로지스틱 회귀 (Logistic Regression)

- 종속변수가 범주형 데이터일 때 사용하는 대표적인 '분류(Classification)' 알고리즘
- 독립변수의 선형 결합을 계산하고, 이를 '로지스틱 함수(또는 시그모이드 함수)'에 적용하여 사건이 발생할 확률(0과 1 사이의 값)을 예측
- 이 확률값을 기준으로 특정 임계값(보통 0.5)보다 크면 클래스 1로, 작으면 클래스 0으로 분류
- **로지스틱 함수 (시그모이드 함수)**: $\sigma(z) = \frac{1}{1 + e^{-z}}$
    - z는 독립변수의 선형 결합($\beta_0 + \beta_1x_1 + ...$)입니다.
    - 이 함수는 입력값 z를 항상 0과 1 사이의 값으로 변환하여 확률로 해석할 수 있게 해줍니다.

로지스틱 회귀 종류

- 종속변수의 클래스 개수에 따라 다음과 같이 나뉩니다.

1. **이진 로지스틱 회귀 (Binary Logistic Regression)**
    - 종속변수의 클래스가 2개인 경우
    - e.g. 합격/불합격, 생존/사망, 정상/불량
2. **다중 로지스틱 회귀 (Multinomial Logistic Regression)**
    - 종속변수의 클래스가 3개 이상이고 순서가 없는 경우
    - e.g. 품종 A/B/C, 선호하는 스포츠 농구/축구/야구
    - 소프트맥스(Softmax) 함수를 사용하여 각 클래스에 속할 확률을 계산
3. **순서형 로지스틱 회귀 (Ordinal Logistic Regression)**
    - 종속변수의 클래스가 3개 이상이고 순서가 있는 경우
    - e.g. 고객 만족도 상/중/하, 신용 등급 1/2/3등급

적용 가능한 상황

- 종속변수가 범주형인 분류 문제를 해결하고자 할 때 사용됩니다.
- 각 독립변수가 종속변수(의 특정 클래스)에 미치는 영향력(오즈비, Odds Ratio)을 통계적으로 해석하고 설명해야 할 때 매우 유용합니다.
- 모델이 간단하고 계산 비용이 적어, 빠르고 해석 가능한 기본 모델(Baseline Model)로 자주 활용됩니다.

구현 방법

`statsmodels`와 `scikit-learn` 라이브러리 모두에서 구현할 수 있습니다.

- `statsmodels`는 통계적 해석(계수의 유의성, 오즈비 등)에 강점
- `scikit-learn`은 머신러닝 파이프라인(데이터 분할, 교차 검증, 하이퍼파라미터 튜닝 등)에 통합하기 용이

용도

- 범주형 종속변수를 예측하고, 각 예측 클래스에 대한 확률을 추정
- 독립변수와 결과(로그-오즈) 간의 관계를 분석

주의사항

- **선형성 가정**: 일반 선형 회귀와 마찬가지로, 독립변수와 종속변수의 변환된 형태인 '로그-오즈(Log-odds)' 간에 선형 관계가 있다고 가정합니다.
- **다중공선성**: 독립변수 간 높은 상관관계는 계수 추정을 불안정하게 만들 수 있으므로 VIF 등을 통해 확인해야 합니다.
- **클래스 불균형**: 특정 클래스의 데이터가 너무 적으면 모델이 다수 클래스에 편향될 수 있습니다. 이 경우, 오버샘플링(SMOTE), 언더샘플링 또는 `class_weight` 옵션 조정을 고려해야 합니다.

1. 이진 로지스틱 회귀 (Binary Logistic Regression)

- `scikit-learn` **예시**:

```
import pandas as pd
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, roc_auc_score,
classification_report

# 1. 데이터 준비 (예시: 유방암 데이터)
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X = pd.DataFrame(cancer.data, columns=cancer.feature_names)
y = cancer.target # 0: 악성, 1: 양성

# 2. 데이터 분할 및 스케일링
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 하이퍼파라미터
# penalty: 규제 종류 ('l1', 'l2', 'elasticnet', 'none'). (기본값='l2')
# C: 규제 강도의 역수. 작을수록 강한 규제. (기본값=1.0)
# solver: 최적화에 사용할 알고리즘. 데이터셋 크기, 규제 종류에 따라 선택.
('liblinear', 'lbfgs', 'saga' 등) (기본값='lbfgs')
# class_weight: 클래스 불균형 처리를 위한 가중치 ('balanced' 또는 dict 형태). (기본값
=None)
log_reg = LogisticRegression(solver='liblinear', 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)[:, 1] # 양성 클래스(1)에 대한
확률

print("--- Scikit-learn 이진 로지스틱 회귀 평가 ---")
print(f"정확도: {accuracy_score(y_test, y_pred):.3f}")
print(f"AUC: {roc_auc_score(y_test, y_pred_proba):.3f}")
```

```
print("혼동 행렬:\n", confusion_matrix(y_test, y_pred))
print("분류 리포트:\n", classification_report(y_test, y_pred))
'''
정확도: 0.988
AUC: 0.998
혼동 행렬:
 [[ 63   1]
 [  1 106]]
분류 리포트:
              precision    recall  f1-score   support
           0       0.98      0.98      0.98        64
           1       0.99      0.99      0.99       107

    accuracy                           0.99       171
   macro avg       0.99      0.99      0.99       171
weighted avg       0.99      0.99      0.99       171
'''

# 회귀 계수(영향력) 확인
print("\n회귀 계수(coef):", log_reg.coef_)
'''
회귀 계수(coef): [[-0.48431991 -0.46467509 -0.4503601  -0.55685253 -0.15972054
0.65734006
                  -0.54263986 -0.6288608  -0.11581996  0.02977112 -0.88835779
0.36116556
                  -0.2111288  -0.95481577 -0.18663722  0.57519908  0.16879018
-0.25735014
                   0.31928871  0.34294713 -0.93863448 -1.21290675 -0.73206367
-0.97569074
                  -0.7585647   0.02832094 -0.79133303 -0.99061511 -0.86116755
-0.1606306 ]]
'''
```

- `statsmodels` **예시**:

```
import statsmodels.api as sm

X_train_const = sm.add_constant(X_train) # 상수항 추가
log_reg_sm = sm.Logit(y_train, X_train_const)
results = log_reg_sm.fit()
# 헤시안 역행렬 계산 시 에러 발생 가능
# log_reg_sm.fit(method='lbfgs', maxiter=1000, disp=1)을 통해 헤시안 직접역행렬 사용
을 피함
# 근본적인 해결 방법: VIF 확인 → 고VIF 변수 제거 / PCA / 규제(L1/L2) 적용
# 예측만 목적일 경우, scikit-learn으로 진행

print("\n--- Statsmodels 이진 로지스틱 회귀 결과 ---")
print(results.summary())

# 오즈비(Odds Ratio) 계산 및 해석
odds_ratios = pd.DataFrame({
    "Odds Ratio": np.exp(results.params),
```

```
    "p-value": results.pvalues
})
print("\n오즈비:\n", odds_ratios)
# 해석: 'mean radius'의 오즈비가 0.03이라면, 'mean radius'가 1단위 증가할 때마다
# 악성(0) 대비 양성(1)이 될 오즈(odds)가 0.03배가 된다(즉, 감소한다)는 의미.
```

2. 다중 로지스틱 회귀 (Multinomial Logistic Regression)

- `scikit-learn` **예시**:

```
# 1. 데이터 준비 (예시: 붓꽃 데이터)
from sklearn.datasets import load_iris
iris = load_iris()
X_multi = iris.data
y_multi = iris.target # 0, 1, 2 세 개의 클래스

X_multi_train, X_multi_test, y_multi_train, y_multi_test =
train_test_split(X_multi, y_multi, test_size=0.3, random_state=42,
stratify=y_multi)

# 2. 모델 학습
# multi_class='multinomial'로 설정하여 다중 로지스틱 회귀(소프트맥스 회귀) 사용
# solver는 'lbfgs', 'newton-cg', 'saga' 등 지원
# multi_class='multinomial'는 버전 호환 문제 발생하여 'ovr'로 회피 → 실습 환경 참고
후 수정 필요
# multi_log_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs',
random_state=42)
multi_log_reg = LogisticRegression(multi_class='ovr', solver='lbfgs',
random_state=42)
multi_log_reg.fit(X_multi_train, y_multi_train)

# 3. 예측 및 평가
y_multi_pred = multi_log_reg.predict(X_multi_test)
print("\n--- Scikit-learn 다중 로지스틱 회귀 평가 ---")
print(f"정확도: {accuracy_score(y_multi_test, y_multi_pred):.3f}")
print("분류 리포트:\n", classification_report(y_multi_test, y_multi_pred))
'''
정확도: 0.889
분류 리포트:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.86      0.80      0.83        15
           2       0.81      0.87      0.84        15

    accuracy                           0.89        45
   macro avg       0.89      0.89      0.89        45
weighted avg       0.89      0.89      0.89        45
'''

# 각 클래스에 대한 회귀 계수 확인 (클래스 개수만큼의 계수 세트가 생성됨)
```

```
print("\n회귀 계수 (클래스 0, 1, 2 순서):\n", multi_log_reg.coef_)
'''
회귀 계수 (클래스 0, 1, 2 순서):
 [[-0.52971067  0.77840242 -2.13169357 -0.91156277]
 [-0.22525105 -2.19620339  0.68152478 -1.2045097 ]
 [-0.07738728 -0.19895638  2.49940092  2.12506361]]
'''
```

결과 해석 방법

- **scikit-learn**: `coef_` 속성은 각 독립변수가 로그-오즈에 미치는 영향을 나타냅니다. 양수이면 해당 변수 값이 증가할수록 클래스 1의 확률이 높아지고, 음수이면 낮아집니다. `predict_proba()`로 각 클래스에 속할 확률을 직접 확인할 수 있습니다.
- **statsmodels**: `summary()`를 통해 각 계수의 p-value를 확인하여 통계적 유의성을 판단할 수 있습니다. `np.exp(results.params)`를 통해 오즈비(Odds Ratio)를 계산할 수 있으며, 이는 변수가 1단위 변할 때 오즈(성공확률/실패확률)가 몇 배 변하는지를 나타내어 직관적인 해석을 제공합니다.

장단점 및 대안

- **장점**:
  - 모델이 간단하고 학습 속도가 빠릅니다.
  - 계수와 오즈비를 통해 각 변수의 영향력을 직관적으로 해석하고 설명하기 용이합니다.
  - 확률적 예측을 제공하여 결과의 불확실성을 파악할 수 있습니다.
- **단점**:
  - 독립변수와 로그-오즈 간의 선형성을 가정하므로, 복잡한 비선형 관계를 잘 모델링하지 못합니다.
  - 성능이 다른 복잡한 분류 모델(e.g., SVM, RandomForest, Gradient Boosting)에 비해 낮을 수 있습니다.
- **대안**:
  - **서포트 벡터 머신 (SVM)**: 비선형 분류에 강점을 가지며, 특히 커널 SVM은 복잡한 결정 경계를 만들 수 있습니다.
  - **트리 기반 모델 (Decision Tree, Random Forest)**: 변수 간 상호작용과 비선형성을 잘 포착하며, 해석도 비교적 용이합니다.
  - **나이브 베이즈 (Naive Bayes)**: 변수들이 서로 독립이라는 강한 가정을 하지만, 텍스트 분류 등 특정 분야에서 좋은 성능을 보입니다.
  - **신경망 (Neural Networks)**: 매우 복잡한 비선형 관계를 학습할 수 있는 강력한 모델입니다.