부스팅 (Boosting)

- 여러 개의 약한 학습기(weak learner)를 순차적으로 학습시켜, 이전 모델의 실수를 보완해나가면서 점차 강력한 모델을 만들어가는 앙상블 기법
- 이전 모델의 학습 결과가 다음 모델의 학습에 영향을 미치는 순차적인 구조를 가짐
    - 배깅(Bagging)은 각 모델을 독립적으로 병렬 학습하는 것이 차이점
- 이 과정을 통해 모델의 편향(Bias)과 분산(Variance)을 모두 줄여나가는 것이 목표
- **핵심 아이디어**: 간단하고 약한 모델을 여러 개 연결하여, 이전 모델이 예측하기 어려워했던(잘못 예측한) 데이터에 집중하여 다음 모델을 학습시킴으로써 전체적인 성능을 점진적으로 향상시킵니다.

주요 부스팅 알고리즘 종류

1. **AdaBoost (Adaptive Boosting)**:
    - 부스팅의 초기 알고리즘 중 하나입니다.
    - **동작 방식**:
        1. 모든 훈련 데이터에 동일한 가중치를 부여하여 첫 번째 모델을 학습시킵니다.
        2. 첫 번째 모델이 잘못 분류한 데이터의 가중치를 높이고, 올바르게 분류한 데이터의 가중치는 낮춥니다.
        3. 업데이트된 가중치를 반영하여 두 번째 모델을 학습시킵니다. 이 모델은 가중치가 높은, 즉 이전 모델이 어려워했던 데이터에 더 집중하게 됩니다.
        4. 이 과정을 반복하고, 최종적으로 각 모델의 예측을 성능(오류율)에 따라 가중치를 부여하여 합산함으로써 최종 예측을 결정합니다.
2. **그래디언트 부스팅 (Gradient Boosting Machine, GBM)**:
    - AdaBoost처럼 데이터의 가중치를 업데이트하는 대신, 이전 모델의 예측값과 실제값의 차이인 **잔차(residual)**를 다음 모델이 학습하도록 하는 방식입니다.
    - 경사 하강법(Gradient Descent) 원리를 사용하여, 손실 함수(loss function)가 최소화되는 방향으로 잔차를 학습하는 모델을 순차적으로 추가해 나갑니다.
    - AdaBoost보다 더 유연하고 일반화된 형태로, 다양한 손실 함수를 적용할 수 있습니다.
3. **XGBoost, LightGBM**:
    - 그래디언트 부스팅을 기반으로 성능, 속도, 기능 면에서 크게 개선된 최신 라이브러리들입니다.
    - 규제(Regularization), 병렬 처리, 결측치 처리, 조기 종료 등 다양한 고급 기능을 통해 현재 정형 데이터 분석에서 가장 널리 사용되는 알고리즘입니다.

> **자세한 내용은 [부스팅 분류 모델] 및 [부스팅 회귀 모델] 문서를 참고하세요.**

적용 가능한 상황

- 분류/회귀 문제에서 최고의 예측 성능을 얻고 싶을 때.
- 모델의 편향과 분산을 모두 줄여 일반화 성능을 극대화하고 싶을 때.

구현 방법

`scikit-learn`의 `AdaBoostClassifier`, `GradientBoostingClassifier` 및 `xgboost`, `lightgbm` 라이브러리를 사용합니다.

코드 예시 (`AdaBoostClassifier`)

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# 1. 데이터 준비
cancer = load_breast_cancer()
X, y = cancer.data, cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

# 2. AdaBoost 분류기 생성
# base_estimator: 기본 학습기로 사용할 모델. None이면
DecisionTreeClassifier(max_depth=1)가 사용됨.
# n_estimators: 생성할 기본 학습기의 개수.
# learning_rate: 각 학습기의 기여도를 조절하는 학습률.
ada_clf = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=200,
    learning_rate=0.5,
    random_state=42
)

# 3. 모델 학습 및 평가
ada_clf.fit(X_train, y_train)
ada_pred = ada_clf.predict(X_test)
print(f"AdaBoost Classifier 정확도: {accuracy_score(y_test, ada_pred):.3f}") #
0.953
```

장단점 및 대안

- **장점**:
  - 배깅이나 단일 모델에 비해 일반적으로 더 높은 예측 성능을 보입니다.
  - 편향과 분산을 동시에 줄일 수 있습니다.
- **단점**:
  - 모델이 순차적으로 학습되므로 병렬 처리가 어려워 배깅보다 훈련 시간이 오래 걸릴 수 있습니다. (단, XGBoost, LightGBM은 내부적으로 병렬 처리를 지원하여 속도를 개선함)
  - 하이퍼파라미터에 민감하여 튜닝이 중요하고, 잘못 튜닝하면 과적합될 수 있습니다.
- **대안**:
  - **배깅 (Bagging)**: 훈련 속도가 더 중요하고, 병렬 처리가 용이한 환경일 때 좋은 대안입니다.
  - **스태킹 (Stacking)**: 부스팅 모델을 포함한 여러 모델을 결합하여 성능을 더욱 끌어올리는 방법입니다.