

CNC 가공 데이터 학습을 통한 결과 예측 모델

- ❖ 공정이 완료 되기 전 미리 검사 결과를 예측하여
공정 손실 줄이기



2023. 09. 06

박성현

분석 배경

□ 문제 발생 장소 : 자동차 부품사 CNC 가공 공정

□ CNC 가공이란 :

- 1) CNC 시스템을 사용하여 회전하는 작업물을 절삭하는 공정
- 2) 정밀하고 복잡한 형상을 가진 부품을 가공할 수 있어 다양한 산업 분야에 사용됨



□ CNC 가공 Flow chart



□ 문제 발생 내용 :

- 1) CNC 절삭 공구가 마모되거나, 마모 한계치에 도달하게 되면 가공 정밀도가 떨어짐
- 2) 설비 셋팅 조건도 가공 정밀도에 영향을 미침
- 3) 이러한 문제점을 가공이 완료 된 이후 외관검사를 통해서만 알 수 있음
- 4) 문제 파악이 늦어져서 이미 생산 된 제품에 대한 손실이 발생함

분석 목적

➤ CNC 가공 공정 데이터를 학습하여 공정이 완료 되기 전에 미리 제품의 판정 결과를 예측 할 수 있는 모델 개발

분석 효과

➤ 가공 결과를 미리 예측하여 문제 발생 시 즉시 조치 할 수 있음
➤ 불량 수량을 줄여 손실을 줄이고, 생산성을 향상 시킬 수 있음

학습 결과 및 적용 방안

□ 결과 요약

- 베스트 모델 : **CatBoost Classifier**
- 베스트 모델 정확도 : **100%**

□ 학습 모델별 평가 결과

Model	Accuracy	Precision	Recall	F1 Score	AUC
XGBoost	99.97%	99.97%	99.99%	99.98%	100.00%
LightGBM	99.99%	99.99%	100.00%	99.99%	100.00%
CatBoost	100.00%	100.00%	100.00%	100.00%	100.00%
RandomForest	99.93%	99.91%	99.99%	99.95%	100.00%
LogisticRegression	82.68%	85.95%	90.24%	88.04%	89.37%
DNN	99.69%	99.75%	99.81%	99.78%	99.99%
CNN	99.74%	99.87%	99.76%	99.82%	99.99%

- 트리 기반 머신러닝 알고리즘이 DNN, CNN 등 딥러닝 모델보다 정확도가 높게 나타남
→ Feature와 Label 간의 연관성이 복잡하지 않은 경우
- 예측 정확도가 100%를 달성 했기 때문에 검사의 신뢰도만 확보 된다면 자동검사를 도입하여 외관검사를 점차 줄여 나갈 수 있음

□ 실제 공정에 적용 방법

- 자동 검사 설비를 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 형태 및 개수

- 데이터 형태 : CSV 파일 26개 (정형 데이터)
- 25개 : 설비에서 자동 저장 된 csv 데이터
 - 1개 : 결과 요약 csv 데이터
- 수집 장소 : 자동차 부품 제조사의 CNC 가공 공정
- 데이터 개수 : 32,048개 (양품: 22,645개, 불량: 9,403개)
- 데이터 용량 : 14.62 MB
- Features : CNC 가공 시 설비에서 자동 저장 되는 데이터 및 설정 조건 등 56가지

Attributes name	Description	Type	Unit
X_ActualPosition	부품의 실제 x 위치	float64	mm
X_ActualVelocity	부품의 실제 x 속도	float64	mm/s
X_ActualAcceleration	부품의 실제 x 가속도	float64	mm/s/s
X_SetPosition	부품의 참조 x 위치	float64	mm
X_SetVelocity	부품의 참조 x 속도	float64	mm/s
X_SetAcceleration	부품의 참조 x 가속도	float64	mm/s/s
X_CurrentFeedback	x 전류	float64	A
X_DCBusVoltage	x 전압	float64	V
X_OutputCurrent	x 이웃풋 전류	float64	A
X_OutputVoltage	x 이웃풋 전압	float64	V
X_OutputPower	x 이웃풋 전원	float64	kw
Y_ActualPosition	부품의 실제 y 위치	float64	mm
Y_ActualVelocity	부품의 실제 y 속도	float64	mm/s
Y_ActualAcceleration	부품의 실제 y 가속도	float64	mm/s/s
Y_SetPosition	부품의 참조 y 위치	float64	mm
Y_SetVelocity	부품의 참조 y 속도	float64	mm/s
Y_SetAcceleration	부품의 참조 y 가속도	float64	mm/s/s
Y_CurrentFeedback	y 전류	float64	A
Y_DCBusVoltage	y 전압	float64	V
Y_OutputCurrent	y 이웃풋 전류	float64	A
Y_OutputVoltage	y 이웃풋 전압	float64	V
Y_OutputPower	y 이웃풋 전원	float64	kw
Z_ActualPosition	부품의 실제 z 위치	float64	mm
Z_ActualVelocity	부품의 실제 z 속도	float64	mm/s
Z_ActualAcceleration	부품의 실제 z 가속도	float64	mm/s/s
Z_SetPosition	부품의 참조 z 위치	float64	mm
Z_SetVelocity	부품의 참조 z 속도	float64	mm/s

Attributes name	Description	Type	Unit
Z_SetAcceleration	부품의 참조 z 가속도	float64	mm/s/s
Z_CurrentFeedback	z 전류	float64	A
Z_DCBusVoltage	z 전압	float64	V
Z_OutputCurrent	z 이웃풋 전류	float64	A
Z_OutputVoltage	z 이웃풋 전압	float64	V
Z_OutputPower	z 이웃풋 전원	float64	kw
S_ActualPosition	스핀들의 실제 위치	float64	mm
S_ActualVelocity	스핀들의 실제 속도	float64	mm/s
S_ActualAcceleration	스핀들의 실제 가속도	float64	mm/s/s
S_SetPosition	스핀들의 설정 위치	float64	mm
S_SetVelocity	스핀들의 설정 속도	float64	mm/s
S_SetAcceleration	스핀들의 설정 가속도	float64	mm/s/s
S_CurrentFeedback	스핀들 전류	float64	A
S_DCBusVoltage	스핀들 전압	float64	V
S_OutputCurrent	스핀들 이웃풋 전류	float64	A
S_OutputVoltage	스핀들 이웃풋 전압	float64	V
S_OutputPower	스핀들 이웃풋 전원	float64	kw
S_SystemInertia	토크 관성	float64	kg*m^2
M_CURRENT_PROGRAM_NUMBER	프로그램이 CNC에 나열된 번호	float64	
M_sequence_number	실행 중인 G-code 라인	float64	
M_CURRENT_FEEDRATE	스핀들의 순간 공급 속도	float64	
Machining_Process	현재 수행 중인 가공 단계	float64	

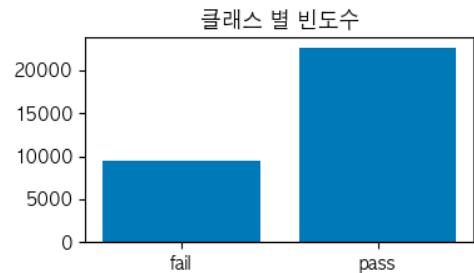
Input Data	Description
No	작업 번호
material	작업 소재
feed rate	Tool 이동 속도 (mm/s)
clamp pressure	소재 Clamping 압력 (bar)

Output Data	Description
tool condition	일정시간 사용한 Tool, 새로운 Tool
machine_completed	가공 완료 여부
passed_visual_inspection	육안 검사 결과

데이터 분석(EDA) 및 전처리

▣ 모든 csv 데이터 파일을 한개의 DataFrame으로 만들기 (concat_df)

```
concat_df.shape  
✓ 0.0s  
(32048, 56)  
  
concat_df.passed_visual_inspection.value_counts()  
✓ 0.0s  
yes    22645  
no     9403
```



- 26개 CSV 파일의 모든 데이터를 합친 DataFrame (concat_df) 의 클래스 빈도 균형 확인
→ Pass, Fail 의 비중이 약 7:3으로 불균형 하지만 학습에 문제 없음

▣ 학습에 불필요한 Feature 제거 및 2개 Feature 원핫인코딩 작업

```
# 불필요한 feature 제거 (모든 값이 같거나 의미 없는 데이터 제거)  
concat_df = concat_df.drop(['Z_CurrentFeedback', 'Z_DCBusVoltage', 'Z_OutputCurrent',  
                           'Z_OutputVoltage', 'No', 'material', 'df_len'], axis=1)  
  
# 이진 데이터 값을 1과 0으로 변환  
concat_df.passed_visual_inspection.replace({'yes':1, 'no':0}, inplace=True)  
concat_df.tool_condition.replace({'worn':1, 'unworn':0}, inplace=True)  
concat_df.machining_finalized.replace({'yes':1, 'no':0}, inplace=True)  
  
# 라벨 열 이름 바꾸기  
concat_df.rename(columns={'passed_visual_inspection' : 'target'}, inplace=True)  
  
# 원핫인코딩 : 'M_CURRENT_PROGRAM_NUMBER', 'Machining_Process'  
# 원핫인코딩 후 2개 열 삭제  
  
df = concat_df.copy()  
  
dum_program_num = pd.get_dummies(df.M_CURRENT_PROGRAM_NUMBER, prefix='p_num')  
dum_process_num = pd.get_dummies(df.Machining_Process, prefix='process')  
df = pd.concat([df, dum_process_num, dum_program_num], axis=1)  
df.drop(['M_CURRENT_PROGRAM_NUMBER', 'Machining_Process'], axis=1, inplace=True)  
print(df.shape)  
  
✓ 0.0s  
(32048, 60)
```

MagicPy

데이터 분석(EDA) 및 전처리

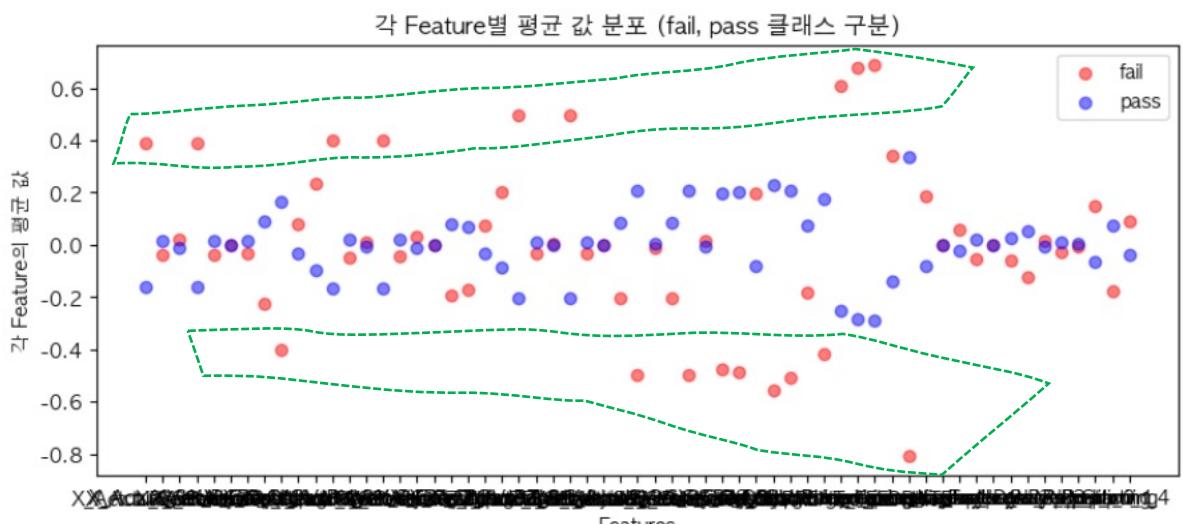
□ Target 클래스 별 Feature 평균 값 분포

```
# target 클래스별 feature 분포 확인
df_target_mean = df_scaled.groupby('target').mean().reset_index(drop=True)
plt.figure(figsize=(10,4))
plt.scatter(df_target_mean.columns, df_target_mean.values[0], alpha=0.5, color='red',
            df_target_mean.columns, df_target_mean.values[1], alpha=0.5, color='blue',
            plt.title('각 Feature별 평균 값 분포 (fail, pass 클래스 구분)')
            plt.xlabel('Features')
            plt.ylabel('각 Feature의 평균 값')
            plt.legend()
            plt.show()

✓ 0.1s
```

```
# feature별로 분포가 크기때문에 스케일링 작업
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)
```



- 일부 Feature에서 Target 별 평균 값에 차이가 나타남
→ 모델 학습이 가능

□ Target별 평균 값 차이가 큰 Feature 확인

```
# target 클래스별(pass, fail) 평균 값 차이가 큰 feature 확인

x1 = pd.Series(abs(df_target_mean.values[1] - df_target_mean.values[0]))
x2 = pd.Series(df_target_mean.columns)
x1_x2 = pd.concat([x1, x2], axis=1)
x1_x2.sort_values(by=0, ascending=False).head(10) # 상위 10개 feature

✓ 0.0s
```

	0	1
45	1.140655	machining_finalized
43	0.977073	clamp_pressure
42	0.961544	feedrate
41	0.862805	M_CURRENT_FEEDRATE
37	0.786793	S_OutputVoltage
38	0.717714	S_OutputPower
32	0.704193	S_SetVelocity
29	0.700479	S_ActualVelocity
25	0.700462	Z_SetPosition
22	0.700419	Z_ActualPosition

클래스별 평균 값의 차이가 큰 Top3 feature

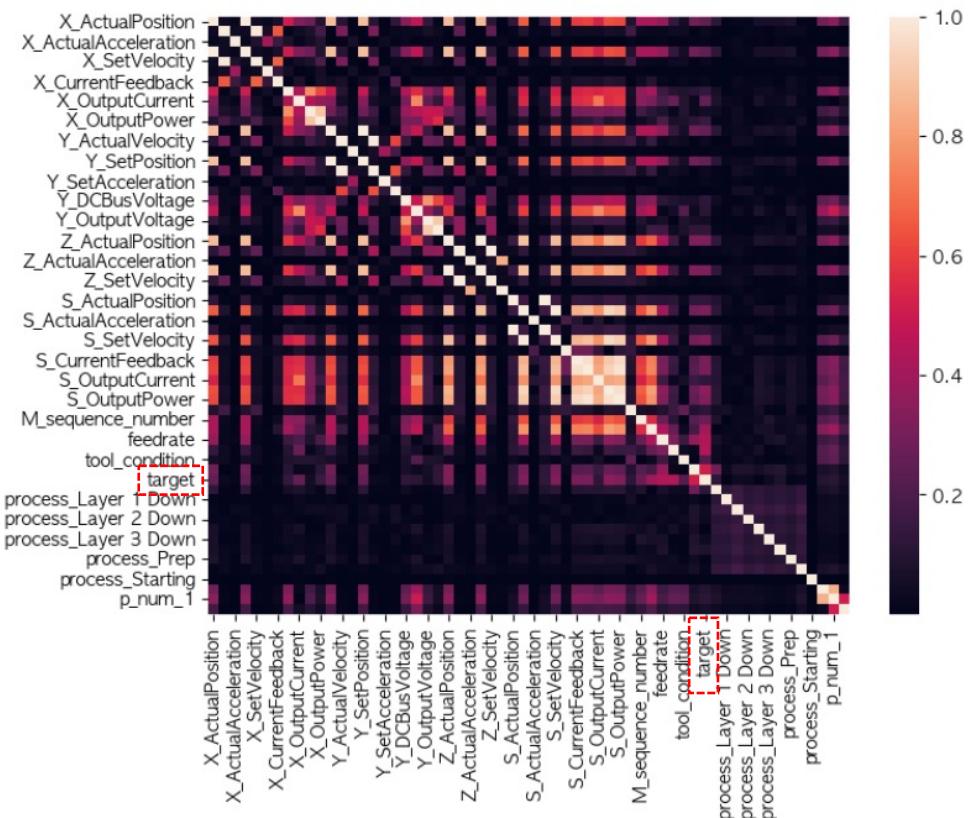
- machining_finalized : 공정 완료 여부
- clamp_pressure : 가공 소재를 Clamping 하는 압력
- feedrate : 가공 Tool의 이동 속도

데이터 분석(EDA) 및 전처리

□ Target 과의 상관계수 분포 확인

```
# 상관계수 히트맵
plt.figure(figsize=(8,6))
df_corr = abs(df.corr())
sns.heatmap(df_corr)
plt.show()

# target과 상관계수 높은 feature top10
print('[target과 상관계수 높은 feature top10]')
print(df_corr.target.sort_values(ascending=False).head(11).drop('target'))
```



	[target과 상관계수 높은 feature top10]
machining_finalized	0.519365
clamp_pressure	0.444883
feedrate	0.437812
M_CURRENT_FEEDRATE	0.392854
S_OutputVoltage	0.358244
S_OutputPower	0.326791
S_SetVelocity	0.320635
S_ActualVelocity	0.318943
Z_SetPosition	0.318936
Z_ActualPosition	0.318916

클래스별 평균 값의 차이가 큰 Top3 feature

- machining_finalized : 공정 완료 여부
- clamp_pressure : 가공 소재를 Clamping 하는 압력
- feedrate : 가공 Tool의 이동 속도

- Target과 상관계수가 높은 Feature 항목과 Target 클래스 별 평균값 차이가 큰 Feature 항목이 일치함
→ 평균 값의 차이가 큰 Feature가 클래스 결정에 영향을 준다.

데이터 분석(EDA) 및 전처리

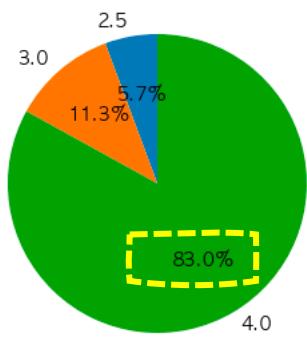
□ 클래스와 상관계수가 가장 높은 3가지 Feature 분석

1) machining_finalized : 공정 완료 여부

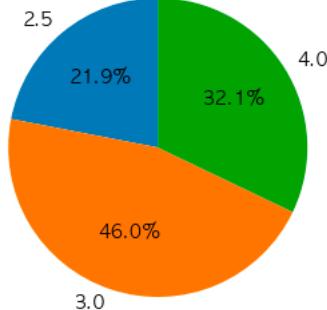
- 공정이 완료 되지 않은 샘플은 모두 Fail로 처리됨
→ 공정 완료 여부가 클래스 결과에 직접적으로 영향을 미침

2) clamp_pressure : 가공 소재를 Clamping 하는 압력

(클래스 0) clamp_pressure



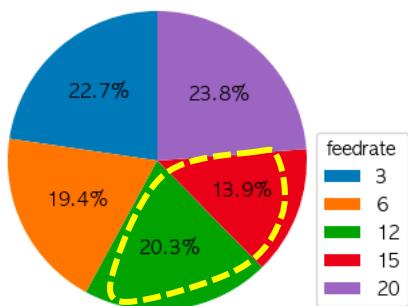
(클래스 1) clamp_pressure



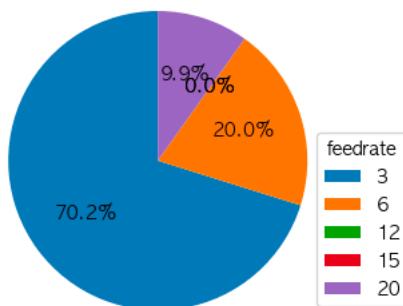
- 0 클래스의 clamp pressure 값 중 83%가 4.0
→ 클래스 별 압력 분포가 다르기 때문에 결과에 영향을 줄 수 있음

3) feedrate : 가공 Tool의 이동 속도

(클래스 0) feedrate



(클래스 1) feedrate



- Feedrate 값 중 0 클래스만 12, 15 값을 가짐
→ Feedrate 항목이 클래스 결과에 영향을 줄 수 있음

데이터 분석(EDA) 및 전처리

□ 데이터셋 분할(Train, Test), 라벨 열 나누기

```
# train, test 데이터셋 나누기 (target 비율을 고려해서)
train_df, test_df = train_test_split(df, test_size = 0.3, stratify=df['target'])
print('[데이터셋 shape]')
print(f'df : {df.shape}')
print(f'train_df : {train_df.shape} (70%)')
print(f'test_df : {test_df.shape} (30%)\\n')

# train, test 데이터셋 인덱스 초기화 (데이터셋 합치기 전 각각의 인덱스를 가지고 있었음)
train_df.reset_index(drop=True, inplace=True)
test_df.reset_index(drop=True, inplace=True)

# train_df, test_df target 빙도 확인
print(f'[train_df target 빙도] \\n{train_df.target.value_counts()}\\n')
print(f'[test_df target 빙도] \\n{test_df.target.value_counts()}\\n')

# 라벨 열 나누기
x_train = train_df.drop('target', axis=1)
y_train = train_df.target
x_test = test_df.drop('target', axis=1)
y_test = test_df.target
print('[데이터셋 shape]')
print(f'x_train shape : {x_train.shape}')
print(f'y_train shape : {y_train.shape}')
print(f'x_test shape : {x_test.shape}')
print(f'y_test shape : {y_test.shape}'')
```

| ✓ 0.0s

```
[데이터셋 shape]
df : (32048, 60)
train_df : (22433, 60) (70%)
test_df : (9615, 60) (30%)
```

```
[train_df target 빙도]
1    15851
0     6582
Name: target, dtype: int64
```

```
[test_df target 빙도]
1    6794
0     2821
Name: target, dtype: int64
```

```
[데이터셋 shape]
x_train shape : (22433, 59)
y_train shape : (22433,)
x_test shape : (9615, 59)
y_test shape : (9615,)
```

- test_size = 0.3 (전체 데이터 중 30%를 test 데이터 셋으로)
- stratify = df['target'] : train, test 데이터셋을 분리한 이후에
도 클래스 비율을 일정하게 유지함
- 학습 시킬 train 데이터의 feature는 59개

학습 모델 구축 및 예측

▣ 머신러닝 4가지 모델 학습 및 결과 예측 (XGBoost, LightGBM, CatBoost, RandomForest)

- Best Model : CatBoost, LightGBM (정확도 : 100%)

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

# 모델명 리스트 (출력시 모델명 텍스트로 사용)
model_list = []
for i in range(len(model)):
    model_list.append(type(model[i]).__name__)

# 캔버스 4개 만들기 (혼동행렬 / ROC Curve / Feat.Imp. 상위 10항목 / Feat.Imp. 전체 분포)
fig, axs = plt.subplots(1, len(model), figsize=(3*len(model), 2.5))
fig2, axs2 = plt.subplots(1, len(model), figsize=(3*len(model), 2.5))

# 캔버스 4개의 title
fig.suptitle('[각 모델의 혼동 행렬]', fontsize=14)
fig2.suptitle('[각 모델의 ROC Curve]', fontsize=14)

# 4개 ML 모델 학습 후 정확도, 혼동행렬, ROC, Feature Importance 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 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"{model_list[i]}:")
    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)}%")

    # 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')

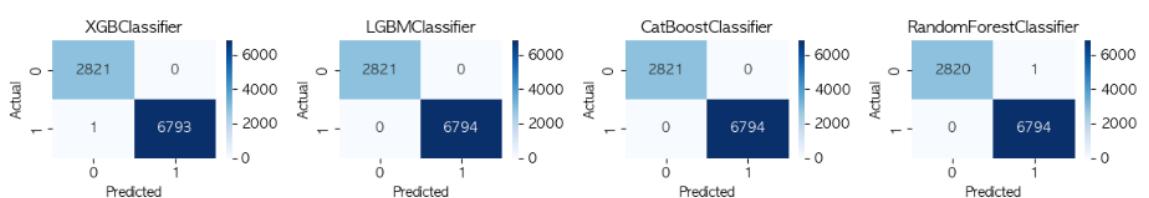
    # 각 모델의 ROC Curve 그리기
    fpr, tpr, thresholds = roc_curve(y_test, probs) # fpr, tp, thresholds 값 할당
    axs2[i].plot(fpr, tpr, color='red', label=f'AUC = {auc_score:.2f}') # ROC Curve 그리기
    axs2[i].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    axs2[i].set_title(model_list[i])
    axs2[i].set_xlabel('FPR')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

학습 모델 구축 및 예측

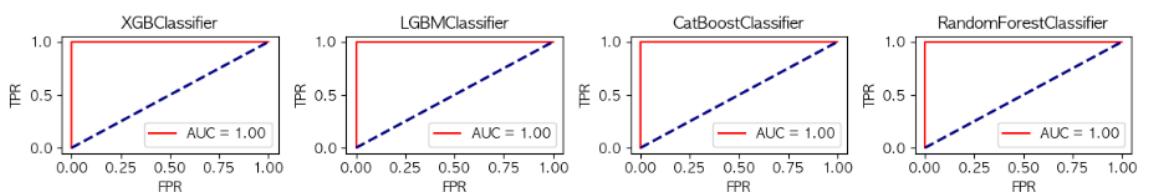
▣ 머신러닝 4가지 모델 학습 및 결과 예측 (XGBoost, LightGBM, CatBoost, RandomForest)

- Best Model : CatBoost, LightGBM (정확도 : 100%)

[모델별 혼동행렬]



[모델별 ROC Curve]



[모델별 평가 지표]

XGBClassifier:

- Accuracy: 99.99%
- Precision: 100.0%
- Recall: 99.99%
- F1 Score: 99.99%
- AUC: 100.0%

CatBoostClassifier:

- Accuracy: 100.0%
- Precision: 100.0%
- Recall: 100.0%
- F1 Score: 100.0%
- AUC: 100.0%

LGBMClassifier:

- Accuracy: 100.0%
- Precision: 100.0%
- Recall: 100.0%
- F1 Score: 100.0%
- AUC: 100.0%

RandomForestClassifier:

- Accuracy: 99.99%
- Precision: 99.99%
- Recall: 100.0%
- F1 Score: 99.99%
- AUC: 100.0%

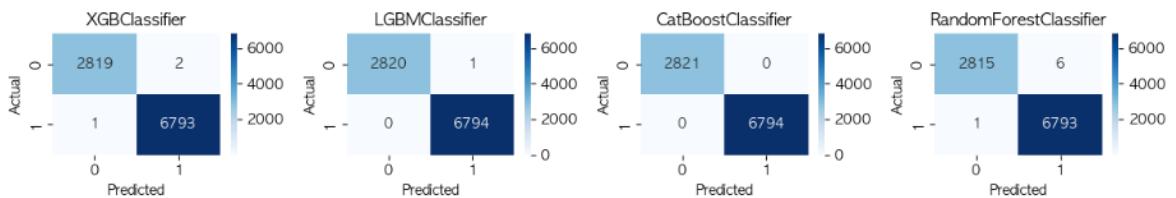
- CatBoost, LightGBM 모델로 학습 시 예측 정확도 100%
- 설비에서 자동저장 되는 데이터 또는 설비 설정 값 이외의 데이터는 삭제 후 학습 시켜볼 필요 있음 (아래 2개의 feature)
 - ✓ machining_finalized : 공정 완료 여부 - (관리자가 별도로 기록)
 - ✓ tool_condition : 일정시간 사용한 Tool인지 새로운 Tool 인지 – (관리자가 별도로 기록)

학습 모델 구축 및 예측

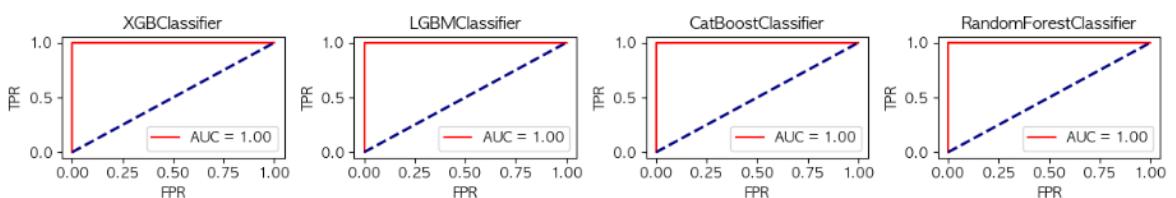
□ 수동으로 기록하는 2개 Feature 삭제 후 재학습 (tool_condition, machining_finalized)

- Best Model : CatBoost (정확도 : 100%)

[모델별 혼동행렬]



[모델별 ROC Curve]



[모델별 평가 지표]

XGBCClassifier:

- Accuracy: 99.97%
- Precision: 99.97%
- Recall: 99.99%
- F1 Score: 99.98%
- AUC: 100.0%

CatBoostClassifier:

- Accuracy: 100.0%
- Precision: 100.0%
- Recall: 100.0%
- F1 Score: 100.0%
- AUC: 100.0%

LGBMClassifier:

- Accuracy: 99.99%
- Precision: 99.99%
- Recall: 100.0%
- F1 Score: 99.99%
- AUC: 100.0%

RandomForestClassifier:

- Accuracy: 99.93%
- Precision: 99.91%
- Recall: 99.99%
- F1 Score: 99.95%
- AUC: 100.0%

- 수동으로 기록되는 Feature 2개 제거 후에도 CatBoost 모델은 예측 정확도 100% 기록
- 설비에서 자동 저장 되는 데이터만으로 학습 가능하다.

학습 모델 구축 및 예측

□ Logistic Regression 모델 학습 결과 - 성능 좋지 않음 (정확도 : 82.7%)

```
## logistic regression

# 로지스틱 회귀 모델 생성 및 학습
model = LogisticRegression(max_iter=20000) # 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)

# 평가지표 출력
print("Logistic Regression 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)}%")

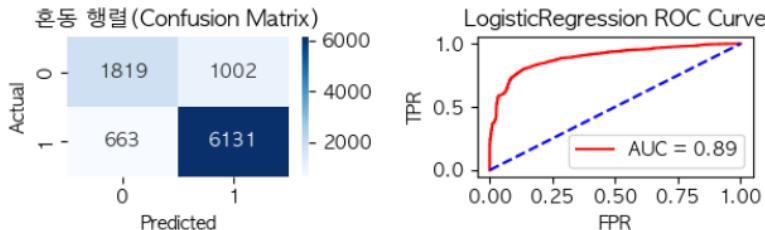
fig, axs = plt.subplots(1, 2, figsize=(6,2))
# 혼동 행렬 히트맵 그리기
confusion = confusion_matrix(y_test, y_pred)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", # 히트맵으로 시각화
            xticklabels=['0', '1'], yticklabels=['0', '1'], ax=axs[0])
axs[0].set_ylabel('Actual')
axs[0].set_xlabel('Predicted')
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('LogisticRegression ROC Curve')
axs[1].set_xlabel('FPR')
axs[1].set_ylabel('TPR')
axs[1].legend(loc="lower right")
fig.tight_layout()
plt.show()
```

✓ 17.7s

Logistic Regression metrics
- Accuracy: 82.68%
- Precision: 85.95%
- Recall: 90.24%
- F1 Score: 88.04%
- AUC: 89.37%

- 로지스틱 회귀 모델의 성능이 좋지 못한 원인 추정
→ 학습 데이터가 비선형 데이터이고, 클래스의
빈도가 불균형하다.



학습 모델 구축 및 예측

□ DNN (Deep Neural Network) 모델 학습 결과 – (예측 정확도 : 99.69%)

```
## DNN 학습 및 예측 결과

# 하이퍼파라미터 설정
epoch = 300 # 학습 반복 횟수
input_dim = x_train.shape[1] # 학습 시킬 feature 개수
act = 'LeakyReLU' # 활성화 함수
opt = 'adam' # optimizer
batch = 128 # batch size
drops = 0 # drop 비율
val_rate = 0.30 # 검증데이터 비율

# 모델 정의(구조)
def make_dnn_model():
    model = Sequential()

    # model.add(Dropout(drops)) # 첫번째 Dense layer 전에 drop을 시켜야 초반 과적합이 안남

    # 1st Dense Layer
    model.add(Dense(128, activation=act, input_dim=input_dim)) # 노드 수, 활성함수 설정, 입력 feature
    model.add(BatchNormalization()) # 배치정규화
    # model.add(Dropout(drops)) # 하이퍼파라미터에서 설정한 비율대로 drop

    # 2nd Dense Layer
    model.add(Dense(64, activation=act)) # 노드 수, 활성함수 설정
    model.add(BatchNormalization()) # 배치정규화
    # model.add(Dropout(drops)) # 하이퍼파라미터에서 설정한 비율대로 drop

    # 3rd Dense Layer
    model.add(Dense(64, activation=act)) # 노드 수, 활성함수 설정
    model.add(BatchNormalization()) # 배치정규화
    # model.add(Dropout(drops)) # 하이퍼파라미터에서 설정한 비율대로 drop

    # Output Dense Layer
    model.add(Dense(2, activation='softmax')) # 2개 클래스의 확률을 출력

    return model

dnn_model = make_dnn_model() # 모델 초기화

# 모델 컴파일
dnn_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')]

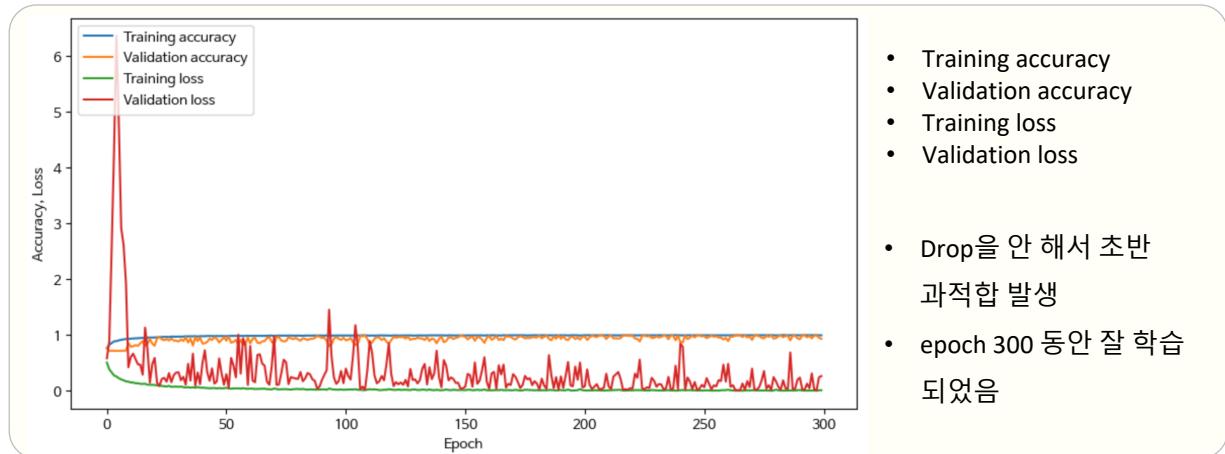
history = dnn_model.fit(x_train, y_train, # 학습데이터, 라벨
                        batch_size=batch, # batch size
                        validation_split=val_rate, # 검증용 데이터의 비율
                        epochs=epoch, # 학습 반복 횟수
                        callbacks=callbacks) # 검증 정확도가 올라갈때만 모델 저장
```

- 출력 및 평가지표 계산 코드는 생략

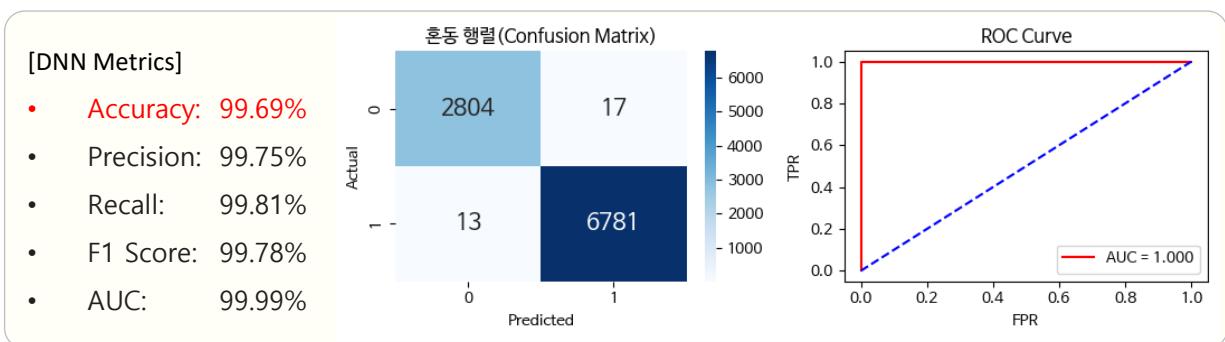
학습 모델 구축 및 예측

□ DNN (Deep Neural Network) 모델 학습 결과 – (예측 정확도 : 99.69%)

[Accuracy, Loss graph]



[평가지표, 혼동행렬, ROC Curve]



학습 모델 구축 및 예측

▣ CNN (Convolutional Neural Network) 모델 학습 결과 – (예측 정확도 : 99.74%)

```
## 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 = 200 # 학습 반복 횟수
input_dim = x_train.shape[1]
act = 'LeakyReLU' # 활성화 함수
opt = 'adam' # optimizer
filters = 32 # Conv1D Layer의 filter 수
batch = 128 # batch size
kernel = 3 # filter의 kernel size
drops = 0 # drop 비율
val_rate = 0.30 # 검증데이터 비율

# 모델 정의(구조)

def make_cnn_model():
    model = Sequential()
    # Conv1D Layer
    model.add(Conv1D(filters=filters, kernel_size=kernel, activation=act,
                     padding='same', input_shape=(input_dim, 1)))
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화
    model.add(MaxPooling1D(pool_size=2)) # MaxPooling1D Layer

    # Flatten Layer
    model.add(Flatten()) # filter에 의해 만들어진 다차원의 데이터를 1차원으로 변환
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화
    # model.add(Dropout(drops))

    # Dense Layer
    model.add(Dense(64, activation=act)) # Dense Layer 노드 수, 활성함수 설정
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화
    model.add(Dropout(drops)) # 과적합 방지를 위한 drop

    # Dense Layer
    model.add(Dense(32, activation=act)) # Dense Layer 노드 수, 활성함수 설정
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화
    model.add(Dropout(drops)) # 과적합 방지를 위한 drop

    # Output Layer
    model.add(Dense(2, activation='softmax')) # 2개 클래스의 확률을 출력

    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

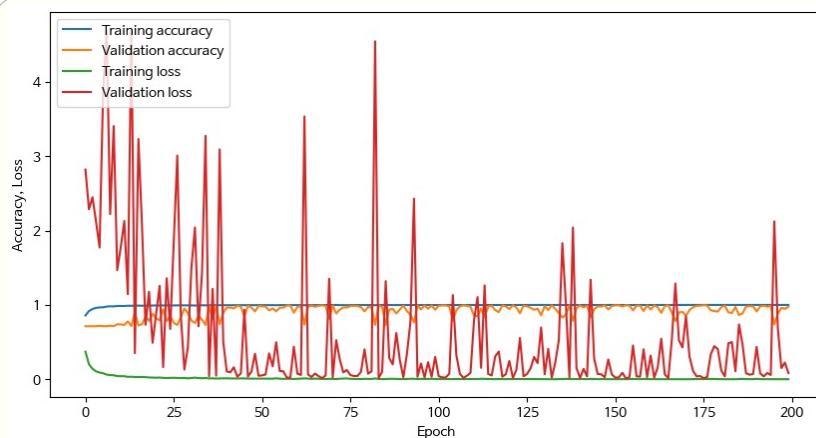
# monitor 지표를 기준으로 베스트 모델을 저장
callbacks = [ModelCheckpoint('best_model.h5', save_best_only=True,
                            monitor='val_sparse_categorical_accuracy', mode='max')]

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
```

학습 모델 구축 및 예측

▣ CNN (Convolutional Neural Network) 모델 학습 결과 – (예측 정확도 : 99.74%)

[Accuracy, Loss graph]

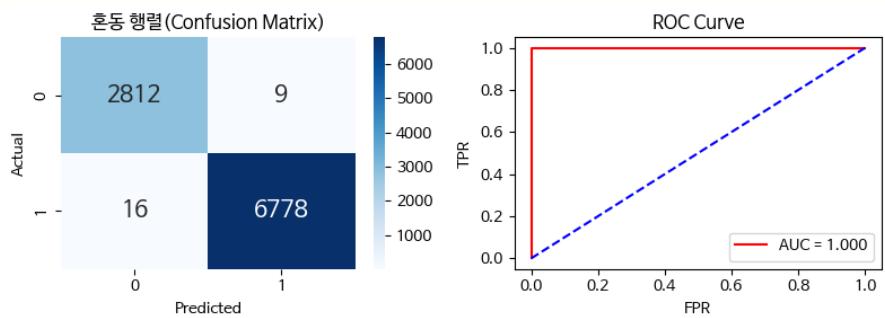


- Training accuracy
 - Validation accuracy
 - Training loss
 - Validation loss
-
- Drop을 안 해서 초반 과적합 발생
 - epoch 200 동안 잘 학습 되었음

[평가지표, 혼동행렬, ROC Curve]

[CNN Metrics]

- Accuracy: 99.74%
- Precision: 99.87%
- Recall: 99.76%
- F1 Score: 99.82%
- AUC: 99.99%



학습 완료

□ 학습 결과 요약은 2페이지 참조

