보팅 (Voting)

- 여러 개의 다른 종류의 분류 모델(e.g. 로지스틱 회귀, SVM, 결정 트리)을 사용하여 동일한 데이터를 학습시킨 후,
  각 모델의 예측 결과를 투표를 통해 종합하여 최종 예측을 결정하는 앙상블 기법
- 서로 다른 모델들이 데이터의 다양한 측면을 학습할 것이라는 아이디어에 기반하며, 단일 모델보다 더 안정적이고 높은 성능을 기대 가능

1. **하드 보팅 (Hard Voting)**:
   - 다수결 원칙에 따라, 여러 모델이 예측한 클래스 중 가장 많이 예측된 클래스를 최종 결과로 선택
   - e.g. 모델 A, B, C가 각각 '클래스 1', '클래스 1', '클래스 2'로 예측했다면, 최종 예측은 '클래스 1'
   - 구현이 간단하고 직관적
2. **소프트 보팅 (Soft Voting)**:
   - 각 모델이 예측한 클래스별 **확률**을 평균 낸 후, 가장 높은 평균 확률을 가진 클래스를 최종 결과로 선택
   - 각 모델의 예측에 대한 '확신도(확률)'를 반영하므로, 일반적으로 하드 보팅보다 더 높은 성능
   - 모든 모델이 클래스 확률을 예측하는 기능(`predict_proba`)을 제공해야 함

적용 가능한 상황

- 단일 모델의 성능을 넘어, 여러 모델의 강점을 결합하여 예측 성능을 최대한으로 끌어올리고 싶을 때 사용합니다.
- 모델의 예측 안정성을 높이고 싶을 때 유용합니다. (서로 다른 모델의 오류를 상쇄하는 효과)
- 로지스틱 회귀, SVM, 결정 트리, KNN 등 서로 다른 관점에서 데이터를 학습하는 다양한 모델들을 결합할 때 효과적입니다.

구현 방법

`scikit-learn`의 `ensemble` 모듈에 있는 `VotingClassifier` 클래스를 사용합니다.

용도

- 여러 분류 모델의 예측을 종합하여 단일 모델보다 더 강건하고 정확한 예측을 만듭니다.

주의사항

- **모델의 다양성**
  - 보팅 앙상블의 효과를 극대화하려면, 사용하는 모델들이 가능한 한 서로 다른 방식으로 예측을 수행하는 것이 좋음
  - 비슷한 모델 여러 개를 사용하는 것보다, 로지스틱 회귀(선형), 결정 트리(비선형), KNN(거리 기반) 등 서로 다른 유형의 모델을 결합하는 것이 더 효과적
- **소프트 보팅과 확률 예측**
  - 소프트 보팅을 사용하려면 `VotingClassifier`에 포함된 모든 모델이 `predict_proba` 메서드를 지원해야 함
  - e.g. `SVC`의 경우 `probability=True`로 설정해야 함

- **성능 보장**
  - 보팅 앙상블이 항상 개별 모델보다 성능이 좋은 것은 아님
  - 성능이 매우 낮은 모델이 포함되면 전체 성능이 오히려 저하될 가능성 존재

```
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.tree import DecisionTreeClassifier

# 앙상블 모델 임포트
from sklearn.ensemble import VotingClassifier

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 (로지스틱 회귀, KNN, SVC에 필요)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 2. 개별 모델 정의
# SVC는 probability=True로 설정해야 soft voting 가능
lr_clf = LogisticRegression(solver='liblinear', random_state=42)
knn_clf = KNeighborsClassifier(n_neighbors=5)
svc_clf = SVC(kernel='rbf', C=10, gamma=0.1, probability=True, random_state=42)
dt_clf = DecisionTreeClassifier(max_depth=3, random_state=42)

# 3. 보팅 분류기 생성
# Hard Voting
# estimators: (모델 이름, 모델 객체)의 리스트
# voting: 'hard' 또는 'soft'
hard_voting_clf = VotingClassifier(
    estimators=[('lr', lr_clf), ('knn', knn_clf), ('svc', svc_clf), ('dt',
dt_clf)],
    voting='hard'
)

# Soft Voting
soft_voting_clf = VotingClassifier(
    estimators=[('lr', lr_clf), ('knn', knn_clf), ('svc', svc_clf), ('dt',
dt_clf)],
```

```
    voting='soft'
)

# 4. 모델 학습 및 평가
# VotingClassifier는 내부적으로 각 모델에 데이터를 전달하므로, 스케일링된 데이터를 사용
hard_voting_clf.fit(X_train_scaled, y_train)
hard_pred = hard_voting_clf.predict(X_test_scaled)
print(f"Hard Voting 정확도: {accuracy_score(y_test, hard_pred):.3f}") # 0.977

soft_voting_clf.fit(X_train_scaled, y_train)
soft_pred = soft_voting_clf.predict(X_test_scaled)
print(f"Soft Voting 정확도: {accuracy_score(y_test, soft_pred):.3f}") # 0.953

# 개별 모델 성능과 비교
lr_clf.fit(X_train_scaled, y_train)
lr_pred = lr_clf.predict(X_test_scaled)
print(f"\nLogistic Regression 정확도: {accuracy_score(y_test, lr_pred):.3f}") #
0.988

knn_clf.fit(X_train_scaled, y_train)
knn_pred = knn_clf.predict(X_test_scaled)
print(f"KNN 정확도: {accuracy_score(y_test, knn_pred):.3f}") # 0.959

svc_clf.fit(X_train_scaled, y_train)
svc_pred = svc_clf.predict(X_test_scaled)
print(f"SVC 정확도: {accuracy_score(y_test, svc_pred):.3f}") # 0.953

# 결정 트리는 스케일링이 필요 없지만, 비교를 위해 스케일링된 데이터로 학습
dt_clf.fit(X_train_scaled, y_train)
dt_pred = dt_clf.predict(X_test_scaled)
print(f"Decision Tree 정확도: {accuracy_score(y_test, dt_pred):.3f}") # 0.924
```

결과 해석 방법

- 보팅 분류기는 여러 모델의 조합이므로, 개별 모델처럼 계수나 특성 중요도를 통해 직접 해석하기는 어렵습니다.
- 모델의 성능은 정확도, 혼동 행렬, AUC 등 일반적인 분류 평가지표를 통해 평가합니다.
- 일반적으로 소프트 보팅이 하드 보팅보다 성능이 좋으며, 가장 성능이 좋은 단일 모델보다도 더 나은 결과를 보이는 경우가 많습니다.

장단점 및 대안

- **장점**:
  - 단일 모델보다 더 안정적이고 높은 일반화 성능을 제공합니다.
  - 서로 다른 모델들의 장점을 결합하여 약점을 보완할 수 있습니다.
- **단점**:
  - 여러 모델을 학습시켜야 하므로 훈련 시간이 더 오래 걸립니다.
  - 모델의 구조가 복잡해져 해석이 어렵습니다.
- **대안**:

- **배깅 (Bagging)**: 랜덤 포레스트가 대표적인 예로, 같은 종류의 모델을 여러 개 만들어 앙상블하는 기법입니다.
- **부스팅 (Boosting)**: 이전 모델의 오차를 보완하는 방식으로 순차적으로 모델을 학습시켜, 일반적으로 보팅이나 배깅보다 더 높은 성능을 냅니다.
- **스태킹 (Stacking)**: 여러 모델의 예측 결과를 다시 훈련 데이터로 사용하여 최종 예측을 수행하는 메타 모델(meta-model)을 만드는, 더 발전된 형태의 앙상블 기법입니다.