배깅 (Bagging)

- **B**ootstrap **Agg**regat**ing**의 약자로, 앙상블 학습의 대표적인 방법 중 하나
- 배깅은 같은 종류의 알고리즘(e.g. 결정 트리)을 여러 개의 다른 훈련 데이터 샘플에 대해 병렬로 학습시킨 후, 그 결과들을 종합하여 최종 예측을 결정하는 방식
- 배깅은 단일 모델(특히 결정 트리처럼 분산이 높은 모델)의 **분산(Variance)을 줄여** 모델을 안정시키고, 과적합을 방지하여 일반화 성능을 높이는 데 매우 효과적
- **부트스트랩 (Bootstrap)**
  - 원본 훈련 데이터셋에서 중복을 허용하여(복원 추출) 원본과 동일한 크기의 여러 데이터 샘플을 만드는 과정
  - 각 부트스트랩 샘플은 원본 데이터의 **약 63.2%** 정도의 고유한 데이터로 구성
- **병렬 학습**
  - 만들어진 각각의 부트스트랩 샘플에 대해 개별 모델(기본 학습기, base learner)을 독립적으로 학습
  - 모든 모델은 병렬적으로 학습될 수 있음
- **취합 (Aggregating)**
  - **분류 문제**: 학습된 모든 모델의 예측 결과를 다수결 투표(Hard Voting)를 통해 최종 클래스로 결정
  - **회귀 문제**: 학습된 모든 모델의 예측값의 평균을 최종 예측값으로 결정

랜덤 포레스트 (Random Forest)

랜덤 포레스트는 배깅의 대표적인 예시이자, 배깅에 추가적인 무작위성을 더한 확장된 모델입니다.

- **기본 원리**: 결정 트리를 기본 학습기로 사용하는 배깅 방식.
- **추가된 무작위성**: 일반적인 배깅은 각 부트스트랩 샘플에 대해 모든 특성을 사용하여 트리를 학습시키지만, 랜덤 포레스트는 각 트리의 노드를 분할할 때마다 **전체 특성 중 일부를 무작위로 선택**하고, 그 선택된 특성 내에서만 최적의 분할을 찾습니다.
- **효과**: 이 추가적인 무작위성 덕분에 앙상블에 포함된 개별 트리들이 서로 다른 모양을 갖게 되어(상관관계 감소), 모델의 일반화 성능이 더욱 향상됩니다.

> **자세한 내용은 [트리 기반 분류] 및 [트리 기반 회귀] 문서를 참고하세요.**

적용 가능한 상황

- 단일 모델(특히 결정 트리)이 훈련 데이터에 과적합되는 경향이 있을 때.
- 모델의 예측 안정성을 높이고 분산을 줄이고 싶을 때.
- 병렬 처리를 통해 대용량 데이터에 대한 학습 시간을 단축하고 싶을 때.

구현 방법

`scikit-learn`의 `ensemble` 모듈에 있는 `BaggingClassifier` 또는 `BaggingRegressor`를 사용하여 어떤 모델이든 배깅 앙상블로 만들 수 있습니다. `RandomForestClassifier`와 `RandomForestRegressor`는 결정 트리

를 기본 학습기로 사용하는, 이미 구현된 배깅의 특수 형태입니다.

코드 예시 (`BaggingClassifier`)

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
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. Bagging 분류기 생성
# base_estimator: 기본 학습기로 사용할 모델. None이면 DecisionTreeClassifier가 사용
됨.
# n_estimators: 생성할 기본 학습기의 개수.
# max_samples: 각 기본 학습기를 훈련할 때 사용할 샘플의 비율 또는 개수.
# max_features: 각 기본 학습기를 훈련할 때 사용할 특성의 비율 또는 개수.
# bootstrap: 샘플을 중복 추출할지 여부 (배깅의 핵심).
# n_jobs: 병렬 처리에 사용할 CPU 코어 수.
bagging_clf = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=5, random_state=42),
    n_estimators=100,
    max_samples=0.8, # 훈련 데이터의 80%만 사용
    max_features=0.8, # 특성의 80%만 사용
    random_state=42,
    n_jobs=-1
)

# 3. 모델 학습 및 평가
bagging_clf.fit(X_train, y_train)
bagging_pred = bagging_clf.predict(X_test)
print(f"Bagging Classifier 정확도: {accuracy_score(y_test, bagging_pred):.3f}") #
0.953

# 비교: 단일 결정 트리
single_dt = DecisionTreeClassifier(max_depth=5, random_state=42)
single_dt.fit(X_train, y_train)
single_pred = single_dt.predict(X_test)
print(f"Single Decision Tree 정확도: {accuracy_score(y_test, single_pred):.3f}") #
0.930
```

장단점 및 대안

- **장점**:
  - 모델의 분산을 효과적으로 줄여 과적합을 방지하고 일반화 성능을 높입니다.
  - 개별 모델들이 병렬적으로 학습되므로, 다중 코어 환경에서 효율적으로 학습할 수 있습니다.

- **단점**:
  - 모델의 해석이 어려워집니다.
  - 편향(Bias)이 높은 모델을 여러 개 앙상블해도 편향이 줄어들지는 않습니다. (주로 분산을 줄이는 데 효과적)
- **대안**:
  - **부스팅 (Boosting)**: 편향과 분산을 모두 줄이는 것을 목표로 하며, 일반적으로 배깅보다 더 높은 성능을 보입니다.
  - **보팅 (Voting)**: 서로 다른 종류의 모델을 결합하여 성능을 높이는 방법입니다.