8. Apply Support Vector algorithm for classification.

**Description:**

Support Vector Machine (SVM) is a supervised learning algorithm used for classification, regression, and outlier detection. It is particularly effective for high-dimensional datasets and is widely used in applications such as text classification, image recognition, and bioinformatics.

SVM aims to find the **optimal hyperplane** that best separates different classes in a dataset.

1. **Hyperplane:**
   * A hyperplane is a decision boundary that separates different classes.
   * In a **2D space**, it is a line.
   * In a **3D space**, it is a plane.
   * In **higher dimensions**, it becomes a more complex structure.
2. **Support Vectors:**
   * These are the **data points closest to the hyperplane**, which influence its position and orientation.
   * The algorithm focuses only on these points to create the best separation.
3. **Margin:**
   * The distance between the hyperplane and the nearest support vectors.
   * **SVM tries to maximize this margin** to improve generalization.

**Types of SVM**

**1️⃣ Linear SVM:**

* Used when the data is **linearly separable** (i.e., can be divided with a straight line).
* The algorithm finds a hyperplane that best separates the two classes.

**2️⃣ Non-Linear SVM:**

* Used when the data is **not linearly separable**.
* Uses the **Kernel Trick** to map data into a higher-dimensional space where it can be separated linearly.

**Kernel Trick in SVM**

The **kernel function** helps transform non-linearly separable data into a higher dimension where it becomes linearly separable.

**Common Kernels:**

* **Linear Kernel:** K(x,y)=x⋅y
* **Polynomial Kernel:** K(x,y)=(x⋅y+c)d
* **Radial Basis Function (RBF) Kernel:** K(x,y)=e−γ∣∣x−y∣∣2
* **Sigmoid Kernel:** K(x,y)=tanh(αx⋅y+c)

To apply a **Support Vector Machine (SVM)** algorithm for classification, follow these steps:

**1. Import Required Libraries**

2. Load the Dataset

3. Data Preprocessing

Split the data into training and testing sets.

Standardize the features (important for SVM).

**4. Train the SVM Model**

Choose the kernel type (e.g., 'linear', 'rbf', 'poly').

5. Make Predictions

**6. Evaluate the Model**

* Confusion Matrix
* Classification Report

**7. Visualize Decision Boundaries (Optional)**

* Visualizing SVM decision boundaries is only feasible with 2D data. For simplicity, we use only the first two features of the Iris dataset:

**8. Tips for Tuning SVM**

* **Kernel:** Try different kernels like 'linear', 'rbf' (Gaussian), and 'poly'.
* **C Parameter:** Controls trade-off between smooth decision boundary and classifying training points correctly.
  + Higher C -> less regularization (focuses more on classifying all training data correctly).
  + Lower C -> more regularization (smoother decision boundary).
* **Gamma (for 'rbf' and 'poly' kernels):** Defines influence of a single training example.
  + Higher gamma -> closer decision boundary to data points.
  + Lower gamma -> decision boundary influenced by more data points.

Python code

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, confusion\_matrix

# Step 1: Load dataset (Iris dataset for simplicity)

data = datasets.load\_iris()

X = data.data

y = data.target

# Step 2: Preprocess the data

# We’ll standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Step 3: Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Step 4: Train SVM model # Using 'rbf' kernel, C=1.0, and gamma='scale'

svm = SVC(kernel='rbf', C=1.0, gamma='scale')

svm.fit(X\_train, y\_train)

# Step 5: Make predictions

y\_pred = svm.predict(X\_test)

# Step 6: Evaluate the model

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Visualize decision boundaries (only for 2 features at a time)

def plot\_decision\_boundaries(X, y, model, title="Decision Boundary"):

h = .02 # step size in the mesh

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title(title)

plt.show()

# Train another SVM model for visualization (Using only first two features)

X\_train\_2D, X\_test\_2D = X\_train[:, :2], X\_test[:, :2] # Use only first two features

svm\_2D = SVC(kernel='rbf', C=1.0, gamma='scale')

svm\_2D.fit(X\_train\_2D, y\_train)

# Plotting decision boundary for the first two features

plot\_decision\_boundaries(X\_train\_2D, y\_train, svm\_2D)

OutPut: