When you train an ML model, you want it to **learn the underlying pattern** in data

| **Concept** | **Description** |
| --- | --- |
| **Underfitting** | The model is **too simple** to capture the true pattern in the data. |
| **Overfitting** | The model is **too complex**, capturing noise or random fluctuations instead of the true pattern. |

# Underfitting

Underfitting happens when a model is **too simple** to learn the underlying pattern in the data.  
It **cannot capture important relationships** between input and output.  
As a result, the model performs **poorly on both training and testing data**.

# Overfitting

Overfitting happens when a model is **too complex** and learns **not only the real patterns but also the noise** or random fluctuations in the training data.  
It performs **very well on training data** but **poorly on new unseen data**.

# Real-Life Examples

| **Situation** | **Description** |
| --- | --- |
| **Student Learning** | Underfitting → student doesn’t study enough, can’t answer even simple questions. Overfitting → student memorizes exact words of the book but fails to answer in different wording. |
| **Weather Prediction** | Underfitting → model only uses temperature, ignoring humidity and wind. Overfitting → model learns exact noise from past data, fails when weather changes slightly. |
| **House Price Prediction** | Underfitting → only uses one feature (size). Overfitting → uses hundreds of unnecessary features like owner’s name, date, etc., and memorizes training houses. |

# Code and Visualization

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

# Step 1: Generate sample data (non-linear)

np.random.seed(0)

X = np.sort(5 \* np.random.rand(80, 1), axis=0)

y = np.sin(X).ravel() + np.random.randn(80) \* 0.3  # True pattern + noise

# Split into train and test data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 2: Try 3 different model complexities

degrees = [1, 4, 15]

plt.figure(figsize=(15, 4))

for i, degree in enumerate(degrees):

    # Create polynomial features

    poly = PolynomialFeatures(degree=degree)

    X\_poly = poly.fit\_transform(X\_train)

    # Train linear regression model

    model = LinearRegression()

    model.fit(X\_poly, y\_train)

    # Predict

    X\_plot = np.linspace(0, 5, 100).reshape(-1, 1)

    y\_plot = model.predict(poly.transform(X\_plot))

    # Plot

    plt.subplot(1, 3, i+1)

    plt.scatter(X\_train, y\_train, color='blue', label='Training Data')

    plt.plot(X\_plot, y\_plot, color='red', label=f'Degree {degree}')

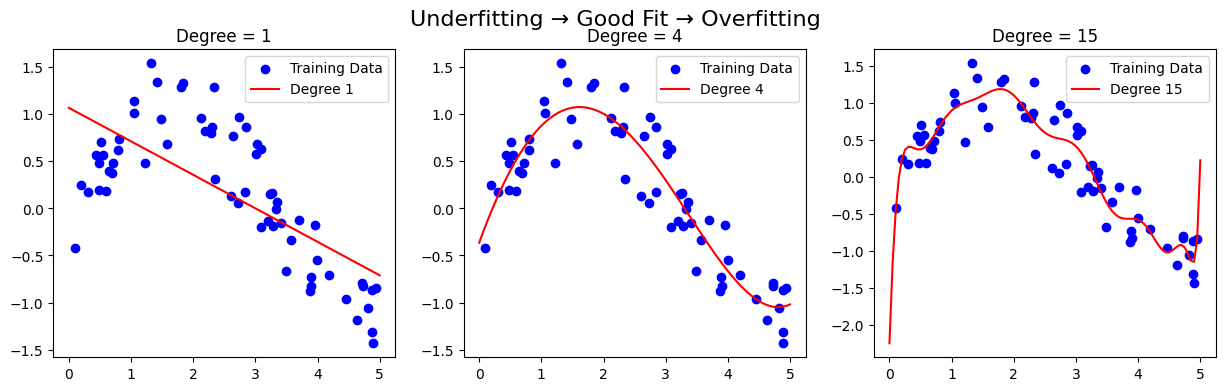
    plt.title(f"Degree = {degree}")

    plt.legend()

plt.suptitle("Underfitting → Good Fit → Overfitting", fontsize=16)

plt.show()

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Visualization (conceptually) | Training Error | Test Error |
| Underfitting | Almost a straight line missing many points | High | High |
| Good Fit | Smooth curve following data trend | Low | Low |
| Overfitting | Very wiggly curve touching every point | Very Low | High |



# Fix

|  |  |  |  |
| --- | --- | --- | --- |
| Concept | Description | Graph Pattern | Fix |
| Underfitting | Model too simple; can’t capture trend | Both training & testing errors high |  Use a **more complex model** (e.g., add polynomial degree, deeper network)   Add more relevant **features**   Reduce **regularization strength**   Train longer (if using iterative methods) |
| Overfitting | Model too complex; memorizes data | Training error low, testing error high |  Use **regularization** (Ridge, Lasso, Dropout, etc.)   Use **cross-validation**   Collect **more data**   Use **simpler models**   Perform **feature selection**   Use **early stopping** (in iterative training)   Use **data augmentation** (for images, text, etc.) |
| Good Fit | Model captures true pattern | Both errors low | Keep tuning & validating |

# Test and Train Error

## Training Error (or Training Loss)

* It is the **error the model makes on the data it was trained on.**
* It shows **how well the model has learned the patterns in the training dataset.**

**Definition:**

Training error = Difference between the predicted output and the actual output **on the training data**.

📌 **Example:**  
If your model was trained on 1000 house prices, and it predicts them with an average difference of $5,000,  
then the **training error** = $5,000.

👉 **Low training error** → the model learned the training data well.  
👉 **High training error** → the model is too simple (underfitting).

## Testing Error (or Validation Error)

* It is the **error the model makes on new, unseen data** — data that was **not used during training.**
* It tells you **how well the model generalizes** to new data.

📘 **Definition:**

Testing error = Difference between predicted output and actual output **on the test data**.

📌 **Example:**  
You test your model on 200 *new* house prices, and it predicts them with an average difference of $9,000.  
Then the **testing error** = $9,000.

👉 **Low testing error** → good generalization.  
👉 **High testing error** → possible overfitting.

## Comparing Training and Testing Errors

|  |  |  |  |
| --- | --- | --- | --- |
| Model Behavior | Training Error | Testing Error | Meaning |
| Underfitting | High | High | Model is too simple; not learning the pattern even in training data. |
| Good Fit | Low | Low | Model learned the pattern and generalizes well. |
| Overfitting | Very Low | High | Model memorized training data; fails on new data. |