📘 Modules 4: Polynomial Regression

1. Addition of Polynomial Features

Concept

Linear Regression models a **linear relationship** between the independent variable *x* and the dependent variable *y*:

$$y = \beta_0 + \beta_1 x$$

However, real-world data is often **nonlinear**, such as curved or parabolic patterns.

> 💡 Therefore, we add **high-degree features (polynomial features)** to model this relationship:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_n x^n + \varepsilon$$

Expanded Feature Matrix

When adding polynomial features, the input matrix X is expanded to:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix}$$

Parameter Estimation (OLS)

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

Prediction

$$\hat{y} = X\hat{\beta}$$

Parameter Interpretation

Symbol	Meaning
β_0	Intercept coefficient
β_1, β_2, …, β_n	Regression coefficients for each degree of x
n	Polynomial degree
ε	Random error term

Illustrative Example

x	y
1	2
2	5
3	10

We choose a degree 2 model:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

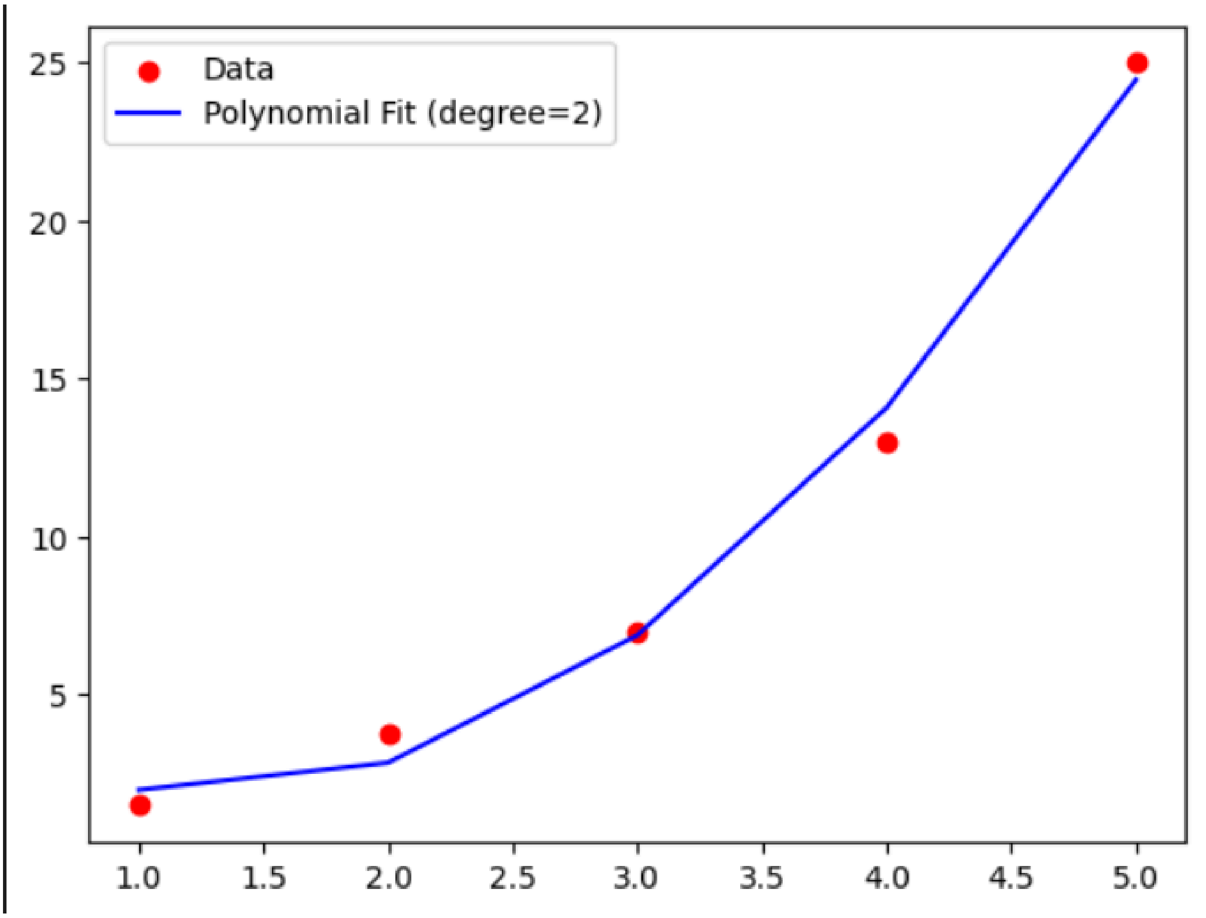
Estimation result:

$$\hat{y} = 0 + 0.5x + 1.5x^2$$

→ The curve fits the data closely.

Code Example

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

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([1.5, 3.8, 7.0, 13.0, 25.0])

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

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

plt.scatter(X, y, color='red', label='Data')
plt.plot(X, y_pred, color='blue', label='Polynomial Fit (degree=2)')
plt.legend()
plt.show()
```



2. Enhancing the Linear Model

Objective

"**Enhancing**" means **upgrading the linear model** so it can **model nonlinear relationships** between inputs and outputs — while maintaining its essence as "linear in the coefficients β".

Why Do We Need "Enhancing"?

Linear Regression assumes:

$$y = \beta_0 + \beta_1 x$$

→ can only draw a **straight line**.

While many natural phenomena have **curved** (nonlinear) relationships:

- Advertising spend vs revenue
- Age vs labor productivity
- Temperature vs machine performance

Linear Regression cannot model these patterns.

How to "Enhance"

Instead of changing the algorithm, we **transform the input (X)** by adding high-degree variables:

$$x \to [x, x^2, x^3, \dots, x^n]$$

→ New model:

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

Although it represents a curved relationship, the model is still **linear in the coefficients β**, so we can use the same OLS formula.

Two Main Objectives

Criterion	Meaning
Prediction	Model more accurately when data has curved trends.
Interpretation	Still able to understand the meaning of polynomial degrees (e.g., x^2 represents curvature).

Polynomial Regression balances accuracy and interpretability.

Geometric Intuition

Polynomial Degree	Curve Shape	Notes
1 (Linear)	Straight line	Simple but may underfit
2 (Quadratic)	Parabola	Usually fits curved data well
5+	Complex curve	Risk of overfitting

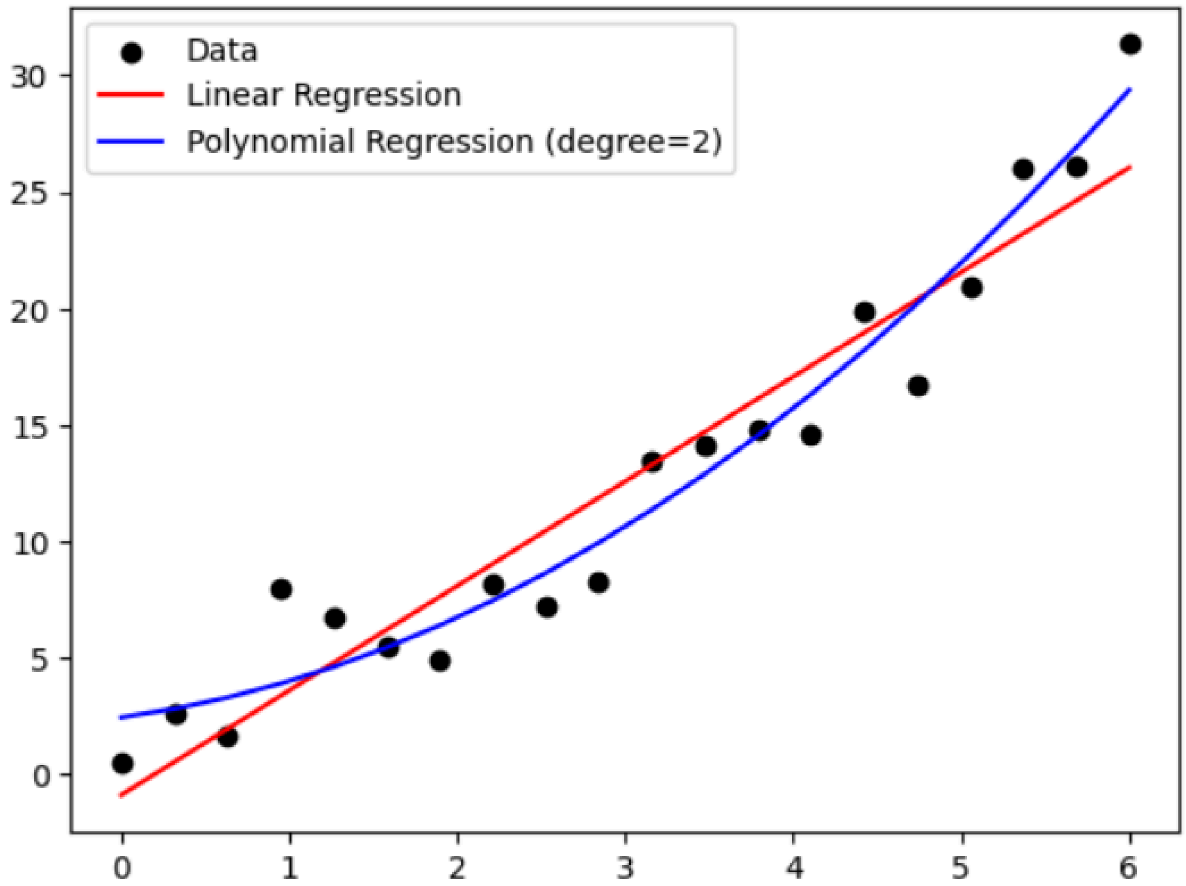
When to Use Polynomial Regression?

Situation	Decision
Curved, nonlinear data	✅ Yes
Linear data	❌ Not needed
Want interpretable model	✅ Can use
Noisy data, few samples	⚠️ Careful — choose low degree

Code Example for "Enhance"

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

X = np.linspace(0, 6, 20).reshape(-1, 1)
y = 2 + 1.5 * X + 0.5 * X**2 + np.random.randn(20) * 2

# Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
y_lin_pred = lin_reg.predict(X)

# Polynomial Regression
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)
y_poly_pred = poly_reg.predict(X_poly)

# Visualization
```

```
plt.scatter(X, y, color='black', label='Data')
plt.plot(X, y_lin_pred, color='red', label='Linear Regression')
plt.plot(X, y_poly_pred, color='blue', label='Polynomial Regression (degree
=2)')
plt.legend()
plt.show()
```



Summary of "Enhancing"

Enhancing the Linear Model = transforming Linear Regression into a flexible nonlinear model by adding polynomial features.

It helps the model capture curved relationships while remaining easy to interpret and fast to train.

3. Extending the Linear Model

Meaning

After **enhancing** Linear Regression with polynomial features, we can **extend** this idea to **more powerful models** for both regression and classification tasks.

Common Extended Models

Model	Task Type	Explanation
Logistic Regression	Classification	Predicts probability of event occurrence (0/1).
K-Nearest Neighbors (KNN)	Regression & Classification	Predicts based on nearest neighbors.
Decision Trees	Regression & Classification	Splits data into nodes based on thresholds.
Support Vector Machines (SVM)	Regression & Classification	Finds optimal hyperplane to separate data.
Random Forests	Regression & Classification	Combines multiple decision trees for higher accuracy.
Ensemble Methods	Both	Combines multiple small models to improve generalization.
Deep Learning	Both	Multi-layer neural networks that model complex nonlinear relationships.

Summary of "Extending"

> **"Enhancing"** improves Linear Regression using polynomial features.
>
> **"Extending"** expands the Linear Regression concept into more powerful models (KNN, SVM, Trees, Neural Networks...).

---

4. Summary + Learning Recap

What We've Learned

- How to add **Polynomial Features** to capture nonlinear relationships
- How to **enhance Linear Regression (Enhancing)** while keeping it interpretable
- How to **extend (Extending)** to more complex models

Quick Summary

Section	Key Content
Addition of Polynomial Features	Adding high-degree features (x^2, x^3, ...)
Enhancing the Linear Model	Improving Linear Regression to capture nonlinear relationships
Extending the Linear Model	Expanding Linear Regression ideas to other models
Summary + Recap	Consolidating theory and applications

Final Conclusion

Polynomial Regression bridges Linear Regression and more complex nonlinear models.

It enables modeling of curved relationships, improving accuracy, and serves as the foundation for understanding advanced machine learning algorithms like **SVM, Tree-based models, Ensemble methods, and Deep Learning**.