Assignment: Support Vector Machine (SVM) Classification Using Linear Kernel

Aim

To implement a Support Vector Machine classifier using a linear kernel on a simple 2D dataset, visualize the decision boundary, and evaluate the classifier's accuracy on test data.

Input

- A set of 2D data points (features) with corresponding binary class labels.
- Data is split into training (80%) and testing (20%) sets.

Example input data:

text
X = [[1, 2], [2, 3], [3, 3], [6, 6], [7, 7], [8, 8]]
y = [0, 0, 0, 1, 1, 1]

Expected Output

- Printed accuracy score of the SVM classifier on the test data.
- A plot showing:
    - Data points colored by class.
    - The decision boundary separating the two classes.
    - Support vectors highlighted.

Example output:

text
Accuracy: 1.00

Plus a visualization plot displaying the classification regions, data points, and support vectors.

Theory

Support Vector Machine (SVM) is a supervised machine learning algorithm used primarily for classification tasks. The goal of SVM is to find the hyperplane that best separates classes in the feature space by maximizing the margin between the closest points (support vectors) of each class. For linearly separable data, this hyperplane is linear.

Key concepts:

- Hyperplane: The decision boundary that separates classes.
- Support Vectors: Data points closest to the hyperplane that influence its position.
- Margin: The distance between the hyperplane and the nearest support vector; SVM aims to maximize this margin to improve generalization.
- Kernel: Helps SVM perform non-linear classification by mapping data into higher-dimensional spaces (here, a linear kernel is used).

Steps / Algorithm

1. Data Preparation:
    - Collect the dataset with features and labels.
    - Split the data into training and testing subsets.
2. Model Creation:
    - Initialize the SVM classifier with a linear kernel.

3. Training:
    - Train the classifier on the training data.
4. Prediction:
    - Predict the class labels for the testing data using the trained model.
5. Evaluation:
    - Calculate the accuracy by comparing predicted labels with true labels.
6. Visualization:
    - Plot the decision boundary and margin.
    - Mark the support vectors and data points.

Code Implementation

```
python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Sample 2D data points and labels
X = np.array([[1, 2], [2, 3], [3, 3], [6, 6], [7, 7], [8, 8]])
y = np.array([0, 0, 0, 1, 1, 1])
# Split data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create and train the linear SVM model
clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
# Predict on the test set
y_pred = clf.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
# Plotting decision boundary and data points
h = 0.02  # 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))
# Predict on mesh grid points to get decision boundary
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# Generate plot
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.Paired)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, edgecolors='k', cmap=plt.cm.Paired, label="Data points")
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100, facecolors='none', edgecolors='r',
label="Support Vectors")
plt.title("SVM Classification with Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

Conclusion

The Linear SVM classifier successfully separated the two classes in the 2D dataset with perfect accuracy on the test set. The plotted decision boundary clearly shows the separating hyperplane, and the marked support vectors indicate the critical points influencing the margin. This lab demonstrated the

effectiveness of SVM for linearly separable data and how it can be implemented using Python's scikit-learn library.