Lesson: Overfitting, Underfitting, and Model Generalization

---

1. Introduction

Machine learning models aim to **learn patterns** from data and make accurate predictions. However, a model's performance depends on how well it generalizes to **new, unseen data**.

- **Overfitting:** The model learns too much from training data, capturing noise instead of general patterns.
- **Underfitting:** The model is too simple and fails to capture important relationships in the data.
- **Generalization:** A well-balanced model that performs well on both training and test data.

---

2. Overfitting: When a Model Learns Too Much

Definition:

Overfitting occurs when a model **memorizes the training data** instead of learning the underlying pattern. It performs well on training data but poorly on test data.

Causes:

1. **Too complex model** (e.g., too many layers in a neural network, too many decision tree splits).
2. **Insufficient training data** leading to memorization instead of learning.
3. **Too many features (high dimensionality)** causing unnecessary noise.

Symptoms:

- High accuracy on training data but low accuracy on validation/test data.
- Large gap between training loss and validation loss.

Solutions to Overfitting:

✅ **Increase training data** – Helps the model generalize better. ✅ **Reduce Model Complexity** – Simplify architecture or reduce parameters. ✅ **Cross-validation** – Evaluating model performance on different subsets of data. ✅ **Early Stopping** – Stops training when validation error starts increasing.

---

3. Underfitting: When a Model Learns Too Little

Definition:

Underfitting happens when a model is **too simple** and cannot capture the patterns in the data, leading to poor performance on both training and test datasets.

Causes:

1. **Model is too simple** (e.g., a linear model for complex relationships).
2. **Insufficient training time** (e.g., stopping training too early).
3. **Too few features** – Not enough information to capture relationships.

Symptoms:

- Both training and test errors are high.
- Model performs similarly on training and test data, but both have low accuracy.

Solutions to Underfitting:

✅ **Increase Model Complexity** – Use deeper networks or more complex algorithms. ✅ **Feature Engineering** – Add more relevant features. ✅ **Train for Longer Epochs** – Allow model to learn better representations.

---

4. Model Generalization: The Ideal Scenario

A well-generalized model:

- **Performs well on both training and test data.**
- **Has a balanced bias-variance tradeoff.**
- **Achieves low error without memorizing noise.**

Bias-Variance Tradeoff:

- **High bias (Underfitting)** → Model is too simple and makes systematic errors.
- **High variance (Overfitting)** → Model is too sensitive to training data.
- **Goal:** Find a balance where bias and variance are both minimized.

- **High Bias (Underfitting)** → A model with **high bias** makes strong assumptions about the data and oversimplifies the relationship between input and output. It lacks flexibility, leading to **systematic errors** (bias) because it fails to capture complex patterns in the data. This results in **high training and test errors**.

  - Example: A **linear regression model** used on a dataset with nonlinear relationships will have high bias and poor predictive performance.

- **High Variance (Overfitting)** → A model with **high variance** is too sensitive to small fluctuations in the training data. It captures even the noise, making it highly adaptable to training data but performing poorly on new, unseen data. This results in **low training error but high test error** (poor generalization).

  - Example: A **deep neural network with excessive layers** trained on limited data may memorize the training set instead of learning general patterns.

- **The Tradeoff**:
    - A simple model (high bias) is prone to **underfitting**, leading to high error on both training and test data.
    - A complex model (high variance) is prone to **overfitting**, where training error is low but test error is high.
    - The goal is to find the **optimal balance**, where the model is **complex enough** to learn meaningful patterns but **not too complex** to memorize noise.

---

5. Practical Example: Overfitting vs Underfitting in a Regression Model

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

# Generate synthetic data
np.random.seed(42)
x = np.linspace(0, 10, 50)
y = np.sin(x) + np.random.normal(scale=0.3, size=x.shape)
x = x.reshape(-1, 1)

# Underfitting: Linear Model
linear_model = LinearRegression()
linear_model.fit(x, y)
y_pred_linear = linear_model.predict(x)

# Overfitting: High-degree Polynomial Model
poly_model = make_pipeline(PolynomialFeatures(degree=10), LinearRegression())
poly_model.fit(x, y)
y_pred_poly = poly_model.predict(x)

# Plot Results
plt.figure(figsize=(10,5))
plt.scatter(x, y, label='True Data')
plt.plot(x, y_pred_linear, label='Underfitting (Linear)', color='red')
plt.plot(x, y_pred_poly, label='Overfitting (Polynomial)', color='green')
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Underfitting vs Overfitting")
plt.show()
```

Explanation of Code:

- We generate **synthetic data** representing a noisy sine wave.
- A **linear regression model** (red) fails to capture the wave → **Underfitting**.
- A **high-degree polynomial model** (green) fits too tightly → **Overfitting**.
- The ideal model would be a **moderate-degree polynomial** balancing both.

---

6. Evaluating Overfitting and Underfitting Using Metrics

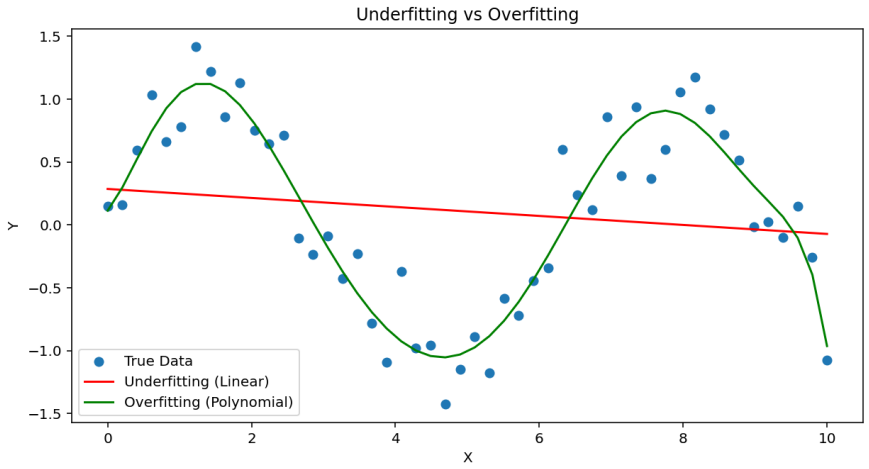
Common Evaluation Metrics:

1. **Mean Squared Error (MSE):** Measures average squared difference between actual and predicted values.
2. **Root Mean Squared Error (RMSE):** Square root of MSE, making it more interpretable.
3. **R^2 Score:** Measures how well the model explains variance in the data.

```
from sklearn.metrics import mean_squared_error, r2_score

# Compute metrics
mse_linear = mean_squared_error(y, y_pred_linear)
r2_linear = r2_score(y, y_pred_linear)

mse_poly = mean_squared_error(y, y_pred_poly)
r2_poly = r2_score(y, y_pred_poly)

print(f"Linear Model - MSE: {mse_linear:.2f}, R² Score: {r2_linear:.2f}")
print(f"Polynomial Model - MSE: {mse_poly:.2f}, R² Score: {r2_poly:.2f}")
```

Interpretation:

- If **MSE is too low on training data but high on test data**, the model is **overfitting**.
- If **MSE is high on both training and test data**, the model is **underfitting**.
- An **ideal model** has a **low MSE** and a **high R^2 score** across both datasets.



This graph visually demonstrates the concepts of **underfitting** and **overfitting** in a regression problem using a synthetic dataset. Here's a detailed breakdown:

1. Understanding the Data (Blue Dots)

- The blue dots represent the **true data points**, which follow a sine wave with some added noise.

- These points are the **ground truth**, meaning a good model should learn the general pattern of this dataset without capturing too much noise.

2. Underfitting (Red Line - Linear Model)

- The red line represents a **linear regression model**, which is too simple to capture the nonlinear relationship in the data.
- This results in **high bias**, meaning the model makes systematic errors by failing to capture important variations in the data.
- **Characteristics of underfitting in the graph:**
  - The red line does not follow the curvature of the data at all.
  - It shows almost no variation, treating the relationship as nearly flat.
  - High error in both training and test datasets, as it fails to learn the underlying pattern.

3. Overfitting (Green Line - High-Degree Polynomial Model)

- The green line represents a **high-degree polynomial regression model**, which tries to fit the training data too precisely.
- It captures **both the actual trend and the noise**, making it overly complex.
- **Characteristics of overfitting in the graph:**
  - The green line fluctuates significantly, following every small deviation in the data.
  - It fits the training data almost perfectly but will likely perform poorly on new, unseen data.
  - It has **low training error but high test error**, meaning it memorizes rather than generalizes.

4. Finding the Right Model

- A **well-generalized model** should be **somewhere between the red and green lines**, capturing the **main pattern without overfitting the noise**.
- A **moderate-degree polynomial regression** (e.g., degree 3 or 4) would likely be ideal in this scenario.

Key Takeaways

✅ The **red line (underfitting)** fails to learn the relationship, leading to high bias.
✅ The **green line (overfitting)** learns too much, leading to high variance.
✅ A good model balances both **bias and variance**, achieving low error on both training and test data.

7. Conclusion

✅ **Overfitting** happens when a model memorizes the training data but performs poorly on new data. ✅ **Underfitting** happens when a model is too simple to learn patterns from data. ✅ **Generalization** is the goal, where a model balances bias and variance. ✅ **Techniques like regularization, cross-validation, and feature selection** help combat overfitting and underfitting.