EXPERIMENT – 09

AIM: To implement a Support Vector Machine (SVM) for a Character Recognition Task and analyze its performance.

DESCRIPTION:

A **Support Vector Machine (SVM)** is a powerful **machine learning algorithm** widely used for both **linear and nonlinear classification**, as well as **regression** and **outlier detection** tasks. SVMs are highly adaptable, making them suitable for various applications such as **text classification**, **image classification**, **spam detection**, **handwriting identification**, **gene expression analysis**, **face detection**, and **anomaly detection**.
SVMs are particularly effective because they focus on finding the **maximum separating hyperplane** between the different classes in the target feature, making them robust for both **binary and multiclass classification**. In this outline, we will explore the **Support Vector Machine (SVM)** algorithm, its applications, and how it effectively handles both **linear and nonlinear classification**, as well as **regression** and **outlier detection** tasks.

Character recognition is the process of classifying written characters into specific categories (e.g., letters or digits). Here, we will use an SVM with an **RBF (Radial Basis Function)** kernel to classify handwritten characters in the MNIST dataset, a popular dataset containing images of handwritten digits from 0 to 9. The SVM will learn to distinguish each character based on the pixel intensity values.

CODE:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np

df = datasets.load_digits()
df.data.shape
```

```
(1797, 64)
```

```
# obtain Train data and Train output
X = df.data  # Each image is flattened to 64 features (8x8 images here for demonstration)
y = df.target
```

```
X
```

```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

```
y
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

```
# Preprocess: Scale data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2)
```

```
# Initialize SVM with RBF kernel
svm_model = SVC(kernel='rbf', gamma=0.01, C=1)
```

```
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

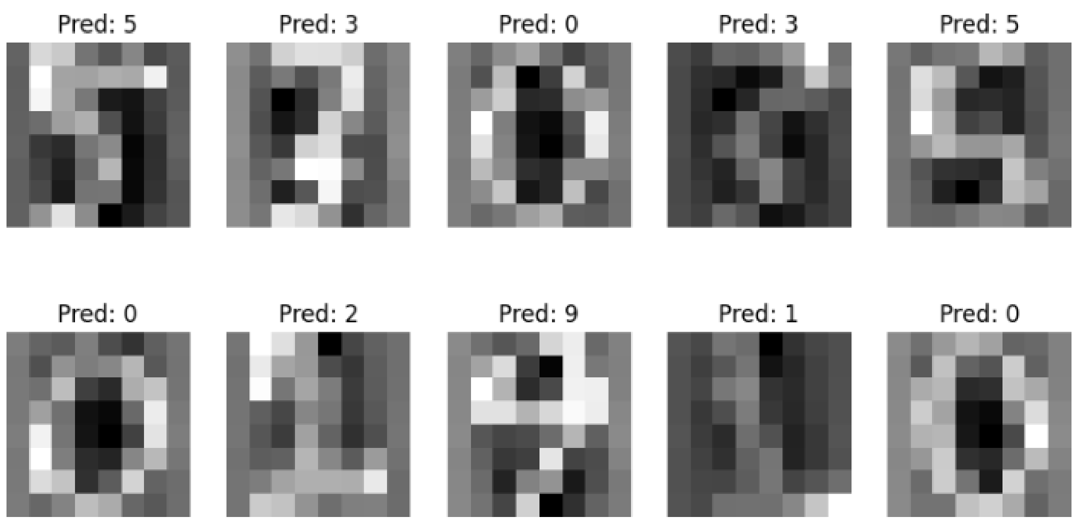
```
print("Accuracy: ", accuracy)
print("Classification Report:\n", classification_rep)
```

```
Accuracy:  0.975
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.97      1.00      0.99        35
           2       1.00      0.98      0.99        42
           3       1.00      1.00      1.00        33
           4       0.98      1.00      0.99        41
           5       0.95      0.93      0.94        41
           6       0.97      1.00      0.99        36
           7       0.97      1.00      0.99        34
           8       0.97      0.97      0.97        32
           9       0.94      0.88      0.91        33

    accuracy                           0.97       360
   macro avg       0.97      0.98      0.97       360
weighted avg       0.97      0.97      0.97       360
```

```
plt.figure(figsize=(10, 5))
for i, index in enumerate(np.random.choice(len(X_test), 10, replace=False)):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[index].reshape(8, 8), cmap='gray')
    plt.title(f"Pred: {y_pred[index]}")
    plt.axis('off')
plt.show()
```



OUTPUT ANALYSIS:

- **Accuracy**: The SVM model achieved an accuracy of 97.5% on the test set, indicating that it correctly classified 97.5% of the digits. This high accuracy demonstrates that the SVM effectively distinguishes between different handwritten digits based on their pixel patterns.
- **Classification Report**: The detailed classification report provides insights into the model's performance for each digit class. With high precision and recall values across most classes, the report confirms that the model reliably identifies each digit, with few misclassifications.
- **Visualization**: By displaying a random sample of test images with their predicted labels, we can visually assess the model's predictions. The high accuracy is generally reflected in these predictions, with most images correctly labeled according to their digit.

CONCLUSION:

Support Vector Machines are effective for character recognition tasks, especially with small to medium-sized datasets. With the MNIST dataset, SVMs can achieve high accuracy due to their ability to classify data in high-dimensional spaces.

Here the model's overall high accuracy of 97.5% demonstrates that SVM is a powerful tool for character recognition, especially on a dataset like MNIST. However, small misclassifications might still occur with more complex or ambiguous character shapes.