나이브 베이즈 분류 (Naive Bayes Classification)

- **베이즈 정리(Bayes' Theorem)**에 기반한 간단하면서도 강력한 확률적 분류 알고리즘
- 베이즈 정리는 데이터의 사전 확률(Prior)과 가능도(Likelihood)를 사용하여 특정 데이터가 어떤 클래스에 속할 사후 확률(Posterior)을 계산
- **베이즈 정리**: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$
    - $P(A|B)$: B가 주어졌을 때 A의 확률 (사후 확률)
    - $P(B|A)$: A가 주어졌을 때 B의 확률 (가능도)
    - $P(A)$: A의 확률 (사전 확률)
    - $P(B)$: B의 확률

나이브 베이즈 분류기는 이 정리를 사용하여, 주어진 특성(X)에 대해 각 클래스(y)의 사후 확률 $P(y|X)$를 계산하고, 이 확률이 가장 높은 클래스를 예측값으로 선택합니다.

- **'나이브(Naive)'한 가정**: 나이브 베이즈의 가장 큰 특징은 모든 특성(feature)들이 서로 **조건부 독립(conditionally independent)**이라고 가정하는 것입니다. 즉, "어떤 클래스에 속한다"는 조건 하에서는, 한 특성의 값이 다른 특성의 값에 전혀 영향을 주지 않는다고 가정합니다. 예를 들어, '날씨'와 '요일'이라는 특성이 '테니스를 친다'는 클래스에 독립적으로 영향을 미친다고 보는 것입니다. 이 가정은 현실에서는 거의 성립하지 않지만, 이러한 단순화 덕분에 모델이 매우 빠르고 효율적으로 작동하며, 많은 경우에 준수한 성능을 보입니다.

`scikit-learn`에서는 특성의 데이터 유형에 따라 여러 종류의 나이브 베이즈 분류기를 제공합니다.

- **가우시안 나이브 베이즈 (GaussianNB)**: 특성들이 연속적인 값이며 정규분포(가우시안 분포)를 따른다고 가정합니다.
- **다항 나이브 베이즈 (MultinomialNB)**: 특성들이 이산적인 값(e.g., 단어의 출현 횟수)일 때 사용됩니다. 주로 텍스트 분류에 많이 쓰입니다.
- **베르누이 나이브 베이즈 (BernoulliNB)**: 특성들이 이진 값(0 또는 1, e.g., 특정 단어의 유무)일 때 사용됩니다.

적용 가능한 상황

- 텍스트 분류(스팸 메일 필터링, 뉴스 기사 카테고리 분류 등)에서 매우 빠르고 효과적입니다.
- 모델 훈련 속도가 매우 중요하고, 빠른 베이스라인 모델이 필요할 때 유용합니다.
- 특성 간의 독립 가정이 어느 정도 만족되는 데이터셋에 적합합니다.

구현 방법

`scikit-learn`의 `naive_bayes` 모듈에 있는 `GaussianNB`, `MultinomialNB`, `BernoulliNB` 클래스를 사용합니다. 여기서는 연속형 특성을 다루는 `GaussianNB`를 중심으로 설명합니다.

용도

- 각 특성이 클래스 예측에 독립적으로 영향을 미친다고 가정하고, 베이즈 정리를 이용해 데이터의 클래스를 분류합니다.

주의사항

- **독립성 가정**: 나이브 베이즈의 성능은 특성들이 얼마나 독립적인지에 따라 크게 달라집니다. 만약 특성 간에 강한 상관관계가 존재하면 모델의 성능이 저하될 수 있습니다.
- **제로 빈도 문제 (Zero-frequency problem)**: `MultinomialNB`나 `BernoulliNB`에서, 훈련 데이터에 특정 단어가 한 번도 나타나지 않으면 해당 단어가 포함된 테스트 데이터의 사후 확률이 0이 되어버리는 문제가 있습니다. 이를 방지하기 위해 `alpha` 파라미터(라플라스 스무딩)를 사용하여 모든 단어가 최소한의 확률을 갖도록 보정합니다. `GaussianNB`는 연속형 데이터를 다루므로 이 문제에서 비교적 자유롭습니다.
- **특성 스케일링**: `GaussianNB`는 각 특성의 평균과 분산을 계산하여 사용하므로, 스케일링이 성능에 큰 영향을 주지 않습니다.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

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

# 2. 가우시안 나이브 베이즈 모델 학습
# GaussianNB 주요 하이퍼파라미터
# priors: 각 클래스의 사전 확률. None이면 데이터에 따라 자동 계산.
# var_smoothing: 분산 계산 시 안정성을 위해 더해주는 작은 값. (기본값=1e-9)
gnb_clf = GaussianNB()
gnb_clf.fit(X_train, y_train)

# 3. 예측 및 평가
y_pred = gnb_clf.predict(X_test)
y_pred_proba = gnb_clf.predict_proba(X_test)

print("--- 가우시안 나이브 베이즈 모델 평가 ---")
print(f"정확도: {accuracy_score(y_test, y_pred):.3f}") # 0.911
print("\n혼동 행렬:\n", confusion_matrix(y_test, y_pred))
'''
 [[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]
'''
print("\n분류 리포트:\n", classification_report(y_test, y_pred))
'''
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.82      0.93      0.87        15
           2       0.92      0.80      0.86        15
```

```
    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45
'''

# 4. 학습된 파라미터 확인
print("\n클래스별 사전 확률 (priors):", gnb_clf.class_prior_) # [0.33333333
0.33333333 0.33333333]
print("클래스별 특성의 평균 (theta):", gnb_clf.theta_)
'''
[[4.98857143 3.42571429 1.48571429 0.24      ]
 [5.94857143 2.73142857 4.23714286 1.30857143]
 [6.68285714 3.00857143 5.63142857 2.06857143]]
'''
print("클래스별 특성의 분산 (sigma):", gnb_clf.sigma_)
'''
[[0.10329796 0.17391021 0.02293878 0.00925715]
 [0.24078368 0.08558368 0.21147755 0.03564082]
 [0.42484898 0.11735511 0.32272653 0.06386939]]
'''
```

결과 해석 방법

- `class_prior_`: 각 클래스의 사전 확률(데이터에서의 비율)을 보여줍니다.
- `theta_`: 각 클래스별로 각 특성의 평균값을 보여줍니다.
- `sigma_`: 각 클래스별로 각 특성의 분산값을 보여줍니다.
- 모델은 이 파라미터들을 사용하여 새로운 데이터가 주어졌을 때 각 클래스에 속할 확률을 가우시안 분포를 통해 계산하고, 베이즈 정리를 적용하여 최종 클래스를 예측합니다.

장단점 및 대안

- **장점**:
  - 모델이 매우 단순하고, 훈련 속도가 매우 빠릅니다.
  - 적은 양의 훈련 데이터로도 좋은 성능을 내는 경우가 많습니다.
  - 고차원 데이터(특히 텍스트)에서 효과적입니다.
- **단점**:
  - 모든 특성이 독립적이라는 '나이브'한 가정이 현실 데이터에서는 잘 맞지 않아 성능에 한계가 있을 수 있습니다.
  - 연속형 특성이 정규분포를 따르지 않을 경우 `GaussianNB`의 성능이 저하될 수 있습니다.
- **대안**:
  - **로지스틱 회귀**: 나이브 베이즈와 같이 빠르고 해석이 용이하지만, 독립성 가정이 없어 더 유연할 수 있습니다.
  - **결정 트리 / 랜덤 포레스트**: 특성 간의 상호작용을 잘 포착하며, 데이터 분포에 대한 가정이 없습니다.
  - **서포트 벡터 머신 (SVC)**: 텍스트 분류 등 고차원 데이터에서 나이브 베이즈와 함께 좋은 성능을 보이는 경우가 많습니다.