다항 회귀 (Polynomial Regression)

- 독립변수와 종속변수 간의 관계가 선형이 아닐 때, 독립변수의 거듭제곱을 새로운 특성으로 추가하여 비선형 관계를 모델링하는 회귀 분석 기법
- 단순 선형 회귀 모델을 확장하여 데이터에 더 잘 맞는 다항식 곡선을 적합시키는 방법

적용 가능한 상황

- 독립변수와 종속변수 간의 관계가 2차, 3차 함수 등 비선형 패턴을 보일 때 사용합니다.
- 산점도를 그렸을 때 데이터가 뚜렷한 곡선 형태를 띨 경우, 선형 회귀보다 더 나은 예측 성능을 기대할 수 있습니다.
- 복잡한 비선형 관계를 비교적 간단한 모델로 표현하고 싶을 때 유용합니다.

구현 방법

다항 회귀는 `scikit-learn`의 `PolynomialFeatures`를 사용하여 구현하는 것이 일반적입니다.
`PolynomialFeatures`는 기존 특성을 다항 특성으로 변환해주는 역할을 하며, 이렇게 변환된 특성을 선형 회귀 모델에 입력하여 학습시킵니다. `LinearRegression`을 함께 사용하여 다항 회귀를 구현합니다.

용도

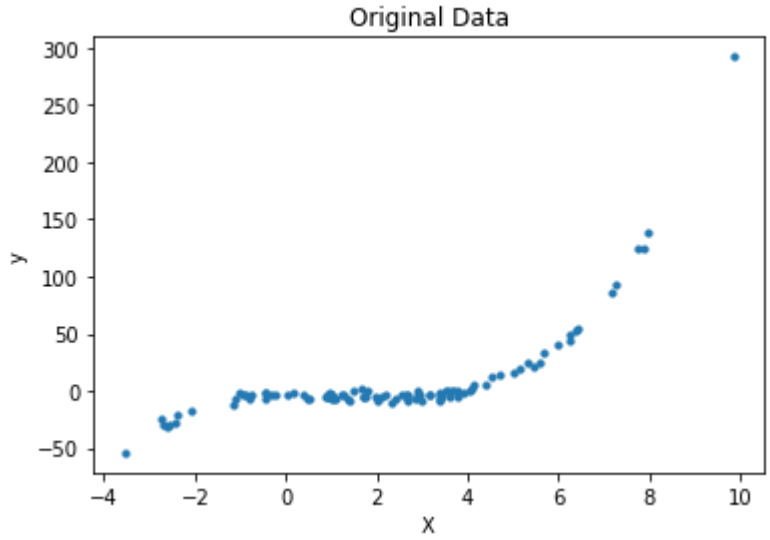
- 데이터의 비선형 관계를 학습하기 위해 독립변수의 다항 특성(e.g., $x^2, x^3, ...$) 및 특성 간의 상호작용 항(e.g., x_1x_2)을 생성합니다.

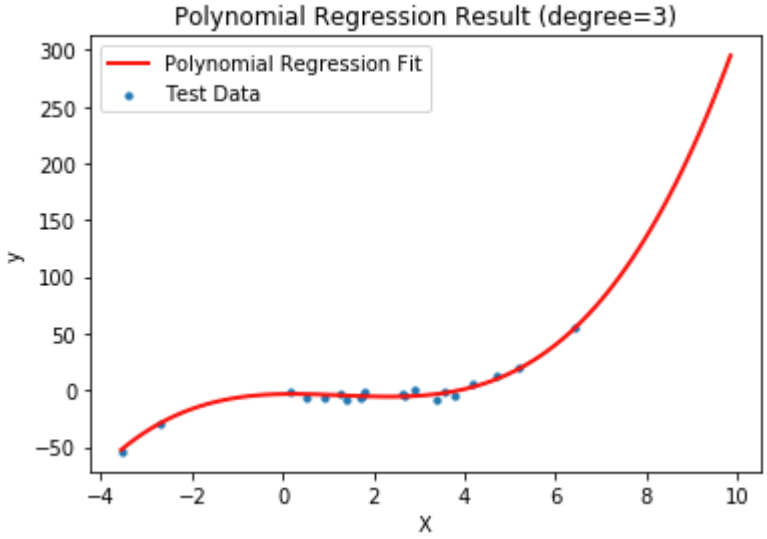
주의사항

- **과적합(Overfitting) 위험**: 다항식의 차수(degree)가 너무 높아지면 모델이 훈련 데이터에 과도하게 적합되어 새로운 데이터에 대한 예측 성능이 저하될 수 있습니다. 적절한 차수를 선택하는 것이 매우 중요합니다.
- **특성 스케일링**: 다항 특성은 값의 범위가 매우 커질 수 있으므로(e.g., x가 100이면 x^2은 10000), 모델의 안정성과 성능을 위해 스케일링(e.g., `StandardScaler`)을 적용하는 것이 좋습니다.
- **다중공선성**: 높은 차수의 다항 회귀에서는 x, x^2, x^3 등 파생 변수들 간에 강한 상관관계가 나타나 다중공선성 문제가 발생할 수 있습니다.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# 1. 데이터 생성 (비선형 관계)
np.random.seed(42)
X = 2 - 3 * np.random.normal(0, 1, 100)
y = X - 2 * (X ** 2) + 0.5 * (X ** 3) + np.random.normal(-3, 3, 100)
X = X[:, np.newaxis] # scikit-learn 입력을 위해 2D 배열로 변환
```

```
# 데이터 시각화
plt.scatter(X, y, s=10)
plt.title("Original Data")
plt.xlabel("X")
plt.ylabel("y")
plt.show()

# 2. 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 3. 다항 특성 생성
# PolynomialFeatures 하이퍼파라미터
# degree: 다항식의 차수 (기본값=2). 높을수록 복잡한 곡선을 만들지만 과적합 위험이 커집
니다.
# interaction_only: True이면 상호작용 항만 생성합니다 (e.g., x1*x2). 개별 변수의 거듭
제곱 항(x1^2, x2^2)은 생성하지 않습니다. (기본값=False)
# include_bias: True이면 편향을 위한 특성(값이 1인 열)을 추가합니다. (기본값=True)
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly_train = poly_features.fit_transform(X_train)
X_poly_test = poly_features.transform(X_test)

# 4. 스케일링 (선택 사항이지만 권장)
# 다항 특성 생성 후 스케일링을 적용합니다.
scaler = StandardScaler()
X_poly_train_scaled = scaler.fit_transform(X_poly_train)
X_poly_test_scaled = scaler.transform(X_poly_test)


# 5. 선형 회귀 모델 학습
lin_reg = LinearRegression()
lin_reg.fit(X_poly_train_scaled, y_train)

# 6. 예측 및 평가
y_pred = lin_reg.predict(X_poly_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print(f"다항 회귀 (3차) MSE: {mse:.2f}") # 7.72
print(f"회귀 계수: {lin_reg.coef_}")     # [  2.11390416 -34.77258549
76.38175949]
print(f"절편: {lin_reg.intercept_}")     # 9.911361945907691

# 7. 결과 시각화
X_new = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_new_poly = poly_features.transform(X_new)
X_new_poly_scaled = scaler.transform(X_new_poly)
y_new = lin_reg.predict(X_new_poly_scaled)

plt.scatter(X_test, y_test, s=10, label="Test Data")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Polynomial Regression Fit")
plt.title("Polynomial Regression Result (degree=3)")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```





결과 해석 방법

- **회귀 계수(`lin_reg.coef_`)**: 변환된 각 다항 특성($x, x^2, x^3, ...$)이 종속변수에 미치는 영향의 크기를 나타냅니다. 계수의 절대값이 클수록 해당 특성의 영향력이 크다고 해석할 수 있습니다.
- **평가 지표(MSE 등)**: 모델의 예측 오차를 나타냅니다. 차수를 바꿔가며 모델을 생성했을 때, 테스트 데이터에 대한 MSE가 가장 낮은 모델이 일반적으로 가장 좋은 모델입니다.
- **시각화**: 산점도와 회귀 곡선을 함께 그려 데이터의 패턴을 얼마나 잘 설명하는지 직관적으로 확인합니다. 훈련 데이터뿐만 아니라 테스트 데이터에서도 좋은 성능을 보이는지 확인하여 과적합 여부를 판단합니다.

장단점 및 대안

- **장점**:
  - 구현이 비교적 간단하면서도 복잡한 비선형 관계를 모델링할 수 있습니다.
  - 결과 해석이 다른 비선형 모델(e.g., 신경망)에 비해 직관적입니다.
- **단점**:
  - 적절한 차수를 사전에 알기 어려워 여러 차수를 시도하며 찾아야 합니다.
  - 차수가 높아지면 과적합 위험이 매우 커지고, 변수 간 다중공선성 문제가 발생할 수 있습니다.
  - 데이터의 범위를 벗어난 값을 예측할 때 성능이 급격히 나빠질 수 있습니다.

- **대안**:
  - **규제 모델 (Ridge, Lasso)**: 다항 회귀와 결합하여 회귀 계수의 크기를 제한함으로써 과적합을 완화할 수 있습니다.
  - **트리 기반 모델 (Decision Tree, Random Forest)**: 데이터의 비선형성과 상호작용을 잘 포착하며, 과적합 제어가 용이합니다.
  - **서포트 벡터 머신 (SVR)**: 커널 트릭을 사용하여 고차원 특성 공간에서 비선형 관계를 효율적으로 모델링합니다.
  - **일반화 가법 모델 (GAM)**: 각 특성에 대해 비선형 함수를 적용한 후 이를 합산하는 형태로, 다항 회귀보다 유연한 모델링이 가능합니다.