부스팅 회귀 모델 (Boosting Regression)

- 여러 개의 약한 학습기(weak learner, 보통 간단한 결정 트리)를 순차적으로 학습시켜, 이전 모델의 오차를 보완해나가면서 점차 강력한 모델을 만들어가는 앙상블 기법
- 랜덤 포레스트가 여러 트리를 독립적으로 병렬 학습하는 것과 달리, 부스팅은 이전 트리의 예측 결과와 실제값의 차이(잔차, residual)를 다음 트리가 학습하는 방식으로 진행
- 이 과정을 반복하면서 오차를 점차 줄여나가고, 최종적으로 모든 트리의 예측을 합산하여 최종 예측값을 생성

1. **그래디언트 부스팅 (Gradient Boosting Machine, GBM)**:
    - 부스팅의 가장 기본적인 형태로, 경사 하강법(Gradient Descent)을 사용하여 손실 함수(loss function)를 최소화하는 방향으로 모델을 학습시킵니다.
    - 각 단계의 트리는 이전까지의 모델이 만든 예측 오차(잔차)를 예측하도록 학습됩니다.
    - `scikit-learn`의 `GradientBoostingRegressor`가 대표적입니다.
2. **XGBoost (eXtreme Gradient Boosting)**:
    - 그래디언트 부스팅을 병렬 처리, 규제, 가지치기 등 다양한 측면에서 개선하여 속도와 성능을 극대화한 라이브러리입니다.
    - **장점**:
        - **규제(Regularization)**: L1, L2 규제 항을 포함하여 과적합 방지 성능이 뛰어납니다.
        - **병렬 처리**: 특성(feature) 단위로 병렬 처리가 가능하여 학습 속도가 빠릅니다.
        - **결측치 자체 처리**: 데이터에 결측치가 있어도 별도의 처리 없이 학습이 가능합니다.
        - **가지치기(Pruning)**: 일반 GBM과 달리, 트리를 끝까지 만든 후 역방향으로 불필요한 가지를 제거하는 가지치기를 수행하여 더 효율적입니다.
3. **LightGBM (Light Gradient Boosting Machine)**:
    - XGBoost와 유사한 고성능 라이브러리지만, 학습 속도에 더 초점을 맞춰 개발되었습니다.
    - **리프 중심 트리 분할 (Leaf-wise Tree Growth)**: 일반적인 트리(Level-wise)가 균형을 맞추며 성장하는 것과 달리, LightGBM은 손실을 가장 크게 줄일 수 있는 리프 노드를 집중적으로 분할하여 더 빠르고 효율적으로 최적의 트리를 만듭니다.
    - **장점**:
        - **매우 빠른 학습 속도**: 대용량 데이터셋에서 XGBoost보다 훨씬 빠른 학습 속도를 보입니다.
        - **적은 메모리 사용**: 효율적인 데이터 구조를 사용하여 메모리 사용량이 적습니다.
    - **단점**: 리프 중심 분할 방식 때문에 데이터의 양이 적을 경우(e.g., 10,000건 미만) 과적합되기 쉬운 경향이 있습니다.

적용 가능한 상황

- 예측 성능이 매우 중요한 캐글(Kaggle)과 같은 데이터 분석 대회에서 가장 널리 사용됩니다.
- 데이터의 비선형성, 특성 간 상호작용이 복잡할 때 뛰어난 성능을 보입니다.
- 대용량 데이터셋을 다룰 때 (특히 LightGBM).

구현 방법

- `scikit-learn`의 `GradientBoostingRegressor`

- 별도의 라이브러리인 `xgboost`의 `XGBRegressor`
- 별도의 라이브러리인 `lightgbm`의 `LGBMRegressor`

용도

- 높은 정확도가 요구되는 연속형 타겟 값 예측 문제.

주의사항

- **하이퍼파라미터 튜닝**: 부스팅 모델들은 성능에 영향을 미치는 하이퍼파라미터가 매우 많아, 최적의 성능을 내기 위해 튜닝이 거의 필수적입니다.
  - `n_estimators`: 생성할 트리의 개수. 너무 많으면 과적합될 수 있습니다.
  - `learning_rate`: 각 트리의 예측 결과를 얼마나 반영할지 결정하는 학습률. 낮을수록 훈련은 오래 걸리지만 더 안정적이고 좋은 성능을 낼 수 있습니다. (보통 `n_estimators`와 반비례 관계로 튜닝)
  - `max_depth`: 각 트리의 최대 깊이.
  - `subsample`, `colsample_bytree`: 과적합을 방지하기 위해 훈련 데이터나 특성을 샘플링하는 비율.
- **조기 종료 (Early Stopping)**: 검증 세트(validation set)의 성능이 일정 반복 횟수(e.g., 50회) 동안 더 이상 향상되지 않으면 학습을 조기에 중단하는 기능입니다. 불필요한 학습을 막고 과적합을 방지하는 데 매우 유용합니다.

```
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 1. 데이터 준비
housing = fetch_california_housing()
X, y = housing.data, housing.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 2. 그래디언트 부스팅 (Scikit-learn)
from sklearn.ensemble import GradientBoostingRegressor
gb_reg = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
max_depth=3, random_state=42)
gb_reg.fit(X_train, y_train)
gb_pred = gb_reg.predict(X_test)
print(f"Gradient Boosting MSE: {mean_squared_error(y_test, gb_pred):.3f}") # 0.294

# 3. XGBoost
# !pip install xgboost
import xgboost as xgb
xgb_reg = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3,
random_state=42, n_jobs=-1)
xgb_reg.fit(X_train, y_train)
xgb_pred = xgb_reg.predict(X_test)
print(f"XGBoost MSE: {mean_squared_error(y_test, xgb_pred):.3f}") # 0.295

# 4. LightGBM
# !pip install lightgbm
```

```
import lightgbm as lgb
lgbm_reg = lgb.LGBMRegressor(n_estimators=100, learning_rate=0.1, max_depth=3,
random_state=42, n_jobs=-1)
lgbm_reg.fit(X_train, y_train)
lgbm_pred = lgbm_reg.predict(X_test)
print(f"LightGBM MSE: {mean_squared_error(y_test, lgbm_pred):.3f}") # 0.289

# 5. 조기 종료(Early Stopping) 기능 활용 (LightGBM 예시)
# 검증 세트 준비
X_train_sub, X_val, y_train_sub, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

lgbm_reg_early = lgb.LGBMRegressor(n_estimators=1000, learning_rate=0.05,
random_state=42)
# early_stopping_rounds: 검증 성능이 향상되지 않아도 기다릴 반복 횟수
# eval_set: 성능 평가에 사용할 검증 세트
lgbm_reg_early.fit(X_train_sub, y_train_sub,
                   eval_set=[(X_val, y_val)],
                   eval_metric='rmse',
                   callbacks=[lgb.early_stopping(100, verbose=True)])

early_pred = lgbm_reg_early.predict(X_test)
print(f"\nLightGBM (Early Stopping) MSE: {mean_squared_error(y_test,
early_pred):.3f}") # 0.191
```

결과 해석 방법

- 부스팅 모델들은 랜덤 포레스트와 마찬가지로 `feature_importances_` 속성을 제공하여 각 특성의 중요도를 확인할 수 있습니다. 이를 통해 어떤 변수가 예측에 큰 영향을 미쳤는지 파악할 수 있습니다.
- 모델의 성능은 MSE, R^2 등 일반적인 회귀 평가지표로 평가합니다.
- 조기 종료를 사용했을 경우, 최적의 반복 횟수(`best_iteration_`)를 확인할 수 있습니다.

장단점 및 대안

- **장점**:
  - 현존하는 머신러닝 알고리즘 중 가장 뛰어난 예측 성능을 보이는 경우가 많습니다.
  - XGBoost와 LightGBM은 빠른 속도와 다양한 편의 기능(결측치 처리, 조기 종료 등)을 제공합니다.
- **단점**:
  - 튜닝해야 할 하이퍼파라미터가 많고, 잘못 튜닝하면 과적합되기 쉽습니다.
  - 랜덤 포레스트에 비해 훈련 시간이 더 오래 걸릴 수 있습니다.
  - 모델의 해석이 복잡합니다.
- **대안**:
  - **랜덤 포레스트**: 부스팅 모델보다 튜닝이 비교적 간단하고, 병렬 처리가 용이하여 훈련 속도가 더 빠를 수 있습니다.
  - **신경망 (Neural Networks)**: 정형 데이터보다 이미지, 텍스트 등 비정형 데이터에서 더 강점을 보이지만, 매우 복잡한 패턴 학습에 사용될 수 있습니다.
  - **스태킹 (Stacking)**: 부스팅 모델을 포함한 여러 다른 종류의 모델들을 결합하여 성능을 더욱 끌어올리는 앙상블 기법입니다.