스태킹 (Stacking)

- 여러 다른 종류의 모델(개별 모델, base model)의 예측 결과를 최종적인 메타 모델(meta model)의 학습 데이터로 사용하여 새로운 예측을 만드는, 정교한 앙상블 기법
- 보팅(Voting)이 각 모델의 예측을 단순 투표나 평균으로 합산하는 것과 달리, 스태킹은 **예측 자체를 학습**하는 2단계 구조를 보유

- **1단계 (Level 1)**:
    1. 훈련 데이터를 여러 개의 fold로 나눕니다 (교차 검증 방식).
    2. 각 fold에 대해, 여러 개별 모델(e.g., 로지스틱 회귀, KNN, SVM, 결정 트리 등)을 학습시킵니다.
    3. 학습된 모델들로 각 fold의 검증 데이터(out-of-fold)에 대한 예측값을 생성합니다.
    4. 이렇게 생성된 예측값들을 모아 메타 모델을 위한 새로운 학습 데이터(특성)를 만듭니다.
    5. 동시에, 1단계의 모델들을 전체 훈련 데이터로 다시 학습시켜, 테스트 데이터에 대한 예측값을 만들고 이를 메타 모델의 테스트 데이터로 사용합니다.
- **2단계 (Level 2)**:
    1. 1단계에서 생성된 예측값들을 특성(feature)으로 하는 새로운 학습 데이터를 사용하여 메타 모델(보통 로지스틱 회귀나 릿지 회귀 같은 간단한 모델)을 학습시킵니다.
    2. 학습된 메타 모델을 사용하여 새로운 데이터에 대한 최종 예측을 수행합니다.

- **핵심 아이디어**: 각 개별 모델이 만든 예측값들을 보고 어떤 모델의 예측을 얼마나 신뢰할지를 메타 모델이 스스로 학습하게 하는 것

적용 가능한 상황

- 예측 성능을 극한까지 끌어올리고 싶을 때 사용되는 고급 기법입니다.
- 캐글(Kaggle)과 같은 데이터 분석 대회에서 상위권 입상을 위해 자주 사용됩니다.
- 서로 다른 유형의 여러 모델(e.g., 선형 모델, 트리 모델, 신경망 등)의 장점을 모두 결합하고 싶을 때 효과적입니다.

구현 방법

`scikit-learn`의 `ensemble` 모듈에 있는 `StackingClassifier` 또는 `StackingRegressor`를 사용합니다.

주의사항

- **과적합 위험**
    - 스태킹은 구조가 복잡하여 과적합될 위험이 음
    - 특히 1단계에서 예측값을 생성할 때, 훈련에 사용된 데이터로 다시 예측을 만들면 데이터 누수(data leakage)가 발생하여 메타 모델이 과적합 발생
    - 이를 방지하기 위해 교차 검증을 통한 out-of-fold 예측값을 사용하는 것이 필수적
        - `scikit-learn`의 `StackingClassifier`는 이를 내부적으로 처리
- **모델의 다양성**

    - 보팅과 마찬가지로, 1단계에서 사용하는 개별 모델들은 가능한 한 서로 다른 유형의 모델을 사용하는 것이 성능 향상에 유리
- **메타 모델 선택**
    - 메타 모델은 보통 간단하고 해석이 용이한 선형 모델(e.g. `LogisticRegression`, `Ridge`)을 자주 사용
    - 너무 복잡한 모델을 메타 모델로 사용하면 과적합 위험이 커질 수 있음
- **계산 비용**
    - 여러 모델을 교차 검증 방식으로 여러 번 학습시켜야 하므로, 훈련 시간이 매우 오래 걸림

코드 예시 (`StackingClassifier`)

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 개별 모델 임포트
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# 앙상블 모델 임포트
from sklearn.ensemble import StackingClassifier

from sklearn.metrics import accuracy_score

# 1. 데이터 준비 및 전처리
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)

# 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 2. 개별 모델(Base Models) 정의
estimators = [
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
    ('knn', KNeighborsClassifier(n_neighbors=5)),
    ('svc', SVC(kernel='rbf', C=10, gamma=0.1, probability=True, random_state=42))
]

# 3. 메타 모델(Meta Model) 정의
# 메타 모델은 보통 간단한 선형 모델을 사용
meta_model = LogisticRegression(solver='liblinear', random_state=42)

# 4. 스태킹 분류기 생성
# estimators: (이름, 모델) 리스트 형태의 개별 모델들
```

```
# final_estimator: 메타 모델
# cv: 개별 모델들이 예측값을 생성할 때 사용할 교차 검증 폴드 수
stacking_clf = StackingClassifier(
    estimators=estimators,
    final_estimator=meta_model,
    cv=5
)

# 5. 모델 학습 및 평가
stacking_clf.fit(X_train_scaled, y_train)
stacking_pred = stacking_clf.predict(X_test_scaled)
print(f"Stacking Classifier 정확도: {accuracy_score(y_test, stacking_pred):.3f}")
# 0.965
```

결과 해석 방법

- 스태킹 모델은 여러 모델이 계층적으로 결합된 매우 복잡한 구조이므로, 내부 동작을 직접 해석하기는 거의 불가능합니다.
- 모델의 성능은 정확도, AUC 등 일반적인 분류 평가지표를 통해 평가합니다.
- `final_estimator_` 속성을 통해 학습된 메타 모델의 계수(`coef_`)를 확인할 수 있으며, 이를 통해 어떤 개별 모델의 예측 결과가 최종 예측에 더 큰 영향을 미쳤는지 간접적으로 유추해볼 수 있습니다.

장단점 및 대안

- **장점**:
  - 여러 모델의 장점을 극대화하여, 일반적으로 다른 어떤 단일 모델이나 앙상블 기법보다도 높은 성능을 낼 수 있는 잠재력을 가집니다.
- **단점**:
  - 구조가 매우 복잡하고 해석이 어렵습니다.
  - 여러 모델을 교차 검증으로 학습시켜야 하므로 훈련 시간이 매우 오래 걸립니다.
  - 과적합의 위험이 높아 세심한 설계와 검증이 필요합니다.
- **대안**:
  - **부스팅 (Boosting)**: 스태킹만큼 복잡하지 않으면서도 매우 높은 성능을 제공하므로, 대부분의 경우 스태킹보다 먼저 시도됩니다.
  - **보팅 (Voting)**: 더 간단한 방법으로 여러 모델을 결합하고 싶을 때 사용합니다.