Linear Regression

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

1. Import Required Libraries

We use NumPy for numerical operations, matplotlib for visualization, and scikit-learn for model fitting and data generation.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error
```

2. Generate and Visualize Synthetic Data

We generate 1D synthetic regression data with noise and visualize it using a scatter plot.

```
# Generate synthetic data
X, y = make_regression(n_samples=200, n_features=1, noise=20, random_state=42)
plt.scatter(X, y, label="Data")
plt.xlabel("Feature")
plt.ylabel("Target")
plt.title("Synthetic Regression Data")
plt.legend()
plt.show()
```

3. Mathematical Derivation: Normal Equation

Given $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, linear regression models $y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$.

Define:

- $X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(N)})^T \end{bmatrix}, y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}$
- Minimize $J(\theta) = \|X\theta - y\|^2$
- The normal equation:

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

  Assumes $X^T X$ is invertible.

4. Fit Linear Regression Model

We fit a linear regression model to the synthetic data using scikit-learn.

```
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
```

5. Evaluate Model Performance (MSE)

We compute the mean squared error (MSE) between the predictions and true values.

```
mse = mean_squared_error(y, y_pred)
print("Mean Squared Error (MSE):", mse)
```

6. Plot Best-Fit Line

We plot the original data and overlay the fitted regression line.

```
plt.scatter(X, y, label="Data")
plt.plot(X, y_pred, color='red', label="Best-Fit Line")
plt.xlabel("Feature")
plt.ylabel("Target")
plt.title("Linear Regression Fit")
plt.legend()
plt.show()
```

7. Frequently Asked Questions (FAQ)

When is the closed-form solution not ideal?

- When the number of features d is large, inverting $X^T X$ is computationally expensive. Use gradient descent instead.

What about regularization?

- Ridge (L2) or Lasso (L1) penalize coefficients to prevent overfitting.

How to detect multicollinearity?

- Compute variance inflation factors (VIFs) for features.

What if noise is non-Gaussian?

- Use robust regression (e.g., RANSAC, Huber) to downweight outliers.

8. Higher-Order Thinking Questions (HOTS)

1. Derive the bias–variance decomposition of linear regression error.
2. How does adding polynomial features transform the normal equations?
3. Compare convergence of batch versus stochastic gradient descent on this problem.

In [ ]:

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

# Generate synthetic data
np.random.seed(42)
X = np.random.rand(200, 1) * 10
true_theta = np.array([5, 2])  # y = 5 + 2x
noise = np.random.randn(200) * 2
y = true_theta[0] + true_theta[1] * X[:, 0] + noise

# Add bias term
X_b = np.c_[np.ones((X.shape[0], 1)), X]
```

```python
# Closed-form solution: theta = (X^T X)^(-1) X^T y
XtX = X_b.T @ X_b
Xty = X_b.T @ y
theta_hat = np.linalg.inv(XtX) @ Xty
print('Estimated coefficients:', theta_hat)

# Predictions
y_pred = X_b @ theta_hat

# Plot
plt.scatter(X, y, label='Data', alpha=0.5)
plt.plot(X, y_pred, color='red', label='Best-Fit Line (from scratch)')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Linear Regression (from scratch)')
plt.legend()
plt.show()
```