_size	2	**3**	4	5
데이터 정규화 StandardScaler()	O	**X**		
배치 정규화 BatchNormalization()	**O**	X		
epoch	200	300	**400**	

데이터 전처리 후 재학습 (1)

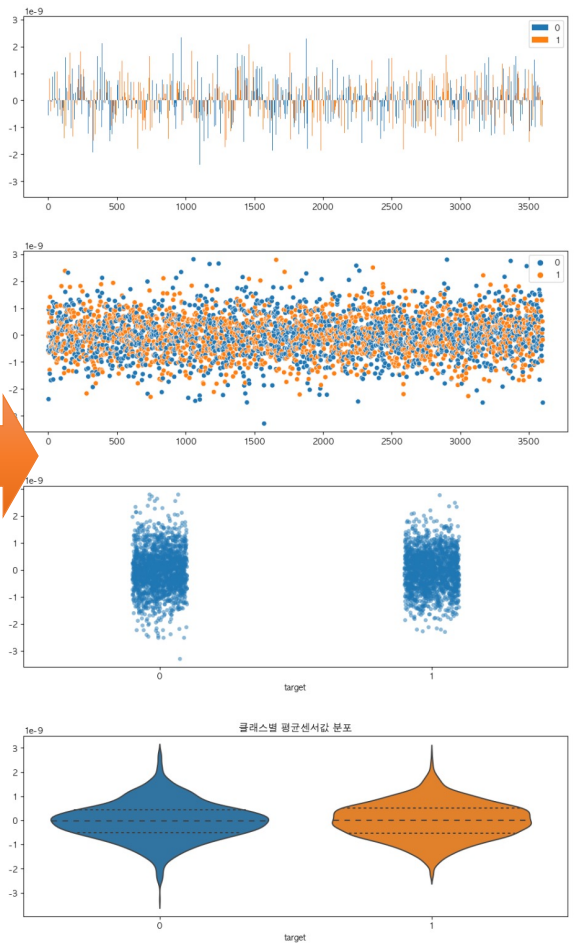
❑ 노이즈 샘플 제거 후 학습 결과 : 성능 개선 되지 않음 (효과 없음)

- 전체 센서의 평균 값이 높은 노이즈 샘플 10개 제거 후 시각화

[노이즈 제거 전]



[노이즈 제거 후]

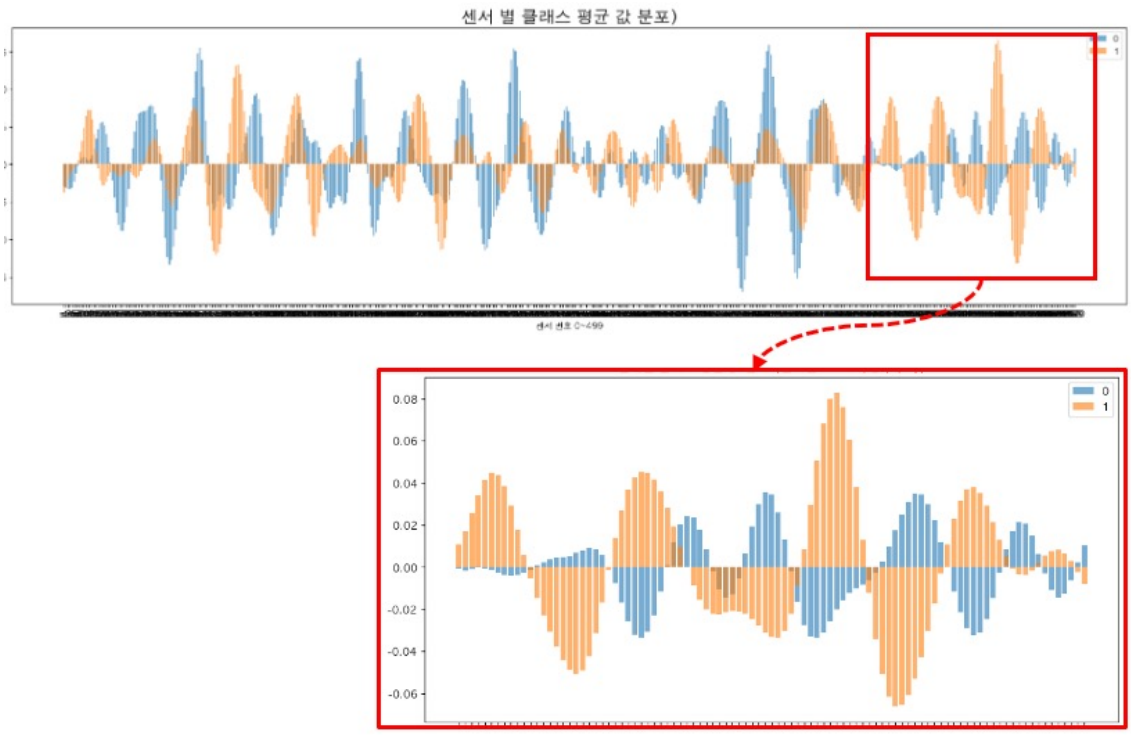


- 평균 값이 높은 상위 5, 10, 20개 샘플 제거 후 학습 및 예측 결과
  - ➢ 성능 개선 되지 않음
  - ➢ 머신러닝 모델 : 예측 정확도 비슷하거나 약간 낮아짐
  - ➢ CNN 모델 : 예측 정확도 비슷함

데이터 전처리 후 재학습 (2)

❑ 후반부 센서(s403~s500)만 학습 후 예측 결과 : 성능 개선 되지 않음 (효과 없음)

- 403번 센서부터 0과 1 클래스의 평균 값이 음수와 양수로 구분 되는 특징



- 센서 번호 s403 ~ s500 만을 학습하여 예측한 결과
  - ➢ 성능 개선 되지 않음
  - ➢ 데이터 부족으로 과소적합 발생
  - ➢ 머신러닝 모델 : 예측 정확도가 모델별로 2 ~ 10% 감소
  - ➢ CNN 모델 : 예측 정확도 7% 감소

학습 완료

❑ 학습 결과 요약은 2페이지 참조