[A5] AutoML 2021313075 백경인

1. 설계한 AutoML pipeline 및 search space

이번 프로젝트에서는 Wine Quality Dataset 을 사용하여 회귀 문제를 해결하는 AutoML 파이프라인을 설계하였습니다. 목표는 다양한 머신러닝 모델을 사용하여 최적의 성능을 낼 수 있는 모델을 찾는 것이었으며, 이를 위해 수업시간에 배운 GridsearchCV 가 아닌 Optuna 를 사용하여 모델 하이퍼파라미터를 자동으로 최적화했습니다.

파이프라인 구성:
1. 데이터 전처리: 데이터는 StandardScaler 를 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 Set 에서 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 set 에서의 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: Optuna 의 병렬 처리 기능을 활용하여 최적화 속도 향상
3. 자동화된 데이터 전처리: 결측값 처리와 특성 변환을 자동화하여 처리 시간 단축

```python
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']

# 500 개 데이터를 Training & Validation 세트로, 나머지는 Test Set 으로
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}")
```