# **[A5] AutoML 2021313075 백경인**

**1. 설계한 AutoML pipeline 및 search space**

이번 프로젝트에서는 Wine Quality Dataset을 사용하여 회귀 문제를 해결하는 AutoML 파이프라인을 설계하였습니다. 목표는 다양한 머신러닝 모델을 사용하여 최적의 성능을 낼 수 있는 모델을 찾는 것이었으며, 이를 위해 수업시간에 배운 GridsearchCV가 아닌 Optuna를 사용하여 모델 하이퍼파라미터를 자동으로 최적화했습니다.  
  
파이프라인 구성:  
1. 데이터 전처리: 데이터는 StandardScaler를 사용하여 표준화되었습니다.  
2. 모델 선택: 최적의 모델을 찾기 위해 RandomForestRegressor, XGBRegressor, LGBMRegressor 사용  
3. 하이퍼파라미터 최적화: Optuna 라이브러리를 사용해 각 모델의 하이퍼파라미터를 자동으로 최적화  
4. 모델 평가: 교차 검증을 통해 각 모델의 성능을 평가하고, Test Set에서 최종 성능을 측정했습니다.  
  
Optuna Search Space:  
각 모델의 하이퍼파라미터 범위는 다음과 같습니다:  
- RandomForestRegressor: n\_estimators, max\_depth  
- XGBRegressor: n\_estimators, max\_depth, learning\_rate  
- LGBMRegressor: n\_estimators, num\_leaves, learning\_rate

**2. 선택한 데이터셋에 대한 최적의 모델 상세 정보 및 test set에서의 성능 평가 결과**

최적 모델: LightGBM  
하이퍼파라미터: {'model': 'LightGBM', 'lgb\_n\_estimators': 87, 'lgb\_num\_leaves': 55, 'lgb\_lr': 0.02020693021513137}  
  
Test Set 성능:  
Test RMSE: 0.6733  
Test R2 Score: 0.3118  
  
**3. 성능 개선 및 AutoML 효율성 개선 방안**

성능 개선 방법:  
1. 특성 선택 및 엔지니어링: 중요 특성을 선택하거나 새로운 특성을 생성하여 성능 개선  
2. 앙상블 기법 활용: 여러 모델을 결합하는 앙상블 기법 적용  
3. 하이퍼파라미터 튜닝 범위 확장: 더 넓은 범위에서 하이퍼파라미터 최적화  
4. 교차 검증 개선: 더 많은 폴드를 사용하여 성능을 더 안정적으로 평가  
  
AutoML 효율성 개선 방안:  
1. Optuna 효율성 향상: n\_trials 수 증가 및 초기 샘플링 개선  
2. 병렬 처리: Optuna의 병렬 처리 기능을 활용하여 최적화 속도 향상  
3. 자동화된 데이터 전처리: 결측값 처리와 특성 변환을 자동화하여 처리 시간 단축

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

from sklearn.pipeline import Pipeline

import optuna

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"

data = pd.read\_csv(url, sep=';')

X = data.drop('quality', axis=1)

y = data['quality']

# 500개 데이터를 Training & Validation 세트로, 나머지는 Test Set으로

X\_train\_val, X\_test, y\_train\_val, y\_test = train\_test\_split(X, y, train\_size=500, random\_state=42)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_val, y\_train\_val, test\_size=0.2, random\_state=42)

def objective(trial):

model\_name = trial.suggest\_categorical('model', ['RandomForest', 'XGBoost', 'LightGBM'])

pipeline\_steps = [('scaler', StandardScaler())] # 스케일링

if model\_name == 'RandomForest':

n\_estimators = trial.suggest\_int('rf\_n\_estimators', 1, 150, step=10)

max\_depth = trial.suggest\_int('rf\_max\_depth', 3, 10, step=1)

model = RandomForestRegressor(n\_estimators=n\_estimators, max\_depth=max\_depth, random\_state=42)

elif model\_name == 'XGBoost':

n\_estimators = trial.suggest\_int('xgb\_n\_estimators', 1, 150, step=10)

max\_depth = trial.suggest\_int('xgb\_max\_depth', 3, 10, step=1)

learning\_rate = trial.suggest\_loguniform('xgb\_lr', 0.01, 0.3)

model = XGBRegressor(n\_estimators=n\_estimators, max\_depth=max\_depth, learning\_rate=learning\_rate, random\_state=42)

else: # LightGBM

n\_estimators = trial.suggest\_int('lgb\_n\_estimators', 1, 150, step=1)

num\_leaves = trial.suggest\_int('lgb\_num\_leaves', 30, 100, step=5)

learning\_rate = trial.suggest\_loguniform('lgb\_lr', 0.01, 0.2)

model = LGBMRegressor(n\_estimators=n\_estimators, num\_leaves=num\_leaves, learning\_rate=learning\_rate, random\_state=42)

pipeline\_steps.append(('model', model))

pipeline = Pipeline(pipeline\_steps)

score = cross\_val\_score(pipeline, X\_train, y\_train, cv=3, scoring='neg\_mean\_squared\_error').mean()

return -score

study = optuna.create\_study(direction='minimize')

study.optimize(objective, n\_trials=30)

print("Best Trial:")

print(study.best\_trial.params)

best\_params = study.best\_trial.params

pipeline\_steps = [('scaler', StandardScaler())] # 스케일링

if best\_params['model'] == 'RandomForest':

model = RandomForestRegressor(

n\_estimators=best\_params['rf\_n\_estimators'],

max\_depth=best\_params['rf\_max\_depth'],

random\_state=42

)

elif best\_params['model'] == 'XGBoost':

model = XGBRegressor(

n\_estimators=best\_params['xgb\_n\_estimators'],

max\_depth=best\_params['xgb\_max\_depth'],

learning\_rate=best\_params['xgb\_lr'],

random\_state=42

)

else: # LightGBM

model = LGBMRegressor(

n\_estimators=best\_params['lgb\_n\_estimators'],

num\_leaves=best\_params['lgb\_num\_leaves'],

learning\_rate=best\_params['lgb\_lr'],

random\_state=42

)

pipeline\_steps.append(('model', model))

final\_pipeline = Pipeline(pipeline\_steps)

final\_pipeline.fit(X\_train, y\_train)

y\_pred\_val = final\_pipeline.predict(X\_val)

val\_rmse = mean\_squared\_error(y\_val, y\_pred\_val, squared=False)

print(f"Validation RMSE: {val\_rmse:.4f}")

y\_pred\_test = final\_pipeline.predict(X\_test)

test\_rmse = mean\_squared\_error(y\_test, y\_pred\_test, squared=False)

test\_r2 = r2\_score(y\_test, y\_pred\_test)

print(f"Test RMSE: {test\_rmse:.4f}")

print(f"Test R2 Score: {test\_r2:.4f}")

print("\n===== 최종 결과 =====")

print("최적 모델:", best\_params['model'])

print("하이퍼파라미터:", best\_params)

print(f"Test RMSE: {test\_rmse:.4f}")

print(f"Test R2 Score: {test\_r2:.4f}")