Supervised Learning-Regression

Polynomial Regression using sample dataset

Polynomial Regression is an extension of linear regression where the relationship between the independent and dependent variables is modeled as an n-degree polynomial. It's useful when the data shows a non-linear relationship, but we still want to model it with linear techniques by transforming the input features.

The general form of a polynomial regression model is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \epsilon$$

where:

- y is the dependent variable.
- x is the independent variable.
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the polynomial terms.
- ϵ is the error term, accounting for noise in the data.

The degree of the polynomial n, determines the complexity of the model. For example:

- n=1: Linear regression (a straight line).
- n=2: Quadratic regression (a parabola).
- n=3: Cubic regression (a more flexible curve), and so on.

Why Use Polynomial Regression?

Polynomial regression is useful when data shows a non-linear trend that cannot be captured by a simple linear model. Examples include:

- Modeling the growth of populations.
- Predicting stock prices with seasonal trends.
- Tracking temperature fluctuations over time.

It provides a simple way to extend linear regression for capturing such non-linear patterns without resorting to more complex models like neural networks or non-parametric techniques.

Steps for Polynomial Regression

1. Import necessary libraries
2. Define the dataset
3. Transform the features into polynomial features
4. Fit the polynomial regression model
5. Predict new values
6. Visualize the result

Step 1: Import necessary libraries

- **numpy**: For handling array operations.
- **matplotlib.pyplot**: For plotting the graphs.
- **LinearRegression**: The linear regression model from sklearn.
- **PolynomialFeatures**: This will be used to transform the input features into polynomial features.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

Step 2: Define the dataset

- **X**: The independent variable (input), representing numbers 1 through 10.
- **Y**: The dependent variable (output), following a quadratic pattern (i.e., Y=X^2)

```
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
Y = np.array([1, 4, 9, 16, 25, 36, 49, 64, 81, 100])  # Quadratic data (Y = X^2)
```

Step 3: Transform the features into polynomial features

- **PolynomialFeatures(degree=2)**: Transforms the input features into polynomial features of degree 2. For example, if X = [x1], it becomes X_poly = [1, x1, x1^2].

- **fit_transform(X)**: Transforms the input array X into polynomial features.

```
poly_features = PolynomialFeatures(degree=2)
X_poly = poly_features.fit_transform(X)
```

Step 4: Fit the polynomial regression model

- **LinearRegression()**: Initializes the linear regression model.
- **model.fit(X_poly, Y)**: Trains the linear regression model using the polynomial features of X and the dependent variable Y.

```
model = LinearRegression()
model.fit(X_poly, Y)
```

Step 5: Make predictions using the model on the training data

- **model.predict(X_poly)**: Uses the trained polynomial regression model to make predictions based on the original X values transformed into polynomial features.

```
Y_pred = model.predict(X_poly)
```

Step 6: New data for prediction

- **new_data**: Represents new data points (11, 12, 13) that were not part of the original dataset.
- **new_data_poly**: Transforms the new data into polynomial features.
- **new_predictions**: Predicts the Y values for the new data using the polynomial regression model.

```
new_data = np.array([11, 12, 13]).reshape(-1, 1)
new_data_poly = poly_features.transform(new_data)
new_predictions = model.predict(new_data_poly)
```

Step 7: Display the predicted prices for new data

- **zip(new_data, new_predictions)**: Loops through each pair of new data points and their corresponding predicted values.
- **print()**: Displays the predictions for the new data.

```
print("Predicted values for new data:")
for data, price in zip(new_data, new_predictions):
    print(f"X: {data[0]} -> Predicted Y: {price:.2f}")
```

Step 8: Visualization

- **plt.scatter(X, Y, color="blue")**: Plots the original data points.
- **plt.plot(X, Y_pred, color="red")**: Plots the regression line (polynomial curve) based on the predicted values.
- **plt.scatter(new_data, new_predictions, color="green")**: Plots the predicted values for the new data points.
- **plt.show()**: Displays the plot.

```
plt.scatter(X, Y, color="blue", label="Actual Data")
plt.plot(X, Y_pred, color="red", label="Polynomial Regression Line")
plt.scatter(new_data, new_predictions, color="green", label="New Predictions")
plt.title("Polynomial Regression (Degree = 2)")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```