Polynomial Regression

A notebook covering the theory, mathematics, and Python implementation of polynomial regression.

1. Brief Idea

Polynomial regression extends linear regression by modeling the relationship between the independent variable and the dependent variable as an nth-degree polynomial. It captures non-linear trends in data while remaining a linear model in terms of parameters.

2. Context and Backstory

Polynomial regression has been used since the early days of statistics to model non-linear relationships. It is widely used in economics, engineering, and the natural sciences. However, high-degree polynomials can lead to overfitting, so regularization or model selection is important.

3. Mathematical Derivations and Formulae

The model is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \epsilon$$

The design matrix X includes columns for $x, x^2, \ldots, x^n$. The parameters β are estimated by minimizing the sum of squared errors, as in linear regression:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

4. Diagrams and Visual Aids

Figure: Polynomial regression fit to non-linear data.

(Insert a plot or image here if desired.)

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Generate data
np.random.seed(0)
x = np.linspace(0, 10, 100)
y = 0.5 * x**3 - 2 * x**2 + x + 3 + 20 * np.random.randn(100)

# Transform features
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(x.reshape(-1, 1))

# Fit model
model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)
```

```
# Plot
plt.scatter(x, y, label='Data', alpha=0.5)
plt.plot(x, y_pred, color='red', label='Polynomial Fit')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (degree 3)')
plt.legend()
plt.show()
```

6. Frequently Asked Questions (FAQ)

1. **When should I use polynomial regression?** When data shows a non-linear trend that can be captured by a polynomial.
2. **What are the risks of high-degree polynomials?** Overfitting and poor generalization.
3. **How to choose the degree?** Use cross-validation or domain knowledge.
4. **Is polynomial regression still a linear model?** Yes, it is linear in the parameters.

7. Higher-Order Thinking Questions (HOTS)

- Derive the normal equations for polynomial regression of degree n.
- Discuss the bias-variance tradeoff in polynomial regression.
- How would you regularize polynomial regression to prevent overfitting?

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate data
np.random.seed(0)
x = np.linspace(0, 10, 100)
y = 0.5 * x**3 - 2 * x**2 + x + 3 + 20 * np.random.randn(100)

# Build polynomial features (degree 3)
def poly_features(x, degree):
    X = np.ones((x.shape[0], degree+1))
    for d in range(1, degree+1):
        X[:, d] = x**d
    return X

X_poly = poly_features(x, 3)

# Closed-form solution for least squares
# beta = (X^T X)^(-1) X^T y
XtX = X_poly.T @ X_poly
Xty = X_poly.T @ y
beta = np.linalg.inv(XtX) @ Xty

y_pred = X_poly @ beta

# Plot
plt.scatter(x, y, label='Data', alpha=0.5)
plt.plot(x, y_pred, color='red', label='Polynomial Fit (from scratch)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (degree 3, from scratch)')
plt.legend()
plt.show()

print('Estimated coefficients:', beta)
```

In [ ]:

```python
import numpy as np
import matplotlib.pyplot as plt

# Generate data
np.random.seed(0)
x = np.linspace(0, 10, 100)
y = 0.5 * x**3 - 2 * x**2 + x + 3 + 20 * np.random.randn(100)

# Build polynomial features (degree 3)
def poly_features(x, degree):
    X = np.ones((x.shape[0], degree+1))
    for d in range(1, degree+1):
        X[:, d] = x**d
    return X

X_poly = poly_features(x, 3)

# Gradient Descent for least squares
beta = np.zeros(X_poly.shape[1])
alpha = 1e-6
n_iter = 10000
for i in range(n_iter):
    y_pred = X_poly @ beta
    grad = 2 * X_poly.T @ (y_pred - y) / len(y)
    beta -= alpha * grad
    if i % 2000 == 0:
        mse = np.mean((y_pred - y)**2)
        print(f"Iter {i}, MSE: {mse:.2f}")

# Final prediction
y_pred = X_poly @ beta

plt.scatter(x, y, label='Data', alpha=0.5)
plt.plot(x, y_pred, color='green', label='GD Fit (from scratch)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (degree 3, Gradient Descent)')
plt.legend()
plt.show()

print('Estimated coefficients (GD):', beta)
```

In [ ]:

```python
# Predict on new data using fitted coefficients (from scratch)
def predict_poly(x_new, beta):
    X_new = np.ones((x_new.shape[0], len(beta)))
    for d in range(1, len(beta)):
        X_new[:, d] = x_new**d
    return X_new @ beta

# Example: predict for x = [2, 4, 6, 8]
x_new = np.array([2, 4, 6, 8])
y_new_pred = predict_poly(x_new, beta)
print('Predictions for new x:', x_new)
print('Predicted y:', y_new_pred)
```

In [ ]:

```python
# Compute Mean Squared Error (MSE) from scratch
def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

# Example: MSE for training data (closed-form solution)
y_pred_train = predict_poly(x, beta)
mse_train = mean_squared_error(y, y_pred_train)
print('Training MSE (closed-form):', mse_train)
```

```
# Example: MSE for new data
mse_new = mean_squared_error(y_new_pred, y_new_pred)  # y_new_pred is just a demo; replace wi
print('MSE for new predictions (demo):', mse_new)
```