로지스틱 회귀 (Logistic Regression)

- 지도 학습의 대표적인 선형 **분류(Classification)** 알고리즘
- 회귀가 아닌 분류에 사용되며, 각 데이터가 특정 클래스에 속할 확률을 예측
- 독립변수들의 선형 결합을 계산한 뒤, 이를 시그모이드(Sigmoid) 또는 소프트맥스(Softmax) 함수에 통과시켜 0과 1 사이의 확률값으로 변환
- **이진 분류 (Binary Classification)**: 클래스가 2개일 때, 시그모이드 함수를 사용하여 특정 클래스(e.g., Positive class)에 속할 확률을 계산합니다. 이 확률이 임계값(보통 0.5)보다 크면 해당 클래스로, 작으면 다른 클래스로 분류합니다.
- **다중 분류 (Multinomial Classification)**: 클래스가 3개 이상일 때, 소프트맥스 함수를 사용하여 각 클래스에 속할 확률을 계산하고, 가장 높은 확률을 가진 클래스로 분류합니다.

로지스틱 회귀는 내부적으로 로그 손실(Log Loss 또는 Cross-Entropy Loss)을 최소화하는 방향으로 학습됩니다.

적용 가능한 상황

- 예측하고자 하는 종속변수가 범주형(e.g., '예/아니오', 'A/B/C 타입')일 때 사용합니다.
- 모델의 예측 결과를 확률로 해석하고 싶을 때 유용합니다.
- 모델이 간단하고 계산 비용이 적어, 빠르고 해석 가능한 베이스라인 모델로 널리 사용됩니다.

구현 방법

`scikit-learn`의 `linear_model` 모듈에 있는 `LogisticRegression` 클래스를 사용합니다.

용도

- 특성(feature)들의 선형 조합을 통해 데이터가 어떤 클래스에 속할지 예측하고, 그 확률을 추정합니다.

주의사항

- **선형성 가정**: 독립변수와 로그-오즈(log-odds) 간의 선형 관계를 가정하므로, 복잡한 비선형 결정 경계를 만들기는 어렵습니다.
- **특성 스케일링**: 규제(Regularization)를 사용하는 경우, 특성 스케일링(e.g., `StandardScaler`)을 통해 모든 특성의 단위를 맞춰주는 것이 성능에 중요합니다.
- **다중공선성**: 특성들 간의 강한 상관관계는 계수 추정을 불안정하게 만들 수 있습니다.
- **클래스 불균형**: 특정 클래스의 데이터가 너무 적으면 모델이 다수 클래스에 편향될 수 있습니다. 이 경우, `class_weight='balanced'` 옵션을 사용하거나, 오버샘플링(SMOTE) 등의 기법을 적용해야 합니다.

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

```
# 1. 데이터 준비 (붓꽃 데이터 - 다중 분류)
iris = load_iris()
X = iris.data
y = iris.target

# 2. 데이터 분할 및 스케일링
# 층화 샘플링(stratify=y)을 통해 훈련/테스트 세트의 클래스 비율을 동일하게 유지
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

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

# 3. 모델 학습
# LogisticRegression 주요 하이퍼파라미터
# penalty: 규제 종류 ('l1', 'l2', 'elasticnet', 'none'). (기본값='l2')
# C: 규제 강도의 역수. 작을수록 강한 규제. (기본값=1.0)
# solver: 최적화 알고리즘. ('liblinear'은 이진 분류만 지원, 'lbfgs', 'saga' 등은 다중
분류 지원)
# multi_class: 다중 분류 방식 ('ovr'는 일대다, 'multinomial'은 소프트맥스). (기본값
='auto')
# class_weight: 클래스 불균형 처리 ('balanced' 또는 dict).
log_reg = LogisticRegression(
    C=1.0,
    solver='lbfgs',
    multi_class='multinomial', # 소프트맥스 회귀 사용
    random_state=42
)
log_reg.fit(X_train_scaled, y_train)

# 4. 예측 및 평가
y_pred = log_reg.predict(X_test_scaled)
y_pred_proba = log_reg.predict_proba(X_test_scaled)

print("--- 로지스틱 회귀 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
'''

# 예측 확률 확인 (첫 5개 샘플)
print("\n예측 확률 (첫 5개 샘플):\n", pd.DataFrame(y_pred_proba[:5],
columns=iris.target_names))
'''
      setosa  versicolor  virginica
0  0.000033    0.101593   0.898374
1  0.014130    0.789644   0.196226
2  0.003589    0.620538   0.375873
3  0.021821    0.629399   0.348780
4  0.005156    0.384716   0.610128
'''

# 5. 학습된 계수 확인
# 다중 분류에서는 (클래스 개수, 특성 개수) 형태의 계수가 생성됨
print("\n절편 (intercept):", log_reg.intercept_) # [-0.3296466   1.78618973
-1.45654313]
print("회귀 계수 (coefficients) shape:", log_reg.coef_.shape) # (3, 4)
```

결과 해석 방법

- `coef_`
  - 각 특성이 각 클래스의 로그-오즈(log-odds)에 미치는 영향을 표시
  - 이진 분류에서는 (1, n_features) 형태, 다중 분류에서는 (n_classes, n_features) 형태로 표현
  - 계수의 부호와 크기를 통해 특정 특성이 어떤 클래스로 분류될 확률을 높이는지/낮추는지 알 수 있습니다.
- `intercept_`: 절편 값
- `predict_proba()`
  - 각 데이터가 각각의 클래스에 속할 확률을 반환
  - 이 확률값을 통해 모델이 얼마나 확신을 가지고 예측했는지 알 수 있으며, 분류 임계값을 조정하는 데 사용할 수 있습니다.
- **평가 지표**:
  - **정확도(Accuracy)**: 전체 예측 중 올바르게 예측한 비율.
  - **혼동 행렬(Confusion Matrix)**: 실제 클래스와 예측 클래스를 행렬 형태로 보여주어, 어떤 클래스에서 오류가 발생했는지 상세히 알 수 있습니다.
  - **정밀도(Precision), 재현율(Recall), F1-점수(F1-Score)**: `classification_report`를 통해 확인할 수 있으며, 특히 클래스가 불균형할 때 모델 성능을 더 정확하게 파악할 수 있습니다.

장단점 및 대안

- **장점**:
  - 학습 속도가 빠르고 구현이 간단합니다.
  - 모델이 선형적이어서 계수를 통해 결과를 해석하기 용이합니다.
  - 예측 확률을 제공하여 유연한 활용이 가능합니다.
- **단점**:
  - 선형 결정 경계만 만들 수 있어 복잡한 데이터 패턴을 학습하기 어렵습니다.
  - 데이터가 매우 많거나 차원이 높을 때 다른 모델에 비해 성능이 떨어질 수 있습니다.

- **대안**:
  - **서포트 벡터 머신 (SVC)**: 커널 트릭을 사용하여 비선형 결정 경계를 효과적으로 만들 수 있습니다.
  - **결정 트리 / 랜덤 포레스트**: 데이터의 비선형성과 상호작용을 잘 포착하며, 해석도 비교적 용이합니다.
  - **그래디언트 부스팅 (XGBoost, LightGBM)**: 일반적으로 로지스틱 회귀보다 훨씬 높은 분류 성능을 보입니다.
  - **K-최근접 이웃 (KNN)**: 간단한 비선형 분류기로, 데이터의 지역적 구조를 기반으로 예측합니다.