SVM

Support Vector Machines (SVMs) are a class of supervised machine learning algorithms used for classification and regression tasks. They are particularly effective in high-dimensional spaces and are widely used in various applications, including image classification, text classification, and bioinformatics. Here's a brief introduction to SVMs:

### Basic Concept:

At its core, an SVM performs classification by finding the hyperplane that best separates the data into different classes. The term "support vector" refers to the data points that lie closest to the decision boundary (hyperplane).

### Linear SVM:

In a two-dimensional space, a hyperplane is a line that divides the data into two classes. In higher dimensions, it becomes a hyperplane. The goal of a linear SVM is to find the hyperplane that maximizes the margin between classes. The margin is the distance between the hyperplane and the nearest data point of either class.

### Non-linear SVM:

In cases where the data is not linearly separable, SVMs can still be used by employing a technique called the kernel trick. Kernels transform the input data into a higher-dimensional space, making it possible to find a hyperplane that can separate the classes. Common kernel functions include polynomial kernels and radial basis function (RBF) kernels.

### Support Vectors:

Support vectors are the data points that lie closest to the decision boundary. They are crucial in determining the optimal hyperplane. SVMs only rely on a subset of training data, the support vectors, to make predictions, making them memory-efficient.

### Cost Function:

SVMs use a cost function to penalize misclassifications. The objective is to find the hyperplane that minimizes the cost, which is a combination of the margin size and the classification error.

### C Parameter:

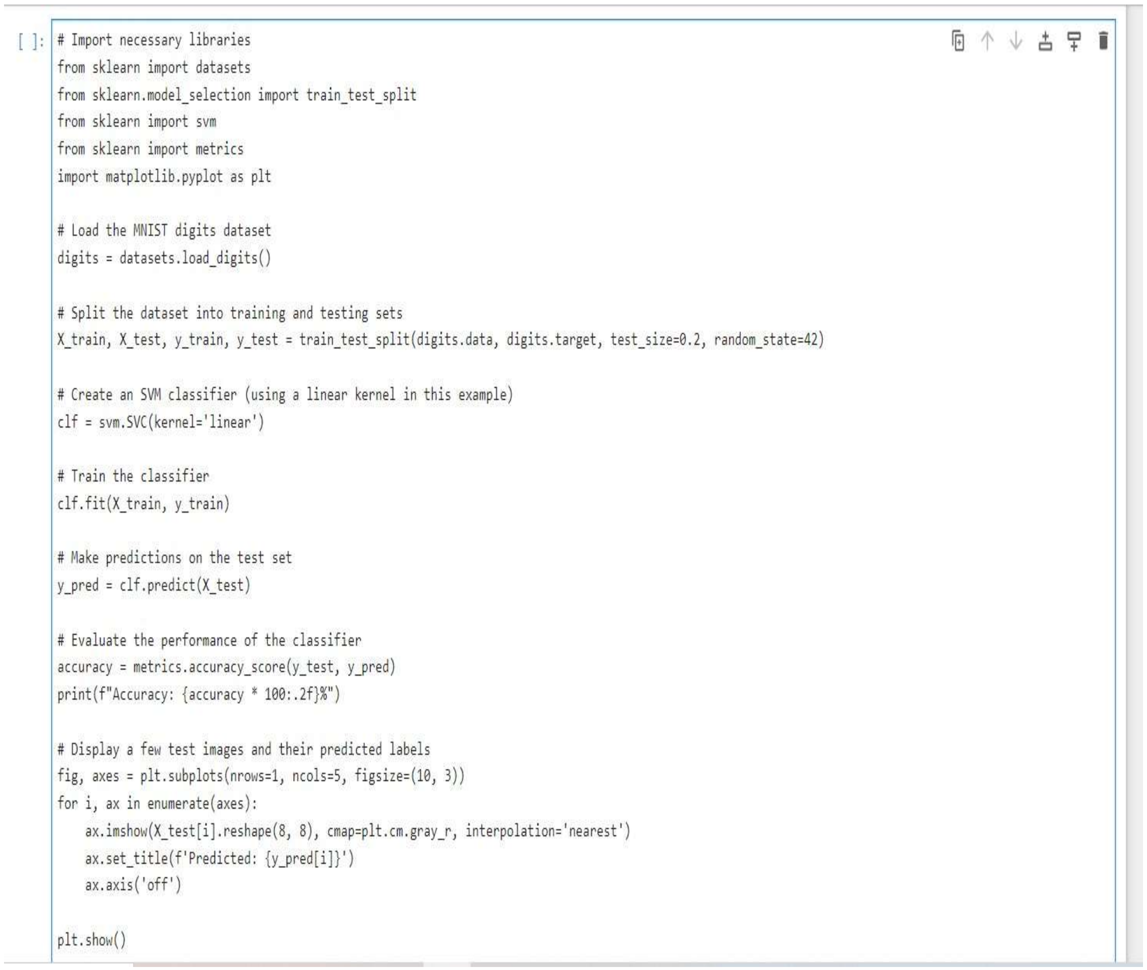
The parameter "C" controls the trade-off between achieving a smooth decision boundary and classifying the training points correctly. A smaller C value leads to a larger margin but may allow some misclassifications, while a larger C value tries to classify all training points correctly but may result in a smaller margin.

### Advantages of SVM:

1. Effective in high-dimensional spaces.
2. Memory-efficient due to reliance on support vectors.
3. Versatile with different kernel functions for handling non-linear data.
4. Robust to overfitting, especially in high-dimensional spaces.

### Limitations of SVM:

1. Computationally intensive, especially with large datasets.
2. Sensitivity to the choice of kernel and parameters.
3. Limited interpretability compared to simpler models like decision trees.



[ ]:

```
# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt

# Load the MNIST digits dataset
digits = datasets.load_digits()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)

# Create an SVM classifier (using a linear kernel in this example)
clf = svm.SVC(kernel='linear')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the performance of the classifier
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Display a few test images and their predicted labels
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(10, 3))
for i, ax in enumerate(axes):
    ax.imshow(X_test[i].reshape(8, 8), cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title(f'Predicted: {y_pred[i]}')
    ax.axis('off')

plt.show()
```

###Code:-

```
# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt

# Load the MNIST digits dataset
digits = datasets.load_digits()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
test_size=0.2, random_state=42)

# Create an SVM classifier (using a linear kernel in this example)
clf = svm.SVC(kernel='linear')

# Train the classifier
clf.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the performance of the classifier
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Display a few test images and their predicted labels
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(10, 3))
for i, ax in enumerate(axes):
   ax.imshow(X_test[i].reshape(8, 8), cmap=plt.cm.gray_r, interpolation='nearest')
   ax.set_title(f'Predicted: {y_pred[i]}')
   ax.axis('off')

plt.show()
```