Lesson: Understanding Overfitting, Underfitting, and Generalization Using the Iris Dataset

1. Introduction to the Iris Dataset

The **Iris dataset** is one of the most famous datasets in machine learning. It is often used for classification problems and is an excellent example to understand **overfitting, underfitting, and generalization**.

Dataset Overview

- **Goal:** Classify flowers into three species based on four features.
- **Features (Input variables):**
    1. **Sepal Length (cm)**
    2. **Sepal Width (cm)**
    3. **Petal Length (cm)**
    4. **Petal Width (cm)**
- **Target variable (Output class):**
    - **Setosa**
    - **Versicolor**
    - **Virginica**
- **Dataset size:** 150 samples (50 samples per class)

Why Use the Iris Dataset?

✅ Small and easy to visualize.
✅ Well-balanced (each class has the same number of samples).
✅ Useful for understanding **bias-variance tradeoff**.

---

2. Understanding Overfitting, Underfitting, and Generalization

When training a machine learning model on the **Iris dataset**, we want to find a model that correctly classifies new flowers while avoiding **underfitting and overfitting**.

Underfitting (High Bias)

📌 **Definition:** The model is too simple and fails to learn the patterns in the data.
📌 **Example in Iris Dataset:** Using a **linear classifier** to separate all three species when the

decision boundaries are nonlinear.
📌 **Symptoms:**

- High training and test error.
- The model misclassifies even training examples.
- The decision boundary is too simple.

Overfitting (High Variance)

📌 **Definition:** The model is too complex and learns unnecessary noise from the training data.
📌 **Example in Iris Dataset:** Using a **very deep decision tree** that memorizes training examples but performs poorly on new data.
📌 **Symptoms:**

- **Near-perfect accuracy on training data but poor performance on test data.**
- Decision boundary is too complex, capturing noise rather than the real pattern.

Generalization (Ideal Scenario)

📌 **Definition:** The model **learns patterns** in the training data and performs well on unseen data.
📌 **Goal:** Balance **bias and variance** so that the model can correctly classify new flowers.
📌 **Example in Iris Dataset:** Using a **moderate-depth decision tree** or **SVM with an appropriate kernel**.

---

3. Practical Example: Classifying the Iris Dataset

Step 1: Load the Dataset

```
from sklearn.datasets import load_iris
import pandas as pd

# Load Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target  # 0: Setosa, 1: Versicolor, 2: Virginica

# Display first few rows
print(df.head())
```

✅ **Outcome:** Loads the dataset into a DataFrame for easy manipulation.

Step 2: Splitting Data (Training vs. Testing)

To evaluate generalization, we split the data:

```
from sklearn.model_selection import train_test_split

# Split into train (80%) and test (20%) sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, 
test_size=0.2, random_state=42)
```

✅ **Why?** Ensures we test the model on **unseen** data to detect overfitting.

Step 3: Training an Underfitting Model (Linear Model)

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Train a simple model (underfitting)
model_underfit = LogisticRegression(max_iter=50)  # Low iterations
model_underfit.fit(X_train, y_train)

# Evaluate performance
train_acc = accuracy_score(y_train, model_underfit.predict(X_train))
test_acc = accuracy_score(y_test, model_underfit.predict(X_test))

print(f"Underfitting Model - Train Accuracy: {train_acc:.2f}, Test Accuracy: 
{test_acc:.2f}")
```

📌 **Expected Result:**

- Both train and test accuracy will be **low** (~70-80%).
- The model is **too simple** and does not capture class differences well.

Step 4: Training an Overfitting Model (Complex Model)

```
from sklearn.tree import DecisionTreeClassifier

# Train a highly complex model (overfitting)
model_overfit = DecisionTreeClassifier(max_depth=10)  # Overly deep tree
model_overfit.fit(X_train, y_train)

# Evaluate performance
train_acc = accuracy_score(y_train, model_overfit.predict(X_train))
test_acc = accuracy_score(y_test, model_overfit.predict(X_test))

print(f"Overfitting Model - Train Accuracy: {train_acc:.2f}, Test Accuracy: 
{test_acc:.2f}")
```

📌 **Expected Result:**

- **Train accuracy will be 100%** (memorization of training data).
- **Test accuracy will drop significantly** (~80-90%) due to **poor generalization**.

Step 5: Training a Well-Generalized Model

```
# Train a well-balanced model
model_balanced = DecisionTreeClassifier(max_depth=3)  # Moderate complexity
model_balanced.fit(X_train, y_train)

# Evaluate performance
train_acc = accuracy_score(y_train, model_balanced.predict(X_train))
test_acc = accuracy_score(y_test, model_balanced.predict(X_test))

print(f"Balanced Model - Train Accuracy: {train_acc:.2f}, Test Accuracy:
{test_acc:.2f}")
```

📌 **Expected Result:**

- **High but not perfect training accuracy** (~95%).
- **Similar test accuracy** (~92-96%), meaning good generalization.

---

4. Visualizing Decision Boundaries

To see overfitting vs. underfitting in action, let's plot the decision boundaries.

```
import numpy as np
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions

# Convert dataset to 2D for visualization (using petal length & width)
X_vis = iris.data[:, [2, 3]]
y_vis = iris.target

# Train models
model_underfit.fit(X_vis, y_vis)
model_overfit.fit(X_vis, y_vis)
model_balanced.fit(X_vis, y_vis)

# Plot decision boundaries
plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
plot_decision_regions(X_vis, y_vis, clf=model_underfit)
plt.title("Underfitting Model (Linear)")

plt.subplot(1, 3, 2)
plot_decision_regions(X_vis, y_vis, clf=model_overfit)
plt.title("Overfitting Model (Deep Tree)")

plt.subplot(1, 3, 3)
plot_decision_regions(X_vis, y_vis, clf=model_balanced)
plt.title("Balanced Model (Good Generalization)")

plt.show()
```

✅ **Interpretation:**

- The **underfitting model** has **simple decision boundaries** that misclassify many points.
- The **overfitting model** has **highly irregular boundaries** that fit the training data too closely.
- The **balanced model** has a **smooth, reasonable separation** that generalizes well.

---

5. Conclusion

✅ **Underfitting** happens when the model is too simple and has **high bias**.
✅ **Overfitting** occurs when the model is too complex and has **high variance**.
✅ **Generalization** is the key to **good model performance** on unseen data.
✅ The **Iris dataset** is a great example to visualize these concepts using **decision boundaries**.